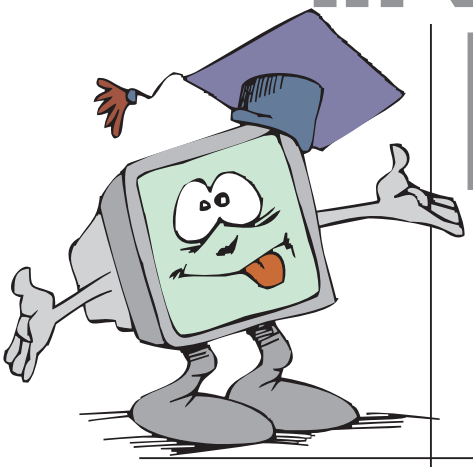
Емил Келеведжиев
Зорница Дженкова

АЛГОРИТМИ ПРОГРАМИ И ЗАДАЧИ



Ръководство за начална подготовка по информатика за олимпиади и състезания

Второ допълнено издание

РЕГАЛИЯ 6
София

Ръководството е предназначено за подготовка на начинаещи ученици до ниво, което им позволява да решават най-трудните задачи, давани у нас на олимпиадите и състезанията по информатика за началните възрастови групи. Отделните глави в книгата започват с основите на програмирането с езика C/C++ и постепенно навлизат в методите за съставяне и реализиране на алгоритми.

Предложени са завършени програми и анализ на решенията на повече от 30 конкурсни задачи от проведените състезания през последните години.

В приложение са дадени условията на най-новите 50 задачи от национални състезания за началните възрастови групи.

За авторите:

Емил Келеведжиев – научен сътрудник в Института по математика и информатика при Българската академия на науките, член на Екипа за извънкласна работа по информатика при Съюза на математиците в България и един от ръководителите на Националния отбор по информатика.

Зорница Дженкова – учител по информатика в Природо-математическата гимназия в Габрово и ръководител на извънкласни занятия за подготовка на олимпиади и състезания.

РЕГАЛИЯ 6
София 1700, п.к. 172
тел. 979-38-42
www.regalia6.com

Второ допълнено издание

Рецензент Стоян Капралов
Редактор Мария Маникова

ISBN 954-745-090-5

ПРЕДГОВОР КЪМ ВТОРОТО ИЗДАНИЕ

България е определена за домакин на Международната олимпиада по информатика през 2009 г. Така страната ни затвърждава лидерската си позиция в света на състезателната информатика, започнала с българското домакинство на Първата олимпиада от 1989 г.

В динамичната област на състезанията по информатика у нас за по-малко от две години – от времето на излизане от печат на първото издание на настоящото ръководство, изискванията към подготовката на състезателите значително се повишиха. От есента на 2004 г. беше направено и ново възрастово разпределение по състезателни групи. Те са вече пет: група E за ученици от 4. – 5. клас, група D за 6. – 7. клас, група C за 8. – 9. клас, група B за 10. – 11. клас и елитната A група – за последния, 12. клас. Новата програма за учебното съдържание е публикувана в кн. 4 от 2004 г. на сп. „Математика и информатика“. При така настъпилите промени, настоящето ръководство може да се използва за подготовка в група E – за най-малките и за начална подготовка при следващата възрастова група D. От 2005 г, освен за група A, вече се провежда заключителен финален кръг на Националната олимпиада и за група D, което ще повиши интереса на учениците от 7. клас за отлично представяне и с цел продължаване на образованието им в профилирани училища.

Настоящето ново издание се наложи поради големия интерес от страна на читателите и заради пълното изчерпване на предишния тираж. Във второто издание са поправени някои печатни грешки и е добавено приложение с условията на 50 задачи от национални състезания за групи E и D, проведени от началото на 2004 до края на 2005 г.

Авторите,
4 декември 2005 г.

ПРЕДГОВОР КЪМ ПЪРВОТО ИЗДАНИЕ

У нас интересът към олимпиадите и състезанията по информатика е традиционно висок. През последните години и особено след 2000-та, българският национален средношколски отбор постигна забележителни успехи в участията си на Международната олимпиада по информатика, като зае между трето и седмо място през отделните години в неофициалното отборно класиране. В индивидуалното класиране Велин Цанов бе на трето място в света през 2002 г. в Корея, а Ивайло Рисков на второ през 2003 г. в САЩ! И всичко това бе завоювано при огромната конкуренция с отбори от над 70 държави в тази бързо развиваща се и изключително популярна нова област на човешка дейност.

Предизвикателство за преподавателите е как да започнат подготовката на най-малките ученици (от 4. или 5. клас), за да могат те да повторят и надминат успехите на сегашните първенци. В страната съществува ежегоден богат календар от състезателни изяви по информатика (три кръга на Националната олимпиада, Зимни празници по информатика, Есенен и Пролетен турнир, няколко регионални състезания, задочни конкурси на списанията Computer и PC Magazine, и др.), насочени като стъпала към върха на Балканската и Международната олимпиада. Възрастовите групи при повечето състезания в страната са четири: група D за ученици от 4. – 6. клас, група C за 7. – 8. клас, група B за 9. – 10. клас и елитната A група, в която се състезават учениците от 11. и 12. клас.

При липсата на каквато и да било задължителна училищна форма за обучение по алгоритмична информатика за учениците от 4. и 5. клас, единственият начин те да бъдат запознати със състезанията и да се подготвят за участие в D-групата, е извънкласната работа и/или самообразованието. Настоящето ръководство има за цел да помогне на преподавателите да се ориентират по каква програма да водят занятията в школите с тези ученици. За предлаганата схема авторите са използвали публикацията на Венета Богданова и Галина Момчева „Структуриране на учебното съдържание в извънкласните форми по информатика“, сп. „Математика и информатика“, кн. 1, 2001 г. Тя от своя страна е резултат от обсъжданията по време на семинара, проведен от Екипа за извънкласна работа по информатика към Съюза на математиците в България в началото на септември 2000 г. в Габрово.

В процеса на подбора на темите за ръководството силно влияние оказаха самите състезателни задачи, давани през последните години у нас.

Предназначение. Основната цел на авторите е била да напишат книга за тези преподаватели, които вече имат определен опит да работят с деца и да ги подготвят за състезания, познават задачи, предлагани на ученическите състезания по информатика, и владеят необходимите програмистки и алгоритмични умения. Освен това настоящето ръководството би могло да се използва и от самите ученици (частично или изцяло) в тяхната самостоятелна работа.

Разпределение. Всяка от главите в книгата би могла да се разглежда като разширен урок, за който – по преценка на преподавателя – да бъдат отделени от едно до три занятия от по два учебни часа. Материалът от първа до девета глава е предназначен да се използва като първа част от подготовката и може да бъде преподаван през първата учебна година при работата с най-малките ученици (5. клас) или през първия учебен срок – при по-големите (6. клас). Разбира се, книгата може да се ползва и за работа с още по-големи ученици, когато те за първи път се запознават със съвременната състезателна информатика.

На отделни места в края на някои от главите даденият материал може да се окаже труден за начинаещите и препоръката ни към преподавателя е да го отложи за повторно изучаване. Това се отнася например за действията с низове.

Език. Използваният език за програмиране е подмножество на C (без строго да се очертава кое е то – нещо подобно има в специалното предаване на опростен английски език на радио „Гласът на Америка“ – не се определя формално в какво се състои опростяването). В публикуваните в книгата програмни фрагменти се ползват и някои елементи от C++, които улесняват начинаещите – например оператори за вход и изход (`cin` и `cout`) и по-свободното деклариране на променливи. Така всъщност езикът за програмиране в изложението е C++. Предполага се, че читателят има начални представи за технологията на програмирането – запознат е със среда за разработка на програми, знае какво е компилатор, изходен текст и изпълним файл, и т.н. и умее да работи на потребителско ниво в MS DOS и MS Windows. Въпреки че използваният език е C/C++, ръководството не налага този език и почти всички публикувани програми и фрагменти от тях могат да се ползват след лесна преработка, от програмиращите на Паскал, например.

Обхват. Според целта, поставена от авторите, материалът в ръководството е насочен към спецификата на задачите, които се предлагат през последните години у нас на състезанията по информатика за по-малките възрастови групи. Затова при избора на някои от алгоритмите в книгата е предпочетена простотата пред бързодействието, необходимо при следващите възрастови групи.

Ще отбележим, че книгата не е общо пособие по информатика и информационни технологии за начинаещи, защото извън обхвата ѝ са останали твърде много теми, а предложеният материал на отделни места е труден за неподготвения читател и това се отнася особенно за някои от конкурсните задачи, давани на състезанията.

Благодарности. Авторите са задължени на всички членове и сътрудници на Екипа за извънкласна работа по информатика към Съюза на математиците в България и на преподавателите в страната, които подготвят малките ученици за състезания по информатика, защото те са подбудителите за написване на настоящото ръководство. Специална благодарност изказваме на Стоян Капралов и на Мария Маникова, защото тяхната помощ значително подобри ръкописа на книгата.

Авторите,
6 януари, 2004

АЛГОРИТМИ И ПРОГРАМИ
(УВОД ЗА УЧИТЕЛЯ)

За определен кръг задачи, главно от областта на математиката, информатиката, инженерните и природните науки, е възможно да се формулират точни правила (инструкции или предписания) за решаването им. *Алгоритъм се нарича всяка точно описана последователност от действия, чрез прилагането на които един изпълнител може да реши конкретна задача от разглеждан клас задачи.*

Понятието алгоритъм се характеризира със следните особености, които го отличават от други възможни начини за решаване на проблеми:

1) *Масовост.* Алгоритъмът се прилага върху всеки конкретен представител на определен клас от задачи.

2) *Дискретност* (*стъпковост*). Всеки алгоритъм е съставен от отделни стъпки, някои от които се изпълняват многократно с определена цикличност.

3) *Детерминираност* (*определеност*). Стъпките в алгоритъма са строго и недвусмислено определени и изпълнителят, за когото са предназначени, е в състояние да ги извърши.

4) *Резултатност.* Решението на задачата се получава след прилагане на краен брой стъпки на алгоритъма.

Понятието алгоритъм е основно за съвременната точна наука. Названието му произлиза от латинизираната форма на името на великия средноазиатски математик от 9. век – Ал Хорезми. Той е описал как се извършват четирите основни аритметични действия с цели числа, записани в обичайната десетична бройна система. Първоначално под алгоритъм са разбирали само правилата за тези действия, а днес смисълът на понятието е значително разширен.

Представяне на алгоритмите. Елементарен начин за представяне на един алгоритъм е чрез *словесно описание* – с думи от нашия естествен език се посочват действията, които трябва да се извършат. Такова описание е разбираемо за широк кръг читатели, понеже не използва специални правила, но в много случаи то е обемисто, непрегледно и връзките между отделните действия се проследяват трудно.

Друг начин за представяне на алгоритъм е чрез *блок-схема.* В този случай се получава визуална представа за възможните разклонения на алгоритмичния процес, което помага за проследяване на логическите връзки между отделните действия. Използват се множество фигури и стрелки, които имат определен естествен смисъл.

Програмиране на алгоритъм. Всеки алгоритъм е предназначен за определен изпълнител. Преди появата на компютрите (приблизително до средата на 20. век), основен изпълнител на алгоритми е бил човекът.

Днес основен изпълнител на алгоритми е компютърът. Сега в много от случаите използваните в практиката алгоритми стават все по-сложни и трудни и се прилагат за обработване на голямо количество данни. Затова все по-немислимо е такива алгоритми да се изпълняват без компютър.

За да бъде изпълнен един алгоритъм от компютър, той трябва да бъде представен и въведен по подходящ начин в компютъра. Представянето и въвеждането на даден алгоритъм в компютъра се нарича *програмиране*. В практиката програмирането е съпроводено с различни други дейности, например съставяне на ръководства за потребителите на програмата.

Език за програмиране. Компютрите могат да изпълняват само разбираеми за тях команди. Един алгоритъм, представен чрез словесно описание или чрез блок-схема, не може да бъде разбран и изпълнен от компютър. Необходимо е описанието на алгоритъма да бъде направено съгласно определени правила. Тези правила се свеждат до строго използване на точно определени думи и символи и като цяло съставят *езика за програмиране*. Алгоритъм, написан на език за програмиране, се нарича *програма*.

Езикът за програмиране е изкуствено създаден език. Той има точно определено и ограничено предназначение, което го отличава от българския или от друг естествен език. Характерна особеност на езика за програмиране е неговата „нетърпимост“ към най-малките грешки, които го отклоняват от правилата му.

Компютърните специалисти са създали доста разнообразни и многобройни езици. Не всичките са универсално предназначени за програмиране на алгоритми – например езикът HTML (ейч ти ем ел) се употребява за изграждане на уеб-страници и почти няма възможности за представяне на алгоритми.

Съществува група от езици за програмиране, които по своята структура са близки до архитектурата на компютъра и затова са подходящи за програмиране на алгоритми за управление на елементи от тази архитектура, например на периферни устройства. Езиците от този вид са известни като *машинни езици* и като *асемблерни езици* и са исторически първите създадени езици за програмиране.

Развитието на програмирането е довело до появата на следваща група езици, известни като *процедурно-ориентирани*. Техен родоначалник е FORTRAN (Фортран), с по-късни представители Basic (Бейсик), Pascal (Паскал), C (Си) и др.

В последното десетилетие значително разпространение получават *обектно-ориентираните езици*. Техни основни представители са C++ (Си плюс плюс) и Java (Джава). Днес те се използват от професионалните програмисти и са подходящи за изграждане на сложни, диалоговоработещи софтуерни системи, включително и в мрежови среди.

Език за програмиране C/C++. Той е типичен представител на две големи групи езици за програмиране: процедурно-ориентираните и обектно-ориентираните. Чрез изучаване на неговите основни елементи може да се получи обща представа за другите езици от двете групи. Той може да се разглежда и като *средство за представяне и описание на алгоритми* по начин, необвързан с програмиране за компютър. Затова езикът C/C++ има характеристиките и на *общ алгоритмичен език* и сполучливо заменя другите средствата за представяне – словесното описание и блок-схемите.

Вход и изход на алгоритъм. Всеки алгоритъм е предназначен за решаване на определен *клас от задачи*, т.е. за такава съвкупност от задачи, в която отделните представители имат общи или сходни елементи във формулировките си. Клас от задачи, разглеждан като едно цяло, се нарича *масова задача*. Тя има *входни данни*. Когато входните данни се заместят с точно определени величини, получаваме *конкретна задача*. Резултатът от работата на алгоритъма върху конкретната задача води до изработване на *изходните данни*, които всъщност представят решението на конкретната задача. Образно казано, алгоритъмът преработва входните данни в изходни.

Обикновено алгоритмите се разглеждат върху масови задачи. Формално погледнато, за решаването само на една конкретна задача (а не на масова задача) няма смисъл от прилагане на алгоритъм. Решението на конкретната задача се състои от една или няколко величини, които не е нужно да се изработват в резултат на последователност от действия – просто те могат да се запишат еднократно (разбира се, след като веднъж са били намерени).

Ефективност (бързодействие). Нека за една дадена масова задача имаме няколко алгоритъма, с помощта на всеки от които се намира правилно решението ѝ. Как може да се прецени кой от тези алгоритми е по-добър, щом всеки от тях решава задачата? Един общоприет критерий е бързината за намиране на решението – прието е ефективността на даден алгоритъм (и на програмата, в която той е реализиран) да се измерва с количеството елементарни стъпки, които е необходимо да се извършат, за да се стигне до решението. От своя страна, това количество от стъпки се измерва чрез зависимостта му от параметър, който по общоприет начин отразява размера на входните данни. Без да навлизаме в подробни и по-строги разсъждения, ще отбележим, че например, когато входните данни са редица от числа, за техен размер се взема броят N на числата в редицата. Тогава, ако разполагаме с два алгоритъма A1 и A2 за решаване на задача с такива входни данни, и ако алгоритъмът A1 я решава с брой стъпки, равен на N, а алгоритъмът A2 – с брой стъпки, изразяващ се с N^2, ясно е, че първият алгоритъм е за предпочитане. Това се вижда от различния брой стъпки, необходим за посочените в таблицата стойности на N и от различното време за работа (пресметнато при условното приемане, че една стъпка се изпълнява за 0.1 секунда):

N	брой стъпки на A1 (с порядък N)	време за работа на A1	брой стъпки на A2 (с порядък N^2)	време за работа на A2
10	10	1 секунда	100	10 секунди
100	100	10 секунди	10000	$\approx$ 16 минути
1000	1 000	$\approx$ 1 минута	1 000 000	$\approx$ 1 ден
1 000 000	1 000 000	$\approx$ 1 ден	10^{12}	$\approx$ 3 000 години

От таблицата става ясно колко голяма е разликата между времето за работа на линейния по бързодействие алгоритъм (този с брой стъпки N) и квадратичния (с брой стъпки N^2) при големи стойности на N. На практика, ако поискаме „приемливо време за работа“ – десетина минути, виждаме, че A1 би могъл да реши задача при най-голяма стойност на $N \approx 10\,000$, докато за A2 тази най-голяма стойност е $N \approx 100$.

Обобщение на линейната и квадратичната сложност на алгоритъм е *полиномиалната* сложност, при която порядъкът на елементарните стъпки е N^p, където p е фиксирано цяло положително число. В практиката се срещат и други порядъци за изразяване на сложността, например логаритмичен и експоненциален.

Логаритмичен порядък има тогава, когато броят на елементарните операции е число, пропорционално на броя на цифрите на N в някоя избрана от нас бройна система (броят на цифрите на едно и също число N в различни бройни системи се отличава с константен множител, независещ от N). Често срещан случай е оценяване чрез броя на цифрите от представянето на N в двоична бройна система (в този случай за порядъка на стъпките на алгоритъма се използва означението $\approx \log_2 N$). Алгоритмите с логаритмичен порядък на стъпките са едни от най-бързите. За примера, даден в таблицата по-горе, ако вместо алгоритъм A1 използваме друг, който обаче има сложност от порядъка на $\log_2 N$, лесно ще пресметнем, че времето за работа при $N = 1\,000\,000$ ще бъде около 2 секунди, което е „невероятно“ по-бързо от „един цял ден“!

Експоненциален порядък на сложността на даден алгоритъм има тогава, когато времето за работа е представено с израз, чието поведение е сходно с това на показателната функция 2^N. Ясно е, че тази величина расте много бързо с увеличаването на N и алгоритми с тази характеристика при практически пресмятания биха могли да се ползват само за твърде малки стойности на N. За примера от таблицата по-горе, ако вместо алгоритъм A1 използваме друг, който е с експоненциален порядък, при $N = 10$ се получават $2^N \approx 1000$ стъпки, които изискват около 100 секунди работа, но при $N = 100$, времето за работа става $2^N \approx 10^{30}$ секунди, което надминава възрастта на Вселената, съгласно съвременните космологични теории.

Среда за разработване на програми. За да създадем и проверим една програма, в повечето случаи използваме специален софтуер, чрез който се осигурява т. нар. среда за разработване. При състезанията по информатика на участниците се предоставя персонален компютър с предварително инсталирана операционна система (версия на MS Windows и/или Linux) и с няколко среди за програмиране. Традиционно (и в съгласие с Международната олимпиада по информатика) това са Borland Pascal 7.0, Borland C++

3.1, Free Pascal и средата DJGPP със системата RHIDE за разработка на програми на C++. Състезателят може да избира едно или даже няколко от тези софтуерни обкръжения и да работи с тях.

Настоящето ръководство не е специфично ориентирано към конкретна среда за разработване на програми. За препоръчване е обаче да се използва някоя от средите, посочени по-горе. Докато първите две имат все още статут на лицензионен софтуер, то другите две се разпространяват изцяло свободно и могат да бъдат изтеглени безплатно и легитимно от Интернет, съответно от `http://www.freepascal.org/` и от `http://www.delorie.com/djgpp/`.

Като допълнителна възможност, особено за начинаещите, може да препоръчаме още две среди за програмиране на C/C++, които са подходящи за решаване на задачите при по-малките възрастови групи. Изтеглянето на софтуера за тях е свободно: Turbo C++ 1.01, продукт на Borland, който може да бъде взет от `http://community.borland.com/museum/` и Dev-C++, предлаган в сайта `http://www.bloodshed.net/devcpp.html`. Dev-C++ е удобен за работа в MS Windows.

Обучение за разработване на програми. Ученикът трябва да усвои особеностите при въвеждането на текста на програма чрез клавиатурата. За да изпълни програмата, трябва да умее да активира съответните команди от обкръжението. Това е възможно по различни начини, в зависимост от конкретната среда – например чрез кликване с мишката върху подходящ бутон, чрез избор на определена точка от съответно меню или с използване на функционален клавиш. При средите на Borland най-често използваният клавиш за тази цел е F9.

Изпълняването на програмата започва с подготвителна работа, извършвана от компютъра. През част от този период, наречен фаза на *компилация*, става ясно дали програмистът е допуснал *синтактични грешки*, т.е. дали е нарушил строгите правила за писане на езика за програмиране. През това време *компилаторът* изпълнява задачата си да преобразува програмния текст в програмен код, който да е подходящ за изпълнение от компютъра.

Ако фазата на компилация не показва синтактични грешки, следва зареждане на създадения изпълним програмен код в *оперативната памет* на компютъра. При това се извършва и резервиране на достатъчно място в същата оперативна памет, необходимо за променливите от програмата – за да могат те да съхраняват стойностите си. Освен това, средата извърша и *свързване* на кода на програмата с код от други програми, например взети от стандартните библиотеки на системата, в която се работи.

След този етап *процесорът* на компютъра започва същинското изпълнение на програмата. По време на това изпълнение, когато една променлива получава нова стойност, стойността се помества точно на това място в паметта на компютъра, което е било предназначено за нея, и ако там е имало предишна

стойност, тя се изгубва. Когато се появи необходимост да се използва стойността на някоя променлива, например, за да се пресметне определен израз, тази стойност се извлича чрез обръщение към съответното място в паметта.

Ученикът трябва да е подготвен за случая, когато фазата на компилация намери *синтактични грешки*. Това е доста често срещано явление, особено при първото пускане на новоразработена програма. Затова и всяка среда за програмиране има специално предвидени средства за *проверки и поправки на програми*. Съответната дейност се нарича *редактиране* на програмата. В най-простите случаи системата посочва позицията в текста, където вероятно е синтактичната грешка, и редактирането от страна на програмиста се свежда до добавяне, премахване или заменяне на един или няколко знака от текста на програмата, с цел да се удовлетворят правилата на езика. След това програмистът стартира компилатора отново.

Ученикът трябва да знае, че липсата на синтактична грешка съвсем не означава, че програмата ще работи правилно. Например, ако в програмата за пресмятане лице на правоъгълник действието умножение `a*b` е сменено погрешно със събиране `a+b`, компилаторът няма да посочи синтактична грешка, но и резултатът от работата на програмата няма да е правилен.

Откриването и поправянето на *смислова грешка* понякога е много по-трудно от справянето със синтактичните грешки. Затова и цялостната проверка, особено на една по-сложна програма, е трудоемък процес. Ученикът трябва да знае, че възможен метод за извършване на такива проверки е пробно многократно изпълняване на програмата, като за входни данни се използват различни, добре подбрани *тестови примери*.

Не трябва да се забравя, че почти всяка програма трябва да бъде запазена за по-нататъшно използване. Ученикът трябва да умее да прави това чрез записването ѝ върху дискета, друга дискова памет (твърд диск) или на друг постоянен носител, например флаш-драйв. В средата за разработване текстът на програмата се съхранява обикновено само в оперативната памет и може да се загуби, например при прекъсване на електрозахранването.

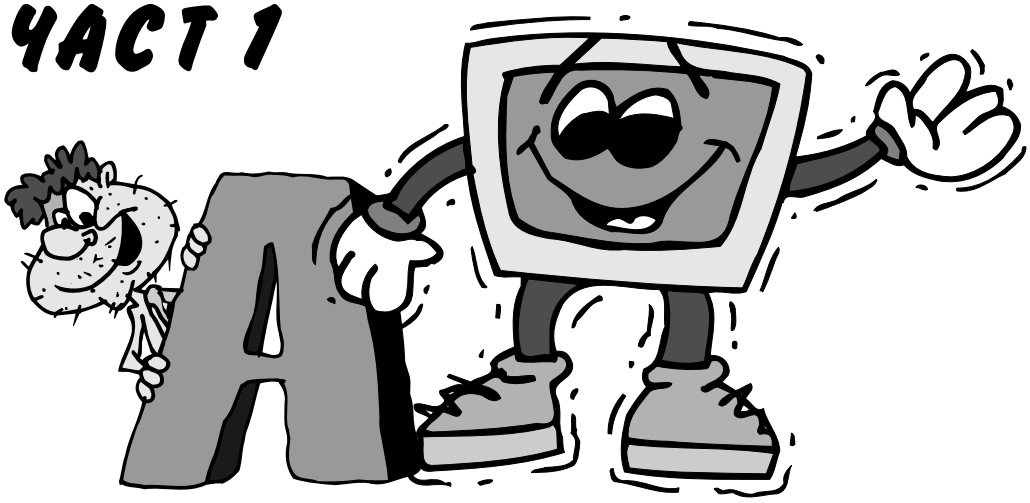
Използване на ръководства за избраната среда. За да усвои една среда за работа, обучаемият, освен да прави практически упражнения с нея, задължително трябва да се научи да ползва съответното ръководство. Всички среди притежават вградена система за даване на помощна информация (help), а много други сведения за средата могат да бъдат намерени в подходящи сайтове в Интернет. Начинът за ползването им трябва да бъде добре усвоен. Обикновено текстовете в тези системи са написани на английски език и за разбирането им са необходими поне начални познания по езика.

Изходни текстове на публикуваните в книгата програми за решаване на конкурсните задачи, както и други бележки, могат да бъдат намерени на следния адрес в Интернет:

`http://www.math.bas.bg/~keleved/books/D/`

Забележка: Програмите, приложени в отделните глави на книгата, както и към всяка от конкурсните задачи, са ориентирани за използване в средата на Turbo C++ 1.01 или Borland C++ 3.1. При работа с други компилатори е възможно да се наложат някои малки промени в изходния текст, например да се осигури тип `int` за функцията `main` и съответно наличие на оператор `return 0;` а понякога и да се премахне окончанието `.h` от определени имена на заглавни файлове, например да се използва `iostream` вместо `iostream.h` и пр.

ЧАСТ 1



Глава 1. „ЗДРАВЕЙ, СВЯТ“

Всяка програма, написана на C/C++, е последователност от знаци, които се разполагат в редове. Обикновено знаците са букви от латинската азбука, цифри и специални (препинателни) знаци. На определени места е възможно да се използват и букви от българската азбука. Програмата трябва да бъде написана и оформена при спазването на строги синтактични правила. Съгласно тези правила, които ще описваме постепенно, една възможно най-проста програма може да има вида:

```
#include<iostream.h>
void main()
{
cout << "Hello, world";
}
```

След като напишем, компилираме и изпълним тази програма, ще установим, че единственото, което тя върши, е да изведе на екрана на компютъра последователността от знаци „Hello, world“ (извежда ги без кавичките). Този израз, преведен на български, гласи „Здравей, свят“.

Важно е, когато работим, да не забравим да запазим текста на програмата върху твърдия диск на компютъра. За целта файлът с този текст – наричан изходен текст на програмата (source – от английски „източник“, произнася се „соурс“) трябва да получи име. Това име се определя от програмиста и обикновено е свързано смислово с предназначението на програмата. Например нашата първа програма може да бъде наречена `hello.cpp`. Да отбележим,

че окончанието `.cpp` показва, че програмата е написана на езика C++ и за правилна работа с нея това окончание трябва да се използва задължително.

Операторът `cout` (произнася се „си аут“), съдържащ знаците `<<` и текста, заграден в кавичките, е пример за най-прост начин за *извеждане на данни*. Без такива оператори не можем да видим какво прави програмата и не можем да проверяваме работата ѝ. Важна особеност за всички оператори са задължителните точка и запетая `(;)`, с които те завършват.

Написаната по-горе програма съдържа единствена *функция*, означена като `main()` (произнася се „мейн“). По-нататък ще видим, че програмите, написани на езика C/C++, обикновено съдържат и други функции, но присъствието на функцията с името `main()` е задължително. От първия оператор, написан в тялото ѝ (тялото е частта, заградена с фигурните скоби `{...}` след `main()`), започва изпълнението на цялата програма.

В предложената по-горе най-проста програма има само един *оператор*. Операторите са основните елементи, с които се изграждат телата на функциите. Всеки оператор задава действие, което компютърът извършва при изпълнението на програмата.

Създаване на програми. За да създадем и проверим една програма, както вече беше отбелязано в увода, използваме специално компютърно обкръжение, наречено *среда за разработване на програми*. Популярни са средите Borland C++, DevC++, Rhide Gnu C++, Microsoft Visual C++ и др. В настоящето ръководство няма да разглеждаме подробно особеностите на тези среди и ще предполагаме, че читателят умее да използва достатъчно добре поне една от тях. Съвсем кратко описание на работата с Borland C++ 3.1 е дадено в Приложение 1.

Ще отбележим, че някои елементи в текста на програмата са зависими от средата за разработване и от конкретната версия на езика, която тя поддържа. Например включването на описанието `#include<iostream.h>` (произнася се „инклуд ай оу стрийм“), което е необходимо за оператора `cout`, в някои случаи може да има вида `#include<iostream>` или даже да бъде пропуснато от програмиста, защото би могло да се подразбира от системата.

Друга особеност е, че при някои от компилаторите се изисква името `main` на главната функция да стои след определението `int`, вместо `void`, както е в нашия пример. Но когато функцията `main` е определена с `int`, необходимо е в повечето случаи последният оператор в тялото ѝ да бъде `return 0;`

Коментари. Част от текста на една програма са *коментарите*. Това са пасажи, които не се обработват от компилатора, а служат за записване на пояснителни бележки от автора на програмата. Предназначени са за хората, които ще проверяват или видоизменят програмата. В езика C++ има два начина за оформяне на коментари. Единият е чрез заграждане на текста със специалните скоби, оформени от по два знака: `/*` – за начало и `*/` – за

край. Другият начин е чрез написване в началото на ред от програмата (или някъде другаде в реда) на две наклонени черти – `//`. Тогава всичко, което е написано от тези черти до края на реда се пропуска от компилатора. Следва пример, в който към дадената в началото програма са добавени коментари и някои нови елементи:

```
// Програма hello.cpp
// "най-простата програма"
#include<iostream.h>
void main()
{
/* следва оператор за извеждане на текст, */
/* който завършва със специалния символ \n, */
/* предназначен за преминаване на нов ред */
cout << "Hello, world\n";

/* следва оператор, чрез който изпълняваната */
/* програма чака натискане на клавиша "Enter" */
cin.get();
}
```

От коментарите в тази програма може да научим например за какво се използва специалният символ `"\n"`. Двата знака `\n` не се отпечатват, а управляват преминаването на нов ред.

Наличието на оператора `cin.get()` често е необходимо, когато разработваме програмата, защото без него изпълнението ѝ би завършило „мигновено“ и ние ще видим на екрана отново прозореца на средата за работа, без да успеем да прочетем това, което програмата извежда. Употребата на този оператор обикновено не е необходима при изпълняване на програмата в прозорец на DOS, а при съставяне на програми, които решават задачи за различни състезания и олимпиади, съгласно общоприетия регламент, същият оператор трябва да се избягва задължително. Възможно е в процеса на разработването да напишем този оператор на едно или повече места в програмата, но в окончателната ѝ версия за състезанието трябва да го премахнем. Това може да стане например чрез „коментирането“ му, т.е. чрез употребата на знаците за коментар.

Понякога операторът за извеждане на текст, освен за отпечатване на всякакви съобщения на английски, на български или на друг език, може да се използва и за получаване на прости фигури върху екрана, съставени от букви или други знаци. Следващата програма отпечатва правоъгълник от звездички:

```
#include<iostream.h>
void main()
{
   cout << "**********\n";
   cout << "*        *\n";
   cout << "*        *\n";
   cout << "*        *\n";
   cout << "**********\n";
}
```

Променливи. Освен да извежда данни, програмата в почти всички случаи също трябва и да чете данни. Един възможен начин за това се постига чрез оператора `cin`, разгледан по-долу. Прочетените данни се съхраняват в *променливи*. Променливите могат да бъдат както единични букви от латинската азбука (например `A`, `B`, `C` или `a`, `b`, `c`), така и последователности от няколко букви и цифри, но задължително започващи с буква (например `x1`, `suma`, `katet`). Променливите, означени с малки и с големи букви, са различни – не трябва да объркваме `A` и `a`. `Suma` не е същата променлива като `suma`

Различните променливи приемат различни по *тип* стойности. Често се използват променливи, чиито стойности са *числа*.

Обявяване на променлива. Преди да бъде използвана, една променлива трябва да бъде *обявена* (*декларирана*). За променливите, които ще приемат само *целочислени стойности*, това може да стане с оператора `int`, например `int x;`. За променливите, които ще приемат не само целочислени стойности, а и дробни – е възможно обявяване с оператора `double`, например `double y;`. Целите стойности, както и дробните, могат да имат и знак минус, например -5. Дробните стойности са десетични дроби, като цялата им част се отделя от дробната с *десетична точка*, например 3.14. С един оператор могат да се обявят няколко променливи от един тип, например:

```
int x, y, z;
```

Една променлива се обявява само веднъж. Повторното ѝ обявяване води до грешка, посочвана от компилатора.

Константи. Освен променливи, в програмите се срещат и друг вид данни, наричани *константи*. Примери за константи са всички означения на числа в програмата, например `5`, `1000`, `-3` или `3.14`. Очевидно стойността, която те носят, не може да се променя по време на работата на програмата. Тя е *самоопределена* от смисъла на написаното.

Присвояване на стойност. За разлика от константите, стойността на една променлива може да се променя по време на работата на програмата; затова се употребява и названието „променлива“. Задаването на стойност на разглеждана променлива е едно от най-простите действия, които се извършват

при програмирането. В езика C/C++ това става, като се напише името на променливата, последвано от знака `=` и числото, което ще стане нейна стойност. Например `x1 = 1;`. Това действие се нарича присвояване на стойност, а операторът, който го извършва – *оператор за присвояване.*

Освен присвояване на константа, една променлива може да присвои стойността на друга променлива. Например след изпълняване на оператора `x=y;`, променливата `x` ще получи стойността на променливата `y`. По-нататък ще видим, че при присвояване дясната част на оператора (след знака за равенство) може да бъде и израз, съдържащ променливи и константи, например `x=y+z+1;`.

Когато на една променлива от целочислен тип `x` се присвоява дробна стойност, целочислената променлива приема само *цялата* част от дробната стойност. Например след присвояването `x = 3.9;` променливата `x` ще получи целочислена стойност `3`.

На много места в програмите се налага да се променя стойността на някоя променлива, но така, че новата ѝ стойност да зависи от старата. Например стойността на променливата `x` може да се увеличи с единица или да се удвои. Операторите за това се записват съответно като `x = x + 1;` и `x = 2*x;`. Възможно записване на същите оператори по друг начин е съответно `x++;` и `x *= 2;`. По-нататък ще разгледаме подробно такива аритметични оператори.

В следващия пример е показана програма, чрез която можем да въведем от клавиатурата стойност на променливата `a`. Освен това в текста на програмата се присвоява една определена стойност 3.14 на друга променлива `b` и накрая програмата извежда тези две стойности:

```
#include<iostream.h>
void main()
{
  int a;
  double b;
  cin >> a; cin.get();
  b=3.14;
  cout << "a=" << a << " b=" << b;
  cin.get();
}
```

Наличието на оператора `cin.get();` на две места в програмата може да се разглежда като спомагателно средство при разработката ѝ, както беше отбелязано по-горе, и често в окончателния вариант този оператор трябва да се премахне. В програмните фрагменти в книгата такъв оператор като правило няма да фигурира, но разработващите тези програми трябва да го употребяват при необходимост.

Стандартен вход и изход. Основното предназначение на операторите `cin` и `cout` е да осигурят начин за въвеждане на данни от клавиатурата и съответно за извеждане (отпечатване) на данни върху екрана. Тези оператори реализират *стандартния вход и изход* на програмата. Забележете, че при оператора за въвеждане посоката на знаците `>>` показва, че данните „влизат“ от *стандартния вход* `cin` в променливата: `cin >> a;`. При извеждане посоката е ориентирана обратно – от променливата към *стандартния изход*: `cout << a;`.

Когато трябва да въведем или изведем повече от една променлива, знаците `>>` или `<<` се поставят и между отделните променливи, например

```
cin >> a >> b;
```

първо ще изчака да въведем от клавиатурата стойността на променливата `a`, след това – на `b`.

При оператора `cout`, както видяхме, е възможно да извеждаме и всякакъв текст, като го отделяме със знаците `<<` от имената на променливите, например:

```
cout << "следва стойността на променливата a: "
     << a
     << ",а това е стойността на b: "
     << b
     << "\n";
```

Разбира се, този фрагмент може да се напише и като непрекъсната последователност:

```
cout << "следва стойността на променливата a: " << a <<
",а това е стойността на b: " << b << "\n";
```

Ще се отпечата същото, но е по-малко прегледно за програмиста, който пише програмата.

Навсякъде между знаците на текста може да поставяме специалния символ `"\n"`, когато искаме на съответното място отпечатването да премине на нов ред, например:

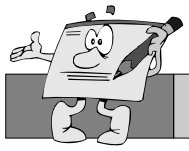
```
cout << "краят на реда е тук\nот тук започва нов ред";
```

Стил на писане. Видяхме, че текстът на програмата може да бъде написан с различно поставяне на празните интервали, с различни отстъпи и подравняване от ляво и по различни начини биха могли да се оставят празни редове. Препоръчва се тази свобода на езика C/C++ да се използва разумно, за да се постигне по-голяма прегледност. Например да се поставят в началото на някои от редовете определен брой празни позиции, за да могат съответни имена или знаци да се разположат подравнени вертикално.

Мета-език за описания. За улеснение, при описанията на правилата на изучаван език за програмиране е прието да се използва общо схематично представяне (мета-език) за конструкциите му. Например общият вид на една проста програма на C++ може да изглежда така:

```
// коментар
#include<име_на_файл,_който_се_включва>
#include<име_на_друг_файл,_който_се_включва>
// коментар
void main()
{
 Оператор_1;
 Оператор_2;
 .....
 Оператор_N;
}
```

При това описание всички изрази, дадени с наклонен шрифт (*италик*), не са буквална част от езика и например програма във вида, даден по-горе, няма да се компилира успешно. За правилното ѝ обработване от компилатора е необходимо изразите в италик да бъдат заменени с подходящи конструкции от езика за програмиране. Например от изучените досега "*име_на_файл,_който_се_включва*", може да се замени с `iostream.h`, а "*Оператор_1;*" с `cout << "Hello\n";`. В горното описание се подразбира, че някои от конструкциите, означени с италик, могат да бъдат пропускани в окончателния текст на програмата.



УПРАЖНЕНИЯ

1. Допустими ли са букви от кирилицата в програми на C/C++? При изписване на операторите правим ли разлика между главни и малки букви?
2. За какво се използват коментарите? Какви са правилата за записване на коментари в езика C/C++? Какви знаци могат да се използват в коментарите?
3. Посочете примери за имена на променливи.
4. Как се обявяват променливи величини с целочислени стойности? А с дробни? Необходимо ли е да бъдат обявени всички променливи, които ще се използват в програмата?

5. Променливата `a` е от целочислен тип. Проверете възможно ли е в програма да извършим присвояването `a=3.14;`. След него какъв ще е типът на променливата `a`? Каква ще е стойността ѝ?

6. С какъв знак завършва всеки оператор? Възможно ли е на един ред да се запишат два или повече оператора?

7. Може ли да се промени стойността на константа в програмата?

8. За какво служи операторът `cin.get();`?

9. Какво е мета-език за описания? Може ли програма, която е представена схематично (с мета-език), да бъде успешно компилирана?

10. Кои от следните редици от знаци са коректни оператори за присвояване, кои не са и защо?

а) `-x = y;` б) `x = -y;` в) `m + n = p;` г) `z = m + n;`

11. Защо след знаците `>>` при оператора за въвеждане `cin` не може да пишем заграден в кавички текст?

12. Обяснете различния начин, по който се извежда буквата `a`, когато е заградена от кавички и когато не е:

```
cout << "a" << a;
```

13. Какво ще отпечата `cout << "\n\n\n";`?

14. Какво ще отпечата

```
cout
<<"o   o\no   o\no   o\nooooo\no   o\no   o\no   o\n";?
```

Упътване: Пребройте внимателно празните интервали.

15. Напишете програма, която обявява целочислена променлива `num`, след което присвоява на тази променлива стойност 100 и накрая показва стойността на екрана по следния начин:

`100 е стойността на променливата num`.

16. Напишете програма, която въвежда от клавиатурата стойност на променливата `N`. След това присвоява на тази променлива нейната собствена стойност, увеличена толкова пъти, колкото е стойността на `N`. Програмата трябва да изведе резултата на екрана. Поставете коментари в програмата.

17. Да се напише програма, която отпечатва на екрана равнобедрен триъгълник от звездички.

18. Какъв ще бъде резултатът от изпълнението на следващата програма?

```
#include <iostream.h>
int main()
 { int a, b;
   cin >> a >> b >> a >> b >> a;
   cout << a << " " << b << " "
        << a << " " << b << " "
        << a << "\n";
   return 0;
 }
```

19. С програма изведете на екрана фигурата „сърце“:

```
 oo     oo
o   o   o
 o      o
   o  o
     o
```

Глава 2. ПРЕСМЯТАНЕ ПО ФОРМУЛА

Чрез програма можем да въведем едно или няколко числа, да извършим зададени аритметични действия с тях и да изведем резултата на екрана, за да го видим. Например следната програма въвежда две числа, пресмята тяхната сума и я извежда на екрана:

```
#include<iostream.h>
void main()
{
 int a,b,c;
 cout << "a=";
 cin >> a;
 cout << "b=";
 cin >> b;
 c=a+b;
 cout << "a+b=" << c << "\n";
}
```

Променливите, които се използват в горната програма, са означени с буквите `a`, `b` и `c`. Те са декларирани чрез оператора `int a,b,c;`. С това се означава, че те ще приемат за свои стойности цели числа. Едновременно с обявяването (декларирането) на променливите, е възможно и присвояването на начални стойности за някои от тях. Например:

```
int x, y=0, z, t=1;
```

Могат да бъдат изведени две или повече числа. Например, ако `a` и `b` имат смисъл на дължини на страните на правоъгълник, следващата програма извежда обиколката и лицето на този правоъгълник. Понеже тук променливите `a` и `b` приемат не само цели стойности, а и дробни, те са декларирани чрез оператора `double a,b;`:

```
#include<iostream.h>
void main()
{
 double a,b;
 cout << "a=";
 cin >> a;
 cout << "b=";
 cin >> b;
 cout << 2*(a+b) << " " << a*b << "\n";
}
```

При някои задачи резултатът се извежда след последователно пресмятане по няколко формули.

Пример 1. Пешеходец вървял по различни местности. Неговата скорост на движение в равнината била v_1 км/ч, в гората – v_2 км/ч и в планината – v_3 км/ч. Времето за движение било съответно t_1, t_2 и t_3 ч. Какъв път изминал пешеходецът?

```
#include<iostream.h>
void main()
{
 double v1, v2, v3, t1, t2, t3, s1, s2, s3, s;
 cout << "Въведете v1, v2 и v3 :";
 cin >> v1 >> v2 >> v3;
 cout << "Въведете t1, t2 и t3 :";
 cin >> t1 >> t2 >> t3;
 s1 = v1*t1;
 s2 = v2*t2;
 s3 = v3*t3;
 s = s1 + s2 + s3;
 cout << "Изминатият път е " << s << " км.\n";
}
```

Във формулите се използват знаците `+`, `-`, `*` и `/`, съответно за събиране, изваждане, умножение и деление, както и скоби `( )`, съгласно познатите математически правила.

Употреба на скоби. Възможно е, вместо да записваме няколко присвоявания с междинни спомагателни променливи, да напишем едно присвояване с употреба на скоби. Например последователността от трите оператора

```
x=a+b;
y=c-d;
z=x*y;
```

може да се замени с един оператор

```
z=(a+b)*(c-d);
```

Целочислено деление. Важна особеност е, че знакът за деление, приложен между две целочислени променливи, води до пресмятане на *цялата част от резултата при делението*. Например фрагментът

```
int a=7, b=3;
cout << a/b;
```

ще отпечата 2, докато при фрагмента

```
double a=7.0, b=3.0;
cout << a/b;
```

резултатът, който се извежда на екрана, е `2.333333`.

Размяна на стойности. Елемент на много алгоритми е *размяната на стойностите* на две променливи. За пример да приемем, че първоначално променливите `a` и `b` имат стойности 3 и 4, т.е. в програмата е било извършено присвояването

```
int a, b;
a=3; b=4;
```

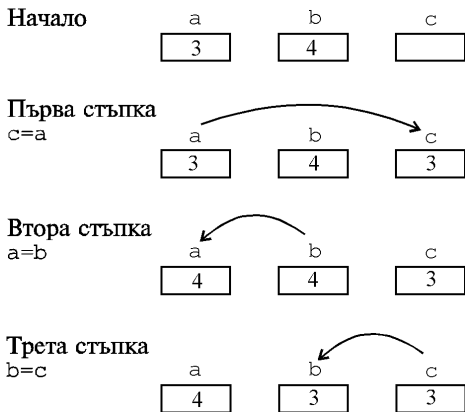
Как да разменим тези стойности? Ако напишем

```
a=b;
b=a;
```

това няма да доведе до желаната размяна, защото след изпълнението на първия оператор ще се окаже, че и `a`, и `b` имат една и съща стойност, равна на 4. Как да направим така, че променливата `a` да получи стойността, която е била в `b`, а променливата `b` да получи стойността на `a`? Проблемът се решава чрез въвеждане на спомагателна променлива `c`:

```
int c;
c=a; a=b; b=c;
```

Схематичното представяне на процеса е следното:



Като страничен ефект стойността на променливата `c` накрая става равна на последната стойност на `b` – тази стойност всъщност е първоначалната стойност на `a`.

Ще отбележим, че преди първото присвояване променливата `c` *няма стойност*. По-точно казано, стойността ѝ е неопределена – равна е на това, което

е съдържала тази част от паметта на компютъра, която е заета от променливата. Естествено изискване е, когато пишем програми, да осигурим подходящата стойност на всяка променлива, за да използваме след това тази стойност.

Съкратена форма на оператори за присвояване. Честата употреба на някои конструкции е накарала създателите на езика C/C++ да въведат по-кратки техни форми. Добавянето на стойността на променливата `b` към стойността на променливата `a`, както знаем, се задава с оператора `a = a + b;`. Същото действие, съгласно правилата на езика C/C++, може да се запише като `a += b;`. Сходни означения са въведени за случаите, когато вместо действието събиране е използвано изваждане, умножение и деление. Така операторите

`a = a - b;`, `a = a * b;`, `a = a / b;`,

могат да бъдат заменени съответно от

`a -= b;`, `a *= b;`, `a /= b;`.

Друго често употребявано действие – добавянето на единица към стойността на променливата `a`, както знаем, удобно се записва като `a++`. Аналогичното означение за изваждане на единица от стойността на същата променлива е `a--;`.

Операция за остатък при деление. В езика C/C++ е предвидена специална операция за пресмятане на *остатъка при делението* на две цели числа. За знак на тази операция е избран знакът процент %. Този смисъл на означението % трябва да имаме предвид, когато го срещнем в аритметична операция и в текста на програмата не трябва да го приемаме за процент.

Пример 2. Ако `a` и `b` са променливи от целочислен тип и е извършено присвояването `a=5` и `b=3`, тогава стойността на израза `a%b` е равна на 2, защото 2 е остатъкът при делението на 5 с 3.

Отделяне на цифрите на единиците и десетиците на двуцифрено число. Когато пресметнем остатъка от делението на дадено цяло число с числото 10, получаваме цифрата на единиците на даденото число. Например 37 % 10 е равно на 7. Цифрата на десетиците на даденото число може да намерим като резултат от целочисленото деление с 10. Така пресмятането на резултата от израза 37/10 дава числото 3, което е цифрата на десетиците на числото 37.

Предназначението на следващата програма е да въведе едно положително двуцифрено число и да отпечата цифрата на единиците и на десетиците му:

```
#include<iostream.h>
void main()
{
 int a;
 cout << "Въведете цяло число между 10 и 99: ";
 cin >> a;
 cout << "Цифрата на единиците е " << a%10 << "\n";
 cout << "Цифрата на десетиците е " << a/10 << "\n";
}
```

Пресмятане на числото чрез неговите цифри. Когато е дадено, че променливата `x` съдържа цифрата на десетиците, а променливата `y` съдържа цифрата на единиците на двуцифрено число, тогава самото двуцифрено число се получава от израза `10*x+y`.

Отделяне на цифрите при трицифрени числа. Цифрата на единиците се намира по същия начин, както по-горе. Но в този случай при целочисленото деление на даденото число с 10 ще се получи двуцифрено число. За него вече знаем как да отделим цифрите му – неговата цифра на единиците е всъщност цифрата на десетиците на първоначално даденото число, а цифрата на десетиците на двуцифреното число е равна на цифрата на стотиците от първоначалното число.

Пример 3. Нека е дадено числото 321. Остатъкът при делението му с 10 е 1 – цифрата на единиците. По-нататък, като разделим целочислено 321 на 10, получаваме 32. За това двуцифрено число остатъкът при делението с 10 е 2, а целочисленото деление с 10 дава 3. Виждаме, че 2 и 3 са съответно цифрата на десетиците и цифрата на стотиците от първоначално даденото число 321.

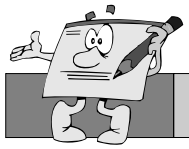
Следващата програма въвежда трицифрено число `a`, отделя цифрите му (съответно в `x`, `y` и `z`), след което възстановява в променливата `c` първоначално въведеното число чрез умножаване на цифрите му съответно с 1, 10 и 100 и събиране. Променливата `b` служи за съхраняване на спомагателното двуцифрено число:

```
#include<iostream.h>
void main()
{
 int a,b,c,x,y,z;
 cin >> a;
 z = a%10;
 b = a/10;
 y = b%10;
 x = b/10;
 c = 100*x + 10*y + z;
}
```

Тази програма няма изход на екрана.

Точки през равни разстояния. По дължината на една алея са забити едно след друго еднакви колчета, като всеки две съседни колчета са на разстояние 1 метър едно от друго. Върху колчетата са написани последователни номера 1, 2, 3 и т. н. Напишете програма, която въвежда два от тези номера `a` и `b`, като номер `a` е по-малък от номер `b`, и извежда *броя* на колчетата между колчетата с номера `a` и `b` и *разстоянието* в метри между същите две колчета.

```
#include<iostream.h>
void main()
{ int a, b;
  cin >> a >> b;
  cout << b-a-1 << "\n";
  cout << b-a << "\n";
}
```



УПРАЖНЕНИЯ

1. Ако в една програма са обявени `int x = 5` и `double y = 2`, проверете колко е `5/2` и `x/y`.

2. Каква ще бъде стойността на променливите `X` и `Y` в резултат от изпълнението на всяка от дадените последователности от оператори?

а)
```
int X = 2;
X = X * X;
X = X * X * X;
```

б)
```
double X = 1.5;
X = 2 * X + 1;
int Y = X/2;
Y = X + Y;
X = X - Y;
```

в)
```
int X = 5.8;
double Y = -7.9;
Y = X;
X = Y;
```

3. Каква ще бъде стойността на променливата `X` при изпълнение на последователността от оператори?

```
X = 2;
X = X + X;
X = X - X;
```

4. Определете стойността на целочислените променливи, ако това е възможно:

а)
```
A = 21 / 5
B = 20 % 5
C = 14 / 6.0
D = 14 % 0
E = 5 % 13
```

б)
```
A = 2 % 3
B = 36.0 % 6
C = 81 / 0
D = 38 / 6
E = 3 / 2
```

5. Напишете програма, която въвежда две числа, пресмята и извежда тяхната сума, произведение и разлика.

6. Напишете програма, която въвежда две числа от клавиатурата, пресмята остатъка от делението на първото с второто и извежда резултата на екрана.

7. Напишете програма, която пресмята лицето на квадрат, ако е даден неговият периметър.

8. Дадени са три числа. Напишете програма, която намира тяхното средноаритметично.

9. Сменете стойностите на променливите `a` и `b` в програма, без да използвате допълнителна променлива.

Упътване: `a=a+b; b=a-b; a=a-b;`

10. Дадено е разстоянието между два града в метри. Напишете програма, която пресмята това разстояние в километри.

11. Даден е правоъгълник с размери 543 мм на 130 мм. Какъв най-голям брой квадрати със страна 130 мм може да се отрежат от него?

12. Напишете програма, която въвежда едно двуцифрено число, пресмята произведението на десетиците и единиците на числото и извежда получената стойност на екрана.

13. Напишете програма, която пресмята сумата от цифрите на едно четирицифрено число.

14. Дадено е трицифрено число.

а) Напишете програма, с която да отпечатате на екрана числото, получено от цифрите на даденото, прочетени отдясно-наляво.

б) Представете си, че изтривате първата цифра, като едновременно с това я дописвате най-накрая. Напишете програма, с която да отпечатате полученото число.

в) Напишете програма, която разменя местата на първата и втората цифра на числото.

15. Дадено е цяло число K, не по-малко от 1 и не по-голямо от 365. Напишете програма, която присвоява на целочислената променлива `A` една от стойностите 1, 2, 3, 4, 5, 6 или 0, в зависимост от това, какъв ден от седмицата (понеделник, вторник, сряда, четвъртък, петък, събота или

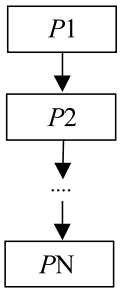
неделя) е K-тият ден на годината, в която 1 януари е понеделник. Има ли значение дали годината е високосна?

16. Една година на Юпитер има продължителност около 12 земни години. Напишете програма, която конвертира (превръща) земни дни в юпитериански години. Задайте броя на земните дни в програмата. Програмата трябва да пресметне и изведе съответния брой юпитериански години.

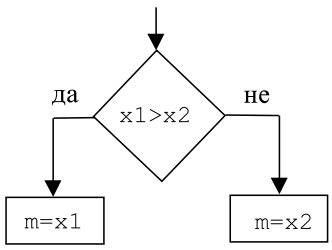
17. Гравитацията на Луната е около 17% от тази на Земята. Напишете програма, която да дава възможност на потребителя да въвежда своето тегло и след това да извежда теглото си на Луната.

Глава 3. РАЗКЛОНЯВАНЕ НА ПРЕСМЯТАНИЯТА

В разгледаните досега програми пресмятанията се извършваха последователно, т.е. програмата се състоеше от няколко оператора $P1$, $P2$, ..., PN (това бяха оператори за обявяване на променливи, оператори за присвояване на стойности или оператори за вход или изход) и тези оператори се изпълняваха (извикваха) един след друг. Схематично работата на програмата в този случай може да се изобрази чрез *линейна блок-схема*:



В програмите често се налага на някои места да се извършва само едно от две или повече предвидени действия, като изборът зависи от резултата на проверката на дадено *условие*. В такива случаи, при работата на програмата не се изпълняват безусловно един след друг всичките написани в нея оператори. Кои от операторите да се изпълнят, се определя в зависимост от съдържанието на някои от данните в програмата. Например, ако искаме на променливата `m` да се присвои по-голямата от стойностите, които се съдържат в променливите `x1` и `x2`, трябва да сравним стойностите на `x1` и `x2` и в зависимост от резултата на сравнението да изпълним или оператора `m=x1`, или оператора `m=x2`. Схематично това може да се представи чрез следната блок-схема:



При илюстрациите чрез блок-схеми условията, които се проверяват, се записват в ромбовидни клетки, а изпълнимите оператори – в правоъгълници. В зависимост от резултата на проверката („да“ или „не“) се следва пътят, означен със съответните стрелки.

Когато програмираме на езика C/C++ горния пример, използваме спе-

циално предвидена конструкция, за да укажем кой от двата оператора за присвояване, `m=x1` или `m=x2`, да бъде изпълнен в зависимост от резултата на проверката `x1>x2`. Това е условният оператор:

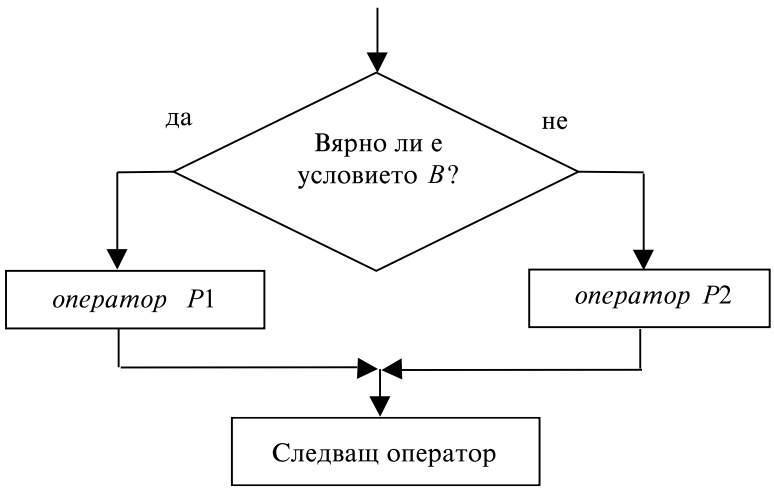
```
if(x1>x2)
 {
  m=x1;
 }
else
 {
  m=x2;
 }
```

По-общо казано, условният оператор се употребява във вида:

```
if(B)
 {P1;}
else
 {P2;}
```

При него `if` и `else` са служебни думи („if“ – „ако“, „else“ – „в противен случай“), B означава логическо *условие*, а $P1$ и $P2$ са оператори. При използването на този оператор в текста на програма, B трябва да бъде заместено с конкретно условие, а $P1$; и $P2$; – с конкретни оператори.

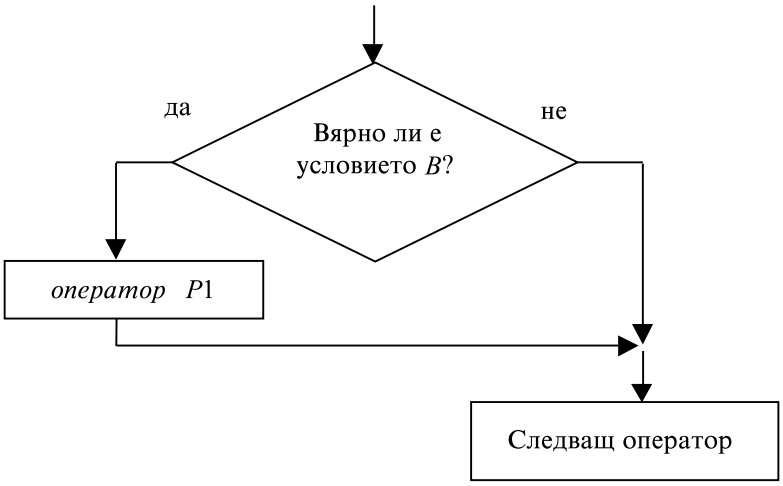
Условният оператор работи по следния начин: ако условието B е в сила, изпълнява се операторът $P1$; а операторът $P2$; се пропуска; в противен случай – когато условието B не е в сила – операторът $P2$; се изпълнява, а операторът $P1$; остава неизпълнен. И в двата случая управлението след това се предава на следващия оператор след условния оператор. Представянето на тези действия чрез блок-схема има вида:



Съществува съкратен вид на условния оператор, в който отсъства частта започваща с `else`, т.е. остава само конструкцията:

`if(`B`){`$P1$`;}`

При тази форма операторът $P1$; се изпълнява само когато условието B е в сила. Когато това условие не е в сила, операторът $P1$; се пропуска и изпълнението на програмата продължава със следващия оператор. Схематично това е представено на блок-схемата:



Логическите условия могат да бъдат разнообразни по вид, но най-простите от тях са *отношенията*. Те са записи на равенства или неравенства. Например

```
a == b
```

е условие, с което се проверява дали променливите `a` и `b` имат равни стойности. Други примери за отношения са

```
x < y+z
b*b-4*a*c > 0
```

В общия случай отношението се състои от два израза, съединени с един от знаците „`<`“, „`>`“ или с един от двойните знаци „`==`“, „`!=`“, „`<=`“, „`>=`“. Последните означават съответно „е равно на“, „не е равно на“, „е по-малко или равно на“ и „е по-голямо или равно на“. За проверка на равенство се използва двойният знак „`==`“, защото единичното равенство е знак, който се използва при присвояване на стойност.

Забележка: При математическите означения се използват знаците „$\leqq$“, „$\geqq$“ и „$\neq$“ вместо съответните означения „`<=`“, „`>=`“ и „`!=`“ в езика C/C++. Освен това, при математически разглеждания за сравнението „равенство“ се използва, както знаем, единичният знак „$=$“, вместо „`==`“ в езика C/C++.

Следващият пример показва как да използваме оператора `if`, за да пресметнем в променливата `m` по-малкото (а не по-голямото) от числата `x1` и `x2`:

```
if (x1 < x2){m = x1;} else {m = x2;}
```

В този пример на мястото, заградено с фигурни скоби, има само по един оператор. Възможно е там да напишем и повече от един оператор, т.е. в общия случай условният оператор може да приеме следния вид:

`if`(B)
`{`$P1$; $P2$; ... PN;`}`
`else`
`{`$Q1$; $Q2$; ... QN;`}`

където B е условие, а $P1$; $P2$; ... PN; и $Q1$; $Q2$; ... QN; са други оператори.

Когато обаче искаме да използваме във фигурните скоби само един оператор, тогава може да не пишем самите фигурни скоби; така програмистите на езика C/C++ често пишат

```
if (x1 < x2) m = x1; else m = x2;
```

Съставен оператор (блок). Съвкупност от няколко оператора, написани един след друг и разположени между отваряща и затваряща фигурна скоба, се нарича *съставен оператор* или блок. На много места, където, съгласно правилата на езика, може да се напише един оператор, се позволява да се напише и съставен оператор. Възможно е, както видяхме, съставният оператор да съдържа и само един оператор. Вече видяхме пример за употреба на съставни оператори в дадената по-горе конструкция на оператора `if...else`.

Когато променливата `x` е дефинирана в един блок, достъпът до нея е възможен само в рамките на блока и разбира се, в тези оператори, които са написани след дефиницията на `x` в блока.

Пример 1. Пресмятане на максимума `max` и минимума `min` на две числа `x1` и `x2` с използване на съставни оператори:

```
if (x1>x2)
{
  max = x1;
  min = x2;
}
else
{
  max = x2;
  min = x1;
}
```

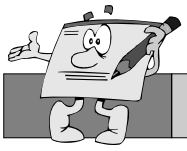
След изпълнението на така написания фрагмент променливата `max` ще съдържа по-голямата от двете стойности, намиращи се в `x1` и `x2`, а променливата `min` ще съдържа по-малката от тези две стойности. Ако стойностите на `x1` и `x2` са равни, променливите `max` и `min` ще съдържат една и съща стойност, равна на общата стойност на `x1` и `x2`.

Пример 2. Следващата програма въвежда две числа `a` и `b` и извежда резултата от делението `a/b`, само ако това е възможно, т.е. когато променливата `b` няма нулева стойност. В противен случай програмата извежда подходящо съобщение:

```
#include<iostream.h>
void main()
{ int a, b;
  cin >> a >> b;
  if(b != 0) cout << a/b <<"\n";
  else cout << "Делене на нула\n";
}
```

Пример 3. Да се напише програма, която събира или изважда две цели числа. Първо се въвежда „код“ на операцията (1 – за събиране и 2 – за изваждане), а след това се въвеждат числата. Накрая резултатът се извежда на екрана.

```
#include<iostream.h>
void main()
{ int a,b,c;
  cout << "Въведете код на операцията: ";
  cin >> c;
  cout << "Въведете първото число: ";
  cin >> a;
  cout << "Въведете второто число: ";
  cin >> b;
  if(c==1)
   cout << "Сумата на числата е " << a+b << "\n";
  else
   cout << "Разликата на числата е " << a-b << "\n";
}
```



УПРАЖНЕНИЯ

1. Какво е предназначението на условния оператор? Какви форми има той?

2. Кой оператор се изпълнява, ако условието в кратката форма на оператора за условен преход не е удовлетворено?

3. За какво най-често се използва съставен оператор?

4. Възможно ли е някой от операторите в съставен оператор също да е съставен оператор?

5. Напишете програма, която въвежда две различни числа и отпечатва текста „По-голямото от въведените числа е“, последван от стойността на по-голямото число.

6. На цялата променлива `a` да се присвои последната цифра от цялата част на положителното дробно число `x`. Например, ако `x = 54.367`, `a = 4`.

7. Има ли разлика в смисъла на следните два програмни фрагмента?

а) `x = 0; if (a > b ) x = a + b;`

б) `if (a<=b) x = 0; else x = a + b;`

8. Посочете грешките в следните фрагменти:

```
if (a>0) s = 0 else s = 1;
```

```
if (s == 0) f = 1/s;
```

9. Напишете програма, която определя дали дадено въведено число е:

а) отрицателно, или не е; б) четно, или не е.

10. Да се напише програма, която проверява дали даден интервал от числа [P, Q] е коректно зададен (`P<=Q`), и ако не е, да размени стойностите на `P` и `Q`.

11. Да се напише програма, която показва две числа и изисква от потребителя да събере числата и да въведе отговора. След това програмата казва дали отговорът е верен.

12. Да се напише програма, която превръща метри в сантиметри и обратно, в зависимост от избора на потребителя (определя се с код на операцията: 1 или 2).

Глава 4. ПРИЛОЖЕНИЯ НА УСЛОВНИЯ ОПЕРАТОР

Намиране на най-малкото от 3 числа. Нека на променливите `a`, `b` и `c` са присвоени числа. Задачата е да се намери най-малкото от тях. Методът, по който ще извършим това, словесно описваме така:

• На спомагателна променлива с име `min` присвояваме стойността на `a`.

• Сравняваме `b` с `min`. Ако стойността на `b` е по-малка от стойността в `min`, присвояваме стойността на `b` в `min`. В противен случай `min` не се променя.

• Сравняваме `c` с `min`. Ако стойността на `c` е по-малка от стойността в `min`, присвояваме стойността на `c` в `min`. В противен случай `min` не се променя.

• Търсеният резултат е стойността на `min`.

Забелязваме, че стойността на `min` играе ролята на *текуща минимална стойност* – например преди последното сравнение `min` съдържа най-малката стойност измежду стойностите на първите две разгледани до текущия момент стойности – тази на `a` и тази на `b`.

Програмната реализация е следната:

```
#include<iostream.h>
void main()
{int a, b, c, min;
 cin >> a >> b >> c;
 min=a;
 if(b<min)min=b;
 if(c<min)min=c;
 cout << min << "\n";
}
```

Подреждане по големина на 3 числа. Разглеждаме три различни стойности, зададени съответно в три променливи `a`, `b` и `c`. Поставяме задачата: чрез няколко размени на тези стойности (ако се налага) да направим така, че големините на числата, намиращи се в променливите `a`, `b` и `c` да бъдат в нарастващ ред, т.е да са изпълнени неравенствата `a < b` и `b < c`. Пренареждането на стойностите по указания начин се нарича *сортиране във възходящ ред*. Начинът, по който решаваме задачата, има следното словесно описание:

• Въвеждаме стойности за `a`, `b` и `c`.

• Сравняваме стойностите на `a` и `b`. Ако `b<a`, извършва се размяна на стойностите на `a` и `b`. В противен случай размяна не се извършва.

• Сравняваме стойностите на `a` и `c`. Ако `c<a`, извършва се размяна на стойностите на `a` и `c`. В противен случай размяна не се извършва.

• Сравняваме стойностите на `b` и `c`. Ако `c<b`, извършва се размяна на стойностите на `b` и `c`. В противен случай размяна не се извършва.

• Извеждаме стойностите на `a`, `b` и `c`.

Програмната реализация е следната:

```
#include<iostream.h>
void main()
{ int a,b,c,t;
  cin >> a >> b >> c;
  if(b<a) {t=b; b=a; a=t;}
  if(c<a) {t=c; c=a; a=t;}
  if(c<b) {t=c; c=b; b=t;}
  cout << a << " " << b << " " << c << "\n";
}
```

За да се убедим в правилността на тази програма, трябва да разгледаме всички възможности, които могат да се случат. Тук ще се ограничим с разглеждането на случая, когато стойността на `a` е най-голямата. Тогава при първата размяна тази стойност ще попадне в променливата `b` и след това, при третия оператор `if`, тази стойност ще бъде присвоена на променливата `c`. С това се осигурява, че най-голямата стойност ще отиде на последното място в редицата `a`, `b`, `c`. Със сходни разсъждения се убеждаваме, че и останалите две стойности ще попаднат на правилните места.

Пример 1. Напишете програма, която въвежда две различни цели числа в променливите `a` и `b`, след което на променливата, която има по-малка стойност, присвоява сумата на двете въведени числа, а на другата променлива присвоява произведението на двете въведени числа.

За да решим задачата, използваме две спомагателни променливи `x` и `y`, на които предварително присвояваме сумата и произведението от стойностите на `a` и `b`:

```
#include<iostream.h>
void main()
{ int a,b,x,y;
  cin >> a >> b;
  x=a+b;
  y=a*b;
  if(a<b) {a=x;b=y;}
  else{a=y;b=x;}
  cout << a << " " << b << "\n";
}
```

Пример 2. На лист хартия били написани 3 цели положителни числа a, b и c, за които е изпълнено, че $a + b = c$. Някой изтрил едно от числата.

Напишете програма, която въвежда числата в реда a, b и c, като на мястото на изтритото число въвежда 0. След това програмата трябва да възстанови изтритото число и да изведе трите числа.

Понеже има три възможности за мястото на изтритото число, трябва последователно да проверим кое от числата е нула и след това с подходящи аритметични операции да го възстановим. За възстановените числа използваме променливите `x`, `y` и `z`.

```
#include<iostream.h>
void main()
{ int a,b,c,x,y,z;
  cin >> a >> b >> c;
  if(a==0)
   {x=c-b; y=b; z=c;}
  if(b==0)
   {x=a; y=c-a; z=c;}
  if(c==0)
   {x=a; y=b; z=a+b;}
  cout << x << " " << y << " " << z << "\n";
}
```

Вложени оператори `if`. При решаване на по-сложни задачи се налага някои от участващите оператори в оператора `if` също до бъдат оператори `if`. Често срещано е написването на оператора `if` след думата `else`. Например следният фрагмент от програма зарежда в променливата `s` стойност $-1, 0$ или 1, в зависимост от това дали `x` има отрицателна, нулева или положителна стойност:

```
if(x<0) s = -1;
else if(x==0) s = 0;
else s = 1;
```

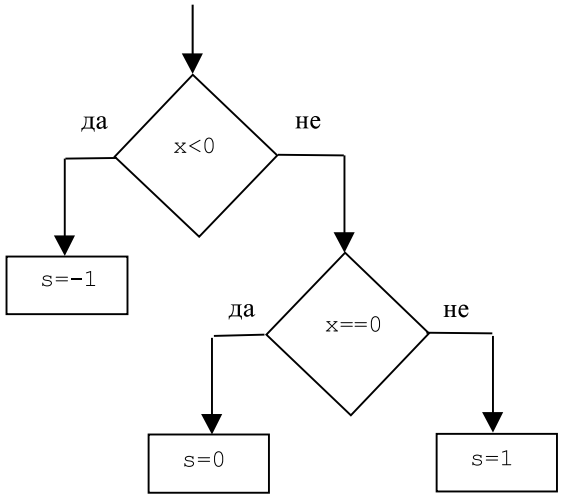
Този фрагмент може да бъде написан по еквивалентен начин така:

```
if(x<0) s = -1;
else {if(x==0) s = 0; else s = 1;}
```

Изразено с други означения, стойността, която той пресмята, е:

$$s = \begin{cases} -1, & \text{ако } x < 0 \\ 0, & \text{ако } x = 0 \\ 1, & \text{ако } x > 0 \end{cases}$$

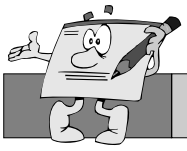
Представяне чрез блок-схема:



Програма за решаване на уравнението $ax = b$**.** Нека да разгледаме уравнението $ax = b$, в което коефициентите a и b са зададени като произволни цели или дробни числа. Често срещана задача е да се намери при каква стойност на x, равенството $ax = b$ е вярно. Тази задача се нарича задача за решаване на уравнението $ax = b$. Следващата програма проверява дали коефициентът a е различен от нула, с което определя колко решения има уравнението и отпечатва решението, когато то съществува и е единствено.

```
#include<iostream.h>
double a,b;
void main()
{ cout << "Въведете коефициентите\n";
  cout << "a: "; cin >> a;
  cout << "b: "; cin >> b;
  if(a==0)
  {
   if(b!=0) cout << "Уравнението няма решение\n";
   else cout << "Уравнението има безброй решения\n";
  }
  else
  {
   cout << "Уравнението има единствено решение x=";
   cout << b/a << "\n";
  }
}
```

Например за въведени `a = 2` и `b = 7` стойността на a не е нула и стойността на израза `b/a` е равна на `3.5`. Това показва, че уравнението $2x = 7$ има решение $x = 3.5$.



УПРАЖНЕНИЯ

1. Какво се постига чрез влагане на условни оператори?

2. Даден е фрагментът

```
if (n == 1) f = 3*x + 2;
else if (n == 2) f = 3*x - 2;
else f = 0;
```

Определете кои от написаните по-долу две съвкупности от оператори са еквивалентни на дадения фрагмент:

а)
```
f = 0;
if (n == 1) f = 3*x + 2;
if (n == 2) f = 3*x - 2;
```

б)
```
f = 0;
if (n == 1) f = 3*x + 2;
else f = 3*x - 2;
```

3. Какво трябва да променим в програмата за сортиране във възходящ ред на три числа, за да ги сортираме в низходящ ред?

4. Ако променливата `a` има стойност 8, определете каква стойност ще има променливата `b` след изпълнението на оператора:

```
if (a > 4) b = 5; else
 if (a < 4) b = -5; else
  if (a == 8) b = 8; else b = 3;
```

5. Да се напише програма, която въвежда от клавиатурата едно цяло число. Ако числото е кратно на 3, програмата трябва да изведе същото число. Ако то е кратно на 7, програмата трябва да пресметне произведението на числото с 4 и да изведе резултата. Ако числото не е кратно нито на 3, нито на 7, програмата трябва да изведе 0.

6. Да се напише програма, която въвежда от клавиатурата числата `x` и `y`. Ако `x` е по-малко от нула, програмата трябва да присвои на `z` стойността на по-малкото от двете числа `x` и `y`; в противен случай да присвои на `z` полусумата на въведените числа.

7. Дадени са три дробни числа. Да се напише програма, която проверява кое е по-голямо: тяхната сума или тяхното произведение.

8. Променливата Y зависи от променливата X по следния начин:

$$Y = \begin{cases} 4X - 7, & \text{ако } X < 0 \\ 0, & \text{ако } X = 0 \\ 5 + 3X, & \text{ако } X > 0 \end{cases}$$

Да се напише програма, която по дадено X намира съответната стойност на Y.

9. Разглеждаме променливите `a`, `b`, `c` и `d`. Да се напише програма, която въвежда техните стойности и извежда на екрана:

а) най-голямата стойност, въведена за някоя от променливите;

б) най-малката стойност, въведена за някоя от променливите;

в) името на променливата с максимална стойност.

10. Да се напише програма, с която се въвежда код за аритметична операция (1 – за събиране, 2 – за изваждане, 3 – за умножение, 4 – за деление). След това се въвеждат две числа и програмата събира, изважда, умножава или дели двете числа, в зависимост от избраната операция.

11. Напишете програма, която пресмята обиколката на правоъгълник, триъгълник или трапец, в зависимост от избора на потребителя.

12. Докажете, че даденият в началото на настоящата глава алгоритъм за сортиране на 3 числа работи правилно. За целта проверете всичките възможни случаи на първоначална наредба на стойностите в променливите.

КОНКУРСНА ЗАДАЧА

4.1. Тухла (Пролетен турнир, Пловдив, 2002, зад. D2). Тухла с правилна правоъгълна форма има дължина x, ширина y и височина z см. Тези размери са цели числа и не са по-големи от 1000 см. Напишете програма, която въвежда числата x, y и z чрез клавиатурата и извежда на екрана едно число, равно на минималната площ в кв. см, която трябва да се изреже от ламаринен лист така, че през получената дупка да може да мине тухлата.

Решение: Може да опитаме да проврем тухлата по три различни начина, като необходимата минимална площ се изразява в отделните случаи с произведенията `x*y`, `x*z` или `y*z`. Приложената програма пресмята в променливата `m` най-малкото от тези три произведения. Особеност е използването на типа `long int`, вместо `int`. Налага се от факта, че при някои компилатори (например Borland (Turbo) C++ 3.1) числата от тип `int` са твърде „къси“ и при въведени данни в диапазона до 1000 е възможно да се получат неверни резултати след извършване на умноженията. Тази особеност на компютърната аритметика ще бъде разгледана по-нататък в книгата.

```
#include<iostream.h>
long int x,y,z,m;
void main()
{
 cin >> x >> y >> z;
 m=x*y;
 if(x*z<m) m=x*z;
 if(y*z<m) m=y*z;
 cout << m << "\n";
}
```

Глава 5. СЪСТАВНИ ЛОГИЧЕСКИ УСЛОВИЯ

Съставни логически условия. Освен простите условия, представени във вид на отношения с равенства или неравенства, езикът C/C++ позволява при оператора `if` да бъдат използвани и съставни логически условия. Един начин да ги образуваме е да вземем две отношения, да ги заградим в кръгли скоби и да ги свържем с един от *логическите съюзи*: „и“, „или“, „не“.

Ако A и B са условия, то:

1) **(A) и (B)** е ново условие, което е вярно тогава и само тогава, когато и двете условия A и B са верни; в езика C/C++ това условие се означава с `(A)&&(B)`;

2) **(A) или (B)** е ново условие, което е вярно тогава и само тогава, когато поне едно от условията A или B е вярно; в езика C/C++ това условие се означава с `(A)||(B)`;

3) **не (A)** е ново условие, което е вярно тогава и само тогава, когато A не е вярно; в езика C/C++ това условие се означава с `!(A)`.

Примери: Нека `a` и `b` са числа, за които е изпълнено, че `a<b`. Тогава:

Съставното условие `(a<=x) && (x<=b)` е вярно точно за тези стойности на `x`, за които `x` принадлежи на интервала [`a`, `b`].

Съставното условие `(x<a) || (b<x)` е вярно точно за тези стойности на `x`, за които `x` лежи извън интервала [`a`, `b`].

Съставното условие `!(a<x)` е вярно точно за тези стойности на `x`, за които е изпълнено, че `x<=a`.

Съставните логически условия могат да се образуват не само от прости условия, а и от други съставни. Възможно е от две или повече съставни логически условия отново да получим друго съставно условие, като ги съединим с някои от логическите съюзи.

Пример 1. Разглеждаме трите числови променливи `a`, `b` и `c`. Напишете фрагмент от програма, който отпечатва стойността на най-голямата от тях.

Решение:

```
if((a>=b)&&(a>=c)) cout << a;
else if((b>=a)&&(b>=c)) cout << b;
else if((c>=a)&&(c>=b)) cout << c;
```

Пример 2. Дадени са положителните числа a, b, c и d. Да се напише програма, която проверява дали правоъгълник с дължини на страните a и b може да се постави вътре в правоъгълник, чиито страни са с дължини c и d, така че всяка от страните на единия правоъгълник да е успоредна или перпендикулярна на страните на другия.

Решение: За може да се направи поставянето, трябва да е изпълнено, че числата в поне една от двойките a, b или b, a са съответно по-малки от числата

в двойката c, d. С други думи, трябва да е изпълнено поне едно от двете съставни условия

$(a < c)$ и $(b < d)$ или $(b < c)$ и $(a < d)$.

Така намираме, че за успешното поставяне трябва да е изпълнено съставното условие, съставено от двете по-горе написани съставни условия, т.е.

$((a < c)$ и $(b < d))$ или $((b < c)$ и $(a < d))$.

Записано на езика C/C++ съставното условие изглежда така:

```
((a < c) && (b < d)) || ((b < c) && (a < d))
```

Цялата програма е следната:

```
#include<iostream.h>
void main()
{ int a,b,c,d;
  cin >> a >> b >> c >> d;
  if(((a < c) && (b < d)) || ((b < c) && (a < d)))
   cout << "Да, може\n";
  else cout << "Не, не може\n";
}
```

Пример 3. Да се напише програма, която проверява дали през правоъгълно отверстие с дължини на страните a и b може да премине тухла с дължини на ръбовете x, y, z, така че всеки ръб да е успореден или перпендикулярен на страните на правоъгълното отверстие.

Решение: За да премине тухлата, тя трябва да се завърти така, че да се намери поне една двойка измежду шестте двойки $(x, y), (y, x), (x, z), (z, x), (y, z)$ и (z, y), в която двете числа са съответно по-малки от a и b.

```
#include<iostream.h>
void main()
{ int a, b, x, y, z;
  cin >> a >> b;
  cin >> x >> y >> z;
  if( ((x<a)&&(y<b)) ||
      ((y<a)&&(x<b)) ||
      ((x<a)&&(z<b)) ||
      ((z<a)&&(x<b)) ||
      ((y<a)&&(z<b)) ||
      ((z<a)&&(y<b))
    )
    cout << "Да, може да премине\n";
 else cout << "Не, не може да премине\n";
}
```

Проверка дали годината е високосна. Както знаем, една година g е високосна, когато g се дели на 4 без остатък, като специално правило се прилага, когато последните две цифри на g са нули. Тогава годината е високосна, ако g се дели на 400 без остатък. Това може да се изрази чрез съставно логическо условие:

```
int g;
cin >> g;
if(((g%4==0)&&(g%100 != 0))||(g%400==0))  cout << "високосна";
else cout << "не е високосна";
```

Например високосни са 2000, 2004, 2008 г., а 1700, 1800 и 1900 – не са.

Намиране на следваща дата. Ще разгледаме задачата за написване на програма, която по зададена дата, състояща се от ден d, месец m и година g, намира датата на *следващия ден.* Предполагаме, че трите стойности d, m и g се въвеждат от клавиатурата като цели числа и че формират правилна (съществуваща) дата.

Ако денят не е последен в месеца, той трябва да бъде увеличен с 1, в противен случай търсеният ден трябва да е първият за следващия месец. При това обаче, ако въведеният месец е декември (т. е. има номер 12), следващият месец трябва да бъде януари (т.е. да има номер 1) и номерът на годината да се увеличи с 1. По-специални действия трябва да се направят, когато въведената дата е през февруари и годината е високосна. За целта в променливата p предварително се пресмята кой е последният ден от месеца:

```
if(m==2)
 { if(((g%4==0)&&(g%100 != 0))||(g%400==0)) p=29;
   else p=28;}
else if((m==4)||(m==6)||(m==9)||(m==11)) p=30;
else p=31;
```

След това се извършва основното пресмятане:

```
if(d<p)d++;
else
{d=1;
 if(m<12)m++;
 else{m=1; g++;}
}
```

Стойност на условията. Условията, които се употребяват в оператора if, независимо дали са прости, или сложни, също имат стойност. Тя може да се види чрез оператора cout. Например cout << (2<3); ще отпечата 1, а операторът cout << (2>3); ще отпечата 0. Общото правило е, че стойността на всяко условие е 1, тогава и само тогава, когато условието е изпълнено (т.е. – вярно); в противен случай стойността е 0.

Понякога, ако стойността на условието е 0, казваме, че стойността е „лъжа“, а когато е 1 – че е „истина“.

Самите стойности 0 и 1 могат да се употребяват като условие в оператора `if`. Така операторът P, фигуриращ в условния оператор `if(1)`P; винаги ще бъде изпълняван (и тогава операторът `if` всъщност се превръща в безусловен), а операторът Q, фигуриращ в `if(0)`Q`;` – никога няма да действа.

Ще отбележим, че за оператора `if(...)` няма значение дали стойността, която се е получила в кръглите скоби, е 1, или е произволно друго ненулево число – операторът ще извърши едно и също действие. Всъщност той различава само нулева и ненулева стойност и интерпретира като единица всяка стойност, която е различна от нула. Затова програмистите често вместо `if(x != 0)` пишат по-краткото `if(x)`. Обяснява се с факта, че стойността на условието `(x != 0)` е нулева или ненулева в едни и същи случаи, когато съответно е нулева или ненулева стойността на `x`. Аналогично, вместо `if(x==0)` може да напишем `if(!x)`.

Проверка за равносилност (еквивалентност). За да проверим дали две съставни условия са *равносилни*, т.е дали условията имат винаги една и съща стойност при всички възможни стойности на променливите в тях (и следователно може да заменим едното условие с другото в оператора `if`), преглеждаме всички възможности за стойности на променливите им. Например за условията `x==0` и `!x` всичките възможности за стойности на `x` са две: 1) `x` има нулева стойност; 2) `x` има ненулева стойност. Съставяме таблица с тези стойности:

	`x` има нулева стойност	`x` има ненулева стойност
стойност на условието `x==0`	1	0
стойност на условието `!x`	1	0

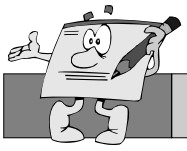
От таблицата се вижда, че двете условия са равносилни, защото имат една и съща стойност, както и да избираме стойността на `x`.

Таблици за стойностите на основните видове съставни условия:

(A) и (B)	$A=0$	$A=1$
$B=0$	0	0
$B=1$	0	1

(A) или (B)	$A=0$	$A=1$
$B=0$	0	1
$B=1$	1	1

	$A=0$	$A=1$
не (A)	1	0



УПРАЖНЕНИЯ

В упражненията по-долу, както и в упражненията от следващите глави в книгата, се подразбира, че трябва да се напише програма, с която да се реши задачата (освен, ако изрично е казано друго):

1. Дадено е число n. Ако числото е нечетно и след удвояването му не надминава 30000, го удвоете, иначе го оставете без изменение.

2. Дадено е трицифрено число n, в записа на което няма цифра нула. Съставете програма, която проверява дали числото n е кратно на всяка своя цифра.

3. Проверете има ли сред три дадени числа равни.

4. Дадено е трицифрено число. Определете дали:

а) сумата от неговите цифри е двуцифрено число;

б) произведението от неговите цифри е трицифрено число;

в) произведението от цифрите му е по-голямо от друго дадено число;

г) сумата от цифрите му е кратна на 5;

д) сумата от неговите цифрите е кратна на друго дадено число;

е) е *палиндром*, т.е. дали се чете еднакво отляво-надясно и отдясно-наляво.

5. От клавиатурата се въвеждат две числа. Съставете програма, която сравнява числата и в зависимост от резултата извежда на екрана подходящия знак („<“, „>“, „=“).

6. Как се образуват съставните логически условия?

7. Определете каква е стойността (истина или лъжа) на следното логическо условие:

```
((A==0) && (! (B==0))) || ((! (A==0)) && (B==0))
```

при стойности на променливите:

а) `A=0`, `B=0`; б) `A=0`, `B=1`.

Съставете таблица за стойностите.

8. Каква ще бъде стойността на `x`, след изпълнение на операторите:

```
x = 0;
if (x) x = x+2; else x = 2;
```

9. Какво ще отпечатат операторите:

а) `cout << ((3<8)&&(2>5));`

б) `cout << (!(3==3))|| (8>=12);`

в) `cout << (0|| (2-3<0));`

10. Напишете програма, с която се въвежда времето от деня (в часове) и в зависимост от въведения час, по ваша преценка, програмата пожелава „добро утро“, „добър ден“, „добър вечер“ или „лека нощ“.

11. Да се напише програма, с която се въвеждат дължините на страните на триъгълник. Програмата да извежда `0`, ако не съществува триъгълник с въведените дължини на страните. Ако такъв триъгълник съществува, да изведе съобщението „Равностранен“, „Равнобедрен“ или „Разностранен“, в зависимост от вида на триъгълника.

Упътване: За да съществува триъгълник със страни `a`, `b`, `c` трябва да бъде изпълнено условието:

`(a>0)&&(b>0)&&(c>0)&&(a+b>c)&&(a+c>b)&&(b+c>a)`

Глава 6. ИЗБОР ОТ НЯКОЛКО ВЪЗМОЖНОСТИ

Оператор за избор. Ако трябва да се избере една от две възможности, това може да се направи чрез оператора `if` ... `else`. При избор между повече от две възможности може да използваме вложени оператори `if`. Например, ако програмата трябва да отпечата с думи съдържанието на променливата `x`, за която знаем, че е ограничена да приема само стойностите 1, 2 или 3, може да използваме фрагмента

```
if(x==1) cout << "едно";
else if(x==2) cout << "две";
else if(x==3) cout << "три";
```

Съществува друга конструкция, която се предпочита при обработване на няколко възможни целочислени стойности на `x`. Тя заменя горните вложени оператори и представа за вида ѝ може да получим от следния фрагмент:

```
int x;
cin >> x;
switch(x)
{
 case 1: cout << "едно"; break;
 case 2: cout << "две"; break;
 case 3: cout << "три"; break;
}
```

Един по-общ вид на *оператора за избор* е следният:

```
switch(X)
{
 case C1: P11; P12; ...; break;
 case C2: P21; P22; ...; break;
 ...
 case CN: PN1; PN2; ...; break;
 default: Q1; Q2; ...;
}
```

Думите `switch`, `case`, `default` и `break` са служебни думи[1] от езика C/C++. X е променлива или израз, който има целочислена стойност или приема стойност от друг изброим тип. Един такъв тип, различен от целочисления, ще бъде разгледан по-долу. X се нарича *селектор*.

Между фигурните скоби след `switch(`X`)` се записват клаузи, започващи с думата `case`, след която непосредствено следва една от константните стой-

[1] За приблизителното им произношение виж Приложение 3

ности (или константни изрази) $C1$, $C2$, ..., CN, всяка от които може да бъде стойност на селектора X.

Съществува и специална клауза, започваща с думата `default` и при нея няма константен израз. Тази клауза не е задължителна и може да не присъства.

Действието на оператора за избор е следното: Пресмята се стойността на X и ако тя е равна на някоя от стойностите, написани между думите `case` и двоеточията, се изпълняват последователно съответните оператори, написани там, докато се срещне думата `break`. Ако стойността на X не съвпада с нито една от написаните между думите `case` и двоеточията стойности, се изпълняват операторите след `default`. Ако обаче думата `default` не присъства, в този случай се изпълнява операторът, следващ след затварящата фигурна скоба, т.е. следващият написан в програмата оператор след разглеждания `switch`.

Следният пример показва как се обработва случаят, когато стойността на `x` може да не е 1, 2 или 3. Тогава ще се отпечата съответното съобщение:

```
switch(x)
{
 case 1: cout << "едно"; break;
 case 2: cout << "две"; break;
 case 3: cout << "три"; break;
 default: cout << "стойността не е 1, 2 или 3";
}
```

Общата форма на правилата, с които се оформя операторът за избор, позволява на мястото на единичния `case` да се напишат няколко, отделени с двоеточия. Без да уточняваме как трябва да изглежда по-общата конструкция, като пример ще посочим фрагмент, който отпечатва дали едно цяло число `x` от интервала 0, .., 9 е нечетно (odd), или е четно (even):

```
switch(x)
{
 case 1: case 3: case 5: case 7: case 9:
 cout << "odd"; break;
 case 0: case 2: case 4: case 6: case 8:
 cout << "even"; break;
}
```

Знаков тип данни. Освен простите числени типове данни, съществува един основен нечислов тип данни, който се отнася към изброимите типове в езика C/C++. Това са данните от *знаков тип*. За този тип се употребява названието char (съкращение от англ. дума „character“ – „знак“ и се произнася „чар“ или „кер“). Когато една променлива е определена с този тип,

```
char c;
```

тя приема стойности, които са знаци. Такива могат да бъдат например буквите от латинската или българската азбука или специалните (препинателните) знаци. Присвояването на тези стойности става чрез заграждането им в единични кавички, например `c='x'`. За променлива от този тип може да се употреби операторът `cin >> c;` – за прочитане от клавиатурата и операторът `cout << c;` – за отпечатване на екрана.

Знакова стойност. Възможните стойности, които може да приеме една променлива `c` от тип `char`, са 256 на брой и са дадени в таблицата на ASCII-кодове в Приложение 2. Показани са само *видимите знаци*, които имат номера от 32 до 255. Знакът с номер 32 е *празният интервал*, наречен още *шпация*. Знаците с номера от 0 до 31 обикновено не служат за извеждане на екрана, а имат специални управляващи функции, които тук не разглеждаме.

Номер на знак. От таблицата в Приложение 2 става ясно, че на всеки знак се съпоставя еднозначно номер, т.е. всичките знаци са номерирани. Например знакът, съответстващ на главната латинска буква 'A', има номер 65, а знакът '@' е с номер 64.

В програма може да отпечатаме номера на един знак, като извършим предварително присвояване с променлива от тип `int`. Впрочем, така правилно ще се отпечатат номерата на тези знаци, които са разположени от началото на таблицата до първия знак в реда, започващ с номер 128. Причината за тази особеност тук няма да разглеждаме.

```
char c='A';
int i=c;
cout << i << "\n";
```

Да отбележим, че трябва да правим разлика между числените стойности 0, 1, 2, ..., 9, които се присвояват на променлива от тип `int` и знаковите стойности на цифрите '0', '1', '2', ..., '9', които се присвояват на променлива от тип `char`.

Получаване на число от цифра. Нека променливата `c` е от тип `char` и е приела стойност, равна на знак, който е някоя от цифрите '0', '1', '2', ..., '9'. Получаването на числовата стойност на този знак може да стане чрез израза:

```
int i = c-'0';
```

Правилността на горната формула следва от факта, че номерата на цифрите в таблицата на ASCII-кодове са подредени в същата последователност, както и самите цифри.

Променлива от знаков тип може да се използва за селектор в оператора за избор.

Римски цифри. Както знаем, римските цифри се означават с главните латински букви I, V, X, L, C, D и M и съответните им стойности са 1, 5,

10, 50, 100, 500 и 1000. Следващата програма въвежда една римска цифра и извежда съответната ѝ стойност в десетична бройна система:

```
#include<iostream.h>
char c;
void main()
{cin >> c;
 switch(c)
 {
  case 'M': cout << 1000; break;
  case 'D': cout <<  500; break;
  case 'C': cout <<  100; break;
  case 'L': cout <<   50; break;
  case 'X': cout <<   10; break;
  case 'V': cout <<    5; break;
  case 'I': cout <<    1; break;
 }
}
```

Отпечатване на числата от 1 до 9 с римски цифри. Следващият програмен фрагмент показва какво всъщност е съответствието между тези два начина за записване на числата от 1 до 9:

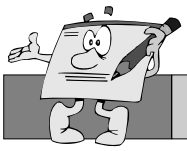
```
int i;
cin >> i;
switch(i)
{
 case 1: cout << "I"; break;
 case 2: cout << "II"; break;
 case 3: cout << "III"; break;
 case 4: cout << "IV"; break;
 case 5: cout << "V"; break;
 case 6: cout << "VI"; break;
 case 7: cout << "VII"; break;
 case 8: cout << "VIII"; break;
 case 9: cout << "IX"; break;
}
```

Избор на вариант. С оператора за избор се илюстрира идеята как се взема решение при определени обстоятелства, когато са възможни много варианти. Например да се реши какво действие да се извърши, в зависимост от въведена стойност от клавиатурата.

Представената по-долу програма въвежда две числа от клавиатурата, след което, според вида на следващия натиснат клавиш: '+', '-', '*' или '/',

пресмята сумата, разликата, произведението или частното на тези две числа. В програмата има и елементарна обработка на грешката, която може да възникне, ако потребителят вместо някой от четирите знака за аритметични операции въведе друг знак или се опита да дели на нула.

```
// program calculator
#include<iostream.h>
double a,b,c;
char op;
int res;

void main()
{
 cout << "Въведете първото число: ";
 cin >> a; cin.get();
 cout << "Въведете второто число: ";
 cin >> b; cin.get();
 cout << "Въведете + - * /\n";
 cin >> op; cin.get();
 res=0;
 switch(op)
  {
   case '+': c=a+b; res=1; break;
   case '-': c=a-b; res=1; break;
   case '*': c=a*b; res=1; break;
   case '/': if(b!=0.0){c=a/b; res=1;} break;
  }
if(res==1) cout << "Резултатът е " << c;
else cout << "Невалидна операция";
cin.get();
}
```



УПРАЖНЕНИЯ

1. Какво представлява селекторът в оператора за избор?
2. Напишете програма, която по зададени месец и година определя броя на дните в месеца.
3. Напишете програма, която по въведен номер на месец от годината извежда с думи сезона.

4. Напишете програма, която въвежда буква от клавиатурата и отпечатва съобщение дали буквата е гласна, или съгласна.

5. Какво не е правилно във фрагмента, който започва така:

```
double f;
cin >> f;
switch (f)
{
 case 10.05: ...
```

6. Въведете число $X (X \leqq 1000)$ и след това изведете същото число, представено на естествен език. Например при въведено 7 програмата трябва да изведе „седем“; при въведено 243 програмата трябва да изведе „двеста четиридесет и три“.

7. Променете следната конструкция с оператори `if...else` в равносилна конструкция с оператора `switch`:

```
if (ch == 'L') cout << "Load";
else if (ch == 'S') cout<< "Save";
else if (ch == 'E') cout<< "Enter";
else if (ch == 'D') cout<< "Display";
else if (ch == 'Q') cout<< "Quit";
```

8. Проверете дали следващата програма правилно отпечатва с римски цифри всяко въведено цяло число между 1 и 9.

```
#include<iostream.h>
char c1='I', c2='V', c3='X';
int v;
void main()
{ cin >> v;

  switch(v)
  {case 9 :
   case 4 : cout << c1;
  }

  switch(v)
  {case 9 : cout << c3; break;
   case 8 :
   case 7 :
   case 6 :
   case 5 :
   case 4 : cout << c2;
  }
```

```
  switch(v)
  {case 8 : cout << c1;
   case 7 : cout << c1;
   case 6 : cout << c1; break;
   case 3 : cout << c1;
   case 2 : cout << c1;
   case 1 : cout << c1;
  }
}
```

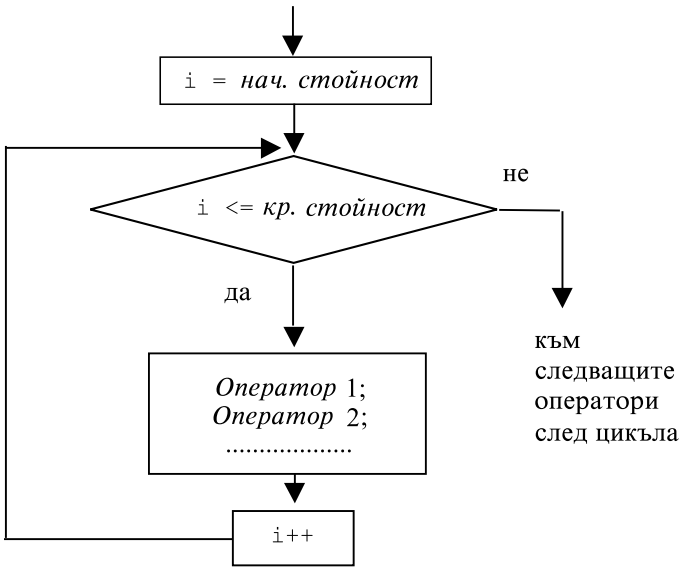
Глава 7. ПОВТАРЯЩИ СЕ ДЕЙСТВИЯ

Езиците за програмиране предлагат удобни начини за представяне на повтарящи се действия. Всъщност наличието на такива средства е една от най-важните възможности при програмирането. Например вместо да записваме последователно 100 пъти оператора `cout <<"*";`, с което ще се отпечатат 100 звездички на екрана, може да използваме само един оператор за оформяне на *цикъл*:

```
for(int i=1; i<=100; i++) {cout << "*";}
```

В тази конструкция името на оператора за цикъл е `for`, а управлението на цикъла се осъществява чрез трите израза, написани между знаците точка и запетая `(  ;  ; )` в заграденото от кръгли скоби място след думата `for`. Операторът във фигурните скоби се нарича *тяло* на цикъла. Възможно е в тялото да участват повече оператори, а не само един. Когато там използваме само един оператор, може да не употребяваме фигурните скоби. По-общ вид на този оператор за цикъл (но не най-общият) е:

`for(int i =` *начална стойност*`; i<=` *крайна стойност*`; i++)`
`{`*Оператор*1; *Оператор*2`;...}`



Вместо буквата `i` може да се използва произволно друго означение за променлива. Обявата на типа `int` в някои случаи може да не присъства – например, когато променливата `i` е била вече обявена с този тип. Т.е. операторът може да изглежда и така:

`for(i =` *начална стойност*`; i<=` *крайна стойност*`; i++)`
`{`*Оператор*1; *Оператор*2;...`}`

Променливата `i` в цикъла (наричана също *параметър* на цикъла) играе роля на *брояч* – след всяко поредно изпълнение на операторите в тялото на цикъла тя се увеличава с единица. Тялото на цикъла се изпълнява за всяка една от целочислените стойности на `i`, започвайки от началната стойност и завършвайки с крайната.

Например следният оператор ще отпечата целите числа от 1 до 100, като ги разделя с по един празен интервал:

```
for(int i=1; i<=100; i++)
 {cout << i; cout << " ";}
```

Възможно е за начална и крайна стойност на брояча, освен константи, да се ползват променливи или изрази, като в повечето случаи се препоръчва те да са такива, че след пресмятането им да получат целочислена стойност. Така приложенията на описания начин за задаване на повтарящи се (циклични) пресмятания се увеличават съществено. Следващият пример показва как може да пресметнем сумата на целите числа от 1 до N, където N се въвежда от клавиатурата:

```
int N;
cin >> N;
int S=0;
for(int i=1; i<=N; i++) S = S + i;
cout << S;
```

Пример 1. Сумиране на числа, въведени от клавиатурата. Да се напише програма, която въвежда от клавиатурата цялото число $N > 0$, след това въвежда още N числа и отпечатва сумата им.

```
#include <iostream.h>
void main()
{
 int N;
 double A, S;
 cin >> N;
 S=0.0;
 for(int i=1; i<=N; i++)
 {
  cin >> A;
  S += A;
 }
 cout << S;
}
```

Пример 2. Намиране на най-голямото число измежду няколко числа, въведени от клавиатурата. Да се напише програма, която въвежда от клавиатурата цялото число $N > 0$, след това въвежда още N числа и отпечатва най-голямото от тези въведени N на брой числа:

```
#include <iostream.h>
void main()
{
 int N;
 double A, M;
 cin >> N;
 cin >> M;
 for(int i=2; i<=N; i++)
 {
  cin >> A;
  if(A>M) M = A;
 }
 cout << M;
}
```

Да отбележим, че в този пример, за разлика от предишния, променливата в цикъла се изменя от 2 до N, а не от 1 до N. С първото въведено число се зарежда променливата `M` и след това всеки път, когато се окаже, че въведеното от клавиатурата число `A` е по-голямо от `M`, програмата прави промяна, като стойността на `A` се присвоява на `M`. Така накрая `M` ще съдържа най-голямото от всички въведени числа.

В по-общия случай разглежданият оператор за цикъл може да има вида

`for(i =` *израз_1* `; i<=` *израз_2* `; i++) {` *Тяло* `}`

където най-често (но не задължително) двата израза – *израз_1* и *израз_2*, имат целочислена стойност и вторият израз не зависи пряко или косвено от стойността на параметъра на цикъла. Тогава последователността от пресмятания се образува по следния начин: означаваме съответно с a и b целочислените стойности на *израз_1* и *израз_2*. Ако $a \leqq b$, операторът започва своята работа, като променливата `i` последователно приема стойности, равни на a, $a + 1$, $a + 2$, ..., b, и за всяка от тези стойности се изпълнява по веднъж тялото на цикъла. Ако $a > b$, тялото на цикъла въобще няма да бъде изпълнено.

Пресмятане на факториел. Произведението на последователните цели числа от 1 до n се използва често и за него е прието названието n факториел, а математическото означение е $n!$ (n, последвано от удивителен знак !). Например $3! = 1{\cdot}2{\cdot}3 = 6$ и $5! = 1{\cdot}2{\cdot}3{\cdot}4{\cdot}5 = 120$.

За да натрупаме в променливата `f` стойността на $n!$ използваме програмен

фрагмент, сходен с фрагмента за пресмятане на сумата от няколко последователни цели числа, но операцията събиране е заменена с умножение:

```
int f=1;
for(i=1; i<=n; i++) f = f*i;
cout << f;
```

Степенуване. Пресмятането на степента a^b, т.е. намирането на произведението, получено от умножаването на числото a със себе си b пъти, програмираме чрез следния цикъл ($a^b = \underbrace{a \cdot a \cdot \cdots \cdot a}_{b}$):

```
int p=1;
for(i=1; i<=b; i++) p = p*a;
cout << p;
```

Забелязваме, че в тялото на този цикъл не участва параметърът `i`. Сега този параметър всъщност е използван само за *брояч*, който изброява повторенията на тялото, т.е. на оператора `p = p*a;`.

Цикъл `for`, при който броячът намалява. При някои пресмятания е по-удобно да използваме цикъл с намаляващи стойности на параметъра:

```
for(int i = начална стойност; i>= крайна стойност; i--)
{ Тяло }
```

Тази конструкция се различава от предишните по присъствието на `i--`, вместо `i++` и знаците `>=`, вместо `<=`. В този вид операторът за цикъл се изпълнява по следния начин: ако целите числа a и b са съответно *началната стойност* и *крайната стойност* и $a \geqq b$, операторът започва своята работа, като променливата `i` последователно приема стойности, равни на $a, a-1, a-2, ..., b$, и за всяка от тези стойности се изпълнява по веднъж тялото на цикъла. Ако $a < b$, тялото на цикъла въобще няма да бъде изпълнено.

Чрез тази форма на оператора можем да отпечатаме числата от 1 до 9, но в обратен ред:

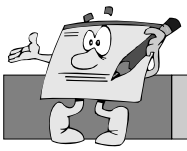
```
for(int i = 9; i>= 1; i--) cout << i << " ";
```

Цикъл `for`, при който параметърът се променя по зададен начин. Възможно е след втория знак точка и запетая в скобите след думата `for` вместо `i++` или `i--` да се напише по-сложен израз, с който да се укаже как да се променя параметърът на цикъла в края на всяка стъпка. Например вместо да се увеличава с единица (което се указва с `i++`), може да искаме параметърът `i` да се увеличава с определена константна стойност `c`. Това може да се направи, като се напише `i=i+c` или `i += c`. Например следният оператор отпечатва нечетните числа от 1 до 11:

```
for(int i=1; i<=11; i +=2) cout << i << " ";
```

Същият оператор няма да промени действието си, ако фигуриращото при него условие `i<=11` заменим с `i<12`.

Възможна е употребата на още по-обща и със значително по-големи възможности форма на оператора `for`, която тук не разглеждаме.



УПРАЖНЕНИЯ

1. В програмния фрагмент

```
s = 0;
for(i = 5; i <= 3; i++) { s += i;}
```

колко пъти ще се изпълни тялото на цикъла?

2. Защо в програмата за пресмятане на факториел на числото n и в програмата, извършваща степенуване, първоначално на променливата `f` и съответно на `p`, е присвоена стойност 1?

3. Колко пъти ще се изпълни тялото на циклите, които имат следното начало:

а) `for (I=-3;I<=17;I++) ...`

б) `for (j=-17;j>=-3;j--) ...`

в) `for (h='a';h<='z';h++) ...`

4. По какво се различават цикъл `for`, при който броячът се увеличава, и цикъл `for`, при който броячът намалява?

5. Какво е предназначението на дадените цикли и каква ще бъде последната стойност на променливата s?

а)

```
s = 0;
for (j=1;j<=5;j++) s = s + 2*j;
```

б)

```
s = 0;
for (j=1;j<=20;j++) {if (j%3 == 0) s = s + j;}
```

6. От колко оператора може да се състои тялото на цикъл?

7. Да се изведат всички цели числа от N до 1, всяко на различен ред.

8. Да се изведат всички цели числа от 5 до 100, които се делят на 5.

9. Отпечатайте в стълб:

а) всички цели числа от 20 до 35;

б) квадратите на всички цели положителни числа от 10 до `b` (стойността на `b` се въвежда от клавиатурата; `b <= 10`);

в) третите степени на всички цели положителни числа от `a` до 5 (стойността на `a` се въвежда от клавиатурата; `a <= 5`);

г) всички цели числа от `a` до `b` (стойностите на `a` и `b` се въвеждат от клавиатурата; `b>=a`).

10. Отпечатайте в стълб таблицата за умножение със 7.

11. Да се напише програма, с която се пресмята сумата

$$1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/10.$$

12. Пресметнете произведението на a и b, като в програмата не използвате други аритметични операции, освен операцията събиране.

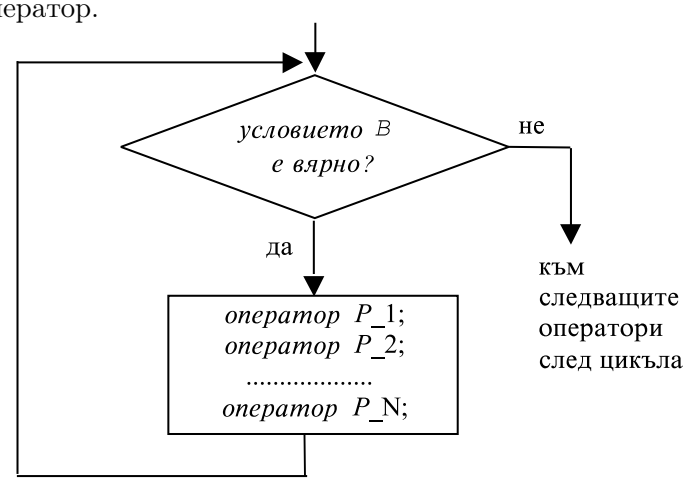
Глава 8. ЦИКЪЛ С УСЛОВИЕ

Разгледаната форма на оператора за цикъл `for` използваме предимно, когато предварително е известен броят на повторенията в цикъла. Когато този брой се определя от събития, които настъпват по време на работата на цикъла, е подходящо да използваме други средства за оформяне на многократни пресмятания – оператора `while` и неговата разновидност `do...while`.

Общият вид на първия от споменатите по-горе оператори за цикъл е:

`while(` B `) {` P_1`;` P_2`; ...` P_N`; }`

При него B е *условието* на цикъла, а P_1; P_2; ... P_N; са оператори, образуващи *тялото* на цикъла. Запазената дума „while“ (произнася се „уайл“) в превод от английски означава „докато“. Цикълът се изпълнява по следния начин: проверява се условието B и ако то е вярно, се изпълняват операторите от тялото на цикъла. След това отново се проверя условието B и ако то е вярно, отново се изпълняват операторите от тялото и т.н. Тази последователност от действия се повтаря неколкократно и спира щом при поредната проверка се окаже, че условието B е станало невярно. След спирането на цикъла (употребява се още изразът „излизане от цикъла“), изпълнението на програмата продължава със следващия след цикъла оператор. Ако условието B е невярно в самото начало, преди да започне изпълнението на цикъла, тогава цикълът въобще не се изпълнява и направо се преминава към следващия след него оператор.



Условието B е от същия вид, както е при оператора `if`. Както и на други места в езика C/C++, ако вместо няколко оператора в тялото на цикъла искаме да напишем само един, може да не го заграждаме с фигурни скоби.

За да има смисъл използването на цикъла `while(` B `) {` P_1`;` P_2`; ...` P_N`; }`, трябва действието на операторите в неговото тяло обикновено да

води пряко или косвено до промяна в стойностите на някои променливи в програмата (които участват в условието B). Това е необходимо, за да може в определен момент да се промени верността на условието B. В противен случай, цикълът може да се изпълнява „безкрайно“ много пъти и програмата ще изпадне в състояние на *зацикляне*, което обикновено не е желателно.

Пример 1. Безкраен цикъл е `while(1);`. Ако го напишем в програма и я изпълним, ще се наложи да я прекъснем аварийно (например с използване на клавишната комбинация Ctrl-Break при работа в средата Borland C++).

Пример 2. При начална стойност на променливата `x`, равна на числото 5, тялото на следващия цикъл ще се повтори 5 пъти, защото след толкова повторения тази променлива ще стане равна на 0 и условието на цикъла ще стане невярно (от началото до петия път включително, това условие остава вярно):

```
int x=5;
while(x > 0) x--;
```

Пример 3. Следният фрагмент ще доведе до зацикляне:

```
int x=1;
while(x > 0) x=1;
```

Пример 4. *Редица на Фибоначи* (италиански учен от 13. век) се нарича редица от цели числа, първите две от които са единици, а всяко следващо, като се започне от третото, се получава като сума на двете стоящи непосредствено преди него числа от редицата. Лесно можем да напишем първите няколко елемента на тази редица:

1, 1, 2, 3, 5, 8, 13, 21, ...

Следният програмен фрагмент отпечатва последователно числата от редицата на Фибоначи, докато намери число от тази редица, което е по-голямо от 1000:

```
int a=1; cout << a << " ";
int b=1; cout << b << " ";
while(b<=1000)
{ int c=a+b; cout << c << " ";
  a=b;
  b=c;
}
```

Чрез оператора за цикъл `while` може да програмираме същото, каквото и с оператора `for`. Например фрагментът

```
for(int i=1; i<N; i++) cout << i << " ";
```

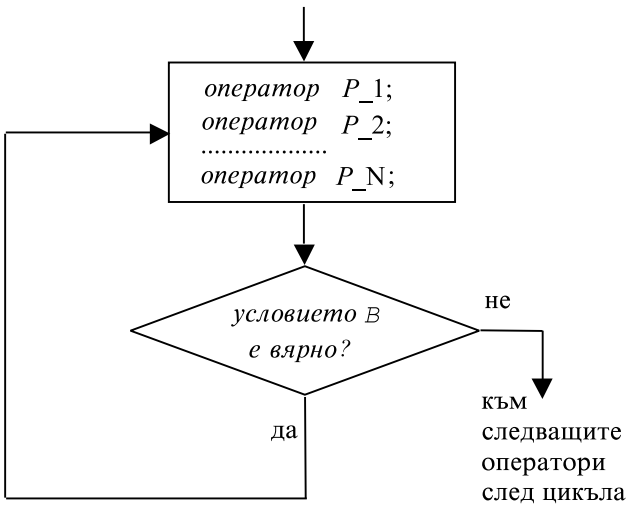
ще извърши същите действия, както фрагмента

```
int i=1;
while(i<N) {cout << i << " "; i++;}
```

Оператор за цикъл с проверка на условието след тялото му. Понякога е по-удобно условието за излизане от цикъла да се проверява след изпълняване на тялото му. Тогава е уместно да използваме друга конструкция за цикъл:

```
do{ P_1; P_2; ... P_N; }
while( B);
```

При нея условието B и тялото { P_1; P_2; ... P_N; } са написани в обратен ред спрямо предишния оператор за цикъл. Запазената дума „do“ (произнася се „ду“) в превод от английски означава „направи“. Този оператор за цикъл се изпълнява по следния начин: извикват се последователно операторите от тялото и след това се проверява условието B. Ако то е вярно, продължава се отново (поне още веднъж) изпълнението на тялото на цикъла; ако условието B не е вярно, по-нататъшното изпълнение на тялото на цикъла се прекратява и се продължава със следващия оператор, написан след този оператор за цикъл.



За разлика от първата разновидност на оператора `while`, сега тялото на цикъла винаги ще се изпълни поне веднъж. И сега действието на операторите от тялото на цикъла в почти всички случаи би трябвало да води пряко или косвено до промяна в стойностите на някои променливи в програмата, за да може да се промени в определен момент и верността на условието B – това условие би трябвало да зависи от тези променливи.

Пример 5. Да се напише програма, която чете от клавиатурата цели числа; прекратява този процес, когато бъде въведено числото нула, и след това отпечатва сумата на всички въведени числа.

```
#include<iostream.h>
void main()
{
 int a;
 int s=0;
 do {cin >> a; s +=a;} while(a!=0);
 cout << s;
}
```

Пример 6. Чрез следващия фрагмент програмата прочита от клавиатурата стойността на променливата `b`. След това чете от клавиатурата други числа, сумира ги и този процес се прекъсва, когато тяхната текущо натрупана сума стане по-голяма от стойността на променливата `b`:

```
int a, b, s=0;
cin >> b;
do {cin >> a; s +=a;} while(s<=b);
```

Вложени цикли. Когато в тялото на един цикъл поместим друг оператор за цикъл, получаваме интересната и често употребявана конструкция с вложени цикли. Във всяка стъпка на *външния цикъл* се изпълнят всички стъпки на *вътрешния*. Например фрагментът

```
for(i=1;i<=3;i++)
{ cout << "*";
  for(j=1;j<=5;j++) cout << j;
}
```

ще отпечата:

```
*12345*12345*12345
```

Пример 7. Програма, която отпечатва таблицата за умножение на двойките числа от 1 до 9:

```
#include<iostream.h>
#include<iomanip.h>
void main()
{
 for(int i=1; i<=9; i++)
 {
   for(int j=1; j<=9; j++)
     cout << setw(3) << i*j << " ";
   cout << "\n";
 }
}
```

След отпечатване с вътрешния цикъл на числата в един ред, на екрана се извежда символът `"\n"` за преминаване на нов ред. Включването `#include<iomanip.h>` е необходимо, поради използването на конструкцията

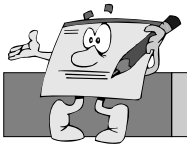
`setw(3)` при оператора за извеждане `cout`. По този начин се осигурява полето, в което ще се отпечатва числовата стойност, да има ширина от 3 позиции. Числата се разполагат подравнени надясно в тези позиции. Така едноцифрените числа ще се изведат с два празни интервала отпред, а двуцифрените – с един. Като цяло ще се изведе таблица, подравнена по колони:

```
1   2   3   4   5   6   7   8   9
2   4   6   8  10  12  14  16  18
3   6   9  12  15  18  21  24  27
4   8  12  16  20  24  28  32  36
5  10  15  20  25  30  35  40  45
6  12  18  24  30  36  42  48  54
7  14  21  28  35  42  49  56  63
8  16  24  32  40  48  56  64  72
9  18  27  36  45  54  63  72  81
```

Пример 8. Отпечатване таблица на ASCII-кодове. Следващата програма отпечатва таблица, сходна с дадената в Приложение 2. В тялото на вътрешния цикъл се пресмята стойността `n = 32+16*i+j` и се присвоява на променлива от тип `char`, за да се отпечата като знак, а не като число. Изменението на двата индекса (`i` – за външния цикъл и `j` – за вътрешния) е така подбрано, че стойността на получения израз `32+16*i+j` последователно пробягва без повторения всичките цели стойности от 32 до 256. Стойността 32 се получава при `i` = 0 и `j` = 0, а стойността 255 – при `i` = 13 и `j` = 15. Всеки път преди изпълнението на вътрешния цикъл се отпечатва число, което е равно на номера на първия знак, който ще се отпечата след това на реда. В края на реда се извежда `"\n"` за преминаване на нов ред.

```
#include<iostream.h>
void main()
{ char c;
  int n;
  for(int i=0;i<=13;i++)
  {
   n = 32+16*i;
   if(n<100) cout << " ";
   cout << n << ": ";
   for(int j=0;j<=15;j++)
    {n = 32+16*i+j;
     c = n;
     cout << c << " ";
    }
   cout << "\n";
  }
}
```

При изпълнение на горната програма „се гарантира“, че само тази част от отпечатаната от нея таблица на ASCII-кодове, която съдържа знаци с номера от 32 до 127, ще съвпадне със съответната част на таблицата от Приложение 2. Това е така, защото втората част от тази таблица не е стандартизирана еднакво за различните операционни системи и зависи от настройките им.



УПРАЖНЕНИЯ

1. Променете програмата за отпечатване на таблицата с ASCII-кодове, така че да се отпечатат номерата на знаците (а не знаците) в таблица, подравнена по колони.

2. Защо в програмата за отпечатване на таблицата с ASCII-кодове не се извеждат знаците с номера от 0 до 31? Какво ще се получи, ако се опитате да ги изведете?

3. За какво служи операторът `if(n<100) cout << " ";` в програмата за отпечатване на таблицата с ASCII-кодове?

4. Изпълнете програмата за отпечатване на таблицата с ASCII-кодове. Съвпада ли изцяло получената таблица с таблицата от Приложение 2? Виждате ли букви от кирилицата в получената таблица и къде са те?

5. Колко пъти ще се изпълни тялото на цикъла (всички променливи са от тип `int`)?

а)
```
B=1; x=27;
while (B)
 {B=(x>=3); x=x/8+1;}
```

б)
```
B=0; r=45;
while (!B)
 {B=(r==15); r=(r%4)+15;}
```

в)
```
B=1; x=64;
while (B)
 {B=(x>=4); x=x/6+2;}
```

6. Колко пъти ще се изпълни тялото на цикъла и кое ще бъде последното изведено число?

а)
```
int I;
I = -5;
do
{
 I++;
 cout << setw(3) << I;
}while (I <= 11 );
```

б)
```
int a = 3;}
while (a > 0)
{
 a--;
 cout << a << "\n";
}
```

7. Каква е разликата между оператора за цикъл `for` и циклите с условие?

8. В кои случаи с цикъл `for` лесно може да се изрази цикъл с условие?

9. Как може чрез цикъл с условие да се изрази цикъл `for`?

10. Решете някои избрани от вас задачи от глава 7 с помощта на цикъл `while` или `do...while`.

11. От какво зависи краят на циклите с неизвестен в началото брой повторения?

12. Каква е разликата между циклите, организирани с `while`, и циклите, организирани с `do...while`?

13. Отпечатайте всички числа на Фибоначи от интервала $[p, q]$. Опитайте в различни програми да използвате и трите вида оператори за цикъл.

14. Разполагаме с `B1` банкноти от по един лев, `B2` банкноти от по два лева и `B5` банкноти от по пет лева. Да се напише програма, която определя всички възможни начини за изплащане на сума `s`, като се използват банкноти от един, два и пет лева.

15. Да се напише програма, която извежда върху екрана всяка от следните две фигури от цифри и след това извежда двете заедно, както е показано:

```
1 2 3 4 5 6 7
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
1 2 3 4 5 6 7
  2 3 4 5 6 7
    3 4 5 6 7
      4 5 6 7
        5 6 7
          6 7
            7
```

КОНКУРСНИ ЗАДАЧИ

8.1. Звезди (Есенен турнир, Шумен, 2001, зад. D1). Да се въведат от клавиатурата числата M и N и да се изведат на екрана един до друг N „квадрата“, всеки със страна от M звездички ($M > 2, N > 0$). Например при $M = 5$ и $N = 3$ трябва да се получи следната картина:

```
* * * * * * * * * * * * * * *
*       *       *       *
*       *       *       *
*       *       *       *
* * * * * * * * * * * * * * *
```

Решение: Пресмятаме броя на звездичките в горната (и в долната) основа на цялата фигура: `N*M-N+1`. Отпечатваме тези звездички с цикъл в началото и в края на програмата. За отпечатването на останалите звездички използваме троен цикъл – външният с `M-2` повторения за редовете; следващият с `N`

повторения за звездичките по стените (без последната); най-вътрешният – с `M-2` повторения за празните позиции:

```
#include<iostream.h>
int M, N;
void main()
{ cin >> M >> N;
  int i,j,k;
  for(i=1;i<=N*M-N+1;i++) cout << "*";
  cout << "\n";
  for(i=1;i<=M-2;i++)
  {
    for(j=1;j<=N;j++)
    { cout << "*";
      for(k=1;k<=M-2;k++) cout << " ";
    }
    cout << "*\n";
  }
  for(i=1;i<=N*M-N+1;i++) cout << "*";
  cout << "\n";
}
```

Забележка: При големи стойности на M и N изобразяваните фигури може да не се съберат на екрана и да се изкривят.

8.2. Триъгълници (Есенен турнир, Шумен, 2002, зад. D1). Нека N е зададено число, не по-малко от 3 и не по-голямо от 20. Напишете програма, която въвежда от клавиатурата числото N и някакъв друг знак и след това извежда на екрана равностранен триъгълник, всяка от страните на който се състои от N пъти въведения от клавиатурата знак, така както е показано на примерите.

Пример 1	Пример 2	Пример 3
Ако въведете:		
`3`	`4`	`7`
`*`	`$`	`@`
програмата да изведе:		

`  *`
`* *`
`*****`
	`  $`
	`$ $`
	`$   $`
	`$ $ $ $`
		`  @`
		`@ @`
		`@   @`
		`@     @`
		`@       @`
		`@         @`
		`@ @ @ @ @ @ @`

Решение: Върхът на триъгълника отпечатваме след достатъчен брой празни интервали на реда (избрали сме `n-1`), за да може след това страните му да се разположат, без да излизат от екрана. Празните позиции се извеждат с първия оператор за цикъл. След това се извеждат `n-2` реда и във всеки от тях се отпечатват по два знака. За целта с първия от вътрешните цикли се отпечатва необходимият брой празни позиции преди първия знак (този брой намалява с единица при всеки следващ ред), а с втория вътрешен цикъл – необходимият брой празни позиции между първия и втория знак (този брой се увеличава с две единици при всеки следващ ред). Накрая се отпечатват знаците от основата на триъгълника.

```
#include<iostream.h>
void main()
{int a,b,i,j,n;
 char c;
 cin >> n >> c;
 for(j=1;j<=n-1;j++) cout << " ";
 cout << c << "\n";
 a=n-2; b=1;
 for(i=1;i<=n-2;i++)
 {
   for(j=1;j<=a;j++) cout << " ";
   cout << c;
   for(j=1;j<=b;j++) cout << " ";
   cout << c << "\n";
   a=a-1; b=b+2;
 }
 for(j=1;j<=n;j++) cout << c << " ";
 cout << "\n";
}
```

8.3. Минимакс (Пролетен турнир, Ямбол, 2003, зад. D2). На вход постъпва поредица от цели положителни числа между 1 и 30000, като не се знае предварително колко е дълга поредицата (тя може да е толкова дълга, че да не се събира в паметта на компютъра). Знае се обаче, че всяко число е на отделен ред и че поредицата завършва с числото 0. Напишете програма, която намира максималното и минималното число от поредицата.

Входните данни се четат от клавиатурата. На всеки ред имаме по едно цяло положително число между 1 и 30000 – поредният елемент от поредицата. Входът задължително завършва с ред, съдържащ числото 0.

Резултатът се извежда на екрана. На първия ред се извежда намереният максимален елемент от поредицата, а на втори ред се извежда намереният минимален елемент от поредицата.

Решение: В приложената програма цикълът `while` работи, докато въвежданите числа са различни от нула. В променливите `min` и `max` се получава съответно търсеният минимум и максимум. Първоначално `min` е заредена с възможно най-голямата стойност според условието на задачата, а `max` – с най-малката.

```
#include<iostream.h>
void main()
{ int min=30000, max=1;
  int a;
  cin >> a;
  while(a>0)
  { if(a<min)min=a;
    if(a>max)max=a;
    cin >> a;
  }
  cout << max << "\n" << min << "\n";
}
```

Глава 9. СЪЧЕТАВАНЕ НА ЦИКЛИ И РАЗКЛОНЕНИЯ

Когато в тялото на цикъл използваме условни оператори, възможностите ни за управление на процеса на пресмятанията значително се увеличават. Например фрагментът без условен оператор

```
int i=1;
while(i<=100) {cout << i << " "; i++;}
```

отпечатва последователните цели числа от 1 до 100, а фрагментът

```
int i=1;
while(i<=100)
 {if(i<10) cout << 0;
  cout << i << " "; i++;
 }
```

също ще отпечата последователните цели числа от 1 до 100, но едноцифрените от тях ще бъдат с *водеща нула* отпред.

Цикличен брояч. Често се използва целочислена променлива, чиито стойности в едни цикъл се променят циклично, например неколкократно се повтарят като последователността от числа 1, 2, 3, ..., p, където p е по-малко от максималната стойност на брояча на цикъла. В следващия фрагмент броячът на цикъла `i` се променя от 1 до 20, но стойността на извежданата променлива `j` се променя циклично от 1 до 5:

```
int j=1;
for(int i=1;i<=20;i++)
{
  cout << j << " ";
  j++;
  if(j>5) j=1;
}
```

Стойността на `j` се отпечатва в тялото на цикъла и след това се увеличава с единица, но когато се окаже, че тази стойност надмине 5, отново се извършва присвояването `j=1;`. Така *цикличният брояч* `j` по време на работата на цикъла приема стойностите:

```
1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

Излизане от цикъл с оператора `break`. Понякога се налага да излезем „веднага“ от тялото на цикъла – когато се е оказало, че определено условие е станало вярно. Това става с прилагане на оператора `break`, който от своя страна най-често се използва в тялото на оператор `if`. Операторът `break` може да се използва във всеки един от трите вида цикли `for`, `while` и `do...while`.

Пример 1. В следващия цикъл броячът `i` се променя от 1 до 10. В тялото на цикъла има оператор за въвеждане на число `j` от клавиатурата. Искаме, ако въведем стойност, която е по-голяма от текущата стойност на брояча в цикъла, да прекратим изпълнението на този цикъл (но ако и десетте пъти въведем стойност, която е по-малка или равна на текущата стойност на брояча, цикълът ще завърши по обичайния начин):

```
int i=1,j;
while(i<=10)
{ cout << "i=" << i << "\n";
  cin >> j;
  if(j>i) break;
  i++;
}
```

Пример 2. Може да комбинираме безкраен цикъл с оператор `break` в тялото му. Тогава единственият начин за излизане от цикъла е именно чрез оператора `break`. Следващият фрагмент чете в цикъл двойки числа от клавиатурата, отпечатва частното им, но когато се въведе делител, равен на нула, процесът се прекратява, преди да бъде извършено деление с нула.

```
double a,b;
while(1)
{ cin >> a >> b;
  if(b==0.0) break;
  cout << a/b << "\n";
}
```

Понякога операторът `break` служи за *аварийно излизане* от цикъл, както се вижда от предишния пример.

Изчерпващо търсене. Използването на операторите за цикъл се среща в програми за търсене, при което променливите се изменят в зададени граници. Пробягването на стойностите се реализира от оператор за цикъл, а проверката дали е открито това, което се търси, се прави чрез оператор за условен преход.

Пример 3. Да се намерят и отпечатат всички цели числа между 1 и 100, които се делят на 7.

За написването на програма, която решава задачата, използваме аритметичния оператор, който пресмята *остатъка при действието деление*. Нека `a` и `b` са две променливи. Както вече беше споменато в предните глави, в езика C/C++ операторът за пресмятане на остатък е означен чрез знака %. Т.е. остатъкът от делението на стойността на `a` със стойността на `b` се записва като израз по следния начин: `a%b`. Ясно е, че ако остатъкът от делението на числото a със 7 е равен на 0, то числото a се дели на 7, а при стойност на

остатъка, различна от 0, числото `a` не се дели на 7. Като имаме предвид, че споменатият остатък се представя с израза `a%7`, може да запишем следния програмен фрагмент за решаване на задачата:

```
for(int a=1;a<=100;a++)
 if(a%7==0) cout << a << " ";
```

Завършената програма ще отпечата следната редица:
`7 14 21 28 35 42 49 56 63 70 77 84 91 98`

Пример 4. За кое най-малко цяло положително число n е вярно неравенството $n < n^3/333$? Преди да започнем да пишем програмата, нека да пресметнем с калкулатор стойностите на $n^3/333$ за n от 1 до 10. Намираме приблизително:

n	1	2	3	4	5	6	7	8	9	10
$n^3/333$	0.003	0.24	0.81	0.19	0.38	0.65	1.03	1.54	2.19	3.003

Виждаме, че поне в рамките на таблицата n не е по-малко от съответната стойност на $n^3/333$. Но забелязваме, че $n^3/333$ като че ли расте все по-бързо и може би ще достигне стойността на n. Следващата програма увеличава в цикъл стойността на n всеки път с единица, започвайки от 1, и спира, когато за първи път намери стойност на n, такава, че $n < n^3/333$.

```
#include<iostream.h>
void main()
{int n=0;
 while(1)
 {n++;
  if(n<(n*n*n)/333.0) break;
 }
 cout << "n=" << n << "\n";
}
```

Отговорът, който програмата отпечатва, е $n = 19$.

В разгледания пример се търси само едно число и затова при намирането му е удобно да излезем от цикъла с `break`.

Пример 5. Дадени са целите числа a, b и c. Да се намерят всички цели числа x и y от интервала $[-9, 9]$, за които е изпълнено равенството $ax+by = c$.

Намирането и отпечатването на всички търсени двойки x и y може да се реализира с вложени цикли:

```
for(int x=-9; x<=9; x++)
 for(int y=-9; y<=9; y++)
  if(a*x+b*y==c) cout << x << " " << y << "\n";
```

Пример 6. Да се отпечатат всички тройки от цели положителни числа a, b и c, по-малки от 100, за които е изпълнено, че $a^2 + b^2 = c^2$. Такива тройки числа се наричат *питагорови тройки*.

Решение 1: Програма, която решава задачата чрез пълно изчерпващо търсене, е следната:

```
#include<iostream.h>
void main()
{
 for(int a=1;a<100;a++)
 for(int b=1;b<100;b++)
 for(int c=1;c<100;c++)
 if(a*a+b*b==c*c)
  cout << a << " " << b << " " << c << "\n";
}
```

Решение 2: Усъвършенстване (т.е. повишаване на бързодействието) в предишната програма може да се направи, като забележим, че не е нужно да проверяваме всички двойки `a` и `b`, а само тези, за които `a <= b`. Освен това, след като сме взели `a` и `b`, очевидно е, че има смисъл да търсим измежду тези стойности за `c`, които са по-големи от `b`. Така получаваме следната програма, която също би могла да търпи следващо усъвършенстване:

```
#include<iostream.h>
void main()
{
 for(int a=1;a<100;a++)
 for(int b=a;b<100;b++)
 for(int c=b+1;c<100;c++)
 if(a*a+b*b==c*c)
  cout << a << " " << b << " " << c << "\n";
}
```

Отделяне на цифрите на дадено цяло положително число. В главата за пресмятане по формула разгледахме частен случай на тази задача за двуцифрени и трицифрени числа. Сега, като разполагаме с възможностите на операторите за цикъл, може да решим задачата в по-общ случай. Нека стойността на даденото цяло положително число е записана в променливата `a`. Като си припомним как намирахме цифрата на единиците и след това как от даденото число преминавахме към число с една цифра по-малко, може да опишем словесно метода, който ще използваме, по следния начин:

Започваме със стойност в променливата `a`, равна на първоначалното число, и достигаме до нова стойност `a` по време на работата. От нея преминаваме към следваща стойност за `a`, като отделяме поредната цифра чрез операцията `a%10`, а стойността на `a` за следващата стъпка полагаме равна на резултата от

целочисленото деление с 10, т.е. `a=a/10`. Очевидно този процес ще продължи, докато `a>0`. Програмният фрагмент, който извежда цифрите на числото една по една *в обратен ред*, т.е. първо цифрата на единиците, след това цифрата на десетиците и т.н., е следният:

```
cin >> a;
do
{
 cout << a%10 << "\n";
 a=a/10;
}
while(a>0);
```

Съставяне на число по неговите цифри. Ако въведем от клавиатурата последователно една по една цифрите на числото, като започнем от старшата, може да образуваме самото число и да го запишем в целочислена променлива, като използваме т.нар. *схема на Хорнер* (Уилям Хорнер, английски математик от 19. век):

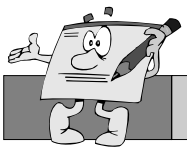
```
int a,b,n;
cout << "Въведете броя на цифрите ";
cin >> n;
cout << "Въведете цифрите от старшата към младшата,\n";
cout << "по една цифра на ред\n";
a=0;
for(int i=1; i<=n; i++)
{
 cin >> b;
 a = 10*a + b;
}
cout << "Числото е " << a << "\n";
```

Забележка: Старша цифра на многоцифрено число е тази, която съответства на най-големия десетичен разред, а младша – тази, която съответства на разреда на единиците.

Десетични дроби. Като използваме само стандартни целочислени променливи, можем да отпечатаме даден брой k от цифрите на десетичното представяне на дробта $1/n$, където n е цяло положително число:

```
int r=10;
for(int i=1; i<=k; i++)
{ int c = r/n;
  cout << c;
  r = r%n;
  r = 10*r;
}
```

Читателят може да провери, че всъщност е програмиран обичайният „ръчен“ метод за делене на числото 1 с числото n. Трябва да обърнем внимание на факта, че при по-големи стойности на n, стойността, която ще се получава в променливата `r`, може да стане по-голяма от максимално допустимата за целочислените променливи от типа `int` и програмата ще работи неправилно. *Въпроси, свързани с излизане извън границите на допустимите стойности за целочислените променливи, ще разгледаме по-нататък.*



УПРАЖНЕНИЯ

1. Променете програмния фрагмент от настоящата глава, с който е получен цикличният брояч, така че в него да не се използва условен оператор, а да се приложат два вложени цикъла.

2. Защо в програмата от Пример 4 операторът за деление е записан като `(n*n*n)/333.0`, а не например `n*n*n/333`?

3. Напишете програма, която отпечатва числата от таблицата в Пример 4.

4. Напишете програма, която отпечатва даден брой k от цифрите на десетичното представяне на дробта m/n, където m и n са цели положителни числа и $m < n$.

5. Какво ще отпечата следващият цикъл?

а)
```
for(int i=1;i<100;i++)
  {cout << i << " "; if (i==10) break;}
```

б)
```
for(int i=0;i<5;i++)
  {for (int j=0; j<100; j++)
    {cout << j << " ";
      if (j==5) break;}
  }
```

6. Напишете самостоятелно няколко програми, които използват `break`, за да излизат от цикли. Опитайте и с трите вида цикли.

7. Покажете начин за създаване на безкраен цикъл.

8. Въведете последователност от N цели числа. Намерете сумата от всички отрицателни числа измежду въведените.

9. Въведете последователност от N произволни числа. Определете колко пъти числата от тази последователност сменят знака си от положителен на отрицателен и обратно.

10. Въведете последователност от N цели числа. Намерете колко са нулите в нея.

11. Да се изведат в намаляващ ред всички четирицифрени числа, в които няма повтарящи се цифри.

12. Напишете програма, с която се играе играта „познай магическото число“. Правилата са следните: играчът има 10 опита, за да отгатне магическото число. Ако въведеното число е равно на избраното магическо число, нека програмата да изпише съобщението „Позна!“. В противен случай програмата да дава информация дали въведената стойност е по-голяма или по-малка от магическото число и след това да дава възможност на играча да направи нов опит. Този процес да продължи, докато играчът отгатне числото или изчерпи всичките 10 възможни опита.

КОНКУРСНИ ЗАДАЧИ

9.1. Автопарк (Есенен турнир, Шумен, 2003, зад. D3). Група разузнавачи трябвало да проучат състоянието на автопарка на противника. За целта те проникнали в базата и установили, че противниците притежават леки автомобили (с по 4 колела), товарни автомобили (с по 6 колела) и мотори с кош (с по 3 колела). Те внимателно преброили превозните средства и установили, че общият брой колела е N, след което ги описали. За съжаление при изтеглянето разузнавачите попаднали на засада и се наложило да прекосят река, при което част от документацията се намокрила и се загубила важна информация. Когато прегледали останалата част, установили, че се чете ясно само броят на колелата. За да се възстанови информацията, се наложило да се напише програма, която намира всички различни комбинации на трите вида превозни средства за определения брой колела. Тъй като програмистът на групата бил много болен (много силно настинал при преминаването на реката) се налага вие да им помогнете, като напишете програма, която прочита от клавиатурата цялото число N ($3 < N < 50$) и отпечатва на отделни редове на екрана различните комбинации на превозните средства, като ги подрежда така: на първо място броя на леките автомобили, на второ – броя на товарните автомобили и на трето – броя на моторите с кош.

Пример: Вход: 16
Изход:
1 0 4
1 1 2
1 2 0
4 0 0

Решение: Задачата се решава с изчерпващо търсене. Ако означим с `x`, `y` и `z` съответно броя на леките автомобили, на товарните автомобили и на моторите с кош, търсим всички възможности, за които `N=4*x+6*y+3*z`. Очевидно (но грубо) ограничение за изменението на стойностите на `x`, `y` и `z` е интервалът от 0 до `N`, където извършваме търсенето. Читателят може да използва и по-добри граници за изменение на отделните променливи (например от `0` до `N/4` за `x`; от `0` до `N/6` за `y`; от `0` до `N/3` за `z`).

```
#include<iostream.h>
void main()
{ int x,y,z,N;
  cin >> N;
  for(x=0;x<=N;x++)
  for(y=0;y<=N;y++)
  for(z=0;z<=N;z++)
  if(N==4*x+6*y+3*z)
   cout << x << " " << y << " " << z << "\n";
}
```

9.2. Числа (Есенен турнир, Шумен, 2001, зад. D3).

а) За числото 135 е изпълнено $135 = 1 + 3.3 + 5.5.5$; намерете всички трицифрени числа с това свойство, т.е. $\overline{abc} = a + b.b + c.c.c$.

б) За числото 1676 е изпълнено $1676 = 1 + 6.6 + 7.7.7 + 6.6.6.6$; намерете всички четирицифрени числа с това свойство, т.е. $\overline{abcd} = a+b.b+c.c.c+d.d.d.d$.

Решение: а) Извършваме пълна проверка с троен цикъл за всички възможни стойности на цифрите:

```
#include<iostream.h>
int a, b, c;
void main()
{
 for(a=1;a<=9;a++)
 for(b=0;b<=9;b++)
 for(c=0;c<=9;c++)
  if(100*a+10*b+c==a+b*b+c*c*c)
   cout << a << b << c << "\n";
}
```

9.3. Различни начини (Зимни празници, Плевен, 2002, зад. D1). Да се напише програма, която въвежда от клавиатурата естествено число S, $6 \leqq S \leqq 50$ и извежда на екрана броя на начините, по които числото S може да се представи като сбор на три различни естествени числа.

Примери:

Вход:	`6`	`10`
Резултат:	`1`	`4`

Пояснения: `6=1+2+3`, `10=1+2+7=1+3+6=1+4+5=2+3+5`

Решение: Извършва се изчерпващо търсене с троен цикъл за всички възможни стойности на `a`, `b` и `c`, за които `s==a+b+c`. За да се осигури, че `a`, `b` и `c` са различни, цикълът за `b` пробягва само стойности, по-големи от текущата стойност на `a`, а цикълът за `c` – стойности, по-големи от текущата стойност на `b`.

```
#include<iostream.h>
int a,b,c,s,n;
void main()
{cin >> s;
 n=0;
 for(a=1;a<s;a++)
 for(b=a+1;b<s;b++)
 for(c=b+1;c<s;c++)
  if(s==a+b+c) n++;
 cout << n << "\n";
}
```

9.4. Сладкиши (Национална олимпиада, областен кръг, 2003, зад. D3). Иван имал за харчене S стотинки и решил да си купи от любимите три сладкиша. Първият сладкиш струвал X стотинки, вторият – Y стотинки, а третият – Z стотинки. Иван решил така да избере по колко броя от всеки сладкиш да купи, че да му останат колкото може по-малко пари. За да стане това, май ще му е нужна вашата помощ.

Да се напише програма, която решава задачата. Програмата трябва да въвежда от един ред на клавиатурата целите положителни числа S, X, Y и Z, разделени с по един интервал. Всяко от числата не надхвърля 30000. Програмата не трябва да извежда никакви подсказки, преди да започне въвеждането на числата.

След като избере съответния брой сладкиши от всеки вид, програмата трябва да ги изведе на един ред на екрана, последвани от остатъка от парите R (за който Иван иска да е колкото може по-малък). Извежданите числа трябва да са разделени с по един интервал.

Пример: Вход: `72 23 17 31` Изход: `1 1 1 1`

Решение: В дадената по-долу програма променливите `a`, `b` и `c` служат за получаване на търсения брой сладкиши, а в променливата `r` се пресмята търсеният остатък от парите. Извършва се изчерпващо търсене с променливи `a` и `b` при подходящо подбрани граници – пробягват се целите стойности на

`a` от интервала `0 ,..., s/x` и за всяка избрана стойност на `a` другата променлива `b` пробягва целите стойности от `0` до `(s-a*x)/y`. За всяка двойка стойности на `a` и `b` се разглежда стойността `s-a*x-b*y`, като се има предвид, че се търси `c`, което да минимизира `r` в израза `r=s-a*x-b*y-c*z`. За целта се пресмята остатъкът от делението `r0=(s-a*x-b*y)%z` и се сравнява с текущата минимална стойност на `r`. При получаване на нулев остатък се излиза от цикъла с `break`, защото минимумът е намерен. Ако не се получи нулев остатък, програмата ще работи докато завършат всички повторения в двата вложени цикъла. В един от „най-лошите“ случаи на входни данни `(2999 2 2 2)` е необходимо време за работа около 3 секунди при процесор с тактова честота 1 GHz. Предполагаме, че данните са такива, че винаги съществува решение. Програмата извежда едно от възможните решения.

```
#include<iostream.h>
int s, r, x, y, z;
int a, b, c, a0, b0, c0, b2, r0;

void main()
{cin >> s >> x >> y >> z;
 r=s;
 for(a=0;a<=s/x;a++)
 { b2=(s-a*x)/y;
   for(b=0;b<=b2;b++)
   { r0=(s-a*x-b*y)%z;
     if(r>r0) {r=r0; a0=a; b0=b;}
     if(r==0)break;
   }
 }
 c0=(s-a0*x-b0*y-r)/z;
 cout << a0 << " " << b0 << " " << c0 << " "
      << r << "\n";
}
```

Забележка: Ако програмираме търсене с три вложени цикъла, например

```
for(a=0;a<=s/x;a++)
for(b=0;b<=s/y;b++)
for(c=0;c<=s/z;c++)
```

и търсим минималната неотрицателна стойност на `s-a*x-b*y-c*z`, програмата би работила твърде дълго време, за да намери решение на задачата при даденото в условието ѝ ограничение за максимална стойност 30000 за входните данни.

9.5. Линийки (Есенен турнир, Шумен, 2002, зад. D2). Дадени са две еднакви непрозрачни измервателни линийки с разграфени милиметри. По

дължината на всяка линийка са пробити по няколко малки дупки, като дупките са на еднакво разстояние от ръба на линийката. Дупките започват от нулевото деление и след това са разположени равномерно с разстояние между всеки две съседни дупки, равно на M милиметра за едната линийка и на N милиметра за другата. Считаме, че дупки са пробити и на самите крайни деления на линийките, ако там се пада да се пробие дупка, съгласно казаното по-горе. Дължината на всяка от линийките е L милиметра.

Напишете програма, която въвежда от клавиатурата стойностите на M, N и L, и извежда на екрана броя на дупките, които се виждат през двете линийки, след като те се поставят точно една над друга. Стойностите, които ще се въвеждат, са цели положителни числа, като стойностите на M и N са по-малки от 50, а L може да има най-голяма стойност, равна на 5000.

Пример: Въвеждаме: `1 2 4`. Програмата трябва да изведе: `3`.

Решение: В цикъл преброяваме тези числа от 0 до `L`, които са едновременно кратни на `M` и `N`:

```
#include<iostream.h>
int M, N, L;
void main()
{cin >> M >> N >> L;
 int b=0;
 for(int i=0;i<=L;i++)
  if((i%M==0)&&(i%N==0)) b++;
 cout << b << "\n";
}
```

9.6. Търговия (Национална олимпиада, областен кръг, 2003, зад. D1). Фирма „Нова Измама“ извършва продажби по следната схема: Клиент X, който желае да си купи лаптоп, чиято цена при другите фирми е a лева, внася само b лева (доста по-малко от a) и когато от вноските на следващите клиенти се съберат нужните пари, клиентът X получава стоката. Например, ако лаптопът струва 2000 лева, фирмата обявава размер на вноската 100 лева и ако всеки ден се намира по един нов клиент, който да я плаща, първият клиент X ще получи желания от него лаптоп на 20-тия ден (като е платил само 100 лева), вторият клиент Y, ще получи своя лаптоп на 39-тия ден от внасянето на парите (пак само за 100 лева) и т.н. Всичко изглежда като ново откритие в пазарната икономика, но след известно време все повече клиенти започват да чакат все по-дълго, даже прекалено дълго...

Да се направи програма, която пресмята на кой ден след като е внесъл парите си, K-тият клиент ще получи лаптопа си. Предполага се, че всеки ден фирмата намира точно по един нов клиент. Програмата трябва да въведе от клавиатурата стойностите на a, b (цели числа левове, по-малки от 10 000 и $a > b$) и $K (K < 100)$ и да изведе на екрана номера на търсения ден.

Пример 1:	*Пример 2:*
Вход	Вход
`2000 100 2`	`1000 600 1`
Изход	Изход
`39`	`2`

Решение: В дадената по-долу програма броячът `i` в цикъла `while` показва номера на деня от началото на дейността на фирмата. В променливата `s` се натрупва вземаната от клиентите сума. Когато натрупаната сума стане по-голяма или равна на цената на лаптопа, увеличава се броячът на удовлетворените клиенти `p` и от `s` се изважда цената на лаптопа (това става с оператора `if(s>=a){p++; s -= a;}`). Когато броят на удовлетворените клиенти стане равен на `k`, това означава, че е дошъл редът на `k`-тия клиент. Тогава цикълът прекратява работа си, като преди това се отпечатва числото `i-k+1`, равно на номера на търсения ден.

Особеност е използването на типа `long int`, вместо `int`. Налага се от факта, че при някои компилатори (например Borland (Turbo) C++ 3.1) числата от тип `int` са твърде къси за нуждите на програмата и е възможно с тях да се получат неверни резултати. Тази особеност на компютърната аритметика ще бъде разгледана по-нататък в книгата.

```
#include<iostream.h>
long int a,b,k;
void main()
{ cin >> a >> b >> k;
  long int i,s,p;
  i=0; s=0; p=0;
  while(p != k)
  { i++;
    s += b;
    if(s>=a){p++; s -= a;}
    if(p==k) cout << i-k+1 << "\n";
  }
}
```

9.7. Намери две (Пролетен турнир, Пловдив, 2002, зад. D3). Ако a и b са две зададени цели положителни числа между 1 и 175, не е никак трудно да намерите сбора им $s = a + b$ и произведението им $p = a * b$. Изобщо, ако са дадени две от числата a, b, s и p, трябва да може да намерите (може би не толкова лесно) другите две. Напишете програма, която чете от клавиатурата четири цели числа, разделени с по един интервал. Две от числата са нули, а другите две – различни от нула. За въведените четири числа програмата определя с какви числа трябва да се заменят двете нули, така че за получените четири ненулеви числа да е вярно следното: сборът на първите две да е равен

на третото, а произведението на първите две – на четвъртото. Програмата трябва да изведе на екрана четирите получени числа, разделени с по един интервал.

Пример 1:
Вход: `2 25 0 0`
Изход: `2 25 27 50`

Пример 2:
Вход: `20 0 27 0`
Изход: `20 7 27 140`

Пример 3:
Вход: `5 0 0 25`
Изход: `5 5 10 25`

Пример 4:
Вход: `0 0 15 56`
Изход: `8 7 15 56`

Решение: Съществуват 6 възможности за мястото на двете нулеви стойности:

```
0 0 s p
0 b 0 p
0 b s 0
a 0 0 p
a 0 s 0
a b 0 0
```

В програмата се обработват тези случаи и се има предвид, че числата `a`, `b`, `s` и `p` са цели, т.е. при всички случаи се търсят решения с цели стойности. При обработката на първия случай, т.е. когато трябва да се намерят числа `a` и `b`, за които е дадена тяхната сума и произведение, се прилага изчерпващо търсене с променливи за търсене `x` и `y`. В останалите случаи са програмирани формули, с които става пресмятането. Ако няма решение с цели числа, програмата не извежда нищо.

```
#include<iostream.h>
int a,b,s,p;
void main()
{ cin >> a >> b >> s >> p;
  if((a==0)&&(b==0))
   { for(int x=1;x<=175;x++)
     for(int y=1;y<=175;y++)
     if((x+y==s)&&(x*y==p)){a=x;b=y;}
   }
  else if((a==0)&&(s==0)){a=p/b; s=a+b;}
  else if((a==0)&&(p==0)){a=s-b; p=a*b;}
  else if((b==0)&&(s==0)){b=p/a; s=a+b;}
  else if((b==0)&&(p==0)){b=s-a; p=a*b;}
  else {s=a+b; p=a*b;}
```

```
  if((s==a+b)&&(p==a*b))
  cout << a << " "
       << b << " "
       << s << " "
       << p << "\n";
}
```

9.8. Равни стойности (Есенен турнир, Шумен, 2001, зад. D2). От клавиатурата се въвеждат последователно няколко цели положителни числа, след което се въвежда числото нула. Да се намери и изведе на екрана броят на числата в най-дългата поредица от последователни равни числа. Например, ако въведените числа са

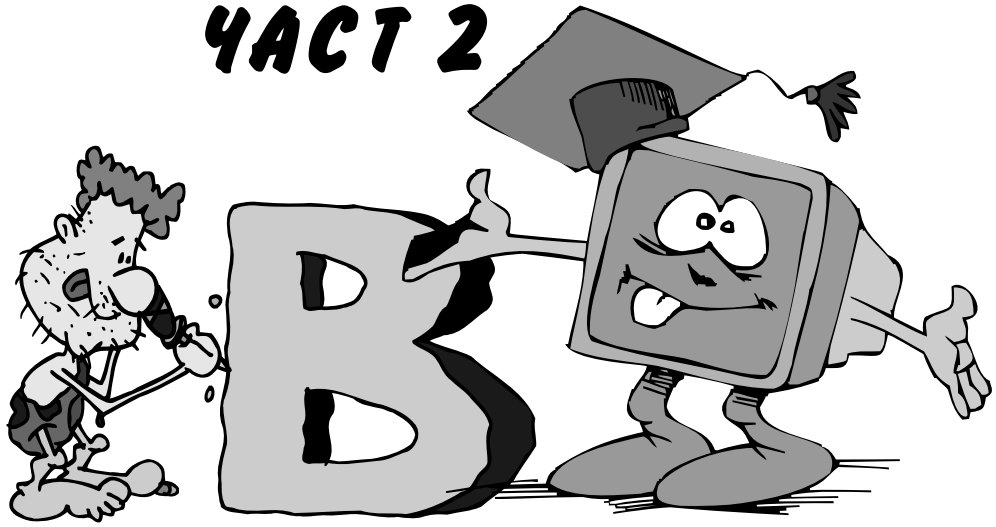
4 5 5 6 6 6 8 0

резултатът трябва да бъде 3, защото има поредица от 3 последователни равни числа (6,6,6) и това е най-дългата поредица от равни числа. Числата, които се въвеждат, са по-малки от 1000, но броят им може да бъде много голям.

Решение: Прочитаме поредното въведено число в променливата `a`, а в променливата `b` пазим предишното. Когато те са еднакви, увеличаваме с единица текущия брой `c` на равните числа в поредицата, завършваща с последното въведено; в противен случай, променяме при необходимост текущо намерения максимум `m` и присвояваме единица на `c`.

```
#include<iostream.h>
void main()
{int a,b,c=0,m=1;
 cin >> a; b=a;
 do
 {if(a==b)c++;
  else
   {if(m<c)m=c;
    c=1; b=a;
   }
  cin >> a;
 }
 while(a!=0);
 if(m<c)m=c;
 cout << m << "\n";
}
```

ЧАСТ 2



Глава 10. ФУНКЦИИ

Досега във всяка програма сме използвали функцията `main()`. В езика C/C++ е възможно да програмираме *собствени функции*, както и да използваме предварително създадени *стандартни функции* на средата за работа. В тази глава ще се запознаем с начални сведения за функциите, за да можем да ги прилагаме в някои прости случаи.

При съставянето на програма често се налага да се използва една и съща последователност от оператори повече от веднъж. В такъв случай е уместно да приложим предоставената в езика възможност за означаване с едно име на повтарящата се последователност. След това, когато е необходимо, на съответните места в програмата можем да напишем това име, а не цялата последователност.

Група от оператори, означена с едно име (тези имена се наричат *идентификатори*), дава пример за най-прост вид функция, която няма параметри. Използването в програмата на това име е известно като *извикване* на функцията или *обръщение* към нея.

Описание на функция. За да се въведе име (идентификатор) `f` на функцията, представляваща всъщност последователност от няколко оператора P_1, P_2, ..., P_K, или казано с други думи – съставен оператор, в програмата трябва да се включи *описание* (*дефиниция*) на функцията от следния вид:

```
void f()
{
  P1; P2; ... PK;
}
```

Например, за да пресметнем и отпечатаме стойностите на периметъра `L` и лицето `S` на правоъгълник със страни, чиято дължина е зададена чрез променливите `a` и `b`, може да използваме следния фрагмент в програма:

```
L=2*(a+b); cout << L << "\n";
S=a*b; cout << S << "\n";
```

Когато се налага да се пресметнат тези стойности, както за дадения правоъгълник, така и за друг, например за такъв, който има страни, равни на удвоените стойности на страните на дадения, може използваме фрагмента:

```
// пресмятане за първия правоъгълник
 L=2*(a+b); cout << L << "\n";
 S=a*b; cout << S << "\n";
// промяна в променливите:
 a=2*a; b=2*b;
// пресмятане за втория правоъгълник:
 L=2*(a+b); cout << L << "\n";
 S=a*b; cout << S << "\n";
```

Забелязваме, че първите четири оператора са еднакви с последните четири. За да избегнем повторното им записване, дефинираме функция с име `p`:

```
void p()
{
 L=2*(a+b); cout << L << "\n";
 S=a*b; cout << S << "\n";
}
```

Сега фрагмента за намиране периметрите и лицата на двата правоъгълника записваме по-кратко:

```
p();
a=2*a; b=2*b;
p();
```

Цялата програма има следния вид:

```
#include<iostream.h>

int a,b,L,S;

void p()
{
 L=2*(a+b); cout << L << "\n";
 S=a*b; cout << S << "\n";
}
```

```
void main()
{
 cin >> a >> b;
 p();
 a=2*a; b=2*b;
 p();
}
```

Обърнете внимание на използването на кръгли скоби след името на функцията.

Описанието на функцията `p()`е написано преди функцията `main()`, в която има обръщение към `p()`. Ние ще спазваме това правило и в други сходни случаи: ако в една функция `g()` има обръщение към друга функция `f()`, функцията `f()`трябва да бъде описана преди описанието на `g()`. Ще отбележим, че езикът C/C++ позволява и друг начин за подреждане, който тук не разглеждаме.

Обърнете внимание, че променливите `a`, `b`, `L` и `S`, които се използват във функцията `p()`, са обявени преди описанието на функцията.

Функция с параметри. За да използваме функцията `p()`, участващите в нея променливи `a` и `b` трябва да бъдат обявени преди описанието на функцията. Освен това, преди извикването на функцията тези променливи трябва да получат определена стойност.

В езика C/C++ е предвидена възможността за създаване на *функция с параметри*, които позволяват имената на променливите да се използват по-гъвкаво. Описанието на съответната функция `p(a,b)` с параметри `a` и `b` е следното:

```
void p(int a, int b)
{
 L=2*(a+b); cout << L << "\n";
 S=a*b; cout << S << "\n";
}
```

Виждаме, че в заглавния ред, между кръглите скоби, са записани параметрите, предхождани от името на типа им. За да извикаме тази функция, трябва след името ѝ, на местата на променливите `a` и `b` в скобите, да напишем имена на променливи или изрази, които искаме да използваме. Тогава, за да пресметнем и отпечатаме разглежданите величини за двата правоъгълника, достатъчно е да напишем:

```
p(a, b);
p(2*a, 2*b);
```

Параметрите, които присъстват в описанието на функцията, се наричат *формални параметри*, а когато се заместят с конкретни променливи или изрази при извикването на функцията, се наричат *фактически параметри*.

Понякога вместо параметър на функция се използва названието *аргумент* на функция.

Функции, които връщат стойност. Това са функции, извикването на които води до пресмятане на стойност от определен тип, която се свързва с името на функцията. Типът на пресметнатата стойност може да бъде някой от основните типове `int`, `double` или `char`. В много случаи употребата на идентификатора на такава функция е сходна с употребата на идентификатор на променлива в дясната част на оператор за присвояване или в израз. Например, ако стойността на една функция `f(x,y)` има тип `double` или `int`, тя може да се срещне в оператор за присвояване от вида `x=x+f(x,y)` или в оператор за цикъл от вида

```
while(f(x,y) < f(y,x)) x=f(x,x);
```

Ако стойността на една функция има тип `char`, функцията може да се използва по начин, сходен с използването на променлива от същия тип. Например, ако функцията `c(a)` връща стойност от тип `char`, тази функция може да участва в оператор за присвояване от вида `s=c(a)`, където `s` е от тип `char`, или например в условния оператор:

```
if(s == c(a)) s='a'; else s=c(b);
```

Описанието на една функция, която връща стойност, трябва да започне с типа на връщаната стойност `int`, `double` или `char` (вместо `void`, което се пише, ако функцията не връща стойност). Например

`int f(double a, int b){`*Тяло на функцията*`}`

Тялото на функцията, както споменахме, е съставен оператор, т.е. има вида {P_1; P_2; ... P_K}. Измежду операторите P_1, P_2, ..., P_K, които се разполагат в тялото на функцията, сега задължително трябва да присъства оператор от вида

`return` *стойност*`;`

Този оператор определя стойността, която се връща от функцията, щом функцията бива извикана на друго място в програмата. Освен това, достигането до този оператор предизвиква излизане от тялото на функцията (независимо от това дали след този `return` има следващи оператори в тялото) и предаване на управлението на оператора, намиращ се след мястото, където е извикана самата функция.

За пример да разгледаме описание на функция, която връща по-голямата от стойностите на двата си целочислени параметъра `a` и `b`:

```
int max(int a, int b)
{
 int c;
 if(a>b)c=a; else c=b;
 return c;
}
```

Променливата `c`, която е обявена в тялото на тази функция, е достъпна само в това тяло и се нарича *локална* променлива.

Еднакви имена. За да не настане объркване, не трябва да употребяваме едни и същи имена за означаване на локални променливи и за параметри в описанието на една функция.

Но твърде често програмистите употребяват едни и същи имена за означаване на глобални и локални променливи, което се позволява от езика. В този случай трябва да се знае, че при наличието на глобална променлива и локална променлива с едно и също име, в тялото на функция се има предвид локалната променлива. Аналогично е положението с едноименни глобална променлива и параметър на функция – в тялото на функцията се има предвид параметърът на функцията.

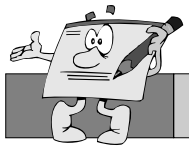
Функциите като средство за структуриране на програмата. С функции се работи удобно. Много от разгледаните досега програмни фрагменти е възможно да бъдат оформени като функции и след това да бъдат ползвани чрез извикването на тези функции.

Пример. Намиране на дата след определен брой дни. Ще разгледаме задачата за написване на програма, която по зададена дата, състояща се от ден `d`, месец `m` и година `g`, намира датата, която ще бъде след `k` на брой дни. Предполагаме, че трите стойности `d`, `m` и `g` се въвеждат от клавиатурата като цели числа и че те формират правилна (съществуваща) дата. В частния случай, когато `k` е равно на едно, задачата вече беше решена в Глава 5. Да запишем програмния фрагмент за намиране на следващата дата като описание (дефиниция) на функция с име `next()`, която ползва и променя глобалните променливи `d`, `m` и `g`.

```
void next()
{ int p;
  if(m==2)
   {if(((g%4==0)&&(g%100 !=0))||(g%400==0)) p=29;
    else p=28;
   }
  else if((m==4)||(m==6)||(m==9)||(m==11)) p=30;
  else p=31;

  if(d<p)d++;
  else
  {d=1;
   if(m<12)m++;
   else{m=1; g++;}
  }
}
```

Всяко обръщение към функцията `next()` води до промяна на глобалните променливи `d`, `m` и `g`, така че те ще вземат стойности, съответстващи на датата за следващия ден. Следователно, ако извикаме функцията `next()` последователно `k` на брой пъти в един цикъл, ще получим коя ще бъде датата след `k` дни. Следната програма въвежда една дата, съставена от ден `d`, месец `m` и година `g`, след това въвежда стойността на цялото положително число `k` и извежда коя дата ще бъде на `k`-тия ден след първоначално въведената дата. Предполагаме, че в програмата по-долу, след обявата на глобалните променливи, е написана дефиницията на функцията `next()`:

```
#include <iostream.h>
int d, m, g, k;
// място, където е написана
// дефиницията на функцията next()
void main()
{
  cin >> d >> m >> g;
  cin >> k;
  for(int i=1;i<=k;i++) next();
  cout << d << " " << m << " " << g << "\n";
}
```



УПРАЖНЕНИЯ

1. Съгласно правилата на езика C/C++ операторът `return` може да се ползва и при описание на функция, която не връща стойност. Тогава той се записва като `return;`. Обяснете смисъла на това използване.

2. Напишете отново програмата от примера, но така че да не използвате функции. Коя от двете програми е по-удобна за работа – тази от примера или тази, която написахте?

3. Разгледайте следните две описания на функции без параметри:

```
void P()
{x=x-y;
 y=x+y;}

void Q()
{y=x+y;
 x=x-y;}
```

Еднакви ли ще са „последствията“ при извикването на `P()` и на `Q()`?

4. Напишете програма, която отпечатва квадратите на две числа, въведени от клавиатурата. За целта съставете и използвайте функция `sqr()`, която да пресмята квадратите на числата.

5. Напишете функция `cub`, която пресмята третата степен на числото `a`. Използвайте тази функция, за да решите задачата от Пример 4 в Глава 9.

Упътване: Една възможна дефиниция на функцията `cub()` е следната:

```
int cub(int a) { return a*a*a;}
```

6. Напишете функция `swap()`, която разменя две глобални целочислени променливи `a` и `b`.

Упътване:

```
void swap()
{ int t;
  t=a; a=b; b=t;
}
```

7. Напишете функция, която връща по-малката от стойностите на двата си целочислени параметъра `a` и `b`.

8. Какво ще се отпечата при обръщение към функцията `f1()`?

```
void f1()
{ cout << "A";
  return;
  cout << "B";
}
```

9. Какво е неправилното в тази функция?

```
void fun()
{  int I;
   cin >> I;
   return I;
}
```

10. Напишете функция, която връща сумата на трите си целочислени параметъра.

11. Как функциите връщат стойност?

12. Може ли да се употребяват едни и същи имена за означаване на глобални и локални променливи?

13. Да се състави функция за пресмятане на:

а) факториела на числото n; б) степента a^b.

Упътване: Използвайте фрагментите от Глава 7.

КОНКУРСНА ЗАДАЧА

10.1. Дати (Зимни празници, Плевен, 2002, зад. D3). Дните на 2002 година са номерирани с числата от 1 до 365: `01.01.2002` има номер `1`; `02.01.2002` – номер `2`; и т.н., `31.12.2002` има номер `365`. Да се напише програма, която въвежда от клавиатурата пореден номер N и извежда на екрана съответната дата.

Примери:

Вход:	`40`	`89`
Резултат:	`09.02.2002`	`30.03.2002`

Решение: Възможно е да използваме направо програмата, дадена в примера от Глава 10. Но понеже става дума за една определена година – 2002, която не е високосна, може да опростим решението, което е дадено там, като запазим основната му идея. При извеждане на датата са взети мерки за отпечатване на водещи нули, ако денят или месецът се изразяват с едноцифрено число.

```
#include<iostream.h>
int d=1,m=1;
void next()
{int p;
 if(m==2) p=28;
 else if((m==4)||(m==6)||(m==9)||(m==11)) p=30;
 else p=31;
 if(d<p){d++;}else{d=1; m++;}
}
void main()
{int n;
 cin >> n;
 for(int i=1;i<n;i++) next();

 if(d<10) cout << 0;
 cout << d << ".";
 if(m<10) cout << 0;
 cout << m << ".2002\n";
}
```

Глава 11. ДЕЛИМОСТ НА ЧИСЛАТА

В тази глава ще разгледаме програмиране на задачи, отнасящи се към основите на теорията на числата.

Положителни делители на число. Да се отпечатат всички положителни делители на цялото положително число a. За да решим задачата, ще проверим последователно за всяко цяло число i от 1 до a дали числото i дели a без остатък, т.е. дали i е делител на a:

```
for(int i=1; i<=a; i++)
 if(a%i==0) cout << i << " ";
```

Просто число. Както знаем, едно цяло положително число $a > 1$ се нарича *просто*, когато то няма други положителни делители, освен единицата и себе си. В противен случай, числото се нарича *съставно*. Числото 1 не е нито просто, нито съставно.

Разлагане на прости множители. Всяко естествено (цяло положително) число, което е по-голямо от 1, може да се запише като произведение на *прости множители*. Например 1001 е равно на 7.11.13. Простите множители 7, 11 и 13 понякога се наричат *прости делители* на числото 1001. Същото число 1001 има и други делители, които не са прости, например 77 е делител на 1001, но 77 не е просто число.

В следващия програмен фрагмент последователно проверяваме за всяко цяло число p от 2 до N дали това число p дели N; ако го дели, продължаваме да проверяваме за същото p, но заменяме N с N/p; ако не го дели, продължаваме с $p + 1$, вместо p.

```
p=2;
while(p<=N)
{ while(N%p==0)
   {cout << p << " "; N /=p;}
  p++;
}
```

Да отбележим, че ще се отпечатат само простите делители, въпреки че числата `p`, с които проверяваме (и някои от тях се отпечатват), пробягват всички числа от 2 до `N`, а не само простите числа в този интервал. Обяснението идва от факта, че `p` пробягва в нарастващ ред целите числа от 2 до `N`. Тогава, в момента, когато за първи път `p` ще раздели `N` без остатък, `p` ще бъде просто число. Цялостна програма, реализираща горния фрагмент, е дадена като решение на Конкурсна задача 1 към настоящата глава.

Канонично разлагане на прости множители. Предишният програмен фрагмент отпечатва всеки прост множител на въведено число N толкова пъ-

ти, колкото пъти този прост множител се среща в разлагането на N. Например за $N = 3528$, ще се изведе

```
2 2 2 3 3 7 7
```

В някои случаи е по-удобно всеки повтарящ се множител да се отпечата само веднъж, но последван от броя на повторенията (или единица, ако този множител се среща точно веднъж). Броят на повторенията на всеки множител се нарича *кратност* на множителя. В следващата програма е реализиран с изменения горният фрагмент за разлагане на множители, като е въведен брояч `b` на повторенията. Преди да се влезе във вътрешния цикъл `while`, се прави проверка дали поредната стойност на `p` дели `N` и ако е така, броячът `b` се зарежда с 1. Резултатът се извежда като редица от двойки числа (множител и кратността му), заградени със скоби и разделени със запетайка вътре в скобите.

```
#include<iostream.h>
int N,p;
void main()
{ cin >> N;
  p=2;
  while(p<=N)
  { if(N%p==0)
    { cout << "(" << p << ",";
      N /=p;
      int b=1;
      while(N%p==0)
       {b++; N /=p;}
      cout << b << ") ";
    }
   p++;
  }
  cout << "\n";
}
```

При въведена стойност 3528 програмата ще отпечата

```
(2,3) (3,2) (7,2)
```

Това означава, че $3528 = 2.2.2.3.3.7.7 = 2^3 3^2 7^2$.

Проверка дали дадено число е просто. Ще разгледаме задачата за написване на програма, която въвежда цяло положително число `a>1` и определя дали то е просто.

Решение 1: В следващата програма променливата `b` е използвана като брояч на делителите на `a`. Проверява се кои цели числа между `2` и `a-1` делят `a`. Ако броят им е нула, числото `a` е просто.

```
#include<iostream.h>
void main()
{ int a;
  int b=0;
  cin >> a;
  for(int i=2;i<a;i++)
   if(a%i==0) b++;
  cout << "Числото " << a << " е ";
  if(b==0) cout << "просто";
  else cout << "съставно";
}
```

Решение 2: Задачата може да бъде решена чрез оператора за цикъл `while`. Така имаме възможност да обявим резултата при първото намиране на делител на числото, т.е. не е нужно да правим излишни проверки, след като вече сме установили, че числото `a` е съставно:

```
#include<iostream.h>
void main()
{ int a;
  cin >> a;
  int i=2;
  while(a%i !=0) i++;
  cout << "Числото " << a << " е ";
  if(i==a) cout << "просто";
  else cout << "съставно";
}
```

Ще отбележим, че цикълът `while` в тази програма няма да се изпълнява безкрайно, защото при увеличаването на стойността на променливата `i` всеки път с единица, в определен момент тя ще стане равна на стойността на `a` и тогава стойността на `a%i` ще бъде равна на нула.

Решение 3: Възможно е да решим задачата с използване на цикъла `for`, но със средство, което позволява да се излиза от цикъла при настъпване на определено събитие. За целта използваме оператора `break`:

```
#include<iostream.h>
void main()
{ int a;
  int b=0;
  cin >> a;
  for(int i=2;i<a;i++)
    if(a%i==0) {b=1; break;}
```

```
  cout << "Числото " << a << " е ";
  if(b==0) cout << "просто";
  else cout << "съставно";
}
```

В тази програма променливата `b` се използва като *флаг*. Тя може да приеме в програмата само две стойности: `0` или `1`. Според стойността ѝ след излизането от цикъла `for`, може да се определи как е станало това излизане. Стойността `b==1` показва, че излизането е станало чрез оператора `break`, което в тази програма означава, че за някоя по-малка от `a` стойност на `i` е било изпълнено, че `i` дели `a`, т.е. `a` не е просто число.

Функция за проверка дали дадено число е просто. Всяка от предишните програми за проверка дали въведеното число е просто, може да се видоизмени във функция, която приема като параметър числото, което ще се проверява. Функцията ще връща стойност 1 или 0 в зависимост от това дали променливата, поставена на мястото на параметъра, е просто, или не е просто число. По-долу е дадена една възможна дефиниция на такава функция с име `prime`.

```
int prime(int a)
{if(a==1) return 0;
 for(int i=2;i<a;i++)
  if(a%i==0) return 0;
 return 1;
}
```

Ако за някоя стойност на `i` в цикъла `for`, където `i` се променя от `2` до `a-1`, се окаже, че `i` дели `a`, веднага се излиза от тялото на функцията чрез `return 0;` – това означава, че числото `a` не е просто. В противен случай, след завършване на цикъла `for` се излиза от тялото на функцията чрез `return 1;` – числото `a` е просто.

Отпечатване на простите числа от зададен интервал. В цикъл с изменение на брояча `i` от `m` до `n` можем да отпечатаме простите числа между `m` и `n`, като всеки път проверяваме чрез функцията `prime(i)` дали стойността на брояча `i` е просто число:

```
for(int i=m; i<=n; i++)
 if(prime(i)) cout << i << " ";
```

Например за `m=1` и `n=50` ще се отпечатат първите прости числа, по-малки от 50:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

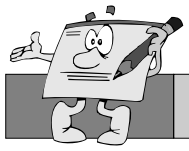
Най-голям общ делител. Следващата функция връща най-голямото цяло положително число, което едновременно дели двете цели положителни числа `a` и `b`.

```
int nod(int a, int b)
{
  int d,i=1;
  while((i<=a)&&(i<=b))
   { if((a%i==0)&&(b%i==0)) d=i;
     i++;
   }
   return d;
}
```

Взаимно прости числа. Две цели положителни числа a и b се наричат взаимно прости, ако нямат общ делител, по-голям от 1. Следващата функция `p(a,b)` връща 1, ако числата `a` и `b` са взаимно прости. В противен случай функцията връща `0`.

```
int p(int a, int b)
 {if(nod(a,b)==1) return 1; else return 0;}
```

Забележка: За решаване на задачите от тази глава съществуват значително по-бързи алгоритми, които тук не разглеждаме. Някои от тях могат да се намерят в посочените източници в края на книгата.



УПРАЖНЕНИЯ

1. Кои числа са прости и кои съставни?
2. По какво се различава разлагането на прости множители от каноничното разлагане на прости множители?
3. По какво се различават трите решения на задачата за определяне дали дадено число е просто?
4. Защо се налага проверката `if(a==1) return 0;` в началото на тялото на функцията `prime()`?
5. Напишете по друг, по-ефективен начин, тялото на функцията `nod(a,b)`, така че търсенето да се извършва „отгоре-надолу“, с цел да се излезе от тялото веднага щом се намери най-големият общ делител.
6. Отпечатайте минималното число, по-голямо от 200, което е кратно на 17.
7. Напишете програма, която отпечатва простите числа, намиращи се между 1 и 100, и след това променете програмата, така че тя да отпечатва простите числа между 1 и N, където стойността на N се въвежда от клавиатурата.

8. Напишете програма, която въвежда две цели положителни числа и отпечатва техния най-голям общ делител.

9. Напишете програма, която въвежда няколко двойки цели положителни числа и отпечатва за всяка от двойките съответния най-голям общ делител на числата в двойката.

10. Напишете програма, която въвежда две цели положителни числа `x` и `y` и отпечатва тяхното най-малко общо кратно.

Упътване: Чрез изчерпващо търсене открийте първото по големина число `z`, равно на или намиращо се след по-голямото от числата `x` и `y`, такова, че едновременно са изпълнени двете условия: `x` дели `z` и `y` дели `z`.

Забележка: Възможно е задачата да бъде решена по друг начин – ако знаете, че най-малкото общо кратно `nok(x,y)` е равно на резултата от делението на `x*y` с най-големия общ делител `nod(x,y)`.

11. Едно цяло положително число се нарича съвършено, ако е равно на сумата от своите делители (като в списъка на делителите тук не включваме самото число). Например 28 е съвършено, защото $28 = 1+2+4+7+14$. Напишете функция, която определя дали едно число е съвършено.

12. Напишете програма, която отпечатва първите няколко съвършени числа. Кое е следващото съвършено число след 28?

13. За дадено цяло положително число намерете най-малкото следващо след него число, което е просто.

14. За дадено цяло положително число намерете най-малкото следващо след него число, с което то е взаимно просто. Как се решава тази задача най-ефективно?

15. Намерете няколко числа, такива, че за всяко число `x` от тях, ако означим с `y` най-малкото следващо след `x` просто число, а със `z` – най-малкото следващо след `x` число, което е взаимно просто с `x`, да е изпълнено, че `y` не съвпада със `z`.

16. Програмирайте следващата функция, която реализира *алгоритъма на Евклид*:

```
int euclid (int x, int y)
{
 while(x != y)
 if(x>y)x=x-y; else y=y-x;
 return x;
}
```

Проверете за възможно по-голям брой двойки цели положителни числа `x` и `y` дали резултатът, получен с дадената в настоящата глава функция `nod(x,y)` съвпада с резултата, получен от функцията `euclid(x,y)`.

КОНКУРСНА ЗАДАЧА

11.1. Прости множители (Национална олимпиада, общински кръг, 2002, зад. D1). Да се направи програма, която въвежда от клавиатурата естествено число N и го разлага на прости множители ($N > 1$).

Примери:
Вход: $N = 6$
Резултат: $6 = 2.3$

Вход: $N = 1000$
Резултат: $1000 = 2.2.2.5.5.5$

Вход: $N = 1001$
Резултат: $1001 = 7.11.13$

Решение: Последователно проверяваме за всяко цяло число p от 2 до N дали това число p дели N; ако го дели, продължаваме да проверяваме за същото p, но за $N = N/p$, вместо за N; ако не го дели, продължаваме с $p+1$, вместо p. Приложената програма отпечатва простите множители на N, като ги отделя с по един празен интервал:

```
#include<iostream.h>
int N,p;
void main()
{cin >> N;
 p=2;
 while(p<=N)
    {while(N%p==0)
      {cout << p << " "; N /=p;}
     p++;
    }
 cout << "\n";
}
```

Забележка: С малки промени в програмата може да се направи така, че тя да отпечатва и точките между множителите, както е дадено в примерите след условието на задачата.

Глава 12. МАСИВИ

Когато трябва да използваме няколко *еднотипни променливи*, между които има определена смислова връзка, може да употребим за тях подходящо избрани имена, с което да подчертаем тази връзка. Например имената да са съставени от едно и също начално съчетание от букви, но да завършват с различни числа, например `a1`, `a2` и `a3`. Обаче неудобството на този вид означения е очевидно, особено при нарастване на броя им. Затова в езиците за програмиране се използват подходящи конструкции за съвкупности от такива еднотипни променливи. В езика C/C++ за целта е въведено понятието *масив*. Една променлива от тип масив, която съдържа едновременно например 20 различни целочислени стойности, може да бъде обявена по следния начин:

```
int a[20];
```

Обърнете внимание, че при това обявяване са използвани квадратни скоби, в които е записан броят на елементите на масива. След обявяването на името (идентификатора) `a`, обръщенията към отделните му елементи се извършват чрез използване на означенията `a[0]`, `a[1]`, `a[2]`, ... , `a[19]`. Така всъщност разполагаме с 20 отделни променливи от целочисления тип `int`, като първата е `a[0]`, а последната – `a[19]`. Това е доста сходно с използването на простите променливи `a0`, `a1`, ..., `a19`, но предимствата идват от възможността номерът на променливата да бъде заменен с друга променлива, наричана *индекс* и традиционно означавана с някоя от буквите `i`, `j`, `k` и т.н. Ако `x` е променлива от тип `int`, допустимо е да използваме присвояванията `x=a[i];` и `a[i]=x;` при зададена стойност на `i` в границите от 0 до 19 и е възможно да проверяваме верността на условието `x==a[i]` или други подобни условия, в които знакът за равенство е заменен с някой от знаците за неравенства.

Можем да използваме елементите на масива в оператор за цикъл. Като пример ще посочим, как се отпечатват всичките му елементи:

```
for(int i=0; i<20; i++) cout << a[i] << " ";
```

Типът на елементите на масива се нарича *базов тип* за масива. Базовият тип, освен целочислен (`int`), може да бъде и друг, например дробен (`double`). Тогава съответното обявяване ще изглежда така:

```
double a[20];
```

Броят на елементите в един глобален масив, освен с цяло неотрицателно число, може да се зададе с наименована *константа*. Пример за определяне на константа с име `N` и масив `a`:

```
const int N=100;
double a[N];
```

Броят на елементите на масива понякога се нарича *дължина* на масива. Важно правило, което трябва да спазваме при работа с масиви, е да НЕ допускаме обръщение към елемент в масив, чийто индекс има стойност, която е извън границите, в които този индекс е определен. Една от често срещаните грешки, появяваща се при проверката на програмите, е предизвикана от такова *излизане извън границите на индекса на масив*.

Задаване стойности на елементите на масив. Самото обявяване на масив не определя, изобщо казано, какви стойности ще получат неговите елементи. Затова в много случаи първото нещо, което трябва да направим с елементите, преди да извършваме други обработки, е да осигурим за тях подходящи начални стойности. Например в много случаи те се зареждат с нули:

```
for(int i=0; i<N; i++) a[i]=0;
```

Съгласно стандартите, реализирани в повечето компилатори на C/C++, всеки масив от числа, обявен като глобална променлива (т.е. обявен извън коя да е функция), се зарежда автоматично с нулеви стойности, без това да бъде явно програмирано. Например следната програма ще отпечата редица от пет нули:

```
#include <iostream.h>

int a[5];

void main()
{
  for(int i=0; i<5; i++) cout << a[i] << " ";
}
```

Стойностите на елементите в масива могат да се въвеждат и чрез клавиатурата. За целта понякога е желателно да отпечатваме и ориентиращо (подканващо) съобщение за ползващия програмата потребител, за да може той по-лесно да се ориентира кой е номерът на елемента, който трябва да получи стойността си:

```
for(i=0; i<N; i++)
 {
  cout << "Въведете стойност на елемент с номер "
       << i << " :";
  cin >> a[i]; cin.get();
 }
```

При въвеждане на данни в програма, с която се решава задача за олимпиада или друго състезание, обикновено не трябва да се отпечатват каквито и да

било съобщения. Така въвеждането на редица от `N` числа от входа на програмата и зареждането с тях на последователните елементи на масив става по-просто:

```
for(i=0; i<N; i++) cin >> a[i];
```

Когато броят на елементите в масива не е голям, възможно е да ги заредим с конкретни стойности на мястото, където масивът се обявява. Това става чрез дописване на знака за равенство „=“, следван от последователните стойности, разделени със запетайки и заградени с фигурни скоби така, както е показано в следния пример:

```
const int N=5;
int a[N]={0,1,2,3,4};
```

Този фрагмент, освен обявяване на масив `a` от 5 елемента, задава и следните стойности на тези елементи: `a[0]=0`, `a[1]=1`, `a[2]=2`, `a[3]=3` и `a[4]=4`.

Прилага се и зареждане със случайни стойности, което се използва при проверката на някои методи за решаване на задачи. Например при работа в среда на Borland C++ за целта може да бъде приложена стандартната функция `random()`. За да я използваме, в програмата трябва да се включи `#include<stdlib.h>`. Функцията `random()` моделира генериране на случайни числа, т.е. на редици от числа, сходни с тези, които можем да получим при хвърляне на монета (редица от нули и единици), зарче (редица, съдържаща цифри $1, 2, ..., 6$), при игра на рулетка и т.н. Извикването на `random(`p`)` връща случайно цяло число от интервала $0, 1, \ldots, p-1$. За да заредим масива `a` с числа, сходни с тези, които се получават при хвърляне на зарче, използваме цикъла:

```
for(i=0; i<N; i++) a[i]=1+random(6);
```

„Актуална“ дължина на масив. Броят на елементите в един глобален (т.е. достъпен навсякъде) масив се обявява като *постоянна величина*, която задаваме при съставянето на програмата. Но при много задачи се налага дължината на масива да се определи от стойност, получена по време на работа на програмата, например въведена в *променливата* `N` от клавиатурата. Тогава как да обявим масива, след като предварително не знаем дължината му? Обикновено постъпваме по следния начин:

Проверяваме в условието на задачата колко е най-голямата възможна стойност, която може да приеме въвежданата променлива `N`. Ако например най-голямата ѝ стойност е 100, тогава обявяваме масив с тази дължина, например чрез декларацията `int a[100];`. В програмата използваме `N` като дължина на масива, т.е. оставяме неупотребени елементите с индекси между стойността на `N` и 100. Всъщност в случая `N` става *актуална дължина* на масива, а 100 е неговата *максимална дължина*.

Сума от елементите на масив. Намирането на тази сума е една от най-простите задачи при обработване на масиви. Обявяваме допълнителна променлива `s`, която трябва да е от същия тип, от който е и базовият тип на масива. След нулиране на тази променлива, натрупваме сумата в нея чрез цикъл:

```
s=0;
for(i=0; i<N; i++) s += a[i];
cout << s;
```

Търсене в масив. Когато елементите на масива са заредени със стойности, може да се постави въпросът дали между тях съществува такава, която е равна на стойността, намираща се в някаква друга променлива `x`. Възможен подход за решаване на тази задача е да започнем последователно обхождане чрез цикъл на всички елементи в масива и да спрем, когато намерим това, което търсим.

```
for(int i=0; i<N; i++)
  if(a[i]==x){cout<<"found"; break;}
```

В този фрагмент при намиране на търсената стойност (когато се окаже, че условието за равенство `a[i]==x` е станало вярно), освен отпечатване на думата `found` (от английски – „намерено“), се извършва и *излизане от цикъла* чрез оператора `break`. Така, ако това, което търсим, се намира в началото на масива, тялото на цикъла `for` няма да се изпълнява толкова много пъти, колкото е общият брой на елементите в масива, а по-малко – само докато бъде открит елемент със стойност, равна на стойността на `x`.

Възможно е търсената стойност да не присъства измежду стойностите на елементите в масива. Тогава излизането от цикъла ще стане по обичайния начин – след като бъдат изчерпани всичките последователно нарастващи стойности на брояча `i`. Следващият фрагмент е усъвършенстван вариант на предишния, защото при него се извежда и номерът на елемента от масива, където е открита търсената стойност (всъщност, извежда се първият такъв номер, ако в масива има няколко елемента с една и съща стойност, равна на стойността на `x`). Когато търсената стойност не е открита, извежда се съответно съобщение. За целта се използва допълнителна променлива `j`, първоначално заредена с числото -1. Ако след излизане от цикъла тази стойност се запази, това показва, че търсената стойност не съществува в масива.

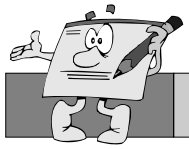
```
int j=-1;
for(int i=0; i<N; i++)
  if(a[i]==x){j=i; break;}
if(j==-1)
 cout << "търсената стойност не е намерена";
else
 cout << "търсената стойност е намерена и има номер "
      << j;
```

Намиране най-големия по стойност елемент в масива. Ще използваме променливата `m`, за да запишем търсената най-голяма стойност и ще ни е необходима променливата `j`, за да запишем индекса, при който тази най-голяма стойност е открита. За да обосновем начина на решаване на задачата, разглеждаме масива, като променяме мислено горната граница на индекса му. Най-напред, ако тази горна граница е 0, тогава е очевидно, че `m` е равно на `a[0]`, а `j` има стойност 0. След това, ако вече сме определили колко са `m` и `j` за частта от масива, съдържаща елементите от `a[0]` до `a[i-1]`, то добавянето към тази част на `a[i]` ще изисква промяна за `m` и `j`, но само ако `m<a[i]`. Така написваме следния фрагмент от програма за намиране най-голяма стойност в масив:

```
m=a[0];
j=1;
for(i=1; i<N; i++)
 if(m<a[i]){m=a[i]; j=i;}
cout << "m=" << m << " j=" << j;
```

Последният оператор извежда на екрана намерената най-голяма стойност `m` и номера `j` на елемента от масива, където тя се намира. Всъщност, ако тази най-голяма стойност се среща на повече от едно място в масива, изведената стойност на `j` ще покаже номера на първия подред елемент с тази стойност.

Намирането на най-малкия елемент в масива става по аналогичен начин. Единствената промяна, която трябва да направим в предишния фрагмент от програма, е да сменим знака „<“ с „>“ в условието на оператора `if`.



УПРАЖНЕНИЯ

1. Кога е удобно да използваме масиви?
2. Как се обявява променлива от тип масив?
3. За какво служи функцията `random()`?
4. Посочете грешките в следните фрагменти от програма, ако са в сила декларациите:

```
const int N=100;
double p[N], q[N];
int m[N], l[N];
int i;
```

1)
```
cout << "Въведете стойности на елементите на масива";
  cin >> p >> m;
```

2) `m=l; l=q; q=m;`

3) `l[i]= p[i]+m[i];`

4) `if (m == l) cout << "Yes";`

5) `p[100] = p[101] - p[99];`

6) `q = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};`

5. Дадено е цяло положително число $N < 100$. Напишете програма, която въвежда N от клавиатурата и след това въвежда още N числа, с които зарежда последователни елементи на масив.

6. Заредете елементите на масив със стойности, равни на:

а) номерата им;

б) квадратите на номерата им;

в) нула, ако номерът е просто число, и единица, в противен случай.

7. Заредете елементите на масив със стойности, така че първият елемент на масива да има стойност, равна на номера на последния му елемент; вторият елемент на масива да има стойност, равна на номера на предпоследния му елемент; ... ; последният елемент на масива да има стойност, равна на номера на първия му елемент.

8. Намерете броя на положителните елементи в масив.

9. Изведете стойностите на елементите на масив в обратен ред.

10. Даден е масив `a`. Да се „прехвърлят“ стойностите на елементите му в два масива – единият да съдържа отрицателните, а другият – положителните стойности измежду елементите на `a`.

11. Пресметнете средното аритметично от стойностите на елементите на масив.

12. Намерете и изведете сумата на тези елементи на масив, които са нечетни числа.

13. Определете има ли сред елементите на даден масив такива, които са степени на 2.

14. Отпечатайте елементите на даден масив от цели числа, които са кратни на 3.

15. Нека `int a[5];`. Проверете допустим ли е операторът `cout << a;`.

16. Намерете най-дългата подредица от стойности на последователни елементи в даден масив, така че всичките стойности в тази подредица да са положителни числа.

17. Заредете масив с последователните прости числа.

18. Напишете програма, която въвежда N числа ($0 < N < 100$) и отпечатва:

а) въведените числа, но в ред, обратен на въвеждането им;

б) за всяко i-то въведено число, сумата на числата от i-тото до последното.

КОНКУРСНИ ЗАДАЧИ

12.1. Хотел (Зимни празници, Варна, 2003, зад. D1). Галактическият пътешественик Йон Тихи веднъж посетил един хотел с много, много стаи, които били номерирани с естествените числа 1, 2, 3, ... Първоначално вратите на стаите били затворени. Йон Тихи преминал покрай всичките стаи, като отворил вратите им. След това преминал втори път, но само покрай стаите с номера 2, 4, 6, ... и т.н., като затварял вратите им. Преминал трети път, но само покрай стаите с номера 3, 6, 9, ... и т.н., като отварял вратите, които били затворени, и затварял тези врати, които били отворени. Йон Тихи продължил своето странно занимание, започвайки всеки път от врата с все по-голям номер, като при k-тото си преминаване се спирал само при всяка k-та стая и ако вратата ѝ била отворена, той я затварял, и обратно, ако била затворена – той я отварял.

Напишете програма, която въвежда от клавиатурата едно цяло положително число n и отпечатва на екрана броя на стаите с отворени врати от първата до стаята с номер n включително. Числото n, което се въвежда, няма да е по-голямо от 1000.

Пример: Вход: `3` Изход: `1`

Решение: Редицата от врати се моделира с елементите на масива `a[i]`. Ако `a[i]=0`, това означава, че `i`-тата врата е затворена, а при `a[i]=1` считаме, че `i`-тата врата е отворена. Понеже този масив е обявен в програмата като глобален, неговите елементи първоначално са нулирани. С външния цикъл `for` се програмира всяко едно от преминаванията на Йон Тихи от първата до последната `n`-та врата, а с вътрешния цикъл `for` се извършва отварянето и затварянето (`if(a[i]==0)a[i]=1;else a[i]=0;`). Накрая се преброяват отворените врати, като се използва броячът `b` и стойността му се отпечатва.

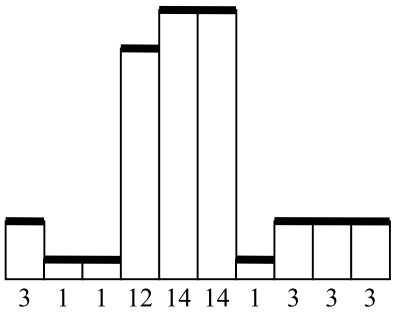
```
#include<iostream.h>
int a[1001];
int n;
void main()
{ cin >> n;
  int i,j;
  for(j=1;j<=n;j++)
   for(i=j;i<=n;i+=j)
    if(a[i]==0)a[i]=1;else a[i]=0;
  int b=0;
  for(i=1;i<=n;i++) if(a[i]==1) b++;
  cout << b << "\n";
}
```

Забележка 1. Може да се докаже, че в края на процеса отворените врати ще имат номера, които са точни квадрати: 1, 4, 9, 16 и т.н. Така задачата се решава с програма, която преброява точните квадрати в интервала от 1 до `n`.

Забележка 2. Условният оператор `if(a[i]==0)a[i]=1;else a[0]=0;` може да бъде заменен в програмата с оператора за присвояване `a[i]=1-a[i];`.

Това е възможно, защото стойностите на `a[i]` са 0 или 1.

12.2. Улица (Пролетен турнир, Ямбол, 2003, зад. D1). От едната страна на улица „Небостъргачева“ има N сгради с плоски покриви ($3 \leqq N \leqq 200$), всяка с височина цяло число – между 3 и 100 метра. Съседните, еднакво високи сгради, образуват чрез покривите си площадки. Например:



В този случай нарисуваните 10 сгради образуват 6 площадки (отбелязани с по-дебели линии).

Да се напише програма, която намира броя на площадките, образувани от сградите на улицата. Входните данни се четат от клавиатурата. На първия ред се въвежда числото N – броят на сградите. На следващите N реда следват височините на сградите. Трябва да се изведе едно единствено число – броят на построените площадки.

Решение: След прочитане на височините на отделните сгради в масива `a` се прави последователно преглеждане на елементите му. В променливата `h` се записва височината на предишната сграда при всяка стъпка в цикъла за преглеждане. Ако при някоя стъпка се намери сграда с различна височина от записаната в `h`, увеличава се с единица броячът `p` на площадките и в `h` се записва новата стойност. Накрая се отпечатва стойността на `h`.

```
#include<iostream.h>
int a[201];
int n;
void main()
{int h,i,j,p;
 cin >> n;
 for(i=1;i<=n;i++) cin >> a[i];
 h=a[1];
 p=1;
 for(i=1;i<=n;i++)
  if(a[i] != h){p++;h=a[i];}
 cout << p << "\n";
}
```

Забележка: Възможно е задачата да бъде решена без използване на масив. Сравнете с Конкурсна задача 9.8.

Глава 13. СОРТИРАНЕ

Изучаването на много въпроси, свързани с общото понятие за последователност от елементи, се улеснява с представянето им като елементи на масив `a[1], a[2], ..., a[N]`. В повечето случаи, за да изясним методите, без да ограничаваме общността на разглежданията, може да считаме, че базовият тип на масива е целочисленият тип `int`, а номерирането на индексите е в интервала от 1 до N. Да отбележим, че в тази глава, за разлика от предишната, разглеждането на номерата на елементите в масива ще започваме с единица. Понеже в езика C/C++ масивите започват с индекс нула, сега при обявяването на масив ще пишем например `int a[N+1]`, когато трябва да имаме елементи с номера от 1 до N, а елементът `a[0]` понякога се използва за съхраняване на някоя друга величина.

Сортиран масив. Един масив `a` се нарича *сортиран* (*подреден*), когато стойностите на елементите му са подредени спрямо номерата си в *растящ* или в *намаляващ* ред. Понятието сортиран масив е тясно свързано с понятията за растяща, намаляваща, строго растяща и строго намаляваща редица от числа.

Примери:
строго растяща редица: 1, 4, 6, 10, 17, 31, 44, 50
растяща редица, която не е строго растяща: 1, 4, 4, 6, 10, 17, 17, 31, 44, 50
строго намаляваща редица: 70, 65, 55, 43, 20, 18, 7, 5, 1
намаляваща редица, която не е строго намаляваща:
70, 65, 55, 55, 43, 20, 18, 7, 5, 1, 1

Прости алгоритми за сортиране. Методите (алгоритмите), с които получаваме сортиран масив, са едни от най-важните в информатиката. Съществуват различни идеи за съставянето им. За по-нататъшното изложение, както споменахме по-горе, ще приемем за определеност, че елементите на масива `a[1]`, `a[2]`, ..., `a[N]` са цели числа. Получаването на сортиран масив, в който тези числа са в растящ ред, означава, че трябва да се образува масив `b[1]`, `b[2]`, ..., `b[N]`, съдържащ същите числа, евентуално подредени по друг начин, но така, че

`b[1]` $\leqq$ `b[2]` $\leqq \ldots \leqq$ `b[N]`.

Ще отбележим, че употребените знаци „$\leqq$“ са математически означения за „по-малко или равно“ и не могат да се използват направо в програма на езика C/C++. Вместо „$\leqq$“, в програмата трябва да се напише двойката знаци „<=“.

Някои измежду числата `a[1]`, `a[2]`, ..., `a[N]` могат да бъдат равни помежду си. Естествено е тогава да се изисква всяко число да влиза в редицата `b[1]`,

`b[2]`, ..., `b[N]` толкова пъти, колкото пъти е влизало в `a[1]`, `a[2]`, ..., `a[N]`.

Много често в практиката дадените N числа `a[1]`, `a[2]`, ..., `a[N]` и финалните N числа `b[1]`, `b[2]`, ..., `b[N]` се разглеждат всъщност като елементи на един и същ масив, в който се извършва преподреждане (вътрешно разместване). Тези две съвкупности от N числа може да се разглеждат като начално и крайно състояние на елементите на един и същ масив.

Ще отбележим, че в началното и в крайното състояние на масива `a` ще има в съвкупност едни и същи числа, ако в процеса на сортирането извършваме само разменяне на стойности в двойки елементи.

Метод на мехурчето. Той е един от най-популярните алгоритми за сортиране. Възможна реализация за масива `a` за сортиране в растящ ред е следната:

```
int ok, i, t;
do
{ok=1;
 for(i=1;i<N;i++)
  if(a[i]>a[i+1])
   {t=a[i]; a[i]=a[i+1]; a[i+1]=t; ok=0;}
}
while(!ok);
```

При всяко изпълнение на тялото на цикъла `for` в горния фрагмент елементите на масива `a` се преглеждат по съседни двойки и ако при някоя такава двойка наредбата между елементите в двойката не е търсената, се прави размяна. След завършване работата на цикъла `for` „най-тежкият“ елемент „пада“ долу, а някои от „по-леките“ се вдигат една позиция нагоре, подобно на мехурчета във вода, от което идва и названието на този алгоритъм – *метод на мехурчето.*

Очевидно е, че само едно изпълнение на цикъла `for` при този алгоритъм не винаги ще е достатъчно за сортирането на целия масив, затова този цикъл се повтаря няколко пъти като вътрешен цикъл, вложен в цикъла `do...while`, докато променливата `ok` остане равна на 1, т.е., докато при някое поредно преминаване не се е налагало да се прави размяна на съседни стойности.

Лесно се съобразява, че броят на повторенията на самия оператор за цикъл `for` в най-лошия случай няма да надмине `N`. Това е така, защото при всяко едно от изпълненията на външния цикъл `do...while` поредният „най-тежък“ елемент застава „над“ поставените преди това от „дъното“ нагоре „тежки“ елементи. От казаното може да се направи заключение, че общият брой на операциите *сравняване* и *размяна* на двойки стойности при този алгоритъм има *порядъка* на N^2 (самият цикъл for може да се изпълни най-много до N пъти, като всеки път тялото на този цикъл се изпълнява $N-1$ пъти).

Пример за действието на метода на мехурчето. Ще илюстрираме метода върху следната редица от числа: 3, 6, 5, 7, 1, 4, 2. В представените по-долу схеми числата са изобразени за всяка от стъпките в отделна вертикална колона. Числата от двойката, които се сравняват, са отбелязани със звездичка, когато не се прави размяната и с кръстче, когато в следващата стъпка те са разменени. „Най-тежкият елемент“, който при всяко преглеждане „пада“, е отбелязан с получерен шрифт.

Първо преглеждане по двойки:

3*	3	3	3	3	3	3
6*	6+	5	5	5	5	5
5	5+	6*	6	6	6	6
7	**7**	**7***	**7+**	1	1	1
1	1	1	1+	**7+**	4	4
4	4	4	4	4+	**7+**	2
2	2	2	2	2	2+	**7**

Второ преглеждане по двойки:

3*	3	3	3	3	3	3
5*	5*	5	5	5	5	5
6	**6***	**6+**	1	1	1	1
1	1	1+	**6+**	4	4	4
4	4	4	4+	**6+**	2	2
2	2	2	2	2+	**6***	**6**
7	7	7	7	7	7*	7

Трето преглеждане по двойки:

3*	3	3	3	3	3	3
5*	**5+**	1	1	1	1	1
1	1+	**5+**	4	4	4	4
4	4	4+	**5+**	2	2	2
2	2	2	2+	**5***	**5**	**5**
6	6	6	6	6*	6*	6
7	7	7	7	7	7*	7

Четвърто преглеждане по двойки:

3+	1	1	1	1	1	1
1+	3*	3	3	3	3	3
4	**4***	**4+**	2	2	2	2
2	2	2+	**4***	**4**	**4**	**4**
5	5	5	5*	5*	5	5
6	6	6	6	6*	6*	6
7	7	7	7	7	7*	7

Пето преглеждане по двойки:

1*	1	1	1	1	1	1
3*	**3+**	2	2	2	2	2
2	2+	**3***	**3**	**3**	**3**	**3**
4	4	4*	4*	4	4	4
5	5	5	5*	5*	5	5
6	6	6	6	6*	6*	6
7	7	7	7	7	7*	7

При следващо преглеждане няма да настъпи нито едно разместване, защото редицата вече е сортирана в растящ ред. С това работа на алгоритъма приключва.

Сортиране с избор. Ще разгледаме друг алгоритъм за сортиране, построен върху идеята, известна като *сортиране с избор*. За определеност отново предполагаме, че елементите на масива трябва да се подредят в растящ ред.

Алгоритъмът работи на стъпки, като при всяка от тях се премества по един елемент от *ненаредената част* на масива в *наредената му част*. Границата между двете части се определя от стойността на променливата `j`, която се променя в цикъл от `1` до `N-1`. Елементите с номера от `1` до `j-1` образуват наредената част, а останалите – ненаредената. Първоначално ненаредената част от масива обхваща целия масив, а наредената част е празна.

Преместването на един елемент от втората (ненаредена) част в първата става по следния начин: Най-напред се открива кой елемент да бъде преместен. За такъв се избира текущият най-малък елемент в ненаредената част. След това той се разменя с `j`-тия елемент. Така на всяка от стъпките броят на елементите в наредената част се увеличава с единица, а броят на елементите в ненаредената част намалява с единица.

Схемата на програмния фрагмент за сортиране изглежда така:

```
for(j=1; j<N; j++)
 {
 // намиране на най-малката стойност m и на нейния
 // индекс p, измежду елементите a[j], a[j+1], ..., a[N]
 //
 // размяна на елементите с индекси j и  p
 }
```

Цялостният вид е следният:

```
int i, j, m, p;
for(j=1; j<N; j++)
{m=a[j];
 p=j;
```

```
   for(i=j+1;i<=N;i++)
   if(m>a[i])
    {m=a[i]; p=i;}
 a[p]=a[j];
 a[j]=m;
}
```

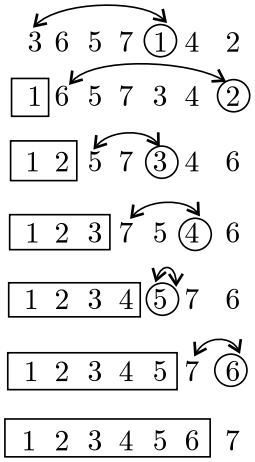
Лесно се съобразява, че сложността (броят на извършваните елементарни операции) при този алгоритъм, както и при предишния „алгоритъм на мехурчето“, има порядъка на N^2. Причина за това е наличието на два цикъла, вложени един в друг.

Пример за действието на метода за сортиране с избор.

На схемата е показано действието на алгоритъма при сортиране на масив от числата:

3, 6, 5, 7, 1, 4, 2

Всеки ред (отгоре-надолу) в схемата илюстрира поредната стъпка. Числата, заградени в правоъгълник, съставят наредената част от масива. На всяка от стъпките се избира най-малкото число от ненаредената част – то е заградено с кръгче. При първата стъпка това най-малко число е 1 и то се разменя с 3. При втората стъпка избраното число е 2 и то се разменя с 6 и т.н.



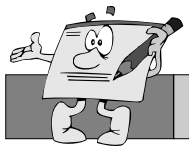
Сливане на два сортирани масива. Дадени са два масива `x` и `y`, като елементите на първия масив са номерирани от 1 до M, а на втория – от 1 до N. Дадено е, че самите елементи са подредени в растящ ред: `x[1]` $\leqq$ `x[2]` $\leqq$ ... $\leqq$ `x[M]` и `y[1]` $\leqq$ `y[2]` $\leqq$ ... $\leqq$ `y[N]`. Често срещана задача е „съединяването“ им в нов масив `z`, който да бъде сортиран `z[1]` $\leqq$ `z[2]` $\leqq$ ... $\leqq$ `z[M+N]`.

Естествено изискване е всеки елемент на `x` и `y` да участва в `z` и то толкова пъти, колкото е сумата от броя на срещанията му в `x` и в `y`. С други думи, масивът `z` трябва да се състои от същите елементи, както `x` и `y`, заедно с кратността им.

Целта ни е да решим задачата с ефективен алгоритъм – по-бърз, отколкото би се получил, ако първо образуваме масива `z`, като допишем елементите на `y` след тези на `x` и след това приложим някой алгоритъм за сортиране върху така получения „слепен“ масив `z`. Предлагаме следната реализация за бързо сливане със запазване на сортировката:

```
k=0; m=1; n=1;
while((m <= M)||(n <= N))
{k++;
 if(n>N){z[k]=x[m]; m++;}
 else if(m>M){z[k]=y[n]; n++;}
 else if(x[m]<=y[n]){z[k]=x[m]; m++;}
 else {z[k]=y[n]; n++;}
}
```

В този програмен фрагмент променливата `k` указва кой пореден елемент на новообразуващия се масив `z` ще се запълва със стойност. Променливите `m` и `n` във всеки момент сочат съответно към елемент от масива `x` и `y`, който ще бъде взет, за да се постави в текущо запълваната позиция на `z`. Когато не е достигнат нито краят на `x`, нито краят на `y`, се взема по-малкият елемент от двата елемента: `x[m]` и `y[n]`. Когато е достигнат единият от тези краища, тогава новият масив `z` се запълва до края си само с елементи, взети от другия от масивите `x` или `y`.



УПРАЖНЕНИЯ

1. Каква е разликата между строго намаляваща и намаляваща редица?
2. В какво се състои методът на мехурчето? Илюстрирайте действието му с друг пример по ваш избор, различен от дадения в текста на настоящата глава.
3. Обяснете как действа методът за сортиране с избор.
4. Какво ще се получи при прилагане на описания в настоящата глава алгоритъм за сливане, ако масивите `x` и `y` не са предварително сортирани?
5. Направете илюстрация за работата на метода за сливане на два сортирани масива.
6. Ако разгледате илюстрацията за работата на метода на мехурчето, ще забележите, че с всяка следваща стъпка на външния цикъл, все по-голям става отрезът от елементи в края на редицата, които не се разместват. Променете дадената програма за сортиране по метода на мехурчето, така че да използвате това наблюдение. С това ще се увеличи бързодействието на алгоритъма.
7. Напишете програма, която въвежда няколко числа от клавиатурата и ги отпечатва в растящ ред.
8. Дадена е редица, съдържаща не повече от 15 знака. Въведете я с програма и отпечатайте отначало всички знаци, които изобразяват цифри, след това всички знаци, които изобразяват малки латински букви, и накрая всички останали знаци, запазвайки реда им в редицата.

9. Намерете началото и дължината на най-дългата редица от равни числа в сортиран масив.

10. Напишете програма, която въвежда елементите на масив и изобразява на екрана илюстрация за действието на метода на мехурчето, приложен за въведения масив.

11. Напишете програма, която слива два произволни масива в нов масив (той, разбира се, не е задължително сортиран).

12. Реализирайте двете описани в настоящата глава програми за сортиране, като добавите броячи, с които да отпечатате броя на извършваните действия за присвояване на стойност и броя на извършените сравнения на стойности по време на работата на тези програми за сортиране на въведен масив с дължина N.

13. Напишете програма, която въвежда няколко трицифрени числа от клавиатурата и ги извежда в растящ ред според сумите от техните цифри.

Глава 14. ФУНКЦИИ И МАСИВИ

Масивът като параметър на функция. Масивите също могат да бъдат параметри на функциите. За да укажем това при дефиницията на функция, трябва при записването на масива като параметър след името му да добавим отваряща и затваряща квадратна скоба. Например във следващата програма функцията `f(a,n)` връща стойността на n-тия елемент от масива `a`. Забележете, че в дефиницията на функцията `f` масивът е отбелязан с буквата `b`, а в програмата е означен с `a`. При извикване на функцията се използва името `a`, което е написано в кръглите скоби след `f`, без самото то да е последвано от квадратни скоби:

```
#include<iostream.h>
int a[100];

int f(int b[], int n)
{return b[n];}

void main()
{a[2]=3; cout << f(a,2);}
```

Функция за проверка дали в масив има стойност нула. Следващата функция `isZero(a,n)` връща стойност 1, ако в масива, зададен като неин параметър, има поне един елемент със стойност нула измежду елементите `a[0], a[1], ..., a[n]`. В противен случай, функцията връща 0.

```
int isZero(int a[], int n)
{ for(int i=0; i<=n; i++)
    if(a[i]==0) return 1;
  return 0;
}
```

Функцията последователно проверява с помощта на цикъл елементите на масива `a` с номера от 0 до n. При първото откриване на нулева стойност веднага се напуска цикълът и едновременно с това и тялото на функцията, като се връща стойност 1 чрез оператора `return 1;`. Ако не е била намерена нулева стойност, тялото на функцията се напуска през другия оператор `return 0;`.

Функция за проверка дали даден масив е сортиран. Функцията `isSorted(a,m,n)` връща 1, когато стойностите на елементите, съставляващи следната част (*отрез*) от масива: `a[m]`, `a[m+1]`, ..., `a[n]`, са в растящ ред.

В противен случай функцията връща 0. Предполагаме, че $0 \leqq m < n$ и n не надминава максималната стойност на индекса на масива `a`.

```
int isSorted(int a[], int m, int n)
{ for(int i=m; i<n; i++)
    if(a[i]>a[i+1]) return 0;
  return 1;
}
```

Функция за проверка дали даден масив е строго сортиран. По сходен начин както функцията `isSorted(a,m,n)` може да съставим функция `isSorted0(a,m,n)`, която да връща 1 или 0, в зависимост от това дали стойностите на елементите, съставляващи отреза: `a[m]`, `a[m+1]`, ..., `a[n]`, са сортирани в строго растящ ред, или не са.

За да напишем тази функция, трябва само да заменим знака `">"` в оператора `if` със знака `">="`:

```
int isSorted0(int a[], int m, int n)
{ for(int i=m; i<n; i++)
    if(a[i]>=a[i+1]) return 0;
  return 1;
}
```

Функция за проверка дали в даден масив има елементи с равни стойности. Следващата функция `isEqVal(a,m,n)` връща 1, когато измежду елементите от отреза `a[m]`, `a[m+1]`, ..., `a[n]` има поне една двойка с равни стойности. В противен случай функцията връща 0. Предполагаме, че $0 \leqq m < n$ и n не надминава максималната стойност на индекса на масива `a`.

```
int isEqVal(int a[], int m, int n)
{
  for(int i=m; i<n; i++)
  for(int j=i+1; j<=n; j++)
   if(a[i]==a[j]) return 1;
  return 0;
}
```

Функция, която намира най-голямата разлика на два елемента в масив. Следващата функция `maxDif(a,m,n)` връща най-голямата разлика по абсолютна стойност между двойки стойности, взети измежду стойностите на елементите `a[m]`, `a[m+1]`, ..., `a[n]`. Предполагаме, че $0 \leqq m < n$ и n не надминава максималната стойност на индекса на масива `a`.

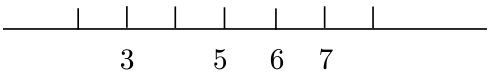
```
int maxDif(int a[], int m, int n)
{
  int max=0;
  for(int i=m; i<n; i++)
  for(int j=i+1; j<=n; j++)
   if(max < abs(a[i]-a[j]))
    max = abs(a[i]-a[j]);
  return max;
}
```

В определението на функцията `maxDif()` е използвана стандартната функция `abs()`, която връща абсолютната стойност на целочисления си аргумент (например `abs(-5)` връща `5`, но `abs(5)` е `5`). При някои среди за работа, за да се ползва тази функция, се изисква да се напише в програмата включването `#include<math.h>` или `#include<stdlib.h>`.

Пример за извикване на `maxDif` дава следната програма, където на мястото на коментара `//maxDif` трябва да се напише горното определение на функцията, за да бъде компилирането успешно. Масивът е инициализиран с тестови данни:

```
#include<iostream.h>
#include<math.h>
int a[10]={0,1,2,3,4,6,7,3,5,9};
//maxDif
void main()
 {cout << maxDif(a,5,8);}
```

При изпълнението на тази програма се отпечатва числото 4. Това е най-голямото разстояние между двойки целочислени точки от съвкупността {6,7,3,5}.



Търсене. В Глава 12 разгледахме задачата за намирането на индекс `i`, за който измежду елементите в масива `a[0]`, `a[1]`, ..., `a[N-1]`, елементът `a[i]` е равен на предварително зададена стойност `v`. Да припомним, че това става с *последователно сравняване*. Ясно е, че броят на проверките в най-лошия случай е толкова, колкото е броят N на елементите, измежду които търсим. Ще посочим още един начин за оформяне на търсенето, като съставим дефиниция на функция, в тялото на която ще използваме оператора `do...while`.

Функцията `search(a,n,v)` връща стойността на индекса, за който за първи път в последователността `a[0]`, `a[1]`, ..., `a[n-1]` е намерена стойността на `v`. Ако такава стойност не е намерена, функцията връща една условно приета в случая стойност, равна на минус едно `(-1)`.

```
int search(int a[], int n, int v)
{ int i=-1;
  do{i++;}while((a[i] != v) && (i < n-1));
  if(a[i]==v) return i;
  return -1;
}
```

От цикъла `do ... while` може да се излезе поради две причини: или е намерено това, което се търси `(a[i] == v)`, или е достигнат краят на масива `(i==n-1)`. Проверката с оператора `if(a[i]==v)...` ни осигурява правилен резултат и в двата случая.

Търсене в сортиран масив. Ако масивът `a` е *сортиран* (т.е. стойностите на елементите му са подредени в растящ или в намаляващ ред), възможно е процесът на търсене да се ускори значително. За целта се прилага методът на *делене наполовина*, известен още с названието *двоично* (*бинарно*) *търсене*. В следващия фрагмент, който реализира този подход, интервалът в който търсим, е заключен между стойностите на индексите `i` и `j`. На всяка стъпка този интервал се дели почти наполовина (за „среда“ се взема целочисленото частно `(i+j)/2`) и процесът продължава в тази от двете „половини“, където се очаква да се намери стойността на `v`. За да изберем в коя от двете „половини“ да продължим, се ориентираме от верността на неравенството `v>a[k]`. Предполагаме, че масивът е сортиран в растящ ред и търсим стойността на `v` измежду елементите `a[1], a[2], ... ,a[N]`.

```
i=1; j=N;
do{
 k=(i+j)/2;
 if(v>a[k]) i=k+1; else j=k-1;
}while((a[k] != v) && (i <= j));
if(a[k]==v) cout << "намерен при индекс k=" << k;
else cout << "не е намерен";
```

Ако след излизане от цикъла `do...while` се окаже, че `a[k]` не е равно на `v`, тогава със сигурност може да твърдим, че не съществува елемент в масива, равен на `v`. Максималният брой изпълнения на тялото на цикъла не надминава броя на последователните разделяния на по две равни части на интервала от индекси 1, ..., N. Оценка за броя повторения в цикъла дава най-малкият степенен показател p, за който 2^p надминава N. Например за $N = 1000$, броят на повторенията в разглеждания цикъл не надминава 10, защото $2^{10} = 1024$. От казаното се вижда, че двоичното търсене има значително предимство в бързодействие пред последователното търсене.

Пример. Ще илюстрираме действието на алгоритъма с пример за търсене на стойността $v = 7$ при масив `a`, съдържащ следните 11 стойности, дадени в таблицата:

номера	1	2	3	4	5	6	7	8	9	10	11
стойности	**1**	**3**	**4**	**7**	**9**	**11**	**12**	**13**	**17**	**20**	**22**

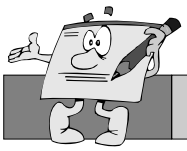
Стъпка 1: $i = 1$; $j = 11$; пресмятаме номера на „средата“ k; получава се $k = 6$; сравняваме v и `a[6]`. Понеже е в сила неравенство $v <$`a[6]`, продължаваме с лявата „половина“. За целта полагаме $j = 5$; разглеждаме отреза от масива между $i = 1$ и $j = 5$:

номера	1	2	3	4	5						
стойности	**1**	**3**	**4**	**7**	**9**						

Стъпка 2: $i = 1$; $j = 5$; пресмятаме номера на „средата“ k; получава се $k = 3$; сравняваме v и `a[3]`. Понеже е в сила неравенство $v >$ `a[3]`, продължаваме с дясната „половина“. За целта полагаме $i = 4$; разглеждаме отреза от масива между $i = 4$ и $j = 5$:

номера				4	5						
стойности				**7**	**9**						

Стъпка 3: $i = 4$; $j = 5$; пресмятаме номера на „средата“ k; получава се $k = 4$; сравняваме v и `a[4]`. Понеже е в сила равенство $v =$ `a[4]`, завършваме. Търсеният елемент е намерен и има номер $k = 4$.



УПРАЖНЕНИЯ

1. В какви случаи се поставят квадратни скоби след името на масив, когато това име е написано като параметър при извикване на функция?
2. Каква е разликата при извикванията `f(a)` и `g(a[n])`, където `a` е дефиниран като масив, а `f` и `g` са функции?
3. Защо в заглавния ред на дефиницията `int isZero(int a[], int n)` от настоящата глава не е указано нищо в квадратните скоби?
4. Формулирайте задача за търсене на елемент в дадена редица от числа.
5. Дайте пример, с който да покажете защо задачите за търсене и сортиране са важни задачи в информатиката.
6. Посочете по два метода за сортиране на масив и за търсене в масив.
7. Определете има ли в редица от цели числа два последователни нулеви елемента. Напишете функция, с която да проверявате това.
8. Да се напишат функции, които въвеждат и извеждат елементите на едномерен масив от цели числа.
9. Напишете програма, с която да проверявате дали въведен масив има елемент със стойност 0 и дали масивът е сортиран в определен отрез.
10. Напишете програма, с която да проверявате дали въведен масив има елементи с равни стойности.

11. Дадени са две сортирани в растящ ред редици `a[1]`, `a[2]`, ..., `a[N]` и `b[1]`, `b[2]`, ..., `b[N]`. Намерете броя на равенствата от вида `a[i] = b[j]` за всевъзможните стойности на `i=1,2,...,N` и на `j=1,2,...,N`. Може ли да направим по-бърз алгоритъм, ако „наистина“ използваме, че дадените редици са сортирани?

12. Напишете функция, която определя дали в един масив има елементи, които са с различни стойности.

13. Напишете функция за сортиране на масив по метода за сортиране с избор.

14. Напишете функция `max(a,m,n)`, която връща най-голямата стойност на елемент от масив `a`, избрана измежду елементите му с номера от `m` до `n`.

15. Напишете функция за търсене в отрез от масив.

16. Напишете функция за двоично търсене в масив.

17. Кога точката върху числовата ос, чиято координата е определена с целочисленото частно `(i+j)/2`, ще бъде истинска среда на съвкупността от целочислените точки върху числовата ос, заключени между точките с номера `i` и `j`?

18. Напишете функция, която намира най-големия общ делител на стойностите на всички елементи в един масив.

19. Въвеждат се N двойки числа (a_1, b_1), (a_2, b_2), ..., (a_N, b_N). Напишете програма, която да извежда двойките в растящ ред по първите числа в двойките, а когато първите числа в две двойки са равни, тези две двойки да се подредят в растящ ред по вторите си числа.

Упътване: Видоизменете по следния начин фрагмента на дадената в Глава 13 програма за сортирате по метода на мехурчето:

```
do
{ ok=1;
  for(i=1;i<n;i++)
  if((a[i]>a[i+1])||
     ((a[i]==a[i+1])&&(b[i]>b[i+1])))
     { int w;
       w=a[i]; a[i]=a[i+1]; a[i+1]=w;
       w=b[i]; b[i]=b[i+1]; b[i+1]=w;
       ok=0;
     }
}while(!ok);
```

Напишете и използвайте функция, която да връща стойността на съставното логическо условие, употребено в оператора `if` в горния фрагмент. Напишете и използвайте друга функция, която да извърши двойната размяна на съответните стойности на елементите от масивите `a` и `b` в дадения фрагмент.

Глава 15. МНОГОЦИФРЕНИ БРОЯЧИ

Моделирането на брояча на електромер или на километраж на кола, показващ изминатия път, може да бъде реализирано чрез масив. Съпоставяме на всяка клетка от брояча по един елемент от масива. Да приемем, че най-бързо изменящата се клетка (елемент на масива) е тази с номер 1. След като стойностите в тази клетка вече са пробягали числата от 0 до 9, трябва да увеличим с единица стойността на следващата клетка (т.е. тази с номер 2), а стойността на клетка 1 трябва да стане отново нула.

Това разсъждение може да продължим и в случая, когато стойността в клетка 2 стане равна на максималната, т.е. 9. Тогава следващата ѝ стойност трябва да е 0, но стойността на клетка номер 3 трябва да се увеличи с единица.

За да напишем програма, която моделира описания многоразреден брояч в общия случай, нека с N да означим броя на разглежданите разреди (клетки). Както е известно, при километража на кола има разреди, които съответстват на единиците, на десетиците, на стотиците и т.н. километри. Най-често стойността на N, т.е броят на клетките, е равен на 6. Нека да използваме и още един параметър M, с който да означим максималната стойност, която може да се появи във всяка от клетките (разредите). В примера с километража на кола тази стойност е $M = 9$, което съответства на използваната в ежедневието ни десетичната бройна система.

Следващият фрагмент показва как се добавя една единица към най-младшата клетка и се осъществява пренос (когато е необходим) към по-старшите клетки:

```
c=1;
for(int j=1;j<=N;j++)
 {
  a[j] += c;
  if(a[j]>M){a[j]=0;c=1;}
  else c=0;
 }
```

Реализиран е цикъл, при който се обхождат клетките с номера от 1 до N. За клетката с номер 1 се извършва увеличаване на стойността ѝ с единица, защото първоначално променливата `c` има стойност 1. При „препълване“, т.е. когато стойността на някоя клетка надмине M, променливата `c` получава отново стойност 1, с цел да бъде прибавена тази стойност към следващата клетка. Когато при излизане от цикъла `for` стойността на `c` не е нула, това ни дава информация, че всички клетки в масива са приели максимално възможната си стойност.

Забележка: Обикновено елементите на масива записваме като `a[1]`, `a[2]`,

..., `a[N]`. Когато с тях моделираме многоцифрен брояч, удобно е да си ги представяме (и отпечатваме) обратно: `a[N]`, `a[N-1]`, ..., `a[2]`, `a[1]`.

В следващата програма е включен предишният фрагмент и той се извиква многократно чрез оператора `do...while`, докато всички клетки на масива `a` се заредят с максималната си стойност. Първоначално масивът `a` е нулиран и при всяка промяна на състоянието му неговото съдържание се отпечатва. Масивът `a` е обявен с по-голяма дължина `N_max`, отколкото е дължината `N`, с която той се ползва в програмата. Това е необходимо например, когато искаме стойността на `N` да се въвежда от клавиатурата.

```
#include <iostream.h>
const int N_max=100;
int a[N_max];
void main()
 {int c,j,M,N;
  cin >> N >> M;
  do
  {for(j=N;j>=1;j--) cout << a[j];
   cout << "\n";
   c=1;
   for(j=1;j<=N;j++)
    {a[j] +=c;
    if(a[j]>M){a[j]=0;c=1;}
    else c=0;}
  }while(c==0);
 }
```

При въведени стойности $N = 3$ и $M = 2$ програмата отпечатва следната последователност от тройки числа (всичките тройки се отпечатват последователно една след друга, но тук за прегледност те са подредени в 3 колони):

```
000     100     200
001     101     201
002     102     202
010     110     210
011     111     211
012     112     212
020     120     220
021     121     221
022     122     222
```

Забелязваме, че всъщност това са всичките възможни начини на комбиниране, които показват как на 3 позиции могат да се поставят знаците 0, 1 и 2. По-общо, така можем да получим всички начини за разполагане на знаците

0, 1, 2, ..., M върху N позиции, по един знак в позиция. Прието е всеки един от тези начини да се нарича *вариация* (по-точно: *вариация с повторение* на знаците – в различните позиции може да има еднакви знаци).

Номериране на вариациите. В предишната програма, там където отпечатваме поредната образувана вариация, може да поставим оператор, който да увеличава с единица стойността на брояч. Така заедно с отпечатването на вариацията ще имаме и нейния пореден номер. Разбира се, номерирането ще се извършва по реда, по който нашата програма поражда вариации. При друг начин на пораждане номерирането ще е друго.

Множества. Ако разполагаме с няколко различни предмета, например с 3 монети от различна стойност: 1, 2 и 5 стотинки, може да се интересуваме по колко различни начина е възможно да се образуват съвкупности от тези предмети. В случая две образувани съвкупности са неразличими, когато са съставени от едни и същи предмети, без значение реда на изброяването им. Например съвкупността от двете монети със стойности 1 и 2 стотинки е същата съвкупност, която бихме описали като съвкупността от две монети със стойности 2 и 1 стотинки. Такива съвкупности, при които се интересуваме от броя и вида на принадлежащите им предмети (но не и от подредбата им), се наричат *множества*.

Всички множества, които могат да се образуват с разглежданите 3 монети, са следните 8 множества:

1) 1 ст.;
2) 2 ст.;
3) 5 ст.;
4) 1 ст., 2 ст.;
5) 1 ст., 5 ст.;
6) 2 ст., 5 ст.;
7) 1 ст., 2 ст., 5 ст.;
8) множество, в които няма монети, т.е. *празното множество*.

Общият брой на всичките възможни множества, образувани от 3 предмета, се изразява с $2 \cdot 2 \cdot 2 = 2^3 = 8$. Това е така, защото всеки от предметите може да участва или да не участва в множеството. При всяка от двете възможности за първия предмет, слагаме по две възможности за втория предмет. Получават се $2 \cdot 2 = 4$ възможности. След това, ако към всяка от тези 4 възможности сложим по 2 възможности за третия предмет, получаваме всичките 8 възможности: $4 \cdot 2 = 8$.

Да съпоставим на всеки от предметите по една променлива, която да приема стойност 1 или 0, и в зависимост от тази стойност да причисляваме предмета към множеството или да не го причисляваме там. Понеже разглеждаме едновременно няколко предмета, удобно е да вземем тези променливи като елементи на масив `a[1], a[2], ... , a[N]`, където N е броят на предметите.

Сега можем да получим (генерираме) всичките възможни множества, които могат да се образуват с N предмета, като използваме всичките стойности на многоцифрен брояч, при който стойностите, които може да приема всяка от клетките му, са 0 и 1. Следващата програма генерира всичките множества от 3 предмета и ги отпечатва. Всъщност тази програма е леко видоизменена версия на предишната. Да напомним, че елементите на глобалния масив `a[100]` са нулирани автоматично:

```
#include <iostream.h>
int a[100], M=1, N=3;
void main()
{int c, j;
 do
 {cout << "{ ";
  for(j=1;j<=N;j++)
   if(a[j]) cout << j << " ";
  cout << "}\n";
  c=1;
  for(j=1;j<=N;j++)
  {a[j] +=c;
   if(a[j]>M){a[j]=0;c=1;}
   else c=0;}
 }while(c==0);
}
```

Резултатът, който се извежда, е следният:

{ }
{ 1 }
{ 2 }
{ 1 2 }
{ 3 }
{ 1 3 }
{ 2 3 }
{ 1 2 3 }

Всяко от множествата е отпечатано на отделен ред и елементите му са заградени с фигурни скоби. Първото от изведените множества е празното.

За начина, по който е използван масивът `a` в програмата, понякога се казва, че това е начин за генериране стойностите на *двоичен вектор.*

Пермутации. Пермутация на числата $1, 2, 3, ..., N$ се нарича всяко тяхно разместване. Една възможност за последователното получаване на всичките пермутации (разместванията) на тези числа може да реализираме чрез програмата, с която отпечатахме всички вариации на числата $0, 1, 2, .., M$, поместени на N позиции. За целта трябва така да променим програмата, че всеки

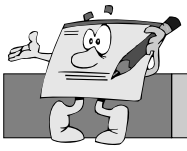
път преди отпечатването да проверяваме дали има повтарящи се стойности в масива `a` и при наличие на такова повторение, да не извеждаме на екрана съответни стойности. Проверката за повтарящи се стойности ще извършим с функцията `isEqVal(a,m,n)`, разгледана в Глава 14. Ще приложим тази функция с извикването `isEqVal(a,1,N)`. Освен това, трябва да вземем M да е равно на $N-1$ и при отпечатването да увеличим с единица всяка извеждана стойност, за да бъде тя в интервала от 1 до N. Следната програма ще изведе всички пермутации на числата 1, 2, 3. Съществено е, че масивът `a[100]` е определен като глобална променлива извън тялото на функцията `main()`, защото по този начин се осигурява автоматично нулиране на елементите му:

```
#include <iostream.h>

int isEqVal(int a[], int m, int n)
 {for(int i=m; i<n; i++)
  for(int j=i+1; j<=n; j++)
   if(a[i]==a[j]) return 1;
  return 0;
 }

int a[100], N=3, M=N-1;

void main()
{int c, j;
 do
 {if(!isEqVal(a,1,N))
   {for(j=N;j>=1;j--) cout << a[j]+1;
    cout << " ";}
  c=1;
  for(j=1;j<=N;j++)
  {a[j] +=c;
   if(a[j]>M){a[j]=0;c=1;}
   else c=0;}
 }while(c==0);
}
```

Резултатът, който извежда програмата, показва шесте различини пермутации на трите числа 1, 2 и 3:

```
123 132 213 231 312 321
```

Забележка: За решаване на някои от задачите в тази глава съществуват по-ефективни алгоритми, които тук не разглеждаме. Някои от тях могат да се намерят в посочените източници в края на книгата.



УПРАЖНЕНИЯ

1. Защо следното твърдение възприемаме като виц: „Имам две монети с обща стойност 55 ст., едната не е от 5 стотинки. Каква е стойността на другата? – Отговор: Другата е от 5 стотинки.“?

Упътване: Двете монети образуват множество (редът в който ги разглеждаме няма значение) и затова понятията „едната“ и „другата“ монета в случая са неуместни.

2. В една програма е дефиниран масив по следния начин: `int a[N]`, където `N` е константа, дефинирана преди това. Напишете функция, която премества *надясно* с по една позиция елементите на масива `a`. Напишете друга функция, която премества елементите *наляво*.

Упътване: За преместване надясно използвайте цикъла

```
for(int i=N-2; i>=0; i--) a[i+1]=a[i];
```

3. Променете програмата за отпечатване на всички пермутации, така че тя да може да прочита стойността на N от клавиатурата.

4. Да се образуват всички начини (вариации) за разполагане на дадените по-долу знаци, по един знак в позиция:

а) знаците 0 и 1 в 3 позиции;

б) знаците 0 и 1 в 4 позиции;

в) знаците a, b и c в 3 позиции.

Можете ли да предвидите колко ще бъде броят на начините?

5. Да се образуват всички пермутации на числата 0, 1, 2, 3, 4.

6. Напишете програма, която по даден номер на вариация намира самата вариация.

7. *Комбинация* на знаците $0, 1, 2, ..., M$, разположени на N позиции, наричаме такава вариация, при която знаците са разположени в строго растящ ред. Напишете програма, която поражда всичките комбинации при въведени стойности на M и N.

Упътване: Видоизменете програмата за отпечатване на всички вариации или пермутации. Вместо функцията `isEqVal` разгледайте функцията `isSorted0` от Глава 14.

8. Напишете програма, която въвежда цялото положително число n и намира всички n-цифрени числа.

Забележка: Когато стойността на n е фиксирана като конкретно число, например 3, задачата може да се реши чрез прилагане на съответния

брой вложени цикли. При стойност на n, която се въвежда от клавиатурата, подходът с вложени цикли е неприложим, защото не знаем какъв ще бъде броят им.

9. Напишете програма, която да извежда всичките четирицифрени числа, които нямат повтарящи се цифри. Колко е броят на тези числа? Обобщете за n-цифрени числа.

10. Напишете програма, която извежда всичките n цифрени числа, имащи k цифри, равни на 0, а останалите им цифри да са равни на 1.

11. Напишете програма, която показва всички начини на образуване на сумата от a стотинки чрез използване на N монети, които са със стойности $b_1, b_2, ..., b_N$ (възможно е някои от тези стойности да са равни помежду си).

12. Разположете 8 царици върху шахматна дъска (с размери 8 на 8), така че никоя от тях да не е под удар на останалите. Решете задачата и за n царици върху шахматна дъска с размери n на n.

13. За m различни предмета са дадени теглото и цената на всеки от тях. Определете кои от предметите трябва да се вземат, така че общото им тегло да не надминава t, а общата им цена да е максимална.

КОНКУРСНА ЗАДАЧА

15.1. Разместване (Национална олимпиада, общински кръг, 2002, зад. D2). Да се направи програма, която въвежда от клавиатурата естествено число M и извежда на екрана M реда, първият от които е 1 2 3 4 5 6 7 8 9, а всеки следващ се получава от предишния, като първото число минава на последно място.

Примери:

Вход: 3
Резултат:
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 1
3 4 5 6 7 8 9 1 2

Вход: 10
Резултат:
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 1
3 4 5 6 7 8 9 1 2
4 5 6 7 8 9 1 2 3
5 6 7 8 9 1 2 3 4
6 7 8 9 1 2 3 4 5
7 8 9 1 2 3 4 5 6
8 9 1 2 3 4 5 6 7
9 1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9

Решение: Записваме целите числа от 1 до 9 в масив и след това в цикъл всеки път вземаме първото от числата и го запазваме в променливата `b`, преместваме останалите с по една позиция наляво и слагаме запазеното в `b` число на последното място в масива.

```
#include<iostream.h>
int a[]={1,2,3,4,5,6,7,8,9};
void main()
{  int i,j,M;
   cin >> M;
   for(j=0;j<9;j++) cout << a[j];
   cout << "\n";
   for(i=2;i<=M;i++)
   {
     int b=a[0];
     for(j=0;j<8;j++) a[j]=a[j+1];
     a[8]=b;
     for(j=0;j<9;j++) cout << a[j];
     cout << "\n";
   }
}
```

Глава 16. НИЗОВЕ

В компютърните програми, освен извършване на операции с числа, твърде често се налага да се обработват и текстови данни. Тези данни могат да се реализират чрез масиви от знаци с базов тип `char`. Да си припомним, че всеки *знак* се записва, като се огражда с единични кавички (апострофи), например `'a'`, `'b'`, `'c'`. Променлива, обявена с тип `char`, може да се използва за присвояване на такива стойности. Например:

```
char x;
x='a';
```

Друг начин за задаване на стойност е това да се извърши едновременно с обявяването на променливата:

```
char x='a';
```

Константи от тип знаков низ. *Знаковият низ* може да се разглежда като последователност от няколко знака. Константи от такъв вид вече сме използвали в оператора за извеждане на данни. Например:

```
cout << "Стойността на x е равна на ";
```

Знаците, оградени с двойни кавички, представляват *константа от тип знаков низ*. Понякога вместо „знаков низ“ казваме за по-кратко „низ“.

Променлива от тип знаков низ. Една константа от тип знаков низ може да бъде присвоена на променлива от тип знаков низ. Това може да стане в момента на обявяването ѝ, например в началото на програмата:

```
char a[ ]="Това е знаков низ";
```

Конструкцията, с която се обявява променливата `a`, показва, че всъщност тази променлива е масив от стойности, чийто тип е `char`. В квадратните скоби след името на променливата не е написан размерът на масива, защото компилаторът лесно може да го определи от броя на знаците, разположени между двете двойни кавички. В този брой се включва и броят на празните интервали. Освен това, всеки знаков низ завършва със специален „невидим“ знак, чието предназначение е да маркира точно края на последователността от знаци. Този специален знак се записва като `'\0'`. Така размерът на масива, определен с декларацията

```
char s[ ]="abcd";
```

е равен на 5 и елементите на този масив имат следните стойности: `s[0]='a'`, `s[1]='b'`, `s[2]='c'`, `s[3]='d'` и `s[4]='\0'`.

Това ни подсказва друг възможен начин за работа с масиви от знаци. Първо обявяваме променливата `s` като масив от знаци с точно определен размер:

```
char s[5];
```

и след това записваме присвояванията:

```
s[0]='a'; s[1]='b'; s[2]='c'; s[3]='d'; s[4]='\0';
```

Броят елементите на масива от знаци, без последния специален знак `'\0'`, се нарича дължина на знаковия низ. Така низът `s` има дължина 4. При обявяването на `s` трябва да напишем за дължина такова число, което да е с единица по-голямо от „истинската“ дължина на низа, поради необходимостта от допълнителен знак `'\0'` за маркиране на края.

Всъщност, при обявяването на променлива от тип знаков низ, се има предвид *максимално допустимата дължина* на низа, който може да се присвои на тази променлива. Независимо че в нашия пример променливата `s` е предназначена за съхранение на низ с дължина 4 знака, тя може да се използва и за низ с по-малка дължина. Например низ с дължина 3 може да се зададе така:

```
s[0]='a'; s[1]='b'; s[2]='c'; s[3]='\0';
```

Възможно е използването на низ от един знак:

```
s[0]='a'; a[1]='\0';
```

или даже на празен низ (с дължина 0):

```
s[0]='\0';
```

Понеже числовата стойност на знака `'\0'` е нула, празният низ е възможно да бъде зададен и чрез присвояването

```
s[0]=0;
```

Така виждаме, че съдържанието и дължината на променлива от тип знаков низ може да се променят по време на работа на програмата. Във всеки момент актуалната дължина на низа `s` може да се получи с прилагане на стандартната функция `strlen(s)`, за която е необходимо включването `#include<string.h>`.

Операторът `cin` може да се използва за въвеждане на низ от клавиатурата, а извеждането му на екрана се осъществява чрез `cout`. Повече подробности за използването на оператора `cin` при въвеждане на низ са дадени в Глава 21.

Забележка: Ако се опитаме да изведем на екрана чрез оператора `cout` низ, за който сме забравили да осигурим знака `'\0'`, определящ края на низа, последствията са *непредсказуеми* – ще започне да се извежда всичко,

което е в паметта на компютъра от началото на низа до ... намирането на знак `'\0'`, за който не е известно къде е.

Пример 1. Следващата програма въвежда от клавиатурата един знаков низ, след това последователно намалява дължината му с единица, докато се получи празният низ. Понеже променливата, в която ще се съхранява низът, е определена с максимална дължина 11, за този низ от клавиатурата трябва да се въведат най-много 10 знака.

```
#include <iostream.h>
#include <string.h>

char s[11];

void main()
{
   int n;
   cin >> s;
   do
   { cout << s << " ";
     n=strlen(s);
     cout << n << "\n";
     s[n-1]='\0';
   }
   while(n>0);
}
```

Достъп до знак от знаков низ. Както при масивите, така и при променливите от тип знаков низ, имаме достъп до всеки отделен елемент на низа чрез указване на неговия номер. Ако `s` е променлива от тип знаков низ, то `s[i]` е валидно обръщение към променлива от тип `char`, при условие, че стойността на индекса `i` е цяло неотрицателно число, ненадминаващо дължината на низа. Използвайки този достъп, можем например да променим само един елемент на низа – чрез присвояването `s[i]=c`, или да присвоим стойността на i-тия елемент на друга променлива `d`, която е от тип `char`, например: `d=s[i];`.

Пример 2. Последователно обхождане на всичките елементи на низа `s`, от елемента с индекс 0 до последния му елемент, се извършва по следния начин, където при обхождането се отпечатват съответните елементи:

```
int n=strlen(s);
for(int i=0;i<n;i++) cout << s[i];
```

Използване на стандартни функции при работа с низове. При работа със знакови низове много често се използват и други стандартни функции,

освен функцията `strlen()`. За разлика от присвояването на число или на знак от променлива от съответен тип, присвояването на низ НЕ може да стане с използваната досега конструкция: *име на променлива = константа*. За целта се прилага стандартната функцията за копиране, наречена `strcpy()`. Ако в програмата са определени два низа по следния начин:

```
char a[10], b[10];
```

зареждането на първата променлива с константата `"xyz"` става чрез извикване на

```
strcpy(a,"xyz");
```

а зареждането на променливата `b` със стойност, която има в променливата `a`, се извършва чрез

```
strcpy(b,a);
```

В общия случай действието на функцията `strcpy()` може да се представи като

`strcpy(`*приемник на стойност*, *източник на стойност*`);`

Програмистът винаги трябва да е сигурен, че максималната дължина на променливата, която е *приемник на стойност*, е по-голяма или равна на актуалната дължина на *източника на стойност*.

Конкатенация. Често използвана операция при работа с низове е слепването на два низа. Тази операция се нарича *конкатенация* и се реализира чрез стандартната функция `strcat()`. Общият случай на употреба на тази функция се записва така:

`strcat(`*приемник на стойност*, *добавена стойност*`);`

Низът, означен като *добавена стойност*, се долепва отзад на стойността, която се съдържа в променливата, означена като *приемник на стойност*. Например, ако `st1` съдържа `"abc"` и `st2` съдържа `"def"`, то след прилагане на извикването `strcat(st1,st2)` променливата `st1` ще съдържа низа `"abcdef"`, а променливата `st2` няма да се промени. Един знак може да бъде долепен отдясно на низа `st1` по следния начин: `strcat(st1,"g")`. Тогава `st1` ще има стойност `"abcdefg"`.

Преобразуване на низ, съдържащ число, записано с римски цифри. В Глава 6 разгледахме програма, която преобразува отделно взета римска цифра, като я отпечатва като десетично число. Сега ще дефинираме функция `v(c)`, която приема като параметър знака `c`, и ако той е римска цифра, функцията връща съответната стойност във вид на цяло число:

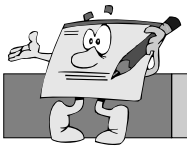
```
int v(char c)
{
   switch(c)
   {case 'M': return 1000;
    case 'D': return 500;
    case 'C': return 100;
    case 'L': return 50;
    case 'X': return 10;
    case 'V': return 5;
    case 'I': return 1;
    default: return 0;
   }
}
```

За да преобразуваме в десетична числова стойност произволен низ, съставен от римски цифри, ще използваме, че съгласно правилата, по които се образуват числата с римски цифри, ако една римска цифра r_1 се намира преди друга (r_2) и числовата стойност на r_1 е по-малка от стойността на r_2, то стойността на r_1 трябва да се извади; във всички други случаи стойностите на римските цифри се натрупват със събиране. Например „XIV“ има стойност $10 - 1 + 5 = 14$.

В следващия фрагмент се използва дефинираната по-горе функция `v(c)`. Стойността на знаковия низ `s` се въвежда от клавиатурата и накрая се пресмята числовата стойност в променливата `a`.

```
char s[20];
cin >> s;
int n=strlen(s)-1;
int a=v(s[n]);
for(int i=n-1;i>=0;i--)
  {if(v(s[i])<v(s[i+1])) a -= v(s[i]);
   else a += v(s[i]);
  }
cout << a;
```

Знаков низ като параметър на функция. Понеже един масив може да бъде параметър на функция, това е възможно и за знаков низ. Например следният заглавен ред на функция е допустим: `int f(char s[ ], int n)`.



УПРАЖНЕНИЯ

1. Каква е разликата между максималната дължина на един низ и неговата актуална дължина?

2. Виждате ли разлика между празния низ `char s[ ]="";` и низа, съдържащ една шпация `char s[ ]=" ";`?

3. В една програма има обявени `char c; char s[100];`. Кои от следните оператори са грешни?

а) `c='abc';` б) `s="abc";` в) `c='c';`
г) `s=c;` д) `s[1]=c;` е) `c=s;`

4. Кои от конструкциите са възможни и кои не са?

а) `char str1[5] = {'a', 'b', 'c'};`

б) `char str2[3] = {'a', 'b', 'c', 'd', 'e'};`

в) `char str3[4] = {'a', 'b', '\0', '\0', '\0'};`

г) `char r[ ] = {'a', 'b', '\0'};`

д) `char q[5]; q = {'a', 'v', 's'};`

е) `char r1[ ];`

5. Какъв заглавен файл е необходимо да включим, когато използваме стандартните функции за работа с низове?

6. В настоящата глава е казано, че на променлива от тип знаков низ не може да се присвои стойност с конструкцията *име на променлива = константа*, затова се налага да се ползва стандартната функция `strcpy()`. Обяснете защо обаче е възможно да напишем

```
char a[ ]="Това е знаков низ";
```

7. Напишете програма, която проверява дали един знаков низ е симетричен спрямо средата си.

8. Напишете функция, която за даден знаков низ пресмята броя на цифрите в него.

9. Напишете функция, която разпознава дали един знаков низ може да бъде идентификатор на променлива.

Упътване: Идентификаторът (името) на променлива в езика C/C++ е последователност от букви от латинската азбука (малки и големи), цифри и знака долна черта `'_'`, като първият знак на идентификатора не може да бъде цифра.

10. Напишете функция, която пренася i-тия знак от един низ на мястото на j-тия знак в друг низ.

11. Даден е знаков низ. Съставете друг низ, съдържащ знаците на дадения, но в обратен ред.

12. Даден е знаков низ. Напишете функция, която пренарежда знаците в същия низ, така че те да бъдат подредени в ред, обратен на първоначалния.

13. Напишете програма, която разменя стойностите на два знакови низа.

14. Напишете програма, която въвежда 3 знакови низа в съответни променливи от тип знаков низ `a`, `b` и `c`, долепя първия низ след втория, като записва резултата в нова променлива от тип знаков низ `d` и накрая разменя стойностите на `c` и `d`.

15. Напишете програма, която въвежда два знакови низа `x` и `y` с еднаква дължина (не по-голяма от 5 знака). Двата низа съдържат само цифрите 0, 1, 2, ... , 9. В програмата трябва да има още 5 променливи от тип знаков низ: `s1`, `s2`, ..., `s5` и тя трябва да зареди толкова на брой променливи измежду тях, колкото е броят на знаците в низа `x`, като низът `s1` трябва да има дължина, равна на стойността на първата цифра в `x`, и всичките му знаци да са равни на такава главна буква от латинската азбука, която има пореден номер в азбуката, равен на първата цифра в низа `y`; низът `s2` трябва да има дължина, равна на стойността на втората цифра в `x`, и всичките му знаци да са равни на тази главна буква от латинската азбука, която има пореден номер в азбуката, равен на втората цифра в низа `y`; и т.н.

КОНКУРСНИ ЗАДАЧИ

16.1. Хистограма (Зимни празници, Варна, 2003, зад. D3). Даден е низ, съставен от N десетични цифри ($1 < N < 20$). Напишете програма, която въвежда низа и отпечатва *хистограмата* на неговите цифри (виж примера). Хистограмата на N числа се състои от N стълбчета от звездички, като стълбчето на всяко число трябва да има толкова звездички, колкото е самото число. Стълбчетата трябва да са подравнени отдолу, като под всяко стълбче трябва да стои и съответната десетична цифра. Стълбчетата трябва да са едно до друго и да са подредени в реда, по който са подредени цифрите в низа.

Входните данни трябва да се четат от клавиатурата. Единственият ред на входа ще съдържа зададения низ. Резултатът – хистограмата – трябва да се изведе на екрана.

Пример: Вход: 2031894 Изход:

```
   *
  **
  **
  **
  **
  ***
 * ***
* * ***
* *****
2031894
```

Решение: След прочитане на низа от цифри и преобразуването им като елементи на целочислен масив, започваме цикъл, в който „отгоре-надолу“ проверяваме в кои стълбчета трябва да има звездички и ги отпечатваме. Накрая отпечатваме и самите числени стойности.

```
#include<iostream.h>
#include<string.h>
char s[21];
int n;
int a[21];

void main()
{ cin >> s;
  int i,j;
  n=strlen(s);
  for(i=0;i<n; i++) a[i]=s[i]-'0';

  for(j=9;j>0;j--)
  {for(i=0;i<n;i++)
    if(a[i]<j) cout << " "; else cout << "*";
   cout << "\n";
  }

  for(i=0;i<n;i++) cout << a[i];
  cout << "\n";
}
```

Забележка: Ако във въведения низ липсва цифрата 9, програмата ще отпечата един или повече празни редове. Предлагаме на читателя да отстрани този недостатък.

16.2. Единствен елемент (Есенен турнир, Шумен, 2002, зад. D3). Напишете програма, която въвежда от клавиатурата редица от знаци и извежда на екрана първия неповтарящ се елемент от редицата. Елемент на въведената редица може да бъде всеки видим знак, който се въвежда от клавиатурата, а краят на редицата се определя с натискане на клавиша Enter. Броят на знаците може да бъде най-малко равен на 1 и най-много на 80. Ако задачата няма решение, програмата не трябва да извежда нищо.

Пример 1:
Въвеждаме: `bab%_cgAb`
Програмата трябва да изведе: `a`

Пример 2:
Въвеждаме: `vvvvvvvvv`
Програмата нищо не извежда.

Решение: С външния цикъл пробягваме елементите в низа `s`. За всеки така избран елемент с номер `i` с вътрешния цикъл пробягваме останалите елементи чрез индекса `j`. Променливата `p` служи за флаг, който отбелязва кога сме открили, че `s[i]==s[j]`. Така намираме търсения елемент `s[i]` с най-малката възможна стойност на индекса си. От програмата се излиза веднага след отпечатването му.

```
#include<iostream.h>
#include<string.h>
char s[100];

void main()
{int i,j,n;
 cin >> s;
 n=strlen(s);
 for(i=0;i<n;i++)
 { int p=0;
   for(j=0;j<n;j++)
    if(i!=j)
     if(s[i]==s[j]) p=1;
   if(p==0) {cout << s[i] << "\n"; return;}
 }
}
```

16.3. Низ (Пролетен турнир, Ямбол, 2001, зад. D1). Даден е низ, съставен от големи и малки латински букви и цифри с дължина N ($2 \leqq N \leqq 62$). Напишете програма, която въвежда низа и определя дали в него има знак (буква или цифра), който се среща поне 2 пъти. Ако няма такъв знак, програмата да извежда думата `NO`, а ако има, програмата да извежда две числа от 1 до N – позициите в низа, в които са намерени еднакви знаци.

Пример:

Вход: abcddcd
Изход: 3 6

Решение: Във външния цикъл с индекса `i` обхождаме последователно знаците от низа `s[i]`. За всяка стойност на `i` чрез индекса `j` на втория цикъл обхождаме знаците от низа след `i`-тия и сравняваме `s[i]` и `s[j]`.

```
#include <iostream.h>
#include <string.h>
char s[100];
void main()
{ int i,j,n;
  cin >> s;
  n=strlen(s);
  for(i=0; i<n-1; i++)
  for(j=i+1; j<n; j++)
   if(s[i]==s[j])
    { cout << i+1 << " " << j+1 << "\n";
      return;
    }
  cout << "NO\n";
}
```

16.4. Най-ляв (Пролетен турнир, Ямбол, 2001, зад. D2). Даден е низ, съставен от големи и малки латински букви и цифри, с дължина N ($50 \leqq N \leqq 255$), в който има повтарящи се знаци. Напишете програма, която въвежда низа и определя най-ляво разположените два еднакви знака в него. Казваме, че два еднакви знака в низа са разположени най-ляво, ако вляво от десния няма други два еднакви знака. Програмата трябва да извежда две цели числа между 1 и N – позициите в низа, в които са намерени най-ляво разположените два еднакви знака.

Решение: Тази задача прилича на предишната, но за дадения там тестов пример с вход `abcddcd` сега изходът трябва да бъде `4 5`, а не `3 6`. За да открием най-ляво разположените два еднакви знака, започваме да търсим отзад напред: за всеки знак с номер `i`, пробягван от последния към първия, търсим

еднакъв с него знак с по-малък номер `j<i`. При всяко намиране запомняме номерата съответно в `i0` и `j0`. Последното извършено присвояване на стойности в `i0` и `j0` ще даде отговора на задачата. Според условието сме сигурни, че във всеки въведен низ има повтарящи се знаци.

```
#include <iostream.h>
#include <string.h>
char s[300];
void main()
{ int i,j,i0,j0,n;
  cin >> s;
  n=strlen(s);
  for(i=n-1; i>0; i--)
  for(j=i-1; j>=0; j--)
   if(s[i]==s[j])
    {i0=i; j0=j;}
  cout << j0+1 << " " << i0+1 << "\n";
}
```

16.5. Кръстосване (Национална олимпиада, областен кръг, 2002, зад D2). Напишете програма, която въвежда от клавиатурата два низа, съставени от главни латински букви, всеки с дължина най-много 15 знака. Ако двата низа имат поне една обща буква, програмата трябва да отпечата хоризонтално на екрана първия от низовете, а втория – вертикално, без да има празни позиции между буквите им, така че двата низа да се пресекат там, където общата буква се среща за първи път и в двата низа. Ако двата низа нямат нито една обща буква, тогава вашата програма трябва да отпечата хоризонтално първия низ, а от позицията, намираща се непосредствено под позицията, следваща след последната буква на първия низ – да отпечата вертикално надолу втория низ.

Пример:

1. Въвеждаме `KLAVIATURA` и `MONITOR`.
Възможен вид на изхода:

```
    M
    O
    N
KLAVIATURA
    T
    O
    R
```

1. Въвеждаме `KOTKA` и `PILE`.
Възможен вид на изхода:

```
KOTKA
     P
     I
     L
     E
```

Решение: Двата низа трябва да се пресекат там, където общата буква се среща за *първи път* и в двата низа. Намирането на номерата `x` и `y` на тези позиции става чрез двойния цикъл в началото на програмата. В зависимост от това дали има обща буква, или няма, се извършва отпечатване на търсената конфигурация ред по ред, отгоре надолу.

```
#include <iostream.h>
#include <string.h>
char a[20], b[20];
int i,j,k,m,n,x,y;

void main()
{ cin >> a; m=strlen(a);
  cin >> b; n=strlen(b);
  x=-1; y=-1;
  for(i=0;i<m;i++)
  for(j=0;j<n;j++)
   if(a[i]==b[j])
    if((x==-1)&&(y==-1)){x=i; y=j;}

  if(x != -1)
  for(i=0;i<n;i++)
  { if(i==y){cout << a << "\n";}
    else
     { for(k=0;k<x;k++) cout << " ";
       cout << b[i] << "\n"; }
  }
  else
  { cout << a << "\n";
    for(i=0;i<n;i++)
    { for(k=0;k<m;k++) cout << " ";
      cout << b[i] << "\n";
    }
  }
}
```

Глава 17. ОПЕРАЦИИ С НИЗОВЕ

Подобно на сравняването на числа (за равенство, по-голямо или по-малко), възможно е да сравняваме два низа `st1` и `st2`. Логически условия, в които участват такива сравнения, могат да се напишат на подходящи места в оператори като `if` или `while` или другаде в програмата. За образуването на тези условия се използва функцията `strcmp()`, за която е необходимо в програмата да се напише включването `#include<string.h>`.

Изразът `strcmp(st1,st2)` има стойност, равна на нула, тогава и само тогава, когато двата низа `st1` и `st2` имат еднаква дължина и всичките им елементи почленно съвпадат. В този случай казваме, че двата низа са *равни*. Например „abc“ и „acb“ не са равни низове, както не са равни низовете „aaa“ и „aa“.

Лексикографска наредба. При сравняване за неравенство между два низа `st1` и `st2` се прилага *лексикографска наредба*. Тази наредба се основава на сравняването на знаци. *Наредбата между знаците* се определя чрез таблицата ASCII (American Standard Code for Information Interchange – Американски стандарт за обмен на информация) – един знак се счита за по-малък от друг, ако има по-малък номер в тази таблица. Например шпацията има номер 32 и следователно е по-малка от всеки друг видим знак. Наредбата на знаците обобщава обичайната наредба на буквите в латинската или в българската азбука, за която знаем какво означава една буква да предхожда друга – например ‘а’ е преди ‘б’, а ‘ъ’ е след ‘ъ’. Всяка главна буква от латинската или от българската азбука е преди коя да е малка буква от същата азбука.

Съгласно общоприетото понятие за лексикографската наредба, за да се определи дали един низ е по-малък от друг низ, се прилага следното правило:

Започваме едновременно поелементно преглеждане на двата низа в посока отляво надясно. В момента, когато в низовете срещнем два съответни знака, които не са равни, тогава за по-малък низ се обявява този, при който съответният знак е по-малък. Ако това не се случи до края на поне един от низовете, за по-малък се обявява този низ, който е по-къс. Ако обаче в този момент двата низа са с равни дължини, това означава, че низовете са еднакви, следователно равни при лексикографската наредба.

Пример. Нека

```
char s1[ ]="Това е първи низ";
char s2[ ]="Това е втори низ";
```

От двата низа `s1` и `s2` вторият е по-малък от първия, защото, когато започнем да сравняваме поелементно знаците им отляво-надясно, първия път, когато попадаме на неравни знаци, имаме буквата „п“ в низа `s1` и буквата „в“

в низа `s2`. Понеже азбучният ред на „в“ е преди „п“, следва че целият низ `s2` е преди `s1`, т.е. `s2` е по-малък от `s1`.

Съгласно правилото на лексикографската наредба, ако първият знак на низа `st1` е по-малък от първия знак на низа `st2`, тогава и целият низ `st1` е по-малък от `st2`.

За наредбата при низове използваме обичайното означение за неравенство: `st1 < st2`, въпреки че то не е стандартно за езика C/C++. Да отбележим, че не винаги, когато един низ е по-малък от друг, той трябва да е по-къс от него. Например низът `"aaa"` е по-малък от `"xx"`. Когато един низ е по-малък от друг, казваме, че той е по-напред в лексикографската наредба или, че първият низ предшества втория.

Примери за лексикографска наредба: Следните условия са верни

```
"a" < "b", "aa" < "ab", "aaa" < "aab", "a" < "aa", "A" < "a",
```

а следните не са верни:

```
"a" < "a", "ab" < "aa", "aab" < "aaa", "ba" < "aa", "aa" < "a".
```

По аналогичен начин на означението „по-малко“ (`<`) се употребява и означение „по-голямо“ (`>`) между низове. При работа с низове се използват още и знаците `"<="`, `">="` и `"="`, чийто смисъл е достатъчно очевиден. В езика C++ е възможно да се програмира употребата на тези знаци, така че тя да се разшири и за отношения между низове, но тук няма да разглеждаме тази възможност.

В езика C/C++ за определяне на отношението между низовете `s1` и `s2` се използва функцията `strcmp(s1,s2)`. Вече споменахме, че ако

```
strcmp(s1,s2)==0,
```

низовете са равни. За другите стойности на тази функция е в сила:

ако `strcmp(s1,s2)<0`, тогава `s1` е по-малък от `s2`;
ако `strcmp(s1,s2)>0`, тогава `s1` е по-голям от `s2`;

Основен пример за практическо прилагане на лексикографската наредба е подреждането на думите в речниците. Лексикографската наредба често се среща и под наименованието *азбучна наредба*.

Други операции със знакови низове. Освен разгледаните досега операции, се прилагат и някои други, за които е удобно да се ползват стандартни функции, включвани с `#include<string.h>`. Една част от тези операции се отнася за *поднизове*. В по-нататъшното изложение подниз на даден низ ще наричаме всяка част от низа, съставена от негови последователно разположени знаци.

Образуване на подниз. Функцията, чрез която може да образуваме подниз, като укажем докъде да бъдат взети знаците от даден низ, е сходна с функцията за копиране на низ и се нарича `strncpy()`:

`strncpy` (*приемник, източник, n*);

Тази функция пренася в променливата *приемник* първите n на брой знака от низа, съхраняван в променливата (или константата) *източник*. Ако низът в *източник* е по-къс от n, тогава се пренася целият низ, включително и неговият завършващ знак `'\0'`. За правилната работа на програмата максималната дължина на променливата *приемник* трябва да е най-малко равна на n. Пример:

```
char a[100];
strncpy(a,"123456789",20);
cout << a;
```

ще доведе до отпечатване на низа `"123456789"`. Ако непосредствено след това изпълним

```
strncpy(a,"abcdefghi",5);
cout << a;
```

ще се отпечата `"abcde6789"`, защото завършващият знак `'\0'` е останал на старото си място.

На мястото на втория аргумент вместо константа от тип низ, както е в предния пример, може да бъде поставена променлива от същия тип:

```
char a[100], b[100];
strcpy(b,"123456789");
strncpy(a,b,5);
a[5]=0;
cout << a;
```

което ще доведе до отпечатване на `"12345"`.

Подниз от междинна позиция. Когато искаме да отделим подниз, който започва от някоя междинна позиция m, можем да укажем на функцията `strncpy()` да третира втория си аргумент като низ, започващ от тази позиция, чрез поставянето там на израза `&b[m]`. Например, чрез фрагмента

```
char a[100], b[100];
strcpy(b,"123456789");
strncpy(a,&b[3],5);
a[5]=0;
cout << a;
```

ще се отпечата низът `"45678"`. По-общо казано, чрез конструкцията

```
strncpy(a, &b[m], n)
```

ще се получи начален подниз в променливата `a`, състоящ се от `n` последователни знака, взети от подниз, намиращ се в променливата `b`, а именно: `b[m]`, `b[m+1]`, ... , `b[m+n-1]`.

Проверка дали един низ е подниз на друг. За да познаем в програма дали стойностите на елементите от един низ `a` присъстват като подниз в друг низ `b`, използваме функцията

```
strstr(b,a);
```

която може да бъде проверена (например чрез оператора `if`) дали връща стойност нула. Върната стойност, която не е нула, показва, че низът `b` съдържа като подниз съдържанието на `a`. Когато върнатата стойност е равна на нула, низът `b` не съдържа никакъв свой подниз, еднакъв със съдържанието на `a`. Например следният фрагмент ще отпечата `"ДА"`:

```
if(strstr("123456789","345") != 0)
  cout << "ДА";
else cout << "НЕ";
```

Указател към знак от низ. Точно казано, стойността, която връща функцията `strstr(b,a)`, когато съдържанието на `a` е подниз на `b`, е *указател* към първото срещане в `b` на стойността на началния знак от `a`. Указател към знак се дефинира чрез

```
char *p;
```

Тогава след присвояването

```
p=strstr("123456789","345")
```

можем да използваме, че всъщност `p` е низ, започващ от първото срещане в `"123456789"` на първия знак от `"345"`, и завършващ в края на `"123456789"`. Така операторът за печат `cout << p;`, приложен след горното присвояване, ще отпечата `"3456789";`.

Общото понятие *указател* е твърде важно за езика C/C++, но тук няма да го разглеждаме подробно. Ще обърнем внимание на възможността, че когато правилно е присвоена стойността на указателя `p` към знак от низ, този указател може да се ползва в оператора за печат `cout << p;`.

Пример 1. Отделяне на подниз. В програма е дадена променливата `a`, съдържаща знаков низ. Да се отпечата този подниз на `a`, чийто първи знак е i-тият знак на `a` (броенето на знаците започваме от 0) и съдържа n знака. Предполагаме, че n не е прекалено голямо, т.е. поднизът, започващ с i-тия знак на `a` и имащ дължина `n`, не излиза извън последния елемент на `a`.

Решение:

```
// три реда с примерни данни:
char a[ ]="abcdef";
int i=2;
int n=3;
//
char *p;
p=&a[i];
p[n]='\0';
cout << p;
```

По-нататък променливата `p` може да се използва по обичайния начин като знаков низ.

Пример 2. Премахване на подниз. В програма е дадена променливата `a`, съдържаща знаков низ. Да се премахне такъв подниз на `a`, чийто първи знак е i-тият знак на `a` (броенето на знаците започваме от 0) и съдържа n следващи знака. Отново предполагаме, че n не е прекалено голямо, т.е. поднизът, започващ с i-тия знак на `a` и имащ дължина `n`, не излиза извън последния елемент на `a`. Премахването на указания подниз означава, че останалите елементи на `a` трябва да бъдат преместени напред, така че да не остане „дупка“.

Решение 1:

```
// три реда с примерни данни:
char a[ ]="abcdef";
int i=2;
int n=3;
//
int j=i;
int k=i+n;
while(a[k] != '\0')
 {a[j]=a[k]; j++; k++;}
a[j]='\0';
cout << a;
```

Решение 2: За да преместим с `n` позиции наляво подниза на `a`, който започва от позиция с номер `i+n` (той има дължина `strlen(a)-n-i+1` заедно с последния завършващ знак `'\0'`) и с това да покрием подниза, който започва от позиция с номер `i`, прилагаме функция за копиране. Така фрагментът, написан след примерните данни в Решение 1, може да се замени с

```
strncpy(&a[i], &a[i+n], strlen(a)-n-i+1);
cout << a;
```

Пример 3. Вмъкване на подниз. В програма са дадени променливите `a` и `b`, съдържащи знакови низове. Да се вмъкне в `a` съдържанието на низа

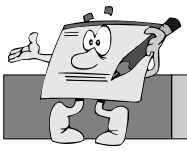
`b`, така че то да стане подниз на `a`, който да започва с i-тия знак на новия низ `a` (броенето на знаците започваме от 0). Предполагаме, че максималната дължина, с която е обявена променливата `a`, е достатъчно голяма, за да може новият низ да се събере в тази променлива.

Решение 1:

```
// три реда с примерни данни:
char a[100]="abcdef";
char b[100]="xyz";
int i=4;
//
int j,k,m,n;
m=strlen(a);
n=strlen(b);
j=m; k=m+n;
while (j>=i) {a[k]=a[j]; j--; k--;}
j=i; k=0;
while (k<n) {a[j]=b[k]; j++; k++;}
cout << a;
```

Решение 2: Ако низът `b` е с достатъчна максимална дължина, можем чрез `strcat(b,&a[i])` да залепим след него съдържанието на подниза от `a`, който започва от знака с номер `i`. След това копираме след знака с номер `i` в `a` получената стойност на `b`. Така фрагментът, написан след примерните данни в Решение 1, може да се замени с

```
strcpy(&a[i], strcat(b,&a[i]));
cout << a;
```



УПРАЖНЕНИЯ

1. Кои са стандартните функции за работа с низове?

2. Каква трябва да бъде лексикографската наредба на следните редици от низове?

а) aaab, bab, bbbba, aa, aab, ababab, b;

б) 000, 100, 111, 010, 110, 101, 001, 011;

в) 12, 56, 1000, 7, 2127, 213, 79, 713;

г) буква, азбука, език, буквар, наредба, аз, асо, финал, азимут.

3. За какво се използва функцията `strcmp(s1,s2)`?

4. Чрез коя функция образуваме подниз, като укажем докъде да бъдат взети знаците от даден низ?
5. По какъв начин можем да отделим подниз, който започва от някоя междинна позиция m?
6. Как ще познаем дали стойностите на елементите на един низ `a` присъстват като подниз в друг низ `b`?
7. Какво е особеното в този програмен фрагмент?

```
char *p;
p=strstr("abc", "abcd");
cout << p;
```

8. Напишете програма, която въвежда два низа и ги отпечатва, като между тях поставя един от знаците `=`, `<` или `>` в зависимост от лексикографската наредба помежду им.
9. Напишете програма, която въвежда знаков низ и определя дали той съдържа само български букви и дали тези букви са азбучно подредени.
10. Дадени са 3 низа. Напишете програма, която намира най-малкия (в лексикографската наредба) низ, който може да се получи при подходящо конкатениране на дадените 3 низа.
11. Напишете програма, която въвежда знаков низ и извършва следното:

    а) заменя всяка последователност `'abc'` от този низ с `'def'`;

    б) премахва всички знаци `'w'` в знаковия низ;

    в) заменя първата срещната буква `'x'` (ако има такава) с `'y'`;

    г) заменя всички малки латински букви с големи латински букви.
12. Дадени са два знакови низа. Напишете програма, която определя дали може от знаците на първия низ да се състави втория низ.
13. Даден е низ, съдържащ не по-малко от 6 знака. Напишете програма, която образува нов низ, състоящ се от първите 3 и последните 3 знака на дадения.

Глава 18. КОМПЮТЪРНА АРИТМЕТИКА

Целочислена аритметика. Както вече видяхме, често използван тип данни е типът на целите числа. Знаем, че в езика C/C++ една променлива `a` от този тип се декларира чрез `int a`. Такава променлива приема положителни и отрицателни цели стойности, както и нула: ..., `-2`, `-1`, `0`, `1`, `2`, ... За разлика от „математическото“ цяло число, което може да бъде произволно голямо, числата, съхранявани в целочислените променливи, не могат да надминат определена максимална стойност, както и не могат да бъдат по-малки от определено отрицателно число. Тези две *гранични стойности* зависят от конкретната реализация на компилатора, с който работите. Например при Borland (Turbo) C++ 3.1 максималната стойност е 32 767, а минималната отрицателна е −32768. При съставянето на програми трябва да се имат предвид тези ограничения, защото аритметични операции с резултат, излизащ извън споменатите граници, дават грешни стойности. Когато има нужда да се работи с по-големи числа, трябва да се използват различни други подходи. Например, когато е необходим по-голям диапазон от този, който осигурява типът `int`, може да се използва типът `long int`. Той не решава изцяло проблема, макар че осигурява интервал от −2 147 483 648 до +2 147 483 647. По-долу ще разгледаме друг метод за работа с големи (наричани още „дълги“) цели числа, при който цифрите на числото се записват като елементи на масив.

Като пример за ефекта, който се получава при излизане извън допустимите граници, може да разгледаме следния фрагмент:

```
int a=32767;
int b=a+1;
cout << b;
```

При Borland (Turbo) C++ 3.1 резултатът, който се отпечатва, е отрицателното число −32 768, т.е. не е това, което очакваме. Всъщност отпечатаната стойност −32 768 е минималната отрицателна стойност, която приема целочислената променлива `b`. Така целите числа от този тип като че ли са подредени в кръг – следващото след най-голямото е най-малкото! Споменатото свойство може да бъде използвано за предсказване на резултата при несложни операции, в които има събиране или изваждане. При по-сложни пресмятания обаче е трудно да предвидим какъв резултат ще се получи, когато се излиза от допустимите граници. Например следният фрагмент пресмята n факториел $n! = 1 \cdot 2 \cdot 3 \cdots n$, но за малки стойности на n:

```
int n;
cin >> n;
int p=1, i=1;
while(i<=n) {p *=i; i++;}
cout << p;
```

При въвеждане на числото 7 за стойност на n резултатът е правилен – стойността на p се отпечатва като 5040, което е равно на 7!. Правилен резултат се извежда и при стойности на n, равни на 1, 2, 3, 4, 5 и 6. Но при входна стойност $n = 8$ програмата работи неправилно при Borland (Turbo) C++ 3.1, защото $8! = 40320$, което е по-голямо от максималната стойност за типа `int`.

Числа с десетична точка. При изпълнението на програма е възможно някои от пресмятанията с числа от тип `double` (наричани също *реални числа*) да се извършат *приближено*. Това е характерна особеност за компютърните системи, защото числените стойности се съхраняват в паметта им в определен, не много голям брой елементарни „клетки“. Този брой понякога може да се окаже по-малък от необходимия за точно представяне на числото, което води да закръгляване и до други неточности. При компютърните пресмятания могат да бъдат наблюдавани и други отклонения от правилата на аритметиката, но тези особености няма да бъдат разглеждани тук.

Дълги цели числа. Дълги цели числа се наричат такива, които не се „вместват“ в целочислена променлива от стандартен тип. Например при определени обстоятелства дори едно шестцифрено цяло число може да бъде разглеждано като дълго, защото не е възможно да бъде съхранено в 16-битова променлива от тип `int` при някои компилатори. Оттук следва, че когато искаме да работим с такива числа, например да извършваме действие събиране с тях, трябва сами да програмираме съответните операции. Естествен подход е да записваме цифрите на едно дълго число в последователни елементи на масив.

Отделяне на цифрите в елементи на масив. В предишните глави разгледахме как се отделят цифрите на дадено цяло положително число, за да се отпечатат една след друга на екрана. Сега можем да ги запазим като последователни елементи в масив, с цел по-нататъшната им обработка. В следващия фрагмент се въвежда цяло положително число в променливата `b` и след това цифрите на това число се записват в елементите на масива `a[0], a[1], ..., [n]`, по ред от младшите към старшите разреди. Едновременно се пресмята и номерът `n` на най-старшия разред, като броят на цифрите е `n + 1`. Масивът `a` е определен да съдържа максимално 101 елемента, което, разбира се, може да се промени от програмиста (но то е напълно достатъчно за работа с променливата `b`, защото тя е от тип `int`):

```
int b;
int a[101];
int n=-1;
cin >> b;
do
{
 n++;
 a[n] = b%10;
 b=b/10;
}
while(b>0);
```

Получаване от знаков низ на цифрите на число. Понякога *дългите числа* са въвеждат като *низ от цифри*. Следващият фрагмент въвежда и обработва знаков низ `s`, за който се очаква, че елементите му са цифри. Те се пренасят в обратен ред в масива `a`, т.е. първият знак от низа става най-старшата цифра в масива `a` и т.н. В Глава 6 показахме, че ако `s[i]` е цифра, то `s[i]-'0'` е целочислена стойност, съответна на цифрата `s[i]`.

```
char s[101];
int a[101];
cin >> s;
int n=strlen(s)-1;
int i=n;
int j=0;
while(j<=n)
 {a[j]=s[i]-'0'; i--; j++;}
```

Съставяне на число по неговите цифри. За решаване на обратната задача – от цифрите на числото, съхранени като елементи на масива `a`, да получим самото число в променливата `b`, използваме познатата от Глава 9 схема на Хорнер в следния фрагмент:

```
// дадена е стойността на n
// и цифрите a[n], a[n-1], ..., a[1], a[0]
int b=0;
for(int i=n; i>=0; i--)
{
 b=10*b + a[i];
}
// стойността на числото е получена в променливата b.
```

Събиране на дълги числа. Нека `x` и `y` са два масива, съдържащи две дълги числа. Елементите с нулеви номера от тези масиви съдържат младшите цифри. Номерът на най-старшите цифри на двете дълги числа се пази съответно в променливите `nx` и `ny` (броят на цифрите е с единица по-голям от

стойностите в тези променливи, защото броенето на елементите на масивите започва от нула). Приемаме, че стойностите на `nx` и `ny` са равни. Ако това не е така, трябва елементите на по-късия масив, които са с номера, по-големи от минимума на `nx` и `ny`, да бъдат запълнени с нули. Следващият фрагмент пресмята в масива `z` сумата на двете дълги числа, като броят на цифрите на тази сума се записва в `nz`. Реализиран е обичайният „ръчен метод“ за събиране на две многоцифрени числа с „молив върху хартия“. Променливата `c` служи за запомняне на преноса към по-горен разред:

```
c=0;
for(i=0; i<=nx; i++)
{
 z[i]=x[i]+y[i]+c;
 if(z[i]>9)
  {z[i]=z[i]-10; c=1;}
 else c=0;
}
if(c){nz=nx+1;z[nz]=1;}
else nz=nx;
```

Подобна реализация е дадена в решението на конкурсна задача 1 към настоящата глава.

Други действия с дълги цели числа. Умножаване на дълго число с цифра е дадено в решението на конкурсна задача 3 в края на настоящата глава. Реализацията на останалите аритметични действия с дълги числа предлагаме в упражненията.

Получаване на запис на число с римски цифри. Вече имаме възможност да предложим завършена програма за това. Отделни нейни елементи разглеждахме в предишните глави. Функцията `r(c1,c2,c3)` служи за отпечатване с римски цифри на всяко число, принадлежащо на някое от следните три множества от числа: $\{1, 2, .., 9\}$, $\{10, 20, .., 90\}$, $\{100, 200, .., 900\}$. Това става, като стойностите на `c1,c2` и `c3` се заместват съответно с тройките римски цифри {I, V, X}, {X, L, C} и {C, D, M}. В главната функция `main()` се използва, че записът с римски цифри на числото `a` е конкатенация от записите на неговите единици, десетици, стотици и хиляди. Да отбележим, че най-голямото число, което може да се запише с римски цифри, е прието да бъде 3999.

```
#include<iostream.h>
void r(int v, char c1, char c2, char c3)
 {
  switch(v)
  {case 9 :
   case 4 : cout << c1;
  }
```

```
  switch(v)
  {case 9 : cout << c3; break;
   case 8 :
   case 7 :
   case 6 :
   case 5 :
   case 4 : cout << c2;
  }

  switch(v)
  {case 8 : cout << c1;
   case 7 : cout << c1;
   case 6 : cout << c1; break;
   case 3 : cout << c1;
   case 2 : cout << c1;
   case 1 : cout << c1;
  }
 }
void main()
 {int a;
  cin >> a;
  while(a>=1000){cout<<'M'; a -=1000;}
  if(a>=100)
   {r(a/100,'C','D','M'); a %=100;}
  if(a>=10)
   {r(a/10,'X','L','C'); a %=10;}
  r(a,'I','V','X');
 }
```

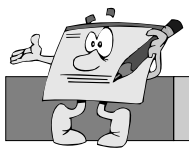
Решето на Ератостен. Определен кръг въпроси, свързани главно с теорията на числата, се решават с т. нар. *метод на решетото*. Исторически той е възникнал при търсене на прости числа (Ератостен е древногръцки математик, живял през 2. век пр. Хр.). Да припомним, че едно цяло положително число, по-голямо от 1, се нарича просто, ако то няма други делители, освен себе си и единицата.

Простите числа в интервала от 1 до n можем да „отсеем“ по следния начин: първо записваме последователно всичките числа от този интервал; след това, започвайки от 2, „задраскваме“ надясно всяко второ число, без самото число 2. На следващата стъпка търсим първото незадраскано число след 2. Това е 3 и сега „задраскваме“ вдясно от него всяко трето число. И въобще, на всяка следваща стъпка търсим първото незадраскано число k и задраскваме надясно от него всяко k-то поред. В древността вместо хартия за писане са

използвали друг материал, може би папирус, при който вместо да задраскват числата, са ги пробождали и така накрая се получавало решето.

В дадения по-долу фрагмент елементите на масива `r` първоначално се зареждат със стойност 1. След това в тялото на цикъла `do...while` най-напред се намира първият незадраскан елемент на масива `r[k]`, т.е. елемент със стойност 1, и след това се задраскват (т.е. нулират) всички следващи след него през стъпка `k`. Накрая се отпечатват номерата на елементите от масива, които са останали с първоначалната си стойност 1, т.е. извеждат се простите числа от 1 до `n`.

```
for(i=2; i<=n; i++) r[i]=1;
k=2;
do
{while ((r[k]==0)&&(k<n)) k++;
 j=k+k;
 while(j<=n){r[j]=0; j +=k;}
 k++;
}while(k<n);
for(i=2;i<=n;i++)
 if(r[i]) cout << i << " ";
```



УПРАЖНЕНИЯ

1. Какво ще отпечата следната програма? (Въпрос за средата Borland (Turbo) C++ 3.1.)

```
#include <iostream.h>
void main()
{
 int a=32767;
 int b=a+4;
 cout << b;
}
```

2. Как да променим дадената в настоящата глава програма за пресмятане на n факториел, за да може тя да пресмята вярно факториела на още няколко числа, по-големи от 7?

3. Обяснете връзките между следните понятия: числа от тип `double`, числа с десетична точка, реални числа, дробни числа.

4. Кои числа се наричат дълги цели числа?

5. Как дългите числа се представят с масив?

6. Обяснете по какви начини можем от цифрите на дадено число (съхранени като елементи на числов масив или на знаков масив) да получим самото число.

7. Напишете програма, която въвежда знаков низ и проверява дали той е правилен запис на число с римски цифри.

8. Две прости числа се наричат *прости числа-близнаци*, ако се различават с 2. Например 29 и 31. С помощта на решетото на Ератостен намерете колкото се може повече двойки от прости числа-близнаци.

9. *Хипотезата на Голдбах* гласи: Всяко четно число $n > 2$ може да се представи като сума на две прости числа. Проверете тази хипотезата за колкото се може по-големи стойности на n.

10. Задание за разработване: Напишете функции и програми, които извършват и прилагат аритметични действия с дълги числа: събиране, изваждане, умножение и деление.

КОНКУРСНИ ЗАДАЧИ

18.1. Сбор (Пролетен турнир, Пловдив, 2002, зад. D1). Напишете програма, която въвежда от клавиатурата един ред (с дължина най-много 80 знака), на който са написани две естествени числа, разделени със знака плюс (+), намира сбора на двете числа и извежда резултата на екрана.

Пример:

Вход	Изход
`2+2`	`4`
`12345678+87654321`	`99999999`
`999999999999999999+1`	`1000000000000000000`

Решение: Програмата реализира метода за преобразуване на дълго число от низ в масив и след това прилага идеята на дадения в настоящата глава фрагмент за събиране на две дълги числа.

```
#include<iostream.h>
#include<string.h>
char s[200];
int a[100], b[100], r[100];
```

```
void main()
{int c,i,j,n;
 cin >> s;
 n=strlen(s);

 i=n-1; j=0;
 while(s[i] != '+')
 { a[j]=s[i]-'0'; i--; j++;}

 i--; j=0;
 while(i>=0)
  {b[j]=s[i]-'0'; i--; j++;}

 c=0;
 for(i=0;i<99;i++)
 { r[i]=a[i]+b[i]+c;
   if(r[i]>9) {r[i]=r[i]-10; c=1;}
   else c=0;
 }
 i=99;
 while(r[i]==0) i--;
 for(j=i;j>=0;j--) cout << r[j];
 cout << "\n";}
}
```

18.2. Сума (Пролетен турнир, Ямбол, 2003, зад. D3). Дадени са две „големи“ числа – броят на цифрите им е между 2 и 100. Напишете програма, която намира сумата на двете числа.

Входните данни се четат от клавиатурата и са разположени на два реда – на всеки е записано по едно „голямо“ число. Резултатът се извежда на екрана на един ред.

Пример:

Вход	Изход
98453472324234701 8735897349879827349827 3	873590719522705977329 74

Упътване: Виж решението на предишната конкурсна задача.

18.3. Умножение (Национална олимпиада, областен кръг, 2002, зад. D1). Напишете програма, която въвежда от клавиатурата едно естествено число, което може да има до 50 цифри, след това въвежда друго естествено число, което е едноцифрено и отпечатва на екрана произведението на двете въведени числа.

Пример: Въвеждаме 123456 и 7. Програмата трябва да отпечата 864192.

Решение: Преобразуваме въведения низ от цифри в масив от числа `a`. След това извършваме умножението на елементите на този масив с едноцифреното число `b` и резултатът получаваме в масива `r`. Накрая елементите на този масив се отпечатват.

```
#include <iostream.h>
#include <string.h>
char s[100];
int a[100], r[100];
int b;

void main()
{ cin >> s; cin >> b;
  int n=strlen(s)-1;
  int i=n,j=0;
  while(j<=n)
   {a[j]=s[i]-'0'; i--; j++;}

  int c=0,w;
  for(i=0;i<=n;i++)
   { w = b*a[i]+c;
     r[i]= w%10;
     c=w/10;
   }
  r[n+1]=c;

  if(r[n+1]>0) cout << r[n+1];
  for(i=n;i>=0;i--) cout << r[i];
  cout << "\n";
}
```

18.4. Сортиране. (Зимни празници, Варна, 2003, зад. D2). Дадено е цяло положително число $N(1 < N < 20)$ и последователност от N различни цели положителни числа, не по-големи от 1000. Напишете програма, която подрежда числата в нарастващ ред според сбора на цифрите им. Ако две числа имат равен сбор от цифрите си, по-напред да бъде числото, което е по-малко.

Входните данни трябва да се четат от клавиатурата, а полученият резултат да се изведе на екрана. Първият ред на входа ще съдържа числото N. Вторият – N-те числа, разделени с по един интервал. Изходът трябва да съдържа числата, разделени с по един интервал и подредени според изискванията.

Пример:

Вход	Изход
5 203 189 41 900 666	41 203 900 189 666

Решение: Прочитаме числата в масива `a` и едновременно пресмятаме в съответните елементи на масива `c` броя на цифрите за всяко от числата. След това прилагаме метода на мехурчето за сортиране, като разместваме тези елементи, за които наредбата на съседните стойности в масива `c` не е необходимата, а когато тези съседни стойности в масива `c` са равни, проверяваме наредбата на съседните стойности в масива `a`.

```
#include<iostream.h>
int a[21],c[21];
int n;

void main()
{ cin >> n;
  int i,j;
  for(i=1;i<=n; i++)
   { cin >> a[i];
     c[i]=0;
     int w=a[i];
     while(w>0)
     { c[i] += (w%10);
       w /= 10;}
   }
  int b;
  do
     { b=0;
       for(i=1;i<n;i++)
       if((c[i]>c[i+1])||
          ((c[i]==c[i+1])&&(a[i]>a[i+1])))
          { int w;
            w=a[i]; a[i]=a[i+1]; a[i+1]=w;
            w=c[i]; c[i]=c[i+1]; c[i+1]=w;
            b=1;
          }
      } while(b);

  for(i=1;i<=n;i++) cout << a[i] << " ";
  cout << "\n";
}
```

Глава 19. ДВУМЕРНИ МАСИВИ

Масивите, които разглеждахме досега, могат да се сравнят с проста таблица, състояща се от един ред данни. Например след изпълнение на оператора

```
for(int i=0; i<=10; i++) a[i]=i*i;
```

елементите на масива `a[i]` ще бъдат равни на последователно разположените данни в таблицата:

номер	0	1	2	3	4	5	6	7	8	9	10
данни	0	1	4	9	16	25	36	49	64	81	100

Естествено обобщение на този вид прости таблици е таблицата, състояща се от много редове. Да разгледаме например дадената по-долу таблица, в която всяка от данните е разположена на мястото, където се пресича определен ред с определен стълб. Всъщност, дадена е таблицата за умножение, в която на мястото, където редът с номер i се пресича със стълба с номер j, стои произведението $i \cdot j$:

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$	$j=10$
$i=0$	0	0	0	0	0	0	0	0	0	0	0
$i=1$	0	1	2	3	4	5	6	7	8	9	10
$i=2$	0	2	4	6	8	10	12	14	16	18	20
$i=3$	0	3	6	9	12	15	18	21	24	27	30
$i=4$	0	4	8	12	16	20	24	28	32	36	40
$i=5$	0	5	10	15	20	25	30	35	40	45	50
$i=6$	0	6	12	18	24	30	36	42	48	54	60
$i=7$	0	7	14	21	28	35	42	49	56	63	70
$i=8$	0	8	16	24	32	40	48	56	64	72	80
$i=9$	0	9	18	27	36	45	54	63	72	81	90
$i=10$	0	10	20	30	40	50	60	70	80	90	100

Всяка клетка от тази таблица се определя с два индекса i и j, за разлика от първата по-проста таблица, където клетките се определят само с един индекс – техния номер. *Двумерна* таблица се нарича таблица, в която клет-

ките се определят с два индекса, а по-простият вид таблици, при които за определянето на клетките се използва един индекс, се наричат *едномерни*.

Както всяка едномерна таблица съответства на познатия ни тип масив (наричан понякога *едномерен масив*), така и за всяка двумерна таблица е предвиден по-сложен вид на масива в езика C/C++, наречен *двумерен масив*.

Деклариране на двумерен масив. Двумерният масив се обявява (декларира) в програмата по начин, сходен с декларирането на едномерен масив. Използват се две двойки от отваряща и затваряща квадратни скоби, между които се записва броят на елементи на масива по всяка от размерностите. Например следната конструкция определя двумерния масив, съответстващ на дадената по-горе двумерна таблица:

```
int a[11][11];
```

Стойностите на елементите на този масив са от целочисления тип `int`, а всеки от индексите на масива може да приема стойност от 0 до 10. Не е задължително двата индекса да имат една и съща максимална стойност. Например може да определим масив

```
int b[6][11];
```

В този случай масивът съответства на правоъгълна таблица с 6 реда и 11 стълба; номерата на редовете пробягват числата 0, 1, 2, ..., 5, а номерата на стълбовете – числата 0, 1, 2, ..., 10. Въобще *първият* индекс съответства на *номера на реда*, а *вторият* – на *номера на стълба* в таблицата.

Освен с базов тип `int`, двумерните масиви могат да бъдат дефинирани и с други базови типове, например `double` или `char`, както е при едномерните масиви.

Правилата, с които се ползват двумерните масиви, са сходни с правилата за работа при едномерни масиви. Например всеки елемент на двумерния масив може да бъде присвояван като стойност на променлива от същия базов тип `b=a[i][j]` и обратното, на всеки елемент от масива може да се присвоява стойност `a[i][j]=b` при задължителното изискване и в двата случая стойностите на двата индекса i и j да са в границите на изменението им, зададено при определянето на масива.

Задаване на стойности. Следният фрагмент от програма показва зареждането на масива `a[i][j]` със стойности, съответстващи на таблицата за умножение:

```
for(int i=0;i<11;i++)
for(int j=0;j<11;j++)
  a[i][j]=i*j;
```

Да припомним, че съгласно стандартите, реализирани в повечето компилатори на C/C++, всеки масив от числа, обявен като *глобална* променлива

(т.е. обявен извън коя да е функция), трябва да бъде зареден автоматично с нулеви стойности, без това да бъде явно програмирано.

Стойностите на елементите в двумерния масив могат да се въвеждат и чрез клавиатурата. За целта трябва да решим в каква последователност ще се извършва това – по редове или по стълбове. По-често се използва *въвеждане по редове*. Това означава, че най-напред се прочитат елементите от първия (нулевия) ред, след това – елементите от следващия ред и т.н., например:

```
const int M=5; const int N=3;
int a[M][N];
for(int i=0;i<M;i++)
for(int j=0;j<N;j++)
 cin >> a[i][j];
```

Когато броят на елементите в масива не е голям, възможно е да ги заредим с конкретни стойности на мястото, където масивът се обявява. Това става чрез дописване на знака за равенство „=“, следван от последователните стойности на елементите *по редове*, разделени със запетайки и заградени с фигурни скоби така, както е показано в следния пример:

```
int a[3][5] =
     {{11,12,13,14,15},
      {16,17,18,19,20},
      {21,22,23,24,25}};
```

Извеждане. Отпечатването на стойностите на декларирания по-горе масив във вид, съответстващ на таблица, може да се осъществи от следния фрагмент:

```
for(int i=0;i<3;i++)
 { for(int j=0;j<5;j++)
     cout << a[i][j] << " ";
   cout << "\n";
 }
```

Сума от елементите на двумерен масив. Осъществява се чрез двоен цикъл (тук и по-долу в примерите ще предполагаме, че разглежданият масив е деклариран чрез `int a[M][N]`, където `M` и `N` са константи):

```
int s=0;
for(int i=0;i<M;i++)
for(int j=0;j<N;j++)
 s += a[i][j];
```

Сумиране по редове и стълбове. Нека да разгледаме два допълнителни едномерни масива `x` и `y`, първият с толкова елемента, колкото са стълбовете в двумерния масив `a`, а вторият с толкова елемента, колкото са редовете в същия двумерен масив, т.е. нека масивите `x` и `y` са декларирани така:

```
int x[N]; int y[M];
```

Понеже всеки стълб от двумерния масив `a` съответства на елемент от едномерния масив `x`, може да поставим задачата: Във всеки елемент на масива `x` да пресметнем сумата от елементите, намиращи се в съответния стълб на `a`. За целта използваме следния програмен фрагмент:

```
for(int j=0;j<N;j++)
{ int s=0;
  for(int i=0;i<M;i++) s += a[i][j];
  x[j]=s;
}
```

Чрез външния цикъл се пробягват номерата `j` на стълбовете и за всеки `j`-ти стълб се извършва сумиране на елементите му. Това става чрез вътрешния цикъл, където индексът `i` пробягва последователно по редове елементите от взетия `j`-ти стълб.

Разгледаното действие се нарича *сумиране по стълбове*. Аналогично се програмира *сумиране по редове*:

```
for(int i=0;i<M;i++)
{ int s=0;
  for(int j=0;j<N;j++) s += a[i][j];
  y[i]=s;
}
```

Търсене. Нека в променливата `v` е дадена определена стойност, за която искаме да знаем дали се среща измежду стойностите в двумерния масив. Задачата решаваме чрез обхождане с двоен цикъл на всички елементи в масива.

```
int i0=-1, j0=-1;
for(int i=0;i<M;i++)
for(int j=0;j<N;j++)
 if(a[i][j]==v) {i0=i; j0=j;}
```

Завършване със стойност, равна на `-1` за коя да е от променливите `i0` или `j0`, означава, че стойността на `v` не е намерена измежду елементите на масива. В противен случай `i0` и `j0` дават индексите на елемента в масива `a`, където стойността на `v` е открита за последен път в процеса на търсенето.

Седлова точка. *Седлова точка* ще наричаме такъв елемент на двумерния масив, който е най-голям в реда си и най-малък в стълба си (по-голям или равен на всеки друг елемент от същия ред и едновременно с това по-малък или равен на всеки друг елемент от същия стълб). Не при всички въведени данни за стойности на елементите в двумерния масив съществува седлова точка. Например в таблицата

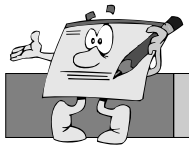
```
1  9  1
4  3  2
7  8  9
```

няма седлова точка, докато в следващата таблица елементът със стойност `5` е седлова точка:

```
1  6  3
2  5  4
3  7  6
```

В следващия програмен фрагмент се извършва обхождане на елементите на двумерния масив и за всеки негов елемент `c=a[i][j]` се прави проверка чрез двата следващи цикъла с брояча `k` дали стойността на `c` е най-голямата в реда си и най-малката в стълба си. Променливата `b` служи за флаг; ако `b==1` след тези проверки, намерена е седлова точка – отпечатват се нейният номер на ред и стълб и стойността ѝ.

```
for(int i=0;i<M;i++)
for(int j=0;j<N;j++)
 { int c=a[i][j];
   int b=1,k;
   for(k=0;k<N;k++)
    if(k != j)
    if(c<a[i][k]) b=0;
   for(k=0;k<M;k++)
    if(k != i)
    if(c>a[k][j]) b=0;
   if(b)
   {cout << i << " " << j << " ";
    cout << a[i][j] << "\n";}
 }
```



УПРАЖНЕНИЯ

1. Проверете възможна ли е декларацията:

```
int a[5][6] =
     {{11,12},
      {13},
      {14,15},
      {16,17,18,19,20},
      {21,22,23,24}};
```

2. Еквивалентни ли са декларациите:

```
int a[4][3] = {{11},{16},{21},{26}};
```

и

```
int a[4][3] = {{11,0,0},{16,0,0},{21,0,0},{26,0,0}};
```

3. Напишете функции за въвеждане и извеждане стойностите на елементите на двумерен масив.

4. Напишете програма, която първо отпечатва елементите на двумерен масив по редове, след това по стълбове.

5. Напишете програма, която въвежда елементите на двумерен масив по стълбове, а ги отпечатва по редове.

6. Напишете програма, която въвежда елементите на двумерен масив, като отпечатва подканващи съобщения за номера на реда и номера на стълба на елемента, който предстои да бъде въведен от потребителя на програмата.

7. Напишете програма, която намира най-големия и най-малкия елемент в даден двумерен масив.

8. Променете фрагмента за намиране на седлова точка, така че той да отпечатва най-големия елемент във всеки ред на дадения двумерен масив.

9. За даден двумерен масив напишете програма, която зарежда в два едномерни масива съответно най-голямата стойност от всеки ред и най-голямата стойност от всеки стълб на двумерния масив.

10. Дадени са числото v и един двумерен масив от числа. Напишете функция, която отпечатва двойките индекси на всички елементи от масива, чиято стойност е равна на v.

11. Дадени са числото v и един двумерен масив от числа. Напишете функция, която връща стойност 0, ако даденото число v се среща измежду стойностите на елементите на масива и 1 в противен случай.

12. Дадени са числото v и един двумерен масив от числа. Напишете функция, която отпечатва номерата на стълбовете в масива, където има стойност, равна на v.

13. Напишете програма, която намира и извежда номерата на онези редове (стълбове) на двумерен масив, в които има поне един отрицателен елемент.

14. Заредете и отпечатайте двумерен масив с таблицата от остатъците при деление `i%j`, където променливите `i` и `j` пробягват стойностите от 1 до 9.

15. Дадена е таблица от m реда и n стълба, в клетките на която са записани дробни числа. Получете нова таблица чрез деление на числата с най-голямото по-абсолютна стойност число от първоначалната таблица.

16. Даден е двумерен масив от числа с равен брой редове и стълбове. Отпечатайте числата, намиращи се на двата диагонала.

17. За двумерен масив от числа намерете най-голямата сума, получена при сумирането на елементите в отделните редове.

18. В двумерен масив от числа увеличете всичките числа с единица.

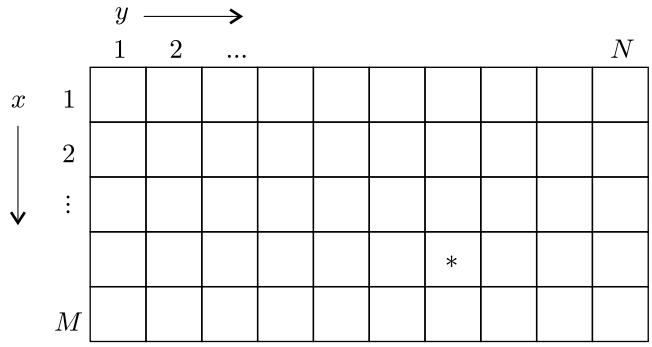
19. В двумерен масив от числа сменете всички отрицателни числа с нули.

20. Разменете два реда в двумерен масив.

21. Една квадратна таблица от числа се нарича *магически квадрат*, когато е изпълнено следното условие: всички суми, получени по всеки от редовете, по всеки от стълбовете и по всеки от двата диагонала са равни помежду си. Да се напише програма, която въвежда числата от дадена квадратна таблица и определя дали те образуват магически квадрат.

Глава 20. ПОЛЕ ОТ КВАДРАТНИ КЛЕТКИ

В много приложения се разглежда поле, съставено от еднакви квадратчета (клетки), подредени в M реда и N стълба. По-долу е даден пример за такова поле, с 5 реда и 10 стълба, при което със звездичка е отбелязана клетката от 4-тия ред и 7-мия стълб:



Номерирането на редовете обикновено се извършва отгоре-надолу, а на стълбовете – отляво надясно. Разглежданото поле представяме с двумерен масив, в който броят на редовете и стълбовете е с единица по-голям от дадените размери M и N.

```
int a[M+1][N+1];
```

Така в програмите ще използваме, че индексите се променят съответно от 1 до `M` и от 1 до `N`, като елементите на масива, при които някой от индексите е 0, може да се ползват за други цели. Обикновено празните клетки се означават с нулева стойност за съответния елемент от масива, а „пълните“ клетки (или тези, които са отбелязани, например със звездичка) – със стойност 1. За разгледания пример трябва да направим присвояването `a[4][7]=1`. Останалите елементи на масива автоматично ще бъдат заредени с 0, ако масивът е обявен като глобален.

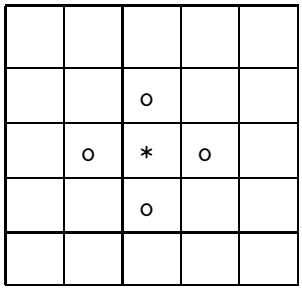
Координати на клетка. Номерът на реда x и номерът на стълба y, в които се намира една клетка, се наричат нейни *координати*. Когато дадена клетка има координати x и y, записваме ги като двойка, заградена от кръгли скоби, и казваме, че разглеждаме клетката (x, y).

Съседни клетки. За дадена клетка с координати x и y може да разгледаме две определения за съседни клетки:

Определение 1. Съседи, с които дадената клетка има обща стена. В общия случай това са 4 клетки, чиито координати са следните двойки числа

$$(x+1, y), \quad (x-1, y), \quad (x, y+1), \quad (x, y-1).$$

По реда на написването им тези двойки дават координатите на клетка, която е съответно *отдолу*, *отгоре*, *надясно* и *наляво* от дадената. На схемата по-долу с кръгчета са означени съседите на отбелязаната със звездичка клетка:



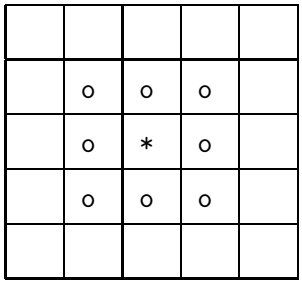
Когато дадената клетка (x, y) е непосредствено до края на полето (това означава, че е изпълнено поне едно от следните 4 условия: или $x = 1$, или $x = M$, или $y = 1$, или $y = N$), тя може да има по-малко от 4 съседа в смисъла на разглежданото определение.

Определение 2. Съседи, с които дадената клетка има обща стена или общ връх. В общия случай това са 8 клетки, чиито координати са следните двойки числа:

$$(x+1, y), \quad (x+1, y+1), \quad (x, y+1), \quad (x-1, y+1),$$
$$(x-1, y), \quad (x-1, y-1), \quad (x, y-1), \quad (x+1, y-1).$$

По реда на написването им тези двойки дават координатите на клетките, които се получават при обхождане около дадената клетка по посока, обратна на часовниковата стрелка, като се започне от клетката, намираща се *отдолу*.

На схемата по-долу с кръгчета са означени съседите на отбелязаната със звездичка клетка:



Отново е в сила наблюдението, че когато дадената клетка (x, y) е непосредствено до края на полето (това означава, че е изпълнено поне едно от следните 4 условия: или $x = 1$, или $x = M$, или $y = 1$, или $y = N$), тя може да има по-малко от 8 съседа в смисъла на разглежданото определение.

Представяне на координатите на съседни клетки. Удобно е да пресмятаме координатите на съседните клетки чрез два масива `dx` и `dy`, в които са записани техните *отмествания* спрямо дадена клетка, съответно по x и по y. Ако разглеждаме първото определение за съседство, определяме в програма двата масива по следния начин:

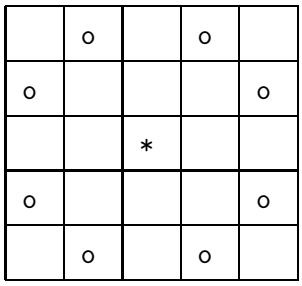
```
int dx[4]={ 1,-1, 0, 0};
int dy[4]={ 0, 0, 1,-1};
```

Когато целочислената променлива `t` пробягва стойностите от 0 до 3, координатите на всичките съседи са измежду двойките `(x+dx[t], y+dy[t])`.

Следващата програма отпечатва със знака „`o`“ местата, където са съседите на клетката с координати `x` и `y`. Местоположението на тази клетка се отпечатва със звездичка „`*`“, а на останалите (празните) клетки с точка „`.`“. Приемаме, че стойностите за `x` и `y` се въвеждат от клавиатурата като коректни данни. Програмата прави проверка кои от съседите са в границите на полето. Дадената клетка се отбелязва в масива `a` чрез стойност 1, а съседите ѝ – чрез 2.

```
#include<iostream.h>
const int M=5;
const int N=10;
int a[M+1][N+1];
int dx[4]={ 1,-1, 0, 0};
int dy[4]={ 0, 0, 1,-1};
void main()
{
  int x,y,t;
  cin >> x >> y;
  a[x][y]=1;
  for(t=0;t<4;t++)
  {
   int x0=x+dx[t]; int y0=y+dy[t];
   if((0<x0)&&(x0<=M)&&(0<y0)&&(y0<=N))
    a[x0][y0]=2;
  }
  for(int i=1;i<=M;i++)
  { for(int j=1;j<=N;j++)
    if(a[i][j]==0) cout << ".";
    else if(a[i][j]==1) cout << "*";
    else cout << "o";
    cout << "\n";
  }
}
```

Шахматен кон. Съгласно правилата на шахматната игра, ако поставим кон в клетката, означена със звездичка, той може да отиде в следните, означени с кръгчета клетки:



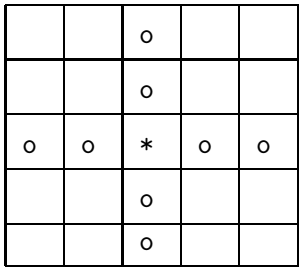
Тогава, ако координатите на означената със звездичка клетка са (x, y), то координатите на означените с кръгчета са:

$$(x+1, y+2), \quad (x+2, y+1), \quad (x+2, y-1), \quad (x+1, y-2),$$
$$(x-1, y-2), \quad (x-2, y-1), \quad (x-2, y+1), \quad (x-1, y+2).$$

Масивите за отместването в този случай се дефинират по следния начин:

```
int dx[8]={1, 2, 2, 1,-1,-2,-2,-1};
int dy[8]={2, 1,-1,-2,-2,-1, 1, 2};
```

Шахматен топ. Той се движи навсякъде в реда си или в стълба си:



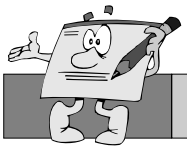
Ако началата позиция на топа е (x, y), клетките, до които той може да стигне, се изразяват с координатите $(x+t, y)$, $(x-t, y)$, $(x, y+t)$, $(x, y-t)$, където променливата t пробягва целите стойности от 1 нататък, но дотолкова, че получените координати да не напуснат полето. Следващият фрагмент зарежда със стойност 2 елементите на масива, съответстващи на клетките, в които може да отиде шахматният топ:

```
t=1; while(x+t<=M) {a[x+t][y]=2; t++;}
t=1; while(x-t>0)  {a[x-t][y]=2; t++;}
t=1; while(y+t<=N) {a[x][y+t]=2; t++;}
t=1; while(y-t>0)  {a[x][y-t]=2; t++;}
```

Шахматен офицер. Той се движи по двата диагонала, които излизат от неговата позиция:

o				o
	o		o	
		*		
	o		o	
o				o

Ако началата позиция на офицера е (x, y), клетките, до които той може да стигне, се изразяват с координатите $(x+t, y+t)$, $(x+t, y-t)$, $(x-t, y+t)$, $(x-t, y-t)$, където променливата t пробягва целите стойности от 1 нататък, но дотолкова, че получените координати да не напуснат полето.



УПРАЖНЕНИЯ

1. Какво трябва да променим в програмата, която отпечатва със знака „o“ местата, където са съседите, *с които дадената клетка има обща стена*, за да се отпечатват и местата, където са съседите, *с които дадената клетка има общ връх*?

2. Напишете програма, която за поле от квадратни клетки от M реда и N стълба отпечатва по подходящ начин всичките клетки, до които може да достигне с един ход шахматен цар, поставен върху клетката с координати (x, y).

3. Видоизменете предишната задача за всяка от шахматните фигури: топ, офицер, кон, дама.

4. Робот се движи в безкрайно поле от квадратни клетки. Във всяка секунда от времето, той се премества надясно с една позиция с изключение на момента, когато настъпва всяка пета секунда. Тогава роботът скача 3 позиции нагоре. Освен това, като добавка към описаното придвижване, роботът всяка 13-та секунда скача 3 позиции наляво, а всяка 17-та – 4 позиции надолу. Напишете програма, която отпечатва координатите на робота за всяка въведена секунда. Началното му положение е $(0, 0)$.

Упътване: Даденото поле в задачата е безкрайно, т.е. не може да бъде представено с масив. Освен това, за разлика от разглежданията в настоящата глава, сега е възможно координатите на клетките да бъдат и отрицателни числа. Такива ще се получат, ако например в началния момент роботът тръгне нагоре или наляво.

За да напишем програма, използваме променливите `x` и `y`, в които ще запазваме текущите координати на робота. Моментът от време `T`, в кой-

то трябва да се отпечата положението му, се въвежда от клавиатурата. Следващият фрагмент пресмята това положение:

```
int x=0, y=0, T;
cin >> T;
for(t=1;t<=T;t++)
{ if(t%5==0) x -=3; else y +=1;
  if(t%13==0) y -=3;
  if(t%17==0) x +=4;
}
cout << x << " " << y << "\n";
```

5. Движението на робот в безкрайно поле от квадратни клетки се управлява от програма. Програмата представлява редица от двойки цели числа (положителни, отрицателни или нули), които задават какво отместване трябва да направи роботът във всяка секунда от времето. Първото число в двойката трябва да бъде добавено към стойността на координатата x, а второто – към стойността на координатата y, за да се получат координатите на робота (x, y) в следващата секунда. Програмата, управляваща робота, завършва с последна двойка числа, равна на $(0, 0)$. Напишете програма, която въвежда програмата за управление на робота (редица от четен брой цели числа, разделени с по една шпация) и отпечатва координатите, в които ще се намира роботът в края на движението му. Началното положение на робота е в клетката $(0, 0)$.

Упътване: В цикъл прочитаме двойки числа `a` и `b`, докато намерим двойката $(0, 0)$. В тялото на цикъла променяме текущите координати x и y на робота:

```
int x=0,y=0,a,b;
cin >> a >> b;
while((a!=0)&&(b!=0))
 {x+=a; y+=b; cin >> a >> b;}
cout << x << " " << y << "\n";
```

КОНКУРСНИ ЗАДАЧИ

20.1. Шахматен кон (Есенен турнир, Шумен, 2003, зад. D2). Дадени са два символа – латинска буква (от a до h) и цифра (от 1 до 8), например d4. Нека те се разглеждат като координати на поле от шахматна дъска, на която

е разположен кон. Да се отпечата шахматната дъска, като с H се бележи позицията на коня, с X се отбележат полетата, които „бие“ шахматният кон и с O – останалите полета. Напишете програма, която въвежда двата символа, определящи позицията на шахматния кон, и извежда на екрана именуваната шахматна дъска по указания начин.

Пример: Вход: `d4`

Изход:

```
8 O O O O O O O O
7 O O O O O O O O
6 O O X O X O O O
5 O X O O O X O O
4 O O O H O O O O
3 O X O O O X O O
2 O O X O X O O O
1 O O O O O O O O
  a b c d e f g h
```

Упътване: Изменете дадената в настоящата глава програма за отпечатване съседите на една клетка, като промените масивите `dx` и `dy`.

20.2. Спирала (Национална олимпиада, областен кръг, 2003, зад. D2). Робот обикаля полетата на правоъгълна дъска с m реда и n стълба. Полетата имат координати: числата от 1 до n по хоризонталата отляво надясно и от 1 до m по вертикалата отгоре надолу.

Обиколката прилича на навиваща се спирала, като всяко поле се посещава точно веднъж. Тя започва от горния ляв ъгъл (координати $(1, 1)$) с движение надясно, после надолу, после наляво, нагоре и отново по същия начин. Във всяка посока движението продължава, докато е възможно, т.е. до достигане на ръба на дъската или на вече посетено поле. Роботът спира, щом посети всички полета.

Например при $m = 4$ и $n = 3$ обиколката изглежда както на следната фигура, където последователно посетените полета са отбелязани със знака `O` и са свързани с линия. Началото на обиколката започва от горния ляв ъгъл:

```
O - O - O
        |
O - O   O
|   |   |
O   O   O
|       |
O - O - O
```

Да се напише програма, която намира координатите на последното посетено от робота поле на дъската. Числата m и n ($1 \leqq m \leqq 1000$, $1 \leqq n \leqq 1000$)

се въвеждат от един ред на стандартния вход, а резултатът се извежда на стандартния изход, както е показано в примера.

Пример: Вход `4 3` Изход `(3,2)`

Забележка: В условието на тази задача думата „поле“ има различен смисъл от употребата ѝ в настоящата глава. Това, което в задачата е наречено „поле“, в текста на настоящата глава се нарича „клетка“.

Решение: Поради големия размер на максималните стойности на m и n за тази задача не е подходящо да използваме масив за представяне на полето от клетки. В променливите `x` и `y` се записват текущите координати на робота. В програмата е по-удобно да моделираме движението от втората посетена точка и затова началните стойности са `x=1; y=n;`. Променливата `c` служи за указване на посоката, в която тръгваме всеки път. Понеже съществуват 4 възможности за това, в `switch(c%4)` се има предвид стойността на остатъка при делене на `c` с 4. Отместванията при движението във всяка от посоките се дават от `dx` и `dy`. Първоначално е зададено, че `dx=m-1` и `dy=n-1`, но след това стойностите на тези отмествания се намаляват с 1 по подходящ начин. В началото на тялото на цикъла `while(1)` се проверяват условията за край, за да се излезе с `break`.

```
#include<iostream.h>
int c,x,y,m,n,dx,dy;

void main()
{ cin >> m >> n;
  dx=m-1; dy=n-1;
  x=1; y=n;
  c=1;
  while(1)
  {
   if(c%2==0)if(dy==0) break;
   if(c%2==1)if(dx==0) break;

   switch(c%4)
   {case 0: y += dy; dy--; break;
    case 1: x += dx; dx--; break;
    case 2: y -= dy; dy--; break;
    case 3: x -= dx; dx--;
   }
   c++;
  }
  cout << "(" << x << "," << y << ")\n";
}
```

Глава 21. МАСИВ ОТ НИЗОВЕ

При въвеждане на редица от думи е удобно да се работи с *масив от низове*. Той е двумерен масив с базов тип `char`. Например декларацията

```
char s[20][100];
```

ни осигурява 20 низа: `s[0]`, `s[1]`, ..., `s[19]`. Всеки от тези низове има максимална дължина от по 100 знака (всъщност 99 видими знака, защото последният знак като правило трябва да бъде `'\0'`, за да маркира края на актуалното съдържане на низа, а то може да бъде с по-малка дължина от максималната).

Всеки отделен елемент `s[i]` може да се употребява по същия начин, както се употребява проста променлива за низ. Но сега имаме предимството, че `s[i]` удобно могат да се използват в оператори за цикъл.

Достъпът до `j`-тия елемент на низа `s[i]` се извършва, както е обичайно за двумерни масиви – чрез използване на означението `s[i][j]`.

Думи. *Дума* ще наричаме *знаков низ, който не съдържа празни интервали* (*шпации*). В това определение за понятието дума не се изисква думата да е непременно смислена редица от букви или да е взета от някакъв речник. Обикновено знаците в една дума са видимите знаци от таблицата на ASCII-кодове.

Въвеждане на думи от клавиатурата. Всяко обръщение към оператора `cin >> s`, където `s` е низ, води до прочитане на една дума от стандартния вход (клавиатурата). За да прочетем последователно няколко думи, прилагаме този оператор в цикъл. В дадената по-долу програма излизането от цикъла `while(1)` става при достигане края на потока от въвежданите данни. За целта е използван поместеният в тялото на цикъла оператор `if(cin.eof()) break;`.

Функцията `cin.eof()` приема стойност 1 в момента, когато се достигне *краят на входния поток* от данни; в останалите случаи нейната стойност е 0. При въвеждане от клавиатурата сигналът, който трябва да подадем, за да съобщим края на въвеждането, е натискане на клавишната комбинация Ctrl-z.

В дадената по-долу програма всяка дума се прочита в спомагателната променлива от тип низ `s` и след това се копира чрез стандартната функция `strcpy()` в `k`-тия низ на масива от низове `a`.

```
#include<iostream.h>
#include<string.h>
char a[100][100],s[100];
```

```
void main()
{int k=-1;
 while(1)
 { s[0]=0;
   cin >> s;
   if(s[0] !=0)
    {k++; strcpy(a[k],s);}
   if(cin.eof()) break;
 }
 cout << "\n\nk=" << k << "\n";
 for(int j=0;j<=k;j++) cout << a[j] << "\n";
}
```

Забележка: Обработването на сигнала от клавиатурата за край на входния поток се извършва по различни начини в зависимост от средата за програмиране, с която е създаден изпълнимия код на горната програма. Например при стандартните настройки след компилиране с Borland C++ 3.1 се получава програма, която изисква след Ctrl-z да се натисне Enter; при Dev-C++ 4.0 е достатъчно само Ctrl-z, а при djgpp 2.95.3 клавишната комбинация Ctrl-z трябва да се въведе, когато курсорът е в началото на нов ред на екрана.

Отделяне на думи от низ. Когато е даден низ, в който има празни интервали, може да поставим задачата за отделяне на думите, съдържащи се в него. По-точно казано, ще искаме да запишем като елементи на масив от низове тези поднизове от дадения низ, всеки от които не съдържа нито един знак за празен интервал и освен това, всеки такъв подниз е отделен отляво и отдясно с поне един празен интервал (с изключение на случаите, когато поднизът е краен). Например, ако е даден низът

`" abcd ef 123 1a 2b c",`

той съдържа 6 думи: `"abcd"`, `"ef"`, `"123"`, `"1a"`, `"2b"` и `"c"`.

Ще използваме стандартната функция `cin.getline()`, която се включва в програмата чрез използваното досега `#include<iostream.h>`. Ако `s` е низ с достатъчна максимална дължина `n`, функцията `cin.getline(s,n);` *въвежда един цял ред* от стандартния вход и го записва в низа `s`.

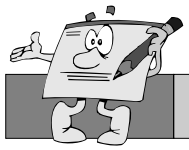
В следващата програма е определена функцията `parse(a)`, чрез която се въвежда един ред от стандартния вход и след това се отделят думите, съдържащи се в него. Тези думи се присвояват последователно на елементите в масива от низове `a[j]`. Броят на отделените думи е с единица повече от стойността, която се получава накрая в променливата `k`. Стойността на `k` се връща като стойност на самата функция `parse(a)`. В главната функция `main()`е показано как може да се отпечатат тези думи по една на ред. Константите `max_W` и `max_L` служат съответно за определяне на максималния брой думи, които могат да се поместят в масива `a` и за определяне на максималната дължина на всяка от думите.

```
#include<iostream.h>
#include<string.h>
const int max_W=100;
const int max_L=100;

int parse(char a[max_W][max_L])
{char s[max_W*max_L];
 cin.getline(s,max_W*max_L);

 int i,j,k=-1;
 char p=' ';
 for(j=0;j<strlen(s);j++)
 {
  if(!((s[j] >= 0)&&(s[j] <= ' ')))
   { if(p==' ') {i=0; k++;}
     a[k][i]=s[j]; i++;
     a[k][i]=0;
   }
  p=s[j];
 }
 return k;
}

char a[max_W][max_L];

void main()
{
 int k=parse(a);
 cout << "\nk=" << k << "\n";
 for(int j=0;j<=k;j++) cout << a[j] << "\n";
}
```

Лексикографско сортиране на масив от низове. Разполагайки с низове, записани в масив, можем да поставим задачата за сортиране на низовете от този масив в лексикографски ред. За целта ще използваме някой от познатите ни алгоритми за сортиране на масив от числа, но за пресмятане на резултата от сравненията сега вместо оператори със знаците „<“ или „>“, ще приложим стандартната функция `strcmp()`, изучена в Глава 17.

В следващия фрагмент е използван алгоритъмът за сортиране по метода на мехурчето за масива `a[0], a[1], ..., a[N-1]`:

```
char a[N][M];
char t[M];
int ok, i;
do
{ok=1;
 for(i=0;i<N-1;i++)
  if(strcmp(a[i],a[i+1])>0)
   {strcpy(t,a[i]);
    strcpy(a[i],a[i+1]);
    strcpy(a[i+1],t);
    ok=0;
   }
}
while(!ok);
```



УПРАЖНЕНИЯ

1. Даден е масив от n низа, представящи цели положителни числа, всяко с не повече от 6 цифри. Намерете средно аритметичното на тези числа.

2. Да се напише програма, която сортира във възходящ ред елементите на редица от низове, представящи имена, не по-дълги от 9 знака. Сортираната редица да се изведе по 5 думи на ред.

3. Напишете програма, която прочита текст от клавиатурата, състоящ се от предварително зададен брой редове, и извежда на екрана тези редове, подредени от последния към първия.

4. Напишете програма, която прочита текст от клавиатурата, състоящ се от предварително зададен брой редове и извежда:

а) най-дългия от редовете;

б) двата най-къси реда, слепени като един ред – първо най-късия и непосредствено до него следващия по дължина;

в) реда, съдържащ най-много знаци, различни от празните интервали;

г) реда, съдържащ най-много главни латински букви;

д) всичките редове, но подредени в намаляващ ред на дължините им.

5. Напишете програма, която прочита текст от клавиатурата, състоящ се от предварително зададен брой редове, и извежда на екрана тези редове, подредени лексикографски.

6. Напишете програма, която прочита текст от клавиатурата, състоящ се от един ред, и извежда на екрана най-дългата дума, съдържаща се в него.

7. Напишете програма, която прочита текст от клавиатурата, състоящ се от един ред, и извежда на екрана думите, съдържащи се в него, така че:

а) да са подредени по дължина, във възходящ ред, по една дума на екранен ред;

б) да са подредени по лексикографска наредба, в низходящ ред, по една дума на екранен ред.

в) да са подредени по брой на гласните букви, във възходящ ред, като всичките думи се изведат на един ред, отделени една от друга с шпации.

8. Напишете програма, която прочита произволен текст от клавиатурата (за край се използва Ctrl-z) и отпечатва броя на думите и броя на буквите от латинската азбука в него.

КОНКУРСНИ ЗАДАЧИ

21.1. Най-дълга дума (Национална олимпиада, общински кръг, 2002, зад. D3). Да се напише програма, която въвежда от клавиатурата няколко думи, написани на един ред, разделени с поне един интервал. След това програмата намира и извежда най-дългата дума от въведените. Ако има няколко най-дълги думи, извежда ги всичките.

Примери:

Вход: tova e primer
Резултат: primer

Вход: ima mnogo dobri deca
Резултат: mnogo dobri

Решение: Използваме разгледаната в настоящата глава функция `parse(a)`, за да отделим думите като елементи в масив от низове `a[j]`. След това в променливата `m` намираме най-голямата от дължините на думите и с последния цикъл отпечатваме всички думи с тази дължина.

```
#include<iostream.h>
#include<string.h>

const int max_L=100;
const int max_W=100;
```

```
int parse(char a[max_W][max_L])
{
 char s[max_W*max_L];
 cin.getline(s,max_W*max_L);

 int i,j,k=-1;
 char p=' ';
 for(j=0;j<strlen(s);j++)
 {
  if(!((s[j] >= 0)&&(s[j] <= ' ')))
   { if(p==' ') {i=0; k++;}
     a[k][i]=s[j]; i++;
     a[k][i]=0;
   }
  p=s[j];
 }
 return k;
}

char a[max_W][max_L];

void main()
{
 int k=parse(a);

 int j,m=0;
 for(j=0;j<=k;j++)
  if(strlen(a[j])>m) m=strlen(a[j]);

 for(j=0;j<=k;j++)
  if(strlen(a[j])==m) cout << a[j] << "\n";

}
```

21.2. Различни думи (Национална олимпиада, областен кръг, 2002, зад. D3). Напишете програма, която въвежда от клавиатурата естествено число N ($N < 100$) и след това N на брой думи. Програмата трябва да изведе на екрана броя M на думите, които не се повтарят, както и самите думи.

Примери:

Вход	Резултат	Вход	Резултат
6	4	8	1
nebe	nebe	program	disk
more	voda	computer	
voda	gora	computer	
gora	reka	program	
more		computer	
reka		disk	
		program	
		computer	

Решение: След като думите се въведат като елементи на масив от низове `a[i]`, променливата `M` се нулира и в следващия двоен цикъл се проверява дали думата с номер `i` е еднаква с някоя от останалите думи с номера `j`. Стойността на променливата `M` се увеличава с единица, само когато `i`-тата дума не е била еднаква с нито една от останалите. Така в крайна сметка в `M` се получава броят на неповтарящите се думи. След като `M` се отпечата, самите неповтарящи се думи се търсят и отпечатват по същата схема, чрез която е получена стойността на `M`.

```
#include<iostream.h>
#include<string.h>

const int max_L=100;
const int max_W=100;

char a[max_W][max_L];

void main()
{
 int i,j, M, N;
 cin >> N;
 for(i=0;i<N;i++) cin >> a[i];
 M=0;
 for(i=0;i<N;i++)
 { int b=0;
   for(j=0;j<N;j++)
   if(i != j)
    if(!strcmp(a[i],a[j])) b=1;
   if(!b) M++;
 }
```

```
 cout << M << "\n";
 for(i=0;i<N;i++)
 { int b=0;
   for(j=0;j<N;j++)
    if(i != j)
     if(!strcmp(a[i],a[j])) b=1;
   if(!b) cout << a[i] << "\n";
 }
}
```

21.3. Най-голямо число (Зимни празници, Плевен, 2002, зад. D2). Да се напише програма, която въвежда от клавиатурата 5 естествени числа (написани на един ред) и извежда на екрана най-голямото число, което може да се получи, когато тези числа се напишат едно след друго в някаква последователност.

Примери:

Вход:	`13 25 9 10 90`	`11111 2222 333 44 5`
Резултат:	`990251310`	`544333222211111`

Решение: Естествените числа, които се въвеждат от клавиатурата, се записват като елементи на масив от низове `a[i]`. В програмата е приложен методът за пораждане на пермутациите на числата 1, 2, 3, 4, 5, описан в Глава 15. Като многоцифрен брояч е използван масивът `p`. При всяка получена пермутация елементите `p[j]` на този масив се използват като индекси за елементите на масива от низове: `a[p[j]]`. В получената, благодарение на всяка една пермутация, подредба, тези низове се конкатенират в низа `s`. От своя страна, низът `s` се сравнява лексикографски с низа `m`, който служи за текуща най-голяма стойност. Накрая в `m` се получава търсеното естествено число във вид на низ.

```
#include<iostream.h>
#include<string.h>

char a[5][100], s[500], m[500];
int p[5];

int eq()
{
  for(int i=0;i<5;i++)
   for(int j=i+1; j<5; j++)
   if(p[i]==p[j]) return 1;
  return 0;
}
```

```
void main()
{int c, i, j;
 for(i=0;i<5;i++) cin >> a[i];
 m[0]=0;
 do
 {
  c=1;
  for(i=0;i<5;i++)
  { p[i] += c;
    if(p[i]>4){p[i]=0; c=1;}
    else c=0;
  }

  if(!eq())
   { s[0]=0;
     for(j=0;j<5;j++) strcat(s,a[p[j]]);
     if(strcmp(m,s)<0) strcpy(m,s);
   }
 }
 while(c==0);
 cout << m << "\n";
}
```

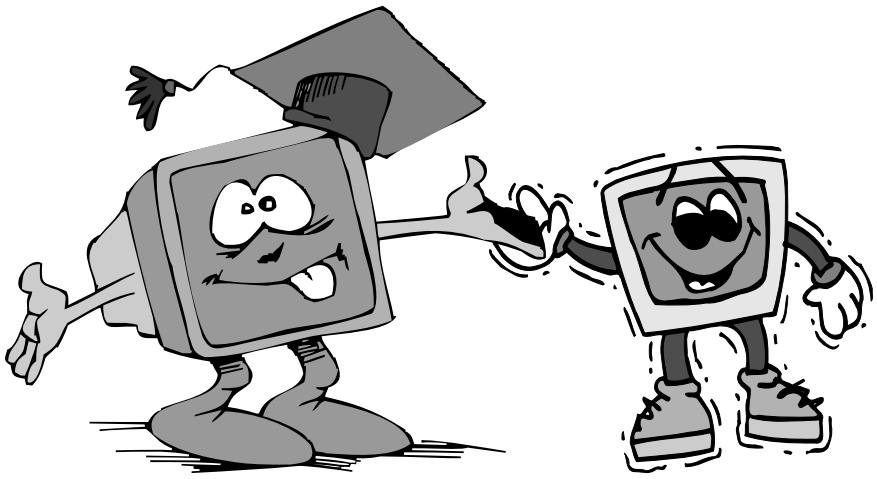
21.4. Думи (Есенен турнир, Шумен, 2003, зад. D1). От клавиатурата се въвежда текст, който е дълъг най-много 80 символа. Дума в текста се нарича последователност от символи, в която не се среща символът интервал. Думите в текста са разделени с един интервал. Да се състави програма, която въвежда текста от клавиатурата и го извежда на екрана, като в него са разменени най-дългата с най-късата дума. Ако има повече от една най-къса и/или повече от една най-дълга дума, да се разменят последната най-къса с първата най-дълга дума. Ако всички думи са равни, текстът не се променя.

Пример 1: Вход: Петър учи много прилежно
Изход: Петър прилежно много учи

Пример 2: Вход: Петър получава винаги много шестици
Изход: Петър много винаги получава шестици

Упътване: Въведете думите от текста като елементи на масив от низове `a[i]`, `i = 0, 1, 2, ..., N-1` и след това намерете най-малкото число `m` и най-голямото число `M` измежду числата `strlen(a[i])`. Ако `m` и `M` не са равни, намерете дума с дължина `M`, тръгвайки да търсите от `i = 0`, като увеличавате `i` и намерете дума с дължина `m`, тръгвайки да търсите от `i = N-1`, като намалявате `i`. Накрая разменете в масива така намерените две думи и отпечатайте елементите на масива, като ги разделите с по една шпация.

ПРИЛОЖЕНИЯ



ПРИЛОЖЕНИЕ 1

Най-важното за работната среда по време на състезание

По време на състезание на всеки участник се предоставя персонален компютър, обикновено с операционната система MS Windows в една от версиите 98, 2000 или XP. Организаторите имат задължението да осигурят необходимата правилна инсталация на софтуера за разработване на състезателните програми.

DOS-прозорец. За да работите, трябва да влезете в DOS-прозорец:

– Един възможен начин за създаване на този прозорец е първо да изберете бутона, намиращ се най-долу вляво на екрана (с надпис „Start“). След като кликнете с мишката върху този бутон, избирате „Run“ от появилото се вертикално меню. След това в малкото текстово прозорче въвеждате от клавиатурата „command“ или „cmd“ и натискате клавиша Enter.

– Друг начин да създадете DOS-прозорец е да кликнете с мишката върху неговата икона на десктопа (ако такава икона е инсталирана там).

Появява се DOS-прозорец. Според предварителните настройки този прозорец може да заеме целия екран или да е същински прозорец. Ако мониторът не е с големи размери, повечето състезатели предпочитат да работят в *режим на цял екран*. За да преминете в режим на цял екран, кликвате с десния бутон на мишката върху горния ляв ъгъл от рамката на DOS-прозореца

и от падащото меню избирате „properties“. Там, евентуално след преминаване върху таба „Screen“, кликвате с мишката върху малкото кръгче с надпис „Full-screen“. След още едно-две кликвания върху бутони „OK“ вече сте в режим цял екран. Сега, ако искате да се върнете към основния екран на операционната система MS Windows, трябва да използвате клавишната комбинация Alt-Tab. Обратно влизате в режим цял екран чрез кликване с мишката върху иконата на DOS-прозореца, която се появява на най-долната лента на екрана.

Друга възможност за превключване от режим „прозорец“ в режим „цял екран“ и обратно е предоставена чрез клавишната комбинация Alt-Enter.

Стартиране на средата за работа. Ще разгледаме случая, когато тази среда е **Borland (Turbo) C++ 3.1**. Работата с останалите среди е твърде сходна и затова няма да ги разглеждаме.

Първо трябва да преминете в *директорията*, която е определена за вашата работа. Ако приемем, че нейното име е `konkurs` и е разположена в главната директория на устройство `C:`, вие трябва да напишете чрез клавиатурата следните команди в DOS, като след всяка натискате клавиша Enter:

```
>C:
>cd \konkurs
```

Знакът „`>`“ се изписва от системата. Нарича се *промптер*. Обикновено отляво на този знак системата изписва текущата директория. Така ще видите:

```
C:\konkurs>
```

За да започнете да разработвате програмата за избрана от вас задача, трябва да стартирате средата за работа чрез въвеждане на името ѝ като команда в DOS, последвано от празен интервал и името на задачата. В нашия случай името на средата за работа най-често е `bc` (понякога `tc`), а ако приемем, че името на програмата, която трябва да направите, е `proba.exe` (точното име трябва да вземете от условието на задачата), въвеждате следното:

```
C:\konkurs>bc proba.cpp
```

(Обърнете внимание на окончанието `.cpp`, което е задължително.)

Появява се основният екран на работната среда. Сега може да въвеждате текста на програмата (предполага се, че щом сте стигнали дотук, все пак знаете с какво да започнете програмата си).

Редактиране. Въвеждането на букви, цифри и други знаци се извършва по познатия начин чрез клавиатурата. За да въведете някои специални знаци (например #), трябва да натиснете и клавиша Shift.

За да поправите забелязана грешка, използвайте клавишните стрелки за придвижване на курсора по екрана или клавиша Backspace – той изтрива текста, върху който минава. Един или няколко знака може да изтриете с

клавиша Del. За да вмъкнете текст, трябва да сте в режим Insert. Този режим е включен в началото, но ако сте излезли от него, може да се върнете, като натиснете едноименния клавиш.

За работа с *големи части (блокове)* от текста се използват следните средства:

– *Маркиране на блок*: поставяте курсора в началото на избран от вас блок и натискате Ctrl-K B; след това поставяте курсора в края на блока и натискате Ctrl-K K. Избраният блок се отбелязва като по-светъл на екрана. След като блокът вече е маркиран, с него може да правите следното:

– *Копиране на блок*: поставяте курсора на мястото, където искате да се появи копие на блока и натискате Ctrl-K C.

– *Преместване на блок*: поставяте курсора на мястото, където искате да се премести блокът и натискате Ctrl-K V.

– *Изтриване на блок*: натискате Ctrl-K Y.

Помощна информация (Help). Много неща може да научите или да си припомните, като четете помощната информация. Един начин да направите това е чрез натискане на клавишната комбинация Alt-H.

Менюта. Натискането на клавиша F10 ви пренася на горната лента, която е главният ред на *менюто.* Там може да се движите с клавишните стрелки и с клавиша Enter да влизате в различни подменюта.

Запазване на работата. Когато работите, натискайте отвреме-навреме клавиша F2.

Създаване на изпълним файл. Извършва се с клавиша F9. Ако имате грешки, които се улавят от компилатора, системата ще ви даде съобщения за тях и вие трябва да се върнете, за да ги поправите. Ако компилирането и свързването преминат успешно, следващата стъпка от ваша страна трябва да бъде стартирането на изпълнението на програмата. Това може да направите с Ctrl-F9, без да напускате средата за работа. Сега може да въведете входни данни и да видите какво се извежда.

За да приложите другия начин за проверка на програмата след успешно компилиране, трябва да излезете от средата за разработка. Този начин ще използва журито. Запазете програмата с F2 и натиснете клавишната комбинация Alt-X. Сега виждате промтера на DOS. Въведената от вас команда

```
>dir
```

ще изпише на екрана списъка на файлове от текущата директория. Между имената им трябва да видите имената `PROBA.CPP` и `PROBA.EXE` (или вместо `PROBA.*`, ще видите името, което сте дали на програмата си). За да изпълните програмата, въведете името ѝ като команда (може и с малки букви):

```
>proba
```

Тествайте я, като въведете съставени от вас примерни входни данни. Ако резултатът ви удовлетворява, може да приключите работата си върху тази задача, в противен случай се върнете в работната среда, за да поправите програмата. За целта отново въведете

```
>bc proba.cpp
```

„Увисване“. Понякога, при допусната грешка в програмата, е възможно след стартирането ѝ тя да престане да реагира на въведени данни от клавиатурата и вие да не можете да се върнете в работната среда. Тогава опитайте да я прекъснете аварийно, като използвате клавишните комбинации Ctrl-C или Ctrl-Break или се опитайте да затворите DOS-прозореца, като първо натиснете Alt-Tab и след това кликнете с десния бутон на мишката върху иконата му на долната лента и изберете Close. Ако и това не помогне, може би остава да изключите и включите отново компютъра. Ето защо има смисъл да запазвате работата си.

Записване върху дискета. Поставете в устройството (A:) предварително приготвената дискета и когато сте в DOS-прозореца, въведете командите за копиране на двата файла върху дискетата:

```
>copy proba.exe a:
>copy proba.cpp a:
```

За да се уверите, че наистина сте записали програмата си, вижте съдържанието на дискетата чрез командата

```
>dir a:
```

Да ви напомним, че вашата програма трябва да е написана така, че *да извежда на екрана само* (*и точно*) *това, което се изисква в условието на задачата.* Всякакви допълнителни съобщения или даже отпечатване на празни редове и допълнителни интервали, които в други случаи може да украсят изхода ви, сега ще бъдат приети от журито за грешка.

ПРИЛОЖЕНИЕ 2

Таблица на ASCII-кодове

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	□
128	Ђ	Ѓ	‚	ѓ	„	…	†	‡	€	‰	Љ	‹	Њ	Ќ	Ћ	Џ
144	ђ	‘	’	“	”	•	–	—	□	™	љ	›	њ	ќ	ћ	џ
160		Ў	ў	Ј	¤	Ґ	¦	§	Ё	©	Є	«	¬	-	®	Ї
176	°	±	І	і	ґ	µ	¶	·	ё	№	є	»	ј	Ѕ	ѕ	ї
192	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
208	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
224	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
240	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Таблицата е получена с използване на шрифт *Courier* в текстообработващата програма *MS Word 97* при установяване на *language Bulgarian.* Показани са само видимите знаци с номера от 32 до 255.

Номерът на всеки знак се получава, като се съберат числата, стоящи отляво на реда и отгоре в стълба на избрания знак. Например българската буква Б има номер в таблицата, равен на 192 +1 = 193.

Забележка: Мястото на буквите от кирилицата в тази таблица съответства на един от приетите стандарти за MS Windows. За да получите местоположението им при най-често срещания стандарт в MS DOS, четирите реда с кирилица, започващи от реда с номер 192 в тази таблица, трябва да преместите така, че да започват от реда с номер 128.

ПРИЛОЖЕНИЕ 3

Списък на някои от използваните в книгата английски думи

английска дума	**приблизително произношение**	**възможно значение на употреба в книгата**	**глави, в които са дадени пояснения**
break	брейк	прекъсни	6, 9, 12
case	кейс	случай	6
char (character)	чар *или* кер (к**е**ръктър)	знак	6, 9, 16, 17
cin	си ин	въведи	1
cout	си аут	изведи	1
default	д*е*ф**о**лт	по подразбиране	6
do	д**у**	направи	8
double	д**а**бъл	двойна точност	1, 18
else	елс	иначе	3
even	**и**вън	четно число	6
for	фор	за	7
found	ф**а***у*нд	намерен	12
get	гет	вземи	1
hello	хел**о***у*	здравей	1
if	иф	ако	3
include	инкл**у***у*д	включи	1
int (integer)	инт (инт*е*дж**ъ**р)	целочислен	1, 18
iostream	ай оу стрийм	входно-изходен поток	1
long	лонг	дълъг	4, 18
main	мейн	главен	1
odd	од	нечетно число	6
prime	прайм	просто число	11
random	р**е**нд*о*м	случаен	12
return	р*е*т**ъ**рн	върни	1, 10
source	со*у*рс	източник	1
string	стринг	низ	16, 17, 21
switch	с*у*ич	превключвател	6
void	войд	празен	1, 10
while	*у*айл	докато	8
world	*у*ърлд	свят	1

ПРИЛОЖЕНИЕ 4

Списък на задачи от проведени състезания, дадени с подробни решения в книгата

4.1. Тухла (Пролетен турнир, Пловдив, 2002, зад. D2)
8.1. Звезди (Есенен турнир, Шумен, 2001, зад. D1)
8.2. Триъгълници (Есенен турнир, Шумен, 2002, зад. D1)
8.3. Минимакс (Пролетен турнир, Ямбол, 2003, зад. D2)
9.1. Автопарк (Есенен турнир, Шумен, 2003, зад. D3)
9.2. Числа (Есенен турнир, Шумен, 2001, зад. D3)
9.3. Различни начини (Зимни празници, Плевен, 2002, зад. D1)
9.4. Сладкиши (Национална олимпиада, областен кръг, 2003, зад. D3)
9.5. Линийки (Есенен турнир, Шумен, 2002, зад. D2)
9.6. Търговия (Национална олимпиада, областен кръг, 2003, зад. D1)
9.7. Намери две (Пролетен турнир, Пловдив, 2002, зад. D3)
9.8. Равни стойности (Есенен турнир, Шумен, 2001, зад. D2)
10.1. Дати (Зимни празници, Плевен, 2002, зад. D3)
11.1. Прости множители (Национална олимпиада, общински кръг, 2002, зад. D1)
12.1. Хотел (Зимни празници, Варна, 2003, зад. D1)
12.2. Улица (Пролетен турнир, Ямбол, 2003, зад. D1)
15.1. Разместване (Национална олимпиада, общински кръг, 2002, зад. D2)
16.1. Хистограма (Зимни празници, Варна, 2003, зад. D3)
16.2. Единствен елемент (Есенен турнир, Шумен, 2002, зад. D3)
16.3. Низ (Пролетен турнир, Ямбол, 2001, зад. D1)
16.4. Най-ляв (Пролетен турнир, Ямбол, 2001, зад. D2)
16.5. Кръстосване (Национална олимпиада, областен кръг, 2002, зад. D2)
18.1. Сбор (Пролетен турнир, Пловдив, 2002, зад. D1)
18.2. Сума (Пролетен турнир, Ямбол, 2003, зад. D3)
18.3. Умножение (Национална олимпиада, областен кръг, 2002, зад. D1)
18.4. Сортиране (Зимни празници, Варна, 2003, зад. D2)
20.1. Шахматен кон (Есенен турнир, Шумен, 2003, зад. D2)
20.2. Спирала (Национална олимпиада, областен кръг, 2003, зад. D2)
21.1. Най-дълга дума (Национална олимпиада, общински кръг, 2002, зад. D3)
21.2. Различни думи (Национална олимпиада, областен кръг, 2002, зад. D3)
21.3. Най-голямо число (Зимни празници, Плевен, 2002, зад. D2)
21.4. Думи (Есенен турнир, Шумен, 2003, зад. D1)

Авторите на публикуваните в книгата конкурсни задачи са членове и сътрудници на Екипа за извънкласна работа по информатика при Съюза на математиците в България: Стоян Капралов, Красимир Манев, Емил Келеведжиев, Бойко Банчев, Виолета Матанска, Антон Шиков, Теодоси Теодосиев, Антония Йовчева, Пламенка Христова и др.

ПРИЛОЖЕНИЕ 5

Петдесет конкурсни задачи

1. Делимост на 11 (Национална олимпиада, общински кръг, 2004, зад. D1). Напишете програма, която въвежда две трицифрени числа a и b и намира всички трицифрени числа в интервала $[a, b]$, на които като се задраска първата цифра и се допише най-накрая, образуваното число се дели на 11.

Входните данни се четат от стандартния вход като две трицифрени положителни числа, написани на един ред и отделени с интервал ($a < b$).

Изходните данни трябва да са записани на отделни редове в стандартния изход.

Пример:

Вход: `304 400`

Изход:
```
314
325
336
347
358
369
380
391
```

2. Половинки (Национална олимпиада, общински кръг, 2004, зад. D2). Да се напише програма, която въвежда на стандартния вход низ, състоящ се от четен брой знаци с дължина N ($2 \leqq N \leqq 80$) и разменя местата на двете му половини. Резултатът да се изведе на стандартния изход.

Пример:

Вход: `aSdfgHjk`

Изход: `gHjkaSdf`

3. Намаляващи числа (Национална олимпиада, общински кръг, 2004, зад. D3). Напишете програма, която въвежда от стандартния вход числото N ($1 \leqq N \leqq 9$) и извежда на стандартния изход: на първия ред числата от 1 до N , на втория – числата от 2 до N и т.н. на N-я ред числото N.

Пример:

Вход: `5`

Изход:
```
12345
2345
345
45
5
```

4. Умножение (Зимни празници, Русе, 2004, зад. D1). Известен е следният древноруски алгоритъм за умножение на две цели числа a и b. Умножаваме числото a по 2, а числото b делим на 2 (ако делимото е нечетно число, вземаме цялата част на частното). Резултатите от изчисленията записваме в две колонки, както е посочено в примера. Повтаряме същите действия, докато b стане единица. За получаване на произведението на a и b събираме всички числа от първата колонка, до които във втората колонка има нечетни числа.

Пример:

$a = 63$, $b = 42$

63	42
126	21
252	10
504	5
1008	2
2016	1

$$a \cdot b = 126 + 504 + 2016 = 2646$$

Напишете програма `RUSKI.EXE`, която умножава две цели числа по древноруския алгоритъм. Входните данни се въвеждат от клавиатурата като две цели положителни числа a и b ($a \leqq 100$ и $b \leqq 100$). Изходните данни трябва да се изведат на два реда на екрана. На първия ред да бъдат числата, които събираме, за да получим произведението, разделени с по един интервал. На втория ред – полученото произведение.

Примери:

Вход	Вход
63 42	7 11
Изход	Изход
126 504 2016	7 14 56
2646	77

5. Симпатична последователност (Зимни празници, Русе, 2004, зад. D2). Професор Х се забавлява като съставя интересни числови последователности. Последното му творение има следния вид:

$$1,\ 121,\ 1213121,\ 121312141213121,\ \ldots$$

Първият член на тази последователност е 1. Всеки от следващите членове се получава като се запише предходният член, до него да се запише номерът на текущия член, след което отново се записва предходният член.

Тъй като дължината на членовете на последователността нараства много бързо и ръчното им изписване е доста трудоемко, професорът се нуждае от програма, която да отпечатва зададен член на тази последователност.

Помогнете му като напишете програма `NICE.EXE`, която по зададено цяло положително число K ($K < 15$) извежда k-тия член на последователността.

Пример:

Вход: `5` Изход: `1213121412131215121312141213121`

6. Думи (Зимни празници, Русе, 2004, зад. D3). Група от N ученика ($N \leqq 30$) трябва да подреди M думи ($M \leqq 20$) по определен критерий. Думите съдържат само малки букви и са с дължина, не по-голяма от 10 букви. Всеки ученик предлага на учителя свой вариант на подредбата. Правилната подредба на думите е известна на учителя. Учителят поставя на всеки ученик оценка от 0 до M, която съответства на броя на думите, поставени от ученика на правилните им места.

Напишете програма `WORDS.EXE`, която пресмята оценките на всеки ученик и намира максималната получена оценка, броя и номерата на учениците, получили тази максимална оценка.

Входът се състои от $N+2$ реда. На първия ред е разположен символен низ, задаващ правилната подредба на думите. Думите в него са разделени с един интервал. На втория ред се задава едно цяло число N – броя на учениците. Следващите N реда съдържат символни низове с подредбите на всеки от учениците.

На първия ред на изхода се извежда максималната оценка, получена от учениците. На втория ред се извеждат номерата на учениците, получили тази оценка, разделени с по един интервал.

Пример:

Вход:	Изход:
`коза кон котка крава куче`	`3`
`5`	`1 2 5`
`кон коза котка крава куче`	
`коза котка кон крава куче`	
`котка коза кон крава куче`	
`коза котка кон куче крава`	
`котка кон коза крава куче`	

7. Игра I (Национална олимпиада, областен кръг, 2004, зад. D1). Събираме си купчинка от N мидички на плажа и играем следната игра:

Ако имаме само една мидичка – играта спира. Ако мидичките могат да се разпределят в три равни по брой купчинки – вземаме едната от купчинките, а всички останали мидички хвърляме в морето. Ако не могат да се разделят на три, но пък могат да се разделят на две равни купчинки – вземаме едната от купчинките, а останалите мидички хвърляме в морето. Ако не могат да се разделят нито по първия, нито по втория начин – намираме на плажа още една мидичка и я добавяме към нашата купчинка. С получената купчинка започваме всичко отначало – казваме, че започваме нов ход в играта.

Накрая винаги ще ни остане една мидичка и играта ще свърши. Интересуваме се колко хода най-много може да продължи играта, в зависимост от големината на началната купчинка.

Ето някои примери:

– ако в началото имаме 1 мидичка – никакъв ход не може да се направи (играта продължава 0 хода);

– ако в началото имаме 3 мидички, играта ще свърши след един ход – ще разделим на три купчинки от по една мидичка, ще изхвърлим двете и ще ни остане една мидичка;

– при две мидички пак ще имаме само един ход до края;

– ако в началото мидичките са 13, ще извършим 6 хода, като броят на мидичките в нашата купчинка ще се променя така: $13 \to 14 \to 7 \to 8 \to 4 \to 2 \to 1$;

– ако имаме 17 мидички в началото, играта ще свърши само след 4 хода: $17 \to 18 \to 6 \to 2 \to 1$.

Напишете програма `GAME.EXE`, която намира за колко хода ще завърши играта при даден начален брой мидички.

Пример:

Вход: `13` Изход: `6`

8. Футболни системи (Национална олимпиада, областен кръг, 2004, зад. D2). Един футболен отбор се състои от 11 играча – вратар и 10 полеви играчи. Разпределението на полевите играчи на защитници (D), полузащитници (M) и нападатели (F) се нарича система във футбола. Например при системата 4-4-2 отборът се състои от 4 защитници, 4 полузащитници и 2 нападатели.

По време на тренировките често се налага да се изпробват различни футболни системи. Треньорите биха искали бързо и лесно да получат всички възможни системи при фиксиран брой на играчите от една линия. Смята се, че няма смисъл да се поставят повече от 6 играчи на една линия. Също така не е добре да се оставя цяла линия празна. Това означава, че D, F и M са цели числа от 1 до 6 включително.

Напишете програма `DMF.EXE`, която пресмята всички възможни системи при зададен брой на играчите от една линия.

Вход: Три цели числа, D, F и M, разделени с по един интервал. Стойност 0 за някое от тях означава, че броят на съответните играчи е неизвестен.

Изход: На първия ред се извежда едно цяло число N – брой на възможните системи. На следващите N реда се извеждат самите системи – три числа на ред, разделени с по един интервал.

Пример:

Вход: `0 4 0` Изход:
```
5
145
244
343
442
541
```

9. Топове (Национална олимпиада, областен кръг, 2004, зад. D3). Върху различни полета на шахматна дъска са поставени 3 топа. Положението на

всеки топ се задава с буква и цифра, като буквата определя стълба (a, b, c, d, e, f, g, h), а цифрата – реда на дъската (1, 2, 3, 4, 5, 6, 7, 8), където се намира топа.

Напишете програма `ROOKS.EXE`, която прочита от клавиатурата три двойки от написани една до друга буква и цифра, чрез които се задава положението на топовете. Самите двойки се въвеждат на един ред и са разделени с по един интервал. Програмата трябва да изведе броя на празните полета, които не се бият от нито един топ. Според правилата на шахматната игра един топ бие всички полета, които се намират на неговия ред или неговия стълб.

Пример:

Вход: `a1 b2 c3` Изход: `25`

10. Триъгълници (Пролетен турнир, Пловдив, 2004, зад. D1). В тази задача ще разглеждаме текстове, оформени като „равностранни“ триъгълници, съставени от нули и единици, като първите ще наричаме 0-триъгълници, а вторите – 1-триъгълници. Всеки триъгълник има дължина на страната, определена от някакво цяло положително число N, по-голямо или равно на три и не по-голямо от 20. Първият ред на триъгълника се състои от една единствена цифра (0 или 1, в зависимост от вида му), вторият ред – от две цифри от съответния вид, разделени с един интервал, който е точно под цифрата от първия ред. Третият ред се състои от три цифри от съответния вид, разделени с по един интервал, като интервалите са точно под цифрите от втория ред и т.н., а последният ред е съставен от N цифри от съответния вид, разделени с по един интервал. 0-триъгълниците се изобразяват изправени – първият ред (с една цифра) отгоре, а последния ред (с N цифри отдолу), докато 1-триъгълниците се изобразяват обърнати – редът с една цифра е отдолу, а този с N цифри – отгоре. Задачата е да направите програма `TREE.EXE`, която по зададени цели положителни N и M отпечатва следната конструкция: два 0-триъгълника с дължина на страните N са долепени един до друг в основите си (страните с по N цифри са една до друга, разделени с един интервал), а 1-триъгълник със страна M е спуснат в образувалото се между двата 0-триъгълника пространство (виж примерите).

Програмата трябва да прочете числата N и M от клавиатурата (стандартния вход), а да изведе резултата на екрана (стандартния изход), като най-лявата цифра на конструкцията трябва да бъде в първа позиция на съответния ред на екрана.

Примери:

Вход:
```
5 7
```
Изход:
```
    1 1 1 1 1 1 1
     1 1 1 1 1 1
      1 1 1 1 1
     0 1 1 1 1 0
    0 0 1 1 1 0 0
   0 0 0 1 1 0 0 0
  0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0
```

Вход:
```
5 3
```
Изход:
```
      0           0
     0 0 1 1 1 0 0
    0 0 0 1 1 0 0 0
   0 0 0 0 1 0 0 0 0
  0 0 0 0 0 0 0 0 0 0
```

11. Артур (Пролетен турнир, Пловдив, 2004, зад. D2). Рицарите на крал Артур са седнали на кръглата маса. Пред всеки има купчинка, съдържаща няколко ореха. По команда на краля, всеки рицар, едновременно с останалите, взема половината от орехите пред себе си и ги слага пред стоящия отляво (ако броят на орехите е нечетен, взема толкова орехи, че броят им да е равен на цялата част на числото, изразяващо половината). Кралят повтаря многократно същата команда, като желае да настъпи момент, когато всичките купчинки от орехи ще съдържат по равен брой.

Напишете програма `ARTUR.EXE`, която въвежда броя на рицарите и началния брой на орехите пред всеки от тях, зададени при обхождане на масата по посока на часовниковата стрелка. Тези числа се въвеждат от един ред на стандартния вход и са разделени с по един интервал. Броят на рицарите не е по-голям от 20, а общият брой на орехите не надминава 1000.

Програмата трябва да изведе на стандартния изход най-малкия брой команди, след които се е изпълнило желанието на краля. Ако желанието на краля не може да се изпълни, програмата трябва да изведе 0. Кралят може да даде най-много 10 000 команди.

Примери:

Вход
```
5 2 1 3 1 18
```
Изход
```
12
```

Вход
```
5 2 1 3 1 17
```
Изход
```
0
```

12. Дроби (Пролетен турнир, Пловдив, 2004, зад. D3). Напишете програма `FRAC.EXE`, която събира обикновени дроби.

От един ред на стандартния вход се въвеждат 4 естествени числа, разделени с по един интервал. Тези числа са по-малки от 1000 и представляват, съответно, числителя и знаменателя на едната и на другата дроб. Въвежданите дроби не са непременно правилни, нито пък несъкратими.

Програмата трябва да изведе на стандартния изход сбора на двете дроби.

Ако сборът е цяло число, се извежда само това число. Ако сборът е число по-малко от 1, се извеждат числителят и знаменателят на резултата, разделени с интервал. Ако сборът е число, по-голямо от 1, се извеждат три числа, разделени с интервали, които са цялата част, следвана от числителя на дробната част и накрая знаменателя на дробната част. Задължително дробната част на резултата трябва да бъде несъкратима дроб.

Примери:

Вход	Вход	Вход
`7 3 2 3`	`2 3 3 36`	`3 4 4 7`
Изход	Вход	Вход
`3`	`3 4`	`1 9 28`

13. Неравенство (Есенен турнир, Шумен, 2004, зад. Е1). Напишете програма `INEQ.EXE`, която определя най-малкото от числата a, b или c, което може да бъде поставено в кръгчето, така че да се получи вярно следното неравенството:

$$2 > 7 - (\bigcirc + 1)$$

Вход: Три цели числа a, b и c, разделени с по един интервал.

Изход: Трябва да се изведе най-малкото от трите числа, за което е изпълнено горното неравенство. Ако за нито едно от трите числа не е изпълнено неравенството, да се изведе съобщението: „`No solution.`“

Пример:

Вход: `3 9 6` Изход: `6`

14. Сейф I (Есенен турнир, Шумен, 2004, зад. Е2). Програмистът Гошо Тарикатски съхранява сорсовете на своите програми в специален сейф. За да го обезопаси срещу крадци решил да създаде сложно система за отключване. Тя се задействала от два различни пулта, от които се въвеждале две трицифрени числа. След като се въведат числата се стартира програма, която генерира кода за сейфа по следния начин:

• Събира първата цифра на първото число с последната цифра на второто число. Ако резултатът е двуцифрено число, цифрите му се събират. Получената цифра се записва като първа за кода.

• Втората цифра се получава като по същият начин се обработят вторите цифри на двете числа.

• След сумирането на последната цифра на първото число и първата цифра на второто по посочената схема, се получава третата цифра на кода.

Гошо е много добър програмист, но е затрупан от поръчки, затова възлага на вас написването на програмата за получаване на кода. Програмата се нарича `KASA.EXE` и получава от клавиатурата две трицифрени числа, след което извежда на екрана получения по горната схема код, който също е трицифрено число.

Пример:

Вход: `492 851` Изход: `551`

15. Прозорец (Есенен турнир, Шумен, 2004, зад. E3). За пореден път в стаята на IV клас в училището в село Каръшко било счупено стъклото на един от прозорците. Както винаги в такъв момент в стаята нямало свидетели. Ръководството на училището, отчаяно от непрекъснатите разходи за нови стъкла възложило на прислужника да намери сред парчетата стъкла на тавана такова, че от него да може да се изреже необходимото за подмяна на счупеното стъкло. За щастие всички съхранени на тавана парчета били правоъгълни, но прислужникът не бил много силен по математика. Той измерил дължината a и ширината b на прозореца, а след това дължината c и ширината d на стъклото.

Сега вече само вие можете да помогнете на прислужника като напишете програма `JAM.EXE`, която прочита от клавиатурата числата a, b, c и d, и отпечатва „YES“, ако от стъклото може да се изреже прозорец и „NO“ в противен случай.

Примери:

Вход:	Вход:
`2 4 3 5`	`2 4 1 8`
Изход:	Изход:
`YES`	`NO`

16. Сейф II (Есенен турнир, Шумен, 2004, зад. D2). Програмистът Гошо Тарикатски съхранява сорсовете на своите програми в специален сейф. За да го обезопаси срещу крадци решил да създаде сложна система за отключване. Тя се задействала от два различни пулта, от които се въвеждали две цели числа, броя на цифрите, на които не бил предварително фиксиран, но е известно, че са не повече от девет и двете числа винаги имат равен брой цифри. След като се въведат числата се стартира програма, която генерира кода за сейфа като първо получава едно цяло число по следния начин:

• Събира първата цифра на първото число с последната цифра на второто число. Ако резултатът е двуцифрено число, цифрите му се събират. Получената цифра се записва като най-лява за числото.

• Втората цифра се получава като по същият начин се обработят втората цифра на първото число и предпоследната цифра на второто число и т. н.

• След сумирането на последната цифра на първото число и първата цифра на второто по посочената схема, се получава последната цифра на новото число.

За да се постигне пълна секретност така генерираното число се сумира с обратното си и получената сума е кода за сейфа. (Под обратното на дадено число разбираме, числото, прочетено в обратен ред. Например: обратното на 345 е 543).

Гошо е много добър програмист, но е затрупан от поръчки, затова възлага на вас написването на програмата за получаване на кода.

Програмата се нарича `KASA.EXE` и получава от клавиатурата две цели числа с равен брой цифри, след което извежда на екрана получения по горната схема код.

Пример:

Вход: `492971 851673` Изход: `1710016`

17. Художник (Есенен турнир, Шумен, 2004, зад. D3). Леонардо Хубавеца бил художник, и като всички и той бил страшно мързелив. В своите картини той много обичал да използва два основни елемента – къщички и елхички. Къщичката била съставена от квадрат като основа и равностранен триъгълник, като покрив (дължината на страната на триъгълника е с 2 знака по-дълга от страната на квадрата). Елхичката има равностранен триъгълник за корона, а броя на символите на стъблото се определя като се раздели страната на триъгълника целочислено на 2. Понеже той творял в много абстрактен стил, тези елементи били съставени от различните символи от компютърната клавиатура. Един ден му писнало да ги рисува самичък и решил да използва така модерните компютърни технологии да му вършат работата. Обадил се на един приятел, който имал връзки с програмисти да му помогне, а той пък се обадил на вас.

Сега от вас се иска да напишете програма `HUD.EXE`, която прочита от клавиатурата 2 числа и 1 символ. Първото число е 1 или 0 и според него трябва да отпечатате къщичка (ако е 0) и елхичка (ако е 1). Второто число (не по-голямо от 20) определя размера на фигурката (дължината на страната на квадрата при къщичката и на триъгълника за елхичката). Между символите, които образуват права линия на един ред се оставя по един интервал. Символа, който се въвежда е този, с който трябва да нарисувате фигурката. Вашата програма трябва да извежда на монитора желаната фигурка.

Примери:

Вход	Вход
`1`	`0`
`5`	`5`
`*`	`*`

Изход

```
    *
   * *
  *   *
 *     *
* * * * *
    *
    *
```

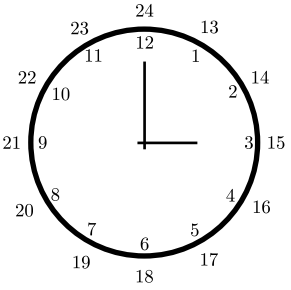
Изход

```
      *
     * *
    *   *
   *     *
  *       *
 *         *
* * * * * * *
  *       *
  *       *
  *       *
  * * * * *
```

18. Часовник (Национална олимпиада, общински кръг, 2005, зад. Е1). Да предположим, че един часовник показва 12 часа. Да се направи програма `CLOCK.EXE`, която въвежда от клавиатурата естествено число N (не по-голямо от 10000) и извежда на екрана броя на пълните завъртания на малката стрелка на часовника за N часа и кое число ще сочи малката стрелка след N часа.

Примери:

Вход:	Изход:
`15`	`1 3`
Вход:	Изход:
`8`	`0 8`
Вход:	Изход:
`24`	`2 12`



19. Конкурс (Национална олимпиада, общински кръг, 2005, зад. Е2). Три манекенки – A, B и C явяват на конкурс за красота и се оценяват по два критерия. Да се напише програма `MAN.EXE`, която определя, коя от манекенките е спечелила първо място по следния начин: първо се събират точките по двата критерия. Манекенката с най-голям сбор се класира на първо място. Ако се получи, че някои от манекенките са с равен брой точки, по-напред се класира тази, която има повече точки по първият критерий. Ако отново имат равен брой точки, първа се класира тази, която е излязла по-напред да дефилира (подредбата на манекенките е: A, B, C).

От клавиатурата се въвеждат три двойки цели числа (не по-големи от 1000). Първата двойка числа са оценките съответно по първия и втория критерий на първата манекенка. Втората двойка числа са оценките съответно по първия и втория критерий на втората манекенка. Аналогично за третата манекенка.

На екрана се извежда една от буквите A, B или C, в зависимост от това коя манекенка е класирана на първо място, като буквата A отговаря на първата манекенка, буквата B на втората манекенка и буквата C на третата манекенка.

Примери:

Вход	Изход	Вход	Изход
`5 6`	`B`	`9 6`	`B`
`7 4`		`13 2`	
`5 3`		`13 2`	

20. Оценки (Национална олимпиада, общински кръг, 2005, зад. E3). Да се напише програма `KLAS.EXE`, която въвежда от клавиатурата едно цяло число N ($N < 100$ – броя на учениците в един клас), след което последователно на нов ред се въвеждат N числа – оценките на учениците от класа. Програмата трябва да извежда средният успех на учениците от този клас.

Примери:

Вход	Изход	Вход	Изход
`5`	`4.20`	`10`	`3.90`
`3 2 6 6 4`		`6 3 2 5 5 3 6 3 4 2`	

21. Ръст (Национална олимпиада, общински кръг, 2005, зад. D1). Представете си, че сте голям баскетболен запалянко и освен това запален компютърен програмист. За да съчетаете двете си любими занимания (гледане на баскетболни мачове и програмиране), сте решили да направите програма, с която за всеки мач да определяте най-високия играч от стартовата петорка на отбора, на който сте привърженик. И така, направете програма `RAST.EXE`, която от клавиатура получава номера (число от 1 до 100) и ръста в сантиметри (цяло число) на всеки един от играчите от стартовата петорка и на екрана извежда номера и ръста на най-високия. Стартовата петорка винаги е такава, че няма двама най-високи играчи с един и същ ръст.

Пример:

Вход: Изход: `12 208`

```
5 201
7 192
12 208
14 196
21 202
```

22. Код (Национална олимпиада, общински кръг, 2005, зад. D2). За да се предпазят от крадци, вашите родители са инсталирали в къщи алармена система. Както всяка алармена система, така и тази има код, с който се пуска в действие, когато от къщи излиза последният човек, и се спира, когато се

прибира първият. Системата е такава, че кодът се състои само от нули и единици. На семеен съвет е решено кодът за системата да бъде съставен като се избере от цялото семейство (с явно гласуване) едно цяло, положително число, което има най-много 3 цифри, и от него се генерира код, съставен само от нули и единици, по следния алгоритъм:

– числото се дели на 2 и се получава частно и остатък (този остатък е 0 или 1). Полученият остатък се взема за последен символ от кода;

– полученото частно от първото деление отново се дели на 2 и пак се получава частно и остатък, като полученият остатък се взема за предпоследен символ от кода;

Продължава се по този начин, като всяка следваща стъпка от получаването на кода се състои в деление на 2 на частното, получено в предходната стъпка и вземане на остатъка като следващ символ от кода, като генерирането на кода върви от дясно на ляво.

Генерирането на кода продължава до момента, в който при поредното деление се получи частно 0 (остатъкът от това деление също влиза в кода).

След измислянето на числото от семейния съвет, на вас е възложено да направите програма `KOD.EXE`, която, получавайки на вход от клавиатурата измисленото число, извежда на екрана получения код за алармената система.

Пример:

Вход `34` Изход `100010`

23. Триъгълен масив (Национална олимпиада, общински кръг, 2005, зад. D3). Един масив A с N елемента ($3 \leqq N \leqq 20$), съдържащ цели числа между 0 и 1000, ще наричаме „триъгълен“, ако съществува цяло число M ($1 < M < N$), такова че:

$$A[1] < A[2] < \cdots < A[M] \quad \text{и} \quad A[M] > A[M+1] > \cdots > A[N].$$

Напишете програма `TRIMAS.EXE`, която на вход от клавиатурата получава масив A (на първия ред се въвежда стойността на N – брой на елементите в масива, а на втория, разделени със шпации, самите елементи) и определя дали масивът е триъгълен или не. На изхода програмата трябва да изведе „да“ или „не“ („да“ – масивът е триъгълен, „не“ – масивът не е триъгълен).

Примери:

Вход:	Вход:
`6`	`7`
`3 40 67 200 123 112`	`56 34 34 115 450 345 211`
Изход:	Изход:
`да`	`не`

24. Обратно число (Зимни празници, Бургас, 2005, зад. E1). Младата програмистка Пепи скучаела, след като си била решила всичките задачи, останали от последното занимание в Школата по програмиране. Както си

драскала различни неща на листче, тя забелязала, че ако вземе някакво цяло положително число A и напише цифрите му в обратен ред, то за полученото обратно число A' може да се окаже едно от трите: A да е равно на A', A да е по-малко от A' или A да е по-голямо от A'. Например, ако A е 1221, тогава $A = A'$, ако A е 1231, тогава $A < A'$, а ако A е 1321, тогава $A > A'$. Пепи веднага си написала програма, която по зададено число да проверява кой от трите случая е в сила за това число. Предлагаме и на Вас тази задача.

Напишете програма `OBRA.EXE`, която чете от клавиатурата едно осемцифрено число A, сравнява го с обратното му число A' и извежда на екрана 0, ако $A = A'$, извежда -1, ако $A < A'$ и извежда 1, ако $A > A'$.

Примери:

Вход: `24354752`	Вход: `32418102`	Вход: `66666666`
Изход: `-1`	Изход: `1`	Изход: `0`

25. Ех, тези прозорци (Зимни празници, Бургас, 2005, зад. Е2). Помните ли как през есента помогнахме с програма на училището в село Каръшко да разреши проблема със счупеното стъклото на един от прозорците в стаята на IV клас. За съжаление случват се и по-лоши работи. Сега, две стъкла са счупени и, разбира се, пак няма нито виновник, нито свидетел на случилото се. Едното от стъклата е с размери A на B, а другото – с размери C на D. Ръководството на училището отново изпратило прислужника на тавана да потърси парчета стъкла, с които да могат да се заменят счупените. Този път се оказало, че на тавана има две правоъгълни парчета с размери U на V и X на Y. На прислужника никак не му се иска да развали и двете парчета, а по-скоро да опита да поправи счупените прозорци като използва само едно от двете. Всички знаем, че той не е много силен по математика и едва ли ще може сам да реши кое от намерените парчета ще е достатъчно да се подменят и двете счупени стъкла.

Ще се наложи пак да помогнете като напишете програма `JAMS.EXE`, която прочита от клавиатурата първо целите положителни A, B, C, D, после целите положителни U, V, X и Y, и извежда на екрана:

`YES YES` – ако всяко от двете парчета е достатъчно да се заменят двете счупени стъкла;

`YES NO` – ако първото е достатъчно, но второто – не;

`NO YES` – ако второто е достатъчно, но първото – не;

`NO NO` – ако нито едно от двете не е достатъчно.

Пример:

Вход:	Изход:
`2 4 3 5`	`YES NO`
`4 7 10 2`	

26. Минимакс (Зимни празници, Бургас, 2005, зад. E3). Напишете програма `MINMAX.EXE`, която определя най-малкото и най-голямото измежду зададени цели числа.

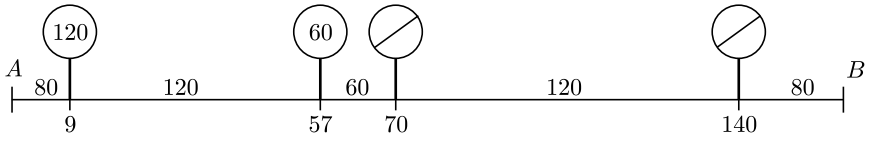
Програмата трябва да въведе от клавиатурата броя на зададените числа (в примера по-долу, броят на зададените числа е 9), след което да въведе, едно по едно, и самите числа.

Програмата трябва да изведе на един ред на екрана най-малкото и най-голямото от зададените числа (в този ред), разделени с един интервал.

Пример:

Вход:	Изход:
9	-7 24
5	
5	
-7	
13	
9	
24	
6	
24	
19	

27. Пътуване (Зимни празници, Бургас, 2005, зад. D1). В страната КАТландия обичайната максимална позволена скорост на движение е 80 км/час. Когато по някоя отсечка се налага максималната скорост да е друга (по-висока или по-ниска от разрешената в момента), правилата за движение повеляват да се постави знак за новата позволена скорост в началото на отсечката и знак, който маркира края на отсечката (вж. фигурата). След напускане на отсечката като позволена се възстановява тази скорост, която е била валидна при навлизане в отсечката.



Напишете програма `TRAVEL.EXE`, която да пресмята времето, необходимо за пътуване от един град A до друг град B, като се знае точното разположение на знаците за промяна и възстановяване на максималната разрешена скорост. Предполага се че през цялото време ще пътуваме с максималната разрешена на съответната отсечка скорост.

Входните данни ще бъдат зададени на стандартния вход. На първия му ред е зададено цялото L – разстоянието в километри от град A до град B ($L \leqq 5000$). На всеки един от следващите редове е описан по един от пътните

знаци, в реда по който срещат при пътуването от A към B. За знаците показващи начало на отсечка съответният ред на стандартния вход съдържа цялото положително R – разстоянието от града A до мястото на знака и новата скорост V – също цяло положително, не надминаващо 130. За знаците показващи край на отсечка съответният ред на стандартния вход съдържа разстоянието R от града A до мястото на знака и -1. Поне една двойка знаци за начало и край на отсечка ще бъде зададена на входа

Резултатът от работата на програмата – времето в часове, необходимо да се отиде от град A до град B като се движим с максиманата позволена на всяка отсечка скорост – да се изведе на стандартния изход, като дробно число с две цифри след десетичната точка (разлики от ± 0.01 се приемат за допустими).

Пример:

Вход:	Изход:
`150`	`1.44`
`9 120`	
`57 60`	
`70 -1`	
`140 -1`	

28. Кръстословица I (Зимни празници, Бургас, 2005, зад. D2). В правоъгълна таблица, клетките на която са запълнени с букви, можем да „скрием“ една дума, като запишем буквите ѝ във всяка една от осемте възможни посоки – надолу, нагоре, надясно, наляво и по четирите диагонала както е показано в следната таблица с думата ЕДНО:

Е		О		Е	Д	Н	О		Е						Е		О						О
Д		Н								Д				Д				Н				Н	
Н		Д		О	Н	Д	Е				Н			Н					Д			Д		
О		Е										О	О							Е	Е			

Напишете програма `CROSS.EXE`, която по зададена таблица и дума да намери колко пъти е скрита дадената дума в дадената таблица.

Програмата трябва първо да прочете от клавиатурата броя на стълбовете S и броя на редовете R на таблицата, разделени с един интервал. Броят на редовете и броят на стълбовете няма да е по-голям от 255. След това програмата трябва да прочете R низа, съставени от по S букви, които задават таблицата. Най-накрая програмата трябва да прочете думата която е „скрита“ в таблицата.

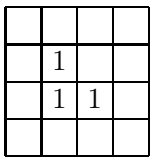
Програмата трябва да изведе на екрана числото, което показва колко пъти зададената дума е „скрита“ в таблицата.

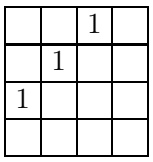
Пример:

Вход:
```
12 8
КРЕСПФЕСДХРД
АНОСГЕДНОЕВП
ЖПРСДННВПДАВ
ДЕОНВООВЖТОР
ВЖОЯЗЛПДПВНЯ
ЕЕЯНВФЛКНВДР
ФВЛЯДЦВТГДЕП
ЛВЦБКЕЯЦБЦВБ
ЕДНО
```

Изход:
```
5
```

29. Игра II (Зимни празници, Бургас, 2005, зад. D3). Пешо Геймъра решил сам да си направи компютърна игра. Разбира се, че като начало това няма да е някоя сложнотия като тези дето се играят в клубовете, а нещо много, много по-просто. Пешо си мислел за квадратна дъска с размери $N \times N$, разделена на еднакви квадратчета със страна 1 (виж фигурата). Всяко квадратче е стая, в която може да се влиза от съседните отляво и отгоре (когато има такива). Някои от стаите са безопасни, а в някои има скрити подаръци от най-различен вид (на фигурата те са означени с 1). Целта на играта ще бъде да се тръгне от най-горното най-ляво квадратче (входа на играта) и да се достигне до най-долното най-дясно (изхода на играта) като се прави опит да се съберат колкото може повече подаръци. За лявата от дъските показани на фигурата в най-добрия случай могат да се вземат всичките 3 подаръка, докато за дясната максималният брой подаръци е 1. За да може да подбира хубави игри, Пешо се нуждае от програма `IGRA.EXE`, която по зададено разпределение на подаръците по стаите да определя максималния брой подаръци, които играчът може да събере.





Описанието на дъската се въвежда от клавиатурата. На първият ред се задава размерът на дъската N ($N < 100$). На всеки от следващите N реда се задава по един низ от нули и единици с дължина N, който описва съответния ред от таблицата на играта, като нула означава празна стая, а единица – стая с подарък.

На стандартния изход програмата трябва да изведе намерения максимален брой подаръци.

Примери:

Вход:	Вход:
4	4
0000	0010
0100	0100
0110	1000
0000	0000
Изход:	Изход:
3	1

30. Автобусни линии (Национална олимпиада, областен кръг, 2005, зад. Е1). Коко живее в голям град и непрекъснато пътува с градски автобуси. Номерата на всички автобусни линии са трицифрени. Веднъж Коко с изненада установил, че номерата на някои линии имат особено свойство: като задраскаме поотделно всяка от цифрите, сумата на две от получените числа е равна на третото число. Напишете програма `BUS.EXE`, която прочита от клавиатурата две цели числа A и B, и извежда на екрана всички числа по-големи или равни на A и по-малки или равни на B, които имат различни цифри и притежават зададеното по-горе свойство, като разполага всяко число на отделен ред.

Пример:

Вход:	Изход:
105 200	105
	106
	107
	108
	109

31. Облицовка на вана (Национална олимпиада, областен кръг, 2005, зад. Е2). Стените на вана с форма на правоъгълен паралелепипед с дължина a см, ширина b см и дълбочина c см трябва да се облицоват с квадратни фаянсови плочки с дължина на страната d см. Да се напише програма `VANA.EXE`, която прочита от клавиатурата естествените числа a, b, c и d, и извежда на екрана минималния брой плочки, които трябва да се купят за облицоването. Приемаме, че ако някъде трябва да се постави част от плочка, трябва да се предвиди цяла плочка за това място.

Пример:

Вход: 200 100 50 10 Изход: 300

32. Автомобилно състезание (Национална олимпиада, областен кръг, 2005, зад. Е3). Ежегодно в с. Победа се провежда автомобилно състезание. Участниците напълват резервоарите на колите и победител е този, който

успее да измине най-дълго разстояние без да налива повече гориво. Като при всяко състезание и тук имало залагания кой ще бъде победител.

Хитран успял да се сдобие с информация за това, колко бензин изразходва за една обиколка на пистата всяка от колите При старта той записвал обема на резервоара и количеството бензин, което колата изразходва за изминаването на една обиколка. Тъй като броят на колите не е известен предварително, накрая Хитран записва две нули. Как Хитран да разбере на кого да заложи?

Напишете програма `AUTO.EXE`, която отпечатва номера на победителя по реда на стартирането и неговото постижение. Номерирането на колите започва от едно и се отчитат само завършените обиколки.

От клавиатурата последователно се въвеждат по две числа на ред – обема на резервоара и необходимото количество бензин за една обиколка. Числата са отделени с един интервал. На последния ред се въвеждат две нули.

Програмата извежда на екрана две числа – първото е стартовия номер на колата победител в състезанието, а второто достигнатото от нея разстояние. Ако има две коли с еднакво постижение победител е тази, която има по-малък номер.

Пример:

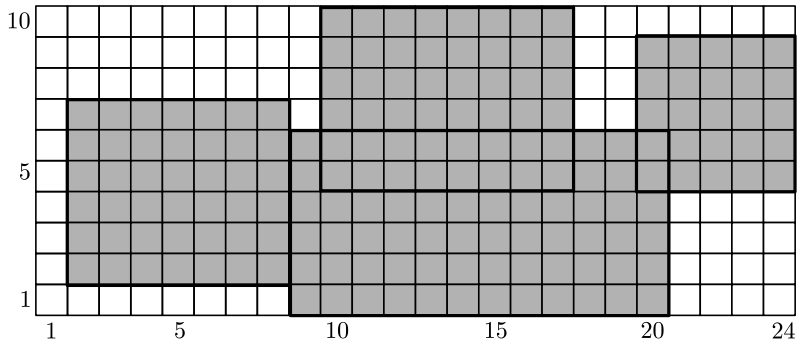
Вход:	Изход:
`60 4`	`3 24`
`80 5`	
`96 4`	
`0 0`	

33. Площадки (Национална олимпиада, областен кръг, 2005, зад. D1). Теренът за спорт на едно училище е правоъгълник с размери N на M метра. Да си мислим, че теренът е разбит с хоризонтални и вертикални прави на квадратчета със страна 1, така че всяко квадратче можем да означим с поредните номера на стълба и реда, в който се намира. На фигурата е показан терен с $N = 24$ и $M = 10$. Най-долното му най-ляво квадратче означаваме с (1,1), най-долното най-дясно – с (24,1), най-горното най-ляво – с (1,10), а най-горното най-дясно – с (24,10). По различни времена върху терена са били асфалтирани K правоъгълни спортни площадки така, че сега част от теренът е асфалтиран, а част – не. На фигурата тези площадки са защриховани и с удебелени граници.

Разположението на всяка площадка се представя с разположението на най-долното си най-ляво и най-горниото си най-дясно квадратче. Училището иска да асфалтира и останалата част от терена, за това е добре да знае колко квадратни метра не са асфалтирани. Напишете програма `SPORT.EXE`, която да определя лицето на неасфалтираната площ (белите квадратчета на фигурата).

Данните се въвеждат от стандартния вход. На първия ред са зададени

целите числа N, M и K ($0 < N, M \leqq 500, 0 \leqq K < 50$). На всеки от следващите K реда са зададени, разделени с по един интервал, целите числа X_1, Y_1, X_2 и Y_2 – разположението на най-долното най-ляво и най-горното най-дясно квадратче на една от площадките.



Програмата трябва да изведе на стандартния изход намереното лице.

Пример:

Вход:	Изход:

`24 10 4`
`2 2 8 7`
`20 5 24 9`
`9 1 20 6`
`10 5 17 10`
	`71`

34. Правоъгълник (Национална олимпиада, областен кръг, 2005, зад. D2). Дадени са N точки ($0 < N < 100$) с координатите си в равнината Oxy. Някои от точките може да съвпадат помежду си. Да се напише програма `RECT.EXE`, която намира най-малкия по площ правоъгълник със страни, успоредни на координатните оси, такъв че всичките дадени точки да се намират във вътрешността или на страните му. Програмата трябва да изведе координатите на центъра на намерения правоъгълник и дължините на страните му.

Данните се четат от стандартния вход. Първият ред съдържа броя N на дадените точки. За всяка точка на отделен следващ ред са записани нейните координати x и y. Те са цели числа със стойности между -1000 и 1000 и са отделени с по един интервал.

На един ред в стандартния изход програмата трябва да изведе 4 числа с десетична точка с една цифра в дробната част. Тези числа трябва да са координатите x и y на центъра на търсения правоъгълник и дължините на страните, които са съответно успоредни на осите Ox и Oy. Изведените числа трябва да са разделени с точно един интервал.

Пример:

Вход:	Изход:
5 -1 3 0 0 2 4 2 -5 7 1	3.0 -0.5 8.0 9.0

35. Тото (Национална олимпиада, областен кръг, 2005, зад. D3). В разгорялата се тото-треска за големия джакпот от играта „6 от 49“ се включил и младият програмист Пешо. Всеки тираж той пускал M комбинации, всяка от по 6 числа. Било доста досадно всеки път да сравнява своите комбинации с печелившите числа от първо, второ и трето теглене.

Пешо направил програма, която да върши тази работа вместо него. Програмата първо въвежда от стандартния вход трите печеливши комбинации, след това числото M и накрая комбинациите на Пешо. Числата във всяка комбинация не са непременно подредени по големина.

Резултатът се извежда на стандартния изход и се състои от M реда, по един за всяка от входните комбинации. Ако комбинация K не печели нищо, редът съдържа само нейния номер K. Ако в комбинация K от теглене X има Y познати числа, на ред K в изхода се извеждат и числата X и Y. От първите две тегления се интересуваме от 3, 4, 5 или 6 познати числа, а от трето теглене – само от шестица.

В програмата на Пешо, обаче имало някаква грешка и веднъж пропуснала тройка от второ теглене. Напишете програма `TOTO.EXE`, която да изпратим на Пешо.

Пример:

Вход:	Изход:
3 7 10 11 33 49 25 4 13 7 28 38 4 7 13 32 37 43 4 1 9 16 25 36 49 3 15 16 22 29 31 7 13 4 32 37 43 3 11 26 33 41 49	1 2 3 2 3 3 6 4 1 4

36. Интервали (Национална олимпиада, национален кръг, 2005, зад. D1). Програмистката Деляна написала програма, която трябвало да подреди няколко числа по големина и да ги изведе на екрана, разделени с по един интервал. За нещастие, Деляна забравила да извежда интервали и например вместо `1 5 12 20 25 88 102` се получавало `1512202588102`

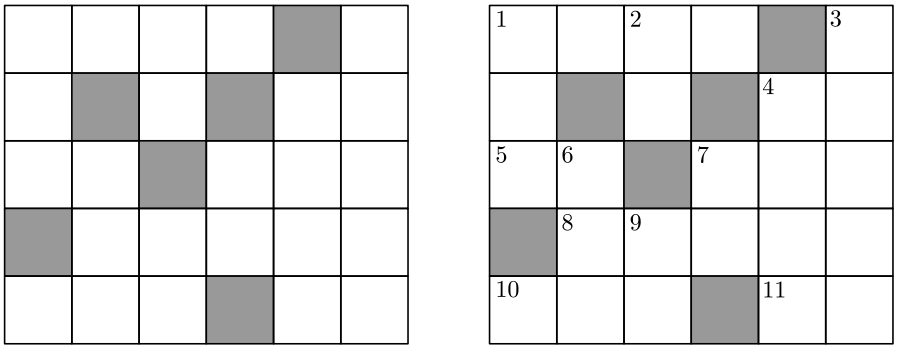
Напишете програма `SPLIT.EXE`, която се опитва да раздели низ от цифри на части, така че да се получат колкото се може повече числа, подредени по големина. Низът се въвежда от стандартния вход и има до 100 цифри. Резултатът се извежда на стандартния изход. Получените числа трябва да не започват с нула и да са по-малки от 10000. Всеки от тестовите примери ще има поне едно решение. Състезателите, които са намерили решение с най-много числа ще получат 10 точки, а състезателите, предложили решение с по-малко числа ще получат пропорционален брой точки.

Примери:

Вход: `455095601`
Изход: `4 5 50 95 601` (10 точки)
или `45 50 95 601` (8 точки)
или `45 509 5601` (6 точки)

Вход: `101212222`
Изход: `10 12 12 222` (10 точки)
или `101 212 222` (7,5 точки)
или `10 121 2222` (7,5 точки)

37. Кръстословица II (Национална олимпиада, национален кръг, 2005, зад. D2). Когато започва съставянето на нова кръстословица, авторът трябва да се справи с една нелека задача. Той взема една мрежа с M реда и N стълба, някои от квадратчетата на която са защриховани. Такава мрежа е показана вляво на фигурата. Думите, които се вписват в кръстословици са с поне две букви и се разполагат в празните квадратчета или в ред, отляво надясно („хоризонтално“), или в стълб, отгоре надолу („вертикално“). Всяко квадратче, в което започва дума трябва да се номерира с положително цяло число, започвайки с 1. Номерата трябва да растат отляво надясно в реда и отгоре надолу по редовете. Хоризонталните думи започват в квадратче, ако то е от първия стълб или квадратчето вляво е запълнено. Вертикалните думи започват в квадратче, ако то е от първия ред или квадратчето отгоре е запълнено (вляво на Фигурата е показана номерацията на мрежата).



Напишете програма `CROS.EXE`, която по зададена мрежа построява исканата номерация.

В първия ред на стандартния вход ще бъдат зададени числата M и N, разделени с един интервал. Следващите M реда са описание на мрежата.

Всеки ред е представен с низ с дължина N, съставен от тирета (-) и звезди (*), като тиретата означават празните, а звездите – защрихованите клетки.

На първия ред на стандартния изход програмата трябва да изведе броя K на номерираните квадратчета. Всеки от следващите K реда описва по едно от номерираните квадратчета, в нарастващ ред на номерата. Редът започва с номера на квадратчето. След него се извеждат номерът на реда и номерът на стълба в които се намира съответното квадратче. Редът завършва с една буква: H – ако в квадратчето започва само хоризонтална дума, V – ако в квадратчето започва само вертикална дума, B – ако в квадратчето започва и хоризонтална дума и вертикална дума.

Пример:

Вход:
```
5 6
----*-
-*-*--
--*---
*-----
---*--
```

Изход:
```
11
1 1 1 B
2 1 3 V
3 1 6 V
4 2 5 B
5 3 1 H
6 3 2 V
7 3 4 B
8 4 2 H
9 4 3 V
10 5 1 H
11 5 5 H
```

38. Аритметика (Национална олимпиада, национален кръг, 2005, зад. D3). Във всеки ред на един текстов файл е записано или цяло положително число, или знак +, или знак *, или отваряща скоба (, или затваряща скоба). Спазено е изискването целият файл да има един от следните 4 вида, където *Последователност* има същия вид, както целия файл:

Знак Цяло число Цяло число	Знак Цяло число (Последователност)	Знак (Последователност) Цяло число	Знак (Последователност) (Последователност)

Пример:
```
+
(
*
13
(
+
14
3
)
)
7
```

Напишете програма `ARIT.EXE`, която постъпково скъсява колонката с данни, като замества по подходящ начин всяка Последователност с резултата от съответното аритметично действие (събиране или умножение) – на всяка стъпка намира последователни отваряща скоба, знак за действие, две числа и затваряща скоба, и ги замества с пресметнатата стойност. Например:

+		+		+	→	228
(		(	→	221		
*		*		7		
13		13				
(	→	17				
+		)				
14		7				
3						
)						
)						
7						

Програмата трябва да изведе на стандартния изход последния резултат като едно цяло число. Данните от входния текстов файл (съдържащ не повече от 200 реда) се четат от стандартния вход, като за означаване края на входа е добавен ред с една нула. Записаните числа, както и всички междинни резултати не надминават 1 000 000.

Пример:

Вход:	Изход:
+	228
(	
*	
13	
(	
+	
14	
3	
)	
)	
7	
0	

39. Календар I (Пролетен турнир, Ямбол, 2005, зад. Е1). В един снежен съботен следобед на 1-ви януари 2005 година Пенчо дремел замислено пред телевизора и си мислел с носталгия за отминалите празници – за подаръците, за вкусните почерпки и ред други подобни приятни неща. Нещастно разсъждавал, че следващите подобни събития ще са чак след 12 месеца. Изведнъж му хрумнало, че други хубави празници (с торти, подаръци и веселба) са

рожденните дни! Окрилен той отворил тефтерчето, където държал записани рожденните дни на приятелите си, за да намери най-близката дата. Изведнъж се сепнал! Ами ако рожденния ден се пада в средата на седмицата?! Преди класно по математика, например?... Ужас! Понеже нямал календар пред себе си, Пенчо решил за миг да седне на компютъра и да напише програма, която да му отговаря на простия въпрос – „Какъв ден от седмицата ще бъде еди-коя си дата от 2005-та година?“ Справил се за по-малко от час...

А вие за колко ще се справите с подобна програма?

Напишете програма `CALENDAR.EXE`, която определя в какъв ден от седмицата се пада зададена дата от 2005-та година.

Входът е от клавиатура като се въвеждат 2 числа разделени с интервал – M D, като M е от 1 до 12 – месецът, в който е рожденния ден, а D е денят.

На изход на екрана се извежда число от 1 до 7, указващо кой ден от седмицата се пада зададената дата, като:

1 – понеделник
2 – вторник
3 – сряда
4 – четвъртък
5 – петък
6 – събота
7 – неделя

Пример:

Вход: `2 15` Изход: `2`

40. Екскурзия (Пролетен турнир, Ямбол, 2005, зад. Е2). Минчо Планинарски тръгнал на планина. За целта той написал бързо един маршрут от обекти, които искал последователно да посети. До всеки от обектите Минчо записал и надморската му височина. Започнал приготовления за похода – раници, обувки и т.н. След 2-3 дни си спомнил за своя списък и решил да види между които две точки от маршрута той ще преодолее най-голяма разлика във височините (независимо дали ще слиза надолу или ще се качва нагоре) – това щяло да му помогне да реши в коя посока да обходи обектите!

Напишете програма `TOUR.EXE`, която по зададен списък от надморски височини определя най-голямата денивелация (разлика във височините) между 2 съседни туристически обекта.

На вход имаме списък с надморските височини на обектите (въвежда се от клавиатура – на всеки ред по едно цяло число между 100 и 9999), като не е известен предварително броят им. За сметка на това знаем, че списъкът свършва, когато вместо надморска височина въведем числото 0 (няма обект с такава надморска височина).

На екрана изведете едно цяло положително число – намерената най-голяма денивелация между 2 съседни туристически обекта.

Пример:

Вход:	Изход
450 1100 1000 1900 800 1300 1700 2100 2500 2925 2300 0	1100

41. Делители (Пролетен турнир, Ямбол, 2005, зад. E3). Дадени са N $(2 \leqq N \leqq 20)$ различни цели числа между 2 и 1000. Напишете програма `DIV.EXE`, която да намира тези от тях, които се явяват делители на сумата им.

Входът е от клавиатура и на първи ред се въвежда N. На всеки от следващите N реда се въвежда по едно число между 2 и 1000.

На екрана изведете последователно (на отделни редове) тези числа от зададените, които отговарят на условието, т.е. делят сумата на N-те числа.

Пример:

Вход:	Изход:
5 23 560 340 2 755	560 2

42. Календар II (Пролетен турнир, Ямбол, 2005, зад. D1). Кирил Познайков е феномен в следния смисъл: той може много бързо, наум, да пресмята какъв ден от седмицата е дадена дата между 1 януари 1980 г. и 31 декември 2099 г.

За да се засече неговият рекорд и да влезе той в „Книгата на Гинес“, трябва да се извърши прецизно засичане на количеството дати, които може да разгадае Кирчо за 1 минута. Тъй като няма друг човек, който да смята като г-н Познайков, то е необходимо да се напише програма `CALENDAR.EXE`, която да помага на журито.

От стандартния вход програмата ще получава един ред с 3 числа, разделени с интервали – D M Y. Както е ясно D е деня от месеца (от 1 до 31), M е месеца в годината (от 1 до 12) и Y е годината (от 1980 до 2099).

На стандартния изход програмата трябва да извежда едно число между 1 и 7, съответстващо на деня от седмицата, в който е зададената дата, като :

1 – понеделник

2 – вторник

3 – сряда

4 – четвъртък

5 – петък

6 – събота

7 – неделя

За справка: 1 януари 1980 г. е бил вторник.

Пример:

Вход: `24 5 2005` Изход: `2`

43. Игра III (Пролетен турнир, Ямбол, 2005, зад. D2). Представете си, че сте се събрали K приятели ($2 \leqq K \leqq 10$) и сте решили да поиграете следната игра:

Слагате N ($5 \leqq N \leqq 1000$) купчинки от камъчета на земята, съдържащи съответно $m_1, m_2, \ldots, m_N$ камъчета. След това започвате играта по предварително определен ред (всички K играча се редуват да правят своя ход). Ходът е следния – играчът избира една купчинка, в която има поне 2 останали камъка и я разделя на 2 нови купчинки, всяка с произволен брой (поне 1) камъчета. Играта се печели от този, който направи последния ход в нея.

При началния жребий на Вас се пада да изберете кой по ред да играете. Точно в този момент Ви хрумва нещо и отскачате до вкъщи под предлог, че трябва да обядвате!

Въпросът е можете ли да напишете програма `GAME.EXE`, която по зададените брой приятели и купчинки от камъчета, вместо вас да определя кой по ред трябва да започнете играта, за да сте сигурен, че ще победите.

Входните данни се четат от стандартния вход и са в следния формат:

На първи ред стоят 2 числа – K и N, съответно брой играчи и брой купчинки. На следващите N реда имаме по едно число между 1 и 1000, което указва колко точно камъчета има в поредната купчинка.

Вашата програма трябва на стандартния изход да изведе едно единствено число между 1 и K – това е поредния номер, под който трябва да започнете играта, за да сте сигурен, че ще спечелите.

Пример:

Вход:	Изход:
2 5 2 8 2 1 10	2

44. Монополи (Пролетен турнир, Ямбол, 2005, зад. D3). Фирма за детски игри решила да пусне на пазара опростен вариант на играта „Монополи“ за по-малки деца. За целта производителите искали в играта да има банкноти само с една стойност.

Ако са известни всички суми, които се налага да се плащат в играта (цени, глоби, премии и т.н.), от Вас се иска да помогнете, написвайки програма `MONOPOLY.EXE`, намираща най-високата стойност, която могат да имат банкнотите, използвани в играта, така че да може да се изплаща всяка от зададените суми с цяло число банкноти.

На стандартния вход се въвеждат следните данни:

– на първия ред стои числото N – броят на различните суми, които се налага да се изплащат в играта ($2 \leqq N \leqq 100$)

– На следващите N реда са зададени самите суми като числа между 2 и 1000.

На стандартния изход се извежда едно единствено число – намерената стойност на банкнотата.

Пример:

Вход:	Изход:
3 12 6 21	3

45. Числа (Есенен турнир, Шумен, 2005, зад. E1). По време на дългите учебни часове нашият приятел Умко от училището в село Каръшко, което в наша чест е наречено „Програмистите на България“, се забавлявал като измислял различни игри с числа. Случайно забелязал, че от едно петцифрено число, чрез разместване на цифрите му, могат да се получат много различни числа и дори ги преброил. Те били 120. Все пак написването им било досадна работа, дори когато ти е скучно в час. Умко решил, че не го интересуват всички тези числа, а само най-малкото и най-голямото от тях. Той започнал да разглежда различни петцифрени числа и за всяко от тях да намира разликата между най-голямото и най-малкото число, получени от неговите

цифри. Скоро и тази работа му омръзнала, но все пак искал на всяка цена да знае тази разлика за всяко едно петцифрено число.

Сега вече само вие можете да му помогнете като напишете програма `NUMBERS.EXE`, която въвежда от клавиатурата на компютъра цяло петцифрено число N и извежда на екрана разликата между най-малкото и най-голямото измежду числата, получени като се разместят цифрите на N.

Примери:

Вход: `56342` Изход: `41976`

Пояснение: Най-голямото число, което се получава от цифрите 2, 3, 4, 5 и 6 на числото 56342, е 65432, най-малкото е 23456. Разликата на тези две числа е точно 41976.

Вход: `10002` Изход: `19989`

Пояснение: Най-голямото число, което се получава от цифрите 1, 0, 0, 0 и 2 на числото 10002, е 20001, най-малкото е 12. Разликата на тези две числа е точно 19989.

46. Бонбонки (Есенен турнир, Шумен, 2005, зад. Е2). Умко (нашият познайник от училище „Програмистите на България“) имал странно хоби. Той имал четири кристални купички, в които грижливо събирал шоколадови бонбонки. Странното било това, че той не обичал да си похапва от тях, а само да си ги гледа разпределени по равен брой в своите купички. Във всяка от тях той държал по 10 бонбонки. Това, че нашето умно момче не обичало шоколадовите бонбонки съвсем не означавало, че и неговата сестра не обичала да си похапва сладко, сладко от тях. Тя само изчаквала Умко да отиде на училище и с огромно удоволствие си похапвала. Така всеки ден след като се връщал от училище, той виждал, че любимите му кристални купички вече не са пълни с по 10 бонбонки във всяка и тичал до магазина, за да набави неоходимите бонбонки. След известно време Умко сменил тактиката. Вместо всеки ден да ходи до магазина и да си купува бонбонки, просто ги премествал от едната купичка в другата, за да станат пак по равен брои в четирите купички. В случаите, когато това не било възможно, Умко все пак тичал до магазина.

Тъй като му омръзнало всеки път да пресмята колко бонбони трябва да размести и да купи, нашият приятел ви моли да напишете програма `BONBONKI.EXE`, която прочита от клавиатурата последователно броя на бонбонките във всяка една от четирите купички и ако е възможно те да се разпределят по равно, извежда на екрана от коя купичка колко бонбонки се изваждат или добавят. Ако бонбонките се изваждат от купичката, пред броя им се поставя знака „−“, а ако те се добавят – знака „+“. Ако броят на бонбонките в съответната купичка остава непроменен, се извежда числото 0 без знак. Ако не е възможно бонбонките да се разпределят по равно в четири-

те купички, програмата извежда на екрана само броя на бонбонките, които Умко трябва да купи от магазина, за да поправи разпределението им.

Примери:

Вход:	Вход:
`2 3 4 6`	`4 9 8 7`
Изход:	Изход:
`1`	`+3 -2 -1 0`

47. Познай цифрата (Есенен турнир, Шумен, 2005, зад. E3). По време на междучасието в училището в село Каръшко играели следната игра: Подреждат се плочки, на всяка от които е изписана цифра от 0 до 9. Плочките са обърнати с надписа надолу, така че да не се вижда коя е цифрата на нея и се знае, че с тях са изписани числата от 10 до 99 в нарастващ ред (101112131415... 979899).

Един от играчите посочва една от плочките, а този който е наред, познава коя цифра е записана на плочката. Печели този играч, който е познал най-много цифри. Нашият добър приятел Умко и този път иска да надхитри приятелите си, но за целта вие трябва да му помогнете като напишете програма `CIFRA.EXE`, която въвежда число k ($1 \leqq k \leqq 180$) и извежда цифрата изписана на k-тата плочка.

Примери:

Вход: `17`	Вход: `28`
Изход: `1`	Изход: `3`

48. Календар III (Есенен турнир, Шумен, 2005, зад. D1). Училището в село Каръшко, което както е известно, признателното ръководство кръсти на наше име: „Програмистите на България“, на 20.11.2005 г. ще празнува своя 70-ти юбилей. По случай празника директорът решил да организира състезание, като всеки един от участниците получил следната задача: по зададена дата (ден, месец, година) и какъв ден от седмицата е била тя, да се отпечата календарът за определен месец от същата година.

Умко (еднин от най-умните и мързеливи участници) ви моли да му помогнете да спечели състезанието, като напишете програма `CALENDAR`, която прочита от клавиатурата данните, зададени от директора и отпечатва календара на екрана на компютъра.

Програмата приема от първия ред на стандартния вход четири цели числа: деня, месеца, годината и деня от седмицата за известната дата (понеделник се отбелязва с 1, вторник – с 2, сряда – с 3 и т.н.), а от втория ред – само едно число – месеца, чийто календар трябва да се отпечата.

На стандартния изход, програмата отпечатва календара по следния начин: на първия ред се отпечатват първите букви на дните (на латиница) на седмицата, отделени с по два интервала. На следващите редове се отпечатват

дните на зададения месец, започващи от 1, като всяка дата е под съответния ден от седмицата за този месец. Едноцифрените дни са разделени помежду си с по два интервала, а двуцифрените – с по един.

Пример:

Вход:
```
19 11 2005 6
1
```

Изход:
```
P  V  S  C  P  S  N
                1  2
3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

49. Милионер (Есенен турнир, Шумен, 2005, зад. D2). Впечатлен от историята за забогатяване на Хенри Форд (започнал от една ябълка, продал я, купил 2 ябълки и т.н.), която госпожата в училището „Програмистите на България“ разказала, Умко, който освен добър ученик е и футболен фен, си изградил следната стратегия за забогатяване, залагайки в „Еврофутбол“:

• За всеки нов тираж се прави един залог с коефициенти 2 или 3. Това значи, че ако заложим K лв. печалбата ще бъде съответно $2K$ лв или $3K$ лв.

• За всеки тираж заложената сума се определя от печалбата от предходен тираж увеличена с 1 лв. Всяка печалба се залага два пъти: веднъж с коефициент 2 и веднъж с коефициент 3. Спазва се правилото, че новата печалба не трябва да е по-малка от предходната.

Помогнете на Умко да намери след колко залагания, следвайки горната стратегия, ще стане милионер, ако в началото заложи 1 лв. Напишете програма `MILIONER`, която въвежда цяло число n ($1 < n \leqq 440000$) и извежда след най-малко колко залагания ще се получи печалба не по-малка от n и каква ще е стойността на тази печалба.

Пример:

Вход:
```
9
```
Изход:
```
5 9
```
Пояснение:
Печалбите са съответно:
```
2 3 6 8 9
```

Вход:
```
25
```
Изход:
```
11 26
```
Пояснение:
Печалбите са съответно:
```
2 3 6 8 9 12 14 18 20 21 26
```

50. Думи в текст (Есенен турнир, Шумен, 2005, зад. D3). За домашно по информатика учителката дала на Умко интересна задача, но той както обикновенно „няма време“ да я реши и пак ще прибегне до вашата помощ.

Все пак училището му носи вашето име: „Програмистите на България“. В задачата се иска да се напише програма `WORDS`, която прочита от първия ред на стандартния вход дума, съдържаща не повече от 20 знака, а от втория – произволен текст, съдържащ не повече от 1000 знака. Програмата трябва да отпечата номера на началния и крайния знак на най-дългата последователност от знаци в текста, които принадлежат на думата. Програмата трябва да може да различава малка от главна буква. Ако в текста не се среща нито една буква от думата, програмата извежда 0.

Примери:

Вход:
```
vaza
Programistite na Bulgaria se sastezavat.
```
Изход:
```
35 38
```

Вход:
```
ana
Atanas yade ananas
```
Изход:
```
13 17
```

Вход:
```
a
aaaa
```
Изход:
```
1 4
```

ИЗТОЧНИЦИ

1. *Абрамов, С.* и др. Задачи по программированию. Москва: Наука, 1988.
2. *Абрамов, С.*, *Зима, Е.* Начала информатики. Москва: Наука, 1990.
3. *Азъмов, П.* Информатика за 9. клас, профилирана подготовка. С.: Просвета, 2001.
4. *Азъмов, П.* Фортран в примери и задачи. С.: Народна просвета, 1985.
5. *Асенова, П. Келеведжиев, Е.* Информатика за 9. клас, задължителна подготовка. С.: Регалия 6, 2001.
6. *Богданова, В.*, *Момчева, Г.* Структуриране на учебното съдържание в извънкласните форми по информатика. – Математика и информатика, 2001, № 1.
7. *Бърнев П.*, *Азъмов П.* Алгоритми. С.: Народна просвета, 1978.
8. *Бъчваров, Ст. и др.* Задачи по програмиране с решения на Паскал. С.: Университетско издателство, 1989.
9. *Дрейфус, М.*, *Ганглоф, К.* Практика программирования на фортране. Москва: Мир, 1978.
10. *Липский, В.* Комбинаторика для программистов. Москва: Мир, 1980.
11. Най-доброто от списание Byte. Специално издание на BYTE Bulgaria. С.: TopTeam Co., 1997.
12. *Наков, П.* Основи на компютърните алгоритми. С.: TopTeam Co, 1998.
13. *Наков, П.*, *Добриков, П.* Програмиране =++ Алгоритми. С.: TopTeam Co, 2003.
14. *Симов, Г.* Програмиране на C++. С.: Сим, 1993.
15. *Стоянов Цв.*, *Порязов, Ст.* Информатика за 9. клас, профилирана подготовка. С.: Архимед, 2002.
16. *Тодорова, М.* Програмиране на C++, част първа. С.: Сиела, 2002.
17. *Шень, А.* Программирование: теоремы и задачи. МЦНМО, 1995.
18. *Шилдт Х.* Езикът C, практически самоучител. С.: СофтПрес, 2001.
19. *Шилдт Х.* Езикът C++, практически самоучител. С.: СофтПрес, 2001.
20. *Brassard, G.*, *Bratley, P.* Fundamentals of Algoritmics. Prentice-Hall, 1996.
21. *Sedgewick, R.* Algorithms in C++. Addison-Wesley Publ., 1992.
22. *Sullivan, K.* The Big Red Book of C. Sigma Press, UK, 1985.
23. `http://infoman.musala.com/`
24. `http://olimpiada.com.ru/`

СЪДЪРЖАНИЕ

ПРИЛОЖЕНИЯ 183-220

Емил Келеведжиев
Зорница Дженкова

АЛГОРИТМИ, ПРОГРАМИ И ЗАДАЧИ
Ръководство за начална подготовка по информатика за олимпиади и състезания

Рецензент: Стоян Капралов
Редактор: Мария Маникова
Графично оформление: Николай Лазаров Цачев

Българска. Второ допълнено издание
Формат 70 × 100/16. Печатни коли 14

Издателство „Регалия 6“
1113 София, ИМИ, ул. „Акад. Г. Бончев“ бл. 8
тел. 979 38 42
e-mail: regalia@abv.bg