Езикът за програмиране U

Бойко Банчев

0 Увод

U е малък динамичен функционален език за програмиране. Той е замислен като средство за удобно и кратко изразяване на алгоритми върху числа, текст и разнообразни абстрактни структури от данни. Основен вид структура, чрез който се представят съставни стойности, е редицата. Тя може да съдържа всякакви налични в езика стойности, включително и други редици. U разполага с множество вградени функции за образуване, преобразуване, извличане на части и други действия с редици. Повечето от тези функции са приложими и към текстови низове.

Особена черта на U е възможността за работа с равнинни фигури. Фигурите се построяват от по-прости фигури чрез няколко вида композиране, следвайки и развивайки идеята за т. нар. *функционална геометрия*[1]. Програмирането и показването на такива фигурни построения е подходящо за онагледяване на различни математически и информационни структури, както и за други цели.

Макар че U има редица черти, които, доколкото ми е известно, не се срещат в другите езици за програмиране, той съдържа и немалко както съзнателни, така и неосъзнати заимствания. Добре забележимо е влиянието най-вече на масивовите и други функционални езици: J, K, Nial, FP, ML и Haskell.

Особена роля за възникването на U има GeomLab[2] – език за начално изучаване на програмиране във функционален стил, който съдържа и въплъщение на функционалната геометрия. Увлечението и задоволството от заниманията ми с GeomLab се превърнаха в подтик за създаването на U, въпреки че самият език не повлия съществено на резултата.

Какво в случая на U означават думите *функционален*, *динамичен* и *малък*, с които го представих в самото начало?

Функционален. Действието на програмата се състои изключително в пресмятане на изрази за получаване на стойности. За образуване на изрази се използва голям брой вградени функции и се съставят нови. Когато дадена функция се прилага, наричаме я още *операция*.

Функциите, освен носител на действие, са и стойности. Както всяка друга стойност, функция може да бъде получена като резултат от действието на функция, предавана като

[1] `http://dx.doi.org/10.1145/800068.802148` или `http://www.ecs.soton.ac.uk/%7Eph/funcgeo.pdf`; обновен вариант: `http://www.brics.dk/%7Ehosc/local/HOSC-15-4-pp349-365.pdf`.

[2] `http://www.cs.ox.ac.uk/geomlab`

аргумент на функция и съхранявана като част от съставна стойност. Прилагат се разнообразни схеми (също функции) за образуване на нови функции от дадени. Едни от най-непосредствените са композирането и частичното прилагане (специализиране) на функции. Други изразяват вариантност, повторност, обхождане, натрупване и други полезни в пресмятанията типове на поведение.

Отсъстват познати от други езици понятия като променливи с менящи се стойности и оператори (команди). Веднъж създадено, привързването на име към стойност в даден контекст остава неизменно. За изразяване на различни форми на повторност вместо цикли разполагаме с функции, представящи нужните действия на по-високо смислово равнище, или прибягваме до рекурсия.

Всяка функция има единствени аргумент и резултат, но те могат да бъдат произволно сложни стойности. В хода на пресмятанията могат да се образуват и третират като цялостни обекти съставни стойности с разнородно съдържание и произволно голям обем. Като правило стойностите са неизменни: дори когато изглежда, че променяме елемент на съставна стойност, всъщност образуваме нова такава стойност.

Въпреки сходството с масивовите функционални езици редиците в U не са масиви, а структура от по-общ вид. Разлики от тези езици има също по отношение на третирането на функции (по-свободно и еднообразно в U), както и в други особености.

Динамичен. Езикът се смята за динамичен, когато голяма част от свойствата на програмите на него се проявяват по време на изпълнението, а не са предопределени от текста. Например типът на стойност на израз в U може да бъде неопределен в текста, тъй като функциите могат да произвеждат различни по тип стойности при различни свои пресмятания – в такива случаи типът става ясен едва с появата на съответната стойност в хода на пресмятането. В частност, неопределен може да бъде типът на аргумента на една или друга функция.

U е динамичен и поради своя интерпретативен и диалогов характер. Образуването на стойности с произволен размер и строеж, включително на нови функции, в хода на изпълнението също е характерно динамична черта. Заделянето и освобождаването на памет за стойности е изцяло динамично и автоматично (неявно). Свързването на имена със стойности също е динамично, но и различно от присвояването в императивните езици.

Малък. Това означава, че U има много малко на брой конструкции. Синтаксисът му е прост, в подходяща степен и еднообразен, и адаптивен, а чрез това – много лесен за овладяване и удобен за четене и писане. Лексикалното излишество в програмите, обратно на грамадното мнозинство от други езици, е много ниско.

Еднообразието е водеща черта в U и се проявява в следното.

- За образуване на съставни стойности се използва единствен механизъм – редиците.[3] Те играят ролята на различни линейни и нелинейни структури от данни, които в други езици се срещат поотделно: динамичен масив, списък, n-ка, стек, опашка, дърво. Редиците опростяват и обръщенията към функции (виж следващата точка).
- Всяка функция има единствен аргумент, а той може да бъде всякаква стойност. В частност, с аргумент редица може да се зададе какъв да е фактически брой аргументи.
- Всяко действие се представя чрез пресмятане на израз, а всеки израз се състои от обръщения към функции, т. е. прилагания на операции.
- Всички прилагания на операции имат инфиксна форма. Същата синтактична форма има и частичното прилагане – това на функция към частично определен аргумент.
- Изразите се пресмятат строго последователно отляво надясно.

Всяка вградена функция извършва добре определено просто действие, а и броят им е неголям – примерно такъв, какъвто е в езика C. Много от вградените функции имат небуквени имена, които са внимателно подбрани така, че да бъдат максимално мнемонични. При желание наред с тези имена могат да се въведат и използват словесни. Такива примерни синоними дори са предложени в определенията на функциите.

Овладяването и използването на U е подобно на това на прост, но изразителен говорим

[3] Без да броим низовете, които се състоят от литери, и фигурите, които се състоят от други фигури.

език. Правилата за съставяне на смислени фрази са възможно най-прости, а усвояването – в случая и измисляне – на повече и по-изразителни думи води до по-съвършен изказ.

U е малък и заради това, че е предвиден за самостоятелна употреба. Той може да се използва като мощен програмируем калкулатор, за самообучение и обучение, за експериментиране и изследване и др. под. В него не са предвидени средства за едромащабно програмиране, такива, които да улесняват програмиране в колектив, както и ред други желателни в производственото програмиране свойства.

1 Програма

Програмата на U е редица от свободни и свързани в определения изрази. Те следват в произволен брой и ред, като всеки два се разделят със знак `;`. Разполагането на частите на програма по редове и в рамките на ред е несъществено за правилността на текста и се избира както се сметне за удобно.

Изпълнението на програма става, като интерпретаторът (след отстраняване на коментарите) пресмята последователно съставящите я изрази. Стойността на свободен израз се показва[4] непосредствено след пресмятането му, а определенията водят до свързване на име или имена със съответните стойности. Ако така определено име се появява в следващите изрази, то обозначава свързаната с него стойност.

Определенията биват прости и блокови и всяко се предшества от знак `++`. Простото определение има вида

шаблон `=` *израз*

където *израз* е свободен израз, а шаблонът чрез съпоставяне със стойността на израза свързва едно или повече имена с тази стойност или части от нея. Следното е пример за програма:

```
++ x = 0,6;         " дава име x на редицата [0;1;2;3;4;5]
++ a/b = x;         " имената a и b се свързват с [0;1;2;3;4] и 5
9*_!a               " пресмята израз, в който участва a
```

Блоковото определение има вида[5]

`{` *определения* `++` *определения* `}`

където определенията от първата група са основни и биват уточнени чрез тези от втората група, помощни за тях. Основните определения са само прости, а помощните могат да бъдат и блокови. Конструкцията създава за по-нататъшно използване свързванията на имена със стойности, посочени чрез основните определения, а помощните могат да се цитират едно друго и в основните определения, но не извън блока – спрямо него те са локални. Във всяка група определенията се разделят едно от друго чрез `;`. Например

```
{s = x+y; p = x*y ++ x = 3; y = x+7}
```

определя имената `s` и `p` със стойности 10 и 30 и това става чрез помощни определения за `x` и `y`. Тъй като последните са локални в блока, извън него всяко от тях може да има друго свързване или да бъде неопределено.

Всички имена, включително и тези на функции, получават стойности по описания начин, посредством свързване в просто или блоково определение. За разлика от повечето други езици, в U не съществува отделен вид конструкция за именуване на функция. Функциите са само един от видовете стойности, а механизмът за именуване на стойности не зависи от вида им.

Освен определения, с `++` се задават и команди от вида

`++` *име на файл* .

Такава команда е равнозначна на появата на нейно място на съдържанието на посочения файл. Името на файла се задава чрез низ. (Вж. по-долу за низове.)

[4] Някои стойности се отпечатват на екрана пряко, други символично, а фигурите се рисуват

[5] Виж също конструкцията *блоков израз* в **Затворени изрази**

Коментар се задава със знака " (двойна права кавичка), след която няма буква. Различават се два вида коментар. „Дългият“ започва с ред, в началото на който стои "{, и продължава включително до ред, започващ с "}, или до края на текста, ако такъв ред няма. Коментар, който не е дълъг, се нарича „къс“ и продължава до края на реда. Дълъг коментар може да бъде вложен в друг такъв. При изпълнението на програма коментарите се пропускат, като от текста се елиминират първо дългите, а после късите.

От казаното произтича следната особеност. Ако в дадена точка от текста на програмата няма незавършен дълъг коментар и се среща ред, започващ с "}, той сам по себе си е къс коментар. Предвид това, действието на дълъг коментар може да бъде отменено, като добавим в началото му ": ""{ става къс коментар, следващите редове – действителна, некоментарна част от програмата, а най-близкият "} вече е не край на дълъг коментар, а къс коментар. Ако обаче даденият дълъг коментар е вложен в друг действащ, отменянето му ще доведе до това външният коментар да завършва преждевременно – там, където е завършвал вложеният. Това променя съдържанието на програмата, освен когато краищата на вложения и на обемащия коментари са съседни.

2 Стойности

Всяка стойност в езика U има някой от следните типове:

числа	низове	фигури	неопределено
булеви	редици	функции	

2.1 Числа

Числата са 64-разредни с плаваща точка.[6] Това включва възможността за точно смятане с цели числа, чиято абсолютна стойност не надхвърля 10^{14}.

Имената $pinf и $ninf обозначават положителна и отрицателна „безкрайни“ числови стойности.

2.2 Булеви

Булевите стойности са „истина“ и „неистина“, представени съответно чрез имената $t и $f. Освен булеви операции, към тях могат да се прилагат и числови – аритметични и сравнения – и тогава $f е равнозначно на 0, а $t на 1.

2.3 Низове

Низовете са редици от литери – букви, цифри, препинателни и др. елементи на някаква азбука. Низ с непосредствено зададено съдържание се записва, като се огради в кръгли скоби, предшествани от ' (единична права кавичка). Например '() е празен низ, а сред елементите на низа '(...(...) е литерата (.

В съдържанието на такъв низ знакът ' има допълнителна роля, ако е следван от коя да е от литерите n,) и ' или е в края на реда. Съчетанието 'n в низ обозначава „край на ред“, ') и '' представят) и ', а в края на реда ' се тълкува като знак за пренос на низа: той продължава на следващия ред.

Особен случай за ' е и когато следващият знак в низа е #, а след него има четири 16-ични цифри: тези общо шест знака представят елемент на низа, чийто код от Уникод е четиризначното число.

[6] В записа на число може да участват знаците . (десетична точка) и - (минус за отрицателни числови константи). Възможно е и пропускане на дробната или цялата част на число, ако тя е 0, напр. 123. или .123. Понеже обаче . и - са и операции, възниква потенциално двусмислие между двете им роли. То се отстранява от следното правило. Знакът . или - се смята за начало на числова константа само ако той стои непосредствено в началото на ред, след шпация или непосредствено следва някой от знаците (, [, и {, а самият той е непосредствено следван от цифра. Точката е част от запис на число и когато стои между цифри или в края на низ от цифри, стига цифрите преди нея да не са част от константа, в която вече се среща точка.

Във всички останали случаи, включително и когато '# присъства, но не е следвано от четирицифрен код, знакът ' представя сам себе си. Например записът '(')rock''n'roll) отговаря на низа)rock'n'roll.

Освен косвено чрез 'n, литерата „край на ред“ може да бъде включена в низ и пряко, като запишем низа на повече от един реда (в този случай в края на реда, след който низът продължава, не трябва да стои ').

За някои низове се допуска съкратено представяне.

Низ, който се състои само от една, различна от (литера, може да се запише без скоби. Така например '', '", 'n и ') са низове с по един елемент, съответно ', ", n и). Литерата (, дори единствена в низ, не може да бъде представена съкратено, а само в скоби: '(().

Без скоби може да се запише и низ с един елемент, ако последният е представен шестнайсетично и е предхождан от '#, както в съдържанието на низ в скоби. В този случай '# не представя низа '(#).

Низ, който започва с буква, състои се от букви и цифри и евентуално съдържа разделители – т. е. такъв, който изглежда като *име* (вж. по-надолу) – също може да се запише без скоби, като пред него се постави ", както в "x·24, "Toledo, "i18n и "a. Последното е същото като 'a или '(a).

2.4 Редици

Редица е наредена съвкупност от стойности. Непосредствено стойност редица се задава чрез квадратни скоби, между които се изброяват елементите ѝ. Всеки елемент се задава с израз, стойността на който е този елемент. Когато са повече от един, елементите се разделят със знака ;. Членове на редица могат да са всякакви стойности, включително и различни по тип и също редици. Празна редица се задава с [].

Елементите на редица могат да се цитират чрез последователни номера – индекси – $0, 1, \ldots$ и в обратен ред, от последния назад: $-1, -2, \ldots$. Прекият достъп до елементите им благоприятно отличава редиците от еднопосочно свързаните, с последователен достъп списъци в други езици. Редиците имат предимство и пред т. нар. динамични масиви в някои езици с това, че добавянето и отнемането на елементи е еднакво удобно във всеки от двата им края.

Строежът на редица може да бъде откриван чрез прилагане на подходящи функции или чрез структурно съпоставяне с шаблон в определение. Съпоставителното разграждане може да става във всеки от двата края на редицата.

Редиците са единственото, но използвано като универсално средство за агрегиране на стойности в U. Освен че непосредствено представят разнообразни линейни структури (масиви, списъци, n-ки, опашки и др.) и дървета, редиците могат да се използват и в ролята на асоциативни таблици, смеси и други съставни обекти. Съвкупности от редици могат да поделят елементи и да образуват произволно сложно свързани структури.

2.5 Фигури

Фигура е правоъгълник с хоризонтални и вертикални страни, който съдържа рисунка. Фигурата няма нито конкретно местоположение в равнината, нито дори размер, а се характеризира само (ако не обръщаме внимание на съдържанието) с *формат* – отношението на ширината към височината ѝ.

От гледна точка на начина на построяването им фигурите са прости и съставни. Простите се получават чрез обръщения към функцията picture, която дава възможност за рисуване чрез геометрични построения.

Съставните фигури се образуват от простите и от други съставни чрез обръщане или завъртане на дадена фигура или съединяване на две фигури. (Фигура се смята за съставна дори ако е образувана от само една проста, но подложена на въртене, обръщане или и едното, и другото.)

Съединяването на фигури се състои в поставяне на едната до, над или върху другата. Ако ги поставим една до друга височините им се изравняват, като всяка от тях запазва

формàта си, и така се образува нова фигура, чиито съдържание и формат се получават от двете дадени. При поставяне една над друга се изравняват ширините. Ако се поставят една върху друга е налице припокриване, а съгласуването на форматите е по-сложно.

Строежът на съставна фигура може да бъде изследван чрез структурно съпоставяне в определение, както за редици.

Съдържанието на фигура може да бъде цветно, а цветът на фигура може да бъде променян. Предвид възможността за припокриване фигурите могат да бъдат и полупрозрачни, с избираема степен на плътност.

2.6 Функции

Различните видове стойности се отличават помежду си с това какви действия, тоест функции, могат да се прилагат към тях. Функциите на свой ред също са стойности – към тях също биват прилагани различни функции. Например действието, което съединява две функции за прилагане една след друга – едно от най-важните действия за функции – образува от тях нова функция, а и самото то е функция. Самото прилагане на функция към аргумент – основното действие с функции, това, заради което те съществуват – е също функция.

Стойности функции в U имат имената на предопределените функции и затворените изрази-функции, а други се добиват чрез прилагане на функции.

Има множество предопределени функции. Тъй като редица действия удобно се онагледяват с познати или лесни за съобразяване знаци, имената на някои от тези функции не са буквени, а се задават с един или два други знака – напр. `+`, `<` и `<:`. Различното именуване не е свързано със свойствата и начина на използване на функциите, а само със съображения за нагледност на записа. Не са различни по свойства и начин на използване и образуваните в програмата функции.

Някои функции имат за аргументи функции, а някои и образуват функции. Всяка функция може да бъде приложена частично, при което също образува функция.

Стойностите функции съдържат собствени екземпляри от несобствените за тях локални променливи, когато има такива променливи.[7]

2.7 Неопределено

Стойността „неопределено“ се бележи с `$` и е предназначена да представя „отсъстваща“ стойност, например алтернатива на очакваната от някое пресмятане стойност, когато то завършва с неуспех.

3 Имена

Имената служат за означаване на стойности. Всяко име се състои от поне една буква, която може да бъде следвана от още букви, цифри и разделители. Буквите са тези от латиницата (по 26 главни и малки), кирилицата (по 33 главни и малки, вкл. буквите от руската азбука Ё, ё, Ы, ы, Э и э) и гръцката азбука (24 главни и 25 малки). Разделител е знакът · (U+00B7) и може да се използва за нагледно разделяне на името на части. В името може да има повече от един разделителя, но не съседни, нито в края му (разделителят *разделя*).

Свързването на име със стойност става чрез определение и може да бъде глобално или локално. В първия случай името обозначава съответната стойност в целия текст на програмата след определението, освен ако не се среща друго определение за същото име и с изключение на евентуални локални определения на това име. Ако следва друго глобално определение, за него от даденото място нататък важи същото.

Локално определение на дадено име отменя в рамките на съответната локална област действието на глобално или външно локално определение за същото име, ако има такова.

[7] Т. е., те са това, което се нарича *lexical closures* (англ.).

Някои числа, булевите константи и фигурата `$np` се наричат в езика със специални имена. Последните се състоят от една или няколко букви, предхождани от знака `$`, имат предопределени и неизменни стойности и са глобално достъпни, т. е. те са предопределени глобални константи. Към тях можем да причислим и стойността „неопределено“ (`$`).

4 Изрази

Както редиците са основна и най-важна структура за представяне на данни, изразите са основна и най-важна структура за описване на действия. Те могат да представят неограничено сложни, съдържателно твърде различни пресмятания, при това нагледно и в еднообразна форма. За да се постигне това, образуването и пресмятането на изрази е подчинено на няколко особености, с които U се различава от останалите езици.

4.1 Строеж и пресмятане

Всеки израз е поредица от елементи. Пресмятането на израз се състои в добиване на стойностите на елементите и извършване с тях на действия. Във всяко действие участват три последователни стойности $u\ v\ w$, а действиетосе състои в прилагане на функцията v към u и w. Понеже и v е стойност, а и стойностите u и w също може да са функции, стойността-функция, която бива прилагана, наричаме *операция*. (Другите две стойности, каквито и да са по вид, разглеждаме като аргументи на операцията.) Понятието *функция* се отнася до вида на една или друга стойност, а *операция* – до мястото и чрез него ролята на дадена стойност в израз.

Така понятието операция в U има синтактичен смисъл, а всяко прилагане на операция е инфиксно.

Елементите на израз се пресмятат строго последователно. За всяка получена по този начин тройка стойности $u\ v\ w$ веднага след намирането на w се извършва прилагане на операция, както бе описано. Получената от това пресмятане стойност се разглежда като стойност на нов елемент на израза на мястото на изходните три: пресмятането на израза продължава по-нататък с нейно участие.

Първичните елементи на израз са непосредствено зададени стойности, например числа, низове, функции и др., заградени в скоби подизрази и затворени изрази. Подизразите в скоби се пресмятат като самостоятелни изрази. Така е и за затворените изрази, но те имат особен начин на пресмятане, както е описано в следващия раздел.

Съгласно описаното правило за пресмятане, всеки израз е поредица от нечетен брой елементи $a\ b\ c\ d\ e\ f\ g\ \ldots$, равнозначна на $((a\ b\ c)\ d\ e)\ f\ g\ \ldots$, в която на четни места стоят операциите (b, d, f, $\ldots$), а на нечетни – техните аргументи (a, c, e, g, $\ldots$). Това правило еднозначно определя точния ред на пресмятане и не използва правила за относителни предимства между операциите.[8]

4.2 Аргументи и прилагания

Въпреки че видът на прилагането на операции подсказва друго, в U всяка функция има *точно един аргумент*. За да приема дадена функция две или повече стойности, образуваме от тях редица: аргументът на функцията е тази редица, а нейните членове условно приемаме за отделни аргументи на функцията. В този смисъл и празната редица може да се приема за отсъствие на аргумент.

Основният начин да пресметнем функция f за каква да е стойност x е да приложим операцията `.` към f и x, записвайки $f\,.\,x$. Функцията `.` също има единствен аргумент, а горното всъщност означава, че този аргумент е редицата `[`f`;`x`]`.

[8] В други езици за програмиране се приема, че операциите са разбити на групи по предимства, например умножението и деленето са преди събирането и изваждането. За операции от една група пък се избира ляво или дясно асоцииране (съдружаване). Въпреки възникващото поради самата нея усложнение в определението на езика, дори такава двуслойна система от предимства не отстранява напълно нееднозначността в пресмятането на израз, ако не е посочен редът на пресмятане на аргументите на отделно взета операция.

В сила е по-общото правило: всяко прилагане $u\ v\ w$ се тълкува като прилагане на v към редицата `[`u`;`w`]`, т. е. $u\ v\ w$ е равнозначно на v `.[`u`;`w`]`.[9]

Така на разположение са два стила на запис. При първия операцията е `.`, като в f`.`x, и се разглежда като „префиксен“, в смисъл на прилагане на f към x. При другия стил функцията се поставя между „два аргумента“ (всъщност два члена на редица-аргумент) и затова го възприемаме като инфиксен. В действителност инфиксни по форма са и двата стила, а разграничаването им е само условно.

Изборът между единия и другия вид запис се прави според това, кой за даден случай смятаме за по-удобен. Редица прилагания се записват изключително в инфиксен стил, напр. пишем `p*q+r<s`, макар че възможни са и записите `<.[+.[*.[p;q];r];s]` и `<.(+.(*.(p\q)\r)\s)` (сходни с префиксния `(< (+ (* p q) r) s)` в езика Lisp).

Различаването на функции като едноместни, двуместни и други е условно не само защото всъщност аргументите са членове на една стойност – редица, а и защото броят на аргументите може да бъде неявен. Например в израза `-.z` функцията `-` може да бъде както обръщане на аритметичния знак („едноместна“), така и изваждане („двуместна“) според това дали `z` е едно число или редица от две. Всъщност тук тя е нито едноместна, нито двуместна, а операцията `.`, на която е аргумент, отговаря за правилното използване на стойността на `-` според тази на `z`.

Да подчертаем, че третирането на стойност като операция или аргумент изцяло зависи от мястото ѝ в израза, а не от това дали е функция. Например в израза `2-5` името `-` обозначава не само функция, а и операция, докато в `-.[2;5]` то сàмо обозначава стойност и самò по себе си не води до пресмятане. Операцията `.` е тази, заради която се изисква стойността на `-` да е именно функция и която предизвиква пресмятането на тази функция.

4.3 Частично прилагане

Записите от вида `_` $f\ e$ или $e\ f$ `_`, където e и f са изрази и стойността на f е функция, следват фòрмата на инфиксно прилагане и образуват стойности, така че също са изрази, но с една особеност: „аргументът“ `_` не се пресмята, той е само формален знак. С помощта на такива изрази получаваме *частично прилагане* на функцията f към наличния аргумент, съответно десен или ляв. Резултатът е функция от липсващия аргумент – този, който следва да е на мястото на знака `_`.

С други думи, функцията `_` $f\ e$ е такава, че `_` $f\ e$`.` m е равнозначно на $m\ f\ e$, а $e\ f$ `_` – такава, че $e\ f$ `_.` m е равнозначно на $e\ f\ m$. Например `_:10` е функция, която дели на 10 и значи `(_:10).2` е 0,2, а `10:_` е функция, която дели 10, така че `(10:_).2` дава 5.

4.4 Затворени изрази

Има три вида изрази, особени с това, че образуват стойности не чрез непосредствено прилагане на операции, а по специфичен начин. Всеки от тях има изрично зададени чрез скоби `{` и `}` начало и край.

Блоковият израз има вида

`{` *израз* `++` *определения* `}`

където частта след `++` съдържа локални определения, разделени помежду си с точка и запетая (`;`). Стойността на блоков израз е тази на посочения в него израз. Локалните определения могат да се използват едно друго и в този израз, но не другаде. Те могат и да не са прости, а да съдържат локални за тях определения.

Условният израз има общ вид

`?{` *условие*$_1$ `::` *израз*$_1$
`;` *условие*$_2$ `::` *израз*$_2$
`;` ...

[9] В други езици за програмиране за обръщение към функция е предвиден особен запис като `f(x)`, `f[x]` или `f x`, който синтактично го отличава от прилагането на операция в израз. В U това действие се извършва *само* чрез изрично прилагане на операция, запазвайки еднообразността на пресмятанията.

; *израз*$_n$
++ *определения* }

Условията се пресмятат последователно, докато се намери истинно. Стойност на условния израз тогава става тази на следващия условието израз. Ако всички условия се окажат неистинни, стойност на условния израз е тази на допълнително зададения *израз*$_n$. Ако такъв няма, получава се `$`.

Частта, започваща с `++` е незадължителна и съдържа локални, помощни за пресмятането на условията и изразите определения, както при блоковия израз.

Изразът-функция произвежда функционна стойност чрез задаване на параметризирана функция (λ-израз). Общият му вид е

`@{` p_1 `(` g_1 `) ::` e_1
`;` p_2 `(` g_2 `) ::` e_2
`; ...`
`++` *определения* `}`

p_1, p_2 и т. н. са шаблони, g_1, g_2, ... – условия, т. е. булеви изрази, а e_1, e_2, ... – изрази от общ вид. Аргументът при повикването на функцията се съпоставя с p_1. Ако това е успешно, проверява се условието g_1 и ако и то се окаже изпълнено, пресмята се e_1 и неговата стойност става стойност на функцията. Ако съпоставянето или проверката не успеят, аналогично се разглежда втората двойка p_2 g_2 и при успех тогава се произвежда стойността на e_2, а при неуспех се продължава със следващата клауза и т. н. Ако при никоя от двойките шаблон-условие не се стигне до успех, функцията произвежда стойността `$`.

Шаблоните в израз-функция имат същия строеж като тези в определение и при успешно съпоставяне също могат да водят до свързване на имена със стойности, като при функции свързванията са локални в рамките на съответно пресмятания израз. Шаблоните във функция обаче могат и да не съдържат имена и така да не водят до свързване – пресмятаният при успешно съпоставяне израз може да няма нужда от него. Освен това се допуска и неуспешно съпоставяне.

Шаблон в израз-функция не се огражда в скоби, но може да се използват скоби за обособяване на подизрази в него.

Всеки две последователни клаузи се разделят с `;`. Знакът `@` освен в началото се използва и в изразите e_1, e_2, ... като име на определяната функция. Също, `@@` се използва за цитиране на „външната функция“ в локални определения, `@@@` за двойно външната и т. н. Така се осигурява възможността за рекурсивни повиквания, които остават анонимни.

Частта, започваща с `++` е незадължителна и съдържа локални (помощни) за функцията определения, подобно на блоковия и условния изрази.

Шаблон, условие или израз могат да бъдат пропуснати. При отсъствие на шаблон или условие, взима се шаблонът, съответно условието от най-близката предходна клауза, която има дадения елемент. Ако в дадена клауза отсъстват и шаблонът, и условието, всяко от тях се „наследява“ от някоя предходна клауза.

Ако при пропуснат шаблон отсъства и предходна клауза с шаблон, за такъв се взима `_`. Ако при пропуснато условие отсъства и предходна клауза с условие, взима се `$t`.

Когато условието в дадена клауза отсъства, пропускаме и ограждащите го скоби.

При пропуснат израз се пресмята този от най-близката следваща клауза, в която има израз, а ако такава клауза няма, резултатът е `$`.

Допустимо е и трите елемента в дадена клауза да отсъстват, но такава клауза с нищо не е полезна.

Пример. Функция за намиране на факториел – три варианта:

```
@{n::?{n=0::1;@.(n-1)*n}}
@{0::1; n::@.(n-1)*n}
@{n::*>(1,,n)}
```

Пример. Функция за намиране на корените на квадратно уравнение. Резултатът е редица от две числа, само едно число или `$` (няма корени):

```
@{[a;b;c] (d>0):: {[-.b-sd:a2; -.b+sd:a2] ++ sd = d^0.5}
 ;        (d=0):: -.b:a2
 ++ d = b^2-(4*a*d); a2 = 2*a}
```

Пример. Функция за намиране на простите числа, не по-големи от дадено цяло положително (решето на Ератостен) – итеративен и рекурсивен варианти:

```
@{n :: f beyond (~.empty<fst).[n,,2;[]].1 . 1
  ++ f = @{[xs/p;ps]::[@{x::x>:p.1>0}%xs; ps/p]}}

@{n :: @{[]::[]; xs/p::@.(@{x::x>:p.1>0}%xs)/p}>~.(n,,2)}
```

Забележка. Когато при пропускане на условие или израз те се заместват с друго условие или израз, последните може да съдържат променливи, несвързани от текущия шаблон. Ако някоя такава променлива не е свързана и в обхващащия контекст, изразът и съответно функцията произвеждат `$`.

Определението на функция може да бъде направено така, че тя да приема всякакви по дължина и съдържание редици или само някои, както и да има различно поведение според дължината на редицата и типовете на елементите ѝ. Това дава възможност да съставяме функции, които на практика съвместяват няколко различни – т. нар. полиморфни функции.

5 Функции с несловесни имена

За всяка функция е посочено името ѝ – един или два знака – следвано от букви за съответния брой и ред на аргументите. Всяка буква отговаря на типа на аргумента, а именно: n – на число, i – на цяло число, b – на булева стойност, q – на редица или низ, s – на низ, p – на фигура, f – на функция, а x – на аргумент от какъв да е тип.

`+` n_1 n_2
`+` q_1 q_2
: **Плюс**: в първия вариант е сбор n_1+n_2 на аргументите, а във втория – конкатенация, т. е. редица или низ от елементите на q_1, следвани от тези на q_2.

`-` n
`-` n_1 n_2
`-` q_1 q_2
: **Минус**: в първия случай е $-n$, във втория – n_1-n_2. В третия е редица, получена от q_1, като за всеки член на q_2 от q_1 е премахнат първият срещнат равен на дадения член (ако има такъв).
Пример. `"mississippi-"dismiss` $\Longrightarrow$ `"sippi`

`-` p
: **ОбрХ**: обръщане на фигурата p спрямо хоризонталата.

`-` p_1 p_2
: **Над**: фигура, образувана от поставяне на p_1 вертикално над p_2.

`*` n
: **Знак**: аритметичен знак на n: $\begin{cases} -1, & n<0 \\ 0, & n=0 \\ 1, & n>0 \end{cases}$.

`*` n_1 n_2
: **Произведение**: $n_1 n_2$.

`*` q i

Редица от съдържанието на q, изброено i пъти едно след друго.
Пример. `[2;5]*3` $\implies$ `[2;5;2;5;2;5]`

`^` n_1 n_2
`^` f i

Степен. В първия случай – $n_1^{n_2}$, а във втория – функция, която е i-кратното прилагане на f (т. е. f^i).
Пример. `(_+[1;2]^3).[777]` $\implies$ `[777;1;2;1;2;1;2]`
Пример. `@{x::'<\x/'>}^3."xy` $\implies$ `'(<<<xy>>>)`

`^` p n

Върти: фигура, получена от p чрез въртене с големина n пъти прав ъгъл, в положителна или отрицателна посока според това дали $n \geq 0$ или $n \leq 0$.

`:` n

Реципрочно на n.

`:` n_1 n_2

Частно: $\frac{n_1}{n_2}$.

`:` q i

Късове: Редица от i на брой редици, на които е разделена редицата (или низа) q. За дължините l_1 и l_2 на всеки два съседни отрязъка (l_1 преди l_2) е изпълнено $0 \leq l_1 - l_2 \leq 1$.
Пример. `"abcdefghijkl : 5` $\implies$ `["abc;"def;"gh;"ij;"kl]`.

`:` i q

Отрези: Редица от първите i на брой члена на q, първите i от останалите и т. н. до изчерпването им (последните са евентуално по-малко от i на брой).
Пример. `5 : "abcdefghijkl` $\implies$ `["abcde;"fghij;"klm]`.

`<:` n

Отдолу: най-голямото цяло число, по-малко или равно на n; $\lfloor n \rfloor$.

`<:` n_1 n_2

ЧОдолу: частно и остатък при целочислено делене n_1`:`n_2 – закръглянето на частното става с $\lfloor\ \rfloor$.
Пример. `34<:-5` $\implies$ `[-7;-1]`
Пример. `-1.2<:1` $\implies$ `[-2;.8]`

`>:` n

Отгоре: най-малкото цяло число, по-голямо или равно на n; $\lceil n \rceil$.

`>:` n_1 n_2

ЧОгоре: частно и остатък при целочислено делене n_1`:`n_2 – закръглянето на частното става с $\lceil\ \rceil$.
Пример. `34>:-5` $\implies$ `[-6;4]`
Пример. `-1.2>:1` $\implies$ `[-1;-.2]`

`|:` n

Най-близко: най-близкото до n цяло число. Ако n е на еднакво разстояние от две цели числа, избира се четното.

`|:` n_1 n_2

ЧОблиз: частно и остатък при целочислено делене n_1:n_2 – закръглянето на частното става до най-близкото цяло.
Пример. `34|:-5` $\Longrightarrow$ `[-7;-1]`
Пример. `-1.2>:1` $\Longrightarrow$ `[-1;-.2]`

`< <= = <> >= >` x_1 x_2

Сравняване: проверка на отношението $x_1 < x_2$, $x_1 \leq x_2$, $x_1 = x_2$, $x_1 \neq x_2$, $x_1 \geq x_2$ или $x_1 > x_2$. Резултатът е `$t` или `$f`, съответно когато отношението е изпълнено и когато не е. Аргументите са или числа (вкл. приравнени на тях булеви стойности), или низове. Низовете се сравняват лексикографски. Функциите `=` и `<>` се прилагат и към други по тип стойности – тогава те проверяват дали x_1 и x_2 са идентични.

`<` i q

Челни: редица или низ от челните i елемента на q при $i \geq 0$ и от всички без челните $-i$ елемента при $i < 0$.
Пример. `3<"abcdefgh` $\Longrightarrow$ `"abc`
Пример. `-3<"abcdefgh` $\Longrightarrow$ `"defgh`
Пример. Подредица от n елемента на q, започвайки от k-я: `n<(-.k<q)`

`<` f q

Назад: $\begin{cases} q_0, & q = [q_0] \\ q_0\ f\ (q_1\ f\ (\ldots\ f\ (q_{n-2}\ f\ q_{n-1})\ldots)), & q = [q_0;q_1;\ldots;q_{n-1}] \\ \$, & q = [] \end{cases}$

(Означенията в горното равенство са като за редици. За низове е аналогично.)
Пример. `-<[6;3;9;2]` $\Longrightarrow$ `10`. (Равнозначно на $6-(3-(9-2)) = 6-3+9-2$ – изобщо, `-<_` образува сбор с алтерниране на знака на членовете.)

`<` f_1 f_2

След: f_1`<`f_2 `.` $x \equiv f_1$ `.(`f_2 `.` x`)`.

`<` f

Свиване: обратното на разлагане по аргумент (виж по-надолу `>`). Примерно определение: `@{f::@{x\y::f.x.y}}`.
Пример. `<.(sum on _).[_^2;1;2;3]` $\Longrightarrow$ `14`.

`>` i q

Последни: редица или низ от последните i елемента на q при $i \geq 0$ и от всички без последните $-i$ елемента при $i < 0$.
Пример. `3>"abcdefgh` $\Longrightarrow$ `"fgh`
Пример. `-3>"abcdefgh` $\Longrightarrow$ `"abcde`

`>` f q

Напред: $\begin{cases} q_0, & q = [q_0] \\ ((\ldots(q_0\ f\ q_1)\ f\ \ldots)\ f\ q_{n-2})\ f\ q_{n-1}, & q = [q_0;q_1;\ldots;q_{n-1}] \\ \$, & q = [] \end{cases}$

(Означенията в горното равенство са като за редици. За низове е аналогично.)
Пример. Функция, равнозначна на `^` за функции: `@{[f;n]:: > > ([f]*n/id)}`.
Пример. Пресмятане на полином („метод на Хорнър“):
`poly = @{[cs;x]:: @{[v;c]::v*x+c} > cs}`
`[2;3;-5;1] poly 2` $\Longrightarrow$ `19`. ($2x^3+3x^2-5x+1|_{x=2}$.)

`>` f_1 f_2

Преди: f_1`>`f_2 `.` $x \equiv f_2$ `.(`f_1 `.` x`)`.

`>` f

Разлагане на f по аргумент. Примерно определение: `@{f::@{x::@{y::f.(x\y)}}}`.
Пример. `{f.4 ++ dv = >.:; f=dv.100}` $\Longrightarrow$ `25`.

`|` n

Големина (абсолютна стойност) на n.

`|` b_1 b_2

Или: ако b_1 е `$f`, стойността на b_2, в противен случай `$t`.

`|` n_1 n_2

Максимум: по-голямото от n_1 и n_2.

`|` q_1 q_2

Обединение на редици-множества. Отначало се образува редица q, еднаква с q_1, след което към нея се добавя последователно всеки член на q_2, който не е равен на член на q. Така всеки член на q_2 попада в резултата не повече от веднъж, при това равните членове на q_2 не се повтарят. Ако в q_1 се срещат равни един на друг членове, те се срещат и в резултата.
Пример. `"star|"trek` $\Longrightarrow$ `"starek`.
Пример. `'()|"abracadabra` $\Longrightarrow$ `"abrcd`. Изобщо `[]|_` (`'()|_`) намира множеството от елементите на редица (низ): редица от всеки различен елемент по веднъж, в реда, в който те се срещат за първи път в дадената редица.

`|` p

ОбрВ: обръщане на фигурата p спрямо вертикалата.

`|` p_1 p_2

До: фигура, образувана от поставяне на p_1 отляво на p_2.

`|` f

Комутиране на функция. Примерно определение: `@{f::@{x\(y\r)::f.(y\(x\r))}}`.
Пример. `4(|.:)5` $\Longrightarrow$ `1.25`.

`|` x f

За: прилагане на функция към предхождащ аргумент; x `|` $f \equiv f\,.\,x$.
Пример. „Прокарване“ на стойност през верига от функции:
x `|` f_1 `|` f_2 `|` $\ldots$ `|` $f_n \quad\equiv\quad f_1$ `>` f_2 `>` $\ldots$ `>` $f_n\,.\,x \quad\equiv\quad f_n$ `<` $\ldots$ `<` f_2 `<` $f_1\,.\,x$.

`&` b_1 b_2

И: ако b_1 е `$t`, стойността на b_2, в противен случай `$f`.

`&` n_1 n_2

Минимум: по-малкото от n_1 и n_2.

`&` q_1 q_2

Сечение: на редици-множества. Образува се редица от последователно добавяните членове на q_1, които са равни на член на q_2. Ако в q_1 се срещат равни един на друг членове, които са равни и на член на q_2, те се срещат и в резултата.
Пример. `"aqua&"neutral` $\Longrightarrow$ `"aua`.

`&` p_1 p_2

Върху: фигура, образувана от поставяне на p_1 върху p_2.

`,` `,,` n_1 `[`n_2`;` n_3`]`

`,` `,,` n_1 n_2

Всяка от функциите `,` и `,,` образува редица от членове на аритметична прогресия с начало n_1, която при $n_1 < n_2$ е растяща, а при $n_1 > n_2$ – намаляваща (при $n_1 = n_2$ посоката е без значение). Стъпката на прогресията n_3 в първия вариант на командите се задава изрично, а във втория се подразбира: 1 при $n_1 < n_2$ и -1 при $n_1 > n_2$ (без значение при $n_1 = n_2$). В редицата-резултат се включват онези членове x на прогресията n_1, $n_1 + n_3$, $n_1 + 2\,n_3$, ..., за които е изпълнено условието от съответната клетка на таблицата:

	$n_1 \le n_2$	$n_1 \ge n_2$
`,`	$n_1 \le x < n_2$	$n_1 \ge x > n_2$
`,,`	$n_1 \le x \le n_2$	$n_1 \ge x \ge n_2$

При $n_1 = n_2$ за функцията `,` такива x не съществуват – тогава резултатът е празна редица.

Пример. `1,,[10;3]` $\Longrightarrow$ `[1;4;7;10]`.

Пример. `5,0` $\Longrightarrow$ `[5;4;3;2;1]`.

Пример. Редицата `1,,n` при $n > 0$ и `[]` при $n = 0$: `~.(n,0)` или `butf.(0,,n)`.

`#` q

Дължина (брой на елементите) на q.

Пример. `#.[3;5;[6;0;1;2];4]` $\Longrightarrow$ `3`.

`\` p

ОбрД1: обръщане на фигурата p спрямо диагонала северозапад-югоизток.

`\` x q

Оглави: редица или низ от x, следвано от елементите на q. Ако q не е редица, тълкува се като редица с единствен елемент q.

`/` p

ОбрД2: обръщане на фигурата p спрямо диагонала североизток-югозапад.

`/` q x

Добави: редица или низ от елементите на q, следвани от x. Ако q не е редица, тълкува се като редица с единствен елемент q.

`~` b

Не: отрицание на b.

`~` f

Отр: отрицание на f – ако за даден аргумент f дава `$t`, резултатът е `$f`, ако е `$f`, той е `$t`, а в останалите случаи – `$`.

`~` q

Обръщане: `[`q_{n-1}`;` ...`;` q_1`;` q_0`]`, където $q_0, q_1, \ldots, q_{n-1}$ са елементите на q. (Аналогично за низ.)

`!` f q

Всяко: `[`f`.` q_0`;` f`.` q_1`;` ...`;` f`.` q_{n-1}`]`, където $q_0, q_1, \ldots, q_{n-1}$ са елементите на q. (f`.[]` $\equiv$ `[]`.) Аналогично за низ.

Пример. Намиране на редиците от префикси и суфикси на дадена редица:
`@{xs::_<xs!(0,,(#.xs))}` и `@{xs::_>xs!(#.xs,,0)}`.

Пример. Извличане от редица q на 4-я, 2-я, първия равен на 0 и 5-я елементи: `q._![4;2;_=0;5]`.
Пример. Редица от елементите на q без тези, чиито индекси са в редицата x (изтриване на посочени по индекс елементи): `q._!(0,(#.q)-x)`.

`!` q_1 q_2

Някои: q_1 е редица от два члена: предикат p и функция f. Резултатът е редица (или низ), получена от q_2, в която всеки елемент x, за който е изпълнено $p.x$, е заменен с $f.x$.

`%` f q

Избор: редица или низ от всички елементи x на q, за които $f.x \equiv$ `$t`, взети в същия ред както в q. (f е булева функция.)
Пример. Редица от елементите на q без тези, които удовлетворяват предиката p: `~.p%q`.
Пример. Редица от индексите, за които елементите на редицата q удовлетворяват предиката p: `q._>p%(0,(#.q))`.

`%` q x_1 x_2
`%` q x_1

Редицата q се състои от редици от по два члена. Всяка двойка се смята за съответствие между стойности – втората съответства на първата (или първата е ключ към втората), а цялата редица се разглежда като асоциативна таблица. Функцията намира първата двойка, в която ключът е равен на x_1, и дава нейния втори член. Ако такава двойка няма, дава се стойността x_2, а ако тя отсъства – `$`.

`%%` f q

Надве: f е булева функция. Резултатът е редица от две редици (или низове) от членовете на q: в първата са онези, които удовлетворяват f, а във втората – останалите. Във всяка от двете редици относителният ред на членовете е същият като в q.

`||` q

Слоеве: редица от редици, съставени съответно от първите, от вторите и т. н. елементи на редиците, членове на q. Резултатът съдържа толкова редици, колкото е дължината на най-късата редица, член на q. В i-та поред редица се намират i-те членове на елементите на q, в същия ред, в който тези елементи са в q.

Ако всички редици в q имат еднаква дължина, функцията е подобна на \ за фигури – образува огледален образ на правоъгълна таблица спрямо диагонала между горния ляв и долния десен краища.

В общия случай функцията `||` съвместява ролите на функциите `transpose`, `zip`, `unzip` и подобните на последните две за повече аргументи, а f`!(||.`q`)` е аналогично на прилагане на `zipWith` и подобните на нея в езика Haskell.
Пример. Скаларно произведение на вектори `x` и `y`: `+>(*!(x||y))`.
Пример. Скаларно произведение на кои да е два вектора: `||>(*!_)>sum`.
Пример. Намиране на $\binom{n}{k}$ – броя на извадките от n предмета по k:
`@{[[a;b];c]::c*a:b}<({_+(n-k)!x||x ++ x=k,0}/1)`.

`.` f x

Към: прилагане на функция f към x.

`.` q i

На: елемент на q, намиращ се на номер (индекс) i. Ако елемент с такъв номер няма резултатът е `$`. Когато $i \geq 0$, броенето започва от началото (с 0 за първия елемент), а при $i < 0$ – от края на q (с -1 за последния елемент и намалява към началото).

. q f

Къде: най-малкото цяло число i, такова че $f\,.\,(q\,.\,i) \equiv$ `$t` (f е булева функция). Ако няма такова i резултатът е `$`.

6 Функции със словесни имена

Предопределените функции със словесни имена са описани в групи по предназначение. За посочване на аргументите им са използвани означения както за функциите с несловесни имена.

6.1 Функции за числа

`int` n

Цяла част на число („отсичане“, или закръгляне на n „към нулата“): $\lceil n \rfloor$.

`rat` n i

Най-близко до n рационално число, чийто знаменател не надминава i. Резултатът е редица с три члена: цялата част и числителя и знаменателя на дробната.

`gcd lcm` n_1 n_2

НОД и НОК. Функциите са приложими и за нецели числа.

Пример. `12 gcd 30` $\implies$ `6`.

Пример. `id on [gcd;lcm].[2.718;3.141]` $\implies$ `[0.009;948.582]`.

`diag` q

Диагонал на „хиперправоъгълник“ по „страни“, членове на q: $\sqrt{q_1^2 + \cdots + q_n^2}$.

`sin cos tan asin acos` n

`atan` n_1 n_2

Прави и обратни тригонометрични функции; `atan` има за аргумент `[`Δy`;`Δx`]`.

`log` n_1 n_2

Логаритъм от n_2 при основа n_1.

`random` n_1 n_2

Всяко от числата n_1 и n_2 се закръгля до най-близкото цяло. Тогава, ако n_1 е 2 или повече, функцията дава (псевдо)случайно цяло число в интервала $[0,\, n_1)$. В противен случай резултатът е реално число в интервала $[0, 1)$.

Ако $n_2 \geq 0$, числото резултат се избира от псевдослучайна редица, в която n_2 посочва някой член за начален. При $n_2 < 0$ такъв избор не се прави – взема се поредното число от редицата. (Всяко $n_2 \geq 0$ избира един определен член на псевдослучайната редица от числа за начален, а следващи обръщения към `random` с $n_2 < 0$ възпроизвеждат следващите членове на редицата.)

6.2 Функции за редици

`empty` x

Ако аргументът е празна редица или низ, дава `$t`. За стойности от друг тип, както и за непразни редица или низ дава `$f`.

`fst` q

Първият елемент на редица или низ: `_.0`.

`bst` q

Последният елемент на редица или низ: `_.-1`.

`butf` q

Редица или низ без първия елемент: `-1<_`.

`butb` q

Редица или низ без последния елемент: `-1>_`.

`rol` q

Циклично изместване на редица или низ наляво: `[`q_1`;...;`q_{n-1}`;`q_0`]`, където $q_0, q_1, \ldots, q_{n-1}$ са елементите на q. Равнозначно на `/on[butf;fst]`.

`ror` q

Циклично изместване на редица или низ надясно: `[`q_{n-1}`;`q_0`;...;`q_{n-2}`]`, където $q_0, q_1, \ldots, q_{n-1}$ са елементите на q. Равнозначно на `\on[bst;butb]`.

`cut` i q

Редица от две редици, които съединени дават q. Ако $i \geq 0$, първата от двете части има дължина i, а ако $i < 0$, втората част е с дължина $-i$.

`cut` f q

Редица от две редици, които съединени дават q. Първата редица съдържа възможно най-дългия начален отрязък от елементи на q, за които предикатът f дава стойност истина.

`update` q_1 q_2

Функцията има два варианта. В единия q_1 е редица от два члена – цяло число i и функция f. Резултатът е редица, получена от q_2, в която $q_2.i$ е заменено с $f.(q_2.i)$. (При недействителен индекс i резултатът е `$`.)

В другия вариант q_1 е редица от три члена – функция-предикат p, друга функция f и каква да е стойност u. Резултатът е редица, получена от q_2, в която първият елемент x, за който $p.x$ дава истина, е заменен с $f.x$. Ако такъв елемент няма, резултатът е редица, получена от q_2, в края на която е приписано u.

`before` q_1 q_2

В единия вариант на функцията q_1 е редица от два члена – цяло число i и каква да е стойност u. Резултатът е редица, получена от q_2, в която u е вмъкнато преди $q_2.i$. (При недействителен индекс i резултатът е `$`.)

В другия вариант q_1 е редица от три члена – функция-предикат p, друга функция f и каква да е стойност u. Резултатът е редица, получена от q_2: ако x е първият елемент на q_2, за който $p.x$ дава истина, $f.x$ се вмъква непосредствено преди него; ако такова x няма, u се приписва накрая на q_2.

`after` q_1 q_2

В единия вариант на функцията q_1 е редица от два члена – цяло число i и каква да е стойност u. Резултатът е редица, получена от q_2, в която u е вмъкнато след $q_2.i$. (При недействителен индекс i резултатът е `$`.)

В другия вариант q_1 е редица от три члена – функция-предикат p, друга функция f и каква да е стойност u. Резултатът е редица, получена от q_2: ако x е първият елемент на q_2, за който $p.x$ дава истина, $f.x$ се вмъква непосредствено след него; ако такова x няма, u се приписва в началото на q_2.

`count` f q

Брой на елементите на q, удовлетворяващи предикат f: `@{[p;xs]::#.(p%xs)}`.

`min max` q
`sum prod` q
`and or` q
`all any` q f

Обобщителни стойности за редица. Имат следните примерни определения и смисъл:
`min = _/$pinf>(&>_)` – най-малко от всички;
`max = _/$ninf>(|>_)` – най-голямо от всички;
`sum = _/0>(+>_)` – сбор от всички;
`prod = _/1>(*>_)` – произведение от всички;
`and = _/$t>(&>_)` – всички са равни на `$t`;
`or = _/$f>(|>_)` – поне едно е равно на `$t`;
`all = @{[q;p]::q.(~.p)=$}` – всички изпълняват предиката f;
`any = @{[q;p]::q.p<>$}` – поне едно изпълнява предиката f.
Функциите са приложими и към празна редица.

`in` x q

Проверява среща ли се стойността x сред членовете на редица q. Примерно определение: `@{[x;q]::q any(_=x)}`.
Пример. Функция, която проверява дали членовете на дадена редица са различни един от друг: `@{[]::$t; x\xs::?{x in xs::$f;@.xs}}`.

`scanl scanr` f q

Подобни на `>` (**напред**) и `<` (**назад**), но образуват редица от всички стойности, получени от прилаганията на f (виж подизразите в определенията на `>` и `<`).
Примерни определения:

```
scanl = @{[f;x\(y\xs)]:: x \ (f @ (x f y \ xs))
          ;[_;xs]      :: xs}
scanr = @{[f;xs/y/x]:: f @ (xs / (y f x)) / x
          ;[_;xs]    :: xs}
```

`from` f x

Редица, получена от последователни прилагания на f. Ако прилагането на f към x дава редица от две стойности, v и x', v се дописва към редицата-резултат и се продължава с прилагане на f към x'. Ако прилагането на f даде `$`, образуването на резултата завършва.

`iterate` f i x

Редица от $i+1$ степени $(0,\ldots,i)$ на прилагане на f към x – $[x;f(x);\ldots;f^i(x)]$: `@{[f;i;x]:: ?{i=0::x; f.(@.[f;i-1;x])}}`.

`eqseq` q_1 q_2

Проверява дали q_1 и q_2 са равни, т. е. имат еднакво съдържание.

`asprefix assuffix asinfix` q_1 q_2

Функциите откриват дали q_1 е равна съответно на начална, крайна или „средна“ подредица на q_2. Ако е така, `asprefix` дава редица от два елемента – съвпадащата част на q_2 и остатъка, `assuffix` – редица от началната и съвпадащата части на q_2, а `asinfix` – редица от началната, съвпадащата и остатъчната части.

`assort` q

Членовете на q се разпределят в множество редици; резултатът е редица от тези редици. Ако q_i е поредният елемент на q, а тези преди него са разпределени в редиците $s_1, s_2 \ldots, s_k$, последният елемент последователно на $s_k, s_{k-1}, \ldots, s_1$ се сравнява с q_i. Ако някое сравнение дава равенство, q_i се добавя в края на съответната редица. Ако не, q_i образува нова редица s_{k+1}.
(`assort` се използва за групиране по равенство или друг признак, за подреждане и други варианти на разпределяне.)
Пример. Намиране на „множеството на редица“ (виж също `|`): `assort>(fst!_)` .

`split` q_1 q_2
Образува редица от редици, всяка съставена от максимален брой последователни членове на q_1, несрещащи се в q_2.

`join` q x
Съединява редиците, членове на q, в една редица, като между края и началото на всеки две съседни подредици добавя x. (Участието в q на празни подредици води до поява на съответните места на съответния брой стойности x.) Примерно определение за редици: `@{[[];_]::[]; [q;u]::@{[x;y]::x/u+y}>q}` .
Пример. Подреждане по „бързия“ алгоритъм: `@{[]::[]; xs/x::@!(_<x%%xs) join x}` .

`flatten` x
Редица от атомарните стойности на x (плоска проекция на x). Обикновено аргументът е редица, но може да бъде и проста стойност. Примерно определение:
```
@{x:: [] f x
 ++ f = @{[rs;x] (type.x="seq):: @>(rs\x)
          ;       ($t)         :: rs/x}}
```

`cart` q
Декартово произведение на редиците (низовете), членове на редицата q. Примерно определение за редици:
```
@{[]    :: [[]]
 ;xs\xss:: + > ([]\(@{x::x\_!(@@.xss)} ! xs))}
```

`sort` f q
Редица от елементите на q, подредени в условно нарастващ ред, като за сравняване на стойностите се използва булевата функция f със смисъл на „проверка за по-малко“. Ако тази функция е `<`, подреждането е именно в нарастващ ред. Подреждането е устойчиво: равните елементи остават в същия относителен ред, в който се срещат в q.

`grade` f q
Редица от индексите на елементите на q в такъв ред, че ако всеки индекс се замени със съответния му елемент, получената редица е подредена точно както ако `sort` е приложена за същите f и q. Примерно определение:
`@{[f;q]::fst!(f on bst sort (0,(#.q)||q))}` .

6.3 Функции за текст

`ccvt` x
Аргументът може да бъде низ, редица или цяло число. Ако е низ, образува се редица от целочислените кодове на елементите му. Ако е редица, тя трябва да има за елементи целочислени кодове на литери, а резултатът е низ от тези литери. Ако аргументът е цяло число, образува се низ с един елемент – литерата с този код.

`frm` s x

Образува низ, печатно представяне на стойността x съгласно формат, зададен чрез низа s. Последният може да съдържа до четири части в следния ред:
признак за подравняване: един от `<`, `>` или `=`, съответно за поставяне на стойността в левия, в десния край или в средата на полето; незаетата част на полето съдържа интервали;
`+`: за отпечатване на `+` пред положително число (ако x е такова);
ширина на полето, в което се поставя стойността на x: цяло положително число;
брой десетични цифри след десетичната точка: цяло положително число, предхождано от точка (има смисъл само ако x е число).
Никоя от частите не е задължителна. Подразбира се поле с „естествена“ за стойността ширина, подравняване надясно и отсъствие на знак `+` за положителните числа.
Пример. `'() frm 2.56` $\Longrightarrow$ `'(2.56)`.
Пример. `'(<4) frm '(abc)` $\Longrightarrow$ `'(abc␣)`.
Пример. `'(=7) frm [3;5]` $\Longrightarrow$ `'(␣[3;5]␣)`.
Пример. `'(+.2) frm 27` $\Longrightarrow$ `'(+27.00)`.

6.4 Функции за функции

`id` x

Идентитет: `@{x::x}`. В частност служи за образуване на редица-двойка: `x id y` $\equiv$ `id.[x;y]` $\equiv$ `[x;y]`.

`const` x

Образува функция, която дава константа (независимо от аргумента си): `@{c::@{_::c}}`.

`when` f b

В зависимост от верността на условието b се избира или функцията f, или `id`: `@{[f;b]::?{b::f;id}}`.

`on` f_1 f_2

Функция, която прилага f_1 към редицата от резултатите от прилаганията на f_2 към всеки от членовете на редица от стойности. Равнозначна на `@{[f;g]::g!_>f}`.

`on` f q

Функция, която прилага f към редицата от резултатите от прилаганията на функциите от редицата q към дадена стойност. Равнозначна на `@{[f;q]::@{x::f.(_.x!q)}}`.
Пример. Средно аритметично: `:on[sum;#]`.
Пример. Функция от функции f и g, която, приложена към аргумент x, дава `f.[x;g.x]`: `f on [id;g]`.

`beyond` f_1 f_2

Функция, която преобразува дадена стойност чрез функцията f_1, докато получи такава, за която предикатът f_2 дава неистина. Резултатът е редица от две стойности: последната от получените, за която f_2 е истина и първата, за която е вярно обратното. Ако f_2 е неистина още за дадената стойност, резултатът съдържа `$` и тази стойност. Примерно определение: `@{[f;p]::$@{[u;v]::?{p.v::v@(f.v);[u;v]}}_}`.
Пример. Двата съседни члена a и b на аритметичната прогресия $x, x+d, x+2d, \ldots$, за които е изпълнено $a \le 100 < b$: `d+_beyond(_<=100).x`.
Пример. Редица от числата на Фибоначи, не по-големи от 1000:
`/on[id;2>_>+]beyond(bst>(_<=1000)).[0;1].0`.

`witheq` f x

Действието на функциите `=`, `-`, `&` и `|` за редици, `in`, `split`, `asprefix`, `assuffix`, `asinfix` и `assort` се основава на сравнение за равенство на прости стойности. По подразбиране то се извършва както при функцията `=` за прости стойности. Функцията `witheq` създава за изброените функции контекст, в който вместо `=` се използва двуаргументният предикат f. x е каква да е стойност: изразът, върху който действа `witheq` (в който повикванията на `=` и пр. действат изменено). `witheq` е псевдофункция, а тези, върху които тя действа са „нечисти“ функции в смисъл, че действието им зависи от контекста.

6.5 Функции за време

`time` x

Текущо време.

Ако $x =$ `'t`, резултатът е часово време и дата във вида `[[`h`;`m`;`s`];[`d`;`m`;`y`]]`. Ако $x =$ `'h` или $x =$ `'d`, резултатът е съответно `[`h`;`m`;`s`]` (час, минута, секунда) или `[`d`;`m`;`y`]` (ден, месец, година). Например `time.'d.0` е ден. Всички времеви стойности са цели числа.

Ако $x =$ `'s`, резултатът е текущо време в секунди, измерено от определена отправна точка.

Ако $x =$ `'p`, резултатът е дата и часово време във вид на текстов низ.

`delay` n

Пауза с посочена продължителност в секунди.

6.6 Функции за фигури

`picture` q_1 q_2

Построяване на фигура.

Редицата q_1 съдържа числа x_1, x_2, y_1 и y_2, които задават границите на правоъгълник: от x_1 до x_2 по хоризонтала и от y_1 до y_2 по вертикала. Правоъгълникът задава координатно пространство за чертане, а форматът на получаващата се фигура е $(x_2 - x_1) : (y_2 - y_1)$.

Редицата q_2 има нула или повече членове – самите те редици от вида

`[`*име*`;`*цвят*`;`*плътност*`;`d`;`*данни*`]` ,

всяка от които задава някакво построение. Името посочва вида на построението, а данните – координати, размери и други геометрични стойности. Стойностите за цвят и плътност са както за функциите `colour` и `opaque`. Ако се зададе `$` за цвят, построяваният обект се рисува в черно, но цветът може впоследствие да бъде променян с `colour`. Числото d е положително или 0. В първия случай се построява линия с дебелина d (в зададените от q_1 единици), а при $d = 0$ – площна фигура, чийто контур е тази линия.

Името се задава с буква.

`'P` отговаря на начупена или многоъгълник и в този случай данните са четен брой числа, които две по две задават координати на върхове.

`'C` или `"CC` задава кръгова дъга, сегмент или сектор. За построителните данни има две възможности – x_0`;` y_0`;` r`;` α`;` β или x_0`;` y_0`;` r`;` x_1`;` y_1`;` x_2`;` y_2. x_0, y_0 и r са съответно координатите на центъра и радиусът на дъгата, а началото и краят ѝ се посочват или чрез ъглите α и β, които сключват с хоризонталата лъчите от центъра на дъгата през тях (в първия вариант), или чрез точки върху тези лъчи (вторият вариант). Ъглите се задават в градуси. Дъгата е тази между двата лъча, движейки се от първия към втория по посока срещу стрелката на часовника. При α=0, β=360 се построява окръжност. Площният вариант на построението е кръгов сегмент (при `'C`), сектор (при `"CC`) или пълен кръг.

При `'B` се построява крива на Безие от втора или трета степен. Данните са съответно 6 или 8 числа – координати на 3 или 4 управляващи кривата точки.

При всички построения видима е само тази част, която попада в зададения с q_1 правоъгълник.

`bar` n

Фигура „греда“ – правоъгълник с формат n.

`blank` n

Прозрачен празен правоъгълник с формат n: (удобен за заемане на определено място без да променя рисунката в него). Примерно определение:
`0 opaque _ < ([1;1;1]colour _ < bar)` .

`arm` n_1 n_2

L-образна фигура с формат n_2/n_1. Левият край на фигурата е греда с формат $1/n_1$, долният – греда с формат n_2, а остатъкът от правоъгълника е прозрачен. Необходимо е да бъде изпълнено $n_1, n_2 \geq 1$. Примерно определение:
`@{[m;n]::?{m>1::bar.(:.(m-1))|?{n>1::blank.(n-1:(m-1));$np};$np}-(bar.n)}` .
Забележка. Двете греди се припокриват в левия долен ъгъл на фигурата и при $n_1 = 1$ фигурата е същата като при `bar.`n_2, а при $n_2 = 1$ – като при `bar.(:`n_1`)`.

`stretch` p
`stretch` p n

В първия вариант дава стойността на формàта за фигурата p. Във втория дава фигура, получена чрез разтягане на p, така че да добие формат, равен на n.

`colour` p
`colour [`n_1`;`n_2`;`n_3`]` p

В първата форма дава цвета на фигура: тройка числа, представящи съставките червено, зелено и синьо като стойности в интервала $[0, 1]$. Във втората форма придава и нова стойност на цвета. По подразбиране фигурите се оцветяват в черно – стойност `[0;0;0]`.

Тази функция се отнася само за неопределения цвят във фигура, който по подразбиране е черен (този, който във функцията `picture` е зададен с `$`). Части на фигурата могат да имат предварително фиксиран цвят, а и някои по начало неопределени може впоследствие да са оцветени с `colour`. Функцията оцветява само тези части, чийто цвят е останал неопределен или вече е бил задаван за същата фигура като цяло.

`opaque` p
`opaque` n p

В първата форма дава текущата плътност (непрозрачност) на фигурата – стойност в интервала $[0, 1]$. Във втората форма придава и нова стойност на плътността. Най-голяма плътност – пълна непрозрачност – се задава с `1`, а при 0 фигурата е невидима. Плътността на фигура има наслагващо действие спрямо тази на частите на фигурата: например, ако дадена фигура има плътност 0,8, а нейна част – 0,5, тази част добива плътност $0{,}8 \times 0{,}5 = 0{,}4$.

`draw` p q

q е редица от низ s и положителни числа w и h – големини в милиметри. Фигурата p се записва с посочен чрез w и h размер във файл с име s. Размерът е такъв, че фигурата заема максимално голям правоъгълник със същия като нейния формат, ширината и височината на който не надминават съответно w и h. Ако вместо w е посочено `$`, тази стойност не се взима под внимание: избира се правоъгълник с височина h. Ако вместо h е посочено `$`, избира се правоъгълник с ширина w. (Поне едното от числата трябва

да бъде зададено. Видът на файла, в който се записва, се определя от разширението на името му: `.eps` за EPS (Encapsulated PostScript) и `.svg` за SVG.) Функцията дава стойност `$t` или `$f` според това дали записването е успешно или не.

6.7 Функции за вход и изход

`write` *s q*

Записва едно след друго, разделени с интервали, текстовите представяния на стойностите от редицата *q*. Записът се помества във файл с име *s*. Ако файлът съществува, записът заменя съдържанието му. Ако *s* е празен низ, записва се в стандартния изходен поток. Ако вместо низ се зададе `$` записване не се извършва. Резултатът е низ – формираният запис, а при неуспех на действието – `$`. (Аргументът `$` е удобен за образуване на низ без той да се записва някъде.)

`writa` *s q*

Подобно на `write`, но ако се записва във файл и той съществува записът се добавя към съдържанието му.

`readf` *s q*

Чете текстов ред и извлича от него стойности. Източникът е файл с име *s* или, ако *s* е празен низ, стандартният входен поток. Редицата *q* посочва какви стойности да се извличат, като изброява типовете им – числа (`"num`) и низове (`"str`). Всеки две стойности в източника са разделени с поне един интервал или табулация. Такива може да има и преди първата и след последната стойности. Чете се до край на ред (`'('n)`) или, ако такъв няма, до края на входа. От прочетения низ последователно се извличат толкова стойности, колкото е посочено в *q* или колкото е възможно до изчерпване на низа. При успешно четене и извличане резултатът е редица от получените стойности и низ (възможно празен), съдържащ остатъчния текст до края на реда или входа. Действието е успешно и ако са извлечени по-малко от посочения брой стойности (но тогава и резултатът е по-къса редица). Ако *q* е празна редица, извършва се „празно четене“ и резултатът съдържа редица от само един низ – от началото на реда до края му или до края на входа. При неуспешно четене или извличане резултатът е `$`.

`reads` *s q*

Подобно на `readf`, но се чете от низа *s*.

6.8 Разни

`nil` *x*

Дава `$t` или `$f` според това дали *x* е `$` или не.

`val` *x*

Обратното на `nil`.

`type` *x*

Типът на стойност като низ: `"num`, `"boo`, `"str`, `"seq`, `"pic`, `"fun` или `"nil`.

Пример. Подобно на `!`, но прилагането на функцията става към всяка атомарна стойност на редицата:

```
@{[f;xs]::@{x::?{type.x="seq::f@@_;f}.x}!xs}  или
@{[f;xs]::@{x::>.@@when(type.x="seq).f.x}!xs} .
```

`next` x_1 x_2

Последователно пресмятане на два (независими) израза, като резултатът е стойността на втория. Еквивалентно на `@{[_;y]::y}`.

`prev` x_1 x_2
: Последователно пресмятане на два израза, като резултатът е стойността на първия. Еквивалентно на `@{[x;_]::x}`.

7 Константи

Предопределените глобални константи имат специални имена.

`$f $t`
: Неистина и истина.

`$pinf $ninf`
: Положително и отрицателно „безкрайни“ числа – съответно по-голямо и по-малко от всяко друго число.

`$e $pi`
: $e = 2,71828\ldots,\ \pi = 3,14159\ldots$.

`$np`
: „Отсъстваща фигура“: като се поставя върху, до, над или под друга фигура последната не се променя.

8 Шаблони

Шаблоните са синтактично средство за описване на строежа на стойности, използвано в определения и във функции-изрази. Успешното съпоставяне на шаблон със стойност води до свързване на имена със стойности и е единственият начин да се направи това. Във функции-изрази шаблоните служат и за различаване на варианти на стойността на аргумента, така че в различните случаи да се извършат различни пресмятания.

Най-простият и най-често прилаган шаблон е този, който се състои от само едно име. Такъв шаблон се съпоставя винаги успешно с която и да е стойност и в резултат посоченото име се свързва с нея – впоследствие то обозначава именно тази стойност.

В общия случай шаблонът е структурна схема, съпоставяна със стойността на израз за установяване на съответствие между части на едното и другото. Тогава шаблонът може да съдържа няколко имена и свързва всяко от тях със съответните части от стойността на израза. Например, ако `s` е коя да е редица с поне два члена, определението

```
a\b/c = s
```

има същия резултат като следните три:

```
a = fst.s; b = butf.(butb.s); c = bst.s
```

Тъй като съставни стойности в U са редиците, низовете и фигурите, тъкмо към тях се прилагат структурни шаблони.

По запис шаблоните са подобни на изрази от ограничен вид. Допустими действия в тях могат да бъдат само някои операции и то записани изрично, а не косвено посочени чрез допълнително присвоени имена. Това са операциите за образуване на редици и фигури `\` и `/` и тези само за фигури `|`, `-`, `&` и `^`. Аргументи могат да бъдат константи и имена; едните съответстват на същите стойности в съпоставяната с шаблона стойност, а другите – на кои да е стойности в контекста на съпоставянето и се свързват с тях.

Шаблоните не са и не съдържат затворени изрази и частични прилагания на функции, но се допуска използване на кръгли скоби за обособяване на аргументи-подизрази. Използва се и синтактичната конструкция `[...]` за задаване на редица.

Разбира се, във връзка с шаблони понятията израз, операция и аргумент са условни. Шаблонът не се пресмята, операциите в него не образуват стойности, а посочват как действителната, съпоставяна с шаблона съставна стойност трябва да бъде образувана, за да е налице успешно съпоставяне. Аргументите на такава операция отговарят на подшаблони, съпоставяни със съответните части на действителната стойност.

Ако на някаква част от действителната стойност искаме да не даваме име, на нейно място в шаблона поставяме фиктивното име `_` – така съпоставянето не води до свързване. Например

```
a\_/c = s
```

е подобно на горното, но дава стойности само на `a` и `c`.

9 Граматика

Символите на използвания при задаване на граматичните правила метаезик имат следния смисъл:

- `' '` непосредствено присъстваща в програмата част;
- `|` алтернативни части (където няма скоби, следването има предимство пред `|`);
- `[ ]` незадължителна част;
- `{ }` списък от един или повече елемента; когато са повече от един, в списъка участва и разделител – той се задава с последната посочена между `{` и `}` част;
- `( )` групиране – заграденото между `(` и `)` се разглежда като един елемент в редицата.

```
програма = { ( израз | '++' ( определение | имефайл ) ) ';' }
определение = именуване | '{' { именуване ';' } локално '}'
именуване = шаблон '=' израз
израз = { стойност стойност }
стойност = име | константа | '_' | редица | '(' израз ')' | затворен
редица = '[' [{ израз ';' }] ']'
затворен = параметричен | условно | функция
параметричен = '{' израз локално '}'
условно = '?{' { израз '::' израз ';' } [ ';' израз ] [ локално ] '}'
функция = '@{' { клауза ';' } [ локално ] '}'
клауза = [ шаблон ] ['('  израз ')'] '::' [ израз ]
локално = '++' { определение ';' }
константа = число | низ | имеоп | '$' | '$f' | '$t' | '$pinf' | '$ninf' | '$e' | '$pi' | '$np
имеоп = '+' | '-' | '*' | ':' | '^' | '=' | '<' | '>' | '/' | '\' | '.' | '#'
      | '!' | '%' | '~' | '|' | ',' | '&' | '<:' | '>:' | '|:'
      | '||' | '<=' | '>=' | '<>' | ',,' | '%%'
```

Забележка. Извън граматиката остават подробностите по записване на низове, числа, имена и коментари – те са предмет на лексическо определяне, за всяко от тях самостоятелно спрямо граматиката. Частичните прилагания и шаблоните са разновидности на изразите и затова също не са обособени в самостоятелни граматични категории. Като вид „стойност“ в израз обаче е посочено участващото и в двете `_`.

0123456789