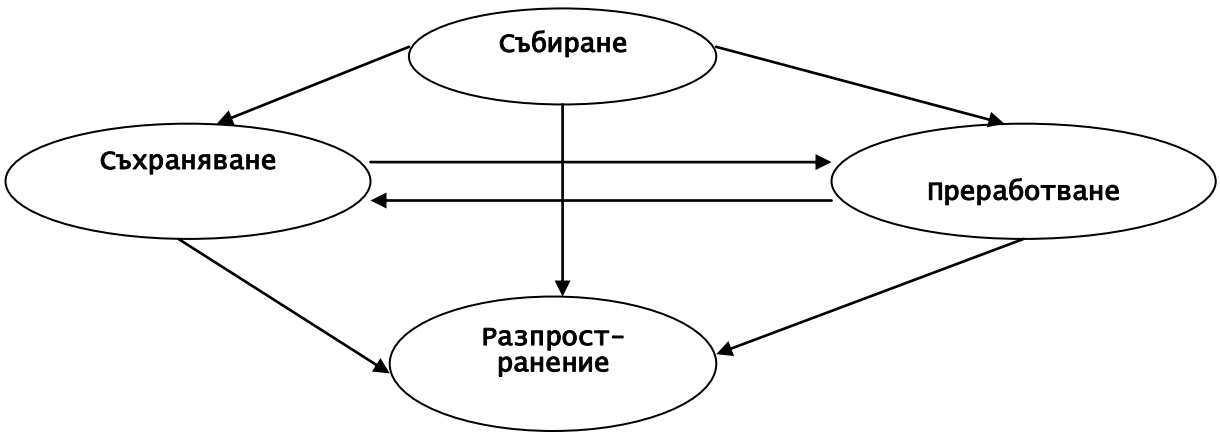
Глава 1

Въведение в компютрите и програмирането

1.1 Компютър

Повечето от вас са използвали вече компютър за работа, за комуникации или за забавление. Компютърът е устройство или система, която е годна да изпълнява редица от операции по точно определен начин. Операциите са често числови изчисления, аналитични преобразувания или обработка на данни, но включват също и входно/изходни операции. Чрез компютрите се извършват: текстообработка, графика, счетоводство, управление на устройства, телекомуникационни и банкови услуги и др. дейности. Тъй като исторически първото им предназначение са изчисленията, наричат се още **изчислителни системи.**

Най-общо, компютърът е средство за представяне и обработване на информация. Неформално информацията е съвкупност от символи, над които се извършват следните информационни дейности: *събиране*, *съхраняване*, *преработване* и *разпространение*. Връзките между тези дейности са илюстрирани на Фиг. 1.



Фиг. 1

До сегашните компютри се стигна след преминаване през следните поколения:

- *нулево поколение* – електронно механични устройства
- *първо поколение* – лампови ЕИМ - ENIAC (Electronic Numerical Integrator And Computer) е първия използваем компютър. Проектиран е от Дж. Преспър Екерт и Дж. Мокли от университета в Пенсилвания и е завършен през 1946 година. Компютърът е бил с огромни размери. Съдържал е около 18 000 електрически лампи.
- *второ поколение* – транзисторни ЕИМ (1948)
- *трето поколение* – компютри на базата на интегрални схеми (миникомпютри)
- *четвърто поколение* – компютри на базата на големи интегрални схеми (микрокомпютри)

През 1946 година, на базата на основните информационни дейности и връзките между тях, Джон фон Нойман създаде архитектура на изчислителна система, която премина през горните поколения и има следния вид:

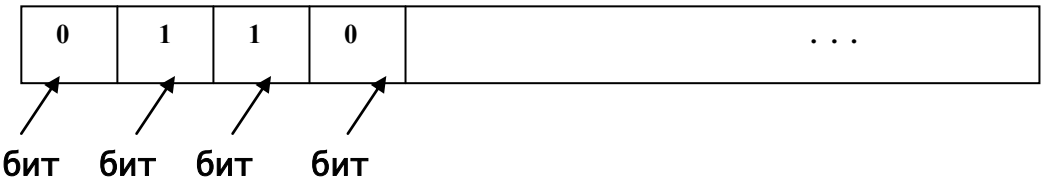
Фиг. 2.

Информацията, която трябва да бъде обработена от компютъра, първо се въвежда в паметта му. Това се осъществява от входните му устройства. Централният процесор (ЦП) преработва информацията от паметта. Резултатът от тази обработка се запомня отново в паметта.

Изходните устройства изобразяват информацията от паметта в подходящ (удобен) за потребителя вид.

Памет

Паметта на компютъра може да бъде изобразена като редица от елементи, всеки от които е носител на информацията – цифрите 0 и 1 (Фиг. 3). Такъв обем информация се нарича **бит**.



Фиг. 3.

Технически не е възможно да бъде реализиран пряк достъп до всеки бит от паметта, поради което няколко бита се групират в **дума**. Думата може да е с размер 8, 16, 32, 64 и т.н. бита. 8 – битовата информационна единица се нарича **байт**. Обемът на паметта се измерва в байтове (B), килобайтове (KB), мегабайтове (MB), гигабайтове (GB) и терабайтове (TB), където:

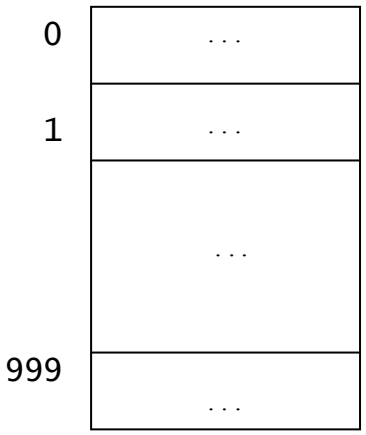
1 KB = 2^{10} B ≈ 10^3 B

1 MB = 2^{20} B ≈ 10^6 B

1 GB = 2^{30} B ≈ 10^9 B

1 TB = 2^{40} B ≈ 10^{12} B.

Обикновено се реализира пряк достъп до всяка дума на паметта. Всяка дума се свързва с пореден номер, наречен неин **адрес**. Номерацията започва от 0. На Фиг. 4. е изобразена памет на компютър, съдържащ 1000 думи, номерирани от 0 до 999.



Фиг. 4.

В паметта на компютъра може да се представя всякакъв вид информация – числа, имена, списъци, картини. Информацията, която се записва в думите на паметта, се нарича **стойност на клетките на паметта**. Всяка дума на паметта съдържа някаква информация (няма празни клетки). Освен това, никоя дума не може да съдържа повече от една даннова единица. Всеки път когато някаква информация се записва в дума от паметта, старата информация в тази дума се унищожава и не може да бъде възстановена. Освен данни, в паметта на компютъра се записват и инструкции (команди), с помощта на които се обработват данните. Има два вида памет: **оперативна (ОП)** и **външна (ВП)**.

ОП е бърза, но скъпа. В съвременните компютри (микрокомпютрите) тя е изградена от чипове. **Компютърният чип** (интегрална схема) се състои от пластмасов или метален корпус, метални конектори и вътрешни връзки, изградени главно от силикон.

Има два типа ОП: **постоянна (ROM)** и **памет с произволен достъп (RAM)**. Паметта ROM съхранява информация, дори когато компютърът е изключен. Най-важните програми на микрокомпютъра са записани в паметта ROM, включително и програмата, която при включване на компютъра проверява дали всичките му компоненти работят правилно. За разлика от паметта ROM, RAM съхранява информация, само когато компютърът е включен. Нейният обем се измерва в килобайтове и мегабайтове.

Първият модел IBM PC имаше 256 KB RAM в системната си платка, но паметта му можеше да се разширява до 640 K чрез включване на допълнителна платка памет към един от куплунгите за разширение на конфигурацията.

ВП осигурява по-евтино съхранение на информация, която се запазва при изключване на захранването. На тази памет информацията се съхранява във вид на файлове.

Файлът е организирана съвкупност от взаимно свързани данни. Изграден е от отделни части информация, наречени **записи**. Всеки файл има **име**, с което той е определен еднозначно в множеството от файлове.

Най-често за ВП се използва **твърд диск** (hard disk). Той се състои от въртящи се плочи с магнитно покритие и глави за четене и писане. Често се използва и **флопидиск** или **дискета**. Дискетата се състои от

пластична кръгла основа, покрита с намагнитизирано вещество и поставена в пластмасова обвивка. Използва се за пренасяне на информация от един компютър на друг. Дискетите са евтини, удобни и сравнително стабилни носители на информация. Основният им недостатък е, че имат малък обем. Широкоразпространени напоследък са компактдисковете CD ROM. Те побират голямо количество информация, евтини са, но са предназначени само за четене.

Централен процесор

Той управлява цялостната работа на компютъра. Изграден е от един или няколко чипа. Вътрешността му е изключително сложна. Например, чипът на Pentium (известна марка процесори) е съставен от над 3,2 милиона транзистора. Произведен е чрез т. нар. 0,8 или 0,6 микронен фотолитографски процес. 0,8 и 0,6 означават широчината в микрони на металните пътечки, които свързват транзисторите му ($1\ \mu = 10^{-6}$).

ЦП се грижи за управлението на системата, за аритметиката и прехвърлянето на данните. Той открива и изпълнява инструкции, извършва аритметични и логически операции, взема данни от паметта и външните устройства и ги връща обратно там.

ЦП се състои от управляващо усройство (УУ) и аритметично-логическо устройство (АЛУ).

УУ координира работата на компютъра. То изпраща разрешаващи и забраняващи сигнали към отделните устройства и по-такъв начин синхронизира работата на системата.

АЛУ изпълнява различни аритметични и логически операции (събиране, изваждане, умножение, деление, логическо събиране, логическо умножение, отрицание, сравнение и др.). Наборът от аритметични и логически операции определя т. нар. **език на компютъра** или **машинен език**. Всеки компютър може да изпълнява инструкциите на своя машинен език.

Първият модел IBM PC използваше за ЦП микропроцесора Intel/8088. Този процесор може да обработва 16 бита (2 байта) информация едновременно. Тактовата честота на компютъра беше 4.77 MHz (мегахерца), което означава, че микропроцесорът получава 4.77 милиона импулса всяка секунда. Днес бързодействието на ЦП е многократно по-голямо от това на първия модел IBM PC. Технологичните подобрения на микропроцесорите се реализират с невероятни темпове.

Благодарение на това, производителността на микропроцесорите се удвоява на всеки 18 месеца.

Входни и изходни устройства

Наричат се още периферни устройства.

Входните устройства служат за въвеждане на информация в паметта на компютъра. Широкоразпространени са клавиатурата, мишката, дисковите и лентовите устройства, камерите, скенерите.

Изходните устройства извеждат резултатите в подходяща за възприемане форма. Най-разпространени са мониторите, дисковите и лентовите устройства, принтерът, тонколонките.

ЦП, RAM паметта и електрониката, която управлява твърдия диск и други устройства, са свързани помежду си посредством множество електрически линии, наречени **шина**. Данните и инструкциите се движат по шината от паметта и периферните устройства до ЦП и обратно. На дънната платка са поставени ЦП, RAM паметта и слотовете, чрез които платките, управляващи периферните устройства, се свързват с шината.

Съвкупността от електронните, електромеханичните и механичните компоненти, изграждащи компютъра, се нарича негова **апаратна част** или **хардуер**. Хардуерът на компютъра е тясно свързан с т. нар. **програмна част**, **програмно осигуряване** или **софтуер**. Програмното осигуряване на компютрите включва: операционни системи, среди за програмиране и приложни програми.

Операционни системи (ОС)

Всеки компютър работи под управлението на някаква ОС. Тя управлява:

- взаимодействието между потребителите и хардуера (чрез команден език);
- периферните устройства (чрез драйвери);
- файловете (чрез файловата система).

Широкоразпространени ОС са MS DOS, UNIX, LINUX.

Среди за програмиране

Те автоматизират цялостния процес по създаването, транслирането, тестването, изпълнението, изменението и документирането на програмите. Средите за програмиране включват език за програмиране, транслатор (компилатор или интерпретатор), свързващ редактор, изпълнителна система, система за проверка на програми, система за

поддържане на библиотеки и текстови редактори. Тези компоненти са свързани в система чрез управляваща програма.

В настоящия курс по програмиране ще използваме средата за програмиране Visual C++ 6.0.

Приложни програми

Те реализират някакво приложение – икономическо, проектантско, инженерно, счетоводно, медицинско и др.

1.2. Програми и програмиране. Езици за програмиране

1.2.1. Програми. Автоматизиране на програмирането

За да бъде изпълнена една или друга обработка над входните данни, трябва да бъде написана редица от инструкции (команди), които компютърът разбира и изпълнението на които да води до решаването на някаква задача. За различните задачи, тези редици са различни.

Редица от инструкции, водеща до решаване на определена задача, се нарича **програма**, а процесът на писане на програми – **програмиране**.

Инструкциите, с помощта на които се записва програма, изграждат език, наречен **език за програмиране**. Всеки компютър си има свой собствен език за програмиране (машинен език) и може да изпълнява програми, написани на него.

Машинният език е множество от машинни инструкции, които процесорът на компютъра може директно да изпълни. Програмата на машинен език е редица от числа, записани в двоична позиционна система.

Една типична редица от машинни инструкции е следната:

1. Премести стойността на клетка 15000 от паметта в регистър AX.
2. Извади 10 от регистъра AX.
3. Ако резултатът е положително число, изпълни командата, намираща се в клетка 25000 от паметта.

За различните процесори тази редица от инструкции се кодира по различен начин. При процесора Intel 80386 това кодиране има вида:

```
161 15000 45 10 127 25000
```

Тази редица от числа се записва в паметта, след което се изпълнява. Но една дълга програма е съставена от хиляди команди и вероятността за грешки при програмиране на машинен език е много голяма.

Първата стъпка в процеса на автоматизация на програмирането е въвеждането на **асемблерните езици**. Те дават кратки имена на командите. Например MOV означава премести, SUB – извади, а JG – ако е по-голямо от 0, премини към …. Като се използват тези команди, горната редица от инструкции се свежда до:

```
MOV AX, [15000]
SUB AX, 10
JG 25000
```

Този фрагмент е по-лесен за четене от хора, но компютърът не разбира тези инструкции. Те трябва да бъдат преведени на машинен език. Тази задача се изпълнява от компютърна програма, наречена **асемблер**. Асемблерът взема редицата от букви MOV AX и я превежда в машинния код 161. Подобни действия асемблерът извършва и над другите команди. Асемблерът именува също и думите от паметта. Това позволява да бъдат използвани буквени имена вместо адреси от паметта. Например, горният фрагмент може да се запише и във вида:

```
MOV AX, [A]
SUB AX, 10
JG WRITE_ERROR
```

и извършва следните действия: Прехвърля стойността на променливата A в регистъра AX; Изважда 10 от регистъра AX; Ако стойността на полученото е положително число, извежда съобщение за грешка.

Асемблерните езици имат големи предимства пред използването на чисти машинни кодове, но имат и съществени недостатъци. Двата най-важни техни недостатъка са, че първо, дори и за най-простите задачи е необходим голям брой инструкции и, второ, редицата от команди се променя от процесор на процесор. Ако компютърът излезе от употреба, програмите трябва да се напишат отново за новата машина.

В средата на 50-те години започнаха да се появяват езици за програмиране от високо ниво. При тях програмистът описва схематично основната идея за решаване на задачата, а специална програма, наречена **транслатор (компилатор** или **интерпретатор)**, превежда това описание в машинни инструкции за конкретния процесор. Например, на езика Паскал, горната редица от инструкции ще се запише във вида:

```
if a > 10 then writeln('error')
```

Работата на компилатира е да обработи тази редица от букви и да я преведе във вида:

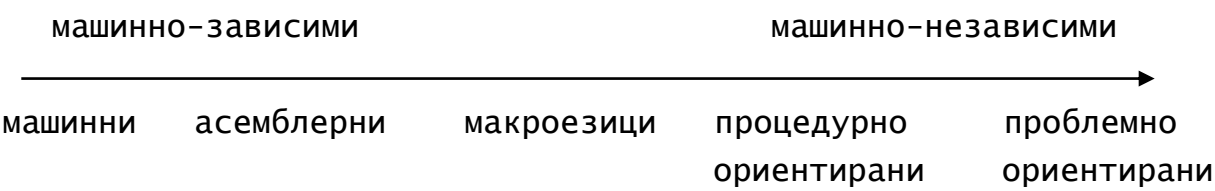
```
161 15000 45 10 127 25000
```

транслаторите са доста сложни програми. В рамките на настоящия курс ще ги приемем за даденост. Важното е, че езиците за програмиране от високо ниво са независими от хардера.

1.2.2. Езици за програмиране

1.2.2.1. Класификация

Сега съществуват стотици различни езици за програмиране. Те могат да бъдат класифицирани по различни признаци. Най-общоприетата класификация е по степента на зависимост на езика от компютъра. Фиг. 5 илюстрита тази класификация.



Фиг. 5.

Машинно-независимите езици се наричат още **езици от високо ниво.**

В настоящия курс ще направим въведение в програмирането на базата на езика за процедурно и обектно програмиране C++, който притежава и средства за програмиране почти на машинно ниво.

Друга класификация на езиците за програмиране е *според стила на програмиране.* Съществуват два основни стила за програмиране: **императивен (процедурен)** и **декларативен (дескриптивен).**

При процедурния стил програмата се реализира по схемата:

Програма = Алгоритъм
+
Структури от данни

Съществуват стотици процедурни езици. Такива са асемблерните, Фортран, Алгол, Кобол, Снобол, Паскал, Ада, C, C++ и др.

Процедурният стил е тясно свързан с фон Нойманoвата архитектура на изчислителна система. Процедурните езици за програмиране могат

да се разглеждат като синтактични обобщения на Нойомановите компютри. Семантиката на тези езици по същество е същата като семантиката на машинните езици. Следователно, процедурните езици са абстрактни версии от високо ниво на фон Нойомановите компютри и като такива, те наследяват всички техни недостатъци.

Един от учените, на който до голяма степен се дължи увековечаването на архитектурата на фон Нойман и на процедурното програмиране е професорът от IBM Дж. Бекус. Той е авторът на езика Фортран, ръководил е редица проекти, свързани с реализиране на процедурни езици за програмиране. Но Бекус, анализирайки недостатъците на фон Нойомановата архитектура, през 1978 година на конгрес на IFIP повдига въпроса: Може ли програмирането да се освободи от бремето на Нойомановия стил? Да разгледаме какви са основните недостатъци на императивните езици?

Линията за връзка между ОП и ЦП е тясното място на Нойомановите компютри. Бекус я нарича “гърло на бутилката”. Изпълнението на машинната програма води до съществено изменение на ОП. Това се достига чрез многократно прехвърляне на стойността на отделни клетки в едната и другата посока. При това обаче съществена част от трафика през това място не са “полезни” данни, а адреси на данни, а също операции, които се използват за пресмятане на такива адреси. Бекус отбелязва, че тясното място на компютъра създава и тясно място в мисленето на програмиста като го кара да си представя цялата обработка на данните клетка по клетка, вместо да решава задачата си в по-мащабни термини. Програмата, написана на императивен език, действа над клетките на паметта, отделени за променливите. Програмистът трябва да опише всички последователни изменения на тази памет, така че като се изпълнят тези изменения на Нойомановия компютър, да се получи търсеният резултат. Така императивната програма трябва да управлява последователните стъпки на работа на компютъра. От необходимостта, обработката на данните в стил “клетка след клетка” да се опише точно и детайлно, следват следните недостатъци: *относително малка мощност, липса на гъвкавост, ниска производителност*.

Важен момент при създаването на езици за програмиране е формалното описание на семантиката на езика. Би било добре това описание да е просто и удобно за доказване на редица свойства на програмите (частична и тотална коректност, завършване на изпълнението, еквивалентност на програмите и др.). Последното не е

така при императивните езици, тъй като те са сложни (косвено следствие от Нойманова архитектура). Оттук пък произтича твърде голямото и сложно формално описание на семантиките им. Поради сложността си, формалното описание на семантиката на императивен език за програмиране не е много по-полезно от неформалното описание на езика. На практика то се използва само при изследването свойствата на малки и елементарни програми, но не и при по-сложни реални задачи. *Така вторият главен недостатък на императивните езици е липсата на полезни математически свойства*. Чрез императивните езици не могат практически лесно да се доказват свойствата на програмите.

Все пак връзката на императивните езици с архитектурата на фон Нойман има и предимства. Главното предимство е ефективната им реализация.

И Бекус, анализирайки недостатъците на императивните езици предлага нов стил за програмиране – FP стила (вид декларативно програмиране).

Декларативните езици са антипод на императивните. При тях в програмата се указва явно какви свойства има желаният резултат, без да се указва как точно този резултат се получава.

Допустим е всеки начин за изпълнение на програмата, който води до получаването на резултат с описаните в програмата свойства. Така при декларативните езици се казва какво се пресмята, без да се определя как да се пресмята. Програмата на декларативен език за програмиране се определя така:

Програма = Списък_от_дефиниции_на_функции или
Списък_от_равенства или
Правила + Факти

а изпълнението ѝ е **редукция, извод** или **доказателство.**

Такива езици са Лисп, Пролог, Pop-11, TabLog, EqLog, Prolog-with-Equality и др.

Тези езици **не** са непосредствено свързани с фон Нойманова архитектура. От това произтича един съществен техен недостатък – ниска ефективност на реализацията им на такива компютри. Така възниква необходимостта от т. нар. ненойманови компютри или *компютри от пето поколение*. Това са компютри, които се характеризират с висока степен на паралелизация.

Сегашните прогнози за тези компютри са свързани с технологията на 21-ви век, определена като нанотехнология или молекулярно производство. При това производство, измерванията ще са в нанометри (1 η = 10^{-9}), а не в микрони (1 μ = 10^{-6}), както е в момента. Представете си стотици хиляди микропроцесора, работещи паралелно, събрани в блокче с размери на бучка захар, като всички принадлежности на персонален компютър се побира в този обем. Предвижда се молекулярното производство да доведе до прецизно управление на производствените процеси на широк кръг механични устройства.

В момента не една технология произвежда все по-малки и по-малки компоненти. Фирмата Hitachi демонстрира запаметяващо устройство само от един електрон, съхраняващ 1 бит информация. Texas Instruments са произвели чип с компоненти 50 пъти по-малки от обикновените. Toshiba е изработила транзистор с широчина 0.04 микрона.

1.2.2.2. Описание на език за програмиране

За да се опише един език за програмиране е необходимо да се опишат неговите **синтаксис** и **семантика**. Всяка програма на някакъв език за програмиране може да се разглежда като редица от символи. Но не всяка редица от символи е програма на съответния език. Да се определи **синтаксисът на един език за програмиране** значи да се зададе множество от правила, които определят как програмите се изграждат като редици от символи, т.е. да се зададе множество от правила, съгласно които една редица от символи е програма. Да се определи **семантиката на език за програмиране** значи да се определят правилата, съгласно които се изпълняват програмите на този език за програмиране. Следователно, синтаксисът определя формата, а семантиката – смисъла на програмата. Задаването става словестно, чрез примери или чрез правила.

При изучаването на езика C++ синтаксисът му ще описваме словестно и чрез метаезика на Бекус-Наур, а за описание на семантиката му ще използваме словестното описание.

Метаезик на Бекус-Наур

Това е първото широкоизползвано формално означение за описание на синтаксиса на език за програмиране. Въведено е като механизъм за описание на синтаксиса на езика Algol 60.

При това описание, имената на синтактичните категории се ограждат чрез ъглови скобки.

Пример

<програма>, <число>, <цяло_число>

са правилно означени синтактични категории.

Знакът ::= означава “това е” или “дефинира се да бъде”. Синтактичните категории, които се дефинират, се поставят отляво на знака ::=, а дефиницията им – отдясно на този знак.

Пример

Означението

<реално_число> ::= <цяло_число> . <цяло_число_без_знак>

определя, че реално число е конструкция, започваща с цяло число, следвано от символа точка и от цяло число без знак.

Знакът | определя алтернативни конструкции и означава ИЛИ.

Пример

Описанието

<цифра> ::= 0 | 1 | 2| … |9

означава, че цифра се дефинира като 0 или 1, или 2, или и т.н., или 9.

Означението {…} има смисъла, че ограденото в него се повтаря 0, 1 или повече пъти.

Пример

<редица_от_променливи> ::= <променлива> {, <променлива>}

означава, че редица от променливи се дефинира като

<променлива> или

<променлива>, <променлива> или

<променлива>, <променлива>, <променлива> и т.н.

Означението […] има смисъла, че ограденото в скобите може да участва в описанието, но може и да бъде пропуснато.

Пример

Описанието

<цяло_число> ::= [+|_]<цяло_число_без_знак>

означава, че цяло число се дефинира или като цяло число без знак, или като цяло число, предшествано от знак + или -.

1.3. Алгоритъм

При процедурния стил програмата се реализира по схемата:

Програма = Алгоритъм

+

Структури от данни

Основен неин елемент е алгоритъмът.

Механизъм за намиране на решение, който е еднозначен, изпълним и завършващ, се нарича **алгоритъм.**

Алгоритъмът като понятие е въведено от Евклид. Наречен е така в чест на арабския математик Ал Хорезми. В средновековието формулата DIXIT ALGORIZMI (така е казал Ал Хорезми) била сертификат за безупречна яснота и достоверност.

Еднозначността гарантира точността на инструкциите на всяка стъпка и определя следващото действие. Свойството *изпълнимост* определя, че всяка стъпка може да се изпълни на практика, а *завършването* – че изпълнението на стъпките ще завърши.

Алгоритъмът е редица от правила, които имат три компоненти: *пореден номер, команда* и *наследник*. Наследникът може да се пропусне ако съвпада със следващата команда от редицата.

Основни начини за описние на алгоритъма са словестният, чрез блок-схема, чрез средствата на някакъв език за програмиране. Ще илюстрираме словестното описние чрез пример.

Пример

Следващият фрагмент дава *общо* описание на алгоритъм за намиране на най-големия елемент на редицата от числа a_1, a_2, …, a_n.

1. m = a_1
2. Сравнява се a_2 с m и ако a_2 е по-голямо от m , то m = a_2
3. Стъпка 2 се повтаря до изчерпване на редицата.

Следва детайлно описание на алгоритъма за намиране на най-големия елемент на редицата от числа a_1, a_2, …, a_n.

1. m := a_1
2. i := 2
3. **Ako** a_i > m **To** изпълни 4 **ИНАЧЕ** изпълни 5
4. m := a_i
5. i := i + 1

6. **Ако** i ≤ n **То** изпълни 3 **Иначе** изпълни 7
7. max := m
8. съобщи max
9. прекрати изпълнението

При това описание n и a_1, a_2, …, a_n съдържат входните данни и се наричат още **входни променливи**, max съдържа резултата и се нарича **изходна променлива**, а m и i са **работни (вътрешни) променливи**.
Решаването на една задача чрез програма на процедурен език се свежда до точно описание на това как да се получи резултатът.

Допълнителна литература

1. П. Азълов, Програмиране. Основен курс. Увод в програмирането, С. АСИО, 1995.
2. К. Хорстман, Принципи на програмирането със C++, С., СОФТЕХ, 2000.