

## Лекция 6

### Алгоритми

Алгоритъмът представлява завършена система от формални правила които точно и еднозначно определят последователността от недвусмислени и изпълними етапи, определящи краен процес за решаване на дадена задача.

В определението на това понятие се съдържат и основните изисквания към алгоритмите. На първо място отделните етапи трябва да са изпълними и то от обекта, който ще ги изпълнява (не изобщо изпълними). Понякога вместо изпълним се използва изразът ефективен. Това означава, че когато се казва ефективен алгоритъм се има пред вид и изпълним.

Следващо изискване е алгоритъмът да е недвусмислен. Това означава, че по време на изпълнение на алгоритъма, при всяко състояние на процеса информацията трябва да е достатъчна, че еднозначно и напълно да се определят действията, които трябва да се изпълнят на всеки от етапите. Това означава, че на всеки етап на алгоритма не са необходими каквито и да са 'творчески способности', а само използването на наличният набор от инструкции на обекта изпълняващ задачата.

Изискването за краен процес, означава, че изпълнението на алгоритъма непременно трябва да го доведе до завършване. Това изискване се налага, поради съществуването на голяма група процеси, които по дефиниция са безкрайни. Такива процеси, се разглеждат в теорията на изчисленията и много често те се апроксимират с крайни процеси.

Класически учебен пример за алгоритъм е алгоритъмът на Евклид. Той се отнася до определянето на най-големият общ делител на две естествени числа. Ако двете числа са равни, то най-големият общ делител съвпада със самите числа. Ако те са различни, най-големият общ делител на двете числа е делител и на разликата между тях. При това, ако се разгледат разликата и по-малкото от двете числа, то техният най-голям общ делител е същият както на изходните числа. Този факт позволява да се дефинира последователност от действия, които довеждат до намиране на най-големият общ делител на числата.

Нека означим изходните числа с  $x$  и  $y$ . Ако те са равни, търсеното решение е едно от двете числа. Ако са различни, да означим разликата между тях с  $g$ , а по-малкото от двете числа с  $h$ . Тогава можем да заменим изходната двойка числа  $x$  и  $y$  с новата двойка  $g$  и  $h$  и да решаваме задачата по-същият начин както в началото. Този процес може да се продължи дотогава, докато се получи двойка числа, които са равни помежду си и това е решението на задачата. Последователността от действия за решаване на задачата могат да се запишат като отделни стъпки:

1. Начало на задачата;
2. Задаване стойности на числата  $x$  и  $y$ ;
3. Проверка: ако  $x > y$ , то премини към т. 6;
4. Проверка: ако  $y > x$ , то премини към т. 7;
5. Изпълнено е:  $x = y$ ; най-голям общ делител е  $z = x$ ; край;
6.  $x = x - y$ ; премини към 3;
7.  $y = y - x$ ; премини към 3;

Този алгоритъм можем да проверим с конкретен числен пример. Нека началните стойности на числата  $x$  и  $y$  са съответно 49 и 21. След задаването на тези стойности, трябва последователно да се изпълнят предписанията от горната поредица след т.3 надолу. Проверката в т. 3 показва, че условието  $x > y$  е изпълнено и изпълнението трябва да се прехвърли към т. 6. Операцията  $x = x - y$ , описана в тази инструкция е често използвана конструкция в програмните езици за компютърните системи. Тя трябва да се разбира, като операция за изваждане на стойността на  $y$  от стойността на  $x$  и

запис на полученият резултат на мястото на  $x$ . В конкретният случай, след изпълнението на тази операция, стойността на  $x$  ще бъде  $49-21=28$ . С новите стойности  $x = 28$  и  $y = 21$  изпълнението на алгоритъма се прехвърля отново в т. 3. Следващите стойности на величините  $x$  и  $y$  в процеса на изпълнение на алгоритъма се променят както следва:

$x$	$y$
28	21
7	21
7	14
7	7

Най-големият общ делител се получава, когато стойностите на  $x$  и  $y$  станат равни помежду си. В случая стойността е 7.

### Описание на алгоритмите

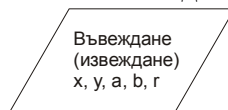
В представеният пример, алгоритъмът беше описан с помощта на естественият разговорен език (словесно-описателен метод). В повечето случаи, този начин на представяне на алгоритмите не е подходящ, особено за алгоритми с голям брой последователни стъпки (инструкции). Тогава се губи прегледността на алгоритъма и се затруднява проследяването на отделните връзки и логически разклонения в него. По-голяма прегледност се получава, когато се използват графични означения за отделните типове инструкции. Графичното представяне на алгоритъмът се нарича блок-схема.

Използват се няколко стандартни геометрични фигури за представяне на различните типове действия от алгоритъма. Последователността в която следва да се изпълняват отделните операции или блокове се указва със стрелки.

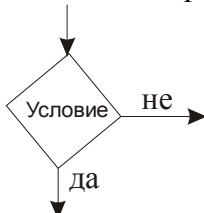
Всяка блок-схема има начален блок с който се указва къде се намира началото на алгоритъма (откъде започва изпълнението на задачата). Аналогично, трябва да има блок за край на алгоритъма. Най-често тези блокове се означава графически със следната фигура:



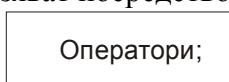
Операциите за въвеждане и извеждане на информация, се представят графически с успоредник, в който се описват величините, които получават стойности или тези които се извеждат към някое от входно-изходните устройства:



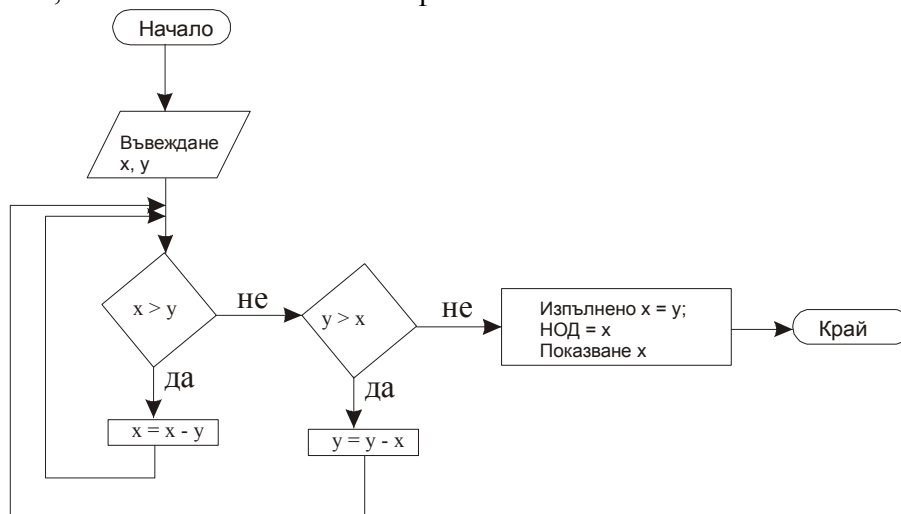
Разклоненията в алгоритмите, се определят от резултата при проверка на някакво логическо условие. При изпълнение на условието (резултат 'да') се избира една последователност от команди, а при неизпълнение на условието (резултат 'не') се избира друга последователност. Графическият символ, за представяне на оператор за разклонение на алгоритъма има вида:



Операциите свързани с обработка на информацията (действия с величините) се изобразяват посредством правоъгълници, в които се описват действията с величините:



Като използваме въведените графически форми за представяне на отделните действия от алгоритмите, разгледаният по горе алгоритъм на Евклид можем да изобразим, както това е показано на фиг 3.9



Фиг. 3.8 Алгоритъм на Евклид

### Итерационни структури

Итерационните структури са фрагменти от алгоритмите за решаване на задачи, в които се организира циклично повторение на определен набор от инструкции, които постепенно приближават процеса до желаният резултат. Този метод се използва за организация на циклични програми с неизвестен предварително брой на повторенията. Обикновено при тези процеси, резултатът от предишното действие се използва като входна информация за следващото действие (цикъл). Контролът на резулирания брой повторения се осъществява на основата на някакво условие, определящо приближаването на изчислителният процес до желаният резултат. Често като условие за контрол на итерационните цикли се използва достигането на предварително зададена точност на изчисляваните величини.

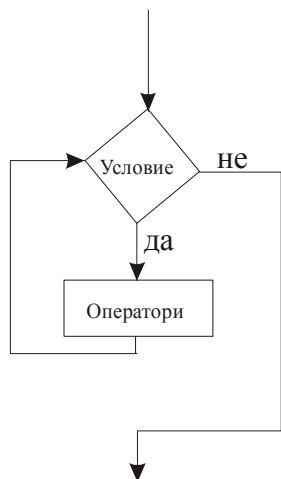
Типичен пример на итерационна структура беше представена при разглеждането на алгоритъма на Евклид. В него, инструкциите от 3 до 7 се повтарят дотогава, докато се получи равенство на числата  $x$  и  $y$ . Това повторение на инструкциите се управлява от двете условия в стъпки 3 и 4 на алгоритъма.

Многократното изпълнение на последователност от инструкции представлява важна алгоритмическа концепция. Един от методите за организация на повторение на изпълнението на поредица от команди (който се явява итерационна структура) се нарича **цикъл**. Характерно за циклите е, че в конструкцията им се прдвиждат специални инструкции, осигуряващи управлението на цикъла набор от операции които се изпълняват многократно. Управляващите инструкции задават началото и края на цикъла и задават структурата и типа на цикъла, а многократно изпълняваните инструкции образуват тялото на цикъла. Инструкцията за начало на цикъла, обикновено се нарича заглавен оператор.

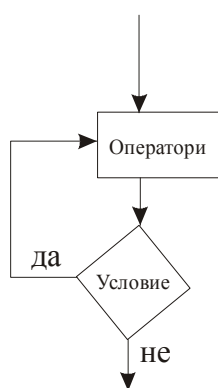
Управлението на цикличните структури обикновено съдържа три операции: инициализация, проверка на условие за край на цикъла и модификация. При инициализацията се установява началното състояние, от което започва изпълнението на цикъла. Самият процес на инициализация представлява задаване на начални стойности на някои величини, които в процеса на изпълнение на цикъла се променят. Операциите за инициализация могат да бъдат вградени в управляващите оператори на цикъла или да бъдат извършени преди началото на цикъла.

Операцията за проверка на условието за край на цикъла е много важна, тъй като тя определя броят на повторенията на инструкциите от тялото на цикъла. Когато такова условие не се проверява или е зададено некоректно, цикълът може да се превърне в безкраен. Модификацията е операция чрез която се променя състоянието на определени обекти (стойности на величини), така че ситуацията да се приближава към изпълнението на условието за край на цикъла.

Представеният по горе алгоритъм (Евклид), е типична циклична структура, тъй като е се извършва многократно повторение на определен набор от инструкции. Като инициализираща операция, може да се разглежда задаването (въвеждането) на стойности на величините  $x$  и  $y$ . Тези стойности определят началното състояние на циклическият процес. Като условие за проверка за край на процеса се използват инструкциите 3 и 4. Когато условието в някоя от тях е изпълнено, то е индикация че процеса трябва да продължи. Само когато и двете условия не са изпълнени (т.е. изпълнено е условието  $x = y$  – условието за край на цикъла) процеса се прекратява. Модифициращи операции се явяват 6 и 7, тъй като чрез тях се променят стойностите на  $x$  и  $y$  и процеса се приближава до условието  $x = y$ . Алгоритъмът на Евклид е специфичен с това, че в тялото на цикъла няма изпълними операции. Изпълнението на цикъла се извършва посредством модифициращите операции 6 и 7 и в известен смисъл, те могат да се разглеждат и като тяло на цикъла.



Фиг. 3.9 Цикъл с предусловие



Фиг. 5.10 Цикъл с постусловие

Използването на циклически структури придава на алгоритмите гъвкавост и прегледност. Съществуват няколко начина на организиране на циклически структури. Първият начин се нарича цикъл с предусловие. При него, първо се проверява условието за продължение на цикъла и ако то е изпълнено се изпълняват операциите от тялото на цикъла (фиг. 3.9). Характерното за този цикъл, е че операциите от тялото на цикъла се изпълняват след проверка на условието за продължение на цикъла и ако то не е изпълнено още в началото, е възможно тялото на цикъла да не се изпълни нито един път.

Другият тип организация на цикъл се нарича цикъл с постусловие. При него първо се изпълняват операциите от тялото на цикъла и след това се проверява условието за продължаване на цикъла (Фиг.3.11). Ако то е изпълнено, управлението се връща за повторно изпълнение, а в противен случай се излиза от цикъла.

В практиката съществуват различни типове задачи, които изискват използването на итерационни структури. Един от възможните случаи за необходимост от многократно повторение на определен набор от инструкции е последователна обработка за няколко еднотипни обекта. В този случай, повторението на инструкциите се определя от броят на обектите в дадената поредица, а условието за завършване на цикъла

се формира при обработката на последният от обектите. При организацията на циклически процеси за този тип задачи, обикновено броят на повторенията е известен предварително.

В други случаи, повторението на инструкциите се извършва върху едни и същи обекти, но при променени условия (променени стойности на величини). В този случай,

условието за завършване на циклическият процес, се задава от някакво специфично състояние на величините участващи в циклическият процес. Пример за използването на такъв цикъл е алгоритъмът на Евклид.

### **Основни концепции на традиционното програмиране.**

Както при използването на обикновеният разговорен език, програмистите създават свой стил и традиции при използването на алгоритмичните езици. По тази причина, няма единни становища за предимствата и недостатъците на съществуващите езици. Не случайно, един от първите езици от високо ниво, FORTRAN, все още се използва, въпреки че бяха създадени множество езици със редица нови идеи. Много програмисти са създали свой стил на програмиране с FORTRAN и тях не ги затруднява решаването на каквато и да е задача. В този смисъл, почти с всеки алгоритмичен език могат да се съставят програми за решаване на по-голямата част от задачите, които възникват в практиката. Все пак, отделните програмни езици имат специфична насоченост и това до голяма степен определя тяхното използване в практиката.

Всеки програмен език може да се разглежда като савкупност от три основни компонента:

- **азбука** на програмния език – съдържа набор от букви, цифри и специални символи, допустими за опизание на отделните команди и оператори. Освен това, в програмния език се включва определена система от специални думи, които се наричат ключови или запазени думи.
- **Синтаксис** – система от правила, които определят допустимите отношения между обектите на езика, на които се подчиняват всички оператори (команди).
- **Семантика** – система от правила, позволяващи точно и еднозначно да тълкуване на обектите от езика.

Основна структурна единица на програмите създавани с различните програмни езици е операторът. Обикновено операторите в програмните езици се разделят на три категории: оператори за декларации; изпълними оператори и коментарии. Операторите за декларации се използват за въвеждане на терминологията, която се използва по нататък в програмата. Например, с тях се обявяват имената и типът на всички данни, които ще се използват в програмата или се указват режими на работа на програмните части и т.н.

Изпълнимите оператори описват отделните стъпки при обработка на информацията от програмата. Те представляват описанието на алгоритъма за решаване на задачата. Коментарите представляват поясняващи текстове, записани в програмата, чрез които се подобрява читаемостта на програмата. Те се игнорират от програмата, която преобразува текста в двоични команди и служат единствено за пояснение към програмния текст.

### **Променливи и константи.**

Както беше отбелязано, вжна стъпка в процеса на усъвършенстване на компютърното програмиране се явява въвеждането на буквени означения (наименования), вместо цифрови адреси за величини, чиито стойности са разположени в паметта на компютъра. Тези наименования се наричат идентификатори, а величините, които означаваме с идентификатори наричаме **променливи**. Това означава, че при изменение на съдържанието на паметта в даден участък, се изменя стойността на величината свързана с определен идентификатор и обратно, за да изменим съдържанието на даден участък от паметта, трябва да изменим стойността на величината с даден идентификатор. Тъй като в програмните езици от високо ниво, почти не се използват конкретни адреси в паметта за записване на информация, всички

данни се именоват. Поради тази причина, променливите играят важна роля в процеса на програмиране.

Освен величини, които променят стойностите си по време на изпълнение на програмите, в практиката на програмирането се използват и величини, чиято стойност остава постоянна. Те могат да се използват или направо като числови стойности или чрез задаване на име на величината и отбелязване, че в процеса на работа нейната стойност не трябва да се променя. Тези величини се наричат **константи**.

#### **Операции, операнди и оператори.**

Операторите са основни логически елементи на компютърните програми. Те се състоят от аритметични или логически изрази, специални синтактични конструкции и спецификации за извикване на подпрограми или функции. Изразите представляват правила за пресмятане на стойности. Те се образуват с помощта на аритметичните и логически операции допустими за даден алгоритмичен език. В тях се използват величини, които се наричат операнди. Операнди могат да бъдат променливи ( $x$ ,  $y$ ,  $im$ ) или константи (10, 3.14159). Най-простият израз се състои само от един операнд - променлива или константа. За задаване на приоритет при изпълнение на операциите се използват скоби. Ето няколко примера на аритметични и логически изрази:

$(a+x)*xr/3.14159$  ;     $-w/(ax+r)$  ;     $bool\ and\ (a>5)$

Един от най-важните и използвани оператори в програмните езици е операторът за присвояване. В лявата част на този оператор се намира променлива, която при изпълнението на оператора получава стойност, а в дясната част - израз, чиято стойност се присвоява на променливата. В различните програмни езици се използват различни начини на записване на този оператор. В език Pascal операторът за присвояване има вида:

$var := expr$  ; , където  $var$  е променлива, а  $expr$  е израз. В език C++ в оператора за присвояване се използва знака '=':  $var = expr$  ;

#### **Типове данни**

Типовете на различните данни се определя от множеството стойности  $G$ , които могат да приемат константите и променливите от този тип и операциите които могат да се извършват с тях. Алгоритмичните езици разполагат с определен набор от типове данни, които се наричат основни (вградени) типове данни. Те са определени (дефинирани) в основното обезпечение на езика и обикновено за тях са има назначени специални ключови думи. Програмните езици позволяват дефинирането и на допълнителни типове данни, за които се специфицира множеството от стойности, които могат да приемат величините и операциите които могат да се извършват с тях.