Тема 1
Програмно осигуряване на компютърните системи

1. Представяне на информацията в компютърните системи

Информацията в компютърните системи се представя под формата на числа в **двоичен код** (двоична бройна система). В този вид, тя се обработва от компютъра.

Двоичната бройна система е в основата на ***Фон Нойманова архитектура*** и този принцип е заложен във всички изчислителни машини. Тя е удобна за кодиране на цифрова информация, тъй като чрез двете стойности 1 или 0 се представят следните данни: *наличие или липса на сигнал; високо или ниско ниво; двете състояния на тригера* и т.н.

Освен двоична, за изобразяване на данните в компютърните системи, се използват шестнадесетична и осмична бройна система. Тъй като 16 (2^4) и 8 (2^3) са степени на 2, преобразуването от двоична в шестнадесетична или в осмична бройна система и обратно, е сравнително бързо.

1.1. Единици за информация

За представяне на информацията в компютърните системи се използва двоична бройна система. Най-малкото количество информация, което може да се съхранява и съответно, основната мерна единица, е **1 бит** (**разряд**). С 1 бит се кодира 1 двоично число: стойност 0 или 1. Друга основна мерна единица е **байт** (на английски **byte**), която съкратено се записва **B**. Един байт съдържа *8 бита*. Байт е основна информационна единица, която се използва като мярка за количеството на информацията и се свързва с нейното вътрешното представяне в компютрите и техните устройства. Мерните единици за количество информация са стандартизирани съгласно международните стандарти ISO/IES 80000-13 и IEEE 1541, които също определят и производните мерни единици за количество информация. Използват се познатите десетични представки в система SI – *кило*, *мега*, *гига* и т.н., както и двоичните представки *киби*, *меби*, *гиби* и т.н.

Производни мерни единици за количество информация с десетични представки са следните:

1kB (кило байт) = 1000 B;

1MB (мега байт) = 1000^2 B;

1GB (гига байт) = 1000^3 B;

1TB (тера байт) = 1000^4 B;

1PB (пета байт) = 1000^5 B;

1EB (екса байт) = 1000^6 B;

1ZB (сета байт) = 1000^7 B;

1YB (йота байт) = 1000^8 B.

Според стандарта ISO/IES 80000-13 мерни единици за количество информация с двоични представки са следните:

1KiB (киби байт) = 2^{10} B = 1024 B

1MiB (меби байт) = 2^{20} B = 1024^2 B;

1GiB (гиби байт) = 2^{30} B = 1024^3 B;

1TiB (теби байт) = 2^{40} B = 1024^4 B;

1PiB (пеби байт) = 2^{50} B = 1024^5 B;

1EiB (ексби байт) = 2^{60} B = 1024^6 B;

1ZiB (себи байт) = 2^{70} B = 1024^7 B;

1YiB (йоби байт) = 2^{80} B = 1024^8 B.

Тъй като информацията в компютърните системи се представя под формата на числа в двоична бройна система, двоичните представки са по-удобни за мерните единици за количество информация. Но, поради широкото използване на десетичните представки, често неправилно точно те се употребяват като се има предвид двоичните. За това способства и един от стандартите (JEDEC), според който 1KB = 1024 (2^{10}) B. В случая, за представката *кило* се използва главна буква K.

1.1. Представяне на *цели числа*

Като цяло, информацията, която се обработва от компютърните системи може да бъде ***числова*** или ***текстова***.

Числовата информация се запомня в един или няколко байта, разположени на съседни адреси от паметта. Начинът по който се използват байтове в числовата клетка се нарича ***формат***. На различните видове числови данни съответстват различни формати.

Всяко въведено цяло число се преобразува в двоична бройна система, след което се записва в клетка памет на съответния адрес. Битовете (***разрядите***) на отделната клетка се използват по съответния начин: в най-старшия бит се разписва знака на числото (0 за положителните числа, 1 за отрицателните), а останалите се използват за неговата стойност. На фиг. 1.2 е показано представянето на числото +5 в един байт (8 разряда).

7	8	5	4	3	2	1	0
0	0	0	0	0	1	0	1
зн.				стойност			

Фиг. 1.2. Представяне на цяло число в един байт

1.3. Представяне на *реални числа*

…

1.4. Представяне на *текст*

Освен цифрова, компютърът обработва и текстова информация. Ето защо, възниква въпроса за представянето на текстови данни в компютърните системи. Кодирането на текстова информация е чрез числена, като на всеки символ съответства число. За представяне на текстова информация в компютърните системи са разработени различни стандарти, но стандартът, който се е наложил в световен мащаб е **ASCII**. Според този стандарт, на всеки символ съответства числова стойност. Съществуват две разновидности:

- 7 битов (стандартен) ASCII код;
- 8 битов (разширен) ASCII код.

С помощта на 7 битовия ASCII код се кодират 2^7 или 128 различни числови комбинации, което означава, че могат да бъдат кодирани 128 различни символа. Чрез 8 битовия ASCII код се кодират 2^8, или 256 различни символа.

Чрез посочените числови комбинации се представят:

- управляващи кодове, например управляващ код CR е с ASCII код 13;
- символите от латинската азбука – главни (A-Z), малки (a-z);
- цифрите от 0 до 9;
- специални символи, като . , : @ # $; % ^ & * и др;
- символи от националните азбуки; псевдо графични символи.

Друга, широкоизползвана съвременна система за представяне на текстова информация е уникод (Unicode). Това е компютърен стандарт за представяне и обработка на символите на почти всички съществуващи писмени знаци в света като всеки символ има свой уникален код. Тъй като символите са десетки хиляди (последната версия на уникод представя над 109 000 символа), за тяхното представяне са необходими повече от 1 байт. Обикновено за повечето символи се използват 2 байта, но някои азбуки, африкански, азиатски, американски, специални символи изискват повече от 2 байта.

Уникод дефинира 1 114 122 позиции в обхвата 0 – $10FFFF_{(16)}$. Обикновено, представянето на символ в Unicode започва със символ U, последвано от шестнадесетичната стойност, представяща символа. Например, U+0058 представя главна латинска буква X.

1.5. Използване на прав, обратен и допълнителен код

.....

2. Програмното осигуряване

Под термина **програмно осигуряване** се разбира: всички програми, които могат да бъдат изпълнени от компютърните системи. В България са добили гражданственост и английските термини: **software** - вместо програмно осигуряване и **hardware -** вместо апаратно осигуряване. Програмното осигуряване се разделя на: **базово** и **приложно**.

2.1. Базово програмно осигуряване

Към ***базовото програмно осигуряване*** спадат средствата за работа на компютърните системи, за разработване, тестване, проверка и изпълнение на приложни програми. Базовото програмно осигуряване се разделя на:

- ***Вътрешно***
- ***Системно***

Вътрешното програмно осигуряване функционално е свързано с вътрешните модули на компютъра и изпълнява роля на програмен интерфейс – връзка на компютъра с останалите програми. То осъществява следните действия: управление на периферните устройства; тест на системата; инициализиране на апаратните устройства; зареждане на операционна система. Към вътрешното програмно осигуряване спадат програмите от ***базовата входно-изходна система (BIOS)***.

Към ***системното програмно осигуряване*** принадлежат ***операционните системи***, ***драйверите***, като част от тях, както и различни програмни средства за създаване на приложни програми. Това са различни видове: ***редактори, транслатори (компилатори), асемблери, дебъгери***. Под термина ***драйвер*** (или ***драйверна програма***) се разбира програма, която служи за управление на устройство.

Към системното програмно осигуряване спадат и системните програми, които обслужват мрежите. Това е т.нар. ***мрежово*** или ***комуникационно програмно осигуряване***.

2.2. Приложно програмно осигуряване

Към приложното програмно осигуряване спадат различни ***пакети приложни програми***. То може да се класифицира като:

- *Специализирани програмни пакети*, като например CAD/CAM системи; информационни системи; счетоводни програми и др.
- *Многофункционални програмни пакети*, като текстообработващи системи, електронни таблици и др.

Приложното програмно осигуряване служи за решавате на определени технически, инженерни, икономически и други задачи от различни сфери на науката, икономиката, производството, здравеопазването. Разли-

ката между двата вида програмно осигуряване е, че докато специализираните програмни пакети се използват само в дадена конкретна област, многофункционалните намират приложение в различни области. Освен това, за работа със специализираните пакети се изискват познания в дадената конкретна област – машиностроене, текстилна промишленост, счетоводство и др.

3. Понятие за *операционна система*

Под операционна система се разбира *съвкупност от взаимно свързани програмни модули, които управляват ресурсите на изчислителната система*. Ресурси на изчислителна система са: процесори, памет, периферни устройства и др.

Операционната система представлява програмната среда за изпълнение на програмните пакети и се счита за връзка между апаратните средства и потребителя. Тя често се определя и като: *набор системни програми, които служат за:*

- Управление на устройства;
- Поддръжка на файлова организация;
- Осъществяване на диалога с потребителя.

Според броя потребители, с които работят операционните системи могат да бъдат класифицирани като:

- еднопотребителски – работят само с един потребител;
- многопотребителски – работят едновременно с много потребители.

Според броя на едновременно работещите приложения операционните системи се класифицират като:

- еднозадачни – позволяват изпълнението на една програма;
- многозадачни – позволяват изпълнението на много програми едновременно.

4 . Езици за програмиране. Транслатори

За да бъде изпълнена една програма, тя трябва бъде написана като поредица машинни инструкции т.е. да е на ***машинен език***. Под термина ***машинен език*** се разбира съвкупността (набор) от инструкции, които може да изпълнява процесора. В исторически аспект, това е първият език за програмиране.

Програмирането на машинен език е твърде трудоемък процес. Програма, написана на този език се състои от числа, които са кодове на операции и операнди (данни за операцията). При това, програмистът би трябвало детайлно да познава апаратните особености на процесора и компютърната конфигурация, за която пише програмата.

В исторически план, развитието на програмните езици продължава с различните видове ***асемблери***. Езикът ***асемблер*** (**Assembler**) е машинно-зависим език т.е. той зависи от вида на процесора. Различните процесори използват различни асемблери.

По същество, асемблерът е развитие на машинния език, като командите се представят не като числа, а във вид на ***мнемонични*** кодове. Например: **MOV** – инструкция за обмен да данни между регистри или регистри-памет; **ADD** – инструкция за събиране. Улеснения за програмирането е и използването на ***символни имена***, ***етикети***, както и ***макроси***, поради което понякога се среща термина *макро-асемблер*.

Въпреки че програмирането на асемблер е по-лесно отколкото на машинен език, програмистът трябва да е запознат с архитектурата на процесора и апаратните особености на компютъра. При съвременните програмни системи твърде рядко се налага писане на програми на асемблер. Обикновено, като език за програмиране се използва алгоритмичен език и евентуално, програмни модули, към който се поставят специални изисквания, например бързодействие, се пишат на асемблер. Тъй като и машинния език и асемблерът са езици, тясно свързани с апаратната част, те се наричат още ***програмни езици от ниско ниво***.

За разлика от тях, програмни езици, които са машинно независими, се наричат ***езици от високо ниво*** или още ***алгоритмични*** езици. Такива са: FORTRAN, BASIC, PASCAL, C, C++ и други.

За да бъде изпълнена от компютъра, програма, написана на алгоритмичен език, тя трябва да бъде преведена на машинен език. Това се извършва от програма, наречена ***транслатор***. Транслаторите биват два вида: ***интерпретатори*** и ***компилатори***.

Интерпретаторът е програма, която превежда и изпълнява ред по ред друга програма, написана на език от високо ниво. *Компилаторът* превежда програма на алгоритмичен език изцяло, като създава т.нар. ***обектен код***. *Обектният код* е програмата, преведената на машинен език.

Тема 2
Алгоритмизация. Базови структури на алгоритмите. Програмиране

1. Алгоритмизация

1.1. Понятие за алгоритъм

Алгоритъм е основно понятие в изчислителната техника. Според Кнут[1], алгоритъм интуитивно се определя като ***съвкупност от указания (инструкции, команди), при изпълнението на които се получава търсеният резултат***. Интерпретация на това определение е дефинирането на алгоритъма като ***крайна поредица инструкции, която еднозначно определя такъв процес на обработка на входните данни, чрез който да се получи търсения резултат***. Всяка инструкция определя вида на действието, необходимите величини за неговото извършване и указва следващото действие. Инструкцията се нарича още ***стъпка*** от алгоритъма.

1.2. Свойства на алгоритмите

Свойствата на алгоритмите определят тяхната същност, както и служат като ориентир дали дадено формално описание е алгоритъм или не. Алгоритмите притежават следните свойства:

- ***Дискретност***

Главна особеност на всеки алгоритъм е дискретния характер на определения от него процес. Алгоритъмът се състои от краен брой действия, наречени ***стъпки***. Всяка стъпка трябва да е съвсем проста (елементарна). Изискването към стъпките от алгоритъма определя следващото свойство – ***определеност***.

- ***Определеност***

Това свойство се изявява в това, че всяко действие от алгоритъма трябва да бъде ясно, точно и недвусмислено определено т.е. не се допуска стъпка от алгоритъма да не е еднозначно определена.

- ***Масовост***

Алгоритмите трябва да работят не само с дадени конкретни стойности на входните данни, а с един по-широк диапазон т. е. да са приложими за различни стойности на входните данни. Колкото по-широк е този диапазон, толкова по-универсален е алгоритъмът. Свойствата дискретност и масовост определят следващото свойство – ***потенциална приложимост***.

- ***Потенциална приложимост***

Безсмислено е да се съставят алгоритми, които нямат приложимост. В случая, става въпрос за потенциална, а не за реална приложимост, тъй

---

[1] *Кнут Д, Исскуство програмированния для ЭВМ. т.I, Москва, Мир, 1976.*

като, не се отчитат необходимите реални разходи на време и машинни ресурси.

Свойството потенциална приложимост има два аспекта: *приложимост към определени входни данни* и *крайност*.

- ***Приложимост към определени входни данни***

Това свойство се явява противовес на свойството масовост. Колкото и широк да е обхватът на входните данни, той не може да е безкраен т.е. алгоритъмът работи за определен диапазон на входните данни. В противен случай е възможно алгоритъмът да не може да бъде приложен.

- ***Крайност***

Свойството крайност определя, че алгоритмичния процес трябва да завърши след определен брой стъпки т.е. алгоритъмът има краен брой стъпки. **Крайност** се свързва със следващото свойство – резултативност.

- ***Резултативност***

При изпълнението на даден алгоритъм, при входни данни в дефиниционната област, трябва да бъде получен краен резултат. Безсмислено е да се съставят алгоритми, които не дават никакъв резултат.

1.3. Начини за описание на алгоритмите

Съществуват три начина за описание на алгоритмите:

- на естествен език;
- графично, чрез използване на ***блокови схеми***;
- чрез ***псевдо-код***.

Естественият език е крайно неподходящ за представяне на алгоритми за решаване на задачи със средствата на компютърната техника и затова не се използва като средство за описание на алгоритми на програми.

Графичният начин за описание на алгоритмите се наложил отдавана, тъй като е удобен за документиране на алгоритми на програми и освен това е стандартизиран. При този начин, алгоритъмът е представен във вид на ***блок-схема***.

При използване на ***псевдо-код***, алгоритъмът се описва с определен набор думи, които дефинират дадени типове конструкции. Като начин за описание на алгоритми, в сравнение с графичния, псевдо-кодът е по-късно появил се, той е по-компактен, но за съжаление не е стандартизиран.

1.4. Блокови схеми

Въпреки че използването на графичния начин за описание на алгоритмите, чрез блок-схеми, е по-стар в сравнение с описанието чрез псевдокод, той се е наложил поради това, че е лесен за съставяне и възприемане на алгоритъм. Като начин за документиране, описанието на блок-схемите

е стандартизирано, което е още едно предимство на графичното описание на алгоритми.

При графичното описание, основните действия (стъпки) от алгоритъма се описват чрез графични блокове. Типът на блока определя вида на действието. Блоковете се съединяват с линии, завършващи със стрелка. Стрелките определят посоката на действието.

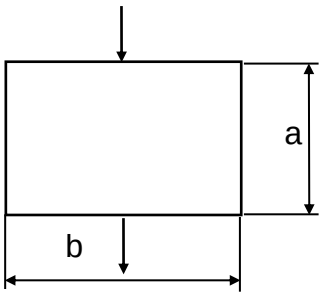
Според нормативните изисквания, възможно е линиите да не завършват със стрелка. По определение се счита, че *посоката на действието върви отгоре надолу, отляво надясно*. В тези случаи, не е задължително поставяне на стрелки. При изменение на така дефинираните посоки, поставянето на стрелки е задължително.

Типовете блокове, както и начина за описание на програми са стандартизирани чрез стандарт БДС ISO 5807[2]. Стандартът определя общите положения – изискванията, условните графични означения и правилата за документиране на програмите, независимо от тяхното предназначение.

Размерите на блоковете в алгоритъма, както и на свързващите ги стрелки, трябва да са еднакви. Това са размери **a, b**, като **a=10,15, 20, 25, … mm, b=1.5a**. Например, при избран размер **a=20mm**, размер **b е 30mm**.

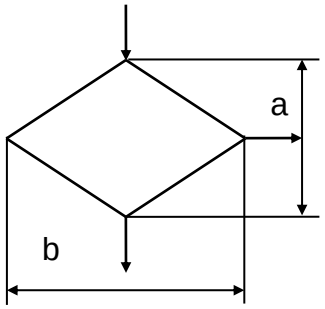
Най-често използвани са следните видове блокове:

- ***Обработка (процес)***



Блок ***обработка*** служи за изпълнение на една или повече операции, в резултат на което се изменя стойността, формата на предаване или разположението на данните. Блокът се нарича още ***функционален***. Той има един вход и един изход.

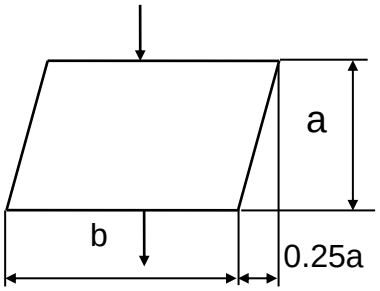
- ***Разклонение***



[2] *Български стандарт БДС ISO 5807 "Обработка на информация. Символи за документация и споразумения за данни, блок-схеми на програмни системи, схеми на програмни мрежи и схеми на системни ресурси", Български институт за стандартизация, Декември 2004 г.*

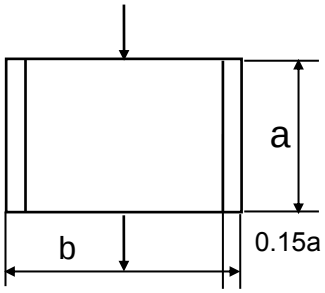
Блок ***разклонение*** служи за извършване проверка на условие, според което се избира посока на изпълнение на програмата. Блокът се нарича още ***логически***. Той има един вход и два или повече изхода.

- ***Вход-изход***



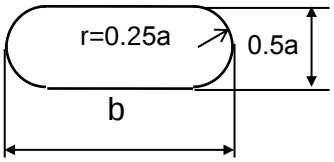
Блокът служи за въвеждане и извеждане на данни и е с един вход и един изход. Поради това, че се използва едновременно за две операции, за удобство, в блока се записва вида на операцията: вход (или въвеждане), изход (или извеждане).

- ***Подпрограма***



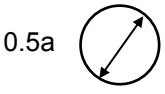
Блокът е подобен на блок ***обработка*** и обозначава отделно описана процедура.

- ***Начало, край***



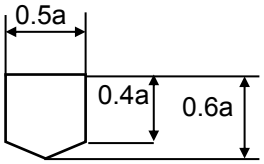
Блокът служи за начало, край или прекъсване на процеса на обработка на данните или изпълнението на програмата. Когато блокът се използва за начало, той няма вход, има един изход. Когато се използва за край е обратно, има един вход и няма изход.

- ***Съединител***



Блокът посочва връзка между прекъснати данни. В кръгът се записва с цифра, която обозначава връзката между отделни клонове на алгоритъма.

- ***Междустраничен съединител***



Блокът свързва прекъснати линии на потока за част от схемата, разположени на различни страници. В блока се записва буква или цифра (почесто буква), която обозначава връзката между отделните страници.

2.4. Видове алгоритми

Най-общо алгоритмите се разделят на три вида:

- **линейни**;
- **разклонени**;
- **циклични**.

Линейни са тези алгоритми, при които се спазва линейната последователност на изпълнение на действията, по реда на тяхното записване. Линейни алгоритми, като цялостно решение на дадена задача се срещат изключително рядко. Обикновено, те са част от друг по-голям алгоритъм.

При ***разклоненията*** алгоритми се нарушава линейно-последователното изпълнение на операциите. Тези алгоритми съдържат блок за разклонение и част от действията, в зависимост от дадено условие, в някои случаи ще се изпълнят, а при други – няма да се изпълнят.

Циклични са тези алгоритми, в които част от действията се изпълняват многократно. Това многократно изпълнение се нарича ***цикъл***.

Всеки цикъл се състои от:

- ***тяло*** – групата действия, които се повтарят многократно;
- ***условие -*** проверка за изход от цикъла.

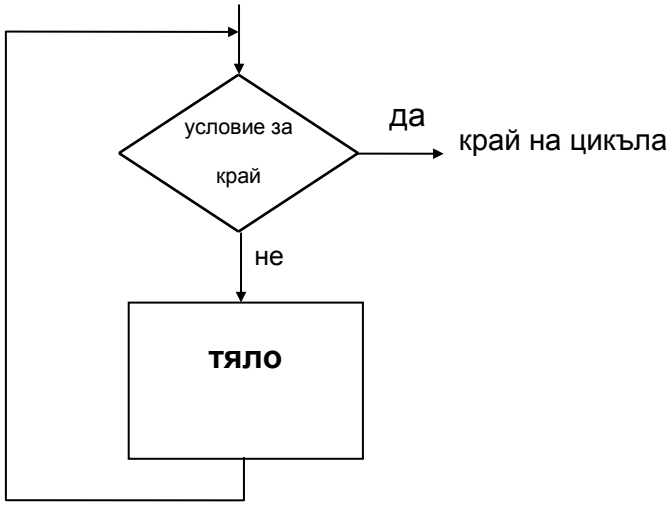
Когато условието за край на цикъла се изпълни, прекратява се по- нататъшното изпълнение на групата действия от тялото. Това означава, че е необходимо, в тялото на цикъла да има действие, което ще промени стойностите, участващи в условието за край. Когато, поради грешки в алгоритъма, условието за край никога не е изпълнено, цикълът се превръща в ***безкраен***.

Организирането на безкрайни цикли по същество е грешно, тъй като противоречи на свойството ***крайност*** на алгоритмите, но, за съжаление, е една от най-често срещаните грешки при програмиране.

В зависимост от това, как са разположени тялото на цикъла и проверката за изход, циклите се разделят на:

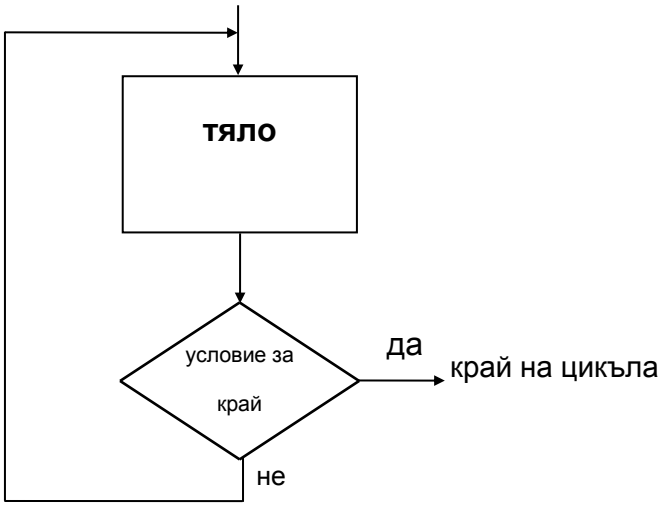
- ***цикъл с предусловие;***
- ***цикъл със следусловие (пост условие).***

При цикъл с предусловие, условието за край е преди тялото (фиг. 2.1), докато при цикъл със следусловие, условието е след тялото на цикъла (фиг. 2.2).



Фиг. 2.1. Цикъл с предусловие

Основната разлика между двата вида цикли е, че *докато цикълът с предусловие е възможно никога да не се изпълни, ако още в началото условието е изпълнено, цикълът със след условие ще бъде изпълнен поне веднъж*.



Фиг. 2.2. Цикъл със следусловие

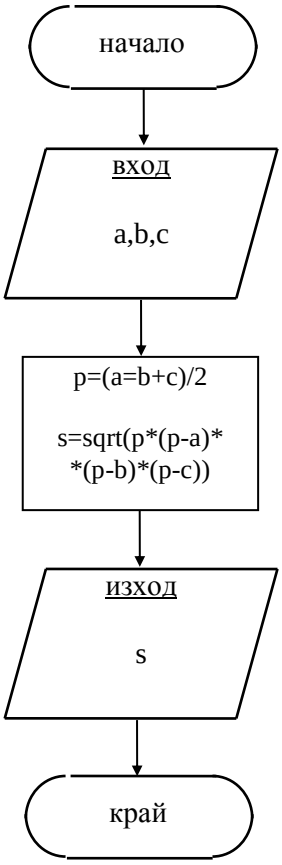
Според броя на повторенията, циклите могат да се разделят и на:

- цикъл с известен брой повторения;
- цикъл с неизвестен брой повторения.

В случая, дали броя на повторенията е известен или не, се определя при изпълнението на алгоритъма, преди да се влезе в цикъла.

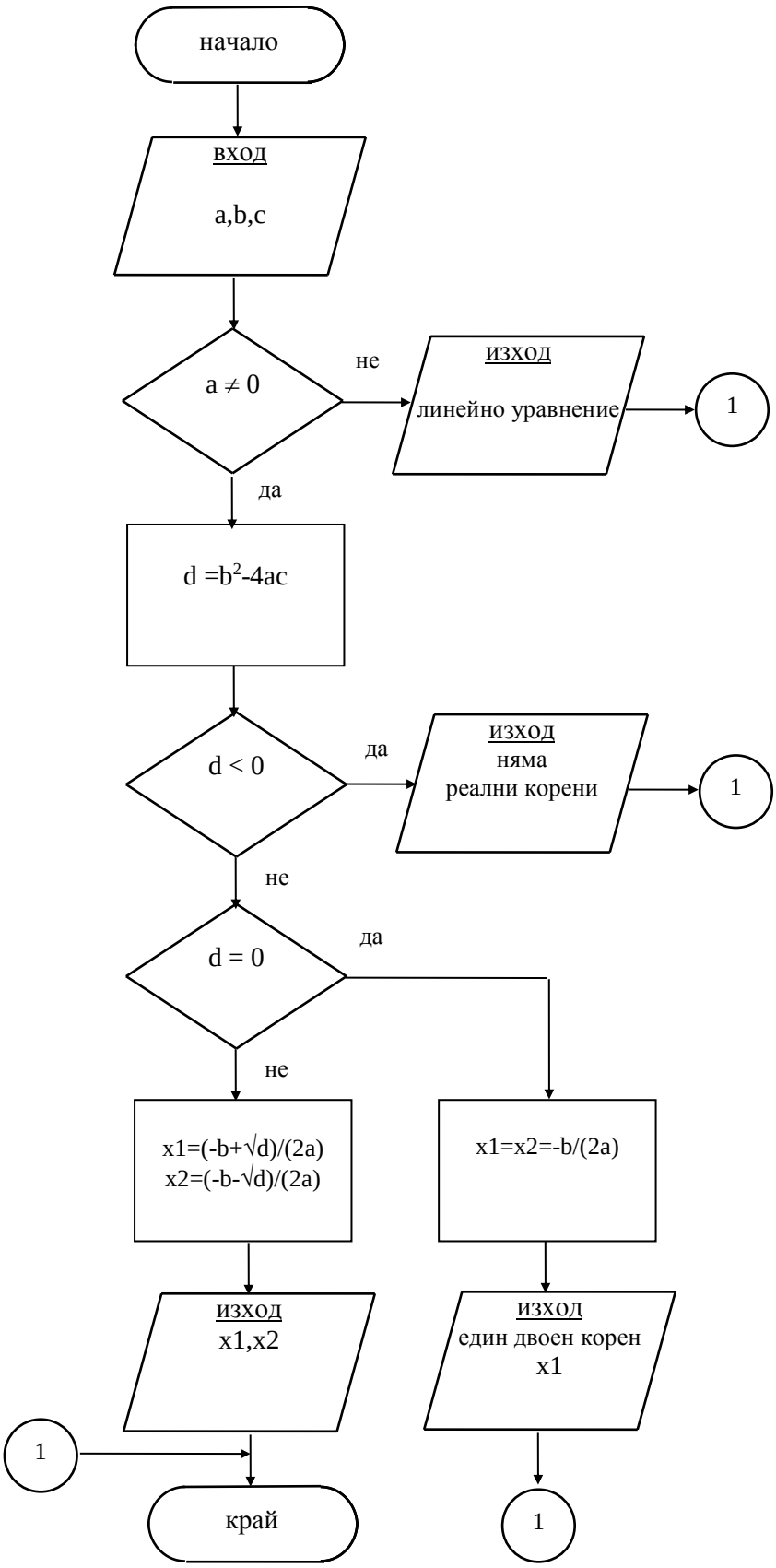
Примери за реализация на алгоритми:

Пример на програмна задача с линеен алгоритъм е: *Да се намери лицето на триъгълник по зададени три страни, като се счита, че входните данни са въведени коректно.* Алгоритъмът на задачата е даден на фиг. 2.3.



Фиг. 2.3. Пример за линеен алгоритъм

Пример за разклонен алгоритъм:
В разклонените алгоритми, според зададено условие се извършва *разклонение* към един или към друг клон от алгоритъма. Типичен пример за задача с разклонен алгоритъм е намирането на корените на квадратно уравнение по зададени стойности на коефициентите. Алгоритъмът на задачата е даден на фиг. 2.4.

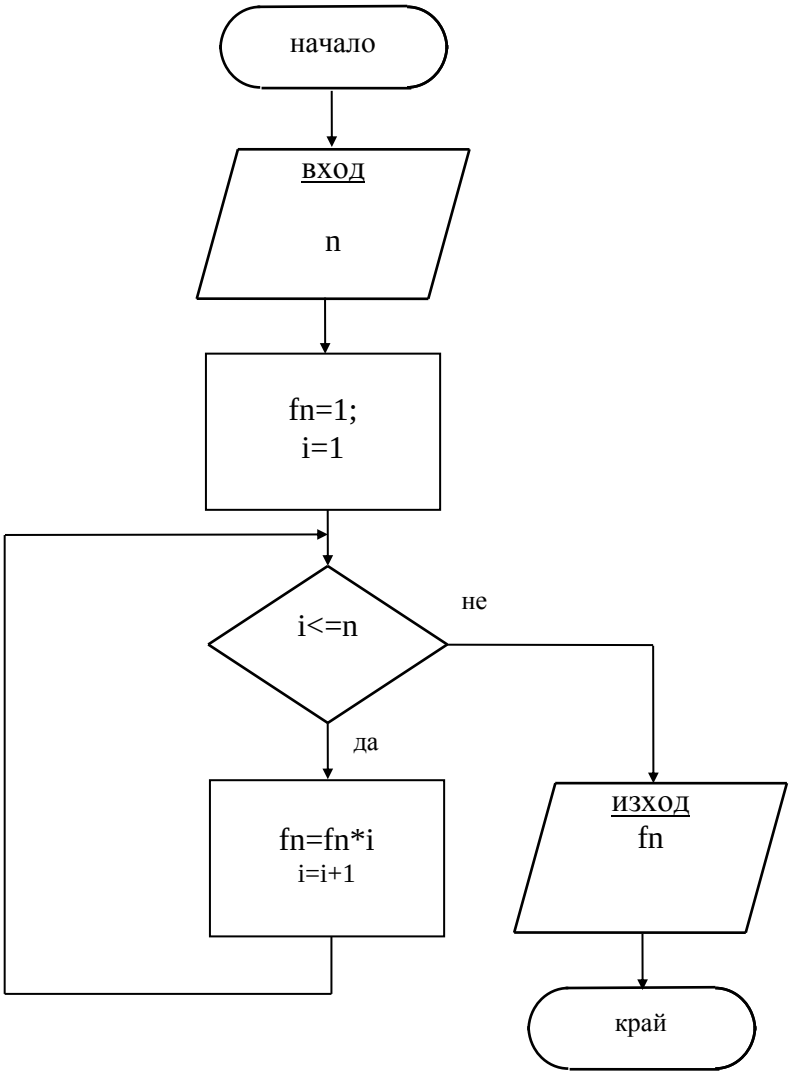


Фиг.2.4. Пример за разклонен алгоритъм

Пример за цикличен алгоритъм:

При цикличните алгоритми част от действията се повтарят многократно. Тези действия съставляват т.нар. **тяло на цикъла**. Важна част от всеки един програмен цикъл е условието за край. Чрез него се определя при какво условие да се прекрати изпълнението на тялото на цикъла. Ако условието за край липсва или ако е зададено некоректно, тогава цикълът се превръща безкраен.

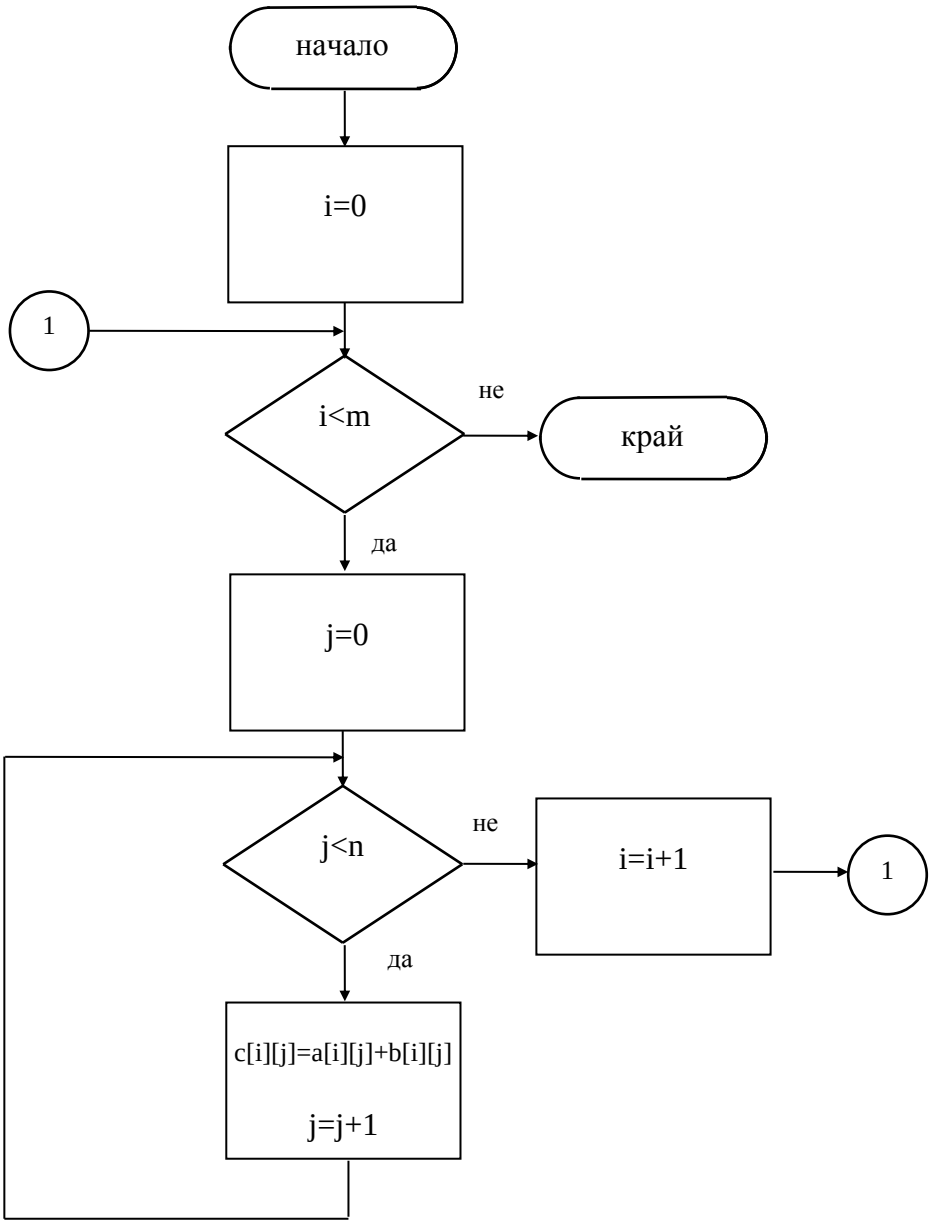
На фиг. 2.5. е представен пример за цикличен алгоритъм - задачата за намиране на n! (n факториел). Стойността на n се въвежда от клавиатурата. Този алгоритъм е типичен пример за цикличен алгоритъм.



Фиг. 2.5. Пример за цикличен алгоритъм

Много често при реализацията на циклични алгоритми се срещат т.нар. ***вложени цикли***. В този случай, в тялото на даден цикъл се организира друг цикъл. Пример за такъв алгоритъм е представения на фиг. 2.6, чрез който се реализира задачата за сумиране на две матрици. Входни матрици са a, b; изходна матрица е c. Счита се, че елементите на

матриците са предварително въведени. Стойностите за m, n определят размерностите на матриците по редове и стълбове, съответно. Индексите на елементите се изменят както е според C и C++, от 0 до m-1 и от 0 до n-1. Цикълът на изменение на елементите по стълбове, реализиран с изменението на брояча j, е вложен в цикъла на изменение на редовете, реализиран с изменението на брояча i. При тази реализация, цикълът на изменение на редовете е ***външен***, а цикълът на изменение на стълбовете е ***вътрешен***. Важно условие за реализацията на вложени цикли е те да не се пресичат т.е. цикълът, който е започнал първи, да свърши последен.



Фиг. 2.6. Пример за цикличен алгоритъм с вложени цикли

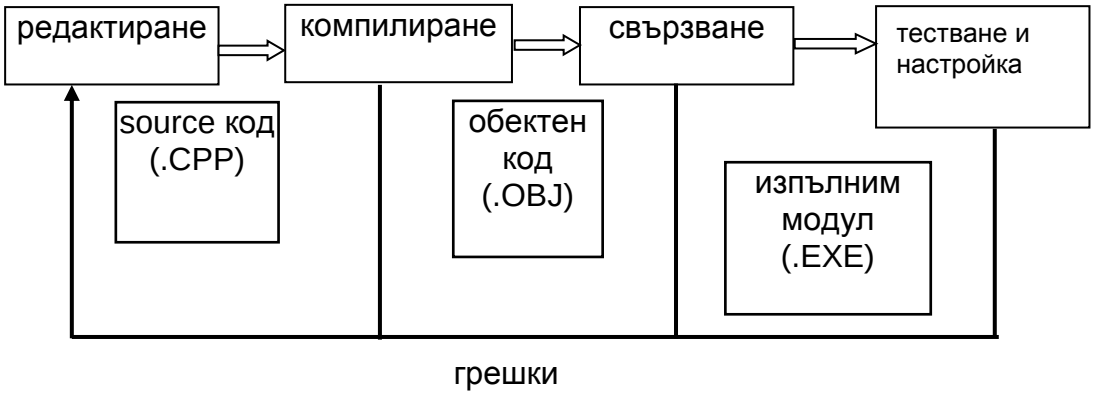
2. Програмиране. Етапи от развитието на програмите

След създаването на алгоритмите, както беше споменато, на етапа ***Реализация*** се създава, тества и настройва програмно осигуряване.

Процесът на създаване на всяка програма преминава през следните основни етапи:

- ***създаване и редактиране на първичния код;***
- ***компилиране;***
- ***свързване;***
- ***тестване и настройка.***

Този процес на създаване на програмен код е изобразен на фиг. 2.7.



Фиг. 2.7. Създаване на програмен код

2.1. Създаване и редактиране на първичния код

На този етап, въз основа на разработения алгоритъм, се създава ***първичния*** (***source***) код на програмата. Това е програмата, записана на алгоритмичен език. Първичният код се записва като ASCII файл. Разширението на файла подсказва на какъв програмен език е записана програмата. Например, за език C, разширението на файла е **.C**, за език C++, разширението е **.CPP**, за език PASCAL се използва разширение **.PAS**, за BASIC - **.BAS**.

2.2. Компилиране

На този етап, се превежда първичният код на програмата, написан на алгоритмичен език, на машинен език. Програмата-транслатор, която осъществява превода на първичния код се нарича ***компилатор***.

Компилаторът анализира първичния код и когато открие синтактични грешки или несъответствия, издава съответните съобщения за грешки или

предупреждения. При грешки в source-кода е необходимо неговото повторно редактиране, с цел тяхното премахване. След това, първичният код отново се компилира. Процесът продължава, докато не бъдат отстранени всички грешки.

При успешно компилиране, се създава т.нар. ***обектен код*** – програма, преведена на машинен език. Обектният код се записва във файл с име, което съвпада с името на първичния код, но с разширение **.OBJ**. Възможно е, името на файла с обектния код да не съвпада с името на файла с първичния код. Тази промяна на имената рядко се практикува.

При разработката на програмно осигуряване се използва т.нар. разделно компилиране. При разработката на даден програмен проект, първичният код е записан в множество файлове. Всеки един от тях може да се компилира самостоятелно, независимо от другите. Този процес е известен като разделно компилиране. В някои случаи получения обектен код от един програмен файл може да се използва самостоятелно, в други програмни системи, независимо от останалите.

2.3. Свързване

На етап ***свързване***, обектният код се свързва със системните библиотеки и други обектни модули. Програмата се прави изпълнима под управлението на операционна система. Свързването се осъществява с програма ***свързващ редактор (linker)***.

При успешно приключване на етапа, свързващият редактор създава т. нар. ***изпълним модул***, записан като изпълним файл за съответната операционна система. За операционни системи MS DOS и MS WINDOWS, изпълнимите файлове са с разширение **.COM** или **.EXE**. При условие, че на етап свързване възникнат грешки, необходимо е първичният код отново да бъде редактиран и повторно компилиран.

2.4. Тестване и настройка

Програмата, която е получена след етапа свързване и записана като изпълним файл (.EXE) е готова за изпълнение, но това не означава, че тя работи коректно. Необходимо е, програмата да бъде тествана дали при съответните входни данни, се получават верни резултати. За тази цел, още преди създаването на алгоритъма, са разработени тестови примери. Въз основа на тези примери, се тества работоспособността на програмата.

В голяма част от случаите, първоначално разработеното програмно осигуряване работи некоректно. Грешките се дължат или на логическо несъответствие с алгоритъма, или на неверен алгоритъм. В етапа ***тестване и настройка*** се откриват и отстраняват логическите грешки в програмата. Разработени са специални програми за тестване и настройка, които се наричат ***дебъгери***. Дебъгерите използват следните средства за тестване на програми:

- постъпково изпълнение на програмата – програмата се изпълнява стъпка по стъпка, като се следи дали отговаря на алгоритъма;

- задаване на точки на прекъсване – програмистът задава място на прекъсване на изпълнението на кода в редове, за които е възникнало съмнение за грешка;
- наблюдение на променливи – при постъпковото изпълнение се следи как се изменят стойностите на по-важните променливи.

Като резултат от всички тези етапи, се създава работоспособно програмно осигуряване, което се документира според изискванията на действащите стандарти.

При разработката на програмно осигуряване се използва интергирана среда за програмиране, която включва всички системни програми, необходими за разработка и тестване на програмни продукти. В това число са включени текстов редактор, компилатор, свързващ редактор и дебъгер.