Алгоритми. Определение, свойства, видове

**Понятие за алгоритъм :**  
Един от най-важните етапи при решение на задача е съставянето на нейният алгоритъм.

*Думата алгоритъм произлиза от името на един от най-големинте учени на Средна Азия* ***Мухамада ибн Мус ал-Хорезми****.  
През 1983 година се отбеляза 1200-годишнина от неговото рождение.   
Той е написал редица кники  по аритметика и алгебра и често   
е наричен бащата на алгебрата, смята се че той поставя нейните основи.  
Ал-Хорезми показва на света с книгите си арабските цифри.  
През ХІІ-ХVІІ век в Европа е имало остра борба междуподържниците на римските цифри и тези предпочитащи арабските.   
Но в крайна сметка през ХVІІ окончателните победителистават АЛГОРИТМИЦИТЕ.*



**Алгоритъм е точно упътване за решаване на един проблем или определен вид проблеми. Той се състои от крайна поредица от указания, които се изпълняват едно след друго и отчасти повтарят в определен ред.**

*При извършване на различни ежедневни действия, при изпълнения на дейности в науката и практиката, човек съблюдава някакви правила,което налага използването на алгоритми при изпълнение на съответната дейност. Редица човешки дейности са алгоритмични по своята природа( разговарянето по телефон, пресмятането в магазин, отиване на училище). Често като пример за алгоритми се посочват готварски рецепти.*

**Свойства на алгоритмите:**  
Алгоритмите притежават следните свойства:

1. **Определеност** - решението на задачата по зададен алгоритъм може да се извърши многократно, по всяко време и от различни хора и при това за едни и същи начални данни ще се получи еднакъв резултат;
2. **Резултатност** - всеки алгоритъм гарантира получаване на краен резултат след изпълнението на краен брой указания.
3. **Дискретност** - алгоритъмът се състои от определени отделни действия, като едва след изпълнението на текущото действие може да се пристъпи към изпълнението на следващото действие;
4. **Масовост** – болшинството алгоритми могат да се изпълняват при достатъчно разнообразен набор от изходни данни, т.е. използват се при решаване не само на една, а на цял клас подобни задачи.
5. **Крайност** - изпълнението на алгоритъма трябва да завършва след краен брой стъпки.

**3. Видове алгоритми -** В зависимост от своята структора алгоритмите могат да бъдат:

1. **Линейни** –При тях последователността от действия винаги е една и съща. Всяка стъпка на алгоритъма има само една предходна и една следваща.
2. **Разклонени** – Последователността от извършвани действия зависи от стойността на входните данни.
3. **Циклични** – Определена част от действията се повтаря многокротно докато не се изпълни зададеното условие.

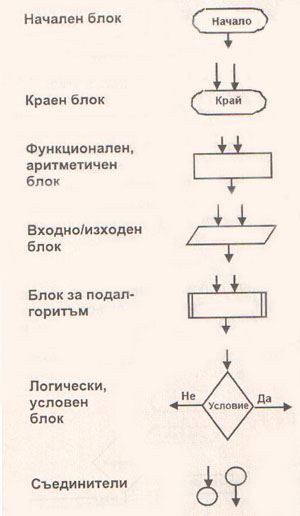
**Методи за представяне на алгоритми**

Всяко указание в даден алгоритъм описва точно определено действие - пресмятане, вземане на решение, преход към следваща стъпка и др. Съществуват различни средства на описание на алгоритмите:

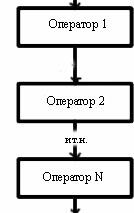
* 1. словесно описание;
  2. описание чрез блок-схеми;
  3. описание чрез език за програмиране;

**Словесното описание** представлява набор от указания, в които чрез думи от някакъв естествен език са посочени действията, които трябва да бъдат извършени. Възможно е при описанието да се използват означения от съответната предметна област. Този начин на описание не е много популярен, тъй като за много думи от естествените езици съществува нееднозначно тълкуване.  
**Описанието чрез блок-схеми** е удобно, тъй като се предлага визуална представа на логическите връзки между отделните действия в алгоритъма. Описанието се извършва от определени символи, наречени блокове, като всеки символ има точно определен смисъл. Блоковете имат вида на геометричните фигури: правоъгълник, ромб, успоредник. Последовател­ността се задава чрез стрелки. Блок-схемата започва с начален блок, в който се записва указанието, с което започва алгоритъма.

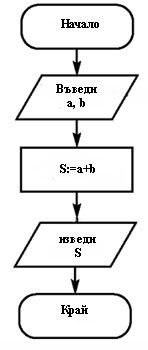
* + - 1. **Основни блокове, използвани в блок-схемите.**



**Блок-схеми на видовете алгоритми**

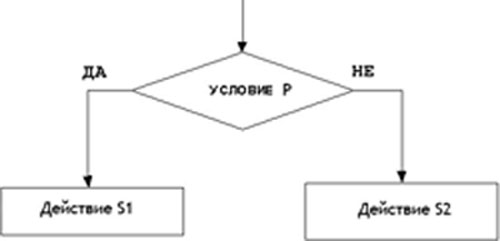


***Линейни алгоритми*** са тези,  
при които последователността  
от действия е една и съща за  
всички  входни данни.  
Всяка стъпка на алгоритъма  
има само една предходна   
и 1 следваща.



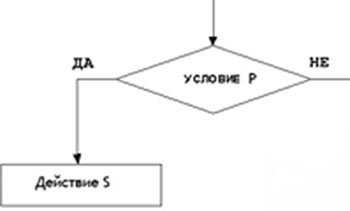
Нека да разгледаме алгоритъма за събиране на 2 числа **a** и **b.**  
Стъпка 1: Начало.  
Стъпка 2: Въведете и запомнете стойностите на **a** и **b**.  
Стъпка 3: На променливата S присвояваме сбора на **a** + **b.**  
**S:=a+b**  
Стъпка 4: Изведете стойността на S като резултат.  
Стъпка 5: Край.

***Разклонени алгоритми* –** Много често се налага да избираме между няколко възможни варианта. Какво решение ще вземеме обикновенно зависи от някакво условие.  
Разклонените алгоритми съдържат команди, които в зависимост от изпълнението или не на дадено условие определят кои са следващите за изпълнение команди. Подобни условия се наричат логически.   
Съществуват две форми на разклонен алгоритъм:



1. Кратка – Използва се тогава,   
   когато е необходимо да се   
   изпълни действието S само

при определено условие Р.   
Ако условието не е изпълнено,   
то командата не се изпълнява.



2. Пълна – Ако се изпълни  
условието Р, се извършва  
действие S1, а ако не се  
изпълни действие S2.

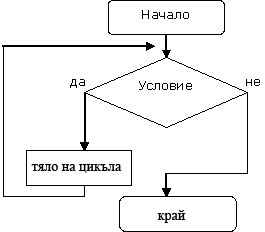
***Циклични алгоритми*** – При тях част от стойностите се повтарят многократно за различни стойности на променливите.

Групата от блокове, които са описани веднъж, но се изпълняват многократно се наричат **ТЯЛО НА ЦИКЪЛА**.

При тях **задължително има условен блок**. А необходимо изискване е да се осигури край на цикличните повторения (излизане от цикъла).

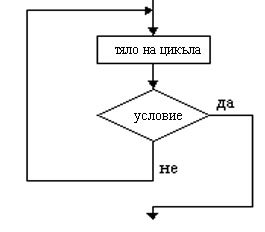
В тялото на цикъла, трябва да се променя стойността поне на 1 променлива участваща в условието за край.В противен случай ще се получи безкраен цикъл.

Пример:  
Пример за цикъл е задачата за намиране сумата на редица от числа. Сумиране на голям брой числа може да се използва в магазин, когато искате да проверите сметката си, или пък класният ръководител определя средният успех на класа.



Циклите биват 3 вида:

**1.С предусловие –**  
при тях първо се проверява условието   
и, ако то е изпълнено, се изпълняват  
операциите от тялото на цикъла.  
Ако не е изпълнено още в началото  
е възможно тялото на цикъла да не  
се изпълни нито веднъж.



**2.С постусловие –**   
при тях първо се изпълняват   
операциите от тялото на цикъла   
и след това се проверява условието   
за продължаване на цикъла.

Ако то е изпълнено се връща за   
повторно изпълнение на тялото,   
ако не е се излиза от цикъла.

**3.С параметър** – това е цикъл със зададено количество на повторенията.

Във всеки един цикъл има 4 елемента, които се търсят:

* + Установяване на началното състояние ( началните стойности);
  + Тяло – група от оператори, които се изпълняват многократно;
  + Условие за изход от цикъла;
  + Обновяване на цикъла (брояч), посредством него става промяната на цикъла.

**Алгоритмични езици. Елементи на езика. Видове езици за програмиране**

**Oписание чрез език за програмиране** - Представените алгоритми, реализирани с блок-схеми не могат да се изпълняват директно от компютрите, понеже са неразбираеми за тях. Езиците за програмиране описват алгоритмите във вид на компютърни програми, които могат да бъдат изпълнени от компютър

Алгоритми, предназначени за изпълнение от компютър се наричат **комюптърни алгоритми.**

**Програмата** представя компютърния алгоритъм във форма и вид, подходящи за изпълнение от компютър, чрез специално създадени за целта **езици за програмиране**.

Компютрите могат да изпълняват единствено разбираеми за тях команди.**Машинна програма** е програма, която е написана на **машинен език** / двоичен код чрез 0 и 1/, който е специфичен за дадения вид компютър и се намира в паметта му.

**Езикът на програмиране ЕП представлява език за записване и разпространение на компютърните алгоритми под формата на програми.**

Описанието на алгоритмите чрез **езици за програмиране** представлява замяна на естествения език с изкуствен. В този изкуствен език са въведени строги правила за записване на изреченията и в него се използват фиксиран набор от стандартни думи.

Една програма написана примерно на езика Паскал, не може веднага да се изпълни от компютъра. /Компютърът непосредствено може да изпълнява само команди и задачи въведени **на машинен език,** т.е. двоичен код. Алгоритъм написан на машинен език се нарича **машинна програма**. /

Програмата, написана на ЕП трябва да се преведе на машинен език. Преводът на програмата се извършва от друга програма наречена ТРАНСЛАТОР.  
Всеки език, всяка негова версия имат свой транслатор. Така работата по превода на машинен език е предоставена на самия компютър, който изпълнява програмата-транслатор.

**Транслатори** (компилатори и интерпретатори) са програми – преводачи от ЕП на машинен език. Преобразуват програмата в машинна програма (МП).

Те също така анализират текста на програмата, разпознават езиковите конструкции и съобщават грешките.Това, че транслаторът не е открил грешка, не е гаранция, че програмата е правилна. Възможно е програмата да съдържа грешки от логическо естество. Най-добре е да се тества програмата с предварително подготвени контролни данни, за които е известно какъв резултат се очаква.

Съществуват два вида транслатори:

* 1. **Интерпретатори.** Чрез тях преводът и изпъпнението на програмата протичат успоредно. Интерпретаторите превеждат програмата всеки път при стартирането и за изпълнение. Машинният превод не се запазва.
  2. **Компилатори.** Чрез тях програмата се стартира на два етапа

**1етап:** превеждат програмата на машинен език и създадената машинна програма се съхранява като изпълнима програма;   
2 **етап -**изпълнява се машинната програма.



**Класификация на езиците** - съществуват голям брой езици за програмиране, които могат да се класифицират по различни признаци.

**Машинни езици** – тези езици са достъпни за компютрите и трудно разбираеми за хората. Те са езици от НИСКО НИВО и са валидни само за конкретен процесор.

Всеки тип компютър има свой език за програмиране и може да изпълнява само програми, чиито оператори са записани само на този език. В машинния език на всеки оператор съответства определена операция, която се изпълнява от компютъра.

**Асемблерни езици** - близки до машинния, но някои команди за аритметични операции се записват с думи;

**Процедурно-ориентирани езици** – от по високо ниво. При тях програмист описва алгоритъма за решаване на дадена задача, а специална програма превежда това описание в машинни инструкции.

**Обектно-ориентирани езици** - те са от най-високо ниво до момента. Съдържат стандартни ви­зуални обекти, които се ползват при създаване на програмите. Такива езици са Visual Basic, Delphi, C++ и др.

**Елементи на езиците** – езиците за програмиране се характеризират с азбука, синтаксис и семантика.  
За да бъдат езиците за програмиране удобни за хората, както и лесно да се възприемат от компютъра се налага те да имат точно определени правила **(синтаксис)** за образуване и въвеждане на отделните програмни конструкции, а от друга да са близки до ползваните от хората езикови средства като думи, числа, формули и т.н. **(азбука).**  
**Семантиката** включва значението, смисъла и съдържанието в които е допустимо да бъдат свързани елементите на езика.

**Алгоритмични езици**

**Процедурно - ориентирани езици. Предимства и недостатъци**

*Тези езици ползват думи от английския език и познатите математически символи. Съдържат готови стандартни подпрограми (процедури), които са достъпни за ползване при създаване на програмите.   
Техен родоначалник е Fortran, a по-късните му представители са Basic, Pascal, C.  
FORTRAN (от FORmula TRANslation). Създаден през 1954г. oт Дж. Бекус и Колектив сътрудници от IBM, САЩ. Предназначен**е зa**решаване на научно-технически задачи и е удобен за числени пресмятания. Създадени са многобройни транслатори за този език;  
COBOL (от Common Business – Oriented Language). Създаден през 1960-1961г. в Държавния комитет на производителите и потребителите на компютри в САЩ. Ориентиран е към обработка на икономическа информация. Широко разпространен език за програмиране, особено в САЩ;  
BASIC (от Beginner’s All-Purpose Simbolic Instruction Code). Създаден е през 1965г. от Дж. Кемени и Р. Kурц в Дартмутския колеж в САЩ за целите на обучението по програмиране. Получава широко разпространение с появата на микрокомпютрите;  
PASCAL (по името на френския учен Блез Паскал). Създаден е през 1968г. от Н.Вирт във Федералния технологичен институт в Цюрих, Швейцария. Предназначен е за образователни цели, процедурно-ориентиран език с много различни транслатори за него;  
PROLOG (от PROgramming in LOGic). Създаден през  1971-1972г. от Ал. Калмерос в Университета на Марсилия, Франция. Предназначен е за целите на логическото програмиране и намира приложение в областта на изкуствения интелект;  
С,  този език  за програмиране  е създаден от Д. Ритчи през 1972г. във фирмата Bell Laboratories, САЩ. Това е език от  високо ниво, който съчетава възможностите за работа на машинно ниво. Един от най-бързо разпространяващите се* ***езици*** *за програмиране.*

***Обектно-ориентирани езици*** *– значително разпространени в последните години. подходящи за изграждане на сложни, диалогово-работещи софтуерни системи, включително и в мрежови среди.Техен  основен градивен елемент са обектите. Такива са С++, JAVA, Visual Basic и др.*

**Структурно програмиране**

**Език за програмиране Pascal**

Езикът за програмиране Pascal е създаден е от швейцарскияучен Никлаус Вирт през 1970г.





Наречен е на френският   
математик Блез Паскал,   
който още през 1642 г.   
изобретил първата механична  
сметачна машина.

 Този език е особено подходящ за първоначално обучение по програмиране.   
*Огромна роля в масовото разпространение на езика Паскал  е изиграла компанията Borland International. Тя е съумяла да създаде знаменитата Turbo-среда. Turbo в перевод от английски означава ускорение. Компилаторът, участващ в състава на  Turbo Pascal, много бързо превежда програмата от език за програмиране в машинен код.*

Тази среда за програмиране на ЕП Паскал осигурява цялата техническа работа - въвеждане на програмата, откриване и сигнализиране за грешки, помощна среда, изпълнение на програмния код и др.

**Азбука на езика Pascal –** Всяка програма на езика Pascal се записва като последователност от символи от неговата азбука.

Както всеки език и Pascal има азбука и препинателни знаци, от тях се изграждат думи и от думите според граматика - по сложни фрази (в случая програми).

**Азбуката на** езика включва:

* 1. Малки и главни букви на латинската азбука : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z, a b c d e f g h i j k l m n o p q r s t u v w x y z. В Pascal не се прави разлика между големи и малки букви;
  2. Цифри :          0 1 2 3 4 5 6 7 8 9
  3. Знаци за операции: + - \* / > >= < <= <> =
  4. Специални знаци: := -> ^
  5. Разделители и скоби: () [] {} : ; , .’

Двойките символи >=, <=, <>, : = не бива да се разделят една от друга с интервал, нов ред или табулация.  
**Думи в** **Pascal -**  От азбуката на езика се изграждат думи. Всеки език за програмиране използва определена съвкупност от думи наречени **служебни**:

* 1. **Идентификатори**- Те служат да именуват различни неща в програмата, като например променливи, подпрограми, типове, записи и др. Те се измислят от програмиста. Идентификаторите представляват последователност от букви, цифри и долно тире, които задължително започват с  латинска буква. Например a, ab, a12, first\_name. В една и съща програмна част не бива да има дублиране на идентификатори (не забравяйте, че не се прави разлика между малки и големи букви).
  2. **Запазени думи** – това са такива идентификатори, които се използват в програмите по предварително определен начин  и които НЕ могат да бъдат използвани по друг начин. Посредством тях се записват основните конструкции на операторите, използвани в езика. Тези думи са около 35 на брой –PROGRAM, BEGIN,CASE,CONST,VAR,DO, END и др.
  3. **Стандартни думи**- това са такива идентификатори, които могат да се използват в програмите по предварително определен начин, НО могат да бъдат използвани по друг начин, например като обикновенни идентификатори. Те са около 37 на брой – read, write, real и др.

***Пример****: Описанието* ***var begin:real*** *e грешно, тъй като запазената дума begin не може да се използва като име на променлива. Описанието* ***var read:real*** *по принцип е допустимо, защото* ***read*** *е стандартна, но не и запазена дума.*

* + **Разделители** – Служат за разделяне на думите и операторите.  Думите се разделят чрез интервал, а операторите с **;**.
  + **Коментари** – Редица от символи заградени със скобите {}. Те се игнорират при изпълнение на програмата и тяхната цел е да направят програмата по ясна. Препоръчително е да се използват между операторите, а не вътре в тях.

**Интегрирана работна среда на система за програмиране**

**Структура на програма в езика  PASCAL**

Всяка програма, написана на програмния език Psacal, има точно определена структура, разбираема за компилаторите, които преобразуват първичния код (текста на програмата) в двоични команди.

Програмата има три основни раздела:

-заглавна част,

- част за декларации и описания

- тяло на програмата.

Пример:

**program** First ;  
**Var** a, b : **integer** ;  
X, y   : **real** ;  
**begin**  
a:=5 ; a:=9 ; y:=2.5 ;  
X:=y\*a/b ;  
writeln (‘tova e moiata parva programa’, X:2:2) ;  
**end**.

Тази програма може да бъде въведена в компютъра, след което той може да я изпълнява толкова пъти, колкото е необ­ходимо.

**Заглавна част** -  Първият ред на програмата е нейното заглавие.  
Заглавието винаги започва със запазената думата **program**. След тази дума се задава името на програмата, което в разглеждания случай е First. Програмистът е този, който избира името на програмата,  и то трябва да бъде идентификатор. Обикновено се подбира наименование, което да е свързано с предназначението на програмата – тук е използвано наименование First, за да се отбележи, че това е първа програма, която създава програмистът. Символът **;** задължител­но се поставя в края на заглавието.

**Декларативна част** - Втората логическа част на програмата е разделът за обявяване (деклариране) на величини, които се използват в програмата или някои други поясняващи декларации и коментари. Тя се състои от 6 раздела, всеки от които може да е празен.

**Деклариране на променливите** - Променливите величини се наричат накратко променливи. Обикно­вено, вместо да сс каже „променливата с име X", се казва на­кратко „променливата X". Имената на променливите се избират по усмотрение на програмиста.

Описанията на променливите започват с **var**, което е съкра­щение на думата variable — „променлива". Всяка една променлива трябва да бъде ясно описана по тип. Декларирането на променлива води до отделяне на памет за тях. Общия вид е следният:

Var<списък с променливи>:<тип>;

В горната програма са декларирани две променливи ***a*** и ***b*** като **целочислени величини** и ***x*** и ***y*** – като **реални величини**. Те са използвани по-нататък за изчисляване на стойността на аритметичен израз.   
**Деклариране на константи –** често в програмите се използва една и съща стойност няколко пъти. В такъв случай се препоръчва константата да бъде декларирана предварително в раздела на константите.  Общия вид е следният:

CONST<име>=<стойност>;

**Деклариране на етикети -** Етикетът се поставя пред оператора, към който ще се предава управлението, като за разделител се използва двуеточие.     <етикет> : <оператор>  
Всички етикети, които се използват в програмата се декларират в раздела на етикетите:

LABEL  < списък с имена на етикети >;

**Деклариране** **на типове -** TYPE <идентификатор>=<име на тип>;

**Деклариране** **на процедури и функции** - PROCEDURE, FUNCTION.

**Изпълнителна част –** Това е третата част и се нарича тяло на програмата.В нея се съдържат основните изпълними оператори.

Тя започва със служебната дума ***begin*** и завършва с ***end*** и точка ‘.’.

Между тези две служебни думи се разполагат изпълнимите оператори, включително и обръщения към подпрограми и функции, описани извън тялото на програмата.

В нашият пример се задават стойности на променливите ***a***, ***b***и ***y***, а стойността на ***x*** се определя от аритметичен израз.

Накрая се извежда текст, в който се съобщава, че това е първа програма на програмиста ‘tova e moiata parva programa’ и след този текст се извежда стойността на променливата ***x***.

**Интегрирана работна среда на система за програмиране**

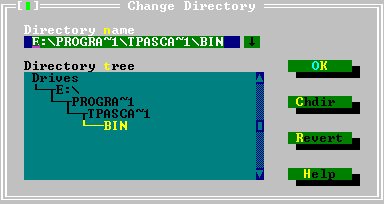
**Среда за програмиране Turbo Pascal 7.0**

**Структура на екрана. Основни правила за работа с диалогови прозорци.**

**Стартиране на** **Turbo Pascal 7.0 -** Обикновенно средата за програмиране се намира в папка  \TP .  
След стратиране се появява прозореца на Turbo Pascal, изобразен на рисунка 1 той има заглавна лента, на която пише **Turbo Pascal 7.0** и лента с менюта.

  
  
**Рисунка 1 -  Turbo Pascal**

      Активиране на лентата с менюта се извършва с клавиши **F10.** Маркерът се премества по пунктове с помоща на клавишите за управление на курсора ( lesson8_clip_image003). За отмяна избор на пункт е достатъчно да се натисне клавиш **Esc.** Този клавиш също се използва за затваряне на диалогов прозорец и изход от подменюто. За инициализация на един от пунктовете е необходимо след активизация на  менюто, да се маркира и се натисне клавиш **Enter**. При това се появява съответното подменю с набор от команди.   
      Централно място заема прозорецът за редактиране. В горния край на прозореца се указва  името на редактирания файл. Ако това е новосъздаден файл, то неговото име е NONAMEХХ.PAS, където ХХ - това е поредния номер на новия файл (номерацията започва от 00). В горния десен ъгъл на прозореца е разположен номерът на активния прозорец и бутон за изменение режима на изобразяване на прозореца с помоща на "мишка" (прозорец или пълен екран). На всеки новоотворен прозорец lesson8_clip_image004, се присвоява следващ пореден номер. Допуска се едновременно да са отворени 99 прозореца.  
      Често използваните команди:  
**F1 Help** - извиква помощ   
**F2 Save** - позволява да се съхрани текстът на програмата от активния прозорец за редактиране на диска.   
**F3 Open** - служи за отваряне на файла с текст на програма за редактиране.  
**Alt+F9** **Compile** - компилира текста на изходната програма от активния прозорец за редактиране.   
**F9 Make**- позволява да се прекомпилира текста на изходната програма от активния прозорец за редактиране и да се използват нейните модули.   
**Alt+F10 Local menu** - извиква (отваря) локално меню.  
Менюто  File съдържа команди за работа с файлове.  
Командите, завършващи с многоточие, извикват диалогови прозорци за изпълнение на своите функции. В дясно на някои команди са изобразени с клавишните комбинации.   
**New** - команда, служеща за създаване на нов файл с текст на програма. За това се създава нов прозорец за редактиране, на който се присвоява следващия ред номер.  
**Open… F3** - служи за извикване на диалогов прозорец, отваряне на файл с текст на програмата   
**Save F2** - съхранява на външно запомнящо устройство внесените във файла изменения в процеса на редактиране. Ако файла няма име (NONEMEXX.PAS), то се извиква диалогов прозорец, позволяващ въвеждането на име и указващ път за съхранение на файла.  
**Save as…** - извиква диалогов прозорец, приличащ прозорец Open, позволяващ въвеждане на ново име и указващ път за съхранение на файла.  
**Save all** - съхранява на диска измененията във файловете, намиращи се в отворените прозорци за редактиране.  
**Change dir**… - позволява да се измени работния каталог. Диалоговия прозорец е представен на рисунка 2.

  
  
**Рисунка 2 - Диалогов прозорец за изменение на работния каталог**

Изборът на каталог/папка/ се осъществява с натискане на **Chdir**. При необходимост да се върне изходното състояние се натиска клавиш **Revert.** След избор на необходимия каталог се натиска клавиш **Ok.**  
**Print** - служи за извеждане текста на програмата от активния  прозорец за редактиране на печат.  
**Printer setup…** - позволява да се настрои  печатащото  устройство.  
**DOS shell** - временен изход в DOS, връщането в IDE се осъществява  с команда Exit.  
**Exit Alt+X** - завършване работата с IDE. Ако в отворените прозорци има не съхранен текст, то се появява информационен прозорец, запитващ подтвърждение за съхранение измененията във файла.   
Меню **Edit,** е предствено на рисунка 3 и съдържа команди за управление на редактирането.

  
  
**Рисунка 3 - Меню Edit**

**Undo Alt+Backspace** - команда за отмяна на действие от предищната операция при редактиране  
**Redo** - отменя действието на команда  Undo.   
**Cut Shift+Del** - служи за изрезване на блок, който ще се отдели от текста и се помества в буфера за обмен (Clipboard) - временна област памет.   
**Copy Ctrl+Ins** - служи  за копиране и отделяне на текст в буфера за обмен.   
**Paste Shift+Ins** - предназначена за вмъкване съдържанието на буфера за обмен в активния прозорец, започвайки с текущата позиция на курсора.   
**Clear Ctrl+Del** - изчиства отделения блок от текста на програмата.        
**Show clipboard** - отваря прозорец със съдържанието на буфера за обмен.   
Командите от менюто за запуск запускат програмата от активния прозорец за  редактиране.  
**Run Ctrl+F9** - запуска програмата, исползвайки параметрите, предадени в нея с с помоща на команди Run/Parameters. Ако от времето на последната компилация изходния код е бил модифициран, програмата автоматически прекомпилира кода, а след това изпълнява програмата.  
**Step over F8** - изпълнява програмата по стъпки, без влизане в тялото на процедурите и функцийте.   
**Trace into F7** - изпълнява програмта по стъпки, със влизане в тялото на процедурите и функцийте.   
**Goto Cursor F4** - изпълнява програмата до реда , в който стои курсора.   
**Program reset Ctrl+F2** - прекратява текущия сеанс на работа с програмата, освобождава памет, и затваря всички открити файлове на използваната програма.   
**Compile Alt+F9** - компилира програмата от активния прозорец на редактора. Ако е открита грешка в статусния ред се дава съобщение.   
**Make F9** - използва се за компилация на програма, състояща се от няколко модула. При това се  прекомпилират всички  модули, в които са били внесени изменения.  
**Build** - използва се за компилация на програма, състояща се от няколко модула. При това се прекомпилират всички изходни текстове на модулите. **Destination** - позволява  да се определи дали изпълнения код да се съхрани на диска (като файл \*.ЕХЕ) или той ще се съхрани в паметта.  
**Primary file…** - позволява да се зададе файл \*.PAS, който ще се компилира, и да се счита за главен при използване на команда Compile/Make F9 или Compile/Build.   
**Information -** показва информационен прозорец за състоянието на програмата  и паметта.

**Стандартни данни. Операции и изрази**

**Деклариране на типове, константи и променливи**  
**Скаларни типове данни. Изброени и ограничени типове. Съвместимост на типовете**

**Типове данни в езика Паскал** – С помоща на типовете данни, програмиста указва на компилатора как да се съхранява информацията в паметта. Работата на програмиста е свързана с обработката на данни. Данните биват 2 вида: **константи и променливи**.

**Константите** имат точно определена стойност, която не се променя по време на изпълнение на програмата, а само се използва в процеса на изчисление.

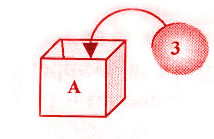
За разлика от тях **променливите** могат **да променят своите стойности в хода на работа** на програмата. Всяко ново значение в клетката от паметта на компютъра изтрива предишната. Затова в даден момент в клетката имаме само 1 значение.

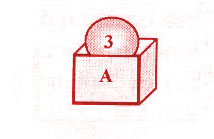
**Основни елементи на алгоритмите**- Преди да разгледаме основните видове алгоритми нека се запознаем с някои от основните понятия при работа с алгоритми.

**а)Величини -** характеризират се с име и тип.Типът определя множеството от допустими стойности на величината. Стойността на величина в даден момент се нарича текуща стойност.

Величините се делят на **константи и променливи**:

- Променлива – Математическата представа за променлива е различна от понятието променлива при алгоритмите. В компютърните алгоритми, променливата означава елемент от паметта, чието съдържание може да се променя. На фиг. 1 имената на променливите са представени като кутии, те са отделени от своите стойности които са изобразани като топки. Резултата е показан на фиг.2 - променливата с име А добива стойност 3.





   - Константи – тяхната стойност не се променя.

**b)Изрази** – за всеки тип величини са дефинирани различни операции, които се използват в изрази за изчисление.   
В изразите величините участват със своите имена, а операциите се извършват върху стойностите им.

*Пример: Вярно ли е записан израза А=А+1 ? От математическа гледна точка – НЕ. Но за програмиста това е абсолютно правилна инструкция, изчисляваща ни сумата от съдържанието на клетка А и числовата константа 1 и получения резултат се присвоява отново на клетка А. След изпълнението старото значение се изтрива.*

**Категории типове данни** - В езика Паскал съществуват следните типове данни:

- скаларни – цял, реален, символен, булев, изборен, подобласт. Стойностите на тези типове са неделими. Те биват стандартни и дефинирани.

-    съставни – масив, запис, множество, файл. Тези данни имат повече от 1 компонента.

-    тип указател – показват мястото в паметта, където се намират другите данни.  
  
**Целочислен тип**  
За означаването и дефинирането на този тип се използва стандартната дума **integer.** Множеството от стойностите на променливите от този тип са всички цели числа от [-32768;32767]. Декларирането става по стандартния начин:  
var x, y: integer;  
Допустимите операции с тези данни са:  
+ - събиране       \* - умножение         div – частното от деление  
- - изваждане                                      mod – остатък от деление



- следните функции, приложени над ця аргумент връщат цял резултат:  
abs(x) – връща абсолютната стойност на x : |x|  
sqr(x) – дава квадрат на x : x\*x  
succ(x) – дава наследника на x :x+1  
pred(x) – дава предшественика на x : x -1  
- функции, приложени над реален аргумент и даващи цял резултат:  
trunc(x) – отрязва дробната част и запазва цялата : trunc 7,9=7  
round(x) :закръгля x.  
***Пример:*** *abs(-2)=2, abs(126)=126, abs(-45)=45,  sqr(-5)=25, sqr(2)=4, succ(56) =57,   
succ(-2) =-1, pred(0)=-1, pred(56)=55****,*** *trunc(3.6)=3, round (3.6)=4*

**Реален тип**   
За декларирането на този тип се използва стандартната дума **real**.

Множеството от стойностите на този език се състои от всички реални числа от -1,7Е38 до 1,7Е38. Елементите на множеството от тип real се наричат реални константи.

Декларирането на променливите става по познатия начин:  
var   
x,y: real;

За стойностите от този тип са възможни следните операции и вградени функции:

+ - събиране                             \* - умножение  
- - изваждане                           / - деление

При реален аргумент следните функции връщат реален резултат

abs(x) – намира абсолютната стойност на x : |x|  
sqr(x) – намира квадрата на х

При реален или цял аргумент следните функции връщат реален резултат:  
sin(x) – намира синуса на даден ъгъл  
cos(x) – намира косинуса на даден ъгъл  
sqrt(x) – намира корен квадратен от x (x>0)

**Булев тип данни**

За означаването и дефинирането на този тип се използва стандартната дума **boolean**. Множеството от стойностите на този език се състои от два елемента – стойтостите **true** (истина) и **false** (лъжа). Тези стойности се наричат булеви константи.

Декларирането на булеви променливи става по познатия начин:

var b,x: boolean;

Над данни от булев тип е възможно да се прилагат някои операции и вградени функции. Опeраторите **not, and и or** (съответно логическо отрицание, логическо умножение и логическо събиране) се прилагат над операнди от булев тип. Резултатът е булева константа или променлива.

На долната таблица са дадени стойности на променливите a и b, както и операции с тях:

a           b           not a              a and b                a or b  
false       false       true               false                  false  
false       true        true               false                  true    
true        false       false              false                  true  
true        true        false              true                   true

Операторите за сравнение също могат да се прилагат над данни от булев тип:

=   равно                                 <=   по-малко или равно  
<> различно                           >     по-голямо  
<   по-малко                            >=   по-голямо или равно

Сравняването се извършва, като се сравнят кодовете на булевите константи.

*Например true<=false получава стойност false, false=true също получава стойност false, true>=false получава true и така нататък.*

Стандартни функции, даващи булев резултат:

Odd(x) – установява дали цяло цисло е четно или не,  
Odd(x) = true ако x е нечетно и обратно.  
Eoln(x) – булева функция, установяваща дали е достигнат края на реда на даден текстов файл. Има стойност true за достигнат край на реда и false за недостигнат.  
Eof(x) – булева функция, установяваща дали е достигнат края на даден текстов файл.

**Булеви изрази**

Булевите изрази са правила за получаване на булева константа. Булевите константи true и false, както и булевите променливи са булеви изрази. Прилагането на булевите оператори not, and и or над булеви изрази също е булев израз.

***Символен тип***

*За означаването на типа се използва запазената дума char. Множеството от стойности на типа се състои от елементите на крайно и наредено множество от символи. Графичните символи имат графично (видимо) представяне – ‘@’ ‘A’ ‘a’ ‘?’ ‘1’ и др. Означават се, като символа се загражда в апострофи.  
Наредбата на символите е следната:  
-    от 0 до 31 – управляващи символи  
-    от 32 до 255 – графични символи (40-57 – графични символи, 65-90 – букви)  
Тази наредба представя знаците, цифрите, главните и малките букви с последователни кодове.  
Променливите от вида се декларират по познатия начин:  
  
var ch: char;*

*Операции и вградени функции  
ord(ch) – намира кода на променливата ch  
chr(i) – намира символа, чиито код е i.  
pred(ch) – намира символа, чиито код е с 1 по-малък от кода на променли ch  
pred(ch)=chr(ord(ch) – 1)  
succ(ch) – намира символа, чиито код е с 1 по-голям от кода на променливата ch  
succ(ch)=chr(ord(ch) +1)  
Оператори за сравнение  - операторите за сравнение са същите, които се използват и в булевите изрази - <, >, =, <>, <=, >=. Резултатът от сравнението е от булев тип.*

**Деклариране на типове, константи и променливи**  
**Скаларни типове данни. Изброени и ограничени типове. Съвместимост на типовете**

**Приоритет на операциите при тип REAL и INTEGER**   
При намирането на стойност на израз операциите и функции­те имат определени приоритети. Операциите с по-голям приори­тет се прилагат преди операциите с по-малък приоритет.  
В езика Паскал с най-голям приоритет са всички функции. След тях с един и същи приоритет са операциите за умножение и деление, в това число и операцията mod. С най-малък приори­тет са операциите за събиране и изваждане.  
Няколко последователни операции с един и същи приоритет се прилагат отляво надясно.

**Използуване на скоби**  
Един израз може да бъде заграден в малки скоби. По този начин се получава израз, който има същите тип и стойност, както и изходният израз.  
Редът за прилагане на операциите се определя от техните приоритети. Този ред обаче може да бъде изменен, като се из­ползуват скоби.

**Приоритет на операциите при тип BOOLEAN -**Операцията  NOT има по –голям  приоритет от операцията AND, коята има по-голям приоритет от операцията OR.  
Показаното отдолу е приоритета на всички операции. Тези с един и същи приоритет са на 1 ред. Функциите имат по-голям приоритет от операциите.

**NOT**  
**\*, /, DIV, MOD, AND**  
**+, -, OR**  
**=, <, >, <>, <=, >=,**

**Оператори**

**Понятие за оператор. Видове оператори**

**Основни команди за описване на алгоритми:**

* 1. За присвояване на стойност;
  2. За въвеждане и извеждане;
  3. За избор на вариант (Условни команди);
  4. За организиране на цикъл;
  5. За обръщение към подалгоритъм.

Операторите на Паскал описват някакви алгоритмични действия.   
**Тялото на програмата** представлява последователност от такива оператори, които се разделят с точка и запетая.   
Ще разгледаме следните оператори на езикаПаскал: за присвояване, празен, съставен, за вход и за изход.

***Оператор за присвояване:***  
Това е един от най-важните оператори в езика Паскал. Синтаксисът му е  
<Променлива> := <израз>  
Величината, която е вляво на знака **:=** задължително е променлива. Тя може да е с поизволен тип с изключение на типа файл. От дясно на оператора задължително стои израз, който е от същия тип, от който е променливата в лявата страна.   
*Пример* ***а:=2****, където а е от тип integer   
Изключение: на променлива от тип реален може да се присвои стойтост от тип цял (тоест реалният тип припокрива типа целочислен).*

***Съставен оператор***  
Съставният оператор е най-често използваният оператор след оператора за присвояване.. Той представлява съвкупност от последователно изпълнявани оператори,заградени в операторните скобки begin.end:  
Begin  
a:=a+3;  
у:=х/5;  
с:=false   
End;  
*По принцип пред оператора end няма точка и запетая, но Турбо Паскал не отчита грешка ако се сложат. Важно е да се разбере, че всичко между begin и end се смята от машината за един оператор.*  
Тялото на програмата заключено между begin и end може да се разглежда като **един единствен оператор**. Единствената разлика е, че след end в този случай се поставя точка, за да разбере машината, че това е краят на програмата.

***Празен оператор***  
Това е оператор, който не съдържа никакви символи и не извършваникакви действия. Отделя се от останалите оператори с **;**.  
Празниятоператор дава възможност след който и да е оператор да се поставятпроизволен брои символи **;**.   
*Пример:  
Съставният оператор   
begin  
х := 1;;;;у := 14;  
End   
съдържа шест оператора: оператор за присвояване х :=* ***1,*** *следван от 3 празни оператора, оператор за присвояване* ***у :=******14,*** *следван от празен оператор.   
Приложението на този оператор е за забавяне на изчислителния процес.*

***Оператор за извеждане на данни***  
Чрез оператора за извеждане на данни на компютърния екран могат да се отпечатват стойностите на избраните величини, както и различни съобщения. В езика Паскал извеждането става с думите **write, writeln**.  
**Write** - означава „пиши", а в случая изведи.  
Къде? - Върху екрана.  
Какво? - Стойностите на изразите, посочени като парамет­ри на оператора write.

**Write ('текст')**  - извежда върху екрана в позицията на курсора текста, ограден с апострофи.

*Пример:   
Операторът write('a= ') извежда върху екрана в пози­цията на курсора текста а=.*

**write(a)**  
където **а** е- променливаизвежда върху екрана в текущата позиция на курсора стойността на променливата а.  
*Пример: Нека* ***а*** *е реална променлива със стойност 2.8. Опера­торът write(a) извежда върху екрана в позицията на курсора 2.8 -стойността на променливата* ***а****.*

Операторът **write** **('a= ', а)** ,където **а** е- променливае еквивалентен на операторите **write('a= '); writе(a).**  
Операторът **writeln** (без параметри) означава, да се премине извеждането на нов ред.  
**writeIn**(списък от параметри**)** е еквивалентен на операторите:  
**write**(списък от параметри);  
**writeln**

***Оператор за въвеждане на данни* –** Този оператор позволява някои от променливите на изпълняваната програма да получат стойности от външно устройство.   
**Read** - означава „чети", а в случая въведи.   
От къде? - От клавиатурата   
Какво? — Стойностите на променливите, указани като параметри на оператора **read**.  
Операторът read(a), където а е променлива величина, се изпълнява по следния начин. Настъпва пауза. **Компютърът чака да се въведе стойност на променливата а от клавиатурата**. Въвежда се стойност за променливата а, която трябва да е от типа, от който е а. След натискане на клавиша за вход (Enter), въведената стойност става стойност на променливата величина а.

Общият вид на оператора е следният:  
**Read (списък от параметри)** където  параметрите са променливи величини.

**Read (а1,а2,...ак**), където (а1,а2,...ак) са променливи, е еквивалентен на операторите  
**Read (а1); Read (а2);....; Read (ак);**

Операторът  **Readln (без параметри)** означaвa да се премине за четене на нов ред.

***Размер на полето* -** Всеки от извежданите резултати заема определено поле в изходния текст. Размерът на полето е равен на броя на симво­лите, които се записват в него.

За всеки извеждан резултат програмистът може да укаже съответен размер на полето. Така с оператора *write (R:E1)* се извежда *R,* като се използува поле с дължина *El.* Необходимо е *Е1* да е цял израз с положителна стойност.  
Освен размер на полето при извеждане на реално число мо­же да се укаже и размер на дробната му част.  
С оператора *Write {R:E1 :Е2)* се извежда стойността на реал­ния израз *R,* като се използува поле с дължина *Е1.*

Стойността на *R* се представя с десетична точка. Броят на цифрите след десетичната точка е *Е2.* Подобно на *E1* и *Е2* трябва да е цял израз с положителна стойност.

Пример:Нека **/** има стойност *25.* С оператора   
*write* (**/** *—10.1);*  
се извежда следният резултат:  
*25—250*  
С оператора  
*write(1 : 3—10.1 : 8);*резултатът се оформя така:  
*25                    -250*  
В оператора  
*write (1:1. —10.1:2);*  
указаните размери иа полето са недостатъчни. При изпълнението на този оператор   полетата се разширяват с необходимия брой позиции и резултатът изглежда така:  
*25—250*

Размерът на полето може да бъде указан както при извеж­дане на числа, така и при извеждане на символи, низове и буле­ви стойности.

**Оператор за безусловен преход, етикети, деклариране**

***Оператор за безусловен преход.*** *Този оператор служи да наруши последователното изпълнение на командите в програмата. Обикновено командите в програмата се изпълняват една след друга. Ако по някаква причина искаме да прескочим няколко команди или да се върнем назад с няколко команди, тогава може да се използва операторът за безусловен преход. Той е нехарактерен за стила на езика Pascal и е остатък от по-старите езици за програмиране. Доказано е, че всяка програма на Pascal може да се напише и без този оператор, като се използват други средства. Граматиката на оператора за безусловен преход е следната:****goto*** *етикет   
където етикетът е цяло положително число. То се поставя пред командата от програмата, към която искаме да се извърши прехода:   
етикет: команда   
Всички етикети в програмата трябва да са различни и да са описани в декларативната и част. Описанието е следното:****label*** *етикет1, етикет2, ... , етикетN;   
Декларирането на етикетите става преди декларирането на константите и променливите (****const*** *и* ***var****) и след декларацията за ползвани библиотеки от готови подпрограми. Операторът* ***goto*** *е опасен за ползване, защото нарушава естествената логика на програмата. Чрез него тя се накъсва и става трудна за асимилиране. Това често води до грешки. Затова използването на* ***goto*** *би трябвало да се избягва. Съществуват обаче случаи, където опита да се избегне* ***goto*** *също усложнява и забавя програмата.*

**Условни оператори. Оператор за избор**

При съзздаването на програми се налага да се опишат действия, чието изпълнение зависи от някакви условия. Такива, действия в програмите се записват посредством условни оператори.  
Оператор, който дава възможност да се изпълни или да не се изпълни друг оператор в зависимост от дадено условие, се нарича **условен оператор**. Условните оператори реализират разклоняващи се изчислителни процеси.  
В езика Паска условния оператор има две форми: **кратка** и **пълна:**

***Кратка форма на условния оператор* -  if** **В then** **S,** където

if (ако) и then (тогава) са запазени думи,

В е булев израз

S е оператор

Условният оператор **if В then S** се изпълнява по следния начин:  
Пресмята се стойността на булевия израз В. В резултат се получава булева константа true или false.

* 1. Ако стойността на В е равна на true, изпълнява се операторът S
  2. Ако стойността на В е равна на false, S не се изпълнява. И в двата случая, след това се преминава за изпълнение на оператора след if-then.

Булевият израз В трябва да е напълно определен.   
След then стои точно един оператор. Ако няколко оператора трябва да се изпълнят след then, необходимо е да се обединят в един съставен оператор.

***Пълна форма на условнин оператор* - if В then SI else S2,** където  
if (ако), then (тогава) и else (иначе) са запазени думи,  
В е булев израз,  
S1 с оператор,  
S2 е оператор

Операторът **if** **В then SI else S2** се изпълнява по следния начин:

* 1. Пресмята се стойността на булевия израз В. В резултат сс получаваtrue или false.
  2. Ако стойността на В е true, изпълнява сс операторът S1 (S2 не се изпълнява).
  3. Ако стойността на В е false, изпълнява се операторът S2 (S1 не се изпълнява).
  4. И в двата случая след това се преминава за изпълнение на следващия оператор.

 Булевият израз В трябва да е напълно определен.  
След  then, а сьщо и след else трябва да стои точно един оператор. Ако след then или след else трябва да се изпълни повече от едно действия, оформя се като съставен оператор.

**Оператор за избор** – С помощта на логическия оператор **if** и съставен оператор можем според някакво условие да разклоним алгоритъма на един или два клона.

Но когато имаме много алтернативни възможности се получава една сложна верига от съставни условни оператори, което прави програмата трудна за анализ и проследяване. Вместо това в  Паскал  се използва оператор за избор.

Той има следнатият вид:   
**case** SEL **of**  
        константа1: команда1;  
        константа2: команда2;  
        .   .  
        константаN: командаN;  
      **else** командаN+1  
 **end**

**case –** ключова дума, която идентифицира  оператора за многоварианте избор.  
SEL – променлива използвана като селектор.   
Действията между които се избира са взаимоизключващи се. Изпълнява се не повече от 1 действие.

**Оператори за цикъл**

Често и в програмирането се налага части от дадена програма да се изпълняват многократно докато дадено условие е изпълнено.

Тогава се използват операторите за цикъл:  
-**while S do A;**  
- **repeat A until S**  
-**for param:=start to finish do A**

Тук S е някакъв логически оператор, чиято истинност се проверява, а А е оператор (много често А е съставен оператор).

**Цикъл с предусловие** - Ако повторенията продължават, докато е в сила дадено условие, и завършват щом то вече не е в сила се използува операторът **While S do A**   
/трябва да се разбира така: докато S е истина изпълнявай А/:

While a>b do  
begin  
x:=sin(a);  
a:=a+pi/10;  
end  
Този цикъл се изпълнява докато а стане по-голямо от b.  В частност може да не бъде изпълнен нито веднъж.

**Цикъл със следусловие** - Ако повторенията продължават само докато зададеното условие още не е изпълнено, и завършат когато то бъде изпълнено, изпълнява се оператора  
**repeat A until S**   
/трябва да се разбира така: повтаряй А докато S станеистина (или докато S е неистина)/:

repeat  
x:=sin(a);  
a:=a+pi/10  
until a>b;  
Този израз ще се изпълнява докато а<=b. Щом а стане по-голямо от b програмата продължава със следващите оператори.

Обърнете внимание, че условията за изпълнение на операторите **while-do** и **repeat-until** са различни. Това придава по-голяма гъвкавост при съставянето на програми.

Друга разлика е, че условието за изпълнение на оператора while-do се проверява в началото, а на repeat-until  в края на операторите. Поради това, repeat-until се изпълнява поне веднъж независимо от условията.

**Цикъл, управляван от параметър -** Ако броят на повторенията е предварително известен, се използва оператора **for.**

|  |  |
| --- | --- |
| *възходяща форма:* | **for i := A to B do S;** |
| *низходяща форма:* | **for i := A downto B do S;** |

* + **for**, **to**, **do** и **downto** – запазени думи;
  + **A** и **B** – начална и крайна стойност на параметъра **i**;
  + **S** – тяло на цикъла (*действията, които ще се повтарят*).

**Възходяща форма:**

* + **A < = B** – **i** приема стойността на **А**, изпълнява се тялото **S**, след което **i** автоматично се увеличава с 1. Повторението продължава, докато **i = B**.
  + **A > B** – **S** не се изпълнява.

**Низходяща форма:**

* + **A > = B** – **i** приема стойността на **А**, изпълнява се тялото **S**, след което **i** автоматично се намаля с 1. Повторението продължава, докато **i = B**.
  + **A < B** – **S** не се изпълнява.

Ако операторите в тялото са повече от един, те задължително се поставят в съставен оператор – **begin** … **end;**  
Стойността на брояча ***не трябва*** да се променя.

***Вложени цикли -*** *когато един цикличен оператор е включен в тялото на друг, той се нарича вложен. Възможно е влагането на повече от един цикличен оператор.  
Зацикляне – Съществуват случаи, когато изпълнението на оператор за цикъл продължава неограничено дълго.Така например изпълнението на оператора   никога не завършва. В такъв случай се казва че се е получило зацикляне на програмата. Същестуват оператори, чието изпълнение завършва при едни стойности на променливите, но води до зацикляне при други.*

**Масиви**

**Масивът е един от най-използваните в програмирането типове данни. Почти винаги, когато в програмата се появя­ва необходимост от работа с редица от еднотипни стой­ности се въвежда и съответен масив.**

**Неговите основни характеристики са:**

* + брой на елементите;
  + елементите са от един и същи тип;
  + елементите са подредени по някакъв признак;

**Масивъте структурен тип данни, чиито стойности се представят с крайна редица от еднотипни стойности. Най общо казано  масивът  е  група елементи от един и същи тип, обединени под общо име.  
На всеки елемент на масива може да се съпостави него­вият пореден номер в установената наредба между елемен­тите, наречен индексна елемента. Индексът определя мяс­тото на елемента в масива. Той позволява да се отличават отделните елементи на масива един от друг и да се обръща към тях.**

***Пример: Обикновенно книгата може да се счита за масив. Защо?  
Отговор: книгата се състои от множество еднотипни елементи – страници, всяка страница има номер (индекс), всички страници са обединени под едно име – името на книгата.******Домът също може да се счита за масив. Защо?Отговор: Домът съответства на даден адрес (град, улица, номер). За елементи на дома може да се считат апартаментите, всеки си  има № (индекс).Шахматната дъска  също може да се счита за масив. Защо?***

**  **

***Футболният отбор също може да се счита за масив. Защо?Отговор: отборът се състои от няколко играчи, всеки от тях има № (индекс).***

***Нека е дадената следната задача:Да се въведе средния успех на всеки ученик от 10а клас в масив (*редица от стойности*), след което да се изведе общия среден успех на класа.  
Учениците и средния успех на всеки един от тях може да се представи чрез таблица*:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| номер в клас | 1 | 2 | 3 | ... | 25 | 26 |
| среден успех | 3.56 | 5.60 | 4.50 | ... | 6.00 | 5.23 |

|  |  |
| --- | --- |
| 26 елемента, зададени чрез пореден номер | |
| 10а клас (1) среден успех на ученик №1 | Достъпът до съответния елемент се осъществява чрез индекса. (номера на съответния ученик) |
| 10а клас (2) среден успех на ученик №2 |
| .......... |
| 10а клас (26) среден успех на ученик №26 |

**Размерност - Броят на признаците, по които са подредени елементите, се нарича размерност.**

* + Редица - едномерен масив
  + Таблица - двумерен масив  
    (достъп до елемент – по два признака: номер на ученик, предмет)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер в клас | Учебен предмет – срочни оценки | | | |
| Математика | Биология | Физика | ..... |
| 1 | 3.00 | 4.00 | 5.0 | ..... |
| 2 | 5.00 | 6.00 | 6.00 | ..... |
| ..... | | | | |
| 25 | 4.00 | 5.00 | 5.00 | ..... |
| 26 | 5.00 | 4.00 | 3.00 | ..... |

**2.Тип масив в ЕП Паскал.   
а)деклариране;  
VAR <име\_на\_масива> : array [A . . B] of <тип\_на\_елементите> ;**

* + array, of – запазени думи;
  + <име\_на\_масива> - задава се от програмиста;
  + <тип\_на\_елементите> - определя с какви елементи ще се работи;
  + A, B – начална и крайна граница на индекса;

Пример:  
**TYPE**   
**klas = array [1 . . 26] of real;**  
**VAR**   
**Init : array [-4 . . 15] of char;**  
**student : klas;**  
**nums : array [‘A’ . . ‘F’] of integer;**

б)множество от стойности;  
**Редица от стойности (за всеки индекс) от типа на елемените.**  
  
**Init – знаков масив с максимум 20 елемента:**  
**Init = ‘А’, ‘v’, ‘8’, ‘\*’, ‘@’ => 5 елемента**  
**Init = ‘&’, ‘(’, ‘J’, ‘-’ => 4 елемента**

в)операции;

* + върху променливи от тип масив не се прилагат операции;
  + върху елементите на масива се прилагат операциите, допустими за обявения тип;
  + достъп до елемент; <име\_на\_масива>[индекс]

Пример:

**Init[-4] => ‘A’**  
**Init[-2] => ‘8’**

г)въвеждане и извеждане на стойности;  
**N – броят на елементите**

* + въвеждане;

**for i := 1 to N do begin**  
**write ( ‘ Въведете елемент номер ’ , i ) ;**  
**readln ( masiv [ i ] ) ;**  
**end ;**

* + извеждане;

**for i := 1 to N do writeln ( masiv [ i ] ) ;**

**Могат да се използват трите вида циклични оператори, които бяха разгледани по-горе.**

Примерна програма:

**Да се напише програма за намиране сумата  на елементите на масив с N (N<100) елемента.**

**Решение:**

**program Masiv1;  
var a:array[1..99] of real;  
i, n : integer;  
s: real;**

**begin**

**repeat  
write(‘n=‘);    
readln(n)  
until (n>0) and (n<100);**

**s:=0;  
for i:=1 to n do  
begin  
write (‘a[i]=‘);  
readln(a[i]);   
s:=s+a[i];  
end;**

**Writeln (‘S=‘,s:5:3)  
end.**

**..........................................................................................................**