Тема 1. Запознаване със средата за програмиране.

1.1 Общи положения.

Преди да започнете да пишете програми, трябва да се запознаете със средата за програмиране, в която ще работите. Целта на часовете по информатика обаче не е да работите с най-новите и скъпо струващи среди за програмиране, а да разберете за какво всъщност са предназначени те – създаването на програми. Тук ще се наблегне на основите, ще се запознаете с основни принципи в програмирането и ще усвоите в начална степен един език за програмиране. Всъщност това е първата стъпка, през която преминава всеки един бъдещ програмист.

Средата за програмиране, с която ще работите е възможно най-проста и по-важното свободно достъпна. За езика Pascal ще използваме компилатора Borland Pascal 7.0, а за C++ - компилатора Borland C ++ 3.1. И двете среди за програмиране са създадени за работа под операционна система DOS.

Въпреки, че тук ще бъде описана само работата със среди за програмиране под DOS, това не ви пречи да работите с такива за Windows. Всички команди, с които ще се запознаете са налични и там, в много случаи дори кратките клавиши за извикването им са същите и бързо бихте се ориентирали в която и да е среда за програмиране за съответния език, след като вече сте се запознали с една такава.

Работната област и на двете среди е оцветена в синьо. Това е мястото където пишете кода на Вашите програми. Над нея се намира лентата с менюта. Отделните менюта могат да се отворят, като се натисне клавиша Alt и съответната буква, оцветена в различен цвят (това важи и за всички програми за Windows, но буквите не са в друг цвят, а са подчертани).

Програмата може да работи в прозорец или на цял екран. За да превключвате между двата режима, използвайте комбинацията Alt+Enter.

Препоръчително е да научите клавишните комбинации за различните действия. Разбира се, ако желаете можете да използвате и мишката, която в режим на цял екран се визуализира като кафяво-оранжеп правоъгълник. Действието й обаче е ограничено само до работа с левия бутон. Десния бутон, заедно с контекстните менюта са привилегия на Windows.

Изхода от програмата не става по познатия Ви за Windows начин. Alt+F4 не извършва затваряне на програмата. Излизането от програмата става от менюто File, като изберете командата Quit или Exit, или като натиснете Alt+X. Ако работите в режим на прозорец, можете да използвате бутона за затвзряне X в горния десен ъгъл.

Основни менюта и команди:

Меню File – както и в програмите, с които вече сте работили, тук можете да намерите основни команди свързани с файловете:

New (създаване на нов файл)
Open (отваряне на съществуващ файл)
Save (запис на промените във файла)
Save as… (запис под ново име или не друго място)
Print (отпечатване на докумнт)
Exit/Quit (изход от програмата)

Меню Edit – тук могат да се намерят команди, свързани с редактирането:
Cut (изрязване – подготовка за преместване на маркиран текст)
Copy (копиране – подготовка за копиране на маркиран текст)

Paste (извеждане на изрязан или копиран текст)
Clear (изтриване на маркиран текст)

Меню Search –команди, свързани с търсене и заместване в текста на документа

Меню Run – тук се намират команди за стартиране и проследяване на програмата

Меню Debug – тук са поместени команди, служещи за откриване и отстраняване на грешките в кода на програмата

Меню Options – тук се намират всички настройки на програмата, като например как точно ще се изпълняват програмите, в кои папки да се търсят библиотечните файлове или какви да са цветовете на работната област.

Опитайте се да промените цветовите настройки в работната област на програмата според собствените си предпочитания: команда Environment/Colors.

Меню Help – помощна информация за програмата и за езика за програмиране.

Работа с клавиатурата:

→ : премества курсора с една позиция надясно

← : премества курсора с една позиция наляво

↑ : премества курсора с един ред нагоре на същата позиция

↓ : премества курсора с един ред надолу на същата позиция

Shift + буква/цифра : главна буква или символ от горен регистър

Enter : преминаване на нов ред

Delete : изтриване на символа, върху който е маркера

Backspace : изтриване на символа, намиращ се преди маркера

Space : оставяне на празен интервал

Tab : табулация – разстояние, равняващо се на няколко празни интервала (7 за Pascal и C++). Клавищът се ползва и в друг смисъл – когато се намирате в меню или диалогов прозорец, натискането му прави активна следващата избираема част – бутон, команда в меню, дори хипервръзка в страница, т.е. тази функция на клавиша е валидна и за Windows)

Shift + Tab : връща към предходна избираема част – бутон, команда и др.

Insert : както в DOS, така и във Windows, този клавищ служи за превключване между два режима – режим на вмъкване (стандартен) и режим на припокриване. В случай, че е включен режим на припокриване, маркерът ще е удебелен, а пък ако сте във Word например, в долната част на програмата ще видите буквите OVR. При стандартния режим на вмъкване всички символи които пишете се появяват преди символа, върху който е маркера. При режим на припокриване новият символ замества символа, върху който е маркера.

PageUp : един екран нагоре

PageDown : един екран надолу

Home : връща маркера в началото на реда

End : премества маркера в края на реда

Esc : изход от диалогов прозорец или меню

Shift + → : маркиране на символи надясно, започвайки от символа, където е маркера

Shift + ← : маркиране на символи наляво, започвайки от символа преди маркера

Shift + ↑ : маркира един ред негоре – от началото на реда до символа, преди маркера за текущия ред и от символа над маркера до края на горния ред

Shift + ↓ : маркира един ред надолу

Ctrl + → : премества маркера с една дума надясно

Ctrl + ← : премества маркера с една дума наляво

Ctrl + Shift + → : маркира една дума надясно

Ctrl + Shift + ← : маркира една дума наляво

Ctrl + Shift + ↑ : маркира абзац нагоре в Word

Ctrl + Shift + ↓ : маркира абзац надолу в Word

Shift + End : маркира от текущата позиция до края на реда

Shift + Home : маркира от предходната позиция до началото на реда

Ctrl + End: премества маркера на най-долния ред на екрана (за Word в края на документа)

Ctrl + Home : премества маркера на най-горния ред на екрана (за Word в началото на документа)

Ctrl + Shift + End : маркира всички редове от текущия до края

Ctrl + Shift + Home : маркира всики редове от реда над текущия до началото

Shift + PageUp : маркира страница нагоре

Shift + PageDown : маркира страница надолу

Ctrl + A : маркира целия текст в Word

Ctrl + Y : изтрива текущия ред

Shift + Delete : изтрива маркиран текст

F1 : помощ / **Ctrl + F1** : помощ за конкретна дума

Alt + F1 : връща към предишния екран разглеждан при получаване на помощ

При опит за изход от програмата или затваряне на документ, както в DOS, така и в Windows, често пъти ще се появява диалогов прозорец, в който ще бъдете запитвани дали желаете да съхрните промените в документа. Отбележете, че за промяна се смята дори поставянето на един празен интервал – за компютъра той е равностоен с осналаните букви. Възможностите са следните:

Yes – изход / затваряне, като преди това се запазят промените. В случай че документът все още не е бил записван, ще бъдете запитани за име на файл и папка, където да бъде създаден.

No – изход / затваряне без да се записват промените

Cancel - отказваме се от затварянето на програмата / документа

Внимание: Препоръчително е, преди да тествате дадена програма, да съхраните промените в нея, в противен случай изпълнението й може да се забави. DOS-овските програми заемат по различен начин RAM паметта от програмите под Windows. Може да се окаже, че не можете да стартирате Интернет браузер, поради факта, че не сте затворили програма, работеща в DOS режим.

1.2 По-важни команди за работа със средата.

Тъй като и двете среди за програмиране са на компанията Борланд, очаквано командите за работа са едни и същи затова ще бъдат описани заедно. Това са командите, които се използват най-често.

F2 – запис на промените

Alt + F9 – компилира кода, т.е. превежда го на машинен език и извежда списък с грешките, ако има такива

F9 – компилира, като създава изпълним EXE файл и извежда списък с грешките, ако има такива

Ctrl + F9 – компилира и изпълнява програмата ако няма грешки, в противен случай извежда списъка с грешките

Alt + F5 – показва потребителския екран с резултата от изпълнението на кода

Ctrl + F1 – помощ за конкретна дума от езика

F3 – отваряне на съществуващ файл

Alt + F3 – затваряне на прозорец

Alt + Backspace – стъпка назад

Shift + Del или **Edit -> Cut** – изрязване на маркиран текст

Ctrl + Ins или **Edit -> Copy** – копиране на маркиран текст

Shift + Ins или **Edit -> Paste** – извеждане на изрязаян / копиран текст

Ctrl + Y – изтриване на ред

Ctrl + Del – изтриване на маркиран текст

Alt + X – изход от програмата

Алтернативи за копиране / местене – изпълняват се при натиснат клавиш Ctrl и последователно натискане на двете посочени букви:

Маркиране : **Ctrl + K + B** (в началото на блока) -> **Ctrl + K + К** (в края на блока)
Преместване: **Ctrl + K + V**
Копиране: **Ctrl + K + C**
Размаркиране: **Ctrl + K + К** (в случая означава край на действието)
Освен това по всяко време може да ползвате комбинацията:
Размаркиране: **Ctrl + K + H**

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Изпробвайте командите от менютата, описани по-горе – създайте нов файл, запишете го в работната си папка, затворете го и вижте с какво разширение е, пак го отворете и т.н.
2. Изпробвайте различните клавиши и клавишни комбинации за въвеждане и редактиране на текст и отбележете в тетрадките си тези от тях, които смятате че ще бъдат използвани най-често. Проверете действието им в Windows, например в Word.
3. Експериментирайте с цветовите настройки на работната област.
4. Отворете и разгледайте помощната част на средата за програмиране. Опитайте да получите помощ за думата circle. Обърнете внимание на това каква информация се получава при търсене за конкретна дума.

Тема 2. Структура на програма.

2.1 Какво има в една програма?

Програма, това е алгоритъм, описан с помощта на някой от езиците за програмиране. Всеки алгоритъм се състои от поредица действия, наречени стъпки. Аналогът на стъпка в програмата се нарича оператор. Следователно ***оператор*** ще наричаме всяко едно действие в рамките на програмата.

В повечето езици за програмиране в края на операторите се поставя знака „**;**”. Може да се каже, че за компютъра този знак означава край на текущото действие, така както за нас точката означава край на изречение. Изключение прави езикът Basic, където краят на действие се обявява с преминаването на нов ред.

Операторите биват прости и съставни. Простите оператори представляват едно действие, а съставните обединяват две или повече прости действия, които бихме искали да се възприемат и изпълняват от компютъра като едно. Част от операторите се наричат управляващи и служат за опереляне на реда, в който се изпълняват действията в програмата, в зависимост от реакцията на потребителя.

В рамките на всяка програма присъстват така наречените ***параметри***, които биват променливи или константни. В тях се съхраняват данните, необходими на компютъра за изпълнението на програмата. В повечето езици за програмиране те трябва да бъдат декларирани в началото на програмата. Така Вие казвате на компютъра по колко памет да задели за всеки параметър и какъв вид информация ще се съхранява в нея. В Basic това действие не е задължително и ако бъде пропуснато компютъра сам определя типа според свързаната с параметъра стойност. Имената на параметрите трябва да започват с буква.

Параметрите, свързани помежду си с математически, логически и операции за сравнение образуват ***изрази***. Те от своя страна не са самостоятелна единица, а представляват градивен елемент за операторите. От друга страна без изрази не би била възможна обработката на информация и програмата щеше да се ограничи просто до отпечатване на екрана на това, което сте въвели от клавиатурата.

В рамките на програмата освен операторите много често се използват ***подпрограми***. Към всеки език за програмиране е осигурен набор от ***стандартни*** подпрограми, като в повечето езици те са разпределени във файлове библиотеки според предназначението си, например математически или графични и др. Вие можете също да създадете подпрограми, които да се използват в рамките на програмата. Такива подпрограми се наричат ***потребителски***.

Наред със всичко изброено до момента в програмите могат да присъстват и така наречните ***коментари***. Това е текст в програмния код, който компютъра игнорира като несъществуващ, а целта му е да подпомогне работата на програмиста. Във всеки език за програмиране има специфичен начин за обозначаване на коментар. В рамките на един коментар програмиста може да добави всякаква информация, която прецени, че му е нужна, например за какво служи дадена променлива, какво прави конкретна част от програмата или цялата програма и др.

2.2 Структура на програма на Pascal.

Най-общо програмата на Pascal изглежда така:

[program *име*;]
[uses *списък_с_модули*;]
[label *списък_с_етикети*;]
[const *списък_с_константи*;]
[type *дефиниции_на_типове*;]
var *декларации_на_променливи*;
[procedure/function *дефиниции_на_подпрограми*;]
begin
оператори;
end.

Както се забелязва, по-голямата част от разделите са незадължителни и затова са оградени в правоъгълни скоби. Наличието им зависи от сложността на програмата според целите, които тя обслужва. Задължителните елементи са удебелени – всяка програма има начало, последователност от действия, които трябва да се изпълнят и край. И тъй като целта на програмите е да извършват някаква обработка на информация, са ни необходими променливи, поради което разделът на променливите, започващ със запазената дума var също е задължителен за всяка програма.

По Ваша преценка можете да дадете име на програмата си и да го запишете в началото на кода (например Program Zad_34;).

В някои случаи Ви се налага да използвате стандартни подпрограми, описани във външни библиотечни файлове. Тези файлове в Pascal се наричат модули (units). За да можете да използвате нужната ви подпрограма, трябва да съобщите на компютъра в кой модул се намира тя (например подпрограмата, която ни позволява да сменяме цвета на текста се намира в модула Crt, следователно за да можем да я ползваме трябва да запишем в началото на програмата uses Crt;).

Следващите пет раздела могат да бъдат записани в произволен ред. В раздела на етикетите се изброяват предварително всички имена, които смятате да използвате за означаване на конкретни редове от програмата. В раздела на типовете можете да създадете потребителски тип данни извън стандартните за езика. В раздела на константите се описват всички константи, които ще бъдат използвани в програмата, заедно с техните стойности. В раздела на подпрограмите пък можете да дефинирате потребителски процедури или функции.

Коментар се отбелязва по два начина: {*коментар*} или (**коментар**)

Коментара може да бъде разположен на един или повече редове, което за компютъра е без значение. Често пъти използването на коментари е свързано и с тестване на отделни части от програмата, като за целта останалите се поставят в коментар. Така коментираните редове не се налага да се пищат повторно, в случай, че се окажат необходими.

При писането на кода няма значение дали ще пишете с главни или малки букви. Всичко зависи от Вашите предпочитания. Добре е обаче да се спазва някакъв стил на записване (краснопис). Обикновено това се постига чрез отместване на редовете навърте, а резултата е по-четлив код.

Пример за програма, която събира две числа, въведени от клавиатурата:

```
var a,b,c : integer;
begin
  write('a= ');
  readln(a);
  write('b= ');
  readln(b);
  c:=a+b;
  writeln('sumata e ',c);
end.
```

За да тествате програмата, наберете кода й в средата за програмиране, натиснете Ctrl+F9, за да я изпълните, след което натиснете Alt+F5, за да видите резултата.

2.3 Структура на програма на C++.

Най-общо програмата на C++ изглежда така:

[*заглавен блок с коментари*]
заглавни файлове
[*дефиниции на константи*]
[*дефиниции на структури / класове*]
[*глобални променливи*]
[*дефиниции и декларации на функции*]
главна функция (main)
[*дефиниции на декларирани функции*]

Както се забелязва, по-голямата част от разделите са незадължителни и затова са оградени в правоъгълни скоби. Наличието им зависи от сложността на програмата според целите, които тя обслужва. Задължителните елементи са удебелени – всяка програма се състои от поне една функция, като главната функция (същинската част на програмата), носи името **main**.

В езика C++ всички подпрограми (функции), които бихме могли да използваме при писането на програмата са групирани според предназначението си и разпределени в подходящо озаглавени файлове, наричани **заглавни** файлове. За да можете да използвате нужната ви подпрограма, трябва да съобщите на компютъра в кой заглавен файл се намира тя (например подпрограмата, която ни позволява да изчистим екрана се намира във файла conio.h, следователно за да можем да я ползваме трябва да запишем в началото на програмата #include<conio.h>). Изброяването на всички необходими файлове става в началото (заглавната част) на програмата, от където идва името им – заглавни.

заглавен блок с коментари – при компилиране коментарите се игнорират, те съдържат допълнителна информация и са предназначени изцяло за програмиста. Освен в заглавния блок, коментари могат да се добавят и навсякъде другаде в програмата. Това става по два начина: //коментар или /*коментар*/. Първия вариант се използва, когато коментара е в рамките на един ред. За компютъра това означава да игнорира всичко до края на реда, записано след двете наклонени черти. Втория вариант се използва при по-дълги коментари, рзаположени на няколко реда. Често пъти използването на коментари е свързано и с тестване на отделни части от програмата, като за целта останалите се поставят в коментар. Така коментираните редове не се налага да се пищат повторно, в случай, че се окажат необходими.

дефиниции на константи – извеждането на константите в началото прави кода по-четлив, намалява грешките при многократно изписване на стойността на константата, а в случай, че се налага, лесно се променя стойността й.

дефиниции на структури / класове – тук се въвеждат нови типове данни, създадени от програмиста.

глобални променливи – това са променливи, които са дефинирани извън функциите и са „видими” за всички функции, дефинирани след тях.

функции – това е мястото, където се дефинират (описват) потребителските подпрограми, т.е. създадени от програмиста. Тези, които не могат предварително да бъдат дефинирани, само се декларират (обявява се, че ще бъдат описани по-късно), а дефинирането им става след края на главната функция.

Общия вид на функция, включително на главната функция main е:

тип_на_резултата име_на_функцията (списък на формалните параметри)
{
тяло (множество оператори)
}

Тялото на функцията се състои от оператори, затворени между две фигурни скоби. В случай, че се налага използването на вътрешни за функцията (локални) променливи, те се дефинират също в тялото й. Типа на параметрите, както и типа на връщания от функцията резултат може да е всеки допустим за езика тип. Функцията може и да няма параметри, а в случай, че не връща някакъв конкретен отговор, като число например – нейният тип се записва като празен (void). Поради това често за главната функция се предпочита такова записване:

void main () {
тяло
}

При писането на кода има значение дали ще пишете с главни или малки букви. Ако сте дали на една променлива име Sum, използването на друго място в програмата на името sum например би предизвикало грешка. Всички имена на параметри и функции трябва да се изписват винаги по един и същ начин. Отчетете, че всички стандартни функии се записват с малки букви, а всички стандартно дефинирани константи – с главни.

Добре е да се спазва и някакъв стил на записване (краснопис). Обикновено това се постига чрез отместване на редовете навърте, а резултата е по-четлив код.

Пример за програма, която събира две числа, въведени от клавиатурата:

```
#include<iostream.h>
void main () {
  int a,b,c;
  cout<<”a= “;
  cin>>a;
  cout<<”b= “;
  cin>>b;
  c=a+b;
  cout<<”sumata e “<<c;
}
```

За да тествате програмата, наберете кода й в средата за програмиране, натиснете Ctrl+F9, за да я изпълните, след което натиснете Alt+F5, за да видите резултата.

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Избройте елементите на програма.
2. Тествайте програмата от урока като замените:

   а) имената на използваните променливи с други;

   б) текста, който се отпечатва;

   в) действието събиране с друго математическо действие.

3. Добавете произволен текст на произволно място в програмата и опитайте да я изпълните. Обърнете внимание за какви грешки предупреждава програмата. Сега поставете текста в коментар и отново опитайте да изпълните програмата.
4. Поставете последователно всеки от редовете в коментар и обърнете внимание:

   а) при липсата на кои редове се съобщава за грешка и каква е тя;

   б) липсата на кои редове не предизвиква грешка;

   в) има ли редове, които могат да бъдат пропуснати, без да се промени крайния резултат от изпълнението на програмата.

Тема 3. Основни оператори.

3.1 Типове данни.

При писането на кода на програмите е необходимо за всеки параметър, и най-вече за променливите, да се укаже какъв ще бъде типа на данните, които ще се съхраняват в него. По този начин компютъра разбира по колко памет трябва да задели за съответния параметър и как трябва да бъде интерпретирана информацията в тази памет. Знаете, че без значение дали ние виждаме цяло или дробно число, знак или цветна точка на екрана, за компютъра тази информация се представя като поредица нули и единици. С определянето на типа данни ние указваме на компютъра като какъв точно вид информация трябва да бъдат разпознати тези нули и единици.

Най-общо типовете данни се делят на ***стандартни*** (предварително дефинирани, вградени в езика за програмиране) и ***потребителски*** (дефинирани от програмиста). В някои учебници думата стандартни се ползва за общо име за изброимите вградени типове, в които има наредба – целочислени, символни и логически. Отбележете си, че в рамките на часовете по Информатика ще наричаме стандартни всички предварително дефинирани за езика типове.

И така, стандартните типове биват ***скаларни*** и ***структурни***. Скаларните или още прости типове са съставени от една компонента – всеки параметър от скаларен тип приема единична стойност. Структурните типове са изградени от повече компоненти. Променлива от тип масив може да съхранява много елементи от един и същ тип – например имената или успеха на всички ученици в училище. Има и други структурни типове, но тук ще се ограничим до изучаването на структурния тип *масив*.

Скаларните типове от своя страна се делят на ***изброими*** (дискретни) и ***реални***. Изброимите биват ***целочислени***, ***символни*** и ***логически***. Целочислените типове служат за съхраняване на цели положителни и/или отрицателни числа. Символния тип служи за представяне на произволен клавиш от клавиатурата. Логическият тип представя логическа стойност от вида истина/неистина. Дори и да нямат явно дефиниран логически тип, езиците работят с логически стойности, които се получават от изрази за сравнение и логически изрази. Реалните типове служат за записване на десетични дроби.

Изборът на подходящ тип е много важна част от създаването на една програма, защото може да се окаже, че при създаването на голяма програма, като игра например, във готовия си вариант тя използва в пъти повече памет, отколкото е нужно за нормалната й работа, а това забавя компютъра.

Ето списък на скаларните типове, които ще използваме:

Диапазон	Памет в байтове	Име на типа	
		Pascal	**C++**
- 128 .. 127	1	shortint	char
0 .. 255	1	byte	unsigned char
-32 768 .. 32 767	2	integer	int
0 .. 65 535	2	word	unsigned
-2 147 483 648 .. 2 147 483 647	4	longint	long
0 .. 4 294 967 295	4		unsigned long
с точност 11-12 знака	4	real	float
с точност 15-16 знака	8	double	double
true / false	1	boolean	
символи	1	char	char

Декларирането на променливите трябва да се извършва преди тяхното използване, в противен случай се появява съобщение за грешка.

Деклариране на променливи в Pascal

Променливите се декларират след запазената дума var така:

променливи **:** *тип***;**

където *променливи* е списък от имената на една или повече променливи, разделени със запетая, които ще бъдат от един и същ тип, а *тип* е името на типа според горната таблица.

Декларациите могат да започват на същия ред, където е var или на следващия. Възможно е на един ред да се записват няколко декларации, както и всяка променлива да е на самостоятелен ред. И в двата случая обаче, кода за програмиста става по-нечетлив. Добра практика е променливите от един тип да се декларират на един ред.

Пример:

```
var a,b : integer;
      c : real;
```

Деклариране на променливи в C++

Променливите могат да се декларират на произволно място в програмата, стига това да стане преди или по време на първото им използване. Декларацията изглежда така:

*тип променливи***;**

където *променливи* е списък от имената на една или повече променливи, разделени със запетая, които ще бъдат от един и същ тип, а *тип* е името на типа според горната таблица.

Възможно е на един ред да се записват няколко декларации, както и всяка променлива да е на самостоятелен ред. И в двата случая обаче, кода за програмиста става по-нечетлив. Добра практика е променливите от един тип да се декларират на един ред.

Пример:

```
int a,b;
float c = 23.5;
```

3.2 Оператори, участващи в изрази.

Математически операции:

Операция	Pascal	C++
Събиране	+	+
Изваждане	-	-
Умножение	*	*
Деление	/	/
Целочислено деление	div	/
Остатък от деление	mod	%

Операции за сравнение:

Операция	Pascal	C++
Равно	=	==
По-малко	<	<
По-голямо	>	>
По-малко или равно	<=	<=
По-голямо или равно	>=	>=
Различно	<>	!=

Логически операции:

Операция	Pascal	C++
Отрицание	not	!
Логическо „и”	and	&&
Логическо „или”	or	||
Изключващо „или”	xor	

Други стандартни подпрограми:

Операция	Pascal	C++
Повдигне на втора степен	sqr()	sqr()
Корен квадратен	sqrt()	sqrt()
Абсолютна стойност (модул)	abs()	abs()
Закръгляване	round()	
Цялата част на число	trunc()	floor(), ceil()
Дробната част на число	frac()	
Предишно	pred()	
Следващо	succ()	

В скобите може да има единичен параметър, както и цял израз. Това важи и за горните операции. За C++ изброените по-горе функции се намират във файла **math.h.**

От математиката знаем, че действията в един израз не винаги се изпълняват в реда на записването им, а според техния приоритет, т.е. има строго определен ред – първо изразите, оградени в скоби, след това степенуване, умножение и деление, и накрая събиране и изваждане. При програмирането компютъра също спазва подобна наредба.

Приоритет на операциите:

Операция	Приоритет
Скоби, стандартни подпрограми	най-висок
Степенуване	
Умножение и всички видове деление	
Събиране и изваждане	
Операции за сравнение	
Логически операции	
Присвояване	най-нисък

Най-напред се изпълняват действията с най-висок приоритет. Действията с един и същ приоритет се изпълняват от компютъра в реда, в който са записани, от ляво на дясно. В случай, че искаме дадено действие да се изпълни преди друго, трябва да го оргадим в скоби. Например, правилния запис на $\frac{1}{1-x}$ е 1/(1-x). Ако пропуснем скобите, все едно сме написали $\frac{1}{1}-x$.

3.3 Оператор за присвояване.

Вече говорихме за командата за присвояване като начин на действие. Казано накратко – командата или операторът за присвояване е действие, при което променливата, записана в ляво от знака за присвояване получава нова стойност.

Общия вид на командата за присвояване е:

промелнива **знак_за_присвояване** ***израз ;***

Знакът за присвояване е = , а езика Pascal е :=

Пример:

X := 2; в Pascal
X = 2; в C++

При изпълнението на тази команда се преминава последователно през етапите:

1) изчислява се стойността на израза;
2) получената стойност се присвоява на променливата (т.е. независимо каква е била стойността на променливата, тя вече има нова стойност и това е стойността на израза).

Действието, което изпълнява тази команда дава възможност да се използват изрази като X=X+1. Може да се мисли по следния начин: в дясно на знака за присвояване променливата участва в израза със старата си стойност, а в ляво ще бъде новата й стойност, която ще получи, след като се изчисли израза вдясно. Така ако X е имала стойност 4, новата стойност на тази променлива ще се получи, като към старата се добави 1, старата стойност се губи и от този момент нататък X вече е 5.

3.4 Оператор за отпечатване на екрана.

Това е операторът, с помощта на който компютъра ни съобщава нещо (например търсения отговор), като го изведе на екрана. С него може да се отпечатват както текст, така и числа, а в случай, че се опитате да отпечатате променлива или израз, на екрана ше се изведе тяхната стойост.

С един оператор можем да отпечатаме и повече от една числови или текстови стойности на екрана. В случая числата, изразите, променливите, текста който искате да се отпечата представляват параметрите на оператора. Може да се каже също списък с параметри.

Оператор за отпечатване в Pascal

write(*списък_с_параметри*);
writeln(*списък_с_параметри*);

където списъка с параметри може да съдържа един или повече параметъра, разделени със запетая, или да бъде празен. Текстовите параметри се заграждат в апострофи. Действието на двата оператора се различава само по това, че втория, след отпечатване на списъка с параметри, преминава на нов ред. Често пъти вторият вариант се избира когато желаем да оставим празен ред на екрана, като за целта списъка с параметри се оставя празен и оператора изглежда така – writeln;

Независимо колко оператора write са записани един след друг, техните параметри се отпечатват на екрана на един ред и без интервали между отделните стойности, докато не се срещне указание за преминаване на нов ред от writeln или друг оператор.

Слепването може да се избегне, като се използват модификатори за ширина. Това става като веднага след съответния параметър, който ще се отпечатва се добавят двуеточие

и цяло число, с което се указва върху колко полета да се разположи стойността при отпечатване. Резултата се извежда в дясната част на предвидения брой полета. В случай, че желаем резултата да бъде подравнен в ляво, трябва да запишем отрицателно число. Ако стойността е по-дълга от предвидения брой полета, модификаторът се игнорира.

При десетичните дроби се добавя втори модификатор, с който се указва броя на знаците след десетичната точка. Този модификатор никога не се игнорира.

Пример:

X := 3;	Резултат:
writeln;	минава на нов ред
writeln('Hello !');	Hello !
write(2);	23
writeln(X);	
writeln(2 + X);	5
writeln(2, X);	23
writeln(2:3, X:3);	2 3 {всяко на 3 полета}
writeln(2:-3, X);	2 3 {първото на 3 полета, ляво подравнено}
Y:=2.5;	
writeln(Y);	2.5000000000E+00
writeln(Y:5:2);	2.50 {5 полета, 2 знака след запетаята}
writeln(2, '+', X, '=', 2+X);	2+3=5

Оператор за отпечатване в C++

cout << *параметър* [<< *параметър*…];

където *параметър* може да бъде променлива, константа, израз или указание за преминаване на нов ред. Както ще видите от примерите, с един оператор cout може да се изведат няколко параметъра, като за целта пред всеки от тях се поставя знака <<. Ако искаме изведените стойности да не се слепват можем да отпечатаме между тях празни интервали.

Дробните числа се извеждат с толкова знака след десетичната точка, колкото са необходими, но не повече от шест. В случай, че числото е с по-голяма точност, то се закръглява до шестия знак.

Внимание: Сигурно сте забелязали, че за обикновено и целочислено деление се използва един и същ знак. Как компютъра разбира коя от двете операции да извърши? Правилото е следното – ако и двете стойности, които трябва да се разделят са цели, то резултата също е цяло число, т.е. изпълнява се операцията за целочислено деление, а ако поне една от двете стойности е десетична дроб, тогава се извършва нормално деление.

Ако искаме делението да се извърши задължително по нормалния начин, непосредствено преди него трябва да добавим (float). По този начин казваме на компютъра, че желаем резултата да е дроб, а не цяло число. Тази операция се нарича *преобразуване на тип* и общия й вид е (*тип*). Може да я използваме преди всеки израз, който желаем да е от точно определен тип. Например, ако поставим (int) пред израз, чиято стойност е десетична дроб, това ще довете да отрязване на цялата част без да се извърши закръгляване.

Пример:

X = 3;	Резултат:
cout<<endl;	минава на нов ред
cout<<"Hello !"<<endl;	Hello !
cout<<2<<" "<<X;	2 3
cout<<2 + X<<endl<<"hi"<<"\n";	5 hi
cout<<5/2<<"\n";	2

```
cout<<(float)5/2<<endl;              2.5
cout<<(int)sqrt(3)<<endl;            1
cout<<sqrt(3)<<endl;                 1.732051
cout<<2<<"+"<<X<<"="<<2 + X;         2+3=5
```

За да използваме оператора cout, трябва да включим заглавния файл **iostream.h**.

3.5 Оператор за въвеждане.

Това е операторът, с помощта на който ние съобщаваме на компютъра каква е стойността на дадена променлива, като я въведем с клавиатурата. Стойността може да бъде числова или текстова, в зависимост от типа на променливата. Компютъра „прочита" стойността след като натиснем клавиша Enter.

С един оператор можем да въведем и повече от една променлива. Добра практика е преди всяко въвеждане, първо да изведем на екрана указание към потребителя, за да знае какво се очаква от него. Указанията най-често са във вида „въведете две числа: ", „въведи ръст в сантиметри: " или просто „a=".

Оператор за въвеждане в Pascal

read(*списък_с_променливи*);
readln(*списък_с_променливи*);

където списъка с променливи може да съдържа една или повече променливи, разделени със запетая, или да бъде празен. Действието на двата оператора се различава по това, че втория, след прочитане стойностите на променливите в списъка, преминава на нов ред. Всъщност, ако въвеждате всяка стойност на отделен ред няма да откриете разлики в действието на тези два оператора. Позволено е обаче, при въвеждане от клавиатурата да записвате група стойности, които искате да въведете на един ред, разделени с интервали. В този случай разликата е следната – ако използвате read, компютъра прочита от реда необходимия брой стойности и ако следващия оператор е отново за въвеждане, той продължава да чете от същия ред, ако пък използваме readln, следващия оператор започва четенето от началото на следващия ред.

Често пъти вторият вариант се ползва в края на програмата без параметри и оператора изглежда така – readln; Такъв оператор просто изчаква натискането на клавиш Enter, за да може да приключи действието му. Така потребителският екран с резултата от изпълнението на програмата не се скрива, а „изчаква" потребителя.

Пример:

	Вход	Резултат:
readln;	Enter	изчаква и минава на нов ред
read(a, b);	1 2 3	a=1, b=2
readln(c);	4 5	c=3
readln(d);	6	d=4
read(x);		x=6

Оператор за въвеждане в C++

cin >> *променлива* [>> *променлива…*];

Както се вижда, с един оператор cin може да се въведат и няколко променливи наведнъж, като за целта пред всяка от тях се поставя знака >>. При въвеждането стойностите могат да се запишат както на отделни редове, така и на един ред, разделени със интервали.

Внимание: Компютъра прочита необходимия брой стойности в реда, в който сте ги записали, а ако сте въвели повече на брой стойности от необходимото, те не се игнорират, а се прочитат от следващите в програмата оператори cin, които биха могли да очакват стойности от съвсем друг тип или естество. Това съответно може да доведе до големи объркванията в алгоритъма и неоткриваеми грешки. Като правило преди всяко въвеждане, извеждайте на екрана подходящ текст с указание към потребителя колко и какви точно стойности се очаква да въведе.

Пример:

int a,b,c; float d;	Вход	Резултат:
cin>>a;	1 2	a=1
cin>>b>>c;	3.4	b=2, c=3 //вместо 3.4
cin>>d;	5	d=0.4 //вместо 5
cout<<"vavedi 2 ocenki:";		
cin>>o1>>o2;	4 5 3	o1=4, o2=5
cout<<"vavedi rysta si:"		
cin>>rust;	172	rust=3 //вместо 172

За да използваме оператора cin, трябва да включим заглавния файл **iostream.h**.

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Какво име и тип бихте избрали за променлива, в която ще се съхранява: ръст в сантиметри; тегло в килограми; средномесечна температура; марка автомобил; ръст в метри; години; среден успех; тегло на хранителна стока в грамове; име на паралелка; брой студени дни в месеца?

2. Запищете на изучавания от Вас език изразите: $4x-5y^2$; $15+(-3)^2+\left[\frac{5}{3}\right]$; $\frac{a-b}{2y}$; $\frac{(x+y)^2}{ab}$; $\left|(x-1)^2-5\right|+12$; ax^3+2x^2+bx+c; $|x+a|+|y+b|$; $\sqrt{1+\frac{|a|+|b|}{a^2+b^2}}$

3. Пресметнете стойността на изразите:
   а) в Pascal: abs (17 mod 5 – 51 div 5);
   б) в Pascal: 5 * 7 mod 2 – 18 div 3 * (succ(sqr(-2)));
   в) в C++: abs(17%3 – 10/3);
   г) в C++: 5 * 7 mod 2 – 5 / 2.0 * sqr(-2).

4. Напишете програма, която пресмята:
   а) сбора на три числа;
   б) среден успех по 4 оценки;
   в) разликата на две числа;
   г) уравнението $ax+b=c$, по въведени *a*, *b* и *c*;
   д) средно геометричното на две числа.

5. Напишете програма, която намира:
   а) лицето и обиколката на квадрат по въведена страна;
   б) периметъра на правоъгълник по въведени две страни;
   в) лицето на трапец по въведени две основи и височина;
   г) периметъра на равнобедрен триъгълник по въведени две страни;
   д) лицето на триъгълник по въведени страна и височина към нея;

е) лицето на правоъгълен равнобедрен триъгълник с известен катет;

ж) лицето на кръг и обиколката на на окръжност по въведен радиус;

з) дължината на хипотенуза на правоъгълен триъгълник по дадени два катета;

и) лицето на квадрат, ако е известен диагонала му;

й) обем на правоъгълен паралелепипед по въведени страни;

к) обема на сфера по въведен радиус;

л) пълната повърхнина и обема на куб по въведена дължина на страна.

6. Създайте програма, която пресмята средния успех на класа за първи срок и съобщава мястото ви спрямо това ниво (т.е. с колко вашата оценка е над или под средното ниво).

7. Създайте програма, която по въведени ръст в сантиметри и тегло в килограми, показва отклонението на теглото спрямо оптималното. Приемете, че оптималното тегло се получава, като от ръста извадите числото 110 за момиче или 100 за момче.

8. Създайте програма, която по въведени две числа намира произведението им и извежда на екрана последната му цифра.

9. Да се напише програма, която въвежда положително трицифрено число и извежда на отделни редове цифрите на стотиците, десетиците и единиците на числото.

10. Дадена стока струва *x* лева. Програмата да изведе цената й в стотинки.

11. По въведено време в минути на екрана да се изведат часове и минути. Например ако се въведе 135 на екрана да се появи 2 hours 15 minutes.

12. Напишете птограма, която разменя стойостите на две променливи.

13. Напишете програма, която намира периметъра на триъгълник ABC, ако са известни координатите на точките A, B и C. *Упътване:* Разстоянието мжду две точки A(x1, y1) и B(x2, y2) се пресмята по формулата $d(A,B)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

14. Напишете програма, която намира лицето на триъгълника ABC, ако са известни координатите на точките A, B и C. *Упътване:* Първо трябва да се намерят дължините на страните на триъгълника, като се използва горната формула, след което лицето пресмятаме по Хероновата формула за лице на триъгълник: $S=\sqrt{p.(p-a).(p-b).(p-c)}$, където p е полупериметъра на триъгълника.

Тема 4. Условен оператор IF.

Често пъти искаме при изпълнение на програма, компютъра да вземе решение каква да бъде следващата му стъпка въз основа на някакъв критерий. Това се постига с използването на управляващия оператор IF (ако).

Действието на оператора IF в най общия случай е следното: компютъра проверява условието, записано след IF и ако то се окаже вярно, изпълнява оператора, записан след условието, а ако то не е вярно се изпълнява оператора, записан след запазената дума ELSE, значението на която е „в противен случай" или „иначе".

Това е така наречената ***пълна форма*** на оператора, при която компютъра задължително избира един от двата оператора и изпълнява само него, след което преминава към оператора, записан след края на оператора IF.

В случай, че желаем да се изпълнят повече от един оператори, за всеки от езиците има предвиден механизъм за обобщаване на няколко оператора в блок, наречен *съставен оператор*. Всички действия в рамките на съставен оператор се изпълняват последователно, без прекъсване.

В някои случаи е по-удобно да се използва ***кратката форма*** на оператора IF. В нея липсва частта ELSE и когато условието се окаже грешно, компютъра просто преминава към следващия оператор в кода на програмата.

Има и задачи, при които се налага компютъра да избере измежду повече от две възможности. В този случай се използва така нареченото ***влагане на оператори***. Това се постига, като в рамките на оператора IF се запише втори такъв, с който се проверява условие, различно от първото. Влагането може да стане както в първата част след условието, така и във втората, след думата ELSE. Самите вложени оператори могат да бъдат както в кратка, така и в пълна форма. Добре е да не се влага на повече от три нива, тъй като кодът на програмата става много труден за проследяване и откриване на грешки.

За да разберете употребата и начина на действие на оператора, ще разгледаме три примера, в които се използват съответно кратката форма, пълната форма и влагане:

Задача 1: Да се пресметне частното на две числа, ако това е възможно.

Алгоритъм: *От математиката знаем, че на нула не може да се дели, следователно след въвеждане на двете числа, трябва да се провери дали второто не е равно на нула и ако е така, компютъра да не прави нищо.*

Задача 2: По въведена от клавиатурата година да се изведе съобщение дали тя е високосна.

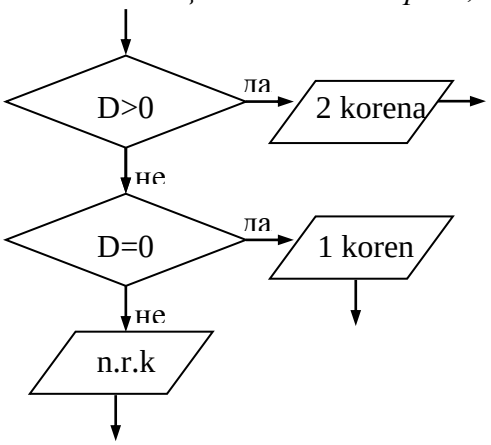
Алгоритъм: *Ако прегледате календара можете да установите, че всички високосни години се делят на четири без остатък, например 2012, 2016 и др. В този случай, най-бързото решение е да проверим дали въведеното число се дели на четири с операцията деление по модул, и ако това е така да съобщим, че годината е високосна, а в противен случай, че не е.*

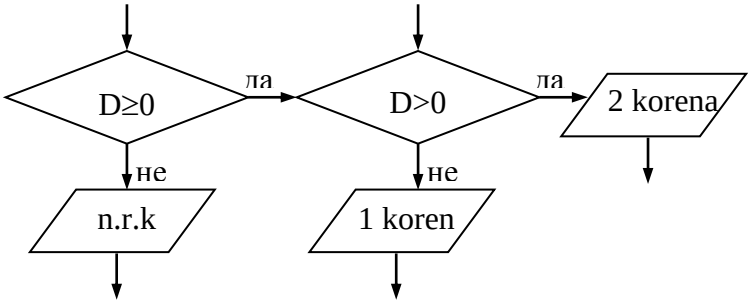
Задача 3: По въведени от клавиатурата три числа *a*, *b* и *c*, които са коефициенти на квадратното уравнение $ax^2 + bx + c = 0$, да се определи броя на реалните му корени.

Алгоритъм: *От часовете по математика знаем, че при отрицателна дискриминанта квадратното уравнение няма реални корени, при положителна има два, а когато стойността й е нула коренът е един. Проблемът е как да организираме проверката – колко и какви условия да проверим.Най-общо вариантите са два:*

1: Проверяваме дали е положителна и ако е така съобщаваме за 2 корена, в противен случай проверяваме дали не е равна на нула. Ако това условие е вярно, значи коренът е един, а ако не е - остава третата възможност – нула. Вижда се, че влагането става в частта ELSE, а проверките са две.

2: Проверяваме дали дискриминантата е неотрицателна и ако това е така веднага правим втора проверка, за да доуточним колко точно са кореният – един или два. В случай че първото условие е невярно отпечатваме, че няма реални корени. По този начаин влагането става в първата част.





И в двата случая се вижда, че проверките са с една по-малко от броя на възможните изходи.

4.1 Pascal.

Кратка форма на оператора:

if *условие* then *оператор;*

В случаите, когато желаем да се изпълнят повече от един оператори, трябва да ги поставим между begin и end. Така те се възприемат като един оператор, често наричан *съставен оператор* или *блок* от оператори.

Операторът може да се запище на колкото реда желаете. Добре е да се избягва варианта на един ред понеже реда става много дълъг, кода излиза извън екрана и се налага превъртане. Най-често запазените думи if и then се записват една под друга. Знак **;** се поставя само на края и евентуално в рамките на оператора след then, ако той е съставен.

Пример: Решение на задача 1:

```
var a, b : integer;
        c : real;
begin
  write('a=');  readln(a);
  write('b=');  readln(b);
  if b<>0 then
    begin
      c:=a/b;
      writeln('chastnoto e: ', c:4:2);
    end;
  readln;
end.
```

Пълна форма на оператора:

if *условие* then *оператор_1* else *оператор_2***;**

И тук правилата са както по-горе. Независимо къде е записана думата then, препоръчително е else да е на едно ниво с if (на един или повече реда надолу в кода). Така много лесно се вижда къде започва втората част на оператора. Този начин на подреждане е още по-важен, когато става дума за вложени оператори, защото се появяват по няколко

думи then и else, а подходящото отместване ни позволява да определим точно в кой случай се намираме.

Пример: Решение на задача 2:

```
var year, ostatuk : integer;
begin
  write('godina: '); readln(year);
  ostatuk := year mod 4;
  if ostatuk = 0 then writeln('visokosna')
  else writeln('normalna');
  readln;
end.
```

Влагане на оператори:

Най-често се използва влагане на едно ниво и то изглежда така:

Влагане в първата част:

```
if условие_1 then
      if условие_2 then оператор_1
      else оператор_2
else оператор_3;
```

Влагане във втората част:

```
if условие_1 then оператор_1
else  if  условие_2 then оператор_2
      else оператор_3;
```

Според нуждите на задачата може да се наложи влагане и в двете части на оператора, както и на повече нива. Самите оператори могат да бъдат в пълна или кратка форма.

Пример: Решение на задача 3 (вариант 1 / вариант 2):

```
var a, b, c, d : real;
begin
  write('a=');  readln(a);
  write('b=');  readln(b);
  write('c=');  readln(c);
  d := sqr(b) – 4*a*c;
  if d > 0 then writeln('2 korena')
  else  if d = 0 then writeln('1 koren')
        else writeln('n.r.k');
  readln;
end.
```

За втория вариант ще запишем само двата вложени оператора, останалата част от програмата е същата.

```
 if d >= 0 then
     if d > 0 then writeln('2 korena')
     else writeln('1 koren')
 else writeln('n.r.k');
```

4.2 C++

Кратка форма на оператора:

if (*условие*) *оператор;*

В случаите, когато желаем да се изпълнят повече от един оператори, трябва да ги поставим между фигурни скоби **{ }**. Така те се възприемат като един оператор, често наричан *съставен оператор* или *блок* от оператори.

Операторът може да се запище на колкото реда желаете. Добре е да се избягва варианта на един ред когато оператора е съставен, понеже реда става много дълъг, кода излиза извън екрана и се налага превъртане. Знак **;** се поставя само на края и евентуално в рамките на оператора, ако той е съставен.

Пример: Решение на задача 1:

```
#include<iostream.h>
#include<conio.h>

void main () {
  clrscr();

  float a, b, c;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  if (b!=0) {
     c=a/b;
     cout<<"chastnoto e: "<<c;
  }

  getch();
}
```

Функцията clrscr() действа като гъба върху дъска - изчиства екрана от текста, който е върху него. Добре е в началото на всяка програма да я добавяме, в противен случай, екрана може да се запълни с информация от предишни изпълнения на програми и да не разбрем кое точно е резултата.

Функцията getch() изчаква натискането на произволен клавиш и по този начин постигаме ефект сякаш компютъра изчаква потребителя да разгледа резултата преди да го скрие. И за двете функции е необходимо включването на заглавния файл conio.h.

След това обяснение лесно може да се достигне до извода, че първите четири реда, както и последните два, ще присъстват във всяка програма, която пишем. Затова и примерите от тук нататък ще бъдат показвани без тези редове. Нещо повече – ако съхраните тези редове във файла noname00.cpp, те ще се появяват всеки път, когато създавате нов файл.

Пълна форма на оператора:

if (*условие*) *оператор_1***;**
else *оператор_2***;**

И тук правилата са както по-горе. Препоръчително е else да е на едно ниво с if (на един или повече реда надолу в кода). Така много лесно се вижда къде започва втората част на оператора. Този начин на подреждане е още по-важен, когато става дума за вложени оператори, защото в комбинация с подходящото отместване ни позволява да определим точно в кой случай се намираме.

Пример: Решение на задача 2:

```
int  year, ostatuk;
cout<<"godina: "; cin>>year;
  ostatuk = year % 4;
  if (ostatuk == 0) then cout<<"visokosna"<<endl;
  else cout<<"normalna"<<endl;
```

Условна операция:

В някои специфични случаи е по-удобно да се използва условната операция. Действието й е подобно на пълната форма на оператора if, но тъй като резултата от изпълнението й е някаква конкретна стойност, тя не може да се използва самостоятелно, а само в рамките на израз, присвояване или оператор за отпечатване. Синтаксисът й е:

условие ? *стойност_1* : *стойност_2*

където *условие* е някакво условие, чиято стойност трябва да се провери, а *стойност_1* и *стойност_2* са резултата от операцията в случай, че условието е било съответно вярно или грешно. Двете стойности трябва да са от един и същ тип.

Ето как би изглеждала горната задача с използване на условна операция:

```
int  year;
char *result;              //promenliva ot tip simvolen niz
cout<<"godina: "; cin>>year;
result = (year % 4 == 0) ? "visokosna" : "normalna";
cout<<result<<endl;
```

Влагане на оператори:

Най-често се използва влагане на едно ниво и то изглежда така:

Влагане в първата част:

```
if (условие_1)
      if (условие_2) оператор_1;
      else оператор_2;
else оператор_3;
```

Влагане във втората част:

```
if (условие_1) оператор_1;
else  if  (условие_2) оператор_2;
      else оператор_3;
```

Според нуждите на задачата може да се наложи влагане и в двете части на оператора, както и на повече нива. Самите оператори могат да бъдат в пълна или кратка форма. Обърнете внимание, че след всеки оператор се поставя знака **;**. Пропускането му би довело до грешка.

Пример: Решение на задача 3 (вариант 1 / вариант 2):

```
float a, b, c, d;
cout<<"a=";  cin>>a;
cout<<"b=";  cin>>b;
cout<<"c=";  cin>>c;
d = b*b – 4*a*c;
if (d > 0) then cout<<"2 korena";
else  if (d == 0) cout<<"1 koren";
      else cout<<"n.r.k';
```

За втория вариант ще запишем само двата вложени оператора, останалата част от програмата е същата.

```
if (d >= 0)
  if (d > 0) cout<<"2 korena";
  else cout<<"1 koren";
else cout<<"n.r.k";
```

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Запишете логически израз, който има стойност истина когато:

   а) *x* е положително;

   б) *x* принадлежи на интервала [10, 20];

   в) *x* е извън интервала [10, 20];

   г) числата *x*, *y* и *z* са равни помежду си;

   д) само две от числата *x*, *y* и *z* са равни помежду си;

   е) всяко от числата *x*, *y* и *z* е положително;

   ж) поне едно от числата *x*, *y* и *z* е положително;

   з) нито едно от числата *x*, *y* и *z* не е положително;

   и) точно едно от числата *x*, *y* и *z* е положително;

   й) *n* се дели без остатък на *p*;

   к) положителните числа *a*, *b* и *c* могат да бъдат страни на триъгълник;

л) точката с координати (*x*, *y*) е в първи или трети квадрант и не лежи върху никоя от осите;

м) x е най-малкото от числата *a*, *b* и *c;*

н) естественото число *n* не завършва на 9;

о) двуцифреното число *n* е с различни цифри;

п) квадратното уравнение $ax^2+bx+c=0$ има точно един корен (а безброй много?);

р) цифрата 5 влиза в десетичния запис на трицифреното число *k*;

с) годината *y* е високосна.

2. Създайте програма, използваща някой от горните логически ирази, например програма, която проверява дали 3 числа могат да са страни на триъгълник.

3. Създайте програма, която определя по-голямото от 2 числа.

4. Създайте програма, която по въведени две числа отпечатва „kvadrat", ако са равни и „pravoagalnik", ако не са.

5. Създайте програма, която по зададени страна и височина към нея намира лицето на тръгълник и ако то е по-голямо от 1000 отпечатва на екрана думата "big", а в противен случай отпечатва "small".

6. Създайте програма, която по зададени две страни на успоредник намира периметъра му. После ако двете страни са равни отпечатва на екрана "romb", а в противен случай "usporednik".

7. Създайте програма, която пресмята $f(x)=\begin{cases}3x-3, за.x\geq 3\\ x+3, за.x<3\end{cases}$

8. Създайте програма, която пресмята $f(x)=\begin{cases}x+y & ako\ \ x<y\\ x-y & ako\ \ x\geq y\end{cases}$

9. Създайте програма, която намира най-малкото от три числа.

10. Създайте програма, която намира най-голямото от четири числа.

11. Създайте програма, пресмятаща корените на квадратно уравнение.

12. Да се провери дали в трицифреното число N има повтарящи се цифри.

13. Създайте програма, която по зададени страна и височина към нея намира лицето на успоредник и ако то е по-малко от 1000 отпечатва на екрана думата "malak", ако е между 500 и 1000 – "sreden", а в противен случай - "goliam".

14. Създайте програма, която по зададена страна на квадрат намира лицето му, ако тя е по-малка от 20, полупериметъра му, ако е равна на 20 и периметъра, ако е по-голяма от 20.

15. Създайте програма, която по зададена страна, ако е по-голяма от 10 намира лицето на кубче, а ако е по-малка от 10 – обема му, а ако е равна на 10 – и двете.

16. Създайте програма, която определя стойността на $f(x)=\begin{cases}x^2-3 & ako\ \ x<3\\ 3 & ako\ \ x=3\\ 5x & ako\ \ x>3\end{cases}$

Тема 5. Оператор за многовариантен избор.

От името на оператора се вижда, че той се използва за разклонение на програмата при повече възможни варианти. В предишния урок уточнихме, че влагането на три и повече оператора IF прави кода много сложен и нечетлив. В такива случай оператора за многовариантен избор се явява доста добро решение.

Операторът работи така – проверява стойността на дадена променлива и според това каква е тя, изпълнява предвидените за този случай действия. Отделните възможни варианти се описват във вида – *възможна стойност* : *съответно действие*. Компютъра проверява възможните стойности в реда, в който са описани и когато стигне до съответствие изпълнява предвидените за този случай действия и предава управлението на следващия в кода на програмата оператор, т.е. при достигане на съответствие не се проверяват следващите варианти. Ако програмиста прецени, че е необходимо, може да опише какви действия да се изпълнят в случай, че не се открие съответствие.

5.1 Pascal: оператор CASE.

Общия вид на оператора е:

case *променлива* of
  списък_стойности_1 **:** *оператор_1*;
  [*списък_стойности_2* **:** *оператор_2*;
  …
  else *оператор_n*;]
end;

където *променлива* е името на променливата, чиято стойност ще се проверява, *оператор_1*, *оператор_2* и т.н. са действията, които трябва да се изпълнят ако стойността на променливата съвпадне с някоя от стойностите, изброени в *списък_стойности_1*, *списък_стойности_2* и т.н. След else се записва операторът, които трябва да се изпълни в случай, че не е открито съвпадение. Тази част не е задължителна.

В случаите, в които желаете да се изпълняват по повече от един оператор, не забравяйте да ги поставите между begin и end.

Списъкът със стойности може да бъде:
стойност [, *стойност***…**] – например 2, 5, 10
стойност **. .** *стойност* – например 2 **..** 10, ‘A’ **..** ‘Z’, ‘a’ **..** ‘z’

Пример:

```
uses crt;      {za da moje da se polzva textcolor()}
var day : integer;
begin
  write(‘vavedi nomer na den ot sedmicata: ‘);
  readln(day);
  case day of
    1 .. 5 : writeln(‘ucheben den’);
    6, 7 : begin
             textcolor(2);
             writeln(‘pochiven den’);
           end;
    else writeln(‘vaveli ste greshno chislo”);
  end;
 readln;
end.
```

5.2 C++: оператор SWITCH.

Общия вид на оператора е:

```
switch (променлива) {
  case стойност_1 : оператори_1; break;
  [case стойност_2 : оператори_2; break;
   case стойност_3 : [case стойност_4 : …] оператори_3; break;
  …
  default : оператор_n;]
}
```

където *променлива* е името на променливата, чиято стойност ще се проверява, *оператори_1*, *оператори_2* и т.н. са действията, които трябва да се изпълнят ако стойността на променливата съвпадне съответно със *стойност_1*, *стойност_2* и т.н. След default се записват операторите, които трябва да се изпълни в случай, че не е открито съвпадение. Тази част не е задължителна. Операторът break прекъсва изпълнението на switch и предава действието на следващия след него оператор в кода на програмата. Това означава, че ако бъде открито съвпадение ще се изпълнят предвидените за него действия и няма да се проверяват за съвпадение следващите случаи описани до края на оператора.

Пример:

```
int day;
cout<<"vavedi nomer na den ot sedmicata: ";
cin>>day;
switch (day) {
    case 1: cout<<"monday"; break;
    case 2: cout<<"tuesday"; break;
    case 3: cout<<"wednesday"; break;
    case 4: cout<<"thursday"; break;
    case 5: cout<<"friday"; break;
    case 6: case 7: cout<<"weekend"; break;
    default: cout<<"no such day";
}
```

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Създайте програма, която по въведена числова оценка извежда оценката с думи.

2. Създайте програма, която по въведено арабско число извежда съответното римско число според таблицата:

Арабски числа	1	5	10	50	100	500	1000
Римски числа	I	V	X	L	C	D	M

3. Създайте програма, която определя вида на въведен символ: гласна, съгласна или друг символ.

4. Създайте програма, която пресмята лицето на една от фигурите: кръг, триъгълник или правоъгълник. Възможните действия да се извеждат в началото под формата на меню.

5. Създайте програма, която по въведен номер на месец извежда съответния лихвен процент в определена банка. За целта ползвайте таблицата:

Месец	1	2	3	4	5	6	7	8	9	10	11	12
Лихвен %	3.8	3.5	3.5	3.5	4.1	4.1	4.1	4.1	4.9	4.9	4.9	3.8

6. Създайте програма, която по въведен номер на месец извежда неговото име и броя на дните в него. За определяне на дните за месец февруари да се поиска въвеждане допълнително и на годината.

7. Създайте програма, която определя стойността на :

а) $f(x)=\begin{cases}\sqrt{x}-1 & ako\ \ x=10,20\\ 0 & ako\ \ x=0\\ x^2 & ako\ \ x\neq 0,10,20\end{cases}$ б) $f(x)=\begin{cases}1 & ako\ \ x\in[10,15]\\ 2 & ako\ \ x=2\\ 17x & za\ \ всички\ \ останали\ случаи\end{cases}$

Тема 6. Оператори за цикъл.

В много случаи при писането на програма се изисква многократно повторение на едно или повече действия. Тогава ползваме оператори за цикъл. Във всеки от езиците има дефинирани три вида циклични оператора – цикъл с предусловие, цикъл с постусловие и цикъл, управляван от променлива (FOR). Освен тези оператори има и още една възможност за организиране на цикъл – чрез комбинирано използване на операторите IF (за проверка на условие) и GOTO (за преминаване към друг ред).

Почти всяка задача изискваща цикъл може да се реши и по четирите начина. В такъв случай може би ще се запитате защо са нужни цели четири варианта на цикличен оператор? Отговорът е следния – четирите варианта предполагат различен начин на действие от страна на компютъра и за всяка задача един или най-много два от вариантите се оказват по-подходящи и с по-кратък код.

Всеки цикъл има *тяло* и *условие*. В тялото се записват действията, които ще се повтарят, а стойността на условието определя кога ще прекъсне повторението на цикъла. Всяко изпълнение на тялото на цикъла, т.е. всяко повторение се нарича *итерация*. В зависимост от местоположението на условието има два вида цикли – цикли с предусловие и такива с постусловие.

При циклите с предусловие най-напред се проверява условието и в зависимост от неговата стойност компютърът решава дали да повтори тялото. Това означава, че е възможно цикълът да не се повтори нито веднъж. Освен стандартно дефинирания цикъл с предусловие, към този вид спада и цикъл FOR – компютъра първо проверява дали е достигната крайната стойност на управляващата променлива и чак след това изпълнява действията в тялото.

При циклите с постусловие първо се изпълняват действията в тялото и чак след това се прави проверка, в резултат на която компютърът решава дали да повтори отново тялото на цикъла. Тялото на този вид цикъл задължително се изпълнява поне един път. Освен стандартно дефинирания цикъл с постусловие, към този вид спада и цикъла, организиран с GOTO – компютъра изпълнява едно или повече последователни действия, достига до оператора IF и ако условието се окаже вярно, се връща нагоре в кода на програмата за да изпълни действията отново.

Най-често в условието се проверява стойността на някаква променлива, наричана още *управляваща променлива*. Логично е стойността на тази променлива периодично да се променя, в противен случай условието би било винаги вярно или винаги грешно. Промяната на стойността най-често се осигурява като се използва оператор за присвояване, в който променливата присъства и в двете страни, т.е. задава се правило как от старата стойност да се получи новата. А как се получава първата стойност? За целта преди цикъла се фиксират начални стойности на всички променливи, които променят стойността си по този начин в тялото на цикъла. Началните стойности се избират така, че при първата итерация да се получат и/или използват първите стойности, за които искаме да се повтарят действията.

За да можете да направите сравнение между различните варианти ще решим две задачи по всеки от четирите начина:

Задача 1: Да се пресметне и отпечата сумата на числата от 1 до 100.

Преди компютъра да започне да събира числата, сумата има стойност нула, понеже още нищо не е добавено.

Задача 2: Да се отпечатат квадратите на четните числа в интервала от 10 до 20.

6.1 Организиране на цикъл с помощта на GOTO.

Операторът GOTO е известен с името оператор за безусловен преход. Може да си го превеждате като „мини на … ред”. При изпълнението си той предава управлението на реда с указаното име, а самия ред може да се намира по-нагоре или по-долу в кода на програмата. Когато предаваме управлението на предходен ред се получава цикъл. Използването само на GOTO обаче води до безкраен цикъл, затова е необходимо да осигурим прекъсването му посредством проверка на някакво условие с IF. Трябва да се погрижим също в даден момент условието да промени стойността си, в противен случай отново бихме достигнали до безкраен цикъл.

Синтаксис на оператора:

goto *етикет*;

За да може да използвате оператора goto, реда на който искате да се върнете трябва да бъде именуван, да му се постави така наречения етикет. Това става, като пред реда се запише име по ваш избор, последвано от двуеточие. Името трябва да започва с буква. За езика Pascal имената на етикетите, които ще бъдат използвани се изброяват в началото на програмата в раздел label.

Да видим как бихме записали първата задача с думи:

1. начало
2. S=0
3. i=1
4. S=S+i
5. i=i+1
6. ако i≤100, мини на стъпка 4
7. отпечатай C
8. край

В това описание стъпки 2 и 3 представляват началните стойности, стъпки 4 и 5 – тялото на цикъла а стъпка 6 – условието за край на цикъла.

Припомнете си как се описва този алгоритъм чрез блок-схема.

Ето как изглежда решението на задачата на Pascal/C++ в този ред:

```
label repeat;
var s, i : integer;
begin
  s:=0;
  i:=1;
repeat: s:=s+i;
  i:=i+1;
  if i<=100 then goto repeat;
  writeln(s);
end.
```

```
…
int s, i;
s=0;
i=1;
repeat: s=s+i;
i=i+1;
if (i<=100) goto repeat;
cout<<”s=”<<s;
…
```

Ето как изглежда решението на втората задача на Pascal/C++ в този ред:

```
label repeat;
var i : integer;
begin
  i:=10;
repeat: writeln(i*i);
  i:=i+2;
  if i<=20 then goto repeat;
  end.
```

```
…
int i;
i=10;
repeat: cout<<i*i<<endl;
i=i+2;
if (i<=20) goto repeat;
  …
```

6.2 Цикъл с предусловие.

Синтаксис на оператора в Pascal

while *условие* do
оператор;

Операторите в цикъла се повтарят докато условието има стойност истина, при неизпълнено условие цикълът прекъсва. По-често *оператор* е съставен оператор – няколко оператора, поставени между begin и end.

Синтаксис на оператора в C++

while (*условие) оператор;*

Операторите в цикъла се повтарят докато условието има стойност истина, при неизпълнено условие цикълът прекъсва. По-често *оператор* е съставен оператор – няколко оператора, поставени между **{** и **}**.

Ето как изглежда решението на първата задача на Pascal/C++ в този ред:

```
var s, i : integer;
begin
  s:=0;
  i:=1;
while i<=100 do
  begin
     s:=s+i;
      i:=i+1;
  end;
  writeln(s);
end.
```

```
…
int s, i;
s=0;
i=1;
while (i<=100) {
   s=s+i;
   i=i+1;
}
cout<<”s=”<<s;
…
```

Ето как изглежда решението на втората задача на Pascal/C++ в този ред:

```
var i : integer;
begin
  i:=10;
while i<=20 do
   begin
     writeln(i*i);
     i:=i+2;
   end;
end.
```

```
…
int i;
i=10;
while (i<=20) {
   cout<<i*i<<endl;
   i=i+2;
}
…
```

6.3 Цикъл с постусловие.

Синтаксис на оператора в Pascal

repeat
оператор
until *условие*;

Операторите в цикъла се повтарят докато условието има стойност лъжа, стане ли вярно цикълът прекъсва. По-често *оператор* е съставен оператор. Непосредствено преди until **не трябва** да има знак **;**

Синтаксис на оператора в C++

do
оператор
while (*условие*);

Операторите в цикъла се повтарят докато условието има стойност истина, при неизпълнено условие цикълът прекъсва. По-често *оператор* е съставен оператор.

Ето как изглежда решението на първата задача на Pascal/C++ в този ред:

```
var s, i : integer;
begin
  s:=0; i:=1;
repeat
  begin
    s:=s+i;
    i:=i+1;
  end
until i>100;
  writeln(s);
end.
```

```
…
int s, i;
s=0;
i=1;
do{
  s=s+i;
  i=i+1;
} while (i<=100);
cout<<"s="<<s;
…
```

Ето как изглежда решението на втората задача на Pascal/C++ в този ред:

```
var i : integer;
begin
  i:=10;
repeat
  begin
    writeln(i*i);
    i:=i+2;
  end
until i>20;
end.
```

```
…
int i;
i=10;
do {
  cout<<i*i<<endl;
  i=i+2;
} while (i<=20);
…
```

6.4 Цикъл FOR.

Този цикъл е известен под името *цикъл, управляван от промнлива*. Характерно за него е, че винаги знаем или можем да определим предварително колко пъти ще се повтори. За целта на компютъра се съобщават началната и крайната стойности на управляващата променлива. Тя последователно заема всяка от стойностите в този интервал, като всеки път цикълът се повтаря. При достигане на крайната стойност се извършва последна итерация и цикълът приключва.

Синтаксис на оператора в Pascal

for *променлива* := *нач.ст.* to/downto *кр.ст.* do
оператор;

където *променлива* е името на управляващата променлива. Стойностите, които тя заема могат да са от всеки изброим тип – целочислен или символен, т.е. можем да броим както от 1 до 26, така и от ‘a’ до ‘z’. В случай, че началната стойност е по-малка от крайната трябва да използваме думата to. Ако пък крайната стойност е по-малката – се използва downto. След всяка итерация стойността на управляващата променлива се увеличава или намалява с единица, в зависимост от това коя дума използваме, докато не се достигне крайната стойност. Ако началната и крайната стойност съвпадат цикълът се изпълнява само

един път, ако пък сме сбъркали думата (to вместо downto или обратно) – нито веднъж. По-често *оператор* е съставен оператор.

Внимание: Операторът е предвиден за използване само на последователни стойности. Ако искаме нещо да се повтаря за числа, които са през равен интервал, например само за четните и в същото време за целта да ползваме for – трябва да правим проверка с if. В такива случаи се използва някой от другите три варианта за цикъл.

Синтаксис на оператора в C++

for (*израз_1*; *израз_2*; *израз_3*)
  оператор;

В *израз_1* се записват всички необходими начални стойности, разделени със запетая.

израз_2 представлява условието за край на цикъла – цикълът се повтаря докато то е вярно, когато стане грешно цикълът прекъсва. Най-често то представлява проверка дали управляващата променлива е достигнала някаква конкретна стойност.

С *израз_3* се задава правилото, по което да се получава следващата стойност на управляващата променлива. Когато искаме стойността й да се увеличава или намалява с единица можем да използваме операциите ++ и --. Например, ако управляващата променлива е i, можем да запишем i++ вместо i=i+1, съответно i-- вместо i=i-1.

Всеки един от трите израза може да бъде пропуснат. Възможно е да не се задават начални стойности, а да се използват такива, които са дефинирани преди цикъла. Ако втория израз се пропусне, компютъра счита, че условието е винаги вярно и в такъв случай трябва да се подсигури друга възможност за прекъсване на цикъла, като се използва някой от операторите за прекъсване. Третия израз се пропуска когато не ни е необходимо правило за промяна стойността на управляващата променлива или е осигурен друг начин за промяната й, например чрез въвеждане от клавиатурата, т.е. промяната може да става в самото тяло на цикъла. Възможно е дори в даден цикъл да се пропуснат и трите израза, но местата им трябва да бъдат фиксирани като в скобите се запищат двата знака **;**

Внимание: Важно е как подбираме трите израза. Ако началната и крайната стойности са равни, цикълът ще се повтори само веднъж. Ако началната стойност е по-малка от крайната, в условието за край се ползва знак < или ≤, но в *израз_3* сме указали намаляване стойността на управляващата променлива, условието ще е винаги изпълнено и цикълът ще се повтаря докато стойността на управляващата променлива не достигне минималната за съответния тип. Ако пък сме указали увеличаване стойността на управляващата променлива, но сме обърнали знака в условието (> или ≥) цикълът няма да се изпълни нито веднъж. Обмислете какви други варианти има.

Ето как изглежда решението на първата задача на Pascal/C++ в този ред:

```
var s, i : integer;
begin
 s:=0;
 for i:=1 to 100 do
   s:=s+i;
 writeln(s);
end.
```

```
…
int s, i;
s=0;
for (i=1; i<=100; i++)
   s=s+i;
cout<<”s=”<<s;
…
```

Ето как изглежда решението на втората задача на Pascal/C++ в този ред:

```
var i : integer;
begin
  for i:=10 to 20 do
     if i mod 2 = 0 then
        writeln(i*i);
end.
```

```
…
int i;
for (i=10; i<=20; i=i+2)
   cout<<i*i<<endl;
…
```

6.5 Влагане на оператори.

Понякога решението на дадена задача изисква в тялото на един цикъл да бъде включен друг цикъл. В такива случаи говорим за влагане на циклични оператори. Когато пишем такива програми трябва да имаме предвид следното:

- за всяко повторение (итерация) на външния цикъл се извършват пълен брой повторения за вътрешния, т.е ако външния се повтаря 5 пъти, а вътрешния 4 пъти, общия брой повторения за вътретшния цикъл ще е 20 (освен ако не се използва оператор за прекъсване);
- вложените оператори могат да са от всеки от разгледаните видове;
- за всеки от циклите трябва да се използва различна управляваща променлива, в противен случай резултата изобщо няма да отговаря на очаквания. Обикновено за име на управлаващите променливи се използват i за външния и j за вътрешния цикъл, а типа им най-често е целочислен;
- повечето задачи, изискват влагане само на два цикъла. Влагането на повече от два циклични оператора прави програмата прекалено сложна за проследяване.

Да видим как с помощта на два вложени оператора FOR може да се отпечата таблицата за умножение в Pascal/C++, записани в този ред:

```
var i, j : integer;
begin
 for i:=1 to 10 do
  begin
    for j:=1 to 10 do
      writeln(i, '*', j, '=', i*j);
    readln;
  end;
end.
```

```
…
int i, j;
for (i=1; i<=10; i++) {
  for (j=1; j<=10; j++)
    cout<<i<<"*"<<j<<
        "="<<i*j<<endl;
  getch();
}
…
```

Ето как работи програмата: i последователно заема стойностите от 1 до 10, като за всяка от тях компютърът отпечатва таблицата за това число, изпълнявайки вътрешния цикъл, след което ни изчаква да натиснем клавиш преди да пристъпи към следващото повторение на външния цикъл.

Изпробвайте програмата, след което опитайте да замените единия или и двата цикъла FOR с цикъл от друг вид.

6.6 Оператори за прекъсване.

Понякога се налага цикъла да бъде прекъснат преди да бъдат извършени пълния брой повторения. В такива случаи се използват операторите за прекъсване.

Операторите за прекъсване на цикъл са два вида:

- оператор, прекратяващ изпълнението на текущата итерация;
- оператор, прекратяващ изпълнението на цикъла.

Първия оператор изглежда така: continue; Най-често се използва в комбинация с оператор if. Когато компютъра стигне до него, той прескача всички оператори записани след това в тялото на цикъла, като преминава направо към следващата итерация.

Втория оператор има вида: break; Този оператор също се използва в комбинация с IF. Когато компютъра стигне до него прекратява действието на цикъла и преминава към оператора записан веднага след цикъла.

Ще разгледаме следната задача – потребителят въвежда 20 числа от клавиатурата, а компютъра ги събира. Ако компютъра въведе 0 цикълът прекъсва, като не се въвеждат

повече числа, а на екрана се отпечатва сумата до момента (Pascal / C++):

```
var i, a, Sum : integer;
begin
  Sum:=0;
  for i:=1 to 20 do
    begin
      readln(a);
      if a = 0 then break;
      Sum:=Sum+a;
    end;
  writeln(Sum);
end.
```

```
…
int i, a, Sum;
Sum=0;
for (i=1; i<=20; i++) {
   cin>>a;
   if (a==0) break;
   Sum=Sum+a;
}
cout<<Sum<<endl;
…
```

Дефинирани и други оператори за прекъсване – exit; и halt; в Pascal за изход от подпрограма/програма; exit(); и abort(); в C++ за изход от програма. С тяхна помощ можете да напишете например програма, която многократно извежда меню и изпълнява избраното от потребителя действие, докато той не избере „Изход от програмата”.

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Създайте програма, която изчислява:

   а) произвеението 1.2.3…N;

   б) степента x^n, n≥0;

   в) сумата на нечетните числа от 1 до 100;

   г) произведението на четните числа от 10 до 20;

   д) броя на числата от 25 до 85, кратни на 5;

   е) броя на броя на простите числа от 1 до 50;

   ж) сумата $S = 101 + 97 + 93 + \cdots + 11$;

   з) произведението $P = \frac{3}{1.2} . \frac{4}{2.3} .... \frac{n+1}{(n-1)n}$;

2. Създайте програма, която намира произведението на числата от 1 до N, които се делят на 3 (N се въвежда от клавиатурата).

3. Създайте програма, която намира произведението на нечетните числата от 1 до N, и проверява дали то се дели на 2, 3 и 5 (N се въвежда от клавиатурата).

4. Създайте програма, която по въведено цяло число отпечатва на екрана всичките му делители (положителни и отрицателни).

5. Да се пресметне и изведе сумата на въведени от клавиатурата числа, до въвеждане на 0.

6. Да се отпечатват квадратите на числа, въвеждани от клавиатурата до въвеждане на 0.

7. Създайте примерно меню, което се повтаря, докато потребителя не въведе съответния клавиш за край според менюто, изведено на екрана. Кой вид цикъл е най-подходящ?

8. По подобие на таблицата за умножение отпечатайте таблици за сбор, разлика и частно на две числа в интервала [1; 10].

9. По въведена сума в лв. да се направи разбивка с минимален брой банкноти.

10. Създайте програма, която отпечатва:

а)	б)	в)	г)
1 1 1 1 1	1 2 3 4 5	1	1 1 1 1 1
2 2 2 2 2	1 2 3 4 5	1 2	2 2 2 2
3 3 3 3 3	1 2 3 4 5	1 2 3	3 3 3
4 4 4 4 4	1 2 3 4 5	1 2 3 4	4 4
5 5 5 5 5	1 2 3 4 5	1 2 3 4 5	5

11. Спрямо горните задачи определете в кои случаи кои видове цикли са най-подходящи.

Тема 7. Масиви.

Много алгоритми изискват няколко променливи от един и същ тип, с които да се извършват едни и същи операции. За такива случаи езиците за програмиране предоставят възможност последователност от еднотипни полета да се назове с едно име, като *елементите* на последователността се различават по поредния си номер в нея, наричан *индекс*. Такава последователност се нарича *масив*. При декларирането на масива се посочва името, типа и броя или вида на индексацията на елементите му, а компютъра заделя необходимото количество памет, така че всички елементи на масива да се разположат последователно в паметта. В някои езици индексите задължително започват от нула, в други е позволено за индекс да се използват поредни стойности от произволен дискретен (изброим) тип, например символи.

С елементите от масив може да се извърщват всички действия, които бихте извършили с нормална променлива. Достъпът до конкретен елемент от масива се осъществява като се посочи името на масива и индекса на елемента, записан в скоби веднага след името. Въвеждането от клавиатурата или извеждането на екрана на стойностите на всички елементи на даден масив се организира в цикъл. Прието е за целта да се използва цикъл FOR, в който управляващата променлива става равна последователно на всички индекси в масива. По този начин спестяваме многократното изписване в програмата на едни и същи оператори.

Употребата на масиви обикновено е свързана с търсене или сортиране. При търсенето всички елементи на масива се обхождат последователно от компютъра като се проверява дали отговарят на даден критерий и ако е така се извършва предвиденото за конкретната задача действие. Сортирането на масива представлява пренареждане на елементите му по големина или по азбучен ред (когато са от символен тип) в нарастващ или намаляващ ред.

7.1 Работа с масиви.

Деклариране на масив

Език	Деклариране	Примери
Pascal	*име* : array [*нач.инд* . . *кр.инд*] of *тип*; Индексите може да са от произволен дискретен тип: 10..20, ‘a’..’z’ и т.н.	Names : array [1 .. 10] of string; age : array [1 .. 10] of integer;
C++	*тип име*[*бр.елементи*]; Индексирането започва от 0 – за 10 елемента например, индексите са от 0 до 9	char names[10][5]; //10 елемента, всяки до 5 знака int age[10];

Достъп до елементите на масив

Pascal	C++
age[4] – четвърти елемент read(age[5]); – вход на 5-ти елемент write(age[1]); – извеждане на 1-ви елемент	age[3] – четвърти елемент cin>>age[4]; – вход на 5-ти елемент cout<<age[0]; – извеждане на 1-ви елемент

Въвеждане и извеждане на масив

Език	Вход	Изход

Pascal	for i:=1 to 10 do
	  readln(age[i]);
		for i:=1 to 10 do
		  writeln(age[i]);

C++	for (i=0; i<=9; i++)
	  cin>>age[i];
		for (i=0; i<=9; i++)
		  cout<<age[i];

Поради използването на управляваща променлива, тя трябва да бъде декларирана преди използването й в цикъла.

7.2 Основни задачи върху масиви.

Почти всички задачи върху масиви са свързани с обхождане на масива. Подобно на въвеждането и извеждането, обхождането също се извършва с цикъл. За следващите задачи ще приемем, че сме въвели масива age с 10 елемента от тип цяло число, които в случая играят ролята на възрастта на 10 човека. В примерите по-долу ще видите как става самото обхождане. За да тествате решенията, трябва да допишете останалата част от кода.

Търсене с отпечатване и броене по критерий

Задача	Да се отпечатат индексите на всички, които са над 30 години	Да се изброят всички, които са по-млади от 20 години

Pascal	for i:=1 to 10 do
	  if age[i] > 30 then writeln(i);
		Broi:=0;
		for i:=1 to 10 do
		  if age[i] < 20 then Broi:=Broi+1;
		writeln(Broi);

C++	for (i=0; i<=9; i++)
	  if (age[i] > 30) cout<<i<<endl;
		Broi=0;
		for (i=0; i<=9; i++)
		  if (age[i] < 20) Broi++;
		cout<<Broi<<endl;

Сума и средно аритметично на елементите на масив

Задача	Да се отпечата сбора от елементите на масива	Да се отпечата средната възраст

Pascal	Sum:=0;
	for i:=1 to 10 do
	  Sum:=Sum+age[i];
	writeln(Sum);
		Sum:=0;
		for i:=1 to 10 do
		  Sum:=Sum+age[i];
		writeln(Sum/10);

C++	Sum=0;
	for (i=0; i<=9; i++)
	  Sum=Sum+age[i];
	cout<<Sum<<endl;
		Sum=0;
		for (i=0; i<=9; i++)
		  Sum=Sum+age[i];
		cout<<(float)Sum/10<<endl;

Намиране на минимален/максимален елемент

<table>
<tr><td>Задача</td><td>Да се отпечатат годините на най-младия</td><td rowspan="3">Алгоритъм:
1. В началото компютъра приема, че най-малкия елемент на масива е първия.
2. Последователно се сравнява стойността, на променливата MIN със всеки елемент на масива от втория до последния и всеки път когато се открие по-малка стойност, тя се присвоява на MIN.

Ако задачата е за максимум, алгоритъма е същия, но се обръща знака за сравнение и името на променливата – MAX. Така всеки път, когато се открие по-голяма стойност, тя се присвоява на MAX</td></tr>
<tr><td>Pascal</td><td>Min:=age[1];
for i:=2 to 10 do
  if age[i]<Min then Min:=age[i];
writeln(Min);</td></tr>
<tr><td>C++</td><td>Min=age[0];
for (i=1; i<=9; i++)
  if (age[i]<Min) Min=age[i];
cout<<Min<<endl;</td></tr>
</table>

7.3 Сортиране на масив.

Една от основните задачи върху масив е подреждането на елементите му в нарастващ или намаляващ ред. Тук ще разгледаме сортирането на масив в нарастващ ред, като за целта използваме метода на мехурчето.

Ако наблюдавате съд в вода или чаша с безалкохолно например, можете да забележите че вътре постепенно се образуват мехурчета, които изплуват на повърхността. Колкото мехурчето е по-голямо, толкова по-бързо изплува то нагоре. От тук идва и името на алгоритъма, а той работи така:

1. Броят на обхожданията на масива е с единица по-малък от броя на елементите му.

2. При всяко обхождане се сравняват последователно всеки два съседни елемента и ако по-малкия от двата е втори подред, местата им се сменят.

3. За реализацията се използват два вложени цикъла, като външния брои колко пъти е обходен масива, а вътрешния извършва самото обхождане и сравняване на елементите.

Да видим как би изглеждало сортирането на масив с пет елемента (4 9 12 6 3):

Първа итерация:
4 9 12 6 3 сравняват се 1-ви и 2-ри елемент – няма нужда от размяна
4 9 12 6 3 сравняват се 2-ри и 3-ти елемент – няма нужда от размяна
4 9 **12 6** 3 сравняват се 3-ти и 4-ти елемент – елементите се разменят
4 9 6 **12 3** сравняват се 4-ти и 5-ти елемент – елементите се разменят
Резултат:
4 9 6 3 12 масивът след приключване на първата итерация

Втора итерация:
4 9 6 3 12
4 **9 6** 3 12
4 6 **9 3** 12
4 6 3 9 12
Резултат:
4 6 3 9 12

Трета итерация:
4 6 3 9 12
4 **6 3** 9 12
4 3 6 9 12
4 3 6 9 12
Резултат:
4 3 6 9 12

Четвърта итерация:
4 3 6 9 12
3 4 6 9 12
3 4 6 9 12
3 4 6 9 12
Резултат:
3 4 6 9 12

Реализация на алгоритъма за Pascal и C++ в този ред:

```
for i:=1 to 4 do
for j:=1 to 4 do
 if A[j+1] < A[j] then
   begin
       Temp: =A[j];
       A[j]:= A[j+1];
       A[j+1]:=Temp;
   end;
```

```
for (i=0; i<=3; i++)
  for (j=0; i<=3; i++)
    if (A[j+1] < A[j]) {
       Temp = A[j];
       A[j] = A[j+1];
       A[j+1] = Temp;
     }
```

Лошото при този метод е, че масива може да се окаже подреден преди последната итерация, дори още в началото, но програмата ще изпълни всички повторения, а това при голям брой елементи отнема повече време.

Възможно е метода да се подобри, като се следи дали при последното обхождане е била извършена размяна на елементи и ако не (т.е. масива вече е подреден) – цикълът да се прекрати. Помислете как може да се реализира това.

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Да се изведат на екрана 5-тия и 10-тия елемент на масив с 12 елемента.
2. Да се изведат на екрана всички елементи на масив от 20 елемента с четни индекси.
3. Създайте програма за намиране на сумата на елементите на масив от 20 числа.
4. Създайте програма за намиране на най-големия елемент на масив от 10 числа.
5. Създайте програма, която по въведен масив от 10 цели числа намира броя на тези от тях, които са по-големи от 0.
6. Да се изброи колко пъти се среща числото 3 в даден масив от 30 елемента.
7. Да се пресметне и изведе средното аритметично на елементите на масив с 50 числа.
8. Да се изведат елементите на масив, които са четни и отрицателни.
9. Да се изведат на екрана всички елементи на масив, чиято стойност е между 10 и 20.
10. Да се изведе броя на елементите на масив от 15 числа, които се делят на 2, 3 и 5.
11. Създайте програма, която по въведен масив от 10 цели числа намира произведението на тези от тях, които са по-големи от 0.
12. Да се изброят елементите в масив, чиято абсолютна стойност е по-голяма от 30.
13. По въведени 10 цели числа в масив да се намери сумата на тези от тях, които са нечетни.
14. Създайте програма, която по въведен масив от 10 цели числа намира броя на тези от тях, които са отрицателни и сумата на останалите.
15. Създайте програма, която намира и извежда сумата на елементите с четни индекси и произведението на тези с нечетни индекси.
16. Да се пресметнат и изведат броя и сумата на елементите на масив, чиято стойност е в интервала [L; R]. Числата L и R се въвеждат от клавиатурата.
17. Да се състави програма, която събира числата в масив, докато те са нарастващи и в момента, в който достигне до число, което е по-малко от предходното, извежда получената до момента сума на екрана.
18. Създайте програма, отпечатваща абсолютната стойност на елементите на масив.

19. Създайте програма, която по въведен масив от 5 цели числа отпечатва на екрана 5 реда със звездички, като броя на звездичките на всеки ред е равен на поредното число от масива.

20. Да се изведат индексите на всички елементи на масив, които имат повече от 4 делителя.

21. Да се изведат на екрана елементите на масив от 25 числа, разположени на 5 реда по 5 числа на ред.

22. Температурите за месец с 30 дни се въвеждат в масив. Да се опеделят: минималната, максималната или средната температура за месеца, според избора на потребителя. За целта да се изведе подходящо меню. В случай, че потребителя е избрал изчисление на средната температура, да се изведе и броя на дните с температури над средната.

23. Създайте програма, която обръща реда на елементите в масив (т.е. ако въведения масив е 8 1 4 7 2, в резултат той да стане 2 7 4 1 8). Броя на елементите на масива да се определя от потребителя и програмата да се организира така, че да работи както за четен, така и за нечетен брой елементи.

24. Създайте програма, която изчислява средноаритметичното на елементите на масив, които са четни.

25. Създайте програма, която подрежда редица от 20 числа в намаляващ ред.

26. Опитайте да измислите алгоритъм за подреждане на масив, различен от „мехурчето”.

Тема 8. Създаване на графики.

8.1 Общи положения.

Във всеки от описаните тук езици има реализирани функции за изчертаване на набор от геометрични фигури, с чиято помощ, в случай на нужда могат да се чертаят и по-сложни такива или дори да се реализира някаква анимация. Преди да преминем към описанието на самите функции, ще отбележим няколко основни момента.

Графичен режим

Преди да можете изобщо да изведете графичен елемент на екрана, трябва да накарата компютъра да премине от текстов във графичен режим. В текстов режим компютъра определя позициите на екрана според това на кой ред и кой символ от реда се намира маркера (в повечето случаи екрана има 25 реда с по 80 символа). В графичен режим позицията се определя като координати, зададени в пиксели, като максималните разделителна способност за компилаторите под DOS е 640x480 точки.

За целта трябва да извикате функцията initgraph, като укажете на компютъра къде да търси драйвера (променлива Gd) за графичния режим (т.е. къде се намира папка BGI), а той сам намира най-подходящия такъв, както и съответстващия му режим на работа (променлива Gm). Освен това трябва да се укаже и съответния библиотечен файл, съдържащ подпрограмите за създаване на графични елементи. Ето как изглежда това в Pascal / C++ (указания път в initgraph е примерен):

```
uses Graph;
var Gd, Gm : integer;
  begin
    Gd:=Detect;
    InitGraph(Gd, Gm, 'D:\pascal\bgi')
…
```

```
#include<graphics.h>
…
void main () {
int gd=DETECT, gm;
initgraph(&gd, &gm, "D:\\cpp\\bgi");
…
```

Координатна система

Ще приемем, че работим с максималната разделителна способност – 640x480.

Началото на координатната система при компютрите се явява горния ляв ъгъл на екрана. Стойностите за координатите x и y за всяка точка измерват нейната позиция съответно по хоризонталата и вертикалата. Колкото по-надясно се намира една точка, толкова по голяма стойност има нейната x координата. Колкото по-надолу се намира, толкова по-голяма стойност има нейната y координата (за разлика от математиката, където движейки се надолу y намалява).

Казано по друг начин, *x* – координатата показва разстоянието в пиксели от левия край на екрана до точката, а *y* – разстоянието в пиксели от горния край на екрана до нея. Възможните стойности са съответно в интервала [0, 639] за x, и [0, 479] за y.

Цветова палитра

Основните цветове в графичния режим са 16, и са с кодове съответно от 0 (черно), до 15 (бяло). Използването на даден цвят става като се използва функцията за смяна на цвета за съответния език, в която като параметър се посочва кода на желания цвят:

Pascal:	setcolor(*номер_на_цвят*);	за графични елементи (в модул Graph)
	textcolor(*номер_на_цвят);*	за текст (в модул Crt)
C++:	setcolor(*номер_на_цвят*);	за графични елементи (в модул graphics.h)
	textcolor(*номер_на_цвят);*	за текст (в модул conio.h)

Цветовете могат да се изписват и с техните имена, като в C++ имената трябва да се изписват изцяло с главни букви. В следващата таблица са описани цветовете и техните кодове:

№ на цвят	цвят	име	№ на цвят	цвят	име
0	черен	Black	8	тъмно сив	DarkGray
1	тъмно син	Blue	9	син	LightBlue
2	зелен	Green	10	светло зелен	LightGreen
3	светло син	Cyan	11	бледо син	LightCyan
4	червен	Red	12	бледо червен	LightRed
5	цикламен	Magenta	13	розов	LightMagenta
6	оранжев/кафяв	Brown	14	жълт	Yellow
7	светло сив	LightGray	15	бял	White

Графични елементи

Основните графични елементи са линия и окръжност. За линия като параметри се задават координатите на двете й крайни точки. Например, ако зададем като координати числата 0, 0, 639, 479, ще се изчертае линия от горния ляв до долния десен ъгъл на екрана, т.е. диагонала. Окръжност се задава с координатите на центъра и големината на радиуса й в пиксели. Например, окръжност с координати на центъра 320, 240 и радиус 240 ще бъде изчертана в средата на екрана и ще се допира до горния и долния ръб на екрана.

В някои езици са дефинирани функции за чертане на правоъгълници. В Basic за целта се използва отново функцията за линия. Правоъгълник се задава, като се посочат координатите на два срещуположни върха в правоъгълника. В случай, че x или y координатите на върховете са равни се получава съответно хоризонтална или вертикална линия.

Предвидени са и функции за изчертаване на елипси или части от елипси/окръжности. В Basic тези възможности са добавени към функцията за окръжност. Елипсата може да бъде разтегната по хоризонталата или вертикалата, но не и под наклон. За целта се задават два различни радиуса – радиус по x и радиус спрямо y, с което се определя разтягането по съответната ос. В случай, че двата радиуса са равни се получава окръжност. За изчертаване на част от окръжност или елипса, трябва да се зададат начален и краен ъгъл, като посоката на изчертаване е обратна на часовниковата стрелка (от математиката). Например при начален ъгъл 180 и краен 360 ще се изчертае долната половина (така можем да начертаем усмивка на човече). При начален ъгъл 0 и краен 360 градуса се извежда цялата окръжност / елипса.

Отделните фигури, които се получават могат да бъдат оцветявани по желание с избран от нас цвят и тип на запълването. Има значение и реда, в който се изчертават отделните фигури – ако се получи застъпване новата фигура припокрива вече изведената.

8.2 Графични функции.

Линии и незапълнени фигури:

line(x1, y1, x2, y2); – чертае линия от точка до точка

lineto(x, y); – чертае линия от текущата позиция до дадена точка

moveto(x, y); – премества ни в дадена точка. Заедно с lineto може да се начертае произволен многоъгълник или фигура, изградена само от линии

rectangle(x1, y1, x2, y2); – чертае правоъгълник със зададени координати на два срещуположни върха. Ако двете x или y координати са равни се получава линия

circle(x, y, r); – чертае окръжност

arc(x, y, *начален_ъгъл*, *краен_ъгъл*, r); – чертае част от окръжност

ellipse(x, y, *начален_ъгъл*, *краен_ъгъл*, rx, ry); – чертае цяла елипса (при зададени начален и краен ъгъл – 0 и 360) или част от елипса

Запълнени фигури:

bar(x1, y1, x2, y2); – чертае запълнен правоъгълник

bar3d(x1, y1, x2, y2, *дълбочина*, *top*); – чертае паралелепипед (блокче) със запълнена предна част, значението на top е – 1(има)/ 0(няма) горна част

fillellipse(x, y, rx, ry); – чертае запълнена елипса

pieslice(x, y, *начален_ъгъл*, *краен_ъгъл*, r); – чертае запълнена част от кръг

sector(x, y, *начален_ъгъл*, *краен_ъгъл*, rx, ry); – чертае запълнена част от елипса

Цветове и запълвания:

setcolor(*цвят*); – задава цвят за чертане на линии и фигури

setbkcolor(*цвят*); – задава цвят на фона

setfillstyle(*стил*, *цвят*); – задава стил на оцветяване за запълнените фигури и за floodfill. Стойностите за стил са от 1 до 12 и определят различни начини на запълване.

setlinestyle(*стил*, *потребителски_стил*, *дебелина*); – задава стил на изчертаване на линиите. *Стил* определя вида на линията и възможните стойности са от 0 до 4. Дебелината може да е 1 или 3 пиксела. С втория параметър може да се задава потребителски вид на линията, в случай, че за стил е въведен код 4. За целта се използва 16-битова стойност, т.е. числа в интервала от 0 до 65 535.

floodfill(x, y, *цвят*); – оцветява произволна област, оградена от едноцветен контур, включително и незапълнени фигури, започвайки от избрана точка. *Цвят* е цвета на контура, до който ще се запълва. Ако контура с този цвят не е затворен се боядисва всичко, понеже боята „изтича" през незатворената част.

Няколко правила

1. Искаме ли да има запълване, задължително ползваме setfillstyle .
2. Всяка запълнена фигура може да се представи като незапълнена, впоследствие оцветена с floofdill т.е.

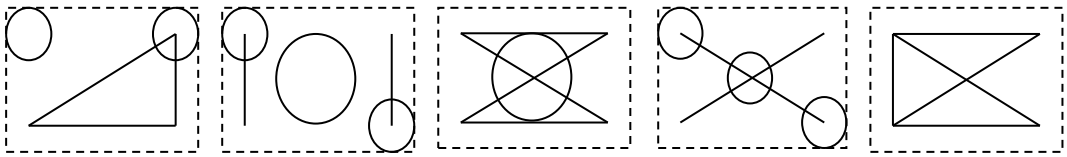
Запълнена фигура + setfillstyle = незапълнена фигура + setfillstyle + floodfill

3. Реда на изчертаване има значение: при препокриване изцяло видима е само последната изчертана фигура.

========== **ВЪПРОСИ И ЗАДАЧИ** ==========

1. Начертайте 5 концентрични окръжности с разлика в радиусите 1 (2, 5 или избран от Вас брой) пиксела. Опитайте да направите това в цикъл FOR.
2. Модифицирайте горната задача така, че да извежда всяка окръжност с различен цвят.
3. Опитайте да направите същото, но с правоъгълници.
4. Начертайте следните фигури:

5. Начертайте следните елементи с избрани от Вас цветове (с пунктир е означен екрана):



6. Опитайте да начертаете следните фигури: