ШАГ компьютерная академия

# ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

# Урок №2

## Програмиране на езика C

**Съдържание**

# 1. Понятието оператор

В предишния урок вие се запознахте с понятията променлива и типове данни. Освен това, в примерите на урока, също така и за домашна работа ние с вас провеждахме над променливите определени действия, тоест сме оперирали с данни. Напълно е очевидно, че думите оператор и опериране имат еднакво значение, следователно, следвайки проста логика:

Оператор — конструкция на езика позволяваща да се провеждат различни действия над данните, довеждащи към определен резултат.

Прието е всички оператори да бъдат подразделяни на групи по вида на действията им. Например, аритметични операции- операции, позволяващи произвеждането на аритметични действия върху данните (поставяне, изваждане и т.н.). За всичките подробни групи в езика С ще говорим по- нататък. За момента е необходимо да обсъдим по-мащабна класификация на всички оператори, приета извън зависимостта на влиянието от съдържанието на променливите. И така, всички оператори се делят на:

**1. Унарные** — оператори на които им е необходим само един операнд (данни над които се провежда действие). С пример за унарен оператор вие сте се запознавали по време на училищната математика – унарен минус, който позволява превръщането на число в отрицателно (3 и –3), или положително (–(–3)). Т. е. Общия синтаксис на унарния оператор е следния:

```
оператор операнд;          или          операнд оператор;
```

**2. Бинарни** — оператори на които им е необходимо два операнда от ляво и от дясно от оператора. Такива оператори вие знаете много- това са +, -, * и т.н. Техния общ синтаксис може да бъде изобразен по следния начин:

```
операнд оператор операнд;
```

**3. Тернарные** — оператори на които им е необходимо три операнда. В езика за програмиране С такъв оператор е само един и с неговия синтаксис ние ще се запознаем малко по- късно..

**Приоритет**

Всички оператори имат приоритет. Долу са приведени оператори в съответствие с приоритетите. По-задълбочено ние ще се запознаем с няколко от тях в днешния урок а за другите ще научим по-напред в процеса на обучение. Естествено, дадената таблица не са представени всички оператори на езика, а за момента най-актуалните за нас.

| Символно обозначение на операциите | |
|---|---|
| Висок приоритет | Висок приоритет |
| () [] . -> | ^ |
| ! *(ун) – (ун) ~ ++ — | \| |
| % * / | && |
| + – | \|\| |
| << >> | ?: |
| < > <= >= | = += –= *= /= %= &= \|= ^= >>= <<= |
| != == | |
| & | Нисък приоритет |

Сега, когато е заложен фундамента от знания в областта на операторите, вие можете да продължите към по-детайлното изучаване на последните, а именно към следващия раздел на урока.

# 2. Аритметически операции с числа

**Добре забравеното старо…**

И така, да започване. Както беше вече отбелязано по-рано, аритметическите операции – това са операции които позволяват да се произвеждат аритметически действия над данните. Повечето от тях вие ги познавате от детството си и въпреки това, нека да систематизираме нашите знания с помощта на таблицата представена долу.

| Име на операция та | Символ използван за обозначение в еизка С. | Кратко описание. Пример. |
|---|---|---|
| Събиране | + | Събира две значения заедно, резултатът е сумата на операндите 5+18 резултат 23 |
| Изваждане | – | Изважда значение, намиращо се от дясно от значението намиращо се от ляво на оператора. Резултат – разликата на операторите 20-15 = 5 |
| Умножение | * | Умножава две значения, резултатът е произведението от операндите 5*10 = 50 |
| Деление | / | Дели значение намиращо се от ляво на значението намиращо се от дясно на оператора. Например: 20/4 = 5 |
| Деление по модул | % | Резултатът от тази операция е остатъка от целочисленото деление, например, ако делим 11 на 3, то цяла част у нас се получава 3, (след като 3*3=9), остатък ще бъде 2, това число и ще бъде резултата от деление по модул:: 11/3 3 цели 2 остатък 11%3 = 2 (остатък) |

**Забележка:**

1. Операция за деление по модули, може да бъде използвана само при целочислени данни. Опит за нарушаване на даденото правило ще доведе до грешка на етапа за компилиране.

2. Ако по малко число ще се разделя на по-голямо с помощта на %, тогава в резултата ще бъде само по-малкото число. 3%10=3

3. Делението по модул на нула е забранено, това ще доведе до некоректна работа на програмата в етапа на изпълнението й.

**Инкремент и декремент.**

Всичките гореописани операции са бинарни, обаче съществуват и унарни аритметични операции, такива операции в училищния курс няма въпреки че са изключително елементарни:

**1. Инкремент** — обозначава се с конструкцията ++. Даденият оператор увеличава съдържанието на която и да е променлива с една единица и презаписва значението на променливата. Например:

```
int a=8;
cout<<a; // на екрана е
числото 8  a++;
cout<<a; // на екрана е числото 9
```

**2. Декремент** — обозначава се с конструкцията —. Даденият оператор увеличава съдържанието на която и да е единица и презаписва значението на променливата

```
int a=8;
cout<<a; // на екрана е
числото 8  a--;
cout<<a; // на екрана е числото 7
```

Много лесно, нали така?! Такива изрази могат да бъдат представени и така: a=a+1 или a=a-1. Трябва да се отбележи, че за литералите не се използват нито инкременти нито декременти, така както е съвършено не логично да се прави както в следващия израз 5=5+1. Това е очевидна грешка. Със това обаче няма да приключим запознанството ни с инкрементите и декрементите. В предишния раздел на урока ние изяснихме, че синтаксиса на унарния оператор може да бъде не само такъв,

```
операнд оператор;
```

но и такъв

```
оператор операнд;
```

Такива форми на запис се наричат постфикс, (оператора се разполага след значението). И инкремента, и декремента имат двете форми. Хайде да разберем какви са различията между формите и в какви случаи тези различия имат значение.

## Пример 1.

```
int a=8;
cout<<a; // на екрана е
числото 8  a++;
cout<<a; // на екрана е числото 9
++a;
cout<<a; // на екрана е числото 10
```

В дадения пример няма разлика между префиксната и постфиксната форма. Както в първия така и във втория случай, значението на променливата a просто се увеличава с единица. Смисъл да се използват различни форми на оператора се появява само тогава, когато в

реда освен самия оператор има и още някаква команда.

## Пример 2.

```
int a=8;
cout<<++a; // на екрана число 9
cout<<a++; // на екрана число 9
cout<<a; // на екрана число 10
```

Преди да разгледаме на части примера, нека да установим три правила:

1.

Принципа за изпълнение на командите в езика С не е еднозначен. Затова по-долу е приведена таблица за направленията на действията на някои от операторите:

2. Ако освен постфиксната форма на инкремента или декремента, в реда има още някаква команда, то първоначално ще се изпълни тази команда и само тогава инкремента или декремента независимо от разположението на командите в реда.

3. Ако освен префиксната форма на инкремента или декремента в реда има още каквато и да е команда, то всичките команди в реда ще се изпълняват от дясно на ляво съгласно приоритета на операторите.

| ОПЕРАТОР | НАПРАВЛЕНИЕ |
|---|---|
| () [] . -> | ОТ ЛЯВО НА ДЯСНО |
| * / % + – | |
| << >> & ^ \| | |
| << = >> = ==! = | |
| && \|\| | |
| Унарен – унарен + ! ++ — | ОТ ДЯСНО НА ЛЯВО |
| ?: | |
| + = – = / = * = % = & = ^ = \| = | |

**Сега по-подробно за примера:**

■ Изначално значението на променливата е равно на числото 8.

■ Командата cout<<++a; съдържа префиксна форма на оператора инкремент, следователно използвайки третото правило, описано горе, ние първо увеличаваме значението на променливата a с единица а след това го показваме на екрана с помощта на командата cout<<.

■ Командата cout<<a++; съдържа постфиксна форма на оператора инкремент, следователно, използвайки второто правило, описано горе, ние първо показваме значението на променливата (все още 9) на екрана с помощта на командата cout<<, а след това увеличаваме значението на променливата a с единица..

■ При изпълнението на следващата команда cout<<a; ще бъде показано изменено (увеличено) значение, тоест число 10.

Изхождайки от предишните теми на дадения раздел на урока ние с вас вече знаем как да опростим неудобната и „некрасива“ запис на x=x+1 или x=x–1, превръщайки я в x++,

или x—. Но по такъв начин ние можем да увеличим и смалим значението на променливата само с единица, а как да бъде с другите числа? Например, как да упростим записа:

X=X+12;

В дадения случай, съществува отново просто решение — използвайки така наречените комбинирани оператори или съкратени аритметични форми. Те изглеждат по следния начин:

| Име на формата | Комбинация | Стандартна запис | Съкратена запис |
|---|---|---|---|
| Присвояване с умножение | *= | A=A*N | A*=N |
| Присвояване с деление | /= | A=A/N | A/=N |
| Присвояване с деление по модул | %= | A=A%N | A%=N |
| Присваивание с изваждане | – = | A=A–N | A– =N |
| Присваивание със събиране | += | A=A+N | A+=N |

Препоръчваме ви за напред да използвате съкратените форми, защото това се смята не само за хубав тон в програмирането, но и значително повишава читаемостта на програмния код. Освен това в няколко източника се упоменава, че съкратената форма се обработва от компютъра по-бързо, повишавайки скоростта на изпълнение на програмата. Сега е точното време да се убедим във всичко гореописано на практика. Защото както се казва е по-добре един път да видиш отколкото сто пъти да чуеш. Вие вече можете да създавате проекти и да добавяте в

тях файлове, общо взето точно това от вас сега се изисква. Напред са представени няколко програми, които вие трябва да напишете за да видите аритметическите операции на практика. Да започнем с проекта наречен Gam

# 3. Прилагане на аритметичните операции

## Пример №1. Игра.

```
// примитивна игра за деца
#include <iostream>
using namespace std;
void main()
{
    int buddies; // количество пирати преди битката
    int afterBattle; //количество пирати след битката

    // Вие сте пират. Колко човека има във вашия отбор ако не броим вас?
    cout<<"You the pirate. How many the person in your command, without you?\n\n";
    cin>>buddies;

    //Внезапно ви нападат 10 мускетари.
    cout<<"Suddenly you are attacked by 10 musketeers \n\n";

    //10 мускетари и 10 пирати загиват в схватката.
    cout<<"10 musketeers and 10 pirates perish in fight.\n\n";

    //пребройване на живите
    afterBattle=1+buddies-10;

    // Останали са ... пирати
    cout<<»Remains only "<<afterBattle<<» pirates\n\n";

    //Печалбата от убитите пирати е 107 златни монети
    cout<<"The condition killed totals 107 gold coins \n\n";

     //По ...  монети за всеки
    cout<<"It on "<<(107/afterBattle)<<"coins on everyone";

    //Пиратите започват голяма битка заради останалите
    cout<<"Pirates arrange greater fight because of remained\n\n";
    //... монети
    cout<<(107%afterBattle)<<"coins \n\n";
}
```

В дадения пример се използва правилото за деление на цялото на цяло — при такова деление дробната част дори ако трябва да я има – се отрязва. По-подробно за това ще бъде разказано в раздела на урока – „Преобразуване на типовете“. В израза (107/afterBattle) – ние ще разберем колко монети ще получи всеки пират, ако бъдат разделени по равно. Освен това оператора за деление по модул ни помага да изясним колко монети ще останат, които ще е невъзможно да се разделят, тоест ние ще получим остатъка от делението на 107 на количеството оцелели пирати. Това са всички особености на примера.

**Пример №2. Окръжност.**

В дадения пример ще бъде демонстрирано използването на аритметическите оператори в програмите, произвеждащи математически изчисления. Името на проекта **Circle**.

Ние ще се убедим че познаването на аритметичните оператори дава възможност за решаване на прости задачи. Обаче не е достатъчно умението за използване на операторите, необходимо е да се разбира какъв ще бъде резултата от тяхното използване. За това ще говорим в следващия раздел.

```
// програма изчисляваща параметрите на окръжност
#include <iostream>
using namespace std;
void main()
{
      const float PI=3.141592;//обозначение на константи - числа пи

      //обявяване на променливи за съхраняване на
      параметри float radius, circumference, area;

      // покана за въвеждане на радиус
      cout<<"Welcome to program of work with rounds\n\n";
      cout<<"Put the radius from rounds\n\n";
      cin>>radius;
      cout<<"\n\n";

      area=PI*radius*radius; // пресмятане на площта на кръга
      circumference=PI*(radius*2); // пресмятане на дължината
      на окръжността

      // извеждане на резултат
      cout<<"Square of round: "<<area<<"\n\n";
      cout<<"length of round: "<<circumference<<"\n\n";
      cout<<"THANKS!!!  BYE!!!\n\n";

}
```

# 4. Преобразуване на типовете

Когато правим нещо, за нас е много важно да знаем какъв ще бъде резултата. Прекрасно ясно е че от продуктите необходими за приготвянето на пилешка супа, надали ще можем да направим торта с крем маскарпоне. Следователно резултата е пряко зависим със съставните части. Същото важи и за променливите. Ако използваме две числа от типа int в операция то е напълно ясно че резултата също ще бъде от типа int. Какво ли ще се случи ако данните са от различен тип? Именно за това ще говорим в текущия раздел на този урок.

И така, преди всичко, нека да разберем как типовете данни взаимодействат една с друга. Съществува така наречена йерархия на типовете, разпределени по старшинство. За да разбираме преобразуването на типовете е необходимо винаги да се помни порядъка на типовете на тази йерархия.

```
bool, char, short-int-unsigned int-long-unsigned
long-float-double-long double
```

Независимо от това че някои типове имат еднакъв размер, в тях се размества различен диапазон от значения, например, unsigned int  за разлика от int може да размести в себе си два пъти повече положителни значения, и затова е по-старши по йерархия независимо че и двата типа имат размер от 4 байта. Освен това, трябва да се отбележи, много важна особеност, отразена тук, ако в преобразуването на типовете участват

такива типове като: bool,char,short,

те автоматично се преобразуват в int.

Сега, нека да разгледаме различните класификации на преобразуването.

**Класификация по диапазона на съдържащите се значения.**

Всичките преобразувания могат да бъдат разделени на две групи относителни от местоположението им в йерархията на типовете участващи в преобразуването .

**1. Смаляващо преобразуване** — при такъв начин на преобразуване – голям тип данни в йерархията се преобразува в по-малък, безусловно, в този случай може да получи загуба на данни, затова със смаляващото преобразуване трябва да бъдем внимателни. Например:

```
int A=23.5;
cout<<A; // на екрана 23
```

**2. Разширяващо преобразуване.** Дадения вид преобразуване, води към така нареченото разширяване на типа данни от малкия диапазон на значения към големия диапазон. За пример се предлага такава ситуация.

```
unsigned int a=3000000000;
cout<<a; // на екрана 3000000000
```

В дадения случай 3000000000 — това е литерал от типа int, който благополучно се разширява до unsigned int, което ни позволява да видим на екрана именно 3000000000, а не нещо друго. Тъй като в обикновения int такова число не може да се събере.

**Класификация по начина на осъществяване на преобразуването.**

Независимо от направлението на преобразуването, то може да се осъществи по един от двата начина.

**1. Неявно преобразуване.** Всичките гореописани примери се отнасят към този тип преобразуване. Такъв вид преобразуване също така наричат автоматически след като то произлиза автоматично без намесата на програмиста, с други думи, ние нищо не прави за да се случи.

```
float A=23,5; - double стана float без
каквито и да допълнителни действия
```

**2. Явно преобразуване.** (второ наименование — привеждане на типовете). В дадения случай, преобразуването произлиза от програмиста, тогава когато е необходимо. Нека да разгледаме прост пример на такова действие:

```
double z=37.4;
float y=(int) z;
cout<<z<<"*** "<<y; // на екрана 37.4 ***37
```

(int)z — е ясно смаляващо преобразуване от типа double към типа int. В този ред произлиза разширяващо неявно преобразуване от получения тип int към типа float. Трябва да се запомни, че всяко преобразуване носи временен характер и действа само в пределите на текущия ред. Тоест променлива z както е била double така и ще остане в продължението на цялата програма, докато нейното преобразуване в int е носило само временен характер.

**Преобразуване на типовете при изразите.**

И ето най-накрая стигнахме до това, за което сме говорили в началото на дадения раздел на урока, как да изясним какъв тип ще бъде резултата от някой израз.

```
int I=27;
short S=2;
float F=22.3;
bool B=false;
```

Ползвайки тези променливи ние се готвим да съставим такъв израз:

```
I-F+S*B
```

В променливата на какъв тип данни на нас ни трябва да запишем резултата? Да се реши това е лесно, ако представим израза :

```
int-float+short*bool
```

Напомняме, че short и bool, веднага ще приемат типа int, така че израза ще изглежда така:

```
int-float+int*int,  при этом false станет 0
```

Умножението int на int ще даде, несъмнено, резултат от типа int. А израза float с int, ще бъде с резултат float, така както, тук в играта влиза ново правило:

**Ако в някои от изразите се използват различни типове данни, то резултата ще бъде приведен към по-големия от тези типове.**

И за завършек – изваждане от int типа float, съгласно току що упоменатото правило отново ще даде float.

По такъв начин резултат от израза ще бъде от тип float.

```
float res= I-F+S*B; // 27-22.3+2*0
cout<<res; // на экране число 4.7
```

Сега когато сте запознати с правилото, на вас не ви се налага да разбивате израза на части, достатъчно е само да се намери най-големия тип, именно от него ще произлезе резултата.

Забележка: Бъдете много внимателни, също така, при съчетаването на променливи от еднакъв тип данни. Например, помните, че ако цяло се дели на цяло, то и резлутата ще е цяло. Тоест, int A=3; int B=2; cout<<A/B; // на екрана е 1, защото резултата е — int и дробната част е загубена. cout<<(float)A/B; // на екрана 1.5, защото резултата е - float. **Пример, използващ преобразуване на типовете.**

Сега, нека да закрепим нашите знания на практика. Ще създадем проект и ще напишем следния код.

```
#include <iostream>
using namespace std;
void main(){
     // обявяване на променливи и запитване за
     въвеждане на данни
     float digit;
     cout<<"Enter digit:";
     cin>>digit;

     /* Дори ако потребителя е въвел число с вещественa част, резултата на израза ще
     се запише в int и веществената част ще бъде загубена, разделяйки числото на 100
     ние ще получим количеството стотици в него. */
     int res=digit/100;
     cout<<res<<"hundred in you digit!!!\n\n";
}
```

Сега когато сте разгледали примера, вие сте се убедили, че с помощта на преобразуването на типовете може не само да се организира временен преход от единия тип в друг, но и да се реши проста логическа задача. Следователно е необходимо да се отнесете към тази тема с внимание Разбирането на преобразуването в бъдеще ще ви помогне не само да решавате интересни задачи, но и да избягвате ненужни грешки.

## Унифицирана инициализация

В C++ 11 е добавен механизъм за унифицирана инициализация, който позволява да се зададе значение от различни програмни конструкции (променливи, масиви, обекти) по еднообразен начин. Да разгледаме в примера инициализация на променливите.

```
int a = {11};  // В a се записва 11 int b{33};
               // В b се записва значение 33
```

За да придадем значение на променливите ние използваме **{}**. Както се вижда от примера, това може да се направи по два начина. Такава форма на инициализация се нарича **инициализация по списък**.

## Смаляване и инициализация по списък

Какво ще се случи при изпълнението на кода?

```
int x = 2.88;
cout<<x; // на екрана се изобразява 2
```

Както вече знаете от изучения материал в дадения пример произтича неявно смаляващо преобразуване, тъй като присвояваме на променливата x от цял тип значение на типа **double**. Обаче, ако използваме **инициализация по списък** компилатора ще генерира грешка на етапа за компилиране, защото тази форма на инициализация защитава от смаляване. Тя не дава да се запише значение от голям размер в тип, който не подържа такъв диапазон от значения.

```
int x = { 2.88 };  // грешка на етапа за компилиране. 2.88 –
double, а x е променлива от цялостен тип
char ch = { 777 };  // грешка на етапа за компилиране. 777 –
int, а ch е променлива от символен тип
// 777 не попада в диапазона от значения char
```

От друга страна:

```
char ch2 = { 23 };     // всичко е вярно. 23 попада в диапазона char
double x = { 333 };  // всичко е вярно 333 – int и попада в диапазона double
```

Ако искате да изявите потенциални проблеми със загуба на данни на етапа за компилиране вие можете да използвате инициализация по списък.

# 5. Логически операции

В програмирането много често е необходимо не само да се привеждат изчисления но и да се сравняват величините помежду си. За това се използват така наречените логически операции. Резултатът от логическите операции винаги излиза като значение true, или значение false, тоест вярно или грешно. Логическите операции се разделят на три подгрупи:

1. Оператори за сравнения
2. Оператори за равенства
3. Логически оператори за обединение и отрицателна инверсия.

Сега нека по-детайлно да разгледаме всяка група оператори.

**Оператори за сравнения.**

Използва се тогава, когато е необходимо да се изясни по какъв начин две величини се отнасят една с друга.

| Символ обозначаващ оператор | Утвърждение |
|---|---|
| < | Левия операнд е по-малък от десния |
| > | Левия операнд е по-голям от десния |
| <= | Левия операнд е по-малък или равен на десния |
| >= | Левия операнд е по-голям или равен на десния |

Смисъла на операциите за сравнение (второто име на операциите за отношение) е в това, че ако утвърждение зададено с помощта на оператора вярно, израз в който той участва се замени на значение true, ако не е вярно – на значение false. Например:

```
cout<<(5>3); // на екрана е единица след като утвърждението (5>3) е вярно.
cout<<(3<2); //на екрана е 0, след като (3<2) е грешно.
```

**Забележка:** Вместо значенията false и true на екрана излизат 0 и 1, защото те са еквивалентни на значенията варно и грешно. В езика C в ролята на истина, също така може да се представи и всяко друго число различно от 1 и от 0, както положително, така и отрицателно.

**Оператор за равенства.**

Използва се за проверка на пълно съответствие или несъответствие на две величини.

Прилагането на тези оператори съвпада с принципите за прилагане на предишната група, тоест на в края си израза се заменя или с вярно или с грешно, в зависимост от утвърждението.

| Символ обозначаващ оператора | Утвърждение |
|---|---|
| == | Левия операнд е равен на десния |
| != | Левия операнд не е равен на десня |

Прилагането на тези оператори съвпада с принципите за прилагане на предишната група, тоест на в края си израза се заменя или с вярно или с грешно, в зависимост от утвърждението.

```
cout<<(5!=3); // на екрана е единица,
след като утвърждението (5!=3) е вярно.
cout<<(3==2); //на екрана е 0, след като (3==2) е грешно.
```

## Логически операции за обединение и отрицателна инверсия.

В повечето от случаите е невъзможно да се разминеш само с едно утвърждение. По-често е необходимо да се комбинират с един или друг образ. Например за да проверим дали едно число се намира от диапазона от 1 до 10 е необходимо да се проверят две утвърждения: числото трябва да бъде едновременно >= 1 и <= 10. За да се реализира такава комбинация е необходимо да се въведат допълнителни оператори.

| Операция | Название |
|---|---|
| && | И |
| \|\| | ИЛИ |
| != | НЕ |

## Логическо И (&&)

Логическото И обединява заедно две утвърждения и възвръща вярно само ако и лявото и дясното утвърждение са варно. Ако дори едно от утвържденията е грешно или и двете, обединения израз ще бъде представен като грешно. Логическото И работи по съкратена схема, тоест ако първото утвърждение е грешно, второто вече не се проверява.

| Утвърждение 1 | Утвърждение 2 | Утверждение1&&Утвърждение 2 |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

Сега да разгледаме пример в който програма получава число и определя дали попада това число в диапазона от 1 до 10.

```
#include <iostream>
using namespace std;
void main()
{
	int N;
	cout<<"Enter digit:\n";
	cin>>N;
	cout<<((N>=1)&&(N<=10));
	cout<<"\n\nIf you see 1 your digit is in diapazone\n\n";
	cout<<"\n\nIf you see 0 your digit is not in diapazone\n\n";
}
```

В дадения пример, ако и двете утвърждения бъдат верни, на мястото на израза ще се постави 1, в противен случай - 0. Съответно потребителя ще може да проанализира получилата се ситуация, използвайки инструкциите на програмата.

**Логическото ИЛИ (||)**

Логическото ИЛИ обединява заедно две утвърждения и показва вярно само когато поне едно от утвържденията е вярно, а грешно в случай че и двете утвърждения са грешни. Логическото ИЛИ работи по съкратена схема, тоест ако първото утвърждение е вярно, второто вече не се проверява.

| Утвърждение 1 | Утвърждение 2 | Утвърждение1&&Утвърждение 2 |
|---|---|---|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

Още един път ще разгледаме пример в който програма получава число и определя попада ли това число в диапазона от 1 до 10. Само че този път ще използваме ИЛИ.

```
#include <iostream>
using namespace std;
void main()
{
	int N;
	cout<<"Enter digit:\n";
	cin>>N;
	cout<<((N<1)||(N>10));
	cout<<"\n\nIf you see 0 your digit is in diapazone\n\n";
	cout<<"\n\nIf you see 1 your digit is not in diapazone\n\n";
}
```

В дадения пример ако и двете утвърждения бъдат грешни, (тоест числото ще бъде не по-малко от 1 и не повече от 10) на мястото за израз ще се сложи 0, противен случай- 1. Съответствено потребителя, също както и в предишния пример, ще може да проанализира получилата се ситуация и да изкара извод.

**Логическо НЕ (!)**

Логическото НЕ представлява унарен оператор и поради тази причина не може да се нарече оператор за обединение. То се използва в случай че е необходимо да се промени резултата от проверка на утвърждение на противоположен.

| Утвърждение | ! Утвърждение |
|---|---|
| true | false |
| false | true |

```
// на екрана ще е 1, след като (5==3) е грешно а неговата инверсия -
вярно. cout<<!(5==3);
//на екрана ще е 0, след като (3!=2) е истина а неговата инверсия -
грешно. cout<<!(3!=2);
```

Логическото отрицание поставя на мястото на утвърждението грешно ако последното е вярно и наопаки, вярно ако утвърждението е грешно. Дадения оператор може да се използва за съкращаване на постановката за условие. Например израза

```
b==0
```

Може съкратено да се запише с инверсията:

```
!b
```

И двата дават на края вярно, в случай че b е равен на нула.

В дадения раздел ние ще разгледаме всевъзможните логически операции, които позволяват да се определи верността на всяко утвърждение. Обаче описаните тук примери са неудобни за редовия потребител, защото анализа на резултатите трябва да проведе не той а програмата. Освен това ако в зависимост от утвърждението е необходимо не просто да се изкарва на екрана резултата на неговата проверка, а да се произведе каквото и да е действие, тук вече потребителя е безсилен. Във връзка с това обладавайки знания за логическите операции е необходимо да се получи допълнителна

информация за възможностите за реализация на едно или друго действие в зависимост от условието. Именно за това и ще говорим в следващия раздел на нашия урок.

# 6. Конструкция на логическия избор if

Сега ние с вас ще се запознаем с оператор, който позволява превръщането на обикновена линейна програма в «мислеща». Дадения оператор позволява да се провери определено утвърждение (израз) дали е верен и в зависимост от получения резултат да произведе едно или друго действие. За начало нека да разгледаме общия синтаксис на дадения оператор:

```
if (утвърждение или израз)
{
  действие 1;

else
{
  действие 2;
}
```

**Основни принципи на работа на оператора if.**

1. В качеството на утвърждение или израз може да излезе каквато и да е конструкция, съдържаща логически оператори или аритметични изрази.

■ if(X>Y) — обикновено утвърждение, ще бъде вярно ако X действително е повече от Y

```
int X=10,Y=5;
if(X>Y){ // вярно
cout<<"Test!!!";// на екрана Test
}
```

■ if(A>B&&A<C) — комбинира утвърждение, състоящото от две части ще бъде вярно ако и двете части са верни

```
int A=10,B=5,C=12;
if(A>B&&A<C){ // вярно
      cout<<"A between B and C";// на екрана A between B and C
}
```

■ if(A–B) — аритметичен израз, ще бъде верен ако A не е равно на B, защото в противен случай (ако са равни) тяхното деление ще даде нула, а нула е грешно

```
int A=10,B=15;
if(A-B){ // -5 е вярно
      cout<<"A != B";// на екрана A != B
}
```

■ if(++A) — аритметичен израз, ще бъде верен ако A не е равно на -1 защото ако A е равно на -1 увеличаване с 1 ще даде нула, а нула е грешно

```
int A=0;
if(++A){ // 1 вярно
      cout<<"Best test!!";// на екрана Best test!!
}
```

■ if(A++) — аритметичен израз, ще бъде верен ако A не е равно на 0, защото в дадения случай се използва постфиксна форма на инкремента, първо ще се проведе проверка на условието и ще бъде намерена нула а след това увеличение на единица.

```
int A=0;
if(A++){ // 0 грешно
      cout<<"Best test!!";// тази форма няма да я видим, след като if няма да се
      изпълни
}
```

■ if(A==Z) — обикновено утвърждение, ще бъде вярно ако A е равно Z

■ if(A=Z) — операция за присвояване, израза ще бъде верен, ако Z не е равен на нула.

**Забележка :** Типична грешка. Много често вместо операция за проверка на равенство ==, по невнимателност се указва операция за присвояване=, и смисъла на израза може да се промени радикално. Такава банална грешка може да доведе до некоректна работа на цялата програма. Ще разгледаме два на пръв поглед идентични примера.

## Правилен пример.

```
#include <iostream>
using namespace std;
void main(){
  int A,B; //обявяваме две променливи

    //молим потребителя да въведе в тях данни
    cout<<"Enter first digit:\n";
    cin>>A;
    cout<<"Enter second digit:\n";
    cin>>B;

    if(B==0){ // ако в B се съдържа нула
      cout<<"You can't divide by zero!!!";// съобщаваме за грешка
    }
    else{ // в противен случай
  cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;// издаваме резултата от делението A на B
    }
    cout<<"\n The end. \n"; // край
}
```

## Грешен пример.

```
#include <iostream>
using namespace std;
void main(){
          int A,B; //обявяваме две променливи
          //молим потребителя да въведе данни в тях
           cout<<"Enter first digit:\n";
           cin>>A;
           cout<<"Enter second digit:\n";
           cin>>B;

            // Приравняваме B към нулата и проверяваме условието, то
           автоматично е грешно if(B=0){ // тази част няма да се изпълни
           никога след като условието винаги е грешно
                  cout<<"You can't divide by zero!!!";
           // съобщаваме за грешка
             }
 else{  // винаги се изпълнява тази част в която A се дели на новоизпечена нула
/* В този ред ще произлезе грешка след като компютъра ще се опита да раздели
       числото на нула  */ cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;
              }
              cout<<"\n The end. \n"; // Тази фраза няма да я видим никога.
 }
```

2. Както вече успяхте да забележите, ако съдържанието в кръглите скоби е вярно, то ще се изпълни действие 1, заключеното във фигурните скоби с конструкцията if при това действие 2 блок 2 else ще бъде игнориран.

3. Ако съдържанието в кръглите скоби е грешно, ще се изпълни действие 2, заключеното във фигурните скоби на конструкцията else при това действие 1 ще бъде игнорирано.

4. Конструкцията else не е задължителна. Това означава че ако няма необходимост да се прави нещо при грешно утвърждение, дадената конструкция може да не указва. Например, програма използваща защита против деление с нула, може да се запише по такъв начин:

```
#include <iostream>
using namespace std;
void main(){
         int A,B; //обявяваме две променливи
         //молим потребителя да въведе данни в тях
         cout<<"Enter first digit:\n";
         cin>>A;
         cout<<"Enter second digit:\n";
         cin>>B;
         if(B!=0){ // ако B не е равно на нула
                 cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B; // провеждаме изчисление
         }
         // в противен случай не правим
         нищо   cout<<"\nThe end.\n";
}
```

5. Ако към блока if или else се отнася само една команда, то фигурните скоби може да не се указват. С помощта на това правило ще направим програмата още по кратка:

```
#include <iostream>
using namespace std;
void main(){
       int A,B; //обявяваме две променливи

        //молим потребителя да въведе данни в тях
        cout<<"Enter first digit:\n";
        cin>>A;
        cout<<"Enter second digit:\n";
        cin>>B;

       if(B!=0) // ако B не е равно на нула
           cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B; // провеждаме изчисление
       // в противен случай е правим нищо
       cout<<"\nThe end.\n";
}
```

Ние току що сме се запознали с условния оператор if и сме обсъдили основните принципи на неговото действие. Преди да преминем към разглеждането на специфичните особености на if и практическите примери, ще направим малко отстъпление и ще разгледаме още един оператор с помощта на когото

може да се постави просто условие.

**Забележка:** Бъдете внимателни: оператор if и оператор else са неразделни! Опит да се вмъкне между тях ред със код ще доведе до грешка на етапа за компилиране..

**Фрагмент от кода с грешка.**

```
....
    if(B==0){ // ако в B се съдържа нула
        cout<<"You can't divide by zero!!!";// съобщаваме за грешка
    }
    cout<<"Hello";// Грешка!!!! Разрив в конструкцията if -
    else!!! else{ // в противен случай
              cout<<"Result = "<<A/B;// издаваме резултата от делението на A на B
    }
            ....
```

**Тернарен оператор.**

Някои условия са доста примитивни. Например, да вземем нашата програма за деление на две числа. Тя е проста и гледната точка на действията и от гледна точка на кода. На операторите if и else им се полага по един ред от кода - действие. Такава програма, може да се упрости повече, използвайки тернарнен оператор.

За начало нека да разгледаме неговия синтаксис:

```
УТВЕРЖДЕНИЕ ИЛИ ВЫРАЖЕНИЕ?ДЕЙСТВИЕ1:ДЕЙСТВИЕ2;
```

Принципа на действие е прост — ако е УТВЪРЖДЕНИЕ или ИЗРАЗ — вярно, изпълнява се ДЕЙСТВИЕ 1, ако е — грешно, изпълнява се ДЕЙСТВИЕ 2.

Хайде да разгледаме действието на дадения оператор в примера:

```
#include <iostream>
using namespace std;
void main(){
    int A,B; //обявяваме две променливи

    //молим потребителя да въведе данни в тях
    cout<<"Enter first digit:\n";
    cin>>A;
    cout<<"Enter second digit:\n"
    cin>>B;

                    В дадения случай, ако B няма да е равно на нула, ще се
              изпълни тази команда седи зад въпросителния знак и на екрана ще се
              покаже резултата от делението. В противен случай, ще се изпълни
              командата стояща след знака двоеточие и на екрана ще бъде
              съобщение за грешка с деление на нула.*/
 (B!=0)?cout<<"Result A/B="<<A<<"/!"<<B<<"="<<A/B:cout<<"You can't divide by zero!!!";

    //край на програмата
    cout<<"\n The end. \n";
}
```

Не е ли така, кода е станал още по-оптимален!? За укрепване на получената информация ще приведем още един по-сложен пример. Програмата ще определя кое от двете числа въведени от потребителя е по-голямо и кое по-малко.

```
#include <iostream>
using namespace std;
void main(){
    int a,b; //обявяваме две променливи

    //молим потребителя да въведе данни в тях
    cout<<"Enter first digit:\n";
    cin>>a;
    cout<<"Enter second digit:\n";
    cin>>b;

    /*Ако, (b>a), на мястото на оператора ?: ще се появи
    b, в противен случай на мястото на оператора ще се
    появи a, по такъв начин това число, което е по-
    голямо ще се запише в променливата max
    .*/ int max=(b>a)?b:a;

    /*Ако, (b<a), то на мястото на оператора ?: ще се
    постави b а противен случай на мястото на оператора ще
    се постави a, по такъв начин числото, което е по-
    голямо ще се запише в променливата min.*/
    int min=(b<a)?b:a;

    // Извеждане на резултата на екрана
    cout<<"\n Maximum is \n"<<max;
    cout<<"\n Minimum is \n"<<min<<"\n";
}
```

И така нека да затвърдим следното: Ако условието и действията зависещи от него са достатъчно прости, ще използваме тернарен оператор. Ако на нас ни е нужна сложна конструкция, то безусловно ще използваме оператора if.

# 7. Стълбичка if – else if

От предишния раздел на урока вие сте се запознали с съществуването на условни оператори. Сега няма да е лошо да да получим информация за особеностите на тяхната работа.

Да предположим че ни е необходимо да напишем програма за отчет на паричните отстъпки, в зависимост от сумата. Например, ако покупателя е купил товар на сума по голяма от 100лв. той получава отстъпка 5%. Ако е повече от 500лв. – 10%, накрая ако е повече от 1000лв. – 25%. Приложението трябва да даде сумата която покупателя е длъжен да заплати ако е получил отстъпка. Сега е необходимо да се намери оптималния вариант за решение на задачата. Името на проекта Discount.

**Вариант за решение № 1.**

```
#include <iostream>
using namespace std;
void main(){
	// обявява се променлива за запазване на първоначалната
	сума int summa;

	// молба за въвеждане на сума с
	клавиатура
	 cout<<"Enter item of summa:\n";
	cin>>summa;

	if(summa>100){ // ако сумата е повече от  100 лв.,
	отстъпка 5% cout<<"You have 5% discount!!!\n";
		cout<<"You must pay - "<<summa-summa/100*5<<"\n";
	}
	if(summa>500){ // ако е повече от 500 лв., отстъпка 10%
	cout<<"You have 10% discount!!!\n";
	cout<<"You must pay - "<<summa-summa/100*10<<"\n";
```

```
        }
        if(summa>1000){ // ако сумата е повече 1000 лв., отстъпка
        25% cout<<"You have 25% discount!!!\n";
        cout<<"You must pay - "<<summa-summa/100*25<<"\n";
        }
        else{ // в противен случай няма
        отстъпка
         cout<<"You have not discount!!!\n";
        cout<<"You must pay - "<<summa<<"\n";
        }

}
```

Даденият пример, на пръв поглед за начинаещия програмист не предизвиква неразбиране, обаче, нека да разгледаме ситуацията в която програмата работи некоректно. Сумата въведена с клавиатурата е равна на 5000. Тази цифра превишава 1000, следователно ние трябва да получим 25% отстъпка. Обаче ще се получи съвсем различно нещо.

1. Всеки оператор if е самостоятелен и не зависи от другите if, следователно независимо от това кое от if ще се изпълни, проверката на условието все едно ще бъде осъществена за всички оператори.

2. Първо се осъществява проверка на условието if(suma>100). 5000, естествено е повече от 100, условието е вярно и се изпълнява тялото if. На екрана ни излиза :

```
You have 5% discount!!!
You must pay - 4750
```

3. Обаче със това програмата не спира – после ще бъде анализирано условието if(suma>500). 5000 е повече от 500, условието отново е вярно и се изпълнява тяло if. На екрана получаваме:

```
You have 10% discount!!!
You must pay - 4500
```

4. И накрая програмата ще провери условието if(suma>1000) което също ще се окаже че е вярно защото 5000 е повече от 1000. И действието свързано с if ще се изпълни отново. На екрана излиза:

```
You have 25% discount!!!
You must pay - 3750
```

По такъв начин, вместо един информационен надпис ние получаваме три. Такова решение на задачата е нерентабилно. Ще се пробваме да го оптимизираме. Името на проекта е **Discount2**.

## Вариант за решение № 2.

```
#include <iostream>
using namespace std;
void main(){
        // обявява се променлива за съхраняване на първоначалната
        сума
        int summa;

        // молба за въвеждане на сума с
        клавиатурата
        cout<<"Enter item of summa:\n";
        cin>>summa;

        // ако сумата е в диапазона от 100 лв. до 500 лв.,
отстъпка 5%
        if(summa>100&&summa<=500){
                cout<<"You have 5% discount!!!\n";
                cout<<"You must pay - "<<summa-summa/100*5<<"\n";
        }
        // ако сумата е в диапазона от 500 лв. до 1000 лв., отстъпка
        5% if(summa>500&&summa<=1000)
```

```
{
                cout<<"You have 10% discount!!!\n";
                cout<<"You must pay - "<<summa-summa/100*10<<"\n";
        }
        if(summa>1000){ // ако сумата е повече от 1000 лв.,
                отстъпка
         25% cout<<"You have 25% discount!!!\n";
                cout<<"You must pay - "<<summa-summa/100*25<<"\n";
        }
        else{ // в противен случай няма отстъпка
                cout<<"You have not discount!!!\n";
                cout<<"You must pay - "<<summa<<"\n";
        }
}
```

Като за начало отново ще си представим, че потребителя е въвел сума с размер от 5000лв.

1. Първо ще се осъществи проверка на условието if(summa>100&&summa<=500). 5000 не влиза в зададения диапазон, условието е грешно и функцията if няма да се изпълни.

2. След това ще бъде проанализирано условието if(summa>500&&summa<=1000). 5000 не влиза и в този диапазон, условието отново е грешно и if няма да се изпълни.

3. И накрая програмата ще провери условието if(summa>1000), което ще се окаже вярно защото 5000 е повече от 1000. И действието на екрана свързано с of ще се изпълни. На екрана излиза:

```
You have 25% discount!!!
You must pay - 3750
```

Изглежда, че с това може да приключим, но нека да проверим още един вариант. Например, потребителя въвежда значение 600. И на екрана излизат следните данни:

```
Enter item of summa:
600
You have 10% discount!!!
You must pay - 540
You have not discount!!!
You must pay - 600
Press any key to continue
```

Такъв обрат на събитията се обяснява лесно:

1. Първо се осъществява проверка на условието if(summa>100&&summa<=500). 5000 не влиза в зададения диапазон, условието е грешно и тялото if няма да се изпълни.

2. После ще бъде анализирано условието if(summa>500&&summa<=1000). 5000 влиза в този диапазон, условието е вярно и тялото if ще се изпълни, на екрана излиза съобщение за 10% отстъпка.

3. И накрая програмата ще провери условието if(summa>1000), което ще излезе грешно. Действията свързани с if няма да се изпълнят, но в дадения самостоятелен оператор if има собствен else, който ще проработи в нашия случай. На екрана ще излезе съотношение за отсъствие на отстъпка.

Извод: в първите примери ние изяснихме, че оператор else се отнася само към последния if. Във втория стигнахме до това, че дадената реализация на програмата не ни устройва. Да разгледаме още един пример за решение. Името на проекта е Discount 3.

## Вариант за решение № 3.

```
#include<iostream>
using namespace std;
void main(){
// обявява се променлива за запазване на първоначалната
сума int summa;

// молба за въвеждане с клавиатура
cout<<"Enter item of summa:\n";
cin>>summa;

if(summa>1000){ // ако сумата е повече от 1000 лв.,
    отстъпка 25% cout<<"You have 25% discount!!!\n";
    cout<<"You must pay - "<<summa-summa/100*25<<"\n";
}
else{ // ако сумата не е повече от 1000 лв. Продължаваме
анализа if(summa>500){ // ако сумата е повече от 500
лв., отстъпка 10%
    cout<<"You have 10% discount!!!\n";
    cout<<"You must pay - "<<summa-summa/100*10<<"\n";
}
else{ // ако сумата е повече от 500 лв. Продължаваме
анализа if(summa>100){ // ако сумата е повече от 100 лв.,
отстъпка 5%
    cout<<"You have 5% discount!!!\n";
    cout<<"You must pay - "<<summa-summa/100*5<<"\n";
}
else{  // ако сумата не е повече от 100 лв.
    Отстъпка няма cout<<"You have not
    discount!!!\n";
    cout<<"You must pay - "<<summa<<"\n";
       }
     }
   }
}
```

След внимателен анализ на дадения пример, ще забележите, че всеки следващ if, може да бъде изпълнен само в случай, че не се изпълни неговият „предшественик“, тъй като се намира в рамките на конструкцията else на последния. «Стълбичка if else if», защото условието в нея се разполага под формата на стълба. Сега ние с вас вече знаем колко е полезна тази конструкция. Остава един последен щрих:

## Оптимизация на кода.

В предишния раздел на урока, прозвуча правилото: Ако към блока if или else се отнася само една команда, то тогава може да не се поставят фигурни скоби. Работата е в това, че конструкцията if else се смята за една цяла командна структура. Следователно, ако в някои else няма нищо освен изложената конструкция, фигурните скоби за такива else може да се пропуснат:

```
#include <iostream>
using namespace std;
void main(){
// обявява се променлива за запазване на първоначалната
сума int summa;

// молба за въвеждане с клавиатура
cout<<"Enter item of summa:\n";
cin>>summa;

if(summa>1000){ // ако сумата е повече от 1000 лв.,
        отстъпка 25%
        cout<<"You have 25% discount!!!\n";
        cout<<"You must pay - "<<summa-summa/100*25<<"\n";
 }

// ако сумата не е повече от 1000 лв. Продължаваме анализа
else if(summa>500){  // ако сумата е повече от 500 лв.,
                отстъпка 10%
                cout<<"You have 10% discount!!!\n";
                cout<<"You must pay - "<<summa-summa/100*10<<"\n";
 }

 // ако сумата не е повече от  500 лв. Продължаваме анализа
else if(summa>100){ // ако сумата е повече от 100 лв., отстъпка 5%
                        cout<<"You have 5% discount!!!\n";
                        cout<<"You must pay - "<<summa-summa/100*5<<"\n";
 }
else{  // ако сумата не е повече от  100 лв. Няма отстъпка
                        cout<<"You have not discount!!!\n";
                        cout<<"You must pay - "<<summa<<"\n";
 }
}
```

Това е всичко! Задачата е решена. Ние сме получили конструкция с множествен избор, съставена от отделни взаимозависими условия. Сега може да преминем към следващите раздели на урока, където ние с вас подробно ще разгледаме още няколко примера за използването на if else.

# 8. Практически пример: създаване на текстов куест

## Постановка на задачите

Вие естествено сте запознати с такъв жанр игри като куест. Героя на такава игра е длъжен да изпълни различни задачи, да отговаря на въпроси, да взима решения от които зависи резултата на играта. Ние с вас ще се опитаме сега да създадем така наречения текстов куест (куест без графика). Нашата задача е да предложим на героя варианти за действия и в зависимост от неговия избор да се подреди ситуация. Името на проекта **Quest.**

## Код за реализация.

```
#include <iostream>
using namespace std;
void main()
{
    // Добре дошли. Три изпитания за честта. Лош магьосник е отвлякъл
    //принцеса и нейната съдба е в твоите ръце. Той ти предлага да
    //преминеш три изпитания за честа в неговия лабиринт.
    cout<<"Welcome. Three tests of honour. The malicious magician has stolen\n\n";
    cout<<"\nprincess and its destiny in your hands. It suggests you\n";
    cout<<"\nto pass 3 tests of honour in its labyrinth.\n";


    bool goldTaken, diamondsTaken, killByDragon;
    //Ти влизаш в първата стая, тук има много злато.
    cout<<"You enter into the first room, here it is a lot of gold.\n\n";
    //Ще го вземеш ли?
    cout<<"Whether you will take it?(1-yes, 0-no)\n\n";
    cin>>goldTaken;
    if(goldTaken) // ако го вземеш
    {
        //Златото остава за теб, но си се провалил в изпитанието. ИГРАТА ПРИКЛЮЧИ!!!
```

```
        cout<<"Gold remains to you, but you have ruined test. GAME is over!!!\n\n";
    }
    else // ако не
    {
        //Поздравления, ти премина първото изпитание!
        cout<<"I congratulate, you have passed the first test abuse!\n\n";
        //Преминаваш в следващата стая. Тя е пълна с брилянти
        cout<<"You pass in a following room. It is full of brilliants \n\n";
        //Ще вземеш ли брилянтите?
        cout<<"Whether you will take brilliants? (1-yes,0-no)\n\n";
        cin>>diamondsTaken;
        if(diamondsTaken)// ако ги вземеш
        {
                //Брилянтите остават за теб, но ти се провали с второто изпитание
                cout<<"Brilliants remain to you, but you have ruined the second test\n\n";
                //ИГРАТА ПРИКЛЮЧИ!!!
                cout<<"GAME is over!!!\n\n";
        }
        else //ако не
        {
                //Поздравления, ти премина второто изпитание за чест!!!
                cout<<"I congratulate, you have passed the second test abuse!!!\n\n";
                //Влизаш в третата стая.
                cout<<"You enter into the third room. \n\n";
                //Дракон е нападнал селянин! Продължаваш на пред
                cout<<"The person was attacked by a dragon! To move further \n\n";
                //без да им обръщаш внимание
                cout<<"Not paying to them of attention (1-yes,0-no)?\n\n";
                cin>>killByDragon;
                if(killByDragon)//ако реши това
                {
                        //Ще се опиташ да минеш незабелязано, но дракона
                        cout<<"You try to pass past, but a dragon \n\n";
                        //те забелязва.cout<<"notices your
                        presence\n\n";
                        //Той те превръща в пепел. Ти си мъртъв!!!
                        cout<<"It transforms you into ashes. You are dead!!!\n\n";
                        //ИГРАТА ПРИКЛЮЧИ!!!
                        cout<<"GAME is over!!!\n\n";
                }
                else//ако не
                {
                        //Поздравления ти премина всички изпитания!!!
                        cout<<"I congratulate, you with honour have was tested all!!! \n\n";
                        //Принцесата е твоя!!!
                        cout<<"Princess gets to you!!!\n\n";
                }
        }
    }
}
```

Независимо от примитивността на примера, вие можете да се убедите, че сега вече, имайки минимални знания, ние можем да напишем програма, способна да забавлява средностатистическия мъник. Това се получава защото ние в ръцете си имаме мощно средство – условните оператори.

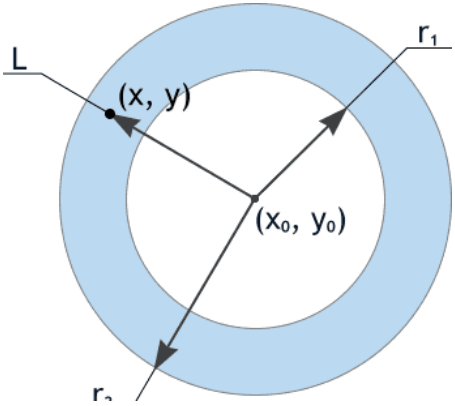
# 9. Практически пример за принадлежност в точка от точка от окръжност

**Постановка на задачите**

На плоскостта е нарисувана окръжност с център в точката (x0,y0). И радиуси с граници r1<r2. Освен това на същата плоскост е дадена точка с координати (x, y). Изисква се да се определи, принадлежи ли тази точка към окръжността. Името на проекта е **Circle2.**

**Решение на задачата**

За решение на задачата е необходимо да се изчисли разстоянието от центъра на окръжността до точката и да се сравни с радиусите:




1. Ако дължината на правата от центъра до точката е по-малка отколкото радиуса на външната окръжност, то тогава точката принадлежи към окръжността .

if(L>=r1&&L<=r2)

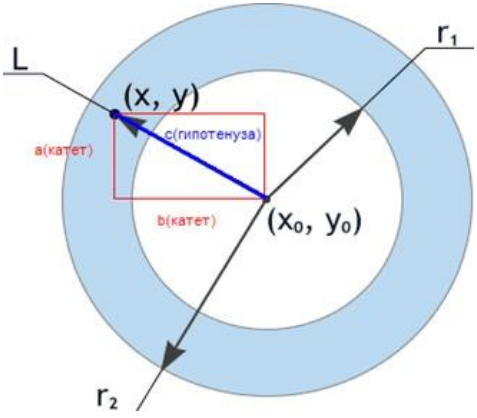
В противен случай точката не принадлежи на окръжността.

2. За изясняване на разстоянието от центъра до точката ние ще се възползваме от Питагоровата теорема — **Квадрата на хипотенузата е равен на сумата от квадратите на катетите.** Следователно — **дължината на хипотенузата е равна на корена на квадрата от сумите на квадрат от катетите.**

**Забележка:** Необходими са ни допълнителни знания за да получим степента и корена квадратен.

1. В програмата е необходимо да се включи библиотека за използване на математически функции с името math.h.

2. За степенуване се използва функцията pow(double num,double exp), където num е числото за степенуване, а exp — самата степен.




3. За извличане на корен квадратен се използва sqrt(double num), където num това е число от което се извлича корена.

```
c=sqrt(pow(a,2)+pow(b,2));
L=c;
```

3. Остава да се изясни дължината на катетите, на рисунката се вижда как да се направи..

```
a=y-y0;
b=x-
x0;
```

Се остава да се съберат всички части на решението в едно цяло.

```
#include <iostream>
#include <math.h>
using namespace std;
void main() {
  // Обявяване на
  променливите
   int x0, y0, r1, r2, x,
  y; float L;

  // Молба за въвеждане на необходимите данни
cout<<"Input coordinates of circle's center (X0, Y0):";
  cin>>x0>>y0;
  cout<<"Input circle radiuses R1 and R2:";
  cin>>r1>>r2;
  cout<<"Input point coordinates (X, Y): ";
  cin>>x>>y;

// Въвеждане на формулата
  L = sqrt(pow(x - x0, 2) + pow(y - y0, 2));

//Анализ на резултатите
  if ((r1 < L)&& (L < r2 )) {
      cout<<"This point is situated inside the circle.\n";
     }
 else {
      cout<<"This point is not situated inside the circle.\n";
     }
```

# 10. Структура на множествения избор switch

Ние вече сме запознати с конструкцията за анализиране на условия – конструкцията if, също така с тернарния оператор. И още един оператор за избор – оператора switch. Представете си, че е необходимо да се напише програма в която се използва меню съставено от пет пункта. Например, малко приложение за децата които умеят да ???? Може да се реализира обработка на избора с помощта на стълбичката if else if, ето така:

```
# include <iostream>
using namespace std;
void main(){
// обявяване на променливите и въвеждане на
значенията с клавиатурата
 float A,B,RES;
cout<<"Enter first digit:\n\";
cin>>A;
cout<<"Enter second digit:\n":;
cin>>B;

// реализация на програмното
меню
 char key;
cout<<"\nSelect operator:\n»;
cout<<"\n + - if you want to see SUM.\n";
cout<<"\n - - if you want to see DIFFERENCE.\n";
cout<<"\n * - if you want to see PRODUCT.\n0";
cout<<"\n / - if you want to see QUOTIENT.\n";

//очакване на избор cin>>key;

if(key=='+') { // ако потребителя е избрал събирне
 .      RES=A+B;
 .      cout<<"\nAnswer: "<<RES<<"\n";
  }
else if(key=='-'){ // ако потребителя е избрал изваждане
```

```
.       RES=A-B;
.       cout<<"\nAnswer: "<<RES<<"\n";
 }
else if(key=='*'){ // ако потребителя е избрал умножение
.       RES=A*B;
.       cout<<"\nAnswer: "<<RES<<"\n";
 }
else if(key=='/'){ // ако потребителя е избрал деление
.       if(B){ // ако делителя не е равен на нула
.                 RES=A/B;
.                 cout<<"\nAnswer: "<<RES<<"\n";
.       }
.       else{ // ако делителя е равен на нула
.                 cout<<"\nError!!! Divide by null!!!!\n";
.       }
 }
else{ // ако въведения символ е некоректен
.       cout<<"\nError!!! This operator isn't correct\n";
 }
}
```

Гореописания пример е напълно коректен, но изглежда малко некрасиво. Дадения код може значително да се упрости, точно за това се използва switch. Той позволява да се сравни значението на променливата с цял ред значения и, срещайки съвпадение да изпълни определено действие.

**Общ синтаксис и принцип на действие.**

Като за начало, ще разгледаме общия синтаксис на оператора:

```
switch(израз){
case значение1:
          действие1;
              break;
case значение2:
          действие2;
              break;
case значение3:
          действие3;
              break;
......
default:
действие_по_подразбира
не;
              break;
}
```

Хайде да анализираме въпросната форма на запис:

1. Израз — тези данни които е необходимо да се проверят за съответствия. Тук може да се сложи променлива (само ако е от типа char или е целочислена), или израз, резултата на който са целочислени данни.

2. case Значение1, case значение2, case значение3 — Целочислени или символни постоянни значения с които се проверява израз.

3. Действие1, действие2, действие3 — Действия, които трябва да се изпълнят ако значението на израза е съвпаднало с значението на case.

4. Ако се е получило съвпадение и се е изпълнило благополучно действието свързано със съвпадналите case, switch завършва работата си и програмата преминава на следващия ред зад закрития от фигурна скоба оператор switch. За дадената функция отговаря оператора break именно той установява изпълнението на switch.

5. Ако в момента на анализа не произлиза съвпадение, проработва функцията default и се изпълнява действие по подразбиране. Оператора default е аналог на оператора else.

Сега нека да погледнем, по какъв начин може да се упрости приведения в началото на темата пример.

# Оптимизация на примера.

```
# include <iostream>
using namespace std;
void main(){

// обявяване на променливите и въвеждане на значенията с
клавиатура
float A,B,RES;
cout<<"Enter first digit:\n";
cin>>A;
cout<<"Enter second digit:\n";
cin>>B;

// реализация на програмното меню
char key;
cout<<"\nSelect operator:\n";
cout<<"\n + - if you want to see SUM.\n";
cout<<"\n - - if you want to see DIFFERENCE.\n";
cout<<"\n * - if you want to see PRODUCT.\n";
cout<<"\n / - if you want to see QUOTIENT.\n";

//изчакване избора на потребителя
cin>>key;

//проверка значението на променливата
key switch(key){
case  '+':    // ако потребителя е избрал
        събиране
 RES=A+B;
        cout<<"\nAnswer: "<<RES<<"\n";
        break; // стоп switch
case  '-':     // ако е избрал изваждане RES=A-B;
        cout<<"\nAnswer: "<<RES<<"\n";
        break; // стоп switch
case  '*':    // ако е избрал умножение RES=A*B;
        cout<<"\nAnswer: "<<RES<<"\n";
        break; // стоп switch  case '/':
        // ако е избрал деление
        if(B){ // ако делителя не е нула
                  RES=A-B;
                  cout<<"\nAnswer: "<<RES<<"\n";
        }
        else{ // ако делителя е равен на нула
                  cout<<"\nError!!! Divide bynull!!!!\n";
        }
        break; // стоп switch
default:    // ако въведения символ е некоректен
        cout<<"\nError!!! This operator isn't correct\n";
        break; // стоп switch
  }
}
```

Както виждате, сега кода изглежда доста по-прост и е по-удобен за четене.

Оператора switch е достатъчно прост за използване, обаче е необходимо да се знаят някои от особеностите на неговата работа:

1. Ако в case се използват символни значения, те трябва да се показват в одинарни кавички, а ако са целочислени, то без кавички.

2. Оператора default може да се разположи на всяко място в системата switch, защото ще се изпълни така или иначе само тогава когато няма нито едно съвпадение. Обаче правилото за „добър тон“ е да се поставя default в края на цялата конструкция.

```
switch(израз){
case значение1:
      действие1;
      break;
case значение2:
      действие2;
      break;
default:
```

```
      действие_по_подразбира
      не; break;
case значение3:
      действие3;
      break;
}
```

3. След последния оператор в списъка (независимо case или default) оператор break може да не се поставя.

```
switch(израз){
case значение1:
      действие1;
      break;
case значение2:
      действие2;
      break;
default:
      действие_по_подразбира
      не; break;
case значение3:
      действие3;
}
switch(израз){
case значение1:
      действие1;
      break;
case значение2:
      действие2;
      break;
case значение3:
      действие3;
      break;
default:
      действие_по_подразбиране;
}
```

4. Операторът default може въобще да не се поставя, в случай че не произлезе съвпадение, просто нищо няма да се случи.

```
switch(израз){
case значение1:
   действие1;
       break;
case значение2:
   действие2;
       break;
case значение3:
   действие3;
       break;
}
```

5. В случай, че е необходимо да се изпълни един или друг набор от действия за различни значения на проверявания израз, може да се запишат няколко отметки подред. Да разгледаме пример на програма, която превежда система от буквени оценки в цифрови.

```
# include <iostream>
using namespace std;
void main(){
// обявяване на променлива, за съхранение на буквена
оценка char cRate;

// молба за въвеждане на буквена
оценка
 cout<<"Input your char-rate\n";
cin>>cRate;

//анализ на въведеното
значение
 switch (cRate) {
case 'A':
case 'a':
        // оценка A или a равно на 5
        cout<<"Your rate is 5\n";
        break;
case 'B':
case 'b':
        // оценка B или b равно на 4
        cout<<"Your rate is 4\n";
        break;
case 'C':
case 'c':
        // оценка C или c равно на 3
        cout<<"Your rate is 3\n";
        break;
case 'D':
case 'd':
        // оценка D или d равно на 2
        cout<<"Your rate is 2\n";
        break;
default:
        // останалите символи са
        некоректни
         cout<<"This rate isn't
        correct\n";
    }
}
```

Примера е отличителен с това, че с помощта на идващите подред case се постига регистронезависимост. Тоест, не е важно каква именно буква ще въведе потребителя, главна или

малка.

**Разпространена грешка.**

Всичко най-важно за оператора switch е казано, остава само да получим информация за това с какъв проблем може да се сблъсне програмиста използвайки даден оператор.

Ако случайно се изпусне break в който и да е блок на case освен в последния, и този блок в последствие заработи, то изпълнението на switch няма да спре. Този блок на оператора case, който ще бъде след вече горе-изпълнения, също така ще се изпълни без проверка.

## Пример за грешка.

```
# include <iostream>
using namespace std;
void main(){

// реализация на програмно меню
int action;
cout<<"\nSelect action:\n";
cout<<"\n 1 - if you want to see course of dollar.\n";
cout<<"\n 2 - if you want to see course of euro.\n";
cout<<"\n 3 - if you want to see course of rub.\n";

//очакване избора на
потребителя
 cin>>action;

//проверява се значението на
променливата  action
switch(action){
case  1:     // ако потребителя е избрал долар
        cout<<"\nCource: 5.2 gr.\n";
        break; // стоп switch
case  2:      // ако потребителя е избрал евро
        cout<<"\nCource: 6.2 gr.\n";
         //break;  подчертано спиране  switch case
3:           // ако потребителя е избрал рубли
        cout<<"\nCource: 0.18 gr.\n";
        break; // стоп switch
default:     // ако избора е некоректен
        cout<<"\nError!!! This operator isn't correct\n";
        break; // стоп switch
 }
}
```

Грешка ще се случай че бъде избран пункт 2 от менюто. В case със значение 2 е поставен оператор за спиране break. На екрана резултата от такава грешка изглежда по следния начин:

```
Select action:

 1 - if you want to see course of dollar.

 2 - if you want to see course of euro.

 3 - if you want to see course of rub.
2

Cource: 6.2 gr.

Cource: 0.18 gr.
Press any key to continue
```

Освен необходимата информация на екрана се показа и това което се е намирало в блока case след обикновена конструкция. Трябва да се избягват такива разпечатки, защото те водят до грешки на етапа за изпълнение.

В днешния урок ние с вас се запознахме с операторите позволяващи да се проведе анализ на каквито и да е данни. Сега вие можете да преминете към изпълнението на домашната работа. Желаем Ви успех!

# 11. Домашна работа

1. Напишете програма която проверява числа въведени с клавиатурата за четност.
2. Дадено е естествено число a (a<100). Напишете програма която изкарва на екрана количеството цифри в това число и сумата на тези цифри.
3. Известно е, че 1 инч е равен на 2.54см. Разработете приложение което конвентира инчове в сантиметри и обратното. Диалога с потребителя да се реализира чрез системно меню.
4. Напишете програма, реализираща популярната телевизионна игра „Стани богат“.