



ОСНОВИ
ПРОГРАМУВАННЯ
НА МОВІ C++

Урок № 2

УМОВИ

ЗМІСТ

1. Поняття оператора.....	4
Пріоритет.....	5
2. Арифметичні операції з числами	6
Добре забуте старе.....	6
Інкремент і декремент	7
3. Застосування арифметичних операцій	12
4. Перетворення типів	15
Класифікація за діапазоном значень	16
Класифікація за способом здійснення перетворення.....	17
Перетворення типів у виразі	18
Приклад, який використовує перетворення типів.....	19
Уніфікована ініціалізація	20
Звуження та спискова ініціалізація	21
5. Логічні операції	22

Оператори порівняння.....	22
Оператори рівності	23
Логічні операції об'єднання та негативна інверсія	24
Логічне І (&&)	24
Логічне АБО ()	25
Логічне НЕ (!).....	27
6. Конструкція логічного вибору if.....	29
Основні принципи роботи оператора if	30
Тернарний оператор.....	35
7. Драбинка if — else if	38
8. Практичний приклад створення текстового квесту	47
9. Практичний приклад на приналежність точки кільцю.....	51
10. Структура множинного вибору switch	54
Загальний синтаксис і принцип дії	56
Поширена помилка	62
11. Поняття enum як перерахування	65
11. Домашнє завдання	75

Додаткові матеріали уроку прикріплені до даного PDF-файлу.
Для доступу до матеріалів, урок необхідно відкрити в програмі
Adobe Acrobat Reader.

1. Поняття оператора

У минулому уроці ви познайомилися з поняттям змінна та тип даних. Крім того, у прикладах уроку, а також у домашньому завданні, ми з вами виконували певні дії над змінними, тобто оперували даними. Цілком очевидно, що слова «оператор» та «оперувати» мають однакове походження, отже, згідно з простою логікою:

Оператор — конструкція мови, що дає змогу виконувати різні дії над даними, що призводять до певного результату.

Усі оператори прийнято поділяти на групи за ознакою їхньої дії. Наприклад, арифметичні операції — операції, що дають змогу виконувати арифметичні дії над даними (додавання, віднімання і так далі). Про всі подібні групи, представлені в мові Сі, ми будемо розповідати надалі. На даний момент треба обговорити більш масштабну класифікацію всіх операторів, прийняту незалежно від їхнього впливу на вміст змінних. Отже, усі оператори діляться на:

1. **Унарні** — оператори, які потребують тільки одного операнда (дані, над якими виконується дія). З прикладом унарного оператора ви вже знайомі з курсу шкільної математики — унарний мінус, який перетворює число на від'ємне (3 і -3), або додатне ($-(-3)$). Тобто загальний синтаксис унарного оператора такий:

оператор операнд;	або	операнд оператор;
-------------------	-----	-------------------

2. **Бінарні** — оператори, які потребують двох операндів зліва і справа від оператора. Таких операторів ви зна-

єте безліч — це $+$, $-$, $*$ тощо. Їхній загальний синтаксис можна зобразити так:

операнд оператор операнд;

3. **Тернарні** — оператори, які потребують трьох операндів. У мові програмування C++ такий оператор лише один, і з його синтаксисом ми познайомимось трохи пізніше.

Пріоритет

Усі оператори мають пріоритет. Нижче наведені оператори відповідно до пріоритетів. Більш поглиблено ми познайомимось з деякими в сьогоднішньому уроці, інші дізнаємось в процесі подальшого навчання. Природно, у цій таблиці представлені не всі оператори мови, а поки що найбільш актуальні для нас.

Таблиця 1

Символьне позначення операції	
Найвищий пріоритет	Найвищий пріоритет
$() [] . ->$	$^$
$! *(ун) - (ун) \sim ++ --$	$ $
$\% * /$	$\&\&$
$+ -$	$ $
$<< >>$	$?:$
$< > <= >=$	$= += -= *= /= \% \& = \wedge = >>= <<=$
$!= ==$	
$\&$	Найнижчий пріоритет

Тепер, коли фундамент знань в області операторів закладений, ви можете переходити до більш докладного вивчення останніх, а саме до наступного розділу уроку.

2. Арифметичні операції з числами

Добре забуте старе...

Отже, розпочинаємо. Як уже було зазначено раніше, арифметичні операції — це операції, що дозволяють виконувати арифметичні дії над даними. Більшість із них вам знайома з дитинства, однак давайте систематизуємо наші знання за допомогою таблиці нижче.

Таблиця 2

Назва операції	Символ, для позначення в мові Сі	Короткий опис. Приклад.
Додавання	+	Додає два значення, результатом є сума операндів: $5 + 18$ результат 23.
Віднімання	—	Віднімає значення, що знаходиться справа, від значення, що знаходиться зліва від оператора. Результат — різниця операндів: $20 - 15$ результат 5.
Множення	*	Перемножує два значення, результатом є добуток операндів: $5 * 10$ результат 50.
Ділення	/	Ділить значення, що знаходиться зліва, на значення, що знаходиться справа від оператора. Наприклад: $20/4$ результат 5.
Ділення з остачею (за модулем)	%	Результатом цієї операції є остача від цілочисельного ділення, наприклад, якщо ми ділимо 11 на 3, то цілих частин у нас виходить 3, (тому що $3*3=9$), остачею буде 2, це число й буде результатом ділення з остачею: $11/3$ 3 цілих 2 остача $11 \% 3 = 2$ (остача).

Примітка:

1. Операцію ділення за модулем, можна застосовувати тільки до цілочисельних даних. Спроби порушити дане правило приведуть до помилки на етапі компіляції.
2. Якщо менше число ділиться на більше за допомогою %, то результатом буде саме менше число.
 $3\%10 = 3$.
3. Ділити за модулем на нуль не можна, це призведе до некоректної роботи програми на етапі виконання.

Інкремент і декремент

Усі вищеописані операції були бінарними, однак існують ще й унарні арифметичні операції, таких операцій у шкільному курсі немає, хоча насправді вони дуже прості:

1. **Інкремент** позначається конструкцією `++`. Цей оператор збільшує вміст будь-якої змінної на одиницю та перезаписує значення змінної. Наприклад:

```
int a=8;
cout<<a; // на екрані число 8

a++;
cout<<a; // на екрані число 9
```

2. **Декремент** позначається конструкцією `--`. Цей оператор зменшує вміст будь-якої змінної на одиницю та перезаписує значення змінної. Наприклад:

```
int a=8;
cout<<a; // на екрані число 8

a--;
cout<<a; // на екрані число 7
```

Досить просто, чи не так?! Такі вирази можуть бути представлені й так: $a=a+1$ або $a=a-1$. Треба зазначити, що для літералів ані інкремент, ані декремент не використовуються, тому що абсолютно нелогічно чинити так: $5=5+1$. Це явна помилка. Однак на цьому ми не закінчимо знайомство з інкрементом і декрементом. У минулому розділі уроку ми з'ясували, що синтаксис унарного оператора може бути не тільки таким:

```
операнд оператор;
```

але й таким:

```
оператор операнд;
```

Такі форми запису зветься постфіксною, (оператор розташовується після значення) і префіксною (оператор розташовується перед значенням). І інкремент, і декремент мають обидві форми. Давайте розберемося, які є відмінності між формами, і в яких випадках ці відмінності мають значення.

Приклад 1

```
int a=8;
cout<<a; // на екрані число 8
a++;
```



```
cout<<a; // на екрані число 9
++a;
cout<<a; // на екрані число 10
```

У цьому прикладі немає жодної різниці між префіксною та постфіксною формами. І в першому, і в другому випадку значення змінної `a` просто збільшується на одиницю.

Сенс використовувати різні форми оператора з'являється тільки тоді, коли в рядку, окрім самого оператора, є ще якась команда.

Приклад 2

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

Перш ніж розбирати приклад, давайте встановимо три правила:

1. Принцип виконання команд у мові C++ неоднозначний. Тому нижче наводиться таблиця напрямку дії деяких операторів.
2. Якщо крім постфіксної форми інкремента або декремента рядок має ще якусь команду, то спочатку виконується ця команда, і тільки потім інкремент або декремент незалежно від розташування команд у рядку.
3. Якщо крім префіксної форми інкремента або декремента рядок має ще якусь команду, то всі команди в рядку виконуються справа наліво відповідно до пріоритету операторів.

Таблиця 3

ОПЕРАТОР	НАПРЯМОК
<code>() [] . -></code>	ЗЛІВА НАПРАВО
<code>* / % + -</code>	
<code><< >> & ^ </code>	
<code><< = >> = == !=</code>	
<code>&& </code>	
Унарний – унарний <code>++ --</code>	СПРАВА НАЛІВО
<code>?:</code>	
<code>+ = - = / = * = % = & = ^ = =</code>	

Тепер докладніше про приклад:

- Спочатку значення змінної дорівнює числу 8.
- Команда `cout<<+a;` містить префіксну форму оператора інкремент, отже, використовуючи третє правило, описане вище, ми спочатку збільшуємо значення змінної `a` на одиницю, а потім показуємо його на екран за допомогою команди `cout<<.`
- Команда `cout<<a++;` містить постфіксну форму оператора інкремент, отже, використовуючи друге правило, описане вище, ми спочатку показуємо значення змінної (все ще 9) на екран за допомогою команди `cout<<.`, а потім збільшуємо значення змінної `a` на одиницю.
- При виконанні команди `cout<<a;` буде показано вже змінене (збільшене) значення, тобто число 10.

З огляду на попередні теми цього розділу уроку, ми з вами тепер знаємо, як спростити незручний і «некрасивий» запис типу `x=x+1` або `x=x-1`, перетворивши його на `x++`, або `x--`. Але таким чином ми можемо збільшувати

і зменшувати значення змінної лише на одиницю, а як бути з будь-яким іншим числом? Наприклад, як спростити запис:

$$X=X+12.$$

У цьому випадку теж є просте рішення — використувати так звані комбіновані оператори або скорочені арифметичні форми. Виглядають вони так:

Таблиця 4

Назва форми	Комбінація	Стандартний запис	Скорочений запис
Присвоєння з множенням	<code>*=</code>	<code>A=A*N</code>	<code>A*=N</code>
Присвоєння з діленням	<code>/=</code>	<code>A=A/N</code>	<code>A/=N</code>
Присвоєння з діленням за модулем	<code>%=</code>	<code>A=A%N</code>	<code>A%=N</code>
Присвоєння з відніманням	<code>- =</code>	<code>A=A-N</code>	<code>A-=N</code>
Присвоєння з додаванням	<code>+=</code>	<code>A=A+N</code>	<code>A+=N</code>

Ми рекомендуємо вам надалі користуватися скороченими формами, адже це не тільки є хорошим тоном у програмуванні, але й значно підвищує читабельність програмного коду. Крім того, у деяких джерелах згадується про те, що скорочені форми обробляються комп'ютером швидше, підвищуючи швидкість виконання програми. Тепер саме час переконатися у всьому вищесказаному на практиці, тому що, як то кажуть, краще один раз побачити, ніж сто разів почути. Ви вже вмієте створювати проекти і додавати в них файли, власне, саме це від вас зараз і потрібно. Далі представлено кілька програм, які вам необхідно набрати, щоб побачити застосування арифметичних операцій на практиці. Почнемо з проекту під назвою Game.

3. Застосування арифметичних операцій

Приклад №1. Гра

```
// примітивна гра для малюків
#include <iostream>
using namespace std;
int main()
{
    int buddies; // кількість піратів до битви
    int afterFight; // кількість піратів після битви

    // Ви пірат. Скільки людей у вашій команді,
    // якщо не брати до уваги вас?
    cout<< "You are a pirate."
           "How many people are there in"
           "your squad, excluding you?\n\n";
    cin>>buddies;

    // Раптово на вас нападає 10 мушкетерів
    cout<<"Suddenly you are attacked by 10 "
          "musketeers \n\n";

    // 10 мушкетерів і 10 піратів гинуть у сутичці.
    cout<<"10 musketeers and 10 pirates are killed "
          "in the fight.\n\n";

    // підрахунок тих, хто залишився в живих
    afterFight=1+buddies-10;

    // Залишилося лише ... піратів
    cout<<"There are only "<<afterFight<<
          " pirates left\n\n";
```

```

// Стан убитих налічує 107 золотих монет
cout<<"The killed ones have a total of 107 gold "
    "coins \n\n";
// Це по ... монет на кожного
cout<<"Which is "<<(107/afterFight)<<"coins per "
    "person";
// Пірати влаштовують велику бійку через
cout<<"Pirates put up a big fight because "
    "of the remained\n\n";
// ... монет, що залишилися
cout<<(107%afterFight)<<"coins \n\n";
return 0;
}

```

У цьому прикладі використовується правило ділення цілого на ціле. При такому діленні дрібна частина, навіть якщо має бути, обрізається. Більш докладно про це буде розказано в розділі уроку — «Перетворення типів». У виразі `(107/afterFight)` ми дізнаємося, скільки монет отримає кожен пірат, якщо розділити їх порівну. Крім того, оператор ділення за модулем допомагає нам з'ясувати, скільки залишиться монет, які неможливо розділити, тобто ми отримаємо залишок від ділення 107 на кількість тих піратів, що вижили. Ось і всі особливості прикладу.

Приклад №2. Окружність

У цьому прикладі буде продемонстровано використання арифметичних операторів у програмах, які виконують математичні обчислення.

Ми переконуємося, що знання арифметичних операторів дає змогу розв'язувати прості задачі. Однак мало вміти використовувати оператори, необхідно ще

розуміти, яким буде результат їхнього використання. Про це й піде мова в наступному розділі.

```
// програма для з'ясування параметрів окружності
#include <iostream>
using namespace std;

int main()
{
    const float PI=3.141592; // позначення константи —
                             // числа пі
    // оголошення змінних для зберігання параметрів
    float radius, circumference, area;
    // запрошення ввести радіус
    cout<<"Welcome to the program for working with"
         "circles\n\n";
    cout<<"Enter the radiuses of your circles\n\n";
    cin>>radius;
    cout<<"\n\n";
    area=PI*radius*radius; // підрахунок площі кола
    circumference=PI*(radius*2); // підрахунок довжини
                                // окружності

    // вивід результатів
    cout<<"The area of the circle is "<<area<<"\n\n";
    cout<<"The circumference is "<<circumference<<
         "\n\n";
    cout<<"THANKS!!! BYE!!!\n\n";
    return 0;
}
```

4. Перетворення типів

Коли, ми що-небудь робимо, нам, безсумнівно, важливо знати яким буде результат. Цілком очевидно, що з тих інгредієнтів, з яких, скажімо, вариться суп харчо, навряд чи можна приготувати торт зі збитими вершками. Отже, результат безпосередньо залежить від складових частин.

Те саме відбувається зі змінними. Якщо, скажімо, складається два числа типу `int`, цілком зрозуміло, що результат так само буде мати тип `int`. Але як бути, якщо дані мають різні типи? Саме про це ми й поговоримо в поточному розділі цього уроку.

Отже, найперше давайте розберемося з тим, як типи даних взаємодіють один з одним. Є так звана ієрархія типів, де всі типи розташовані по старшинству. Аби розбиратися в перетворенні типів, необхідно завжди пам'ятати порядок типів цієї ієрархії.

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

Незважаючи на те, що деякі типи мають однаковий розмір, у них міститься різний діапазон значень, наприклад, `unsigned int`, на відміну від `int`, може вмістити в себе вдвічі більше додатних значень, і тому є старшим за ієрархією, при цьому обидва типи мають розмір 4 байти. Крім того, треба зазначити дуже важливу особливість, відображену тут — якщо в перетворенні типів беруть участь такі типи,

як `bool`, `char`, `short`, вони автоматично перетворюються на тип `int`.

Тепер давайте розглянемо різні класифікації перетворень.

Класифікація за діапазоном значень

Усі перетворення можна розділити на дві групи щодо розташування в ієрархії типів, що беруть участь у перетворенні.

1. Звужуюче перетворення — при такому перетворенні більший тип даних у ієрархії перетворюється на менший тип, безумовно, у цьому разі може відбутися втрата даних, тому зі звужуючим перетворенням потрібно бути обережними. Наприклад:

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

2. Розширююче перетворення. Цей вид перетворення веде до так званого розширення типу даних — від меншого діапазону значень до більшого. Як приклад пропонується така ситуація.

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

У цьому випадку `3000000000` — це літерал типу `int`, який безперешкодно розширюється до `unsigned int`, що й дозволяє нам побачити на екрані саме `3000000000`, а не щось інше. Тоді як у звичайний `int` таке число не поміститься.

Класифікація за способом здійснення перетворення

Незалежно від напрямку перетворення, воно може бути здійснено одним із двох способів.

1. Неявне перетворення. Усі вищеописані приклади належали до цього типу перетворення. Такий вид перетворення також називають автоматичним, тому що воно відбувається автоматично без втручання програміста, іншими словами, ми нічого не робимо для того, щоб воно сталося.

```
float A=23.5; — double став float без
                будь-яких додаткових дій
```

2. Явне перетворення (друга назва — приведення типів). У цьому випадку, перетворення виконує програміст тоді, коли це необхідно. Давайте розглянемо простий приклад такої дії:

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

`(Int) z` — це явне, звужуюче перетворення від типу `double` до типу `int`. У цьому ж рядку відбувається розширююче, неявне перетворення від отриманого типу `int` до типу `float`. Треба запам'ятати, що будь-яке перетворення носить тимчасовий характер і діє тільки в межах поточного рядка. Тобто змінна `z` як була `double`, так і залишиться протягом усієї програми, а її перетворення на `int` носило тимчасовий характер.

Перетворення типів у виразі

Ось ми нарешті підійшли до того, про що говорили на самому початку цього розділу уроку, — як з'ясувати, якого типу буде результат певного виразу. Давайте спробуємо це обчислити, користуючись отриманими знаннями. Припустимо, ми маємо такі змінні:

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

Користуючись цими змінними, ми збираємося скласти такий вираз:

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

У змінну якого типу даних нам треба записати результат? Відповісти на це просто, якщо представити вираз у вигляді типів даних:

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

Нагадуємо, що **short** і **bool** відразу ж візьмуть тип **int**, отже вираз виглядатиме так:

```
int-float+int*int, при цьому false стане 0
```

Множення **int** на **int** дасть, безсумнівно, результат типу **int**. А от додавання **float** до **int** дасть на виході **float**, адже тут вступає в гру нове правило:

Якщо в будь-якому виразі використовуються різні типи даних, то результат приводиться до більшого з цих типів.

Ну й нарешті — віднімання від `int` типу `float`, згідно зі щойно згаданим правилом, знову дасть `float`.

Так, результат виразу матиме тип `float`.

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

Тепер, коли ви знайомі з правилом, немає необхідності докладно розбирати вираз, вистачить лише знайти найбільший тип, саме він і буде результируючим.

Примітка: *Будьте дуже уважні також при поєднанні змінних з однаковими типами даних. Наприклад, пам'ятайте — якщо ціле ділиться на ціле, то і вийде ціле. Тобто `int A=3; int B=2; cout<<A/B; //` на екрані 1, тому що результат — `int` і дробову частину втрачено. `cout<<(float) A/B; //` на екрані 1.5, тому що результат — `float`.*

Приклад, який використовує перетворення типів

Тепер давайте закріпимо знання на практиці. Створимо проєкт і напишемо нижченаведений код.

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

    // оголошення змінних і запит на ввід даних
    float digit;
    cout<<"Enter a digit:";
    cin>>digit;
    /* Навіть якщо користувач ввів число
```

```

        з дійсною частиною, результат виразу
        запишеться в int, і дійсна частина буде
        втрачена; розділивши число на 100, ми отримаємо
        кількість сотень у ньому. */
int res=digit/100;
cout<<res<<"There are this many hundreds"
        "in your digit!!!\n\n";
return 0;
}

```

Тепер, коли ви розглянули приклад, ви, звичайно ж, переконалися, що за допомогою перетворення типів можна не просто організувати тимчасовий перехід з одного типу в інший, але й розв'язати просту логічну задачу. Отже, ви маєте поставитися до цієї теми з належною увагою. Розуміння перетворення надалі допоможе вам не тільки виконувати цікаві завдання, але й уникати непотрібних помилок.

Уніфікована ініціалізація

У C++ 11 був доданий механізм уніфікованої ініціалізації, який дає змогу задати значення різним програмним конструкціям (змінним, масивам, об'єктам) однаковою способом. Розглянемо на прикладі ініціалізації змінних.

```

int a = {11};    // В a записується значення 11
int b{33};       // В b записується значення 33

```

Щоб задати значення змінним, ми використовуємо `{}`. Як видно з прикладу, це можна зробити двома способами. Така форма ініціалізації також називається **списковою ініціалізацією**.

Звуження та спискова ініціалізація

Що станеться при виконанні коду?

```
int x = 2.88;
cout<<x; // на екрані відобразиться 2
```

Як ви вже знаєте з вивченого матеріалу, у цьому прикладі відбувається неявне звужуюче перетворення, тому що ми присвоюємо змінній `x` цілого типу значення типу `double`. Однак, якщо використовувати **спискову ініціалізацію**, компілятор генерує помилку на етапі компіляції, тому що ця форма ініціалізації захищає від звуження. Вона не дає записати значення більшого розміру в тип, який не підтримує такий діапазон значень.

```
int x = { 2.88 }; // помилка на етапі компіляції.
                  // 2.88 – double, а x змінна
                  // цілого типу
char ch = { 777 }; // помилка на етапі компіляції.
                  // 777 – int, а ch змінна символного
                  // типу, тож 777 потрапляє
                  // в діапазон значень char
```

З іншого боку:

```
char ch2 = { 23 }; // все правильно 23 потрапляє
                  // у діапазон char
double x = { 333 }; // все правильно 333 – int
                   // і потрапляє в діапазон double
```

Якщо ви хочете виявляти потенційні проблеми з втратою даних на етапі компіляції, ви можете використовувати спискову ініціалізацію.

5. Логічні операції

У програмуванні частенько необхідно не тільки виконувати якісь обчислення, а й порівнювати величини між собою. Для цього використовуються так звані логічні операції. Результатом логічних операцій завжди є або значення **true**, або значення **false**, тобто істина або брехня. Логічні операції діляться на три підгрупи:

1. Оператори порівняння;
2. Оператори рівності;
3. Логічні оператори об'єднання та негативна інверсія.

Тепер давайте більш докладно розберемо кожен групу операторів.

Оператори порівняння

Використовуються тоді, коли необхідно з'ясувати, як дві величини відносяться одна до одної.

Таблиця 5

Символ, що позначає оператор	Твердження
<	Лівий операнд менший за правий операнд
>	Лівий операнд більший за правий операнд
<=	Лівий операнд менший або дорівнює правому операнду
>=	Лівий операнд більший або дорівнює правому операнду

Сенс операцій порівняння (друга назва — операції відношення) полягає в тому, що якщо твердження, задане

за допомогою оператора, вірне, вираз, у якому він бере участь, буде замінений на значення **true**, якщо невірне — на значення **false**. Наприклад:

```
cout<<(5>3); // на екрані буде одиниця, тому що
              // твердження (5>3) — істина.
cout<<(3<2); // на екрані буде 0, тому що (3<2) — брехня
```

Примітка: Замість значень **false** і **true** на екран виводиться 0 і 1, тому що вони еквівалентні значенням брехня й істина. У мові C++ в ролі істини також може виступати будь-яке інше число відмінне від 1 і 0, як додатне, так і від'ємне.

Оператори рівності

Використовуються для перевірки на повну відповідність або невідповідність двох величин.

Таблиця 6

Символ, що позначає оператор	Твердження
==	Лівий операнд дорівнює правому
!=	Лівий операнд НЕ дорівнює правому

Застосування цих операторів збігається з принципом застосування попередньої групи, тобто на виході вираз замінюється або на істину, або на брехню, залежно від твердження.

```
cout<<(5!=3); // на екрані буде одиниця,
              // тому що твердження (5!=3) — істина.
cout<<(3==2); // на екрані буде 0, тому що (3==2) —
              // брехня.
```

Логічні операції об'єднання та негативна інверсія

Здебільшого неможливо обійтися тільки одним твердженням. Найчастіше необхідно комбінувати твердження тим чи іншим чином. Наприклад, щоб перевірити, чи знаходиться число в діапазоні від 1 до 10, необхідно перевірити два твердження: число має одночасно ≥ 1 і ≤ 10 . Аби реалізувати таку комбінацію, необхідно ввести додаткові оператори.

Таблиця 7

Операція	Назва
&&	І
	АБО
!	НЕ

Логічне І (&&)

Логічне І об'єднує два твердження і повертає істину тільки в тому випадку, якщо і ліве, і праве твердження є істинними. Якщо хоча б одне з тверджень або обидва є брехнею, об'єднаний вираз замінюється на брехню. Логічне І працює за скороченою схемою, тобто, якщо перше твердження — брехня, друге вже не перевіряється.

Таблиця 8

Твердження 1	Твердження 2	Твердження1 && Твердження2
true	true	true
true	false	false
false	true	false
false	false	false

Тепер розглянемо приклад, у якому програма отримує число і визначає, чи потрапляє це число в діапазон від 1 до 10.

```
#include <iostream>
using namespace std;

int 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";
    return 0;
}
```

У цьому прикладі, якщо обидва твердження будуть вірними, на місце виразу підставиться 1, в іншому випадку — 0. Відповідно, користувач зможе проаналізувати ситуацію, що склалася, використовуючи інструкції програми.

Логічне АБО (||)

Логічне АБО об'єднує два твердження і повертає істину тільки в тому випадку, якщо хоча б одне з тверджень є вірним, і брехню в тому випадку, якщо обидва твердження невірні. Логічне АБО працює за скороченою схемою, тобто, якщо перше твердження — істина, друге вже не перевіряється.

Таблиця 9

Твердження 1	Твердження 2	Твердження1 Твердження2
true	true	true
true	false	true
false	true	true
false	false	false

Ще раз розглянемо приклад, у якому програма отримує число і визначає, чи потрапляє це число в діапазон від 1 до 10. Тільки тепер використаємо АБО.

```
#include <iostream>
using namespace std;

int main()
{
    int N;
    cout<<"Enter your number:\n";
    cin>>N;
    cout<<((N<1) || (N>10));
    cout<<"\n\nIf you see 0, your number is in "
         "the range\n\n";
    cout<<"\n\nIf you see 1, your number is out of the"
         "the range\n\n";
    return 0;
}
```

У цьому прикладі, якщо обидва твердження будуть брехнею (тобто число буде не менше ніж 1 і не більше ніж 10), на місце виразу підставиться 0, в іншому випадку — 1.

Відповідно, користувач, як і в попередньому прикладі, зможе проаналізувати ситуацію, що склалася, і зробити висновок.

Логічне НЕ (!)

Логічне НЕ є унарним оператором і у зв'язку з цим не може називатися оператором об'єднання. Воно використовується в тому випадку, якщо потрібно змінити результат перевірки твердження на протилежний.

Таблиця 10

Твердження	! Твердження
true	false
false	true

```
// на екрані буде 1, тому що (5==3) – брехня і її
// інверсія – це істина.
cout<<! (5==3) ;
// на екрані буде 0, тому що (3!=2) – істина і її
// інверсія – це брехня.
cout<<! (3!=2) ;
```

Логічне заперечення повертає на місце твердження брехню, якщо останнє є істиною, і навпаки істину, якщо твердження є брехнею. Цей оператор можна застосувати для скорочення постановки умови. Наприклад, вираз

```
b==0
```

можна скорочено записати за допомогою інверсії:

```
!b
```

обидва записи дають на виході істину в разі, якщо **b** дорівнюватиме нулю.

У цьому розділі ми розглянули всілякі логічні операції, які дозволяють визначити істинність будь-якого

твердження. Однак описані тут приклади є незручними для рядового користувача, тому що аналіз результатів має виконувати не він, а програма. Крім того, якщо, залежно від твердження, необхідно не просто видавати на екран результат його перевірки, а виконувати будь-яку дію, тут уже користувач точно безсилий. У зв'язку з цим, володіючи знаннями логічних операцій, необхідно отримати додаткову інформацію для можливості реалізації тієї чи іншої дії залежно від умови. Саме про це й піде мова в наступному розділі нашого уроку.

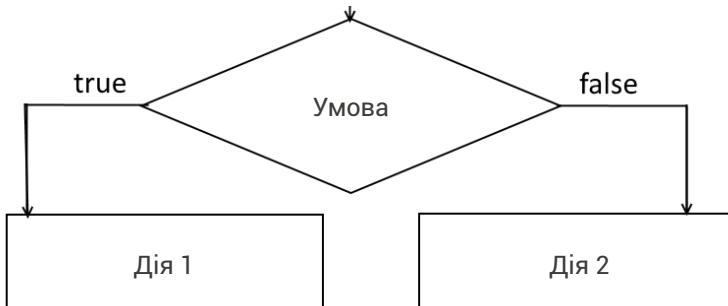
6. Конструкція логічного вибору if

Зараз ми з вами познайомимося з оператором, який дає змогу перетворити звичайну лінійну програму на програму «мислячу». Цей оператор перевіряє якесь твердження (вираз) на істинність і, залежно від отриманого результату, виконує ту чи іншу дію.

Для початку розглянемо загальний синтаксис цього оператора:

```
if (твердження або вираз)
{
    дія 1;
}

else
{
    дія 2;
}
```



Малюнок 1

Основні принципи роботи оператора if

Як твердження або вираз може виступати будь-яка конструкція, що містить логічні оператори або ж арифметичний вираз.

- **if (X > Y)** — звичайне твердження, буде істинне, якщо **X** дійсно більше за **Y**

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

- **if (A > B && A < C)** — комбіноване твердження, що складається з двох частин, буде істиною, якщо обидві частини будуть вірними.

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

- **if (A - B)** — арифметичний вираз, буде істинним, якщо **A** не дорівнює **B**, тому що в іншому випадку (якщо вони рівні) їхня різниця дасть нуль, а нуль — це брехня.

```
int A=10,B=15;
if (A-B) {
    // -5 це істина
    cout<<"A != B"; // на екрані A != B
}
```

- `if(++A)` — арифметичний вираз буде істинним, якщо `A` не дорівнює `-1`, тому що, якщо `A` дорівнює `-1`, збільшення на `1` дасть нуль, а нуль — це брехня.

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

- `if(A++)` — арифметичний вираз, буде істинним, якщо `A` не дорівнює `0`, тому що в цьому випадку використовується постфіксна форма інкремента, спочатку відбудеться перевірка умови і буде виявлений нуль, а потім збільшення на одиницю.

```
int A=0;
if(A++){ // 0 це брехня
    cout<<"Best test!!"; // цю фразу ми не побачимо,
                        // тому що if не виконається
}
```

- `if(A==Z)` — звичайне твердження, буде істинним, якщо `A` дорівнює `Z`.
- `if(A=Z)` — операція присвоєння, вираз буде істинним, якщо `Z` не дорівнює нулю.

Примітка: Типова помилка. Дуже часто замість операції перевірки на рівність `==` через неуважність вказується операція присвоєння `=`, і зміст виразу може радикально змінитися. Така банальна помилка може призвести до некоректної роботи всієї програми. Розглянемо два, здавалося б, ідентичні приклади.

Правильний приклад

```
#include <iostream>
using namespace std;
int main(){
    int A,B; // оголосимо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;
    if (B==0){ // якщо B містить нуль, повідомляємо
                // про помилку
        cout<<"You can't divide by zero!!!";
    }
    else{ // в іншому випадку
        cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;
        // видаємо результат ділення A на B
    }
    cout<<"\n The end. \n"; // кінець
    return 0;
}
```

Приклад з помилкою

```
#include <iostream>
using namespace std;
int main(){
    int A,B; // оголосимо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;
    // Прирівнюємо B до нуля і перевіряємо умову,
    // вона автоматично є брехнею
```



```

if (B=0) { // ця частина ніколи не виконається,
           // тому що умова завжди є брехнею
    cout<<"You can't divide by zero!!!";

    // повідомляємо про помилку
}
else { // завжди виконується ця частина,
       // у якій A ділиться на новоспечений нуль
       /* У цьому рядку відбудеться помилка на етапі
        виконання, тому що комп'ютер спробує
        поділити число на нуль */
    cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;
}
cout<<"\n The end. \n"; // Цю фразу ми не побачимо
                        // ніколи.

return 0;
}

```

Як ви вже встигли помітити, якщо вміст круглих дужок буде істиною, то виконається дія 1, вписана у фігурні дужки конструкції **if**, при цьому дію 2 блока **else** буде проігноровано.

Якщо ж вміст круглих дужок є брехнею, виконається дія 2, вписана у фігурні дужки конструкції **else**, при цьому дію 1 буде проігноровано.

Конструкція **else** є необов'язковою. Це означає, що якщо немає необхідності робити що-небудь при хибності твердження, дану конструкцію можна не вказувати. Наприклад, програму, яка використовує захист проти поділу на нуль, можна записати так:

```

#include <iostream>
using namespace std;

```

```

int main(){
    int A,B; // оголосимо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;
    if(B!=0){ // якщо B не дорівнює нулю,
        cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;
        // виконуємо обчислення
    }
    // в іншому випадку не робимо нічого
    cout<<"\nThe end.\n";
    return 0;
}

```

Якщо до блока **if** або **else** належить тільки одна команда, то фігурні дужки можна не вказувати. За допомогою цього правила зробимо програму ще коротшою:

```

#include <iostream>
using namespace std;
int main(){
    int A,B; // оголосимо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;
    if(B!=0) // якщо B не дорівнює нулю,
        // виконуємо обчислення
    cout<<"Result A/B="<<A<<"/"<<B<<"="<<A/B;
    // в іншому випадку не робимо нічого
    cout<<"\nThe end.\n";
    return 0;
}

```

Ми щойно познайомилися з умовним оператором `if` і обговорили основні принципи його дії. Перш ніж переходити до розгляду специфічних особливостей `if` і практичних прикладів, зробимо невеликий відступ і подивимося на ще один оператор, за допомогою якого можна поставити просту умову.

Примітка. *Будьте уважні: оператор `if` та оператор `else` нерозривні! Спроба вписати між ними рядок коду призведе до помилки на етапі компіляції.*

Фрагмент коду з помилкою

```
....
    if (B==0) { // якщо B містить нуль,
                // повідомляємо про помилку
                cout<<"You can't divide by zero!!!";
            }
    cout<<"Hello";// Помилка!!!! Розрив конструкції
                    // if – else!!!
    else{ // в іншому випадку
        cout<<"Result = "<<A/B;// видаємо результат
                                // ділення A на B
    }
....
```

Тернарний оператор

Деякі умови є дуже примітивними. Наприклад, візьмемо нашу програму ділення двох чисел. Вона проста і з точки зору дій, і з точки зору коду. На оператори `if` та `else` випадає по одному рядку коду — дії. Таку програму, можна спростити ще більше, використовуючи тернарний оператор.

Для початку розглянемо його синтаксис:

ТВЕРДЖЕННЯ АБО ВИРАЗ?ДІЯ1:ДІЯ2;

Принцип дії простий — якщо ТВЕРДЖЕННЯ АБО ВИРАЗ — істина, виконується ДІЯ 1, якщо — брехня, виконується ДІЯ 2.

Давайте розглянемо дію цього оператора:

```
#include <iostream>
using namespace std;
int main() {
    int A,B; // оголосимо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n"
    cin>>B;
    /* У цьому випадку, якщо B не дорівнюватиме нулю,
    виконається та команда, яка стоїть після
    знака питання, і на екрані з'явиться результат
    ділення. В іншому випадку, виконається
    команда, яка стоїть після двокрапки, і на
    екрані буде повідомлення про помилку ділення
    на нуль.
    */
    (B!=0)?cout<<"Result A/B="<<A<<"/!"<<B<<"="<<
        A/B:cout<<"You can't divide by zero!!!";
    // кінець програми
    cout<<"\n The end. \n";

    return 0;
}
```

Хіба код не став ще оптимальнішим?! Для закріплення отриманої інформації наведемо ще один, більш складний,

приклад. Програма буде визначати, яке з двох чисел, введених користувачем, є більшим, а яке меншим.

```
#include <iostream>
using namespace std;

int main(){
    int a,b; // оголошуємо дві змінні
    // просимо користувача ввести в них дані
    cout<<"Enter your first number:\n";
    cin>>a;
    cout<<"Enter your second number:\n";
    cin>>b;
    /* Якщо (b>a), то на місце оператора ?: підставиться b,
       в іншому випадку, на місце оператора підставиться a,
       так, більше число запишеться в змінну max.
    */
    int max=(b>a)?b:a;

    /* Якщо (b<a), то на місце оператора ?: підставиться b,
       в іншому випадку, на місце оператора підставиться a,
       так, більше число запишеться в змінну min.
    */
    int min=(b<a)?b:a;

    // Вивід результату на екран.
    cout<<"\n Maximum is \n"<<max;
    cout<<"\n Minimum is \n"<<min<<"\n";
    return 0;
}
```

Отже, давайте твердо усвідомимо наступне: якщо умова і дії, які від неї залежать, досить прості, будемо використовувати тернарний оператор. Якщо ж нам необхідна складна конструкція, то, безумовно, використовуємо оператор **if**.

7. Дрaбинка if — else if

З попереднього розділу уроку ви дізналися про існування умовних операторів. Тепер непогано було б отримати інформацію про особливості їхньої роботи.

Припустимо, ми маємо написати програму для обліку грошової знижки, залежно від суми. Наприклад, якщо покупець придбав товару на суму більшу ніж 100 грн., він отримує знижку 5%. Більше ніж 500 грн. — 10%, і, нарешті, більше ніж 1000 грн. — 25%. Додаток має видати суму, яку має сплатити покупець, якщо останній отримав знижку. Тепер необхідно знайти оптимальний варіант рішення.

Варіант рішення №1

```
#include <iostream>
using namespace std;
int main(){
    // оголошується змінна для зберігання
    // первісної суми
    int sum;
    // запит на ввід суми з клавіатури
    cout<<"Enter the total amount:\n";
    cin>>sum;
    if(sum>100){// якщо сума більша за 100 грн.,
                // знижка 5%
        cout<<"You have a 5% discount!!!\n";
        cout<<"You have to pay "<<sum-sum/100*5<<"\n";
    }
    if(sum>500){// якщо сума більша за 500 грн., знижка 10%
        cout<<"You have a 10% discount!!!\n";
    }
}
```

```

    cout<<"You have to pay "<<sum-sum/100*10<<"\n";
}
if(sum>1000){// якщо сума більша за 1000 грн.,
             // знижка 25%
    cout<<"You have a 25% discount!!!\n";
    cout<<"You have to pay "<<sum-sum/100*25<<"\n";
}
else{ // в іншому випадку, знижки немає
    cout<<"You have no discount!!!\n";
    cout<<"You have to pay "<<sum<<"\n";
}
return 0;
}

```

Цей приклад, на перший погляд програміста-початківця не викликає нарікань, проте, давайте розглянемо ситуацію, у якій програма відпрацює вельми некоректно. Сума, введена з клавіатури, дорівнює 5000. Ця цифра перевищує 1000, отже, ми маємо отримати знижку 25%. Однак відбудеться зовсім інше.

Кожен оператор `if` є самостійним і не залежить від інших `if`, отже, незалежно від того, яке з `if` виконається, перевірка умови все одно буде здійснюватися для всіх операторів.

Спочатку здійсниться перевірка умови `if(sum > 100)`. 5000, звичайно, більше за 100, умова істинна і виконується тіло `if`. На екрані ми отримуємо:

```

You have a 5% discount!!!
You have to pay 4750

```

Однак на цьому програма не зупиниться — далі буде проаналізовано умову `if(sum > 500)`. 5000 більше за 500,

умова знову істинна і виконується тіло `if`. На екрані ми отримуємо:

```
You have 10% discount!!!
You have to pay 4500
```

Ну і, нарешті, програма перевірить умову `if(sum > 1000)`, яка теж виявиться істинною, тому що `5000` більше за `1000`. І дія, пов'язана з `if`, виконується і тепер. На екран виводиться:

```
You have a 25% discount!!!
You have to pay 3750
```

Таким чином, замість одного інформаційного напису ми отримуємо три. Таке розв'язання задачі є нерентабельним. Спробуємо оптимізувати його.

Варіант рішення № 2

```
#include <iostream>
using namespace std;

int main() {
    // оголошується змінна для зберігання
    // первісної суми
    int sum;

    // запит на ввід суми з клавіатури
    cout<<"Enter the total amount:\n";
    cin>>sum;

    // якщо сума в діапазоні від 100 грн. до 500 грн.,
    // знижка 5%
```



```

if (sum>100&&sum<=500) {
    cout<<"You have a 5% discount!!!\n";
    cout<<"You have to pay "<<
        sum-sum/100*5<<"\n";
}

// якщо сума в діапазоні від 500 грн.
// до 1000 грн., знижка 5%
if (sum>500&&sum<=1000) {
    cout<<"You have 10% discount!!!\n";
    cout<<"You have to pay "<<
        sum-sum/100*10<<"\n";
}

if (sum>1000) { // якщо сума більша за 1000 грн.,
                // знижка 25%
    cout<<"You have a 25% discount!!!\n";
    cout<<"You have to pay "<<
        sum-sum/100*25<<"\n";
}
else { // в іншому випадку, знижки немає
    cout<<"You have no discount!!!\n";
    cout<<"You have to pay "<<sum<<"\n";
}
return 0;
}

```

Для початку знову уявімо, що користувач ввів суму розміром 5000 грн.

Спочатку здійсниться перевірка умови `if (sum>100&&sum<=500)`. 5000 не входить у заданий діапазон, умова помилкова і тіло `if` виконуватися не буде.

Далі буде проаналізовано умову `if (sum>500&&sum<=1000)`. 5000 не входить і в цей діапазон, умова знову помилкова і тіло `if` виконуватися не буде.

І, нарешті, програма перевірить умову `if(sum > 1000)`, яка виявиться істинною, тому що **5000** більше за **1000**. І дія, пов'язана з `if`, виконається. На екран виводиться:

```
You have a 25% discount!!!
You have to pay 3750
```

Здавалося б, на цьому можна зупинитися, але давайте перевіримо ще один варіант. Наприклад, користувач вводить значення **600**. І на екрані з'являються такі дані:

```
You have a 10% discount!!!
You have to pay 540
You have no discount!!!
You have to pay 600
Press any key to continue
```

Такий поворот подій пояснюється легко: спочатку здійсниться перевірка умови `if (sum > 100 && sum <= 500)`. **5000** не входить у заданий діапазон, умова помилкова і тіло `if` виконуватися не буде.

Далі буде проаналізовано умову `if(sum > 500 && sum <= 1000)`. **5000** входить у цей діапазон, умова істинна і тіло `if` виконається, на екран виведеться повідомлення про знижку **10%**.

І, нарешті, програма перевірить умову `if(sum > 1000)`, яка виявиться помилковою. Дія, пов'язана з `if`, виконуватися не буде, але цей самостійний оператор `if` має власний `else`, який відпрацює в нашому випадку. На екран виводиться повідомлення про відсутність знижки.

Вивід: по-перше, ми з'ясували, що оператор `else` належить тільки до останнього `if`. По-друге, прийшли

до того, що й ця реалізація програми нас не влаштовує.

Розглянемо ще один приклад рішення. Назва проекту Discount3.

Варіант рішення №3

```
#include<iostream>
using namespace std;
int main(){
    // оголошується змінна для зберігання
    // первісної суми
    int sum;
    // запит на ввід суми з клавіатури
    cout<<"Enter the total amount:\n";
    cin>>sum;
    if(sum>1000){ // якщо сума більша за 1000 грн.,
                  // знижка 25%
        cout<<"You have a 25% discount!!!\n";
        cout<<"You have to pay "<<
        sum-sum/100*25<<"\n";
    }

    else{ // якщо сума не більша за 100 грн.,
          // продовжуємо аналіз
        if(sum>500){ // якщо сума не більша за 500 грн.,
                     // знижка 10%
            cout<<"You have a 10% discount!!!\n";
            cout<<"You have to pay "<<
            sum-sum/100*10<<"\n";
        }
        else{ // якщо сума не більша за 500 грн.,
              // продовжуємо аналіз
            if(sum>100){ // якщо сума більша за 100 грн.,
                        // знижка 5%
```

```

        cout<<"You have a 5% discount!!!\n";
        cout<<"You have to pay "<<
        sum-sum/100*5<<"\n";
    }
    else{ // якщо сума не більша за 100 грн.,
        // знижки немає
        cout<<"You have not discount!!!\n";
        cout<<"You must pay - "<<sum<<"\n";
    }
}
}
return 0;
}

```

Уважно проаналізувавши цей приклад, ви помітите, що кожен наступний **if** може виконатися тільки в тому випадку, якщо не виконався його «попередник», тому що він розташований усередині конструкції **else** останнього. Отже, ми нарешті знайшли оптимальний код реалізації. Структура, яку ми щойно створили, називається «драбинка» **if else if**, тому що умови в ній розташовуються у вигляді драбинки. Тепер, ми з вами знаємо, яка це корисна конструкція. Залишився останній штрих.

Оптимізація коду

У попередньому розділі уроку прозвучало правило: якщо до блока **if** або **else** належить тільки одна команда, то фігурні дужки можна не вказувати. Справа в тому, що конструкція **if else** вважається однією цільною командною структурою. Відповідно, якщо всередині деяких **else** немає нічого крім вкладеної конструкції, фігурні дужки таких **else** можна опустити:

```

#include <iostream>
using namespace std;

int main(){
    // оголошується змінна для зберігання
    // первісної суми
    int sum;

    // запит на ввід суми з клавіатури
    cout<<"Enter the total amount:\n";
    cin>>sum;
    if(sum>1000){ // якщо сума більша за 1000 грн.,
                  // знижка 25%
        cout<<"You have a 25% discount!!!\n";
        cout<<"You have to pay "<<
            sum-sum/100*25<<"\n";
    }

    // якщо сума не більша за 1000 грн.
    // продовжуємо аналіз
    else if(sum>500){ // якщо сума більша за 500 грн.,
                     // знижка 10%
        cout<<"You have a 10% discount!!!\n";
        cout<<"You have to pay "<<
            sum-sum/100*10<<"\n";
    }

    // якщо сума не більша за 500 грн.
    // продовжуємо аналіз
    else if(sum>100){ // якщо сума більша за
                     // 100 грн., знижка 5%
        cout<<"You have a 5% discount!!!\n";
        cout<<"You have to pay "<<
            sum-sum/100*5<<"\n";
    }
    return 0;
}

```

От і все! Задачу розв'язано. Ми отримали цілісну конструкцію множинного вибору, що складається з окремих, взаємозалежних умов. Тепер можна переходити до наступних розділів уроку, де ми з вами докладно розглянемо ще кілька прикладів використання `if else`.

8. Практичний приклад створення текстового квесту

Постановка задачі

Ви, звичайно, знайомі з таким жанром ігор як квест. Герой такої гри має виконувати різні завдання, відповідати на питання, приймати рішення, від яких залежить результат гри. Ми з вами спробуємо зараз створити так званий текстовий квест (квест без графіки). Наше завдання — пропонувати герою варіанти дій, і залежно від його вибору, будувати ситуацію.

Код реалізації

```
#include <iostream>
using namespace std;

int main()
{
    // Ласкаво просимо. Три випробування честі.
    // Лихий маг викрав принцесу і її доля у твоїх
    // руках. Він пропонує тобі пройти 3 випробування
    // честі в його лабіринті.
    cout<<"Welcome. Three Tests of Honor."
         "A beautiful princess was spirited off by"
         "a wicked magician,\n\n";
    cout<<"\nand her fate is in your hands now."
         "He proposed you\n";
    cout<<"\nto pass Three Tests of Honor in his "
         "maze.\n";
```

```

bool goldTaken, diamondsTaken, killByDragon;
// Ти входиш у першу кімнату, тут дуже багато
// золота.
cout<<"You enter the first room with lots of gold "
      "scattered all over the place.\n\n";
// Ти візьмеш його?
cout<<"Will you take it?(1=yes, 0=no)\n\n";
cin>>goldTaken;
if(goldTaken) // якщо візьмеш,
{
    // золото залишається тобі, але ти провалив
    // випробування. ГРУ ЗАКІНЧЕНО!!!
    cout<<"The gold is yours, but you have "
          "failed the test. GAME OVER\n\n";
}
else // якщо ні,
{
    // Вітаю! Ти пройшов перше випробування честі!
    cout<<"Congratulations! You have passed "
          "the First Test of Honor!\n\n";
    // Ти переходиш до наступної кімнати.
    // Вона сповнена діамантів
    cout<<"You enter the next room. "
          "It is full of diamonds \n\n";
    // Ти візьмеш діаманти?
    cout<<"Will you take the diamonds? "
          "(1=yes,0=no)\n\n";
    cin>>diamondsTaken;
    if(diamondsTaken)// якщо візьмеш
    {
        // Діаманти залишаються тобі,
        // але ти провалив друге випробування
        cout<<"The diamonds are yours, but you "
              "have failed the Second Test \n\n";
        // ГРУ ЗАКІНЧЕНО!!!
        cout<<"GAME OVER\n\n";
    }
}

```



```

else // якщо ні
{
    // Вітаю, ти пройшов друге випробування
    // честі!!!
    cout<<"Congratulations! You have passed "
         "the Second Test of Honor!\n\n";
    // Ти входиш у третю кімнату.
    cout<<"You enter the third room. \n\n";
    // На селянина напав дракон!
    // Рухатися далі,
    cout<<"A peasant is attacked by "
         "a dragon! Move along \n\n";
    // не звертаючи на них уваги
    cout<<"and ignore them "
         "(1=yes,0=no)?\n\n";
    cin>>killByDragon;
    if(killByDragon)// якщо так
    {
        // Ти намагаєшся прослизнути повз,
        // але дракон
        cout<<"You try to sneak past them, "
             "but the dragon \n\n";
        // помічає тебе.
        cout<<"notices you\n\n";
        // Він перетворює тебе на попіл.
        // Ти мертвий!!!
        cout<<"It burns you to ashes. "
             "You died!\n\n";
        // ГРУ ЗАКІНЧЕНО!!!
        cout<<"GAME OVER\n\n";
    }
    else// якщо ні
    {
        // Вітаю, ти з честю пройшов усі
        // випробування!!!
        cout<<"Congratulations! You have passed"
             "all Tests of Honor! "
             "\n\n";
    }
}

```

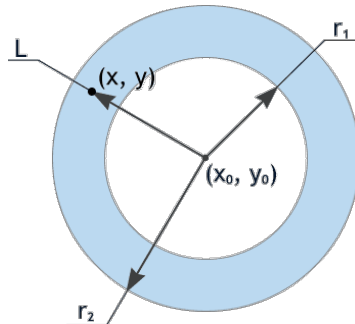
```
        // Принцеса дістається тобі!!!  
        cout<<"You get the princess!\n\n";  
    }  
}  
}  
return 0;  
}
```

Незважаючи на примітивність прикладу, ви можете переконатися в тому, що вже зараз, маючи мінімальні знання, ми можемо написати програму, здатну розважити середньостатистичного малюка. Це відбувається тому, що в наших руках є потужний засіб — умовні оператори.

9. Практичний приклад на приналежність точки кільцю

Постановка задачі

На площині намальовано кільце з центром у точці (X_0, y_0) і радіусами $r_1 < r_2$. На цій самій площині також дана точка з координатами (x, y) . Необхідно визначити, чи належить ця точка кільцю. Назва проекту [Circle2](#).



Малюнок 2

Розв'язання задачі

Для розв'язання задачі необхідно обчислити відстань від центру кільця до точки й порівняти її з радіусами:

1. Якщо довжина відрізка від центру до точки менша за радіус зовнішнього кола і при цьому більша за радіус внутрішньої окружності, то точка належить кільцю.

`if($L \geq r_1$ & & $L < r_2$)`

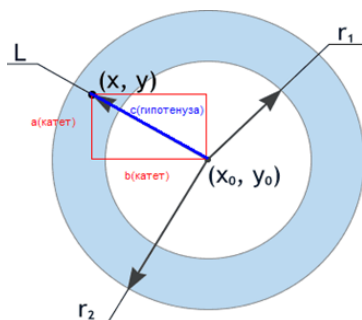
В іншому випадку точка не належить кільцю.

2. Для з'ясування відстані від центру до точки, ми скористаємося теоремою Піфагора: «Квадрат гіпотенузи дорівнює сумі квадратів катетів». Отже, довжина гіпотенузи дорівнює кореню квадратному із суми квадратів катетів.

Примітка: Нам знадобляться додаткові знання для отримання степеня й кореня квадратного.

- У програму необхідно підключити бібліотеку для використання математичних функцій під назвою `math.h`.
- Для зведення в степінь використовується функція `pow(double num, double exp)`, де `num` — це число для зведення в степінь, а `exp` — сам степінь.
- Аби витягти корінь квадратний, використовуємо функцію `sqrt(double num)`, де `num` — це число, з якого витягують корінь. `c=sqrt(pow(a,2)+ pow(b,2));`

$$L=c;$$



Малюнок 3

3. Залишилося з'ясувати довжини катетів, на малюнку видно, як це зробити.

$$\begin{aligned} a &= y - y_0; \\ b &= x - x_0. \end{aligned}$$

Тепер залишилося зібрати всі частини рішення в єдине ціле.

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

using namespace std;
int main() {
    // Оголошення змінних
    int x0, y0, r1, r2, x, y;
    float L;
    // Запит на ввід необхідних даних
    cout<<"Input the coordinates of the circle's "
         "center (X0, Y0):";
    cin>>x0>>y0;
    cout<<"Input the circle's radiuses R1 and R2:";
    cin>>r1>>r2;
    cout<<"Input the coordinates (X, Y) of the point:";
    cin>>x>>y;
    // Вивід формули
    L = sqrt(pow(x - x0, 2) + pow(y - y0, 2));
    // Аналіз результатів
    if ((r1 < L)&& (L < r2 )) {
        cout<<"This point is inside "
             "the ring.\n";
    }
    else {
        cout<<"This point is outside "
             "the ring.\n";
    }
    return 0;
}
```

10. Структура множинного вибору switch

Ми вже знайомі з конструкцією, яка аналізувала умови — конструкцією `if`, а також із тернарним оператором. Ще один оператор вибору — оператор `switch`. Уявіть, що необхідно написати програму, у якій використовується меню, що складається з п'яти пунктів. Наприклад, маленький додаток для малюків, який вміє складати, віднімати тощо. Можна реалізувати обробку вибору за допомогою драбинки `if else if`, ось так:

```
#include <iostream>
using namespace std;
int main() {
    // оголошення змінних і ввід значення з клавіатури
    float A, B, RES;
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;

    // реалізація програмного меню
    char key;
    cout<<"\nChoose the operator:\n";
    cout<<"\n + - if you want to ADD.\n";
    cout<<"\n - - if you want to DEDUCT.\n";
    cout<<"\n * - if you want to MULTIPLY.\n0";
    cout<<"\n / - if you want to DIVIDE.\n";

    // очікування вибору користувача
    cin>>key;
```

```

if (key=='+') { // якщо користувач вибрав додавання
    RES=A+B;
    cout<<"\nAnswer: "<<RES<<"\n";
}
else if (key=='-'){ // якщо користувач вибрав
                    // віднімання
    RES=A-B;
    cout<<"\nAnswer: "<<RES<<"\n";
}
else if (key=='*'){ // якщо користувач вибрав
                    // множення
    RES=A*B;
    cout<<"\nAnswer: "<<RES<<"\n";
}
else if (key=='/'){ // якщо користувач
                    // вибрав ділення
    if (B){ // якщо дільник не дорівнює нулю
        RES=A/B;
        cout<<"\nAnswer: "<<RES<<"\n";
    }
    else{ // якщо дільник дорівнює нулю
        cout<<"\nError! You cannot divide by "
              "zero!\n";
    }
}
else{ // якщо введений символ є некоректним
    cout<<"\nError! "
          "Invalid value \n";
}
}
return 0;
}

```

Вищеописаний приклад цілком коректний, але виглядає дещо громіздко. Цей код можна значно спростити, саме для цього використовується **switch**. Він дає змогу

порівняти значення змінної з цілою низкою значень і, зустрівши збіг, виконати певну дію.

Загальний синтаксис і принцип дії

Для початку розглянемо загальний синтаксис оператора:

```
switch(вираз) {  
  case значення1:  
    дія1;  
    break;  
  
  case значення2:  
    дія2;  
    break;  
  
  case значення3:  
    дія3;  
    break;  
  
  .....  
  
  default:  
    дія_за_замовчуванням  
    break;  
}
```

Давайте проаналізуємо цю форму запису:

1. **Вираз** — ті дані, які необхідно перевірити на відповідність. Тут може вказуватися змінна (але тільки типу **char** або цілочисельна) або вираз, результатом якого є цілочисельні дані.
2. **Case Значення1, case значення2, case значення3** — Цілочисельні або символічні постійні значення, з якими звіряється вираз.

3. **Дія1, дія2, дія3** — дії, які мають виконатися, якщо значення виразу збігається зі значенням `case`.
4. Якщо стався збіг і безберешково виконилася дія, пов'язана з `case`, що збігся, `switch` закінчує свою роботу і програма переходить на наступний рядок після закриття фігурної дужки оператора `switch`. За цю функцію відповідає оператор `break`, саме він зупиняє виконання `switch`.
5. Якщо під час аналізу збігів не відбулося, спрацьовує секція `default` і виконується `дія_за_замовчуванням`. Оператор `default` є аналогом оператора `else`.

Тепер давайте подивимося, як можна спростити наведений на початку теми приклад.

Оптимізація прикладу

```
#include <iostream>

using namespace std;
int main(){
    // оголошення змінних і ввід значення
    // з клавіатури
    float A,B,RES;
    cout<<"Enter your first number:\n";
    cin>>A;
    cout<<"Enter your second number:\n";
    cin>>B;

    // реалізація програмного меню
    char key;
    cout<<"\nSelect operator:\n";
    cout<<"\n + - if you want to ADD.\n";
    cout<<"\n - - if you want to DEDUCT.\n";
```

```

cout<<"\n * - if you want to MULTIPLY.\n";
cout<<"\n / - if you want to DIVIDE.\n";

// очікування вибору користувача
cin>>key;

// перевіряється значення змінної key
switch(key){
case '+': // якщо користувач вибрав додавання
    RES=A+B;
    cout<<"\nAnswer: "<<RES<<"\n";
    break; // зупинка switch

case '-': // якщо користувач вибрав віднімання
    RES=A-B;
    cout<<"\nAnswer: "<<RES<<"\n";
    break; // зупинка switch

case '*': // якщо користувач вибрав множення
    RES=A*B;
    cout<<"\nAnswer: "<<RES<<"\n";
    break; // зупинка switch case '/':
           // якщо користувач вибрав ділення

case '/': // якщо користувач вибрав ділення
    if(B){ // якщо дільник не дорівнює нулю
        RES=A/B;
        cout<<"\nAnswer: "<<RES<<"\n";
    }
    else{ // якщо дільник дорівнює нулю
        cout<<"\nError! You cannot divide by zero!\n";
    }
    break; // зупинка switch

default: // якщо введений символ є некоректним
    cout<<"\nError! Invalid "

```

```

        "value\n";
    break; // зупинка switch
}
return 0;
}

```

Як бачите, код тепер виглядає набагато простіше, і його зручніше читати.

Оператор **switch** досить простий у використанні, проте необхідно знати деякі особливості його роботи:

1. Якщо в **case** використовуються символічні значення, вони мають бути вказані в одинарних лапках, якщо цілочисельні, то без лапок.
2. Оператор **default** може розташовуватися в будь-якому місці системи **switch**, виконуватися він усе одно буде в тому випадку, якщо немає жодного збігу. Однак правилом «хорошого тону» є вказувати **default** у кінці всієї конструкції.

```

switch(вираз) {
case значення1:
    дія1;
    break;
case значення2:
    дія2;
    break;
default: дія_за_замовчуванням
    break;
case значення3:
    дія3;
    break;
}

```

3. Після останнього оператора в списку (**case** або **default**) оператор **break** можна не вказувати.

```
switch(вираз) {  
    case значення1:  
        дія1;  
        break;  
  
    case значення2:  
        дія2;  
        break;  
  
    default:  
        дія_за_замовчуванням;  
        break;  
  
    case значення3:  
        дія3;  
}
```

```
switch(вираз) {  
    case значення1:  
        дія1;  
        break;  
  
    case значення2:  
        дія2;  
        break;  
  
    case значення3:  
        дія3;  
        break;  
  
    default:  
        дія_за_замовчуванням;  
}
```

4. Оператор **default** можна взагалі не вказувати — якщо не знайдеться збігів, просто нічого не відбудеться.

```
switch(вираз) {  
    case значення1:  
        дія1;  
        break;  
  
    case значення2:  
        дія2;  
        break;  
  
    case значення3:  
        дія3;  
        break;  
}
```

5. У разі, якщо необхідно виконувати той самий набір дій для різних значень виразу, що перевіряється, можна

записати кілька міток поспіль. Розглянемо приклад програми, яка переводить систему літерних оцінок у цифрові.

```
#include <iostream>
using namespace std;

int main(){
    // оголошення змінної, для зберігання літерної
    // оцінки
    char cGrade;
    // прохання ввести літерну оцінку
    cout<<"Input your letter grade\n";
    cin>>cGrade;
    // аналіз введеного значення
    switch (cGrade) {
        case 'A':
        case 'a':
            // оцінка A чи a рівноцінна 5
            cout<<"Your grade is 5\n";
            break;
        case 'B':
        case 'b':
            // оцінка B чи b рівноцінна 4
            // а також оцінка C або c рівноцінна 4-
        case 'C':
        case 'c':
            cout<<"Your rate is 4\n";
            break;
            // оцінка C чи c рівноцінна 3
            cout<<"Your grade is 3\n";
            break;
        case 'D':
        case 'd':
            // оцінка D чи d рівноцінна 2
            cout<<"Your grade is 2\n";
            break;
```

```

    case 'F':
    case 'f':
        // оцінка F чи f рівноцінна 1
        cout<<"Your grade is 1\n";
        break;
    default:
        // інші символи є некоректними
        cout<<"This grade is invalid\n";
}
return 0;
}

```

Приклад примітний тим, що за допомогою **case**, що йдуть поспіль, досягається регістронезалежність. Тобто неважливо, яку саме літеру введе користувач — велику чи малу.

Поширена помилка

Усе найголовніше про оператор **switch** сказано, залишилося лише отримати інформацію про те, з якою проблемою може зіткнутися програміст, використовуючи цей оператор.

Якщо випадково пропустити **break** у будь-якому блоці **case**, окрім останнього, і цей блок надалі відпрацює, то виконання **switch** не зупиниться. Той блок оператора **case**, який буде йти слідом за вже виконаними, так само виконається без перевірки.

Приклад помилки

```

#include <iostream>
using namespace std;

```

```

int main(){
    // реалізація програмного меню
    int action;
    cout<<"\nSelect action:\n";
    cout<<"\n Press 1 if you want to see the dollar "
         "rate.\n";
    cout<<"\n Press 2 if you want to see the euro "
         "rate.\n";
    cout<<"\n Press 3 if you want to see the ruble "
         "rate.\n";

    // очікування вибору користувача
    cin>>action;
    // перевіряється значення змінної action
    switch(action){
        case 1: // якщо користувач вибрав долар
            cout<<"\nThe rate is 28 UAH.\n";
            break; // зупинка switch
        case 2: // якщо користувач вибрав євро
            cout<<"\nThe rate is 30 UAH.\n";
            // break; закоментована зупинка switch
        case 3: // якщо користувач вибрав рублі
            cout<<"\nThe rate is 0.44 UAH.\n";
            break; // зупинка switch
        default: // якщо вибір не є коректним
            cout<<"\nError! Invalid "
                 " value\n";
            break; // зупинка switch
    }
    return 0;
}

```

Помилка відбудеться в тому випадку, якщо буде обраний 2 пункт меню. У `case` зі значенням 2 закоментовано оператор зупинки `break`. На екрані результат такої помилки виглядає так:

```
Select action:
Press 1 if you want to see the dollar rate.
Press 2 if you want to see the euro rate.
Press 3 if you want to see the ruble rate.
2
The rate is 30 UAH.
The rate is 0.44 UAH.
Press any key to continue
```

Окрім необхідної інформації, на екрані вивелось те, що знаходилося в блоці `case`, який розташований після помилкової конструкції. Треба уникати таких помилок, тому що вони призводять до помилок на етапі виконання.

11. Поняття enum як перерахування

Зараз настав час вивчити ще один тип даних — **enum**. **Перерахування (enum)** — це набір іменованих цілочисельних констант.

Припустимо, у вашій програмі необхідно створити кілька констант з кодом відповідної країни. Це виглядало б ось так:

```
const int USA = 1;  
const int France= 33;  
const int Ukraine = 380;  
const int Italy = 39;  
const int Australia = 61;
```

Однак такий спосіб тільки нагромаджує програмний код, він займає надто велику кількість рядків. І, як бачимо, це ще не всі країни. Природно, хочеться якось систематизувати, згрупувати цю інформацію. Саме перерахування дозволяє вирішити таку проблему. Синтаксис:

```
enum ім'я { константа1 [= значення1],  
           константа2 [= значення2], ...};
```

Тоді список констант матиме ось такий вигляд:

```
enum countries { USA =1, France=33, Ukraine=380,  
                Italy=39, Australia =61};
```

Після створення такого перерахування країн імена USA, France, Ukraine, Italy, Australia стають усе тими ж цілочисельними константами, що і в прикладі вище.

Також як приклад перерахування можна використовувати список монет США. Унікальність цих монет у тому, що майже кожна з них, окрім свого номіналу, має вже усталену власну назву. Наприклад, один цент називають пенні, а 5 центів — нікель. За значення константи візьмемо номінал монети, а як її ім'я — її загальноприйняту назву.

```
enum coins { penny = 1, nickel = 5, dime = 10,  
            quarter = 25, half = 50,  
            dollar_coin = 100 };
```

Використання перерахувань дає змогу спростити розуміння програм, написаних іншими програмістом, хоч і є більш трудоемним. Подивимося на перерахування американських монет у програмі нижче:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    // Оголошення перерахування монет США  
    enum coins { penny = 1, nickel = 5, dime = 10,  
                quarter = 25, half = 50,  
                large_dollar = 100 };  
    // Оголошення змінної для монети  
    int coin;  
    cout << "Please enter the value of an American coin"  
          << endl;  
    cin >> coin;  
    switch (coin)  
    {
```

```

case penny:
    // Виводимо на екран, що пенні
    // відповідає одному центу
    cout << "penny = 1 cent " << endl;
    // Доповнюємо вивід описом монети
    // На одній стороні є Авраам Лінкольн,
    // а Меморіал Лінкольна – на іншій.
    cout << "The obverse depicts Abraham Lincoln,"
           "and the reverse pictures "
           "the Lincoln Memorial." << endl;
    break;
case nickel:
    // Виводимо на екран, що нікель
    // відповідає 5 центам
    cout << "nickel = 5 cents" << endl;
    // Доповнюємо вивід описом монети
    // На лицьовій стороні монети зображений
    // Томас Джефферсон, і Монтічелло на
    // зворотному боці.
    cout << "The obverse depicts "
           "Thomas Jefferson, and "
           "the reverse pictures Monticello."
           << endl;
    break;
case dime:
    // Виводимо на екран, що дайм
    // відповідає 10 центам
    cout << "dime = 10 cents" << endl;
    // Доповнюємо вивід описом монети
    // На лицьовій стороні монети зображений
    // Франклін Д. Рузвельт, і факел
    // на зворотному боці
    cout << "The obverse depicts Franklin D."
           "Roosevelt and the reverse"
           "pictures a torch oak and"
           "olive branches." << endl;
    break;

```

```

case quarter:
    // Виводимо на екран, що 1/4 долара
    // складає 25 центів
    cout << "quater = 25 cents" << endl;
    // Доповнюємо вивід описом монети
    // На лицьовій стороні монети зображений
    // Джордж Вашингтон, і на зворотному боці -
    // або емблема Сполучених Штатів,
    // або дизайн одного з 50 штатів
    cout << "The obverse depicts George "
           "Washington, and the reverse "
           "pictures either America "
           "the Beautiful Quarters "
           "or State Quarter Series." << endl;
    break;
case half:
    // Виводимо на екран, що 1/2 долара
    // складає 50 центів
    cout << "half = 50 cents " << endl;
    // Доповнюємо вивід описом монети
    // На лицьовій стороні монети зображений
    // Джон Ф. Кеннеді, а на зворотному боці -
    // велика печатка США.
    cout<<"The obverse depicts John F. Kennedy"
           "and the reverse pictures the Seal of"
           "the President of the United "
           "States." << endl;
    break;
case large_dollar:
    // Виводимо на екран, що 1 доларова монета
    // становить 100 центів
    cout << "large dollar = 100 cents" << endl;
    // Доповнюємо вивід описом монети
    // На ній зображена рідна американська
    // героїня Сакаджавея на лицьовій стороні і
    // білоголовий орлан – на зворотній.
    cout << "The obverse depicts Sacagawea, "

```

```

        "who is now an American symbol, "
        "and the reverse pictures a bald"
        "eagle." << endl;

        break;
    default:
        cout << "not found" << endl;
    }
    return 0;
}

```

Нижче надано приклад роботи цієї програми:

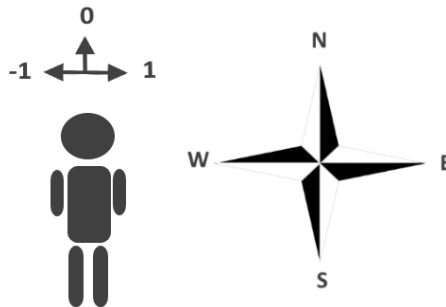
```

C:\Windows\system32\cmd.exe
Please enter a value of american coin
10
dime = 10 cents
It has Franklin D. Roosevelt on the front and a torch on the back.

```

Малюнок 4

А тепер розглянемо ще один приклад — наше завдання змодельовати поведінку робота при його пересуванні. Бувши сучасним пристроєм, наш робот знає про чотири сторони світу: 1 — північ, 2 — захід, 3 — південь, 4 — схід. А також він може виконувати прості команди: 0 — продовжити рух, 1 — поворот направо, -1 — поворот наліво.



Малюнок 5

Так от, наша мета — визначити напрямок робота після виконаної команди. Що для цього потрібно знати? Перше — це куди спочатку дивиться робот, а друге — команда, що виконується в поточний момент.

```
enum command {forward, right, left = -1};
enum direction {north = 1, west, south, east};
```

Можна помітити, що ініціалізація констант необов'язкова. Тоді які значення матимуть ці константи? Якщо жодна з констант не була проініціалізована, тоді перша з них буде за замовчуванням містити значення нуль, а всі інші на одиницю більше, тобто `forward = 0`, `right = 1`, але `left = -1`. Точно з такою ж закономірністю наступні константи мають значення: `north = 1`, `west = 2`, `south = 3`, `east = 4`.

Часто перерахування найкраще використовувати спільно зі `switch`-конструкцією. Саме таким способом більше зрозуміло, які дії потрібно виконувати за поточним значенням `case`.

А тепер розглянемо саму програму:

```
#include <iostream>

using namespace std;
int main() {
    // Оголошення перерахувань для команд і напрямку
    enum command { forward, right, left = -1 };
    enum direction { north = 1, west, south, east };
    // Оголошення змінних для прийнятої команди
    // і початкового напрямку робота
    int Command, Direction;
    cout << "Enter the starting direction of "
         << "the robot" << endl;
```

```

cout << "\t 1- North\n"
      << "\t 2- West\n"
      << "\t 3- South\n"
      << "\t 4- East\n";
cin >> Direction;
cout << "Select action : " << endl;
cout << "\t 0- forward\n"
      << "\t 1- turn right\n"
      << "\t -1- turn left\n";
cin >> Command;

switch (Direction)
// визначаємо початковий напрямок робота
{
    case north: // якщо робот спочатку дивився
                // на північ,
        switch (Command)
        // визначаємо подальшу поведінку робота
        {
            case forward:
                cout << "The robot is looking "
                      "north\n";
                break;
            case right:
                cout << "The robot is looking "
                      "east\n";
                break;
            case left:
                cout << "The robot is looking "
                      "west\n";
                break;
        }
        break;
    case west: // якщо робот спочатку дивився
               // на захід,
        switch (Command)

```

```

// визначаємо подальшу поведінку робота
{
    case forward:
        cout << "The robot is looking "
              "west\n";
        break;
    case right:
        cout << "The robot is looking "
              "north\n";
        break;
    case left:
        cout << "The robot is looking "
              "south\n";
        break;
}
break;
case south: // якщо робот спочатку дивився на південь,
switch (Command)
// визначаємо подальшу поведінку робота
{
    case forward:
        cout << "The robot is looking south\n";
        break;
    case right:
        cout << "The robot is looking west\n";
        break;
    case left:
        cout << "The robot is looking east\n";
        break;
}
break;
case east: // якщо робот спочатку дивився
           // на схід,
switch (Command)
// визначаємо подальшу поведінку робота
{
    case forward:

```

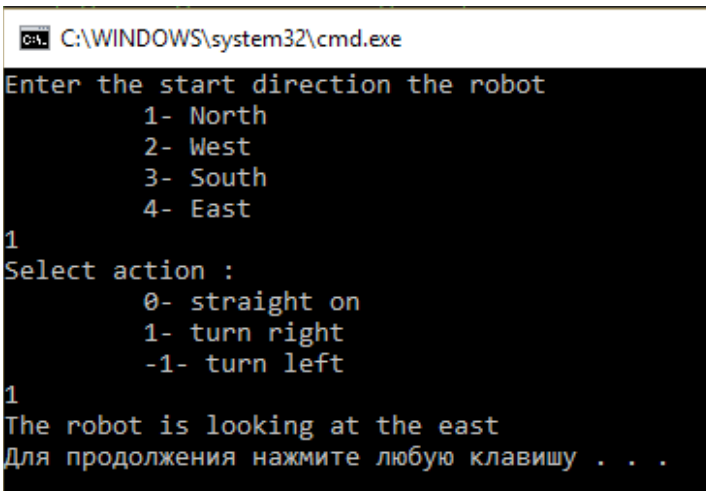


```

        cout << "The robot is looking east\n";
        break;
    case right:
        cout << "The robot is looking south\n";
        break;
    case left:
        cout << "The robot is looking north\n";
        break;
    }
    break;
default:
    cout<<"not found"<< endl;
}
return 0;
}

```

Нижче надано приклад роботи програми:



```

C:\WINDOWS\system32\cmd.exe
Enter the start direction the robot
    1- North
    2- West
    3- South
    4- East
1
Select action :
    0- straight on
    1- turn right
   -1- turn left
1
The robot is looking at the east
Для продолжения нажмите любую клавишу . . .

```

Малюнок 6

Можемо переконатися, що без використання пере-
рахування дуже легко заплутатися в поведінці самої

програми. І саме перерахування допомагають спростити розуміння такого коду.

У сьогоднішньому уроці ми з вами познайомилися з операторами, що дозволяють проводити аналіз будь-яких даних. Тепер ви можете переходити до виконання домашнього завдання. Бажаємо успіху!

11. Домашнє завдання

1. Напишіть програму, яка перевіряє число, введене з клавіатури на парність.
2. Дано натуральне число a ($a < 100$). Напишіть програму, яка виводить на екран кількість цифр у цьому числі і суму цих цифр
3. Відомо, що 1 дюйм дорівнює 2.54 см. Розробити додаток, який переводить дюйми в сантиметри і навпаки. Діалог з користувачем реалізувати через систему меню.
4. Написати програму-калькулятор. Користувач вводить два числа і вибирає арифметичну дію (+, -, *, /, максимум, мінімум). Вивести на екран результат дії.

STEP IT Academy, www.itstep.org

Усі права на захищені авторським правом фото, аудіо та відеотвори, фрагменти яких використані в матеріалі, належать їхнім законним власникам. Фрагменти творів використовуються з ілюстративною метою в обсязі, виправданому поставленим завданням, в межах навчального процесу і в навчальних цілях. Відповідно до ст. 21 і 23 Закону України «Про авторське право й суміжні права». Обсяг і спосіб цитованих творів відповідає прийнятим нормам, не завдає шкоди нормальному використанню об'єктів авторського права і не обмежує законні інтереси автора та правовласників. Цитовані фрагменти творів на момент використання не можуть бути замінені альтернативними, що незахищені авторським правом аналогами, і як такі відповідають критеріям сумлінного і чесного використання.

Усі права захищені. Повне або часткове використання матеріалів заборонено. Узгодження використання творів або їхніх фрагментів проводиться з авторами і правовласниками. Узгоджене використання матеріалів можливе лише за умов згадування джерела.

Відповідальність за несанкціоноване копіювання і комерційне використання матеріалів визначається чинним законодавством України.