Introduction to Programming

(in C++)

*Data types and visibility*

* Data types
* Type conversion
* Visibility
* A data type specifies:
	+ The set of values that data of that type can have (e.g. integer, real, character, Boolean, colour, Greek letter, city, etc.)
	+ The type of operations that can be performed with the data. For example, two integer numbers can be added, the population of a city can be calculated, etc.
* Integer (int). Represent the set of integer numbers.
	+ In practice, computers have a limitation representing integer numbers.
	+ For a 32-bit machine, **int** can represent the numbers in the interval [-(231-1), 231-1].

[-2147483648, 2147483647]

* + Arithmetic operators: +, -, , /, %

Integer division and remainder: 13 / 3 = 4, 13 % 3 = 1

* Real (double). Represent the set of real numbers.
	+ In practice, computers can only represent real numbers in a certain interval and with a certain accuracy.
	+ IEEE 754-1985 standard, double-precision 64 bit:
		- Numbers closest to zero: ±5 × 10−324
		- Numbers furthest from zero: ±1.7976931348623157 × 10308
		- Special representations for 0, + and -
		- See <http://en.wikipedia.org/wiki/IEEE_754-1985>
	+ Arithmetic operators: +, -, , / Real division: 13.0 / 4.0 = 3.25
* Boolean (bool). Represent logic values.
	+ Values: ***false*** and ***true***
	+ Operators: ***not***, ***and***, ***or***.



|  |  |
| --- | --- |
| *x* | ***not*** *x* |
| *false* | *true* |
| *true* | *false* |

|  |  |  |
| --- | --- | --- |
| *x* | *y* | *x* ***and*** *y* |
| *false* | *false* | *false* |
| *false* | *true* | *false* |
| *true* | *false* | *false* |
| *true* | *true* | *true* |

|  |  |  |
| --- | --- | --- |
| *x* | *y* | *x* ***or*** *y* |
| *false* | *false* | *false* |
| *false* | *true* | *true* |
| *true* | *false* | *true* |
| *true* | *true* | *true* |

* Properties of Boolean algebra
	+ Commutativity:
		- a and b = b and a
		- a or b = b or a
	+ Associativity:
		- (a and b) and c = a and (b and c)
		- (a or b) or c = a or (b or c)
	+ Distributivity:
		- a and (b or c) = (a and b) or (a and c)
		- a or (b and c) = (a or b) and (a or c)
	+ Double negation:
		- not (not a) = a
	+ De Morgan’s law:
		- not (a and b) = (not a) or (not b)
		- not (a or b) = (not a) and (not b)



* Character (char). Represent letters, digits, punctuation marks and control characters.
* Every character is represented by a code (integer number). There are various standard codes:
	+ American Standard Code for Information Interchange (ASCII)
	+ Unicode (wider than ASCII)
* Some characters are grouped by families (uppercase letters, lowercase letters and digits). Characters in a family have consecutive codes: 'a'…'z', 'A'…'Z', '0'…'9'
* Operators: given the integer encoding, arithmetic operators can be used, even though only addition and subtraction make sense, e.g. 'C'+1='D', 'm'+4='q', 'G'-1='F'.



ASCII code

* Strings (string). Represent sequences of characters.
* Examples
	+ "Hello, world!", "This is a string", ":-)", "3.1416"
	+ "" is the empty string (no characters)
	+ 'A' is a *character*, "A" is a *string*
* Note: use **#include <string>** in the header of a program using strings.
* The values of most data types can be compared using relational operators:

**== != > >= < <=**

* Relational operators return a Boolean value (*true* or *false*)
* Examples
	+ **5 == 5** is *true*, **5 == 6** is *false*, **5 != 6** is *true*

– **3.1416 <= 7** is *true*, **-5.99 >= 0.1** is *false*

* + **'J' <= 'K'** is *true*, **'a' == 'A'** is *false*
	+ **"Obama" == "Bush"** is *false*, **"Bush" == "Bush"** is *true*,

**"Bush" < "Obama"** is *true*, **"book" < "booking"** is *true*

(relational operators use lexicographical order in strings)

* A variable is declared as:

## type variable\_name;

* Examples

**int** population; **double** distance; **string** my\_name;

* Several variables can be declared together:

**int** age, children, cars;

* After its declaration, the value of a variable is undefined (unknown).
* Expression: a combination of literals, variables, operators and functions that is evaluated and returns a value
* Examples:

**a + 3(i - 1)**  int

**sqrt(x)log(4n)**  double

**(i - 3) <= x**  bool

**(a != b) and (s <= "abc")**  bool

* The operands used in expressions must be consistent with the operators.

## int a, b, n;

…

**(a <= b) + n** (Incorrect expression:

semantic error)

bool int

**cannot add bool to int**

* Operators in expressions are evaluated according to certain rules of precedence



|  |  |
| --- | --- |
| Unary | +, - , not |
| Multiplicative |  / % |
| Additive | + - |
| Relational (inequalities) | > >= < <= |
| Relational (equalities) | == != |
| Conjunction | and |
| Disjunction | or |

* Example: 3 + 45 != (3 + 4)5
* Use parenthesis to change the precedence or when you are not sure about it.

# TYPE CONVERSION

* Consider the following code:

**int i = 5; char a = ‘B’;**

**double x = 1.5; i = i + x;**

**if (i) x = 5a;**

* In many programming languages, the compiler would report several type errors. Possibly:

**int i = 5; char a = ‘B’;**

**double x = 1.5; i = i + x;**

**if (i) x = 5a;**

* In C++, there would be no errors in this fragment of code:

**int i = 5; char a = ‘B’;**

**double x = 1.5;**

**i = i + x; // i gets the value 6 if (i) x = 5a;**

**// the condition of the if statement**

**// would be true and x would get 5**

**// multiplied by the code of ‘B’**

**// converted into double**

* As a general rule, using implicit type conversions is not considered to be a good practice because:
	+ The code is less readable.
	+ The code is less reliable, since unintentional errors may be introduced and they may be difficult to debug.
* Recommendation: to operate with different types, use explicit type conversions

**char(i), int(‘a’), double(i)**

* Never use statements that depend on a particular encoding:

– Wrong: **c == 65, c == char(65), int(c) == 65**

* + Correct: **c == ‘A’**
* Arithmetic operations between integer and real values usually imply an implicit conversion into real values.
* Be careful:

**int i=3, j=2; double x;**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **x** | **=** | **i/j;** | **//** | **x** | **=** | **1.0** |
| **x** | **=** | **i/double(j);** | **//** | **x** | **=** | **1.5** |
| **x** | **=** | **double(i)/j;** | **//** | **x** | **=** | **1.5** |
| **x** | **=** | **double(i/j);** | **//** | **x** | **=** | **1.0** |
| **x** | **=** | **i/2;** | **//** | **x** | **=** | **1.0** |
| **x** | **=** | **i/2.0;** | **//** | **x** | **=** | **1.5** |

# VISIBILITY

* Variables are only visible after their declaration and in the block they have been declared.
* Blocks can include other blocks. The variables of the outer blocks are visible, a priori, in the inner blocks.
* A variable declared in an inner block masks the variables with the same name declared in outer blocks.

**{**

**// a and b are not visible int a = 1, b = 20;**

**// a and b are visible**

**cout << a; // writes 1**

**{**

**// c is not visible, a and b are visible cout << a + b; // writes 21 int b = 3, c = 4;**

**// b and c are visible,**

**// but the outer b is not visible cout << b; // writes 3**

**cout << c; // writes 4**

**}**

**// c is not visible**

**cout << b; // writes 20**

**}**

# C++ if-else

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

## C++ IF Statement

The C++ if statement tests the condition. It is executed if condition is true.

1. **if**(condition){
2. //code to be executed
3. }



## C++ If Example

1. #include <iostream>
2. **using** **namespace** std;
3.
4. **int** main () {
5. **int** num = 10;
6. **if** (num % 2 == 0)
7. {
8. cout<<"It is even number";
9. }
10. **return** 0;
11. }

Output:/p>

It is even number

## C++ IF-else Statement

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }



## C++ If-else Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num = 11;
5. **if** (num % 2 == 0)
6. {
7. cout<<"It is even number";
8. }
9. **else**
10. {
11. cout<<"It is odd number";
12. }
13. **return** 0;
14. }

**Output:**

It is odd number

## C++ If-else Example: with input from user

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a Number: ";
6. cin>>num;
7. **if** (num % 2 == 0)
8. {
9. cout<<"It is even number"<<endl;
10. }
11. **else**
12. {
13. cout<<"It is odd number"<<endl;
14. }
15. **return** 0;
16. }

**Output:**

Enter a number:11

It is odd number

**Output:**

Enter a number:12

It is even number

## C++ IF-else-if ladder Statement

The C++ if-else-if ladder statement executes one condition from multiple statements.

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else** **if**(condition2){
4. //code to be executed if condition2 is true
5. }
6. **else** **if**(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. **else**{
11. //code to be executed if all the conditions are false
12. }



## C++ If else-if Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a number to check grade:";
6. cin>>num;
7. **if** (num <0 || num >100)
8. {
9. cout<<"wrong number";
10. }
11. **else** **if**(num >= 0 && num < 50){
12. cout<<"Fail";
13. }
14. **else** **if** (num >= 50 && num < 60)
15. {
16. cout<<"D Grade";
17. }
18. **else** **if** (num >= 60 && num < 70)
19. {
20. cout<<"C Grade";
21. }
22. **else** **if** (num >= 70 && num < 80)
23. {
24. cout<<"B Grade";
25. }
26. **else** **if** (num >= 80 && num < 90)
27. {
28. cout<<"A Grade";
29. }
30. **else** **if** (num >= 90 && num <= 100)
31. {
32. cout<<"A+ Grade";
33. }
34. }

**Output:**

Enter a number to check grade:66

C Grade

**Output:**

Enter a number to check grade:-2

wrong number

# C++ switch

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;
5. **case** value2:
6. //code to be executed;
7. **break**;
8. ......
9.
10. **default**:
11. //code to be executed if all cases are not matched;
12. **break**;
13. }



## C++ Switch Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** num;
5. cout<<"Enter a number to check grade:";
6. cin>>num;
7. **switch** (num)
8. {
9. **case** 10: cout<<"It is 10"; **break**;
10. **case** 20: cout<<"It is 20"; **break**;
11. **case** 30: cout<<"It is 30"; **break**;
12. **default**: cout<<"Not 10, 20 or 30"; **break**;
13. }
14. }

Output:

Enter a number:

10

It is 10

Output:

Enter a number:

55

Not 10, 20 or 30

# C++ For Loop

The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

1. **for**(initialization; condition; incr/decr){
2. //code to be executed
3. }

**Flowchart:**



## C++ For Loop Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **for**(**int** i=1;i<=10;i++){
5. cout<<i <<"\n";
6. }
7. }

Output:

1

2

3

4

5

6

7

8

9

10

## C++ Nested For Loop

In C++, we can use for loop inside another for loop, it is known as nested for loop. The inner loop is executed fully when outer loop is executed one time. So if outer loop and inner loop are executed 4 times, inner loop will be executed 4 times for each outer loop i.e. total 16 times.

## C++ Nested For Loop Example

Let's see a simple example of nested for loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
3.
4. **int** main () {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. cout<<i<<" "<<j<<"\n";
8. }
9. }
10. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

## C++ Infinite For Loop

If we use double semicolon in for loop, it will be executed infinite times. Let's see a simple example of infinite for loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
3.
4. **int** main () {
5. **for** (; ;)
6. {
7. cout<<"Infinitive For Loop";
8. }
9. }

Output:

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

Infinitive For Loop

ctrl+c

# C++ While loop

In C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

1. **while**(condition){
2. //code to be executed
3. }

**Flowchart:**



## C++ While Loop Example

Let's see a simple example of while loop to print table of 1.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i=1;
5. **while**(i<=10)
6. {
7. cout<<i <<"\n";
8. i++;
9. }
10. }

Output:

1

2

3

4

5

6

7

8

9

10

## C++ Nested While Loop Example

In C++, we can use while loop inside another while loop, it is known as nested while loop. The nested while loop is executed fully when outer loop is executed once.

Let's see a simple example of nested while loop in C++ programming language.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **int** i=1;
5. **while**(i<=3)
6. {
7. **int** j = 1;
8. **while** (j <= 3)
9. {
10. cout<<i<<" "<<j<<"\n";
11. j++;
12. }
13. i++;
14. }
15. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

## C++ Infinitive While Loop Example:

We can also create infinite while loop by passing true as the test condition.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main () {
4. **while**(**true**)
5. {
6. cout<<"Infinitive While Loop";
7. }
8. }

Output:

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

Infinitive While Loop

ctrl+c

# C++ Do-While Loop

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The C++ do-while loop is executed at least once because condition is checked after loop body.

1. **do**{
2. //code to be executed
3. }**while**(condition);

**Flowchart:**



## C++ do-while Loop Example

Let's see a simple example of C++ do-while loop to print the table of 1.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i = 1;
5. **do**{
6. cout<<i<<"\n";
7. i++;
8. } **while** (i <= 10) ;
9. }

Output:

1

2

3

4

5

6

7

8

9

10

## C++ Nested do-while Loop

In C++, if you use do-while loop inside another do-while loop, it is known as nested do-while loop. The nested do-while loop is executed fully for each outer do-while loop.

Let's see a simple example of nested do-while loop in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **int** i = 1;
5. **do**{
6. **int** j = 1;
7. **do**{
8. cout<<i<<"\n";
9. j++;
10. } **while** (j <= 3) ;
11. i++;
12. } **while** (i <= 3) ;
13. }

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

## C++ Infinitive do-while Loop

In C++, if you pass **true** in the do-while loop, it will be infinitive do-while loop.

1. **do**{
2. //code to be executed
3. }**while**(**true**);

## C++ Infinitive do-while Loop Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **do**{
5. cout<<"Infinitive do-while Loop";
6. } **while**(**true**);
7. }

Output:

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

Infinitive do-while Loop

ctrl+c

# C++ Break Statement

The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

1. jump-statement;
2. **break**;

**Flowchart:**



## C++ Break Statement Example

Let's see a simple example of C++ break statement which is used inside the loop.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main() {
4. **for** (**int** i = 1; i <= 10; i++)
5. {
6. **if** (i == 5)
7. {
8. **break**;
9. }
10. cout<<i<<"\n";
11. }
12. }

Output:

1

2

3

4

## C++ Break Statement with Inner Loop

The C++ break statement breaks inner loop only if you use break statement inside the inner loop.

Let's see the example code:

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. **if**(i==2&&j==2){
8. **break**;
9. }
10. cout<<i<<" "<<j<<"\n";
11. }
12. }
13. }

Output:

1 1

1 2

1 3

2 1

3 1

3 2

3 3

# C++ Continue Statement

The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

1. jump-statement;
2. **continue**;

## C++ Continue Statement Example

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=10;i++){
6. **if**(i==5){
7. **continue**;
8. }
9. cout<<i<<"\n";
10. }
11. }

Output:

1

2

3

4

6

7

8

9

10

## C++ Continue Statement with Inner Loop

C++ Continue Statement continues inner loop only if you use continue statement inside the inner loop.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **for**(**int** i=1;i<=3;i++){
6. **for**(**int** j=1;j<=3;j++){
7. **if**(i==2&&j==2){
8. **continue**;
9. }
10. cout<<i<<" "<<j<<"\n";
11. }
12. }
13. }

Output:

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3

# C++ Goto Statement

The C++ goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

## C++ Goto Statement Example

Let's see the simple example of goto statement in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. ineligible:
6. cout<<"You are not eligible to vote!\n";
7. cout<<"Enter your age:\n";
8. **int** age;
9. cin>>age;
10. **if** (age < 18){
11. **goto** ineligible;
12. }
13. **else**
14. {
15. cout<<"You are eligible to vote!";
16. }
17. }

Output:

You are not eligible to vote!

Enter your age:

16

You are not eligible to vote!

Enter your age:

7

You are not eligible to vote!

Enter your age:

22

You are eligible to vote!

# C++ Comments

The C++ comments are statements that are not executed by the compiler. The comments in C++ programming can be used to provide explanation of the code, variable, method or class. By the help of comments, you can hide the program code also.

There are two types of comments in C++.

* Single Line comment
* Multi Line comment

## C++ Single Line Comment

The single line comment starts with // (double slash). Let's see an example of single line comment in C++.

1. #include <iostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. **int** x = 11; // x is a variable
6. cout<<x<<"\n";
7. }

Output:

11

## C++ Multi Line Comment

The C++ multi line comment is used to comment multiple lines of code. It is surrounded by slash and asterisk (/∗ ..... ∗/). Let's see an example of multi line comment in C++.

1. #include <ostream>
2. **using** **namespace** std;
3. **int** main()
4. {
5. /\* declare and
6. print variable in C++. \*/
7. **int** x = 35;
8. cout<<x<<"\n";
9. }

Output:

35