C++ Basics (Part 1)

Srivaths P

Why you should prefer C++
(For Competitive Programming)

- Efficiency and Speed
- Most popular language for CP
- In-built Data Structures and Algorithms (STL)

Goal

To understand:

- Constants and datatypes in C++
- Input/Output in C++
- Various C++ operators
- Conditional statements
- Loops

Be able to write simple programs at the end, such as a prime number checker.

Simplest C++ program

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

int main() {
    cout << "Hello world!" << endl;
}
```

Constants in C++

- Integer constants: 4 | 62 | -90
- Decimal constants: 3.14 | 12.0 | 0.33333
- Character constants: 'f' | '5' | '~' | '\n'
- String literal: “Hello :D” | “MyP@ssw0rd123!”

Output in C++

To output a value, we use the cout operator as follows: cout << value;

To print multiple values in the same line:
cout << value1 << value2 << value3;

To start printing in a new line: endl or '\n'

Arithmetic operators in C++

Arithmetic Operators:

1) + Addition
2) - Subtraction
3) * Multiplication
4) / Division (Quotient)
5) % Modulo (Remainder)

NOTE: C++ follows the BODMAS rule

Variables

Variables are containers that stores specific types of data. They can be modified with the assignment operator “=”

Syntax:

```
datatype variable_name = value;
```

Variables

Variable names cannot:

- Have spaces (use underscore instead)
- Start with a digit
- Be reserved by the compiler
- Already taken by another variable (in the same scope)

NOTE: Keywords/Variables are case sensitive

Datatypes

Datatypes are used to set the “type” of a variable. For example, int is used to declare integer variables.

Two types of datatypes:

- Primitive datatypes
- Derived datatypes

Common Primitive datatypes

1. int (long long int, unsigned int, etc.)
2. char
3. bool
4. float (double, long double)
5. Special type: void

Common Derived datatypes

1. string
2. vector
3. map
4. set
5. priority_queue

Arithmetic Assignment Operators

1. +=
2. -=
3. *=
4. /=
5. %=

Unary Operators

Operators that only need one value/operand are called unary operators.

1. +
2. -
3. ++
4. --

a++ = returns the value, and then increments

++a= increments the value, and then returns

Ex:

```
int a=12, b=12;

cout<<a++<<endl;
cout<<++b<<endl;
cout<<a;
cout<<b;

//output is 12
            13
            13 13
```

a++=>a+=1
a--=>a-=1

a-- = returns the value, and then decrements

--a= decrements the value, and then returns

Ex:

```
a=9;
cout<<--a;
cout<<a;

//output= 8

b=10;
cout<<b--;
cout<<b;
//output: 10
            9
int c=12;
int v=c--;
//v is assigned to 12 and then c is assigned to 11
cout<<v<<" "<<c;

//output: 12  11


Ex:
a=13; b=--a;
//a is decreased to 12 and then assigned to 12
cout<<a<<b;  //output 12 12

//more:
//a--
//what do
//a temp value assigned to 13 then
// a is decreased to 12
//b is assigned to temp value

//same effect as
//b is assigned to a
//a is decresed to 12

but in b=--a; (it has only to step)
//a is decreased to 12 then
//b is assigned to a

so, --a instead of a--, it will a tiny bit faster
```

Input in C++

To output a value, we use the cin operator as follows: cin >> value;

To print multiple values in the same line:
cin >> value1 >> value2 >> value3;

NOTE: Each value must be separated by a space or a new line when taking input.

Check your understanding - 1

1. How will you declare a character equal to exclamatory mark?

2. Take an integer input, and output the value multiplied by 7.

3. Take two values a, b as input, and output three values: a+b and a*b and a/b

   a/b should be a decimal, not an integer

Conditions and Relational Operators

Conditions return a boolean value depending on whether the expression is true or false.

Conditional operators:

==, !=

Relational operators:

<, >, <=, >=

false= 0
true=1

Logical operators

Logical operators perform operations on boolean values or expressions that result in Boolean values.

1. “(expr1) && (expr2)” checks whether BOTH are true.
2. “(expr1) || (expr2)” checks whether EITHER one is true.
3. “!(expr)” returns the OPPOSITE of the result of “expr”

The operators are called AND, OR, NOT operators respectively

Conditional statements

Conditional statements execute a different block of code depending on the boolean value of a condition.

Syntax:

```
if (condition) {
    // something
} else if (another_condition) {
    // something
} else {
    // something
}
```

Check Your Understanding 2

1. Take input of 3 numbers x, y, z and output the maximum using if statements

2. Given marks of a student, grade them from A to D
    1. Between 0 and 30 -> D
    2. Between 30 and 65 -> C
    3. Between 65 and 90 -> B
    4. Between 90 and 100 -> A
    5. Output “Error” if less than 0 or greater than 100.

Loop

Loops are used to repeat a block of code until some condition is satisfied.

There are three types of loops in C++:

1. for loop
2. while loop
3. do-while loop

Loop (Miscellaneous)

- An iteration is defined as one time the loop gets executed. For example, 3rd iteration is the 3rd time the loop is run.
- “break” statement exits the current/innermost loop when executed.
- “continue” statement skips to the next iteration of the current/innermost loop when executed.

“for” loop

Syntax:

```
for (statement1; statement2; statement3) {
    // Code here
}
```

statement1: Executed once before start of loop.

statement2: Condition of the loop. Loop exits if false.

statement3: Executed after each iteration.

```
int i=0;
for(i=0;i<10;i++){
cout<<i;}

for(i=0;i<10;){
cout<<i; ++i;}

for(;i<10;){
cout<<i;i++;}

// all have same output
```

Ex:
```
for(int i=1;(i<10) &&(i%2==0);++i){
    cout<<i;}
```

“while” loop

Syntax:

```
while (condition) {
    // Code here
}
```

Check if the condition is true and then execute the block of code. Repeat.

“do-while” loop

Syntax:

```
do {
    // Code here
} while (condition);
```

Execute the block of code and then check if the condition is true. Repeat.

```
for(int i=0;i<10;i++){
   for(int j=1;j<10;j++){
          if(j==5) break;}
//break work the come here
}
//not come here
```

Scope

A scope is a region of the program.

Every pair of curly braces creates a new scope.

The variables inside the scope cannot be used outside the scope.

```
 b=5;
int main(){
int b=10;
b=123;
cout<<b;
cout<<::b;

::b=23;
cout<<b<<::b;
}
//output: 123  5  123 23
```

Miscellaneous

- A loop inside another loop is called nested loops.
  Syntax:

```
for (s1; s2; s3) {
    for (s4; s5; s6) {
        // Code here
    }
}
```

- Infinite loops are loops that run forever and never end (when the condition is always true)

goto statements

Goto/Jump statements are used to skip to another part of the code.

Considered as bad practice to use goto statements except if it used to exit from a nested loop.

Syntax:

```
label: // creates the label to skip to

goto label; // skips to the specific label
```

In C++ programming, the goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax of goto Statement

```
goto label;
... .. ...
... .. ...
... .. ...
label:
statement;
... .. ...
```

In the syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.

```
// This program calculates the average of numbers entered by the user.
// If the user enters a negative number, it ignores the number and
// calculates the average number entered before it.

# include <iostream>
using namespace std;

int main()
{
   float num, average, sum = 0.0;
   int i, n;

   cout << "Maximum number of inputs: ";
   cin >> n;

   for(i = 1; i <= n; ++i)
   {
      cout << "Enter n" << i << ": ";
      cin >> num;

      if(num < 0.0)
      {
         // Control of the program move to jump:
         goto jump;
      }
      sum += num;
   }

jump:
   average = sum / (i - 1);
   cout << "\nAverage = " << average;
   return 0;
}
```

Output

```
Maximum number of inputs: 10
Enter n1: 2.3
Enter n2: 5.6
Enter n3: -5.6

Average = 3.95
```

You can write any C++ program without the use of goto statement and is generally considered a good idea not to use them.

Reason to Avoid goto Statement
The goto statement gives the power to jump to any part of a program but, makes the logic of the program complex and tangled.

In modern programming, the goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C++ program with the use of break and continue statements.

link read more: https://www.programiz.com/cpp-programming/goto

Check Your Understanding 3

1. Find the sum of the first N natural numbers (Using loops)

2. For the first N natural numbers:

   If number is divisible by 3 and 5, print FizzBuzz
   If number is divisible by 3, print Fizz
   If number is divisible by 5, print Buzz

3. Print a N x M grid similar to the following:
   1 2 3 4
   5 6 7 8
   9 10 11 12

Exercise

Write a program to take a number N as an input, and output whether it is a prime number or not.

(Do not worry about efficiency)

Resources

- https://www.programiz.com/cpp-programming (learning C++ in general)
- https://www.programiz.com/cpp-programming#flow-control (if-else and loops)
- https://www.programiz.com/cpp-programming/nested-loops (nested loops)
- https://www.programiz.com/cpp-programming/goto (goto statements)

Thank you!