CS-101 Programming Fundamentals

Lab Manual

Habib University

PUBLISHED BY HABIB UNIVERSITY

BOOK-WEBSITE.COM

First printing, July 2018

Contents

Week 1

1. Variables, Loops, Conditionals and functions

1.1 Objective

In this lab, we'll learn the definition and assignment of variables. We will also learn the implementation and usage of loops and conditionals. Moreover, we will also learn the use of functions.

1.2 Description

1.2.1 Variables

Variables allows to store data to be used in program. In C++, the variable must be declared with it's data type. Once declared, data of the declared data type can be assigned to the variable (See Example 1.1)
The table below shows size, ranges and the syntax in C++ of some data types.

Data type	Size (in bytes)	Range	Keyword
Character	1	-128 to 127	char
Integer	4	-2147483648 to +2147483647	int
Boolean	1 bit	true/false	bool
Floating point	4	-3.4e-38 to +3.4e-38	float
Double floating point	8	1.7e-308 to 1.7 e+308	double

Table 1.1: Properties of Data Types

■ **Example 1.1**

```
#include <iostream>
int main()
{
    //declaring integers
    int number1;
    number1 = 4;
    int number2 = 5; //alternative way

    //Declaring character
    char character1 = 'a';

    //Declaring bool
    bool boolean1 = true;
    bool boolean2 = false;

    //Declaring float
    float fpoint = 1.777;

    //Declaring double
    double dpoint = 4.99999977787878686857676899;
    return 0;
}
```

Declaring Variables

■

Allocation of memory

Computer's memory can be viewed as a series of cubbyholes. Each cubbyhole is one of many such holes all lined up. Each cubbyhole or memory location is numbered sequentially. These numbers are known as memory addresses. A variable reserves one or more addresses in which a binary value is stored. Each address is mostly one byte (8 bits large)

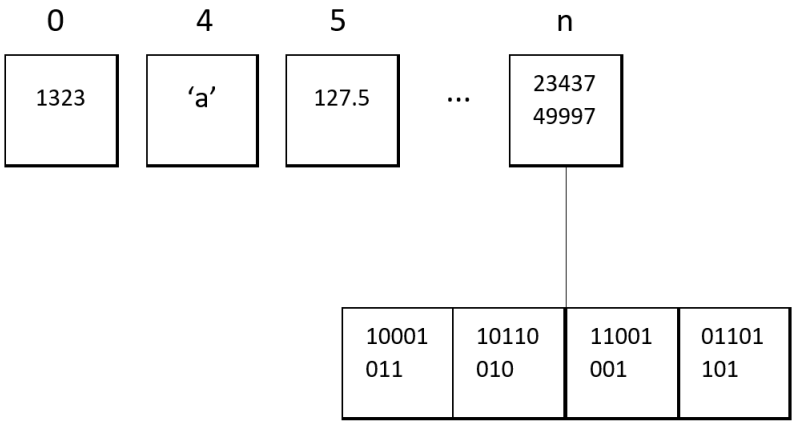

Figure 1.1: Memory Allocation of various data types

In Figure 1.1, the integer '1323' has taken 4 cubby holes or blocks sequentially from address '0' to '3'. Since a character is of 1 byte, character 'a' has taken only one cubby hole of address '4'. The integer '234749997' has taken 4 cubby holes. Each cubby hole is of 8 bits or 1 byte. Therefore the binary representation of the integer is stored sequentially in the cubby holes.

1.2.2 Conditional statements

Conditional statements are used to make decisions based on a given condition.

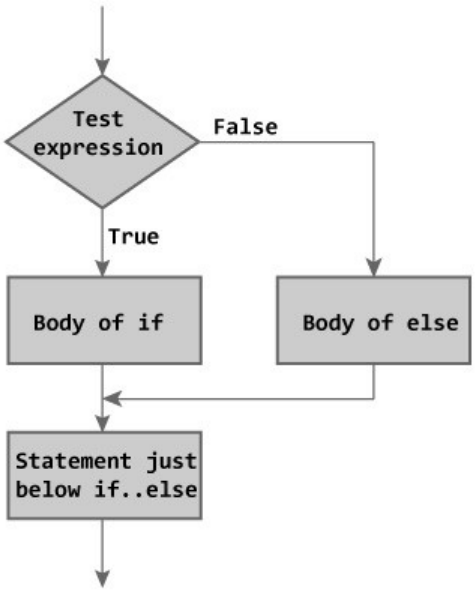

Figure 1.2: Flowchart for if-else conditionals

Comparison operators with truth value

Conditional statements works on the basis of the truth value of the expressions used. There are multiple comparison operators which returns a truth value in boolean and are listed below.

Operator	Description	Usage Example
==	equal to	1 == 1 → true 't' == 'p' → false
!=	not equal to	5! = 5 → false 4.6! = 6.7 → true
<	less than	5 < 4 → false 'a' < 'b' → true (using ASCII value)
<=	less than or equal to	4.5 <= 4.5 → true 3 <= 5.7 → true
>=	greater than or equal to	'b' >= 'b' → true true >= false → true

Table 1.2: Comparison operators

Do not get confused between the assignment operator and equal to operator when using in an if conditional statement

The example below shows an example usage of if conditionals and comparison operators.

■ **Example 1.2**

```
#include <iostream>
using namespace std;
int main()
{
int num; //variable to store input
cout<<"Enter number: "<<endl; //To show prompt
cin>>num; //Taking and storing input value
if(num % 2 == 0) //"a%b" to check the remainder when 'a' is  divided by 'b'
{
cout<<"Number is divisible by 2"<<endl;
}
else
{
cout<<"Number is not divisible by 2"<<endl;
}
return 0;
}
```

Using if condition to check if number is divisible by 2

■

1.2.3 Loops

Loop is a sequence of instructions being repeated until a specific condition or target is reached.

There are three types of loops in C++:

While loop

A while loop statement repeatedly executes a target statement as long as a given condition is true. The syntax of a while loop loop in C++ is:

```
while (testExpression)
{
    statement(s);
}
```

where, **testExpression** is checked on each entry of the while loop.

Here is the flow of control in a while loop:

- The while loop evaluates the **test expression**. If the **test expression** is true, codes inside the body of while loop is evaluated.
- Then, the **test expression** is evaluated again. This process goes on until the **test expression** is false.
- When the **test expression** is false, while loop is terminated.

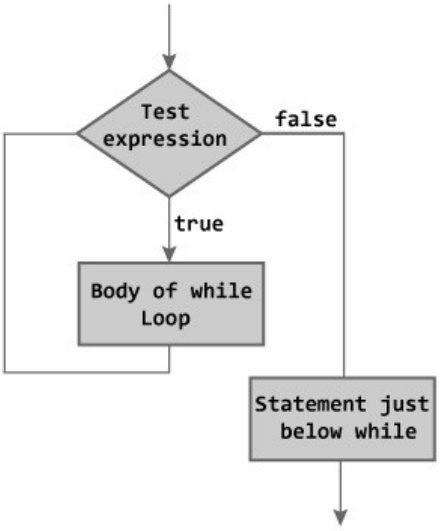

Figure 1.3: Flowchart of while loop

■ **Example 1.3**

```
#include <iostream>
int main()
{
    int num = 1; //variable to be incremented
    int count = 0; //to keep track of number of iterations
    while(count < 5)
    {
        num = num + 1; //or num++ can also be used to increment
        count++;
    }
    return 0;
}
```

Using While loop to increment an integer 5 times

■

Do-while loop

The syntax of do..while loop is:

```
do {
    // codes;
}
while (testExpression);
```

Here is the flow of control in a while loop:

- The codes inside the body of loop is executed at least once. Then, only the **test expression** is checked.
- If the **test expression** is true, the body of loop is executed. This process continues until the **test expression** becomes false.
- When the **test expression** is false, do...while loop is terminated.

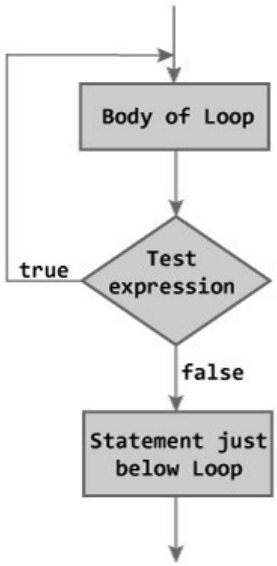

Figure 1.4: Flowchart of do-while loop

■ Example 1.4

```
#include <iostream>
int main()
{
    int num = 1; //variable to be incremented
    int count = 0; //to keep track of number of iterations
    do
    {
        num = num + 1; //or num++ can also be used to increment
        count++;
    }
    while(count<5)
    return 0;
}
```

Using While loop to increment an integer 5 times

■

For loop

A for loop allows you to efficiently write a loop that needs to execute a specific number of times. The syntax of a for loop in C++ is:

```
for(initializationStatement; testExpression; updateStatement)
{
    statement(s);
}
```

Here is the flow of control in a for loop:

- The **initialization statement** is executed only once at the beginning.
- Then, the **test expression** is evaluated. If the **test expression** is false, for loop is terminated. But if the **test expression** is true, codes inside body of for loop is executed and **update statement** is executed.
- Again, the **test expression** is evaluated and this process repeats until the **test expression** is false.

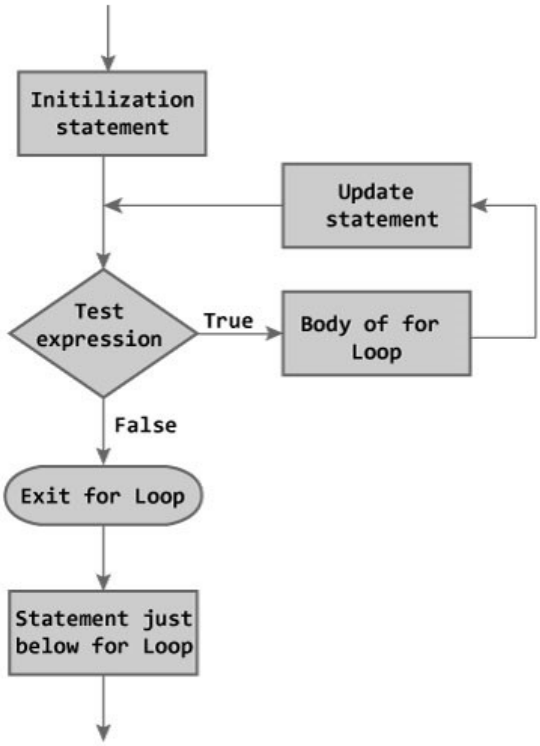

Figure 1.5: Flowchart of for loop

■ **Example 1.5**

```
#include <iostream>
int main()
{
    int num = 1; //variable to be incremented
    for(int i = 0; i < 5; i++)
    {
        num = num + 1; //or num++ can also be used to increment
    }
    return 0;
}
```

Using For loop to increment an integer 5 times

■

■ **Example 1.6**

```
#include <iostream>
int main()
{
    int num = 1; //variable to be incremented
    int count = 0; //to keep track of number of iterations
    while(count < 5)
    {
        num = num + 1; //or num++ can also be used to increment
        count++;
    }
    return 0;
}
```

Using While loop to increment an integer 5 times

■

1.2.4 Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is such that each function performs a specific task.

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

Defining a Function

The general form of a C++ function definition is as follows:

```
return_type function_name( parameter1, parameter2, parameter3,... )
{
    body of the function
}
```

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

- **Return Type** - A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

- **Function Name** - This is the actual name of the function. The function name and the parameter list together constitute the function signature.

- **Parameters** - A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

- **Function Body** - The function body contains a collection of statements that define what the function does.

Declaring a function

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

```
return_type function_name( parameter list );
```

Calling a function

While creating a C++ function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.

When a program calls a function, program control is transferred to the called function. A called function performs defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value.

■ **Example 1.7**

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

// function declaration
int max(int num1, int num2);

int main ()
{
    // local variable declaration:
    int a = 100;
    int b = 200;
    int ret;

    // calling a function to get max value.
    ret = max(a, b);
    cout << "Max value is : " << ret << endl;

    return 0;
}

// function returning the max between two numbers
int max(int num1, int num2)
{
    // local variable declaration
    int result;

    if (num1 > num2)
    result = num1;
    else
    result = num2;

    return result;
}
```

Declaration, definition amd calling of a max function

■

1.3 Problems

Problem 1.1 A certain grade of steel is graded according to the following conditions:

- Hardness must be greater than 50
- Carbon content must be less than 0.7
- Tensile strength must be greater than 5600

The grades are as follows:

- Grade is 10 if all three conditions are met
- Grade is 9 if conditions (i) and (ii) are met
- Grade is 8 if conditions (ii) and (iii) are met
- Grade is 7 if conditions (i) and (iii) are met
- Grade is 6 if only one condition is met
- Grade is 5 if none of the conditions are met

Write a program, which will require the user to give values of hardness, carbon content and tensile strength of the steel under consideration and output the grade of the steel.

Instructions

- Use if-else conditional statements

Problem 1.2 Two frogs named 'FrogPrime' and 'Frogatron' are in a well of depth 1000 feet. They decide to race each other to the top to see who is the winner.

Both frogs can jump 4 feet at a time but slide down 1 foot every time, thus making the total distance covered to be 3 feet.

FrogPrime has a 2 % chance to get an adrenaline rush and jump 5 feet. Frogatron has special claws that have a 2% chance to grab on to a wall, thus it does not slide down 1 foot if the claws connect

The frogs will keep on jumping till one of them clears the well and is declared a winner.

Your program should simulate this behavior. It should find out the winner and report it. It should also report if there is a tie.

It should also display the progress of the frogs at every 50th jump.

It should also display the total number of jumps once the competition is over.

The output should be similar to this:

```
Distance covered by frogPrime: 152
Distance covered by frogatron: 150
Distance covered by frogPrime: 302
Distance covered by frogatron: 301
Distance covered by frogPrime: 454
Distance covered by frogatron: 451
Distance covered by frogPrime: 604
Distance covered by frogatron: 602
Distance covered by frogPrime: 754
Distance covered by frogatron: 752
Distance covered by frogPrime: 905
Distance covered by frogatron: 902
FrogPrime won

Process returned 0 (0x0)   execution time : 0.424 s
Press any key to continue.
```

Instructions

- You can use random number generation to simulate chances. Find out yourself about using random number generation and ranges.
- Think in terms of loops and if-else conditionals

Problem 1.3 Write a function IsPrime(int n) to test whether a parameter is prime. Apply the function in a program which prints all the prime numbers up to 100.

Instructions

- Use modulo operator to check divisibility
- You can use bool to keep check if a number is prime
- Break the question in parts and work on the function first

Problem 1.4 Write a function Fibonacci(int n) which calculates the nth Fibonacci number. Write a main program that uses this function and the function from Problem 1.3 to print out the first 5 Fibonacci numbers that are also primes.

Instructions

- Think in terms of loops and an extensive use of variables to keep track of the needed terms in every iteration

1.4 Feedback

Please write the things you've learned from this lab and suggestions to make it more better and easy to learn.