Lecture 1

February 23, 2021

[9]:
```
!jt -t -r -tf firacode -tfs 15 -mathfs 110 -fs 13 -N -T
```

1 Basic of c++

---

probably the best way to start a programing language is by writitng program

---

[3]:
```
//First c++  program
/*
    TODO:
    1.import iostream
    2.define main function
    3. write Hello world ! to the screen
    4. return
    5. run
*/

// directive to the preprossor
#include<iostream>


int main()
{
    std::cout<<"Hello Ethiopia!";


    return 0;
}
```

[2]:
```
main();
```

```
Hello Ethiopia!
```

fundamental compenent of every C++ program

---

comment

directives to the preprossor

functions

line statements

return

---

[4]:
```
#include <iostream>
using namespace std;
```

[19]:
```
int main()
{
    cout<<"Hello Ethiopia!\n";

    int a = 4;
    int b= 3;

    int c = mul(a,b);
    cout<<c;

    return 0;
}
main()
```

```
Hello Ethiopia!
12
```

[19]:
```
0
```

[6]:
```
int add( int a, int b){
    return a+b;
}
```

[7]:
```
int sub(int a, int b){
    return a-b;
}
```

[8]:
```
int mul(int a, int b){
    return a*b;
}
```

[13]:
```
int dived(int a, int b){
    if(b>0){
        return a/b;
    }else {
```

```
        return 0;
    }
}
```

[ ]:

The program has been structured in different lines in order to more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. for example

---

[20]:
```
int main()
{
    std::cout<<"Hello Addis Ababa!";

    return 0;
}
main();
```

```
Hello Addis Ababa!
```

[21]:
```
int main(){ std::cout<<"Hello Addis Ababa2!"; return 0;} main();
```

```
Hello Addis Ababa2!
```

In C++, the separation between statements is specified with an ending of semicolon(;) at the end of each one, so the separation in defferent code lines does not matter at allfor this purpose. We can write many statements per line or write a single statement that takes many code lines. the division of code in different lines serves only to make it more legible and schematic for the humans that may read it.

Lets us add additional instruction to our first program

:

[1]:
```
#include<iostream>

int main(){

    std::cout<< "Hello World! ";
    std::cout<<"I Write My Second C++ Program.";

    return 0;
}

main();
```

```
Hello World! I Write My Second C++ Program.
```

[14]:
```
#include<iostream>

int main(){ std::cout<< "Hello World! "; std::cout<<"I Write My Second C++␣
 ↪Program.";return 0;}

main();
```

```
Hello World! I Write My Second C++ Program.
```

[15]:
```
#include<iostream>

int main(){

    std::cout<<
        "Hello World! ";
    std::cout<<
        "I Write My Second C++ Program.";

    return 0;
}

main();
```

```
Hello World! I Write My Second C++ Program.
```

1.1 Comments

Comments are parts of the source code disregarded by the compiler. They simply do noting. their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code.

---

[16]:
```
// line comment
/* block comments*/
```

[28]:
```
/*
 my second program in c++
 with more comments.
*/

#include<iostream>

//define main function
int main(){
```

```
    std::cout<< "Hello\n World! ";  //print hello world!
    std::cout<<"I Write My Second C++ Program."; // print I write my second c++␣
↪Program.

    return 0;
}

main();
```

```
Hello
 World! I Write My Second C++ Program.
```

if you include comments within the source code of your program without using comment characters combinations //,/* *//, the compilter will take it as C++ expression, most likely causing one or more several error messages when you compile it.

Variables, Data Types

Let us think I ask you to retain the number 5 in your mental memory. and then I ask you to memorize also the number 2 at the same time. You have just stored two different values in your memory. Now, if I ask you to add 1 to the first numer I said. you should be retaining the number 6 (that is 5+1) and 2 in your memory. values that we could now for example substract and obtaine 4 as result.

[18]:
```
// lets see how computer does it

int a = 5;
int b = 2;

a = 5+1 ;// a is now 6
int result = a - b;
```

[19]:
```
std::cout<< a <<" "<<b <<" "<<result;
```

```
6 2 4
```

[39]:
```
int a = 5;
int b = 6;
```

[43]:
```
std::cout<< "sum of "<<a<<" AND "<< b <<" is "<<a+b;
```

```
sum of 5 AND 6 is 11
```

Therefore, we can define a variable as a portion of memory to store a determined value.

Each variable needs an identifier that distinguishes it from the others, for example, in the previous code the variable identifiers were a, b and result, but we could have called the variables any names we wanted to invent, as long as they were valid identifiers.

Identifiers

- A valid identifier is a sequence of one or more letters.
- digits or underscore characters (_).
- Only letters, digits and single underscore characters are valid.
- variable identifiers always have to begin with a letter.
- Neither spaces nor punctuation marks or symbols can be part of an identifier
- In no case they can begin with a digit
- Another rule can't match any keyword of the C++ language nor compile specific ones.

---

asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete,do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private,protected, public, register,reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template,this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void,volatile, wchar_t, while

---

and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq

Very important: The C++ language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the RESULT variable is not the same as the result variable or the Result variable. These are three different variable identifiers.

[ ]:

[ ]:

Fundamental data types

When programming, we store the variables in our computer's memory, but the computer has to know what kind of data we want to store in them, since it is not going to occupy the same amount of memory to store a simple number than to store a single letter or a large number, and they are not going to be interpreted the same way.

[24]:
```
#include <iostream>

using namespace std;

cout<< "Size of char : " <<sizeof(char) <<"\n";
cout<< "Size of int : " <<sizeof(int) <<"\n";
```

```
cout<< "Size of short int : " <<sizeof(short int) <<"\n";
cout<< "Size of long int : " <<sizeof(long int) <<"\n";
cout<< "Size of float : " <<sizeof(float) <<"\n";
cout<< "Size of double : " <<sizeof(double) <<"\n";
cout<< "Size of wchar_t : " <<sizeof(wchar_t) <<"\n";


return 0;
```

```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 8
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

Declaration of variables

In order to use a variable in C++, we must first declare it specifying which data type we want it to be. The syntax to declare a new variable is to write the specifier of the desired data type (like int, bool, float…) followed by a valid variable identifier. For example:

```
int a;
float mynumber;
```

[25]:
```
int a ;
float mynumber;
```

If you are going to declare more than one variable of the same type, you can declare all of them in a single statement by separating their identifiers with commas. For example:

```
int a, b, c;
```

[1]:
```
int b,c,d;
```

This declares three variables (a, b and c), all of them of type int, and has exactly the same meaning as:

[2]:
```
int a;
int b;
int c;
```

[3]:
```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

Example

[9]:
```
// operating with variables
#include <iostream>

int main ()
{
 // declaring variables:
 int a, b;
 int result;

 // process:
 a = 5;
 b = 2;
 a = a + 1;

 result = a - b;
 // print out the result:

  std::cout <<result;
 // terminate the program:
 return 0;
}

main();
```

4

Scope of variables

All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code, like we did in the previous code at the beginning of the body of the function main when we declared that a, b, and result were of type int.

[12]:
```
#include <iostream>
//using namespace std;

int main () {
   // Local variable declaration:
   int a, b;
   int c;

   // actual initialization
   a = 10;
   b = 20;
   c = a + b;

   std::cout << c;

   return 0;
```

```
}
main();
```

30

[20]:
```
#include <iostream>
using namespace std;
// Global variable declaration:
int g;
```

[21]:
```
int main () {

   // Local variable declaration:
   int a, b;

   // actual initialization
   a = 10;
   b = 20;

   g = a + b;

   cout << g;

   return 0;
}
main();
```

30

[23]:
```
#include <iostream>
using namespace std;

// Global variable declaration:
int g = 20;
```

[24]:
```
int main () {
   // Local variable declaration:
   int g = 10;

   cout << g;

   return 0;
}

main();
```

10

The scope of local variables is limited to the block enclosed in braces ({}) where they are declared. For example, if they are declared at the beginning of the body of a function (like in function main) their scope is between its declaration point and the end of that function.

Initialization of variables

When declaring a regular local variable, its value is by default undetermined. But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable. There are two ways to do this in C++:

```
type identifier = initial_value ; int a = 0; type identifier (initial_value) ; int
a (0)
```

[2]:
```
#include <iostream>
using namespace std;
```

[3]:
```
int main ()
{

 int a=5; // initial value = 5
 int b(2); // initial value = 2
 int result; // initial value undetermined
 a = a + 3;
 result = a - b;
 cout << result;

 return 0;
}
main();
```

```
6
```

Introduction to strings

```
<p style="font-size:120%; font-family:Monospace;">Variables that can store non-numerical va
```

The C++ language library provides support for strings through the standard string class. This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage.

[4]:
```
// my first string
#include <iostream>
#include <string>
using namespace std;
```

[5]:
```
int main ()
{
 string mystring = "This is a string";
 cout << mystring;
 return 0;
}
main();
```

```
This is a string
```

[ ]:
```

```

[6]:
```
// my first string
#include <iostream>
#include <string>
using namespace std;
```

[7]:
```
int main ()
{
 string mystring;
 mystring = "This is the initial string content";
 cout << mystring << endl;
 mystring = "This is a different string content";
 cout << mystring << endl;
 return 0;
}
main();
```

```
This is the initial string content
This is a different string content
```

Constants

constants are expressions with a fixed value

[9]:
```
const int a = 5;
```

[10]:
```
a
```

[10]: 5

[12]:
```
a =10;
```

[13]:
```
int a = 75u;
```

[17]:
```
a = 75l;
a
```

[17]:
```
75
```

Literals

Literals are used to express particular values within the source code of a program

[25]:
```
75 ;// int
75u; // unsigned int
75l; // long
75ul; // unsigned long
```

[25]:
```
75
```

[26]:
```
//floating point Number


3.14159; // 3.14159
6.02e23; // 6.02 x 10^23
1.6e-19; // 1.6 x 10^-19
3.0;     //3.0

3.14159L; // long double
6.02e23f; // float
```

[26]:
```
6.02000e+23f
```

[27]:
```
// character and string literals

'z';
'p';
"Hello world";
"How do you do?";
```

Notice that to represent a single character we enclose it between single quotes (') and to express a string (which generally consists of more than one character) we enclose it between double quotes (").

[30]:
```
#include <iostream>
#include <string>
using namespace std;
```

[32]:
```
//New Line (\n):
cout<<"COMPUTER\nSCIENCE";
```

```
COMPUTER
SCIENCE
```

[34]:
```
//Tab (\t):
cout<<"COMPUTER\tSCIENCE";
```

```
COMPUTER        SCIENCE
```

[41]:
```
//Single Quote (\'):
cout<<"\'COMPUTER SCIENCE\'";
```

```
'COMPUTER SCIENCE'
```

Defined constants (#define)

---

the Boolean literals true and false.

You can define your own names for constants that you use very often without having to resort to memoryconsuming variables, simply by using the #define preprocessor directive. Its format is:

```
#define identifier value
```

[43]:
```
#define PI 3.14159
#define NEWLINE '\n'
```

[ ]:
```

```

[1]:
```
// defined constants: calculate circumference
#include <iostream>
using namespace std;

#define PI 3.14159
#define NEWLINE '\n'
```

[2]:
```
int main ()
{
 double r=5.0; // radius
 double circle;

 circle = 2 * PI * r;

 cout << NEWLINE;
 cout << circle;
 cout << NEWLINE;
 cout << NEWLINE;

 return 0;
}
main();
```

```
31.4159
```

Declared constants (const)

[3]:
```
const int pathwidth = 100;
const char tabulator = '\t';
```

[ ]:
```

```

Operators

[ ]:
```

```