Lecture 2 - C++ Basics

Processing a C++ Program

1. Text editor to write the source code
2. Include preprocessor directives (start with a #)
3. Compile the code (checks for syntax errors then translates to machine language)
4. Execute the program (linker, then loader and then execution)
    - Linker - combines your program with any libary code needed
    - Loader - Loads executable into main memory so that it can run!

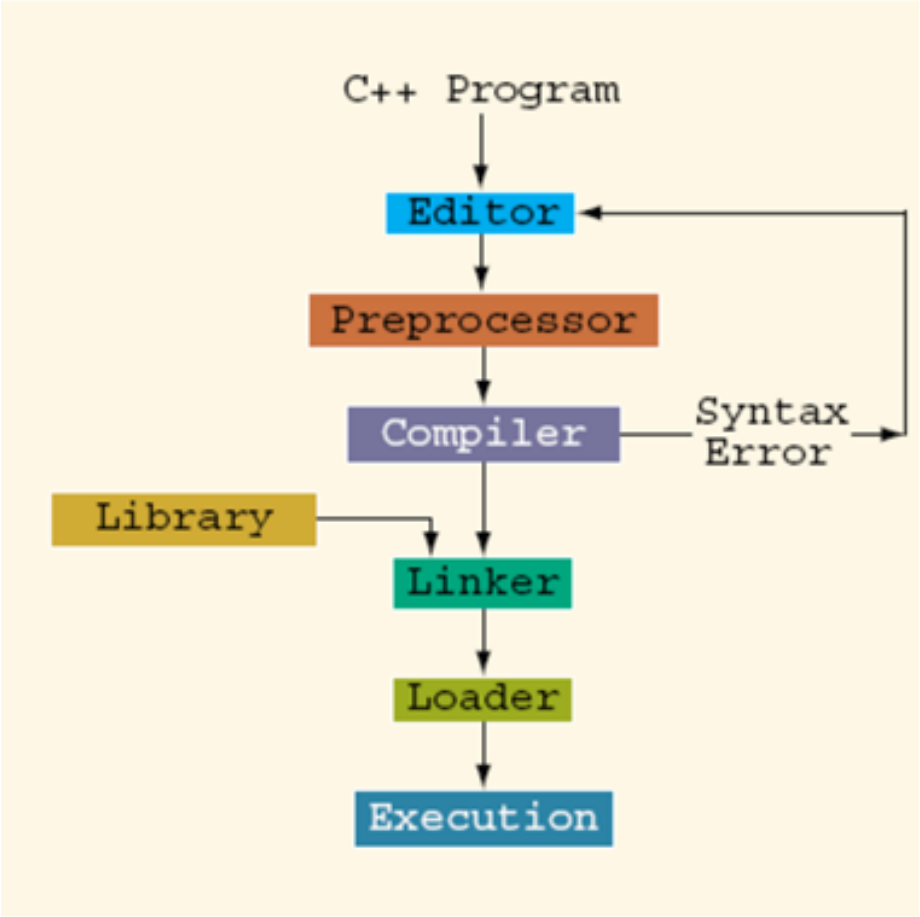

Figure 1: image

I/0 (Standard in and out)

`std::cout` will print to standard out (the terminal)

`std::cin` will read in input from standard in (the terminal again)

`<<` stream insertion operator (put stuff into the stream to print out)

`>>` stream extraction operator (take stuff out of the stream and put it into your program)

Can be cascaded. For example, in the following code, if the numbers 4 and 5 are entered on the terminal by the user then the variable `a` will hold 4 and the variable `b` will hold 5.

```
int a(0), b(0);
std::cin >> a >> b;
```

Note about `std::`

This `std::` is specifying that we want to use the `cout` or `cin` from the std namespace. You will also see in C++ code the statement `using namespace std`. This is, in general, a bad practice because it will potentially cause unintended issues later if you want to reference a function of the same name from another context. We can potentially make things a bit easier by using a typedef statement.

```
typedef std::cout cout_std;
```

Now we can refer to just cout_std instead of std::... but... that is kind of the same amount of typing so, up to you.

Basic Data Types

The following are the basic data types in C++:

Note that this table says 1 bit for a bool, but in reality, it is actually a full byte

auto keyword

(also sometimes called type inference) is a feature that allows the compiler to deduce the type of an object from the object's initializer.

In the example below, the variable `a` has the type of int because we assigned a number `1`:

```
auto a = 1;
```

Data type	Size(bytes)	Range
char	1	-128 to 127
unsigned char	1	0 to 255
short	2	-32,768 to 32,767
unsigned short	2	0 to 65535
int	4	-2147483648 to +2147483647
unsigned int	4	0 to 4294967295
long	4	-2147483648 to +2147483647
Unsinged long	4	0 to 4294967295
float	4	-3.4e-38 to +3.4e-38
double	8	1.7 e-308 to 1.7 e+308
long double	8	1.7 e-308 to 1.7 e+308
bool	1 bit	
void	-	-
wchar_t	2 or 4	1 wide character

Figure 2: image

We will use this auto keyword a lot more when talking about iterators and the standard template library. It will make lives easier. It will mean a slightly slower compilation time, but not much.

Arrays

Primative arrays in C++ can be defined using the following general form:

```
type name[] = {init value(s)
```

For example, the following would create an array of 3 ints with the values 1, 2, and 3 in positions 0, 1, and 2

```
int arr[] = {1,2,3};
```

N-D Arrays can be created by increasing the number of square brackets! For example, this is a 2x3 array (2 rows and 3 columns) with values of 0 for the first row and 1s for the second.

```
int arr[2][3] = { {0,0,0}, {1,1,1} };
```

You can increase to more dimensions as well!

---

if / else statements

Typical if/else if/else statement. Very similar to other languages.

Syntax:

```
if(conditional){
   actions
}
else if(some other conditional){
   actions
}
else {
   actions
}
```

In the following example, "Yay" will be printed if `number` is between 0-1, "Hi" if `number` is between 2-4 and "Boo" if `number` is anything else. If you test this code with `number = 2` then it should print "Hi".

```
int number = 2;
if(number >= 0 && number <=1) {
   std::cout << "Yay";
}
else if(number >= 2 && number <=4) {
   std::cout << "Hi";
}
else {
   std::cout << "Boo";
}
```

---

switch statement

Typical switch statement. Very similar to other languages.

Syntax:

```
switch(variable) {
   case cond1 :
      //do something
      break;
   case cond2 :
      //do something
      break;
   default:
      //do something
```

```
}
```

The following example will switch on a char. Since `c` starts as 'A', "You got an A!" will print.

```
char c = "A";
switch(c) {
   case "A":
      std::cout << "You got an A!";
      break;
   case "B":
      std::cout << "You got an B!";
      break;
   default:
      std::cout << "You did not get an A or B";
      break;
}
```

---

while loops

Typical while loop. Very similar to other languages.

Syntax:

```
init step
while(condition) {
   //do something
   update step
}
```

The example below will print 0 to 9:

```
int i = 0;
while(i < 10) {
   std::cout << i << " ";
   i++;
}
```

Typical do-while loop. Very similar to other languages.

Syntax:

```
init step
do{
   //do something
   update step
}while(condition);
```

The example below will also print 0 to 9:

```
int i = 0;
do{
   std::cout << i << " ";
   i++;
}while(i < 10);
```

Remember that a do while loop will always execute once, whereas a while loop may not if the condition is not satisfied!

---

for loops

Typical for loop. Very similar to other languages.

Syntax:

```
for(init step; condition; update){
   //do something
}
```

The example below will also print 0 to 9:

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

Difference between post and pre increment

The pre-increment (`++i`) will increment first and then print or be assigned to a variable.

In the example below, the value of `i` will be 1 when it prints.

```
int i = 0;
std::cout << ++i;
//i is now equal to 1
```

The post-increment(`i++`) will first print or be assigned to a variable and then increment.

In the example below, the value of `i` will be 0 when it prints. However, it will actually be incremented to 1, so if we printed `i` again we would get a 1.

```
int i = 0;
std::cout << i++;
```

```
//i is now equal to 1
```