C++ Basics

Outline

- Program Structure
- Variables and Data Types
- Operators
- Basic Input/Output (I/O)
- Control Structures
- Functions

Program Structure

```
// my first program in C++
#include <iostream>

int main()
{
   std::cout << "Hello World!“ << std::endl;
}
```

```
# my first program in Python

print(“Hello World!”)
```

How to Compile/Run

- Log in to lumen
- Create your program in a text editor, save it to file
  - You can create it on your local machine using whatever text editor you'd like, and then use winscp to copy it to lumen OR
  - You can use a text editor on lumen to create your file
  - Let's say we named the last program Hello.cpp
- Compile the program
  - g++ Hello.cpp
- Run the program
  - ./a.out
- To compile the program to a different name:
  - g++ -o Hello Hello.cpp
  - Now you can run it like:
    - ./Hello

Comments

```
// single line comment

/* block comment
   spans several lines
*/

/** javadoc style comment
 *
 *
 */
```

Namespaces

```
// my second program in C++
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World! ";
    cout << "I'm a C++ program“ << endl;
}
```

Variables and Types

- Variable names (identifiers)
  - Can include letters, digits or underscores.
  - Must begin with a letter
  - Case sensitive
- Fundamental data types:
  - character
  - integer
  - floating point
  - boolean

Variables and Types

Group	Type names*	Notes on size / precision
Character types	`char`	Exactly one byte in size. At least 8 bits.
	`char16_t`	Not smaller than `char`. At least 16 bits.
	`char32_t`	Not smaller than `char16_t`. At least 32 bits.
	`wchar_t`	Can represent the largest supported character set.
Integer types (signed)	`signed char`	Same size as `char`. At least 8 bits.
	`signed` `short` *`int`*	Not smaller than `char`. At least 16 bits.
	`signed` `int`	Not smaller than `short`. At least 16 bits.
	`signed` `long` *`int`*	Not smaller than `int`. At least 32 bits.
	`signed` `long long` *`int`*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	`unsigned char`	(same size as their signed counterparts)
	`unsigned short` *`int`*	
	`unsigned` *`int`*	
	`unsigned long` *`int`*	
	`unsigned long long` *`int`*	
Floating-point types	`float`	
	`double`	Precision not less than `float`
	`long double`	Precision not less than `double`
Boolean type	`bool`	
Void type	`void`	no storage
Null pointer	`decltype(nullptr)`	

Variables and Data Types

- Declaration
  - C++ is strongly typed – must assign each variable a data type
    - This also tells the compiler how much memory to use
  - Must begin with a letter
  - Case sensitive
- Examples:

```
int a;
float myNumber;
```

- Initialization of variables

```
int x = 0;
// OR
int x;
x = 0;
```

Variables and Types: Integer Types

```
75   // decimal
0113 // octal
0x4b // hexadecimal

75   // int
75u  // unsigned int
75l  // long
75ul // unsigned long
75lu // unsigned long
```

Suffix	Type modifier
u *or* U	unsigned
l *or* L	long
ll *or* LL	long long

Variables and Types: Floating Point

```
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
```

Suffix	Type
f *or* F	float
l *or* L	long double

Variables and Types: Characters and Strings

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

"Left \t Right"
"one\ntwo\nthree"
```

Escape code	Description
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

Strings

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string mystring;
    mystring = "This is a string";
    cout << mystring << endl;
    return 0;
}
```

Variables and Types: Other

```
bool foo = true;
bool bar = false;
int* p   = nullptr;

const double pi = 3.1415926;
const char tab  = '\t';
```

Preprocessor directives:

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

Operators

- Assignment:

```
x = 5;
y = 2 + x;
y = 2 + y;
```

- Arithmetic:

operator	description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Operators

- Compound Operators:

expression	equivalent to...
`y += x;`	`y = y + x;`
`x -= 5;`	`x = x - 5;`
`x /= y;`	`x = x / y;`
`price *= units + 1;`	`price = price * (units+1);`

- Increment / Decrement:

Example 1	Example 2

`x = 3;`
`y = ++x;`
`// x contains 4, y contains 4`
	`x = 3;`
	`y = x++;`
	`// x contains 4, y contains 3`

Operators

- Relational Operators:

operator	description
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- Logical Operators:

operator	short-circuit
&&	if the left-hand side expression is `false`, the combined result is `false` (the right-hand side expression is never evaluated).
||	if the left-hand side expression is `true`, the combined result is `true` (the right-hand side expression is never evaluated).

Operators

- Ternary Operator:

```
condition ? result1 : result2
```

- Examples:

```
7==5 ? 4 : 3      // evaluates to 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3    // evaluates to 4, since 7 is equal to 5+2.
5>3 ? a : b       // evaluates to the value of a, since 5 is greater than 3.
a>b ? a : b       // evaluates to whichever is greater, a or b.
```

Operators

- Bitwise Operators:

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Operators

- Type Casting:

```
int i;
float f = 3.14;
i = (int) f;
```

- Size of a variable (in bytes):

```
x = sizeof (char);
```

Operator Precedence

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

Basic I/O

- Input / Output Streams

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

- Example:

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is 1404.
```

Basic I/O

Input Conversion:

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

```
Enter price: 22.25
Enter quantity: 7
Total price: 155.75
```

Control Structures

- Conditional (if) Statement

```
if (condition) statement
```

```
if (condition) statement1 else statement2
```

- Example:

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else
  cout << "x is 0";
```

Control Structures

- Selection Statement

```
switch (expression)
{
  case constant1:
     group-of-statements-1;
     break;
  case constant2:
     group-of-statements-2;
     break;
  .
  .
  .
  default:
     default-group-of-statements
}
```

- Example:

```
switch (x) {
  case 1:
  case 2:
  case 3:
    cout << "x is 1, 2 or 3";
    break;
  default:
    cout << "x is not 1, 2 nor 3";
  }
```

Control Structures

- Iteration: while loop

```
while (expression) statement
```

- Example:

```
// custom countdown using while
#include <iostream>
using namespace std;

int main ()
{
  int n = 10;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "liftoff!\n";
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

Control Structures

- Iteration: do-while loop

```
do statement while (condition);
```

- Example:

```
// echo machine
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str;
  do {
    cout << "Enter text: ";
    getline (cin,str);
    cout << "You entered: " << str << '\n';
  } while (str != "goodbye");
}
```

```
Enter text: hello
You entered: hello
Enter text: who's there?
You entered: who's there?
Enter text: goodbye
You entered: goodbye
```

Control Structures

- Iteration: for loop

```
for (initialization; condition; increase) statement;
```

- Example:

```
// countdown using a for loop
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "liftoff!\n";
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, liftoff!
```

Control Structures

- Iteration: range-based for loop

```
for ( declaration : range ) statement;
```

- Example:

```
// range-based for loop
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str {"Hello!"};
  for (char c : str)
  {
    cout << "[" << c << "]";
  }
  cout << '\n';
}
```

```
[H][e][l][l][o][!]
```

Functions

- Function Syntax:

```
type name ( parameter1, parameter2, ...) { statements }
```

- Example:

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
  int r;
  r=a+b;
  return r;
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
}
```

```
The result is 8
```

Functions

- Example with no return type:

```
// void function example
#include <iostream>
using namespace std;

void printmessage ()
{
  cout << "I'm a function!";
}

int main ()
{
  printmessage ();
}
```

```
I'm a function!
```

Functions

- Pass by Value

```
int addition (int a, int b)

z = addition (  5  ,  3  );
```

- vs.
- Pass by Reference:

```
void duplicate (int& a,int& b,int& c)

     duplicate (  x  ,  y  ,  z  );
```

Functions

- Pass by Value

```
string concatenate (string a, string b)
{
  return a+b;
}
```

- Pass by Reference:

```
string concatenate (string& a, string& b)
{
  return a+b;
}
```

- Safe and Efficient:

```
string concatenate (const string& a, const string& b)
{
  return a+b;
}
```

Functions

- Default Values in parameters

```
// default values in functions
#include <iostream>
using namespace std;

int divide (int a, int b=2)
{
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << divide (12) << '\n';
  cout << divide (20,4) << '\n';
  return 0;
}
```

```
6
5
```

Functions

- Declaring Functions:
  - Any identifier – function or variable – must be declared before it is used
  - So… can define function above main() function OR
  - Can use a function declaration before code:

```
// declaring functions prototypes
#include <iostream>
using namespace std;

void odd (int x);
void even (int x);

int main()
{
  int i;
  do {
    cout << "Please, enter number (0 to exit): ";
    cin >> i;
    odd (i);
  } while (i!=0);
  return 0;
}

void odd (int x)
{
  if ((x%2)!=0) cout << "It is odd.\n";
  else even (x);
}

void even (int x)
{
  if ((x%2)==0) cout << "It is even.\n";
  else odd (x);
}
```

```
Please, enter number (0 to exit): 9
It is odd.
Please, enter number (0 to exit): 6
It is even.
Please, enter number (0 to exit): 1030
It is even.
Please, enter number (0 to exit): 0
It is even.
```

Summary

- Program Structure
- Variables and Data Types
- Operators
- Basic Input/Output (I/O)
- Control Structures
- Functions