Faculty of Engineering

Sohag University

Chapter 1 C++ Basics

Electrical Engineering Department

1st Year

Programming Languages

Structure of a program

The best way to learn a programming language is by writing programs. Typically, the first program beginners write is a program called "Hello World", which simply prints "Hello World" to your computer screen. Although it is very simple, it contains all the fundamental components C++ programs have:

```cpp
// my first program in C++
#include <iostream>
int main()
{
  std::cout << "Hello
World!";
}
```

```
Hello
World!
```

Structure of a program

Line 1: **// my first program in C++**
Two slash signs indicate that the rest of the line is a comment inserted by the programmer but which has no effect on the behavior of the program. Programmers use them to include short explanations or observations concerning the code or program.
Line 2: #include <iostream>
Lines beginning with a **hash sign (#)** are directives read and interpreted by what is known as the *preprocessor*. **They are special lines interpreted before the compilation of the program itself begins.** In this case, the directive #include <iostream>, instructs the preprocessor to include a section of standard C++ code, known as *header iostream*, that allows to **perform standard input and output operations**, such as writing the output of this program (Hello World) to the screen.
Line 3: A blank line.
Blank lines have no effect on a program. They simply improve readability of the code.

Structure of a program

- **Line 4: int main ()**
- This line initiates the declaration of a function. Essentially, a function is a group of code statements which are given a name: in this case, this gives the name "main" to the group of code statements that follow. Functions will be discussed in detail in a later chapter, but essentially, their definition is introduced with a succession of a type (int), a name (main) and a pair of parentheses (()), optionally including parameters.
  The function named main is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.
- **Lines 5 and 7: { and }**
- The open brace ({) at line 5 indicates the beginning of main's function definition, and the closing brace (}) at line 7, indicates its end. Everything between these braces is the function's body that defines what happens when main is called. All functions use braces to indicate the beginning and end of their definitions.

Structure of a program

- **Line 6: std::cout << "Hello World!";**
- This line is a C++ statement. Statements are executed in the same order that they appear within a function's body.
- This statement has three parts: First, std::cout, which identifies the **st**andar**d** **c**haracter **out**put device (usually, this is the computer screen). Second, the insertion operator (<<), which indicates that what follows is inserted into std::cout. Finally, a sentence within quotes ("Hello world!"), is the content inserted into the standard output.
-

Fundamental data types

- Fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. They can mainly be classified into:
- **Character types:** They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
- **Numerical integer types:** They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
- **Floating-point types:** They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
- **Boolean type:** The boolean type, known in C++ as bool, can only represent one of two states, true or false.

```
/ This line is necessary to be able to output information
to the screen  #include <iostream>

 // The program starts here and carries on line by line 5

int main()

 {

// The list of 4 integer numbers to add together

 int numbers[4] = {1, 5, 9, -3};

 // The sum of the numbers, initialised to zero

 int sum = 0;

 // For each number in the list

  for(int i = 0; i < 4; ++i)

    {
```

```
/* Add the i-th (running from 0 to 3) number
to the sum */
   sum += numbers[i];
  }
   // Output the sum to the screen
 std::cout << sum << std::endl;
  // End the program and send a value of 0
(success) back
  // to the operating system25
  return 0;
 }
```

Fundamental data types

Type sizes above are expressed in bits; the more bits a type has, the more distinct values it can represent, but at the same time, also consumes more space in memory:

Size	**Unique representable values**	**Notes**
8-bit	256	= 2^8
16-bit	65 536	= 2^{16}
32-bit	4 294 967 296	= 2^{32} (~4 billion)
64-bit	18 446 744 073 709 551 616	= 2^{64} (~18 billion billion)

Declaration of variables

```
// operating with variables
#include <iostream>
using namespace std;
int main ()
{
  // declaring variables:
  int a, b;
  int result;
  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;
  // print out the result:
  cout << result;
  // terminate the program:
  return 0;
}
```

Initialization of variables

```
// initialization of variables
#include <iostream>
using namespace std;
int main ()
{
  int a=5;               // initial value: 5
  int b(3);              // initial value: 3
  int c{2};              // initial value: 2
  int result;            // initial value undetermined
  a = a + b;
  result = a - c;
  cout << result;
  return 0;
}
```

Introduction to strings

One of the major strengths of the C++ language is its rich set of compound types.
An example of compound type is the string class. Variables of this type are able to store sequences of characters, such as words or sentences. A very useful feature!

```
// my first string
#include <iostream>
#include <string>
using namespace std;
int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

Constants

Constants are expressions with a fixed value.

Literals

Literals are the most obvious kind of constants. They are used to express particular values within the source code of a program. Theyare used to give specific values to variables or to express messages we wanted our programs to print out, for example, when we wrote:

```
a = 5;
```

The 5 in this piece of code was a ***literal constant***.

Constants

Literal constants can be classified into: integer, floating-point, characters, strings, Boolean, pointers, and user-defined literals.

Integer Numerals

These are numerical constants that identify integer values.
1776
707
-273

Floating Point Numerals

They express real values, with decimals and/or exponents. They can include either a decimal point, an e character (that expresses *"by ten at the Xth height"*, where *X* is an integer value that follows the e character), or both a decimal point and an e character:

Constants

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

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

Special Characters

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 (\)

Operators

Assignment operator (=)

The assignment operator assigns a value to a variable.

```
x = 5;
```

This statement assigns the integer value 5 to the variable x.

```
x = y;
```

This statement assigns to variable **`x`** the value contained in variable **`y`**. The value of **`x`** at the moment this statement is executed is lost and replaced by the value of **`y`**.

Operators

Arithmetic operators (+, -, *, /, %)
The five arithmetical operations supported by C++ are:

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
%	modulo

Operations of addition, subtraction, multiplication and division correspond literally to their respective mathematical operators. The last one, *modulo operator*, represented by a percentage sign (%), gives the remainder of a division of two values.

Operators

Compound assignment (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)

Compound assignment operators modify the current value of a variable by performing an operation on it. They are equivalent to assigning the result of an operation to the first operand:

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);`

Operators

Increment and decrement (++, --)
Some expression can be shortened even more: the increase operator (`++`) and the decrease operator (`--`) increase or reduce by one the value stored in a variable. They are equivalent to `+=1` and to `-=1`, respectively. Thus:

```
++x;
x+=1;
x=x+1;
```

Operators

A peculiarity of this operator is that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable name (++x) or after it (x++).

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 and comparison operators (==, !=, >, <, >=, <=)
Two expressions can be compared using relational and equality operators. The result of such an operation is either true or false (i.e., a Boolean value).
The relational operators in C++ are:

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

Operators

Logical operators (!, &&, ||)

The operator **!** is the C++ operator for the Boolean operation **NOT**. It has only one operand, to its right, and **inverts it**, producing false if its operand is true, and true if its operand is false.

|| OPERATOR (or)		
a	**b**	**a || b**
true	true	true
true	false	true
false	true	true
false	false	false

&& OPERATOR (and)		
a	**b**	**a && b**
true	true	true
true	false	false
false	true	false
false	false	false

Basic Input/Output

C++ uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen, the keyboard or a file. A *stream* is an entity where a program can either insert or extract characters to/from.

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

Basic Input/Output

Standard output (cout)

For formatted output operations, cout is used together with the *insertion operator*, which is written as **<<** (i.e., two "less than" signs).

```
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of
variable Hello
```

Multiple insertion operations (<<) may be chained in a single statement:

```
cout << "This " << " is a " << "single C++ statement";
```

Basic Input/Output

To insert a line break, a new-line character shall be inserted at the exact position the line should be broken. In C++, a new-line character can be specified as `\n` (i.e., a backslash character followed by a lowercase `n`). For example:

```
cout << "First sentence.\n";
cout << "Second sentence.\nThird sentence.";
```

```
First sentence.
Second sentence.
Third sentence.
```

Alternatively, the `endl` manipulator can also be used to break lines.

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

```
First sentence.
Second sentence.
```

The *endl* manipulator produces a newline character, exactly as the insertion of '\n' does

Basic Input/Output

Standard input (cin)

In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.
For formatted input operations, cin is used together with the extraction operator, which is written as **>>** (i.e., two "greater than" signs). For example:

```
int age;
cin >> age;
```

Extractions on **`cin`** can also be chained to request more than one datum in a single statement:

```
cin >> a >> b;
```

Basic Input/Output

cin and strings

The extraction operator can be used on cin to get strings of characters in the same way as with fundamental data types:

```
string mystring;
cin >> mystring;
```

To get an entire line from cin, there exists a function, called getline, that takes the stream (cin) as first argument, and the string variable as second.

Basic Input/Output

```cpp
// cin with strings
#include <iostream>
#include <string>
using namespace std;
int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr <<
   ".\n";
  cout << "What is your favorite
   team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << "
   too!\n";
  return 0;
}
```

```
What's your
   name? Homer
   Simpson
Hello Homer
   Simpson.
What is your
   favorite
   team? The
   Isotopes
I like The
   Isotopes
   too!
```

Basic Input/Output

stringstream

The standard header <sstream> defines a type called **`stringstream`** that allows a string to be treated as a stream, and thus allowing extraction or insertion operations from/to strings in the same way as they are performed on cin and cout. This feature is most useful to convert strings to numerical values and vice versa. For example, in order to extract an integer from a string we can write:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

Basic Input/Output

```cpp
// 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
```

Good luck