

Chapter 1 – Introduction to C++

1. What is a Programming?

A *programming language* is a set of rules for expressing ideas that are to be used as the logic of programs or software. Programmers express ideas within the rules of a programming language in order to create programs. Although often called programs, these files of programmer created code, called *source code*, are not usable by a computer. Instead, these files of programmer created code must be translated into the binary instructions that the processor of a computer can use or execute.

Programs that perform the conversion of code to binary instructions are referred to as *translators*. There are two types of translators, *compilers* and *interpreters*. Interpreters pass binary instructions directly to the processor. For example, BASIC and HTML are programming languages that are most often used with interpreters. Compilers, on the other hand, do not pass binary instructions directly to the processor. Instead, compilers create new files containing binary instructions. These binary instructions can then be reused or executed whenever desired. They can be copied and distributed. The code that was used by a compiler create a file of binary instructions can then be discarded or changed without altering the binary code. Any programming language can be used with a compiler or an interpreter.

Compiler created binary instructions are substantially more efficient than interpreting code into binary instructions. This makes compiled code ideal for large or complex software, such as operating systems or applications programs. Some examples of complied programs are browser programs such as Microsoft's Internet Explorer or Netscape's Communicator and word processor programs such as Microsoft's Word.

While interpreting code into binary is much less efficient than running a compiled program, a file of programmer written code is much smaller than the equivalent file of compiled binary code. This makes interpreted programmer code ideal for situations where the program must be transmitted to a remote computer for execution, such as a program written in HTML for use on the Internet. Internet users run compiled code software, i.e. a browser that contains an interpreter for HTML code that is to be received. When an HTML program is received by a browser, it can be interpreted and run. The inefficiency of running an interpreted program is more than made up for by the reduction in time needed to transmit the HTML code to the user's machine.

2. What is C++?

C++ is, of course, a programming language. C++ was created by Bjarne Stroustrup at the famous AT&T Bell Laboratories in the early 1980's. It is a complex and powerful language that has been in wide use for over 20 years. It is used for a variety of purposes ranging from introducing students to the fundamental ideas of computer science, to production of sophisticated operating systems, and the most commonly used applications programs.

Beyond these generalities, C++ has several features that relate it to and distinguish it from the thousands of other programming languages.

- C++ is a superset of the earlier C programming language. That is, C++ is a substantial expansion of the language C and most parts of C are found in C++.
- C++ is an object oriented 3GL (3rd generation language). Languages are categorized as 1GL - direct coding in binary, 2GL – assembler languages, 3GL – algebraic / procedural languages, 4GL – non-algebraic / non-procedural languages, 5gl – natural language interfaces.
 - C++ is algebraic. It is organized in statements that were developed from the notation of the expressions of algebra. It uses the concepts of symbolic logic developed in algebra.
 - C++ is procedural. A C++ programmer must express operations in terms of specific and detailed descriptions of the procedures, i.e. steps necessary to accomplish a desired goal.
 - C++ is object oriented. *Classes* are the highest level of organization, containing data and algorithms. Instances of classes called *objects* are manipulated by C++ programmers.

3. The C++ Minimal Program

C++, like most 3GL programming languages, requires a minimal structure for a program. Just what a minimal program is depends on the intention of the user. *Standard C++* has different basic requirements than C++ written to run as program in a graphical user interface such as Microsoft Windows. Students using C++ to learn about Computer Science begin with standard C++ and, over time, learn to develop more complex programs.

Standard C++ programs must have a *function* called *main*. Other than that, there are no absolute requirements. Here are two examples of an empty function *main*:

```

void main( )
{
}

int main( )
{
    return 0;
}

```

Most modern compilers will support both of these instances of function *main*, but the one on the left is considered old fashioned and its use is being deprecated. Older compilers may not support the example on the left.

4. Functions

Functions are an essential part of C++ programs. Functions are named units of code that can be invoked or called whenever the programmer desires. All the programmer need do to use an available function is use its name in a program. Functions consist of a *header* and *body*:

header
body

Function headers have three parts:

return-type function-name argument-list

where *return-type* is a data *type* or *class* name, such as *int*, *float*, *char*, etc.
function-name is whatever the creator of the function chooses
argument-list contains the objects to receive information during a call or invocation of the function

Here is an example of a function header for a function that will receive a number then return the square of that number:

double SquareIt(double n)

The body of a function follows the function header and its code is enclosed in curly brackets (often called braces). This has the following form:

header
{
:
statements
:
}

Statements are the step by step instructions that programmers write. They serve as instructions that will be translated into binary code that a processor will be able to execute when a program is run. **All statements in C++ end with a semicolon (;).**

The braces define a *compound statement*. Everything in a compound statement is treated as being one thing. In effect, compound statements make many statements into a single statement.

In many functions, the very last statement executed in the function is the return statement. The return statement has this form:

return *expression*;

where *expressions* are code that evaluates to a result, much like an algebraic expression, but not limited to mathematics

In the return statement, the value calculated for the expression will be returned from the function when the function finishes. This is just what happens when a calculator is used to compute the value of function (such as square root). Here is an example of a function header and body with a return statement:

```
double squareit( double n )
{
    return n * n;
}
```

5. Classes

In C++, *classes* are built to express ideas. In the largest sense, they are used to model the real world. For example, within the computer system of a company, the basic functions of the company could be expressed as a class. This company class could be built of many other classes such as an employee class, a product class, a customer class, and many other classes. In each of these classes, code units called *member functions* are written to express actions that may be taken by and performed on the class. In each of these classes, data is stored to reflect the *state information*. In each of these classes, *member functions* are used to manipulate the state information. The state information of the employee class could include rate of pay and number of dependents. The member functions of the employee class could include operations to report and update the state information, such as calculating the employee's net pay.

There are several types of classes in C++. These include *user-defined classes*, and *standard classes*. The company class, employee class, product class, and customer class of the previous paragraph are all examples of user defined classes. Ultimately, these classes have all, at least in part, been built out of data types and standard classes.

Objects are named instances of classes. Each object has all of the functionality of its class. One way to think of this is to regard the class as a set of blueprints or plans and the object as the finished product. There can be as many objects of one class as needed. For example, in the company example of the first paragraph of this section, one company would have many employees.

6. Types

In C++, there are built-in data types used to represent data and information in a program. (Note: Students familiar with the concept of built-in data types from other languages earlier languages such as Pascal or C will have little problem understanding the built-in data types of C++.) Each object of a built-in type has its own state information and a set of possible operations (member functions). The state information of objects of these types is usually a single value. The operations defined on these types are usually confined to the effects of operators within expressions. Stroustrup defined only 6 fundamental types:

char, short, int, long, float, double

where type *char* includes characters,
types *short*, *int*, and *long* are integers (numbers without fractions)
types *float* and *double* are floating point numbers (i.e. reals, numbers
with fractions expressed in a form of scientific notation)

Properties of these types are listed in the table on the following table (see next page).

Table of the Six Fundamental Types of C++

Type	Description	State Examples	Partial List of Operators	Example Expressions
char	Any character, literal characters must be enclosed in single quotes.	'A' 'a' '*' '2'		'A' 'a' '*' '2'
short	Class of small integers used for conserving memory. Actual size in memory varies between compilers. Short expressions always result in an integer.	1 -6 12 -45	+ (addition and unary) - (subtraction and unary) * (multiplication) / (division) % (remainder)	$4 + 8 - 3$ $12 - 6 / 3$ $6 * 22 * 2$ $3 / 5$
int	Standard class of integers, larger than short. Some compilers make no difference between int and long. Int expressions always result in an integer.	1 -6 12 -45	+ (addition and unary) - (subtraction and unary) * (multiplication) / (division) % (remainder)	$4 + 8 - 3$ $12 - 6 / 3$ $6 * 22 * 2$ $3 / 5$
long	Expanded class of integers, usually taking up extra memory in order to store larger integers. Some compilers make no difference between int and long. Long expressions always result in an integer.	1 -6 12 -45	+ (addition and unary) - (subtraction and unary) * (multiplication) / (division) % (remainder)	$4 + 8 - 3$ $12 - 6 / 3$ $6 * 22 * 2$ $3 / 5$
float	Standard class of reals. Some compilers make no difference between float and long. Float expressions always result in a float.	3.14 -55.0 777.0003 4	+ (addition and unary) - (subtraction and unary) * (multiplication) / (division)	$3.14 * 44.0$ $-23.8 / 1.2$ $0.999 + 0.001$
double	Expanded class of reals. Some compilers make no difference between float and long. Float expressions always result in a float.	3.14 -55.0 777.0003 4	+ (addition and unary) - (subtraction and unary) * (multiplication) / (division)	$3.14 * 44.0$ $-23.8 / 1.2$ $0.999 + 0.001$

In C++, there are no defined sizes for each of the integer (or integral) types, just that short is smaller than int and int is smaller than long. The exact implementation is left up to the compiler maker. Floating point types are defined in the same way, where float is smaller than double. In addition, long may be placed before double to increase the size of the double, creating a *long double*, which must be larger than double.

For these operators, the usual order of operators and operations apply. Operators of equivalent rank are evaluated left to right.

Highest Rank	*	/	%
Lowest Rank	+	-	

In addition to the operators listed, parenthesis can be included in expressions to change the order of operations. For example,

$3 * 2 + 4$ evaluates to 10 while $3 * (2 + 4)$ evaluates to 18

Division operators are often a little confusing at first. The following example evaluates to 3 because integers always evaluate to integers, i.e. no fractions.

$7 / 2$ evaluates to 3

The division of two floating point numbers always resolves to a floating point number.

$7.0 / 2.0$ evaluates to 3.5

The remainder of two integers can also be confusing at first. For example,

$7 \% 5$ evaluates to 2

Seven divided by five results in one, but the % operator (often called the modulus operator) is only interested in the remainder of integer “long division”.

$$\begin{array}{r} 1 \\ 5 \overline{) 7} \\ \underline{5} \\ 2 \end{array} \leftarrow \text{remainder produces this}$$

7. Strings

Although not a type, *strings* (groups of characters) are supported. *String constants* (more correctly called *string literal constants*) are enclosed in double quotes. For example:

“Sam”, “A1”, “1984!”, “This is a sentence made up of one string.”

8. The Function Main Revisited

Now that functions and types have been introduced, it is time to revisit the function `main`.

```
void main()
{
    ... statements go here ...
}
```

The return type of a *main* function is usually *void*, but older compilers may require it to be *int*. If the return type is *void*, no return expression is needed, but if the return type is *int*, the *return* expression must be 0. The argument list (the contents of the parenthesis after the function header) is usually empty. There can be only one *main* function in a program.

The function *main* shown above does nothing except enclose the statements to be performed first. Statements do the real work in a C++ program and the only statement so far introduced is the *return* statement, which does no real work in a function *main* on programs running on most modern operating systems.

The usual next step in an introductory programming book is to present fundamental output statements of the programming language in question. The problem is C++ doesn't have any! In fact, standard input and output is provided via various libraries of standard classes and objects. To bring them into a C++ program, the preprocessor must be used.

9. The Preprocessor

Like its predecessor the C programming language, C++ allows code from other files to be included in a program. In fact, this is essential for programming in C++. This is work done by a part of the C++ environment called the *preprocessor*. Before the code is passed to the compiler for translation to binary instructions, the preprocessor locates the preprocessor instructions and executes them. Preprocessor instructions are easily recognizable in a program because they begin with # (pound or number sign).

The most common preprocessor instruction is the *#include* instruction. Its forms are these: (*Note that there are no semicolons in preprocessor instructions as they are not C++ statements.*)

```
#include <header-file-name>
#include "path-to-a-file"
```

where *header-file-name* is the name of a header file (file ending in .h)
found in the standard search path of the C++ environment
path-to-a-file is the a relative or absolute operating system path to a
specific file.

Note: Older compilers may require the *.h* to appear in the include instruction. For example: `#include <iostream.h>`

Almost all C++ programs must have one or more files of code included. The files containing the I/O classes and objects of your choice must be included in every program you write. For that reason, almost every program that you write will have the following preprocessor include instruction.

```
#include <iostream>
```

Note: This may be `#include <iostream.h>` on older compilers.

This will amend the minimal C++ program to this:

```
#include <iostream>

void main( )
{
    ... statements go here ...
}
```

10. Namespace

C was a good language for writing programs of up to 30,000 lines of programmer code, but after that, just keeping track of names of objects became a critical issue (among other things). Programmers working on separate parts of a large program could easily use duplicate names, and, indeed would find it hard not to reuse names. To help manage object names, *namespaces* were created in C++. In C++, the *namespace* that a file in a C++ program is to use must be stated at the top of the file, generally below any *include* instructions to the preprocessor. This usually is created in the form

```
using namespace name;
```

where *name* is the name of the desired namespace

All of the C++ library of code is located in the namespace *std*. Students almost always write programs in the *std* namespace. All example programs in this text will be written in the *std* namespace. This will amend the minimal C++ program to this:

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

void main( )
{
    ... statements go here ...
}
```

11. Ostream Object cout and a Program Example

By including *iostream* into a program, we have access to the standard class *ostream* and an object of class *ostream* called *cout*, which is used in its own statements to output the results of expressions. It has the ability to output expressions of all of the fundamental types.

Cout statements are written in the following forms:

```
cout << expression;
cout << expression << expression;
cout << expression << expression << ... << expression;
```

where << is the output stream operator
expression is any expression of a type or a string

The following is an example C++ program. Enter, compile, link, and run it.

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

void main() {
    cout << "15 times 3 equals " << 15 * 3;
}
```

When run, this program outputs:

15 times 3 equals 45

Note: Some C++ environments do not provide a pause at the end of the program. This can cause problems when running in a graphical user interface such as Microsoft Windows. In such a case, the window will simply close, preventing the user from seeing the results of the program. Here is an example of how to make the above program pause. Additions to the program are in bold face type.

Example of pausing a program:

```
#include <iostream>
#include <conio.h>
using namespace std;

void main() {
    cout << "15 times 3 equals " << 15 * 3;
    cout << "\n... Press Enter to Continue ...";
    cout.flush();
    getch();
}
```

When run, this program outputs:

*15 times 3 equals 45
... Press Any Key to Continue ...*

Note: Not all compilers support conio.h.

Exercises

1. What is a minimal C++ program?
2. What are the 3 parts of a function header?
3. What is a compound statement and how is it represented?
4. What is the purpose of a return statement?
5. What is the purpose or function of a class?
6. What is state information?
7. What are member functions?
8. What is a data type?
9. What values may be represented by an int or long data type?
10. What values may be represented by a float or a double?
11. Evaluate the following expressions:
 $31 / 4$ $31.0 / 4.0$ $31 \% 4$
12. What is the difference between how a character constant is written and how a string is written?
13. What is the preprocessor?
14. What is the function of the preprocessor *#include* instruction?
15. Why is the preprocessor instruction *#include <iostream>* used in most C++ programs?
16. What is the purpose of having the statement *using namespace std;* in a program.
17. What is the purpose of the *cout* statement?

Programming Assignment 1.1

Write, compile and run a C++ program that outputs your name.

Programming Assignment 1.2

Write, compile and run a C++ program that calculates and outputs your age in days to your nearest birthday.

Programming Assignment 1.3

Write, compile and run a C++ program that calculates and outputs 11 divided by 3, 11.0 divided by 3.0, and the remainder of 11 divided by 3. Before you run the program, predict the output. Are your predictions different from the actual output? If they are, why?