C++: The Basics

Dr. Frazer K. Noble
School of Engineering and Advanced Technology
Massey University
Auckland
New Zealand

March 23, 2017

Introduction

The aim of this document is to introduce C++, without going into too much detail. Therefore, we informally present C++ and the basic mechanisms for organising code into a program.

The Basics

C++ is a compiled language. Its source code is processed by a compiler, which produces object files, which are linked together, generating an executable.

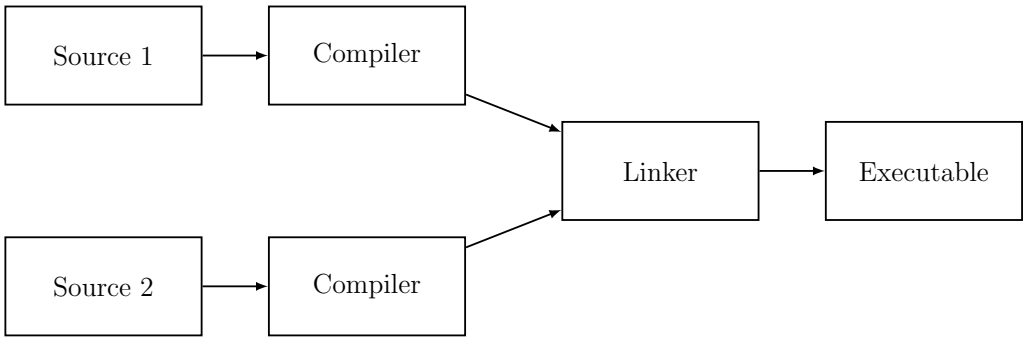

Figure 1: C++ compilation and linking process.

On Windows, an executable has the .exe suffix. Also, an executable generated for Windows is not transferable to Linux or Mac (and vice versa).

With respect to source code, the ISO C++ standard defines to kinds of entities:

1. Core language features. These are built-in types, e.g. **int**, and loops, e.g. **for**.

2. Standard-library features. These are containers, e.g. vector, and input/output (I/O) operations, e.g. cout.

The standard-library is a collection of C++ code, which is provided by every C++ implementation.

C++ is a statically typed language; that is, the type of every entity must be known to the compiler.

Hello World!

The minimal C++ program is

```
int main() {} //The minimal C++ program.
```

This defines a function called main, which has no parameters and does nothing.

The curly braces, {} express grouping. Here, they indicate the start and end of the main() function. The double slash, // begins a comment that extends to the end of the line. A comment is for the user; the compiler ignores it.

Every C++ program must have exactly one main() function. The program starts by executing that function. The **int** value returned by main(), if any, is the program's return value to the "system". Typically, a non-zero value from main() indicates failure.

Here is a program that produces some output:

```
#inlcude <iostream>

int main(int argc, char* argv[]) {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

The line **#include**<iostream> instructs the compiler to *include* the declarations of the standard stream I/O facilities as found in iostream. Without these, the expression

```
std::cout << "Hello World!" << std::endl;
```

would make no sense. The operator << ("put to") writes its second argument onto its first. In this case, the string literal "Hello World! is onto the standard output stream std::cout.

The std:: specifies that the name cout is to be found in the standard-library namespace.

In general, all executable code is placed in functions and called directly, or indirectly, from main().

Types, Variables, and Arithmetic

Every name and expression has a type. For example, the declaration

```
int inch;
```

specifies that inch is of type **int** (an integer variable).

A *declaration* is a statement that introduces a name into the program. It specifies a type for the named entity:

- A *type* defines a set of possible values and set of operations.
- An *object* is some memory that holds a value of some type.
- A *value* is a set of bits interpreted according to a type.
- A *variable* is a named object

C++ offers a variety of fundamental types. For example,

```
bool //Boolean, possible values are true and false.
char //Character, e.g. 'a', 'b', 'c'.
int //Integer value, e.g. 1, 2, 3.
double //Double-precision, floating-point value, e.g. 3.14, 2.3, 1.4
```

Arithmetic operators can be used for appropriate combinations of these types:

```
x+y //plus.
x-y //minus.
x*y //multiplication.
x/y //division.
x%y //remainder (modulus) for integers.
```

So can the comparison operators:

```
x==y //equal.
x!=y //not equal.
x>y //greater than.
x<y //less than.
x>=y //greater than or equal.
x<=y //less than or equal.
```

Note, = is the assignment operator; == is the equality operator.

C++ offers a variety of notations for expressing initialisations, such as =, and a universal form based on curly-brace-delimited initialiser lists:

```
double d = 1.3; //Initialise d with 1.3.
double e {2.6}; //Initialise e with 2.6.

vector<int> v {1,2,3,4,5}; //Initialise a vector of int's.
```

The = form is traditional and dates back to C, but if in doubt, use the general {}-form.

When initialising a variable, you don't need to explicitly state its type when it can deduced from the initialiser:

```
auto i {1}; \\Integer value
auto d {2.0}; \\Double value
auto v {1,2,3,4,5}; \\Vector of int's
```

auto is used when we don't have a reason to mention type and is helpful in avoiding redundancy.

Tests and Loops

C++ provides a conventional set of statements for expressing selection and looping. For example, here is a function that checks if a number is equal to the value 2:

```
bool check() {

  std::cout << "Type a number: ";

  char num {};
  std::cin >> num;

  if ( num == '2' ) {
    return true;
  } else {
    return false;
  }
}
```

To match the << ("put to") operator, the >> ("get from") operator is used for input; std::cin is the standard input stream.

Conclusion

In this document, we have introduced C++; describing the basics of compilation, providing an example program, describing C++'s types, variables, and arithmetic, and simple tests and loops.