Learning C++

**Ch3 Style**

**3.1 Comments**

Use comments on every program to better understand the objective of it and also to put your toughs clear in the making of it.  
 At the beginning of the program is a comment box that contains information about the program.Boxing the comments makes them stand out. There are a couple of sections in the comment box:

* Heading - name & short description
* Author
* Purpose
* Usage - short documentation of the program and how to run it
* References
* File formats - list the files that the program reads or writes and their format
* Restrictions - list any restriction that apply to the program
* Revision history
* Error Handling - if the program encounters an error, what does it do with it ?
* Copyright & License
* Notes

**3.2 C++ code**

The actual code is of the program is structured in 2 parts: variables & executable instructions. Every variable in C++ must be declared.

**3.3 Naming Style**

lower\_case - for variables

UPPER\_CASE - for constants

**3.4 Clarity**

A program should read like a technical paper, organized into sections and paragraphs. Procedures form a natural section boundary. You should organize your code into paragraphs, beginning a paragraph with a topic sentence comment and separating it from other paragraphs with a blank line.

**3.5 Simplicity**

General rules for keep the program readable and simple:

* a single function can not be longer than 1 or 2 pages
* avoid complex logic such as nested **if’s**
* avoid long statements
* split large code files into smaller ones (max 1500)
* when using classes put one per module

**4.0 Basic declarations & expressions**

**4.1 Basic program structure**

Our single function is named main. The name main is special, because it is the first function called. Any other functions are called directly or indirectly from main. The function main begins

with:

**int main( )** {

and ends with:

**return(0);** }

The line return(0); is used to tell the operating system that the program exited normally (status=0). A nonzero status indicates an error. The bigger the return value, the more severe the error. Typically 1 is used for most simple errors, such as a missing file or bad command-line syntax.

**#include <iostream>** - This tells C++ that we want to use the standard input/output system. (The proper name for this is the I/O streams module.)

The main routine contains the instruction: **std::cout << "Hello World\n";**

We are using the standard object **std::cout** (console out) to output the message. A standard object is a generally useful C++ object that has already been defined and put in the standard library. A library is a collection of class definitions, functions, and data that have been grouped together for reuse. The standard library contains classes and functions for input, output, sorting, advanced math, and file manipulation. See your C++ reference manual for a complete list of library functions and standard objects.

**4.2 Simple expressions**

**+, - , \* , / ,%-** modulus

**4.3 The std::cout output object**

The standard object **std::cout** is used to output data to the console. We'll learn what a object is later in Chapter 13, but for now all we have to know is that the operator **<<** tells C++ what to output.

**4.4 Variables & storage**

System variable names (names defined the by the standard library) can begin with underscore.

**4.6 Integers & floats**

int name; //comment

float number;

**4.7 Assignment statements**

variable = expression;

**4.10 Characters**

The type char represents single characters. The form of a character declaration is:

**char** variable; //comment

Characters are enclosed in single quotation marks **('). 'A', 'a'** and **'!'** are character constants. The backslash character (\) is called the escape character . It is used to signal that a special **character follows.** For example, the character **\t** can be used to represent the single character "tab." **\n** is the newline character. It causes the output device to go to the beginning of the next line, similar to a return key on a typewriter. The character **\\** is the backslash itself. Finally, characters can be specified by **\nnn,** where **nnn** is the octal code for the character.

|  |  |  |
| --- | --- | --- |
| **Character** | **Name** | **Meaning** |
| **\b** | Backspace | Move the cursor to the left one ch |
| **\f** | Form feed | Go to top of a new page |
| **\n** | New line | Go to the next line |
| **\r** | Return | Go to the begging of the current line |
| **\t** | Tab |  |
| **\’** | Apostrophe or single quotation mark | the ch ‘ |
| **\”** | “ | the ch ‘’ |
| **\\** | \ | the ch \ |
| **\nnn** | the ch nnn | the ch number nnn (octal) |
| **\xNN** | the ch NN | the ch numb NN (hexadeciamal) |

**ch are enclosed in single quotes and strings in duble quotes**

**4.11 Wide Characters**

**wchar\_t**

It is used to specify "wide characters" which include not only the basic American

characters, but foreign characters as well. A wide character is declared just like a simple character:

char simple; // A simple character

wchar\_t wide; // A wide character

A simple character is declared by putting the character inside single quotes: 'X'. A wide character uses an "L" prefix to indicate that it is a wide character: L' ' .

For example:

simple = 'X';

wide = L'ch';

**4.12 Boolean type -** bool

**5. Arrays, Qualifiers, and Reading Numbers**

**5.1 Arrays**

Arrays allow us to do something similar with variables. An array is a set of consecutive memory locations used to store data. Each item in the array is called an element. The number of elements in an array is called the dimension of the array. A typical array declaration is:

// List of data to be sorted and averaged

**int data\_list[3];**

**5.2 Strings**

Before you can use this type, you must bring in the **std::string** definition with the statement:

**#include <string>**

After this you can declare strings like any other variable:

std::string my\_name;

A string constant is any text enclosed in double quotes. So to assign the variable my\_name the value "Steve", we use the statement:

**my\_name = "Steve";**

Strings may be concatenated using the + operator. A string can be treated like an array of characters. Each character in the string can be accessed through the [ ] operation.

So to get the first character of our string safely, we need to use the statement:

**first\_ch = first\_name.at(0)**

**String methods:**

* get the length - fullname.length
* extract a portion of a str - fullname.substr(first, last)

**5.3 Reading data**

The general form of a std::cin statement is:

**std::cin >> variable;**

This works for all types of simple variables such as int, float, char, and wchar\_t.

Reading strings is a little more difficult. To read a string, use the statement:

**std::getline(std::cin, string);**

**Using assert to handle errors**

#include <iostream>

#include <assert.h>

const int N\_PRIMES = 7; // Number of primes

// The first few prime numbers

int primes[N\_PRIMES] = {2, 3, 5, 7, 11, 13, 17};

int main( )

{

int index = 10;

assert(index < N\_PRIMES);

assert(index >= 0);

std::cout << "The tenth prime is " << primes[inde

return (0);

}

The statement: **#include <assert.h>** tells C++ that we want to use the assert module. Now we

know that the index must be in range. After all, our program could never contain an error that might generate a bad index. But just to make sure, we check to see if it's in range using the statement:

**assert(index < N\_PRIMES);**

**assert(index >= 0);**

Now when the program hits the assert statement, the assertion fails and an error message is issued:

**bound\_c1: bound\_c1.cpp:11: int main( ): Assertion**

**Abort (core dumped)**

The program then aborts. Aborting is a nasty way of handling an error, but it's better than doing nothing at all. (This is not true in every case)

**Size of array** (using sizeof function)

**sizeOfArray = (sizeof(primes)/ sizeof(primes(0))**

**5.5 Multidimensional arrays**

ex: type variable [size1][size2];

Declaring a multidimensional array:

// a typical matrix

**int matrix[2][4] ={**

**{1, 2, 3, 4},**

**{10, 20, 30, 40}**

**};**

This is shorthand for:

matrix[0][0] = 1;

matrix[0][1] = 2;

matrix[0][2] = 3;

matrix[0][3] = 4;

matrix[1][0] = 10;

matrix[1][1] = 20;

matrix[1][2] = 30;

matrix[1][3] = 40;

**5.6 C-Style strings**

To initialize the variable name to "Sam" you would write:

#include <cstring>

char name[4];

int main( ){

std::strcpy(name, "Sam"); // Legal

return (0);

}

C++ uses variable-length strings. For example, the declaration:

#include <cstring>

char a\_string[50];

int main( ){

std::strcpy(a\_string, "Sam");

creates an array (a\_string) that can contain up to 50 characters. The size of the array is 50, but the length of the string is 3. Any string up to 49 characters long can be stored in a\_string. (One character is reserved for the NUL that indicates the end of the string.)

**Routines that work on string variables:**

|  |  |
| --- | --- |
| **Function** | **Description** |
| strcpy(string1, string2) | Copies string2 into string1 |
| strncpy(string1, string2, length) | Copies string2 into string1, but doesn't copy over length characters (including the  end of string character) |
| strcat(string1,string2) | Concatenates string2 onto the end of string1 |
| strncat(string1, string2) | Concatenates string2 onto the end of string1, but only length characters (will  not put an end of string character on the result if length characters are copied) |
| length = std::strlen(string) | Gets the length of the string |
| std::strcmp(string1,string2) | Returns 0 if string1 equals string2;  A negative number if string1 < string2  A positive number if string1 > string2 |

C++ has a special shorthand for initializing strings, using double quotes (") to simplify the initialization. The previous example could have been written:

char name[] = "Sam";

The dimension of name is 4, because C++ allocates a place for the '\0' character that ends the string

**5.7 Types of integers**

**long int number;** - allocates extra storage for the int

**short int number;** - 2bytes

**int number;** - 2-4 bytes

Long integer constants end with the character "L". For example:

**long int var = 1234L;** // Set up a long variable

**They are also signed & unsigned -** By default all are signed. We must specify if we need one to be unsigned.

**Char -**  very short int - only 1 byte long (from -128 to 127) or (0-255). Must be specified if it is signed or unsigned.

Reading and writing very short integers is a little tricky. If you try to use a char variable in an output statement, it will be written as a character. You need to trick C++ into believing that the char variable is an integer. This can be accomplished with the static\_cast operator.

#include <iostream>

signed char ch; // Very short integer

// Range is -128 to 127

int main( ){

std::cout << "The number is " << static\_cast<int>(ch);

return (0);

}

**5.8 Types of floats**

The **float** type also comes in various flavors. float denotes normal precision **(usually 4 bytes**). **double** indicates double precision **(usually 8 bytes)**, giving the programmer twice the range and precision of single-precision (float) variables.

The quantifier **long double** denotes extended precision. On some systems this is the same as **double**; on others, it offers additional precision. All types of floating-point numbers are always **signed**.

On most machines, single-precision floating-point instructions **execute faster** (but less accurately) **than double precision**. Double precision gains accuracy at the expense of time and storage. In most cases float is adequate; however, if accuracy is a problem, switch to double.

**5.9 Constants & Reference declarations**

The keyword const indicates a variable that never changes. To declare a value for pi, we use the statement:

**const float PI** = 3.1415926; // The classic circle

Another special variable type is the **reference type**. The following is a typical reference declaration:

**int count;** // Number of items so far

**int& actual\_count = count;** // Another name for count

The special character & is used to tell C++ that actual\_count is a reference. The declaration causes the names count and actual\_count to refer to the same variable. For example, the following two statements are equivalent:

count = 5; // "Actual\_count" changes too

actual\_count = 5; // "Count" changes too

In other words, a simple variable declaration declares a box to put data in.

**5.10 Qualifiers -**Qualifiers may be thought of as adjectives that describe the type that follows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Special** | **Constant** | **Storage class** | **Size** | **Sign** | **Type** |
| volatile | const | register | long | signed | int |
| <blank> | <blank> | static | short | unsigned | float |
|  |  | extern | double | <blank> | char |
|  |  | auto | <blank> |  | wchar\_t |
|  |  | <blank> |  |  | <blank> |

**5.10.1 Special**

**Volatile**