



CITY *of* BRISTOL  
COLLEGE

College Green Centre

Faculty of Computing and Digital Communication

Documentation Standards:

**C++ Programming House-style**

Document No: DS03/OOPHS	Date: 1 September 2003
Issue: 02	Author(s): Ian Shere

# Contents

1	Purpose .....	1
2	The Coding Practice .....	1
2.1	Lexical Conventions and Code Layout .....	1
2.1.1	Blank Lines .....	1
2.1.2	Block Structure .....	1
2.1.3	Capitalisation .....	2
2.1.4	Length Of Identifiers .....	2
2.1.5	Naming .....	2
2.1.6	Comments .....	2
2.1.7	Multiple Statements Per Line .....	3
2.1.8	Literals .....	3
2.1.9	Gotos .....	3
2.1.10	Global Variables .....	3
2.1.11	Operators .....	3
2.1.12	Class Definitions .....	3
2.1.13	Client Code .....	4
2.1.14	Function Declaration .....	4
2.1.15	Header Files .....	5
2.1.16	Source File Layout .....	5
3	Filenames .....	5
4	Program Template .....	6

## 1 Purpose

This working document defines a coding practice for using the C++ programming language to:

- 1.1 Ensure that C+ code is written to a consistent style and structure
- 1.2 Support the production of reliable, readable and maintainable code.

## 2 The Coding Practice

### 2.1 Lexical Conventions and Code Layout

#### 2.1.1 Blank Lines

A blank line should follow the declaration of variables and immediately after each structure. Additional blank lines can be used to assist in the readability of the code and should be used to segregate distinct sections of code.

#### 2.1.2 Block Structure

2.1.2.1 The block structure of the code will be emphasised by a suitable pattern of indentation.

2.1.2.2 The indentation increment will be 3 or 4 spaces.

2.1.2.3 Braces will be indented to the same level as the control statement with which they are associated

Example :

```
if ( .....)  
{  
    .....  
}
```

2.1.2.4 In *if*, *if else*, *switch / case*, *for*, *while* and *do while* structures the body of the structure must be in all cases a compound statement, i.e. enclosed in `{ }`, note: this does not include the individual case statements.

### 2.1.3 Capitalisation

The conventions for using upper and lower case letters are :

- 2.1.3.1 Keywords (reserved words) in lower case.
- 2.1.3.2 Macro names (introduced by **#define** ) in upper case.
- 2.1.3.3 Majority of lower case for identifiers.

### 2.1.4 Length Of Identifiers

The maximum length of an identifier should not exceed 31 characters.

### 2.1.5 Naming

- 2.1.5.1 Reserved words will not be used as identifiers.
- 2.1.5.2 The names chosen for identifiers should be meaningful but not too verbose.
- 2.1.5.3 All variables should start with a capital letter, if the variable name is made up of more than one word, each word can start with a capital letter,

Example:                    `int UnitPrice;`  
Or     `int Unit_Price;`  
Or     `int Unit_price;`

- 2.1.5.4 When coding mathematical formulae, the identifiers should reflect the mathematical notation embodied in any supporting documents or implied by standard practice.
- 2.1.5.5 Identifiers should not contain two consecutive underscore characters.
- 2.1.5.6 Single letter identifiers are not allowed unless coding mathematical formulae or loop control variables but only lower case i, j or k.

### 2.1.6 Comments

- 2.1.6.1 There must be a comment section at the top of the program containing the programmers name, the date, and a summary description of the purpose of the program.
- 2.1.6.2 The code should be well commented to amplify the function of the code and to provide contextual information.
- 2.1.6.3 All variable declarations should be accompanied by a descriptive comment.
- 2.1.6.4 Function definitions should be preceded by the descriptive comment

section shown in the template.

2.1.6.5 The program body should be annotated by comments where this makes the program clearer.

2.1.6.6 Comments should be clear and to the point. Trivial and unnecessary comments are to be avoided.

2.1.6.7 Comments should not duplicate C++ syntax or semantics. In the following example the comment is redundant :

```
j++;    // Increment j
```

2.1.6.8 Comments should be written in good English with normal punctuation, using both upper and lower case characters.

2.1.6.9 C++ style comments introduced by double slash (//) are to be used for rest of line or single line comments..

2.1.6.10 There shall be a least one space between the comment delimiter and the text of the comment.

### **2.1.7 Multiple Statements Per Line**

Only ONE statement shall appear on each source line. When a line contains more than one statement, readers may miss the second statement.

### **2.1.8 Literals**

If the same literal appears several times in a code segment, this literal should be implemented as a named constant and declared using capitals.

### **2.1.9 Gotos**

The use of GOTOS are not allowed in structured programming and MUST NOT be used.

### **2.1.10 Global Variables**

Global variables should be avoided if possible.

### **2.1.11 Operators**

At least one space is required before and after each operator.

### **2.1.12 Class Definitions**

The class definition should be structured with private access being the first section, followed by protected and the public section being last. If code is generated using the OMT CASE Tool, the skeleton code will require editing as the private and public section are in the opposite order to that required by this

standard. All data members are to be declared in the private or protected access section of the class definition. Individual data members are to be commented as for variables. Member function prototypes in the public interface definition and should be commented as for functions.

#### **2.1.12.1 Member function names**

Methods that fall into the category of ‘getter’ or ‘accessor’, ‘setter’ or ‘mutator’ and ‘initialisation’ functionality should precede the attribute name (except in the case of the initialisation, when it will precede the class name) with the word get, set or init. Business methods will identify their action by a prefix word starting with a lower case character.

Examples:

```
int getAge(void); //getter method for attribute age
void setAge(int Years); //setter method for attribute age
void initPerson(char *Name, char *Teleno, int Age);
//initialisation method of Person class
void addMoney(Money Value1, Money Value2);
//a business function adding 2 money objects
```

#### **2.1.13 Client Code**

The ‘main’ function of the client code should indicate the flow and purpose of the program. It must not simply call one or two other functions. ‘Main’ is the control function of a ‘C’ or ‘C++’ program and as such must demonstrate this control.

#### **2.1.14 Function Declaration**

When declaring a function prototype in the client code, the function name should start with a capital letter. If the name consists of two words, each can start with a capital. The function prototype should also declare variables used in its parameters. If no arguments are required void should be used as the signature.

Example :

```
void StudentDetails ( char Name, int Age );
OR void Student_Details ( char Name, int Age );
OR void StudentDetails (void);
```

#### 2.1.14.1 Function Definition order

In the client code the function definition blocks should appear in the same order as the function declarations under the function prototype section. Ideally, the order of the functions should indicate the logical flow through the program.

#### 2.1.15 Header Files

Separate header files should be used for class definitions when creating Project files.

#### 2.1.16 Source File Layout

The template on the following page shall be used :

### 3 Filenames

All file and path (folder) names must conform to DOS conventions. A file name consists of three parts. The first part (called the core name) can be up to 8 characters long, this is separated by a period (full stop) which is the second part. The third part of a file name is known as the extension, this is up to 3 characters long.

The first part of the file name can consist of any alphabet character, A to Z (upper and lower case are treated as being the same), numeric digits 0, 1 to 9, and the following special characters: @ ('at' symbol); # or £ (cross hash or pound sign); \$ (dollar sign); % (percent symbol); ^ (caret); \_ (underline); - (hyphen); and & (ampersand).

The third part of a file name, the file extension can be user defined and should follow the same rules as that for the core name. However, the following extensions must be used when writing your C++ applications.

C++ source files use the suffix .cpp

C++ header files use the suffix .h

#### 4 Program Template

```
/*===== Program Information =====
NAME OF SOURCE FILE:

PROGRAM DESCRIPTION:

AUTHORS NAME:

DATE: */

//LIBRARIES
#include <iostream.h>

//CLASS DECLARATIONS
class SampleClassName{
private:
    //data members
public:
    //member functions
};
//CLASS MEMBER FUNCTION DEFINITIONS

//DEFINITIONS

//GLOBAL VARIABLES

//FUNCTION PROTOTYPES

//===== MAIN BLOCK =====
void main(void)
{
}

/* ***** FUNCTION *****
* Function Name:
*
* Operation:
*
* Parameters Passed:
*
* Return Values:
***** */

// ===== Program End =====
```