Webster University C++ Coding Standards

"Style Guidelines"

Introductory Version of C++ Coding Standards

This document introduces the Department Coding Standards for
beginning level C++ Programming assignments.

Introduction

Style Guidelines are rules and conventions that typically cover indentation, comments, declarations, statements, white space, naming conventions, programming practices, programming principles, programming rules of thumb, architectural best practices, etc. In almost every organization, standards are a fact of life.

Software programmers follow these guidelines to help improve the readability of their source code and make software maintenance easier. Coding conventions are for the human readers and maintainers of software. Coding conventions are not enforced by compilers.

These **Coding Standards** will be required and enforced for all programs written for any programming course taken at Webster University.

These may not be the same exact standards required by any particular company, but these standards will prepare you for programming with any set of standards.

In order to receive the maximum grade possible on programming assignments, the student must adhere to the coding standards in this document.

= = = = = = = = = = = = = = = = = = =

The main () Function

The header for main() should be:

int main ( ) (note - the parentheses are empty)

The final statement in the main() function should always be: return 0;

Naming Conventions

All names should be meaningful and descriptive and clearly indicate the purpose of the variable, constant or function. Avoid cryptic or confusing names.

Variable names and function names should be very descriptive of their purpose. This enables the program code to more easily understood by anyone reading the code.

Variables should not be reused for different purposes within the same function.

Data Variables	All data variable names should begin with a lower case letter For example: " sumOfGrades "
Loop Counter Variables	Use i, j, k or x, y or c for loop counters without meaning. Use r, c (for row and column) when using 2-dimensional arrays.
Constants	Constant names should be uppercase. For example: " TAXRATE " or " TAX_RATE "
Functions	Function names should begin with a lower case letter and also be a verb phrase (a verb followed by a noun) For example: " calculateTotal "

No Global Variables

Global variables should never be used, unless specified for situations like multi-dimensional arrays.

The preprocessor directive "#define" should never be used as a method of providing constant values.

Indentation

General Rule (ANSI C++ Standard)

Use four (4) spaces for indentation. (This is the most commonly used indentation size.) Every block of code and every definition should follow a consistent indention style.

Note: The use of tab is controversial. The tab character can cause portability problems unless the text editor changes the tab to 4.

Indentation and Curly Braces for Control Structures (ANSI C++)

The opening and closing curly braces of a control structure (e.g. a *for* statement or an *if-else* statement) should always be on an otherwise blank line. The curly braces should appear in the same column as the start column of their corresponding control structure. Related (open and close pair) curly braces should always be aligned in the same column.

** NOTE: there is a single space after 'if' or 'while' and before the open parentheses **

Department Standard:

```
if (  count == 10 )
{
    displayTotal ( );
}
```

Nonstandard:

```
if (  count == 10 )
    {
    displayTotal ( );
     }
```

Department Standard:

```
do
{
    getNextGrade ( );
}
while ( grade ! =  -99 );
```

Nonstandard:

```
do
{
    getNextGrade ( );
}while ( grade ! =  -99 );
```

Department Standard:

```
for ( x = 0; x < 10 ; x++ )
{
   getStudentId (  );
   getStudentName (  );
}
```

Nonstandard:

```
for ( x = 0; x < 10 ; x++ ){
    getStudentId (  );
    getStudentName (  );}
```

Optional Omission of Curly Braces

A control statement with only a single statement
in its scope may omit the curly braces.
(However, this can cause a tricky logic error if a second statement is added later.)

Optional :

```
if (  count == 10 )
   displayTotal ( );
else
   calcTotal ( );
```

Indentation of " if " and " else "

The if statement is coded on a line by itself. Likewise, the else statement is ordinarily coded on a line by itself. However, if the else is followed by an if which is the only if statement subordinate to that else, the if and its conditional expression may go on the same line as the else and no additional nesting or indentation is required.

** NOTE: there is a single space after 'if' and before the open parentheses **

Department Standard:

```
if ( count == 10 )
{
     cout << "\n";
}
else if ( count > 10  )
{
     cout << "\t";
     count = 1;
}
```

Nonstandard:

```
if ( count == 10 )
{
     cout << "\n";
}
else
{
      if ( count > 10 )
      {
          cout << "\t";
          count = 1;
       }
}
```

If a conditional expression is made up of multiple subexpressions, each subexpression should be surrounded in its own set of parentheses.

.

Department Standard:

```
if ( (count == 10 ) && (total < 99) )
{
     cout << "\n";
}
```

Nonstandard:

```
if ( count == 10  &&  total < 99 )
{
     cout << "\n";
}
```

(continued on next page)

If a conditional expression is made up of multiple lengthy subexpressions, such that the entire expression is longer than the typical line length, then begin a new indented line as shown below. The line should break after the && or || following a subexpression.
Each subexpression should be enclosed in parentheses.
When a statement easily fits on one line, there is no need to break the line.

Department Standard:

```
if ( (countBeforeAdjustment  > 0 ) &&  (countAfterAdjustment  < 100 )  &&
    (totalBeforeAdjustment  < 99)  &&   (totalAfterAdjustment  < 99) )
{
      cout << "\n";
}
```

Nonstandard:

```
if (  (countBeforeAdjustment  > 0 ) &&  (countAfterAdjustment  < 100 )  &&  (totalBeforeAdjustment
< 99)   && (totalAfterAdjustment  < 99)  )
{
      cout << "\n";
}
```

* Curly braces should always be on an otherwise blank line.

Line length and Indentation of Continued Lines

Maximum line length of any line of source code shall be no longer than the row of commented stars, including all blank characters. Cleanly split longer lines to improve readability. The continued line should be indented. For example, the following statement breaks the line before a stream operator and then all initial stream operators in each line begin in the same position.
When a statement easily fits on one line, there is no need to break the line.

Department Standard:

```
cout  <<  setw ( 15 )  <<  employeeFirstName  <<  setw ( 15 )  <<  employeeLastName
      <<  setw ( 10 )  <<  employeeID  <<  setw ( 11 )  <<  employeePhone;
```

Nonstandard:

```
cout     <<  setw ( 15 )  <<  employeeFirstName  <<  setw ( 15 )  <<  employeeLastName  <<
setw ( 10 )  <<  employeeID  <<  setw ( 11 )  <<  employeePhone;
```

Indentation of Inline Comments (C++ style)

Inline comments should begin in a column toward the far right and be vertically aligned with one another. Comments should be meaningful and enhance the readability of the program.

Department Standard:

```
int       studentId;                                    //  student SSN

area = (base1 + base2) / 2 * ht;                        //  area of a trapezoid
```

Variable Declaration Style

Each variable must be declared or defined on a separate line. Group variables of the same data type together and indent to the same level. Declare variables at the top of a function.
Do not use complex initializations, such as functions calls within a variable definition statement.

Department Standard:

```
int   numStudents = 0,
      testNumber;
double  studentAverage,
        classAverage;
```

Nonstandard:

```
int   numStudents = 0, testNumber;
double    studentAverage,  classAverage;
```

Nonstandard:

```
int   numStudents;
int   testNumber;
double  studentAverage;
double  classAverage;
```

Separator between Functions

To make reading your code easier and to facilitate locating different functions, each function (and the prototype section) should be separated by a row of 101 commented stars, with a blank line before and after each separator. (You can copy the last line of the header block to use as this separator.)

Department Standard:

```
//*****************************************************************************************************
```

"White Space" between Sections / Blocks of Code

White space is used to enhance readability.
Insert a blank line between the declarations section and other statements in each function.
Insert a blank line between blocks of related statements to make clear the structure of the program.
There should be a blank line before and after any loop, any if or if-else, etc.

Department Standard:

```
int   count;
double   studentAverage;

if (  count == 10 )
{
   displayTotal ( );
}
```

Nonstandard:

```
int   count;
double   studentAverage;
if (  count == 10 )
{
   displayTotal ( );
}
```

"White Space" Before and After Operators, etc.

White space is used to enhance readability.
Use a space before and after every operator or assignment symbol (including the stream operators).
Also use a space between keywords in C++ statements.

Department Standard:

```
x = y +  2 / z - a;
cin   >>  minimum;
cout  <<  setw(7)  <<  minimum;
```

Nonstandard:

```
x=y+2/z-a;
cin>>minimum;
cout<<setw(7)<<minimum;
```

Department Standard:

```
if (x == 5)

while (x > 0)
```

Nonstandard:

```
if(x == 5)

while(x>0)
```

Return ( ) Statements and the "Single Entry-Single Exit" Rule

Every function should have exactly one entry point and one exit point.
It is never permissible to use more than one return statement in a function.

Department Standard:

```
bool   result;

if (  count == 10 )
   result = true ;
else
   result = false ;

return result;
```

Department Standard:

```
bool   result = false;

if (  count == 10 )
   result = true ;

return result;
```

Nonstandard:

```
if (  count == 10 )
   return  true ;
else
   return  false ;
```

Guidelines for Well – Structured Functions

a) Use local variables within functions when the variable is used only within that function.

b) Use arguments / parameters to pass information to functions. NO global variables.

c) Use prototypes for all functions. Prototypes must be given before main(). Function definitions must be after main()

d) To protect arguments that should not be modified by a function, declare the parameters to be value parameters or constant reference parameters rather than reference parameters.

e) Variables should not be reused for different purposes within the same function.

f) Insert a blank line between declarations and statements and between blocks of related statements (such as a loop or if-else block) to make clear the structure of the program.

g) Declare all constants and variables at the top of a function. Do not use complex initializations, such as functions calls within a variable definition statement.

h) Do not use a function call as an argument or value that is sent into a function. (Call the function outside of the argument list.)

i) No **break** statements permitted, except for within the switch statement. No **continue** statements permitted. No **goto** statements permitted.

j) Do not waste the return value from your functions. The return value should be captured by the calling function.

k) Separate functions by a row of 101 commented stars(//******). (Blank line before and after)

Miscellaneous Guidelines

Label all output produced by a program. Do not just display a value without identifying that value.

Use C++ input and output statements (cin / cout) - not C statements (scanf / printf).

Use C++ style comments (//…) - not C style comments (/* …*/)
Exception: It is acceptable to use C-style comments around your program's output, when it has been pasted to the bottom of the .cpp file. The /* and the */ should each be placed on otherwise blank lines.

Comment Block for Every File

The very top of every file must contain a comment block similar to the following, which will be provided by your instructor on World Classroom. The comment block provided will have 101 commented stars.
Use the heading exactly as provided by your instructor.

```
//*************************************************************************************************
//
//              File:                   studentList.cpp
//
//              Student:                Sam Student
//
//              Assignment:             Program  #1
//
//              Course Name:            Programming I
//
//              Course Number:          COSC 1550 - 01
//
//              Due:                    August 21, 2019
//
//
//              This program asks the user to read numbers from a disk file and then displays an ordered list .
//
//*************************************************************************************************
```