Webster University C++ Coding Standards

Introduction

Standards for writing software are often regarded with distaste. However, in almost every organization, standards are a fact of life. 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 these standards.

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.

Data Variables	All data variable names should begin with a lower case letter For example: " sumOfGrades "
Constants	Constant names should be uppercase. For example: " TAXRATE "
Functions / Member Functions	Function names (including public and private member functions) should begin with a lower case letter and also be a verb phrase (e.g. verb followed by noun) For example: " calculateTotal "
Pointers	Pointer names can optionally begin with a lower case 'p' followed by the variable name. Place the * close to the pointer type not the variable name.
Classes	All classes/ object variable names should begin with an UPPER case letter.

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.

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.

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";
}
```

If a conditional expression is made up of multiple subexpressions, and each subexpression is lengthy, then each subexpression should be coded on a separate line.

Department Standard:

```
if ( (countBeforeAdjustment  == 10 ) &&
    (totalBeforeAdjustment  < 99) ) &&
    (totalAfterAdjustment  < 99) )
{
   cout << "\n";
}
```

Nonstandard:

```
if ( (countBeforeAdjustment  == 10 ) && (totalBeforeAdjustment  < 99) ) ) &&
(totalAfterAdjustment  < 99) )
{
   cout << "\n";
}
```

Line length and Indentation of Continued Lines

Maximum line length of a source code line shall be 80 characters, including all blank characters. Cleanly split longer lines to improve readability. The continued line should be indented.

Department Standard:

```
cout    <<  setw ( 25 )  <<  employeeFirstName
        <<  setw ( 10 )  <<  employeeLastName
        <<  setw ( 35 )  <<  employeeHomeAddress;
```

Nonstandard:

```
cout    <<  setw ( 25 )  <<  employeeFirstName  <<  setw ( 10 )  <<
employeeLastName  <<  setw ( 35 )  <<  employeeHomeAddress
```

Indentation of Inline Comments (C++ style)

Inline comments should begin in a column toward the right and be vertically aligned with one another.

Department Standard:

```
int      studentId,          //  student SSN or number or id
         numCredits;         //  number of credits of enrollment
```

Variable Declaration Style

Each variable must be declared 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.

Department Standard:

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

Nonstandard:

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

Nonstandard:

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

"White Space" between Sections / Blocks of Code

White space is used to enhance readability.
Insert blank lines between the declarations section and statements.
Insert blank lines between blocks of related statements to make clear the structure of the program.

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

White space is used to enhance readability.
Use a space before and after every operator or assignment symbol (including the stream operators).

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;
```

Separator between Functions

To make reading your code easier and to facilitate locating different functions, each function should be separated by a row of commented stars, extending to near the right side margin, with a blank line before and after the separator:

Department Standard:

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

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 ;
```

Comment Block for Every File

The very top of every file must contain the following comment block:

```
//*********************************************************************
//
//              File:                   max.cpp
//
//              Student:                Sam Student
//
//              Assignment:             Program  #1
//
//              Course Name:            Data Structures I          ( or   Programming I I,  etc.)
//
//              Course Number:          COSC 3050 - 01
//
//              Due:                    Sept 19, 2012
//
//
//              This program asks the user to read a file of integers from a disk file and then
//              determines the maximum of the three numbers.
//
//              Other files required:
//                      1.      numbers.txt     –  text file of integers
//                      2.      max.h   –   prototypes for all functions needed by max.cpp
//
//*********************************************************************
```

Programs and Functions should be well structured

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

b. Use parameters to pass information to and from functions. NO global variables.

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

d. All functions in a program must be defined after the main function is defined.

e. 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.

f. Use accessor methods (get/set member functions) instead of public variables in objects. All data members should be private.

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

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

i. Use C++ style comments (//…) not C style comments (/* …*/)

Source Code should be Formatted in a Style that Enhances Readability

a) Insert blank lines between declarations and statements and between blocks of statements to make clear the structure of the program.

b) Declare constants and variables at the top of a function.

c) Label all output produced by a program.

d) Separate functions by a row of commented stars(//******)