

Coding Standards for C++ in CS150

Version 6.1

Why have coding standards?

It is a known fact that 80% of the lifetime cost of a piece of software goes to maintenance. Therefore it makes sense for all programs within an organization to be as consistent as possible. Code conventions also improve the readability of the software.

This document specifies the coding standards for all computer science courses using C++ at Pacific University. It is important for you to adhere to these standards in order to receive full credit on your assignments.

The document is divided into four main sections:

- Naming Conventions
- Formatting
- Comments
- Printing

Naming Conventions

- Constants

A constant is to be mnemonically defined using all capital letters and underscore characters such as MAX_NAME_CHARS. Separate each word using an underscore character. Further, your program is to contain no "magic constants." That is, all magic constants must be declared **const** to make the program easier to modify and easier to read. In the case below, 100 is a magic constant and if used in several places throughout a program, can create problems if 100 is to be modified for any reason. Also, the meaning of 100 is not obvious.

Poor Program Style

```
ins.open ("message.dat");
.....
for (indx = 0; indx < 100; ++indx)
{
.....
}
```

Correct Program Style

```
const unsigned short MAX_GRADE_SCORES = 100;
const char INPUT_FILE[] = "scores.dat";

ins.open (INPUT_FILE);
.....
for (indx = 0; indx < MAX_GRADE_SCORES; ++indx)
{
.....
}
```

Note: Constants like 0 and 1 are usually acceptable unless they represent values such as true and false in which case they should be declared as constants.

- Variable Names

1) A variable name is defined in all lowercase letters unless the variable name contains multiple names such as readStudentRecord. After the first word, each subsequent word has the first letter capitalized with the remainder of the word made up of lowercase letters and numbers.

2) Variable names are to be mnemonic unless the variable is being used in a for loop in which case the names i, j, k, l, m, n are acceptable names to be used. If however the nested loop is being used in conjunction with a two-dimensional array, then the names row and column should be used.

3) Global variables must begin with g so that a name such as gHashTable denotes a global variable.

4) To aid in identifying the type of a variable, we will use the following prefixes.

Type Indicator is a	Text Prefix	Variable Name Example
boolean	b	bFlag
null terminated string	sz	char szFileName[10]
globals	g	char gNumFiles

Note: Declare all variables at the top of the function, no in-code variable declarations EXCEPT for in the initializer in a for loop.

Poor Program Style

```
int L (char n[])
{
    for (int i = 0; n[i] != '\0'; ++i);

    return i;
}
```

Good Program Style

```
int strlen (char name[])
{
    int count;

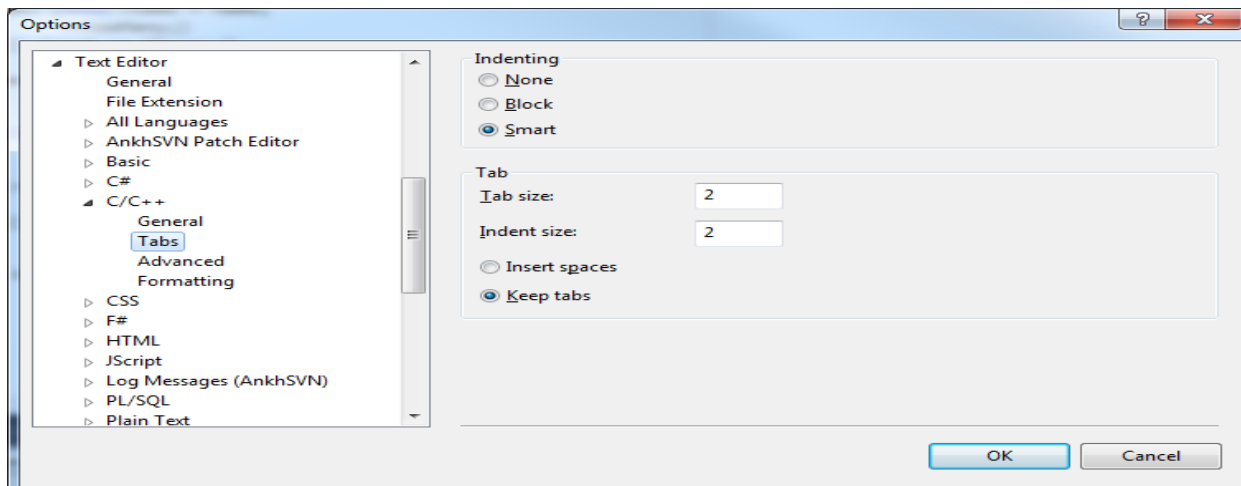
    for (count = 0; '\0' != name[count]; ++count)
    {
    }

    return count;
}
```

Formatting

- Indentation

Two spaces must be used as the unit of indentation per tab. Every IDE (Integrated Development Environment) such as Visual Studio includes an option for changing the number of spaces in a tab. These can usually be found in the preferences section. In Visual Studio 2010 go to Tools->Options ->Text Editor->C/C++->Tabs. At this point make sure the Tab Size and Indent Size are 2 and that the radio button for insert spaces is selected. Select these options before typing in any of your code.



- Line Length

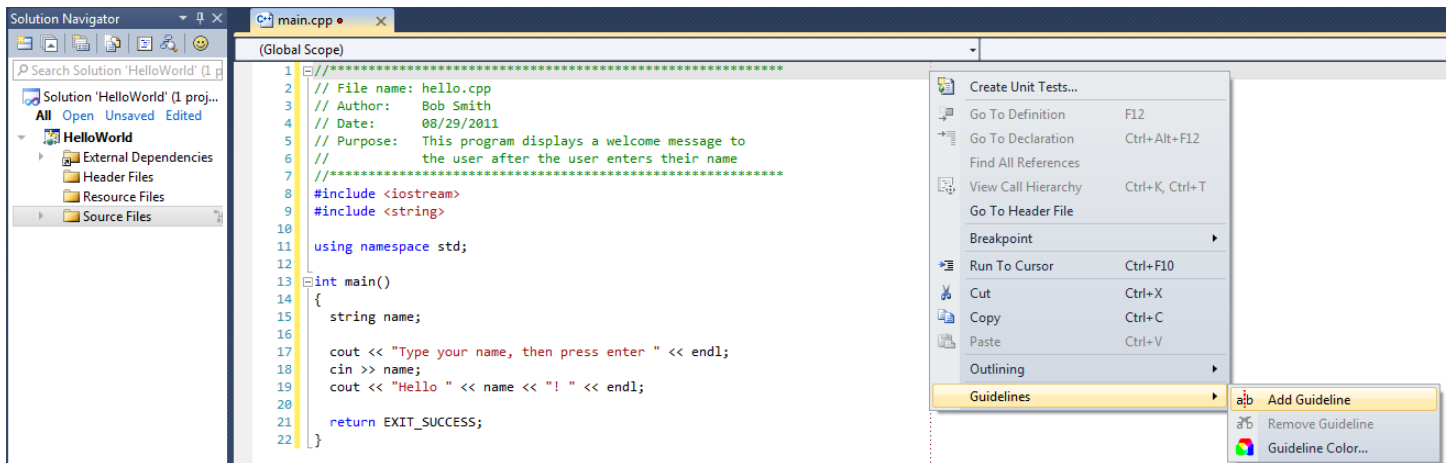
Lines must be no longer than 80 characters. Anything longer than 80 characters is normally not handled well in many terminals and tools. Further, a line longer than 80 will not wrap nicely when outputted.

In Visual Studio 2010, you can set a guideline at 80 columns to help you know when you have typed 80 characters. In order to do this, you need to make sure that the Productivity Power Tools have been installed and enabled.

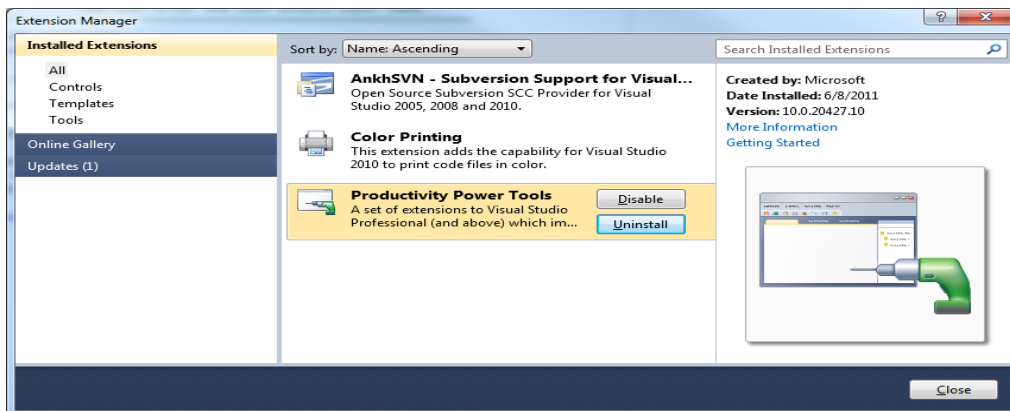


The image shows a screenshot of the Visual Studio gallery page for the 'Productivity Power Tools' extension. The page has a dark blue header with the Visual Studio logo and a search bar. Below the header is a navigation menu with options like 'Home', 'Getting Started', 'Library', 'Learn', 'Downloads', 'Extend', and 'Gallery'. A secondary navigation bar includes 'Gallery Home', 'My Contributions', 'My Favorites', 'My Notifications', and 'Feedback'. The main content area shows the breadcrumb 'Gallery > Tools > Productivity Power Tools'. The extension title 'Productivity Power Tools' is displayed in large blue text, followed by the 'Microsoft' logo and a 'Free' badge. Below this, the 'CREATED BY' field shows 'Microsoft'. The 'REVIEWS' section displays a star rating of 4.5 out of 5, with '(603) Review' text. The 'SUPPORTS' field lists 'Visual Studio 2010'. The 'DOWNLOADS' section features a purple 'Download' button and the number '(998,037)'.

These tools have been installed in the Marsh Labs but you will need to enable them. Once enabled, you can add a guideline. Simply place your cursor as line 80 and right-click. If the Productivity Power Tools are correctly installed you will get the Guidelines->Add Guideline option. After adding a guideline, you can see the dotted red line at column 80.



If you have correctly installed the Productivity Power Tools but they are not enabled, start Visual Studio and then go to Tools->Extension Manager. Make your Productivity Power Tools look like the following:



- Wrapping Lines

If an expression cannot fit on a single line then break it:

- After a comma
- Before an operator

Make sure that the new line is aligned with the beginning of the expression at the same level on the previous line. As an example, consider a long cout statement as follows:

```
cout << "The number of females is: " << female << "The number of males is: " << male << endl;
```

would be written as:

```
cout << "The number of females is: " << female
    << "The number of males is: " << male << endl;
```

- Spaces

All arithmetic and logical operators must have one space before and after the operator. The only exceptions are:

- Unary operators
- The period
- No spaces before the comma and only one space after the comma

- Blank Lines

Use blank lines to separate distinct pieces of code. For example, one blank line before and after a while loop helps the code reader. The important thing to remember is that blank lines must be used consistently.

- Braces

Any curly braces that you use in your program (e.g. with if statements, loop constructs, and functions) must appear on their own lines. Any code within the braces must be indented relative to the braces.

```
if (0 == number % 2)
{
    cout << "Even Number" << endl;
}
else
{
    cout << "Odd Number" << endl;
}
```

Comments

Comments are used to explain the purpose of the code fragment they are grouped with. Comments state what the code is doing, while the code itself shows how you are doing it.

Use comments as follows:

- File Header

The main purpose of a file header is to explain the purpose of the program as briefly as possible. You must include the following sections in your program header:

- File name
- Author
- Date
- Class Title
- Assignment Title
- Purpose

```
//*****  
// File name:  main.c  
// Author:    Joe Bloggs  
// Date:     9/29/11  
// Class:    CS150  
// Assignment: Fraction Calculator  
// Purpose:   This program allows elementary children to enter the numerator  
//           and denominator of two fractions. A common denominator will be  
//           found for adding the two fractions. The resulting fraction  
//           from the addition of the two fractions will be outputted  
//           along with the decimal result of the fraction  
//*****
```

- Declaration Comments

Variables must be given a meaningful variable name. Each variable can have a sidebar comment to the right indicating the variable's purpose if the purpose of the variable is not totally obvious. Do not put any blank lines between the variables being declared. You must also group together variables that are related.

```
int seconds;  
int minutes;  
int hours ;  
char firstName[MAXNAMESIZE];  
char lastName[MAXNAMESIZE];
```

None of the above variables need to be documented because it is obvious what they represent. When your programs become more complicated, some variable names might need a little documentation.

- Sidebar Comments

A sidebar comment appears on the same line as the single statement it is describing. The comment must be brief and not exceed that line. Try and line up your sidebar comments as much as possible for easy reading.

```
value <<= 1;    // multiply value by 2
```

- In-line Comments

In-line comments appear on their own lines and precede the segment of code they describe. You must use in-line comments to describe complex code that is not limited to a single statement. You must use one blank line to separate the comment from the segment of code being described. Every block of code (if statement, for loop, while loop, do-while loop, switch statement, ...) must have an in-line comment preceding the actual code briefly letting the reader know what the code is doing. Here is an example.

```
// Calculate the cost of a quantity of widgets. Widgets have
// a base price for the first 1000. After the first 1000, each
// additional widget costs half price.

if ( PRICE_DROP_QUANTITY < quantityBought )
{
    totalCost = PRICE_DROP_QUANTITY * BASE_PRICE +
        ( quantityBought - PRICE_DROP_QUANTITY ) * BASE_PRICE / 2.0;
}
else
{
    totalCost = quantityBought * BASE_PRICE;
}
else
{
    grossPay = hoursWorked * hourlyWage;
}
```

- Function Comments

Once we get into breaking up our programs into functions, each function needs to be documented as follows:

```
/*
Function:    printFraction

Description: Outputs a fraction in the form
            numerator / denominator to the screen

Parameters: numerator - the numerator of the fraction to be printed
            denominator - the denominator of the fraction to be printed

Returned:   None
*/

void printFraction (int numerator, int denominator)
{
```



```
printf ("%d / %d", numerator, denominator);  
}
```

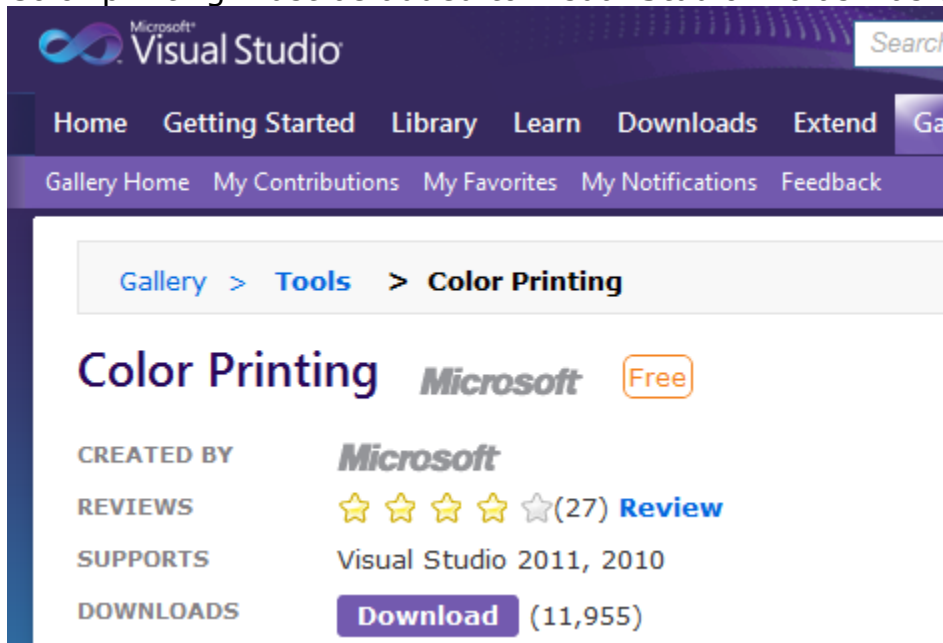
Notice that there is one blank line between the sections Function, Description, Parameters, and Returned. Also, notice that all text for each section is left-justified. Finally, each parameter is on a line by itself and the dashes line up. This is exactly how each function is to be documented.

Printing

When printing your code you must use a fixed width font. Courier and Courier New are examples of fixed width fonts. You must also make sure that your lines do not wrap nor do they get cut off when printing. All printing is to be done in Portrait.

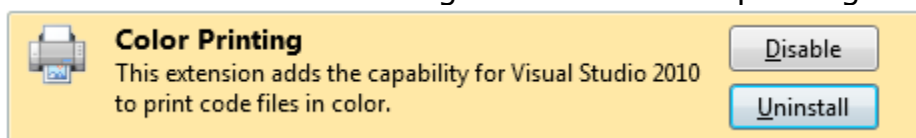
The final output you will turn in is to be printed in color since comments, keywords, strings, etc are highlighted for easy reading. Multiple pages are to be printed double-sided and stapled in the upper-left corner.

Color printing must be added to Visual Studio Pro as was the Productivity Power Tools.



The screenshot shows the Visual Studio gallery interface. At the top, there's a search bar and navigation tabs: Home, Getting Started, Library, Learn, Downloads, Extend, and Gallery. Below this, there's a sub-navigation bar with links: Gallery Home, My Contributions, My Favorites, My Notifications, and Feedback. The main content area shows the 'Color Printing' extension by Microsoft, which is free. It lists 'CREATED BY Microsoft', 'REVIEWS' with 4 stars and 27 reviews, 'SUPPORTS Visual Studio 2011, 2010', and 'DOWNLOADS' with a 'Download' button and 11,955 downloads.

As with the Productivity Power Tools, you must make sure that this extension is enabled. You should see the following for correct color printing.



The screenshot shows the Visual Studio extension status bar for the 'Color Printing' extension. It features a printer icon, the extension name 'Color Printing', and a description: 'This extension adds the capability for Visual Studio 2010 to print code files in color.' There are two buttons: 'Disable' and 'Uninstall'.