Southern New Hampshire University

IT 312 Code Formatting

Styling and formatting your code is important in any programming language. It promotes readability and maintainability of your code when either you or another developer needs to work with your code after its initial creation. While style can include design patterns, you will focus on the aesthetics. This includes naming conventions, tabbing versus spacing, use and placement of braces, and line spacing.

Each developer or company will, over time, develop a code style based on an initial set of format guidelines. By performing an internet search, you will find many sets of formatting guidelines used by various organizations. Follow the guidelines presented here as you develop and format code in this course.

Indentation in Nested Code Blocks

Anytime you have a nested block of code, use an indentation to indicate which lines of code belong within the nested block. This supports readability of the code. It also makes debugging easier as you can quickly see if a line of code is in the wrong spot.

Before:

```
for (int count = 0; count < 5; count++) {
if (foo) {
foo = false;
cout << "Hello";
} else {
foo = true;
cout << "Goodbye";
}
}
```

After:

```
for (int count = 0; count < 5; count++) {
  if (foo) {
    foo = false;
    cout << "Hello";
  } else {
    foo = true;
    cout << "Goodbye";
  }
}
```

Naming Conventions: Be Consistent and Be Descriptive

The language C++ allows for any combination of uppercase and lowercase letters in names. However, in this course there are a few rules you will be following. The first letter of each variable name should be lower case. The first letter of each class name or file name should be upper case.

For variable names themselves, there are options such as **camelCase** and **names_with_underscores**. Both are acceptable, but once you pick a style, stick with it; do not mix conventions.

Also be descriptive with variable names. When you look at your code later, whether a week or a year from now, the variables should describe their purpose. For example, write out the word “length” instead of abbreviating and try to avoid variable names such as x, y, and z.

Before:

```
class rectangle {                  //Capitalize 'r' since it is a class
    int length, Width;             //'W' lowercase since it is a variable
  public:
    void set_values (int, int);  //Mixing naming conventions with next line
    int CalculateArea() {return length * Width;} //'C' lowercase since it
is a method call
};
```

After:

```
class Rectangle {
    int length, width;
  public:
    void setValues (int, int);
    int calculateArea() {return length * width;}
};
```

Use a Space After Commas and Semicolons

Put a space after commas and semicolons that are used within lines of code (a space is not needed after an end of line semicolon). Do not put spaces before these characters.

Before:

```
for (int lorem = 0 ; lorem < 5 ; lorem++) {
  ipsum(var1 ,var2,var3);
}
```

After:

```
for (int lorem = 0; lorem < 5; lorem++) {
  ipsum(var1, var2, var3);
}
```

Southern New Hampshire University

Use a Space Before and After Binary Operators

Binary operators (operators that have two operands) such as <, >=, +, =, etc. should have a space both before and after with the exception of ->. Unary operators (only one operand) such as !, ++, --, etc. should not have extra spaces.

Before:

```
for (int foo=0; foo<5; foo++) {
  sum=x+y*7;
  bar->invokeFunction();
}
```

After:

```
for (int foo = 0; foo < 5; foo++) {
  sum = x + y * 7;
  bar->invokeFunction();
}
```

Place a Blank Line After Significant Blocks of Code

Enter a blank line in your code to distinguish pieces of functionality. Think of this like starting a new paragraph in a paper. After you finish writing about one thought, you make a new paragraph to write about the next. You can also use comments (which will be described later in this document) to highlight the importance of each code section.

Before:

```
void foo() {
  y.save(feandra[0]);
  z.save(feandra[1]);
  luezoid[0].gleblu(y);
  luezoid[1].gleblu(y);
  luezoid[2].gleblu(z);
  luezoid[3].gleblu(z);
  if (dracaena.getAlive() >= 5) {
    //Yikes!
  }
}
```

After:

```
void foo() {

  //Start saving the feandras from the dracaena.
  y.save(feandra[0]);
  z.save(feandra[1]);

  //Now use the luezoids...
  luezoid[0].gleblu(y);
  luezoid[1].gleblu(y);
  luezoid[2].gleblu(z);
  luezoid[3].gleblu(z);

  //Check if the dracaena are still alive.
  if (dracaena.getAlive() >= 5) {
    //Yikes!
  }
}
```

When to Use Braces {}

Braces should be used anytime a nested block of code is longer than one physical line. Even if it is one line, it is still recommended that you use braces to surround the code. This is because it creates consistency in your style and makes later changes easier by eliminating potential bugs due to incorrect nesting.

Before:

```
if (foo)
  for (int count = 0; count < 5; count++) {
    cout << "Hello, " << count << endl;
    doSomething(count >> 2);
  }
```

After:

```
if (foo) {
  for (int count = 0; count < 5; count++) {
    cout << "Hello, " << count << endl;
    doSomething(count >> 2);
  }
}
```

Wrap Text on Long Lines

Monitor sizes vary, which means while you might have a large screen, another developer on your team might not. Make sure that your code doesn't cause a horizontal scroll bar to be able to read the full line.

Also, long lines make it difficult to read the code if you need to print it out for code reviews or debugging.

Before (the line is so long that it wraps in this guide):

```
if (((price1 > 0) && (price1 < 10)) || ((price2 < 100) && (price2 > 80)) ||
((price3 == 20) || (price3 == 50)))
```

After:

```
if (((price1 > 0) && (price1 < 10)) ||
   ((price2 < 100) && (price2 > 80)) ||
   ((price3 == 20) || (price3 == 50)))
```

Use Parentheses to Delineate Precedence

Order of precedence is an important concept in programming. Operators have a pre-defined order of operations in which they will execute. To solve expressions that do not follow this order, use parentheses to show the desired order.

Before:

```
int x = y * z + p * q >> 2;
```

After:

```
int x = (y * (z + p) * q) >> 2;
```

Use a Space After Keywords Such as *if*, *while*, *for*, etc.

Just like we use spaces after commas and semi-colons to make reading easier, we also need spaces after certain keywords, such as those in various control structures (if, while, for, and switch just to name a few).

Before:

```
if(foo) {
  while(true) {
    //fun code here
  }
}
```

After:

```
if (foo) {
  while (true) {
    //fun code here
  }
}
```

When and How to Write Comments

A comment is a statement in the code that is ignored by the compiler. In C++, they are noted by starting the text with two slashes (//). Use comments to describe pieces of code such as chunks of code or complex algorithms. In the second example below, the comment will briefly describe purpose of the logic used in the following control structure. Try to avoid stating the obvious like "//created a variable" or "//started a loop".

What a comment is:

```
//This is a C++ style comment
```

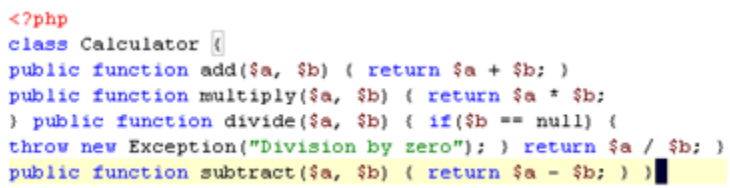
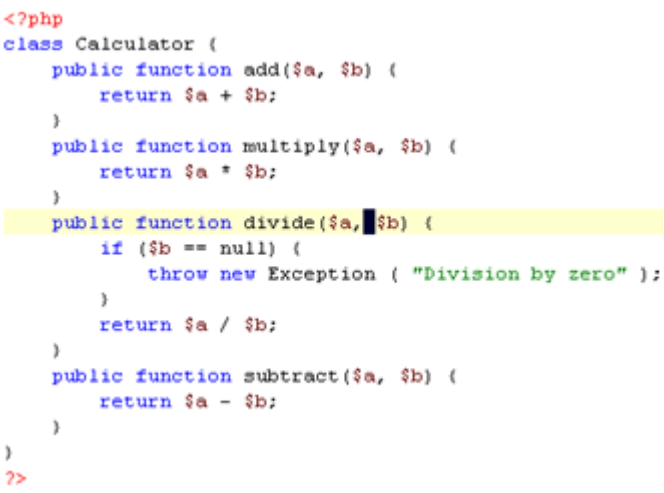
When to use:

```
//This needs a comment
if (expression)
{
  statements;
}
else
{
  statements;
}
```

Southern New Hampshire University

Unformatted Versus Formatted Examples

In the two following sets of images, note the difference in the readability of the code. The code itself remains unchanged. Only the formatting styling has been updated.

Unformatted:	Formatted:
	
Source: Eclipse. (2015). Unformatted code. Retrieved from http://www.eclipse.org/pdt/help/html/formatting_code.htm	Source: Eclipse. (2015). Formatted code. Retrieved from http://www.eclipse.org/pdt/help/html/formatting_code.htm

Southern New Hampshire University

Unformatted:

```
      private function getGreeting(userName:String, userTitle:String):String {
        var lowcaseTitle:String = userTitle.toLowerCase();
        var greeting:String = "";
      switch (lowcaseTitle) {    case "mr":
        greeting = "Hello, Mr. " + userName;break;
        case "mrs":greeting = "Hello, Mrs. " + userName;break;
        case "miss":greeting = "Hello, Miss " + userName;break;
        case "queen":greeting = "Good day, your Majesty";break;
     default:greeting = "Hi, " + userName;
   }return greeting;}

private function greetUser():void {
 var userName:String=nameInput.text;
 var userTitle:String = titleInput.text;
 if((nameInput.text.length == 0)||(titleInput.text.length == 0)) {
    textoutput.text = "Please enter your name and title";
 } else {textoutput.text = getGreeting(userName, userTitle);}}
```

Source: Collingbourne, H. (2008). Unformatted code. Retrieved from http://www.sapphiresteel.com/Blog/ActionScript-Code-Formatting-In.html

Formatted:

```
    private function getGreeting (userName:String, userTitle:String):String
    {
      var lowcaseTitle:String = userTitle.toLowerCase();
      var greeting:String = "";
      switch (lowcaseTitle)
      {
        case "mr":
          greeting = "Hello, Mr. "+userName;
          break;
        case "mrs":
          greeting = "Hello, Mrs. "+userName;
          break;
        case "miss":
          greeting = "Hello, Miss "+userName;
          break;
        case "queen":
          greeting = "Good day, your Majesty";
          break;
        default:
          greeting = "Hi, "+userName;
      }
      return greeting;
    }

    private function greetUser ():void
    {
      var userName:String = nameInput.text;
      var userTitle:String = titleInput.text;
      if ((nameInput.text.length==0)||(titleInput.text.length==0))
      {
        textoutput.text = "Please enter your name and title";
      } else
      {
        textoutput.text = getGreeting(userName, userTitle);
      }
    }
```

Source: Collingbourne, H. (2008). Formatted code. Retrieved from http://www.sapphiresteel.com/Blog/ActionScript-Code-Formatting-In.html