C++ Coding Style Rules

- Align all curly braces vertically to allow easy visual matching.
- Use four (4) spaces for each indentation.
- Indent all statements contained within compound statements (if, for, while, {}, etc).
- Use meaningful names for variables, parameters, subprograms, and classes. (You may use simple names like i, c, s, f, or p for general integers, characters, strings, floats, or pointers respectively.)
- Use the following (Java-like) conventions for identifiers:

Identifier	Convention	Examples
class names	capitalize first word, capitalize all other words (no underscores)	Circle, FilledCircle, ShadedFilledCircle
class member names	lowercase first word, capitalize all other words (no underscores)	draw(), drawLine(), centerX, centerY, radius
function names	lowercase first word, capitalize all other words (no underscores)	findIndex(), factorial(), toLowerCase(), isUpperCase()
named constants	all caps, use underscores to separate words	PI, MAX_BUFFER_SIZE

- Use whitespace around every binary operator (e.g, x + y instead of x+y) and around function arguments (e.g, put(x, y) instead of put(x,y)).
- Keep subprograms simple. At most five simple statements or one compound statement and two simple statements.
- Keep compound statements simple. At most three simple statements per statement list.
- Avoid over parenthesization, e.g, return x + 1; instead of return (x + 1);
- Comment each function, each class, and each class member (describing it's purpose) using a form similar to the example at the very end. (Note I only commented a few members to avoid cluttering up the Class example.)

Examples of Subroutines

A function

```
int factorial( int n )
{
   if ( n <= 0 )
      return 1;
   else
      return n * factorial( n - 1 );
}
```

A procedure

```
void print( int n )
{
   for ( int i = 0; i < n; i++ )
      cout << n << ' ';
   cout << endl;
}
```

An operator

```
ostream & operator << ( ostream & out, Circle c )
{
   return out << "Circle is " << c.radius;
}
```

The main routine and it's arguments

```
#include < iostream.h>

int main()

{
```

```
                while ( true )
{
                    cout << "this is the class that never ends,\n";
                    cout << "yes it goes on and on my friend,\n";
                    cout << "some students started taking it not knowing what it was,\n";
                    cout << "and they'll continue taking it forever just because\n";
                }
                return 0;
            }
```

Examples of statments

simple if/else

```
if ( x < y )
    y = 0;
else
    x = 1;
```

cascaded if/else

```
if ( var1 < var2 )
{
    var1 = ...;
    var2 = ...;
      ...
}
else if ( var1 == var2 )
{
    var1 = ...;
    var2 = ...;
    ...
}
else
{
    var1 = ...;
    var2 = ...;
      ...
}
```

switch statement

```
switch ( factorial( i ) )
{
    case 1:
    case 3:
    case 5:
        cout << "It's odd!\n";
        break;
    case 2:
    case 4:
    case 6:
        cout << "It's even!\n";
        break;
    default:
        cout << "It's less than 1 or greater than 6!\n";
        break;
 }
```

loop statements

```
for ( int i = 0; i < MAX_VAL; i++ )
{
    ...
```

```
}



while ( !cin.eof() )
{
    cin.get( ch );
    cout.put( ch );
}

do // read the first line (including the newline) only
{
    cin.get( ch );
    cout.put( ch );
} while ( ch != '\n' );
```

Examples of Class Definitions

```
/*
   classes Shape, Point, and Rectangle define a class hierarchy
   for geometrical shapes.
*/
class Shape
{
public:
    // draws this shape on the display
    virtual void draw() = 0;
    // inverts this shape along a horizontal line
    virtual void flip() = 0;
    // rotates this shape by number of degrees specified by angle
    virtual void rotate( int angle ) = 0;
};

class Point
{
public:
    // the origin of this point in 2-D space
    int x, y;
    Point( int newX, int newY )
        : x( newX ), y( newY )
    {
    }
};

class Rectangle
    : public Shape
{
private:
    Point tl;
    Point br;
    int angle;

protected:
    ...

public:
    Rectangle();
    virtual ~Rectangle();
```

```
    void draw();
    void flip();
    void rotate( int angle );
    float area();
};
```

Examples of REALLY BAD commenting

All the following comments are poor. Either they state the obvious about the programming language (typical from programmers just learning the language), or they don't add any information about the item they are commenting and the program would actually be more readable if they were deleted. The final offense is pointing out that a particular close brace matches some open brace. If you find yourself needing this, either you aren't aligning your braces properly, or your code is getting too complex.

```
/*
  This is a class defintion for Shape.
*/
class Shape
{
public:
    // These are pure virtual functions
    // draws shape
    virtual void draw() = 0;
    // flips shape
    virtual void flip() = 0;
    // rotates shape
    virtual void rotate( int angle ) = 0;
}; // end of Shape

// A Point class
class Point
{
public:
    // Two integers
    int x, y;
    // a constructor
    Point( int newX, int newY )
        : x( newX ), y( newY )  // construct x and y
    {
      // do nothing here
    } // end of Point
};

class Rectangle
    : public Shape // Rectangle is publically derived from Shape
{
private: // some private members
    Point tl;
    Point br;
    int angle;

protected: // some protected members
    ...

public: // the public members
    Rectangle(); // constructor
    virtual ~Rectangle(); // destructor
    ...
```

```
    void draw(); // some member
    functionsvoid flip()
    {
       if (x != y)
       {
           swap(x,y); // swap the values
           of x and yflip(); // flip again
     } //
     end
      if
    } //
    end
    flip(
      )
     void rotate(
     int angle );


float area();
}; // end Rectangle
```

Get Hands Dirty With Pointers

OBJECT

Using Pointer Variables.

THEORY

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

type *var-name;

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int *ip;            // pointer to an integer
double *dp;         // pointer to a double
float *fp;         // pointer to a float
char *ch          // pointer to character
```

Using Pointers in C++:

There are few important operations, which we will do with the pointers very frequently.

(a) we define a pointer variables

(b) assign the address of a variable to a pointer

(c) finally access the value at the address available in the pointer variable.

This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use ofthese operations:

Sample Program.1

```
#include <iostream.h>
int main ()
{
int var = 20; // actual variable declaration.
int *ip; // pointer variable
ip = &var; // store address of var in pointer variable
cout << "Value of var variable: ";
cout << var << endl;
// print the address stored in ip pointer variable
cout << "Address stored in ip variable: ";
cout << ip << endl;
// access the value at the address available in pointer
cout << "Value of *ip variable: ";
cout << *ip << endl;
return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Sample Program.2

The NULL pointer is a constant with a value of zero defined in several standard libraries, including iostream. Consider the following program:

```
#include <iostream.h>
int main ()
{
int *ptr = NULL;
cout << "The value of ptr is " << ptr ;
return 0;
}
```

```
When the above code is compiled and executed, it produces the following result:
The value of ptr is 0
```

C++ Pointer Arithmetic

As you understood pointer is an address which is a numeric value; therefore, you can perform arithmetic operations on a pointer just as you can a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and –
The arithmetic update will depend on type of pointer.

To understand pointer arithmetic, let us consider that **ptr** is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmatic operation on the pointer:
ptr++

the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer.

Sample Program.3

```
#include <iostream.h>
const int MAX = 3;
int main ()
{
int var[MAX] = {10, 100, 200};
int *ptr;
// let us have array address in pointer.
ptr = var;
for (int i = 0; i < MAX; i++)
{
cout << "Address of var[" << i << "] = ";
cout << ptr << endl;
cout << "Value of var[" << i << "] = ";
cout << *ptr << endl; // point to the next location
ptr++;
}
return 0; }
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Decrementing a Pointer:

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

Sample Program.4

```
#include <iostream.h>
const int MAX = 3;
int main ()
{
int var[MAX] = {10, 100, 200};
int *ptr;
// let us have address of the last element in pointer.
ptr = &var[MAX-1];
for (int i = MAX; i > 0; i--)
{
cout << "Address of var[" << i << "] = ";
cout << ptr << endl;
cout << "Value of var[" << i << "] = ";
cout << *ptr << endl;
// point to the previous location
ptr--;
}
return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var[3] = 0xbfdb70f8
Value of var[3] = 200
Address of var[2] = 0xbfdb70f4
Value of var[2] = 100
Address of var[1] = 0xbfdb70f0
Value of var[1] = 10
```

C++ Pointers vs Arrays

Pointers and arrays are strongly related. In fact, pointers and arrays are interchangeable in many cases. For example, a pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing. Consider the following program.

Sample Program.5

```
#include <iostream.h>
const int MAX = 3;
int main ()
{
int var[MAX] = {10, 100, 200};
int *ptr;
// let us have array address in pointer.
ptr = var;
for (int i = 0; i < MAX; i++)
{
cout << "Address of var[" << i << "] = ";
cout << ptr << endl;
cout << "Value of var[" << i << "] = ";
cout << *ptr << endl;
// point to the next location
ptr++;
}

return 0;}
```

When the above code is compiled and executed, it produces result something asfollows:

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Sample Program.6

```
#include <iostream.h>
const int MAX = 3;
int main ()
{
int var[MAX] = {10, 100, 200};
int *ptr[MAX]; for (int i = 0; i < MAX; i++)
{
ptr[i] = &var[i]; // assign the address of integer.
}
for (int i = 0; i < MAX; i++)
{
cout << "Value of var[" << i << "] = ";
cout << *ptr[i] << endl;
}
return 0; }
}
```

When the above code is compiled and executed, it produces the following result:

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Rest of the pointers use cases in next meetup……

Thanks To: http://doc.ece.uci.edu/~klefstad/s/C++-style.html