C++ Best Programming Practices

Umair Azfar Khan

August 17, 2018

1 Introduction

This document will highlight the coding methodology that you should use while doing programming. You should follow these guidelines to the letter as your marks will depend on them. The rubric for every assignment will use these guidelines by default. Failure in following these guidelines will have your marks deducted. You will get 0.5 marks deducted for every mistake that you make even if the same mistake was repeated.

2 Naming Convention

The first task you encounter when you start to write code is how to name your variables, objects, and functions. The importance of naming conventions can not be underestimated. Providing proper naming will result in self-documenting code, easily comprehended by others. Unless you want to obfuscate the source code, you should follow these guidelines.

Use a name that unambiguously describes a variable or object. A function should contain in its name a verb of the action that it implements. Below are guidelines about the naming conventions you should use for variables, functions and classes(ref: https://www.codeproject.com/Articles/26222/C-Coding-Practices-Guide)

```
//Always give meaningful names
int someVariable;
float barWeight;
unsigned int usersNumber;
bool isEngineStarted;
double circleArea;
double circleArea; //refer to a class private variable
// in code as this->circleArea

void* someVoidPointer;

const int UsersNumber;

int i, j, n, m, tmp; //some local loops variables, temporary
    variables

namespace MyNameSpace;

array<int> users;
array<string> userNames;

class SomeClass;

int DoWork(int timeOfDay);
float CalculateRadius(const& Circle circle);
int StartIOManager();
int OpenDvdPlayer(); //if abbreviation takes more than 2 letters
// do not capitalize the whole word
```

3 Indentation

The whole idea is to clearly define where a block of control starts and ends. It also has the added benefit of warning you when you're nesting your functions too deep which means that you need to revise your code and make it simpler.

```
//Sample indentation
void FaceDetector::estimate_motion(const vec2Dc* search_mask)
{
   if (search_mask == 0)
      m_motion_amount = -1.0f;
   else
   {
      unsigned int motion_pixels = 0;
      unsigned int pixels = 0;
      for (int y = get_dy(); y < mask->height(); y++)
      {
         for (int x = get_dx(); x < search_mask->width(); x++)
         {
            total_pixels++;
            if ((*search_mask)(y, x) == 1)
               motion_pixels++;
         }
      }
      m_motion_amount = float(motion_pixels) / float(pixels);
   }
}
```

4 Comments

You write comments right before a function starts to explain what the function does. You do not need to write comments for every line of code. You should however write comments when there is a necessity to explain your code. It is better to comment out faulty code than to delete it but your final submission should not contain anything commented out.

5 Spaces, Braces, and Parenthesis

For functions and classes, use the following template for definition

```
void some_function(int param)
{
   //function body
}

class SomeClass
{
   //class body
};
```

The spaces in conditionals and math expressions should follow this style:

```
float a, b, c, d;
int e, f, g, h;

a = (b + d) / (c * d);

e = f - ((g & h) >> 10); //good
e =f-( (g&h ) >>10);     //bad
```

6 Multiple Inclusion Guard

Wrap your header file contents with the multiple inclusion guard to prevent double inclusion:

```
#ifndef Foo_h
#define Foo_h
// ... Foo.h file contents
#endif
```

Another way of doing the same is by using the preprocessor directive #pragma once

```
#pragma once

// ... Foo.h file contents
```

7 Class Layout

When you declare a class, remember that if you do not provide definitions for the following functions, they will be provided automatically by C++:

- constructor
- copy constructor
- assignment operator
- address-of operator (const and non-const)
- destructor

```
//declaring that class
class SomeClass
{
};

//you get these functions provided automatically
class SomeClass
{
   public:
   SomeClass() { }                  //constructor
   ~SomeClass() { }                 //destructor
   SomeClass(const SomeClass &rhs); //copy constructor
   SomeClass& operator=(const SomeClass& rhs);
   SomeClass* operator&();
   const SomeClass* operator&() const; //operators;
};
```

Provide class declarations in the public, protected, and private order. This way, the users of a class will be able to immediately see the public interface they need to use:

```
#ifndef ClassName_h
#define ClassName_h

class ClassName
{
public:
   ClassName();
   ClassName(const ClassName& classname);
   virtual ~ClassName();
   // Operators
   const ClassName& operator=(const ClassName& classname);
   // Operations
   int do_work();
   int start_engine();
   int fire_nuke();
   // Access
   inline unsigned int get_users_count() const;
   // Inquiry
   inline bool is_engine_started() const;
protected:
//protected functions for descendant classes
private:
//private data and functions
   unsigned int m_users_count;
   bool m_is_engine_started;
};

// Inlines
inline unsigned int SomeClass::get_users_count() const
{
   return m_users_count;
}

inline bool SomeClass::is_engine_started() const
{
   return m_is_engine_started;
}

#endif ClassName_h
```

8 Some Pointers

- Instead of deleting faulty code, always comment it out. You may need to use it in future and it will be problematic to write it all over again. Once the problem has been fixed, only delete the commented out code then.
- A function should not be more than 20 lines long. If it goes over that count then try to make another function.
- Always use git hub while making your project or assignments

9 Conclusion

When doing programming, you have to think about it as your fingerprint. When someone sees your code they acquire a certain frame of reference about your character and abilities. It exhibits how organized and methodical you are. Following the above mentioned guidelines will automatically make it a force of habit to write nice and clean code. It will not only help others in following your code but will also help you whenever you return to your old code.