C++ Programming Style Guide

D. S. Hwang
IDA Lab.
Dankook University

1 Introduction

- ⋆ All major real-world software projects have a "house style"("coding guide-line", "project standard", etc).
- ⋆ Make sense that different projects have followed different styles.
- ⋆ No one true style has not accepted for all developers.
- ⋆ We plan to follow a commonly used guideline in order to keep our coming code expressed integrative and maintainable.

2 Naming

- ⋆ Use a single capital letter to start type names such as class name, user-defined types: `Stack`, `Temperature`.
- ⋆ Names of non-types are not capitalized: `x`, `temp`.
- ⋆ Use underscores fro multi-part names: `sym_tbl`, `max_size`.
- ⋆ Don't use all capitals!

3 Indentation

- ⋆ Use 4 whitespaces for indentation.
- ⋆ Follow "K&R Style" or "Kernighan and Ritchie Style".

```
// if statement:
if(a == b) {
    ...
} else {

// loop statement:
for(int i=0;i < vec.size(); i++){
    ...
}

// switch statement
switch(a) {
case a:
        //
        break;
    ...
default:
        //
        break;
}
```

```
// function:
double sqrt(double d)
{
    ...
}

// class or struct:
class Temp_reading {
public:
    ...
private:
    ...
};
```

4 Whitespace

- ⋆ Use blank lines between functions and between classes to separate logically different sections of declaration code.
- ⋆ Don't use dense text.
- ⋆ Use a single line for an `if`, `for`, `while`-statement or a second line only if the resulting line is very short and simple.
- ⋆ Don't use many parentheses.

```
void fun()
{
  Vector<string> v;
  int x = 7; char* p = 29; // don't

  string s;
  while(cin>>s)
     v.push_back(s);
```

```
    while(cin>>s) v.push_back(s);
}

class X {
    ...
};
```

- ⋆ Put spaces near the "pointer to" declaration operator ∗.

```
int* p;    // do it this way
int  *p;   //don't
int  * p;  // don't
int*p;     //don't
```

- ⋆ Do initialize variables if possible when defining them.

```
int* p = &v[i];
```

5 Comment

- ⋆ Use comments to explain what a code does
- ⋆ Comments are good for
  - state intention(what is this code supposed to do)
  - simply state strategy(the general idea of this is ...)
  - state invariant, pre-conditions and post-conditions
- ⋆ Start code files(`fn.h`, `fn.hpp`, `fn.cpp`, `fn.c`) with a comment the name of a designer, the date, and what the program is supposed to do.
- ⋆ Don't overuse comments.

```
// D.S.Hwang, 5/30/2017
// solution for exercise 6.5

/*
      Write a Fibonacci numbers.
      Find the largest Fibonacci number that fits in an int
*/
```

```
#include "std_lib_facilites.h"

void f()
// Compute the series and note when the int overflows;
// the previous value was the largest that fit
//
{
    ...
}
```

6 Declarations

⋆ Use one line per declaration and comment variables.

```
const int max = v.size()/2; // maximum partition size
int       nmonths = 0;      // number of months before current date

int p, q, r, b; // don't
```

⋆ Write a function name and arguments on a single line.

```
int find_index(const string& s, char c)
// func c's position in s( -1 means 'not found')
{
   ...
}
```

7 Variables and constants

⋆ Always initialize variables if possible.

⋆ Don't declare a variable or constant before an appropriate value occurs.

```
vector<int> make_random_numbers(int n)
{
    if(n<0) error(''make_random_number: bad size'');
    vector<int> res(n); // Good!
```

```
        ...
    return res;
}
```

- ⋆ A variable that is immediately used as the target for an input operation may not be initialized.

```
int x;
cin >> x;
```

```
vector<string> vec;
for( string buf; cin >> buf ) vec.push_back(buf);
```

- ⋆ Don't use "magic constants":

```
for(int i=0; i < 32; ++i){
    ...
}
```

```
const int mmax = 32; //put a comment on mmax
    ...
for(int i=0; i < mmax; ++i){
    ...
}
```

- ⋆ Use `const` if a variable doesn't change.

8 Expressions and operators

- ⋆ Avoid overly long and complicated expressions.
- ⋆ Preper prefix `++count` to postfix `count++`
- ⋆ Don't "hide" assignments in the middle of expressions

```
z = a + (b = f(x)) * c; // don't
```

9 Language feature use

- ⋆ Always to play with C++ language features.
- ⋆ Don't use explicit type conversion if possible.
- ⋆ Don't use macros except for `#include` guards.
- ⋆ Avoid global variables
- ⋆ Avoid naked `deletes` and `news`.

10 Line length

- ⋆ Lay out the code so that if fits into a resonably-sized window.
- ⋆ Don't rely on automatic line wrap

```
cout << item_name
     << '': unit = '' << unit_count
     << ''; number of units= '' << number_of_units
     << '\n';
```

11 Error handling and reporting

- ⋆ Assume that the code:
  1. should produce the desired results for all legal inputs
  2. should give reasonable error messages for all illegal inputs
  3. need not worry about misbehaving hardware
  4. need not worry about misbehaving system software
  5. is allowed to terminate after finding an error
- ⋆ If the program detects an error, it may exit by a call of `error()`.

12 Compiler errors and warnings

- ⋆ A program should compile cleanly: no warnings and no errors.