C++ Coding Convention

Google coding style을 따르는 것을 권장합니다.

예)

The #define Guard

All header files should have `#define` guards to prevent multiple inclusion. The format of the symbol name should be `<PROJECT>_<PATH>_<FILE>_H_`.

To guarantee uniqueness, they should be based on the full path in a project's source tree. For example, the file `foo/src/bar/baz.h` in project `foo` should have the following guard:

```
#ifndef FOO_BAR_BAZ_H_
#define FOO_BAR_BAZ_H_

...

#endif  // FOO_BAR_BAZ_H_
```

Order of Includes

Order your includes as follows:

1. dir2/foo2.h. (header)
2. A blank line
3. C system files.
4. C++ system files.
5. A blank line
6. Other libraries' .h files. (dependencies)
7. Your project's .h files. (dependencies)

```
// This file is foo/server/fooserver.cpp
#include "foo/server/fooserver.h"

#include <sys/types.h>
#include <unistd.h>
#include <vector>

#include "base/basictypes.h"
#include "base/commandlineflags.h"
#include "foo/server/bar.h"
```

Scope: namespaces

```
// In the .h file
namespace mynamespace {

// All declarations are within the namespace scope.
// Notice the lack of indentation.
class MyClass {
 public:
  ...
  void Foo();
};

}  // namespace mynamespace
```

```
// In the .cc file
namespace mynamespace {

// Definition of functions is within scope of the namespace.
void MyClass::Foo() {
  ...
}

}  // namespace mynamespace
```

Naming

General Naming Rules

권장되는 이름 양식의 예

```
int price_count_reader;    // No abbreviation.
int num_errors;            // "num" is a widespread convention.
int num_dns_connections;   // Most people know what "DNS" stands for.
int lstm_size;             // "LSTM" is a common machine learning abbreviation.
```

쓰면 안될 이름 양식의 예

```
int n;                     // Meaningless.(의미가 불명확)
int nerr;                  // Ambiguous abbreviation.(모호한 축약어)
int n_comp_conns;          // Ambiguous abbreviation.
int wgc_connections;       // Only your group knows what this stands for.(너랑 너
의 팀만 알수 있음)
int pc_reader;             // Lots of things can be abbreviated "pc".(여러가지로 해
석될 수 있는 축약어 사용)
int cstmr_id;              // Deletes internal letters.(모음을 없애버렸음)
FooBarRequestInfo fbri;    // Not even a word.(정말 하나도 모르겠다)
```

File Names

Filenames should be all lowercase and can include underscores (_) or dashes (-). Follow the convention that your project uses. If there is no consistent local pattern to follow, prefer "_".

Examples of acceptable file names:

- my_useful_class.cc
- my-useful-class.cc
- myusefulclass.cc
- myusefulclass_test.cc `// _unittest and _regtest are deprecated.`

Variables

The names of variables (including function parameters) and data members are all lowercase, with underscores between words. Data members of classes (but not structs) additionally have trailing underscores. For instance: `a_local_variable`, `a_struct_data_member`, `a_class_data_member_`.

```
string table_name;  // OK - uses underscore.
string tablename;   // OK - all lowercase.
```

```
string tableName;   // Bad - mixed case.
```

Variables

배울 수 있는 개념: { 변수, 초기화, 대입 }, { 값의 복사 }, { 스코프와 지역변수 }, { 메모리: 스택 }

아래의 코드는 컴파일이 불가하다. 에러코드를 디버깅해보자.

```
#include <iostream>

int main(int argc, char * argv[]) {
  int x = 5; // this is an initialization
  x = x - 2; // this is an assignment
  std::cout << x << std::endl; // x = 3

  int y = x;
  std::cout << y << std::endl; // y = 3

  // Quiz
  x = x - 1; // x = 2
  std::cout << y << std::endl;  // #1

  {
    int x = x;
```

```
    std::cout << x << std::endl;  // #2
  }
  std::cout << x << std::endl;  // #3

  int z;
  std::cout << z << std::endl; // #4

  return EXIT_SUCCESS;
}
```

```
  {
    // To wrap statments with culy brackets, "{}", bring all of variables inside
to be localized. O.S. starts pushing them on the stack memory, ends up in poping
all.
    // Variable "x" is localized and cannot be initialized by itself!
    //int x = x;
    int x = y;
    std::cout << x << std::endl;  // #2
  }
  std::cout << x << std::endl;  // #3

  // The result of using a variable which has not been initializated is
problematic and undeterminable! -> undefined behavior
  //int z;
  //std::cout << z << std::endl; // #4
```

Chapter 1 comprehensive quiz

Question 1

Write a single-file program (named main.cpp) that reads two separate integers from the user, adds them together, and then outputs the answer. The program should use three functions:

- A function named “ReadNumber” should be used to get (and return) a single integer from the user.
- A function named “WriteAnswer” should be used to outputthe answer. This function should take a single parameter and have no return value.
- A main() function should be used to glue the above functions together.

Hint: You do not need to write a separate function to do the adding (just use operator+ directly).
Hint: You will need to call readNumber() twice.
Hint: If you’re using visual studio with precompiled headers, don’t forget to #include “stdafx.h”.

```
// main.cpp
#include <?>

int ReadNumber() {
  std::cout << "Enter a number: ";
```

```
  ...
  return ?;
}

void WriteAnswer(int x) {
  std::cout << "The answer is " << ?
}

int main() {
  ...
  return EXIT_SUCCESS;
}
```

```
results:

Enter a number: 3
Enter a number: 2
The answer is 5.
```

Question 2

Modify the program you wrote in exercise #1 so that ReadNumber() and WriteAnswer() live in a separate file called "io.cpp". Use a **forward declaration** to access them from main().

Hint: If you're having problems, make sure io.cpp is properly added to your project so it gets compiled.

Question 3

Modify the program you wrote in #2 so that it uses a header file (named io.h) to access the functions instead of using forward declarations directly in your code (.cpp) files. Make sure your header file uses **header guards**.