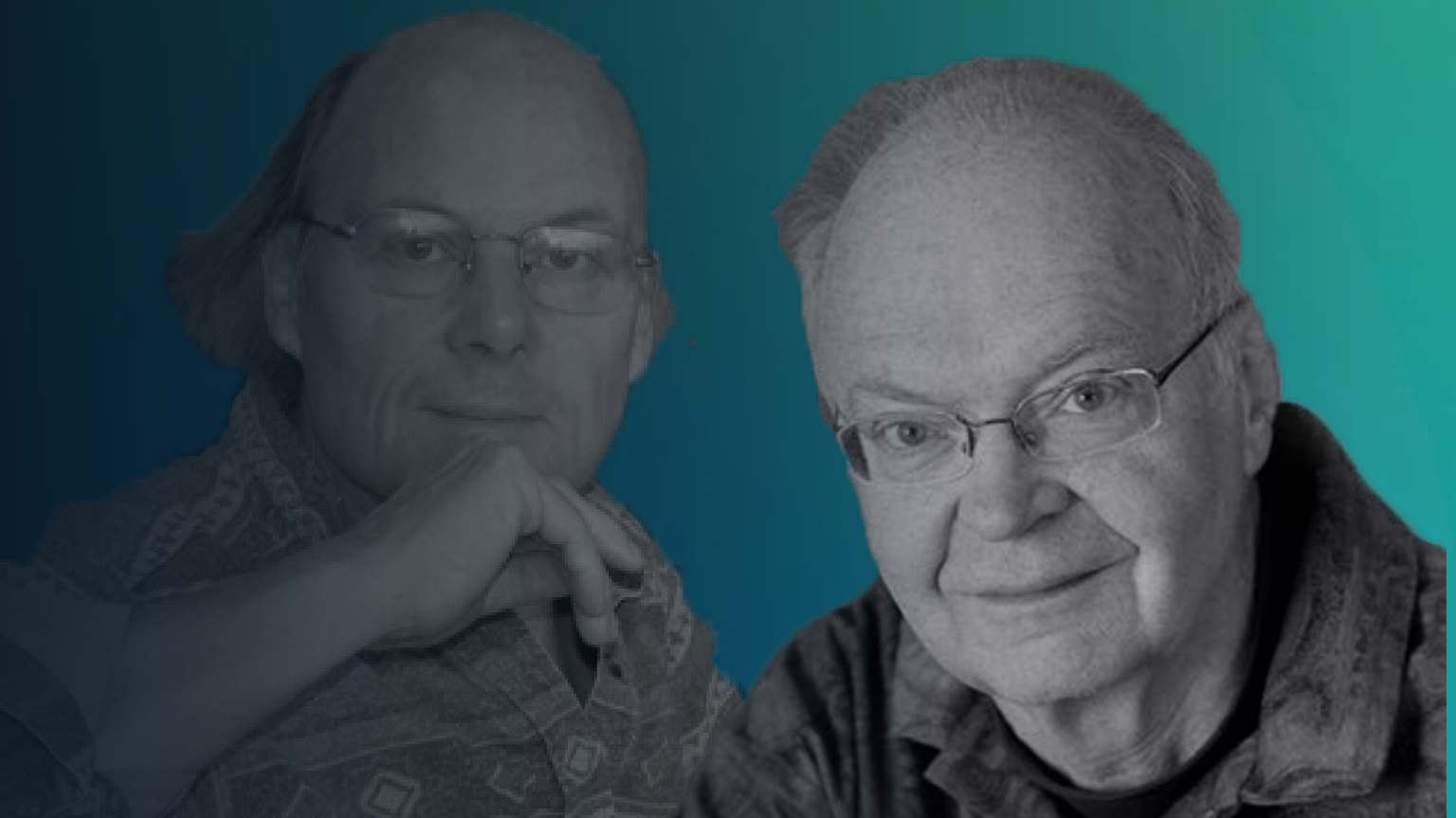
I Know What You Did Last Faculty

: C++ Coding Standard



ISL Lab Seminar

Hansol Kang

Contents

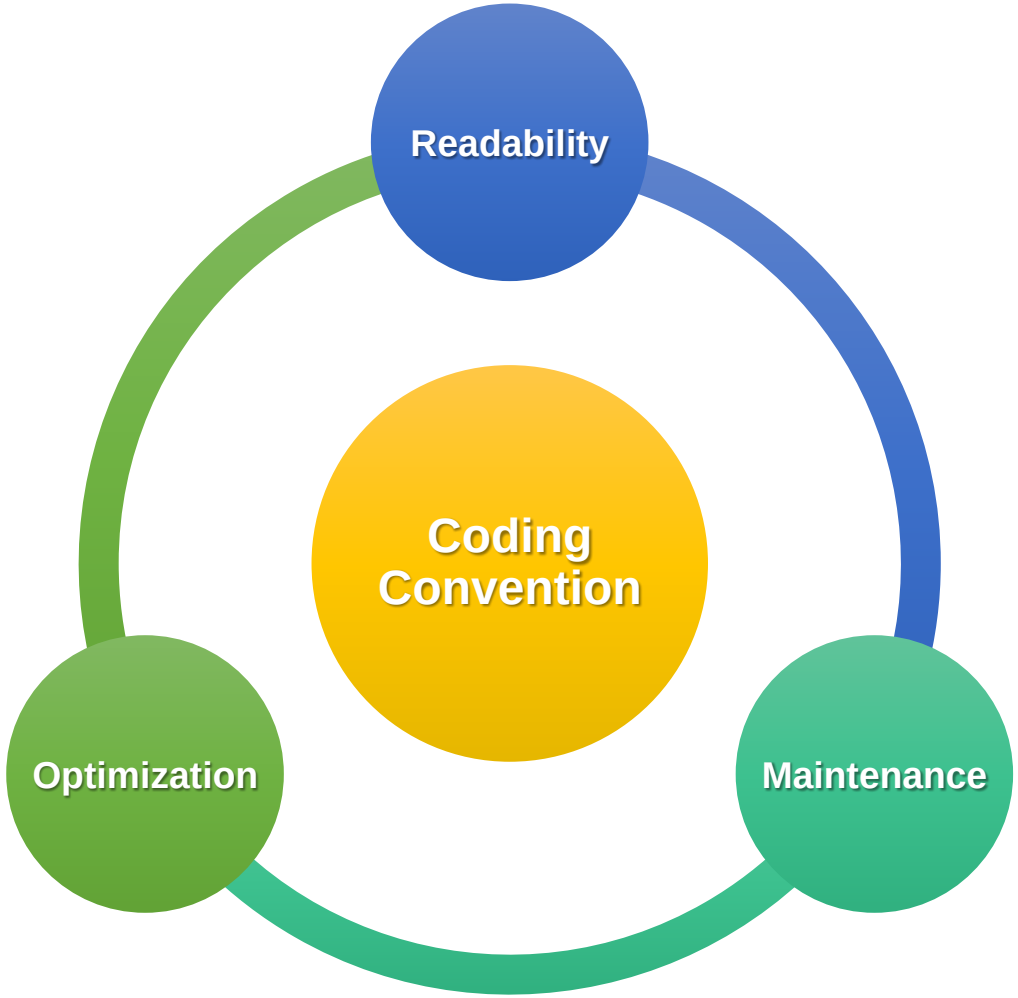
Introduction
Coding Convention

Coding Convention
Rule of thumb
Naming
Header file
Conditional Statement
Comment

Summary

Introduction

- Coding Convention(Coding Standard)




```
if (a > a_1 && b < b_1 && c < c_1 || d>d_1)
```

```
int abc(int a, int b)
{
        Do something
}
```

Do you really need to make it a function?

Coding Convention

Rule of Thumb by 포프

- Readability first (your code should be your documentation most of the time)
- Crash/Assert early. Don't wait until the worst case happens to make the crash condition.
- Follow IDE's auto formatted style unless you have really good reasons not to do so. (Ctrl + K + D in VC++)
- Learn from existing code

Coding Convention

- Naming

Camel case : 각 단어의 첫 문자를 대문자로 표기하며, 맨 처음 문자는 소문자로 표기.(대문자로 단어를 구분)

kurtCobain

Pascal case : 첫 단어를 대문자로 표기.

KurtCobain

Snake case : 단어 사이를 underscore로 구분하여 표기.

kurt_cobain

Hungarian notation : 데이터 타입을 의미하는 접두어를 사용하여 표기.

strKurtcobain

지양하는 추세. **Why?**

1. **IDE의 발달로 데이터 타입을 표기할 필요 x**
2. **변수의 의미를 파악하는 것이 더욱 중요**

Coding Convention

- Google C++ Naming(General Naming Rules)

  함수 이름, 변수 이름, 파일 이름은 **약어를 피하고 서술적으로** 작성.

```
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.

int n;                           // Meaningless.
int nerr;                        // Ambiguous abbreviation.
int n_comp_conns;                // Ambiguous abbreviation.
int wgc_connections;             // Only your group knows what this stands for.
int pc_reader;                   // Lots of things can be abbreviated "pc".
int cstmr_id;                    // Deletes internal letters.
```

  가능하다면 상세한 이름을 사용할 것. 글자 길이를 줄이는 것보다 **새로 읽는 사람이 즉시 이해**하는 것이 더 중요.

Coding Convention

- Google C++ Naming(File Names)

  파일 이름은 모두 소문자이어야 하고, **underscore나 dash**로 단어 사이를 연결함. 반드시 underscore일 필요는 없으며, 프로젝트에서 사용하던 관례를 따름.

  my_useful_class.cc
  my-useful-class.cc
  myusefulclass.cc

url_table.h	클래스 선언
url_table.cc	클래스 정의
url_table-ini.h	많은 코드를 포함한 인라인 함수

Cf. Inline function

인라인 함수는 대부분 헤더에 선언하는 것을 기본으로 그 길이가 너무 긴 경우 inl.h로 따로 작성할 것.

인라인 함수는 함수가 작을 때만(10라인 이나 그 이하) 사용할 것.

인라인 함수의 무분별한 사용은 지양.

```
inline int max_limit(double x){
        return x>255? 255 : (int)x;
}
```

Coding Convention

- Google C++ Naming(Type Names)

  타입 이름은 대문자로 시작하며 underscore 없이 단어마다 첫 글자로 대문자를 사용. 클래스, 구조체, typedef, 열거형을 포함한 모든 타입에 대해 같은 규칙이 적용.

```
// Class and structure
class UrlTable {
struct UrlTableProperties {
```

```
// Enumerated type
enum UrlTableErrors {
```

Coding Convention

- Google C++ Naming(Variable Names)

변수 이름은 **모두 소문자**로 작성하며 **단어 사이에 underscore**를 사용. **클래스 멤버 변수는 이름 끝에 underscore**를 사용.

```
string table_name;  // OK - uses underscore.
string tablename;   // OK - all lowercase.
string tableName;   // Bad - mixed case.
```

```
class TableInfo {
private:
        string table_name_;  // OK - underscore at end.
        string tablename_;   // OK.
};
```

구조체는 보통 변수처럼 사용.

```
struct UrlTableProperties {
        string name;
        int num_entries;
}
```

Cf. structs vs. classes

C++에서 struct와 class 키워드는 거의 똑같이 동작.

데이터를 나르는 수동적인 객체의 경우에만 struct를 사용하며, 그 외의 모든 경우에는 class를 사용.

structs는 멤버의 값을 읽고 쓰는 것 이외의 어떤 기능도 허용하지 않음.

필드의 접근/변경은 메서드 호출이 아닌 직접 필드에 접근하는 방식으로 작성할 것.

더 많은 기능이 필요하다면 class가 적당하며 **불확실한 경우 class**로 만들 것.

Coding Convention

- Google C++ Naming(Constant Names)

  k로 시작하는 대소문자가 섞인 이름을 사용.

  ```
  const int kDaysInAWeek = 7;
  ```

  지역변수인지, 전역변수인지, 클래스의 일부인지와 상관 없이 **모든 컴파일 시점 상수들**은 다른 변수들과 조금 다른 이름 규칙을 사용. k로 시작하여 매 단어의 첫 글자를 대문자로 사용.

Coding Convention

- Google C++ Naming(Function Names)

  일반 함수들은 대소문자가 섞인 방식을 사용. accessors와 mutators는 해당하는 변수의 이름과 같은 것을 사용.

  AddTableEntry()
  DeleteUrl()

  OpenFileOrDie() crash가 발생할 수 있는 함수

  함수 이름은 대문자로 시작하여 각 단어의 **첫 글자를 대문자로 쓰고, underscore는 사용하지 않음**. 함수의 실행 중 **crash가 발생할 수 있다면 함수의 이름 뒤에 OrDie** 를 붙인다.

Coding Convention

- Google C++ Naming(Function Names cont.)

  accessors와 mutators (get 과 set 함수)는 접근 또는 변경을 하려는 **변수의 이름과 일치하는 이름**을 사용.

```
class MyClass {
public:
        ...
        int num_entries() const { return num_entries_; }
        void set_num_entries(int num_entries) { num_entries_ = num_entries; }

private:
        int num_entries_;
};
```

Coding Convention

- Google C++ naming(Macro names)

  일반적으로 **매크로는 사용하지 않는 것이 좋으며**, 절대적으로 필요하다면 대문자와 underscore로 작성.

```
#define ROUND(x) ...
#define PI_ROUNDED 3.0
```

Coding Convention

- Google C++ Header Files(The #defile Guard)

  모든 헤더 파일은 **여러 번 포함되지 않기 위해 #define 가드**를 사용. 유일성을 보장하기 위해 #define 가드는 프로젝트의 소스 트리의 절대 경로에 기반함.

  <PROJECT>_<PATH>_<FILE>_H_ 으로 작성

  예를 들어 프로젝트에 foo/src/bar/baz.h 파일이 있다면 foo는 아래와 같은 가드를 가져야 함.

```
#ifndef FOO_BAR_BAZ_H_
#define FOO_BAR_BAZ_H_

...

#endif  // FOO_BAR_BAZ_H_
```

Coding Convention

- Google C++ Header Files(Names and Order of Includes)

  가독성을 높이고 숨겨진 종속성을 피하기 위해서 일관된 순서를 사용. 모든 프로젝트의 헤더 파일은 디렉터리 단축 표시인 **. (현재 디렉터리)이나 .. (부모 디렉터리)을 사용하지 않고** 프로젝트의 소스 디렉터리의 하위 요소로 나열.

  예를 들어 google-awesome-project/src/base/logging.h는 아래와 같이 #include되어야 함.

  ```
  #include "base/logging.h"
  ```

  예를 들어 dir2/foo2.h에 있는 것들을 구현하거나 테스트하기 위한 dir/foo.cc나 dir/foo_test.cc를 작성 시 아래의 순서대로 작성할 것.

  1. **dir2 / foo2.h**
  2. **C 시스템 파일**
  3. **C++ 시스템 파일**
  4. **다른 라이브러리의.h 파일**
  5. **현재 프로젝트의.h 파일**

Coding Convention

- Google C++ Header Files(Names and Order of Includes cont.)

  예를 들면 google-awesome-project/src/foo/internal/fooserver.cc의 include들은 아래와 같이 작성할 수 있음.

```
#include "foo/public/fooserver.h"  // Appropriate location

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

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

Coding Convention

- Conditional Statement

  조건문에서 인수는 비교하고자 하는 대상을 왼쪽에 배치할 것.

  “당신의 나이” 가 **“10”** 보다 큰가?

  “10” 이 **“당신의 나이”** 보다 작은가?

```
If (received < expected)        //good
if (received > expected)        //bad
```

  부정이 아닌 긍정을, 간단한 것을, 흥미로운 것을 먼저 처리할 것.

```
if (a == b) {                   if (a != b) {
}                               }
else {                          else {
}                               }
```

Coding Convention

- Conditional Statement

  조건이 길어질 때 리팩토링해서 사용할 것.

☹ if (복잡하고길고어렵고아무튼그런조건문1 && 복잡하고길고어렵고아무튼그런조건문2 || 복잡하고길고어렵고아무튼그런조건문3)

bool disconnected = 복잡하고길고어렵고아무튼그런조건문1 && 복잡하고길고어렵고아무튼그런조건문2 || 복잡하고길고어렵고아무튼그런조건문3;

🙂 if (disconnected)

Coding Convention

- Conditional Statement

  삼항 연산자를 적절히 사용할 것.


```
time_str += (hour >= 12) ? "pm" : "am";
```


```
return exponent >= 0 ? mantissa*(1 << exponent) : mantissa / (1 << -exponent);
```


```
if (hour >= 12) {
        time_str += "pm";
}

else {
        time_str += "am";
}
```


```
if (exponent >= 0) {
        return mantissa*(1 << exponent);
}

else {
        retun mantissa / (1 << -exponent);
}
```

Coding Convention

- Comment

  생각을 기록하고 나올 것 같은 질문을 예측할 것. 또한 코드의 결함을 설명하는 것을 두려워 하지 말 것.

  TODO : 아직 하지 않은 일
  FIXME : 오작동을 일으킨다고 알려진 코드
  HACK : 아름답지 않은 해결책
  XXX : 위험한 것. 큰 문제가 있는 경우

  //TODO: 에디트 박스 숫자 이외에 입력 방지(完)
  //TODO: 에디트 박스 숫자 길이 제한.

  //TODO: 사이즈 줄인 것과 원본을 따로 관리하여, 나중에 비디오 저장이 가능하도록 함.
  //HACK: 현재 1번 영상 최적화.

Coding Convention

- Comment

  모호한 네이밍에는 주석을 달지 말고 네이밍을 수정할 것.

```
//반환하는 항목의 수나 전체 바이트 수와 같다.
//Request가 정하는 대로 Reply에 일정한 한계를 적용한다.
void CleanReply(Request request, Reply reply);
```

```
//'reply이 cont/byte/등과 같이 'request'가 정하는 한계조건을 만족시키도록 한다.
void EnforceLimitsFromRequest(Request request, Reply reply);
```

Summary

- 모든 네이밍은 약어 사용을 피하고 서술적으로 작성할 것.
- 가독성 좋게 조건문을 작성할 것.
- 코드를 수정하는 것을 최우선으로 하고, 주석은 반드시 필요한 경우에만 작성할 것.

Future Work

Paper Review

- [x] ~~Vanilla GAN~~
- [x] ~~DCGAN~~
- [ ] LS GAN
- [ ] BEGAN
- [ ] ~~Pix2Pix~~
- [ ] Cycle GAN

Proposed Model

- [ ] SpyGAN(about depth)

Tools

- [x] ~~Document~~
- [x] ~~Programming~~
- [x] ~~PyTorch~~
- [ ] Python executable & UI

Mathematical Study

- [ ] Linear algebra
- [ ] Probability and statistics
- [ ] Information theory

Others

- [x] ~~Level Processor~~
- [x] ~~Coding Standard~~
- [ ] Ice Propagation

Reference

[1] 포프 C++ Coding Standards

(https://docs.google.com/document/d/1cT8EPgMXe0eopeHvwuFmbHG4TJr5kUmcovkr5irQZmo/edit#heading=h.r2n9mhxbh2gg)

[2] Google C++ Style Guide (Original)

(https://google.github.io/styleguide/cppguide.html)

[3] Google C++ Style Guide (Translated)

(http://jongwook.kim/google-styleguide/trunk/cppguide.xml)

[4] 읽기 좋은 코드가 좋은 코드다

(https://www.slideshare.net/e2goon/ss-33769330)

[5] [코딩원칙?] if문. 그외 가독성을 올리자.

(https://blog.naver.com/soguns/120139779253)

[6] 무조건 if for 문에 {}를 써야하는가.....

(http://www.gamecodi.com/board/zboard.php?id=GAMECODI_Talkdev&no=3727)

[7] [읽기 좋은 자바스크립트 코딩 기법] 문장과 표현식(조건문과 반복문)

(https://jojoldu.tistory.com/6)

Q & A

Thank you for your attention

Appendix

- {} 써야하는가? 붙여야 하는가?

```
if (condition) {
        doSomething();
}
else {
        doSomethingElse();
}

if (condition)
{
        doSomething();
}
else
{
        doSomethingElse();
}
```

경험상으로 { } 가 엉켜서 컴파일 에러나 런타임 에러가 발생해 시간을 낭비.

IDE가 자동으로 들여쓰기해주는 간단한 if for에서 에러를 낸 적은 없음.

안정성이라면 세계 제일이라는 NASA의 표준 코드 Style

ASI