

인트아이 C++ 심화 스터디

C P P C H E C K , C L A N G F O R M A T 설치 및 사용법

C + + C L A S S

스터디 소개

스터디 일정		매주 월/목 오후 3시~5시
1	1월 16일(월)	스터디 소개 Cppcheck, ClangFormat 설치 및 사용법 C++ Class
	1월 19일(목)	C++ RAI / Smart Pointer와 Guard 패턴
2	1월 23일(월)	C++ Functor, Lambda, Closure
	1월 26일(목)	C++ 동시성 제어 (Thread, Mutex, Semaphore)
3	1월 30일(월)	C++ Template
	2월 2일(목)	C++ SFINAE, Concepts 기초
4	2월 6일(월)	Vcpkg 설치 및 사용법 Boost 라이브러리 기초 (Asio, Beast)
	2월 9일(목)	스터디 마무리



참고 자료

- C++ Core Guidelines: C++ 창시자 '비야네 스트롭스트롭'이 작성한 성서
<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>
한국어 번역본: <http://cppkorea.github.io/CppCoreGuidelines/> (번역이 비어 있는 곳은
원서 참고)
- cppreference.com: C++ 함수 사용법 등 찾아보기 좋음

코드를 잘 짜는 방법???

- 디자인 패턴, 설계 이론 공부 (도메인 주도 설계 등)
- 남이 만든 프로그램 코드 읽어보기 (오픈소스 분석)
- 각 언어의 특색 있는 기능을 적극적으로 이용하기
- 코딩 도구의 도움을 받기
- ...



Formatter와 Linter

- Formatter: 코딩 스타일 (들여쓰기 방식, 괄호 위치 등)을 통일 시켜주는 도구
- Linter: 코딩 스타일과 잠재적 오류를 모두 잡아주는 도구

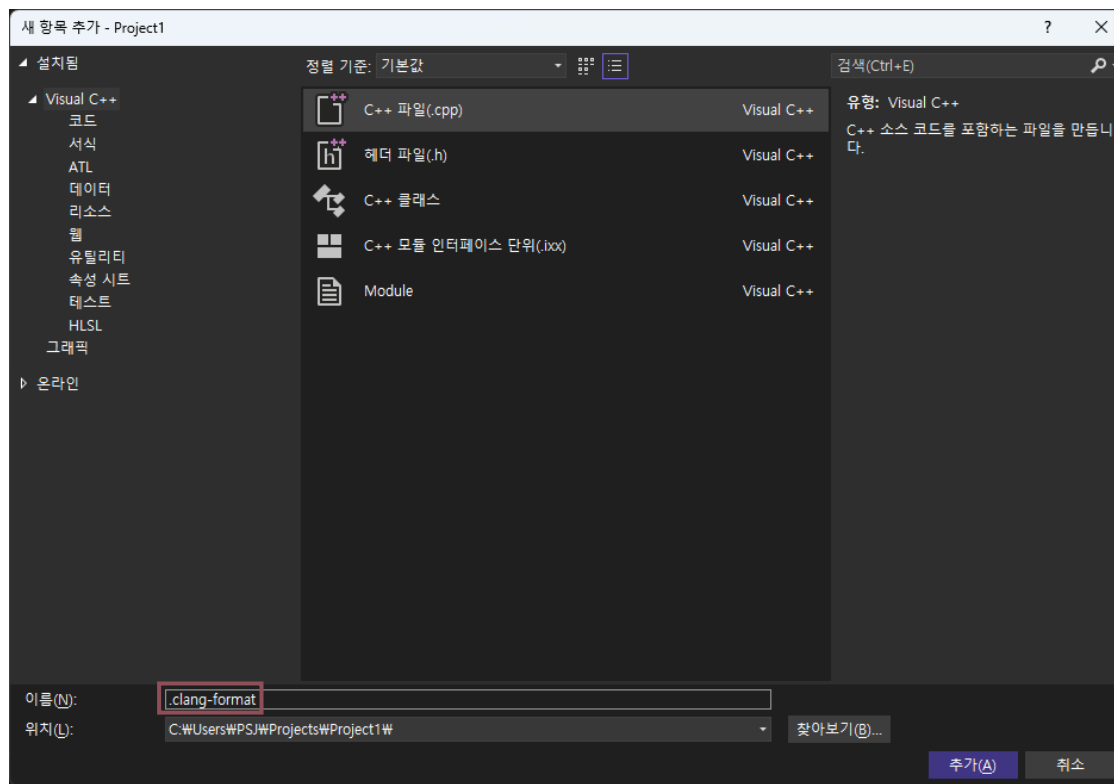
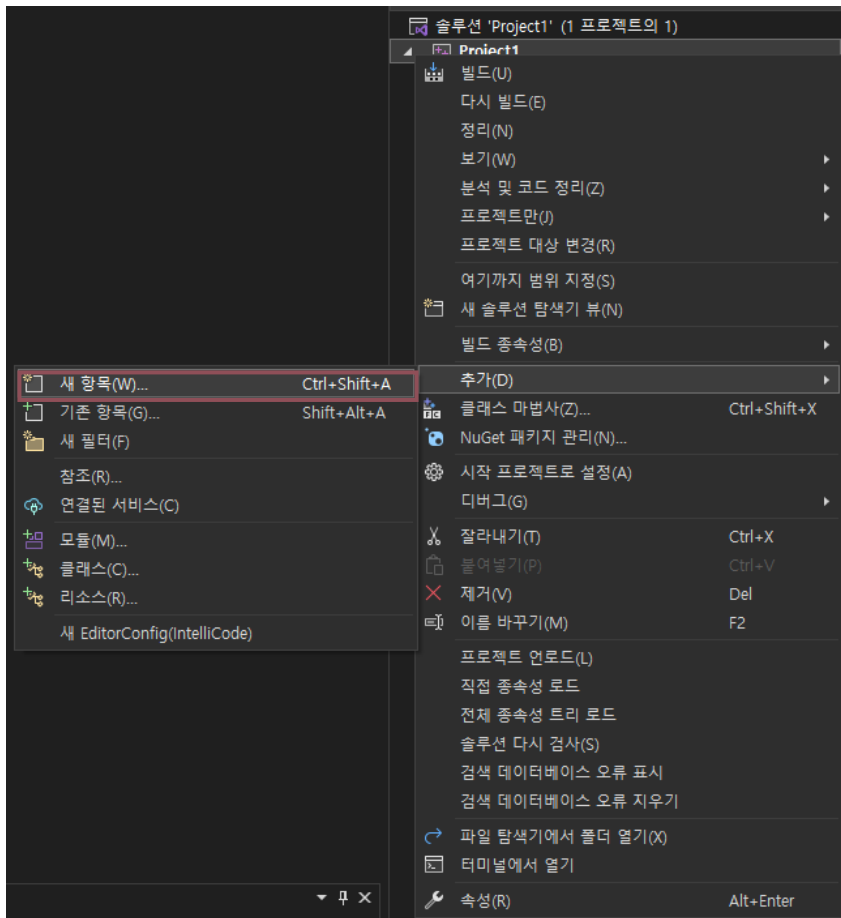
C++에서는 ClangFormat과 Cppcheck를 사용

프로그램 버그보다는 컴파일 오류가 나옴

버그를 만들 가능성 있는 안 좋은 코드를 컴파일 때 다 잡아버리면,
프로그램 동작 중에 버그가 발생할 확률이 낮아짐

ClangFormat

Visual Studio에 내장되어 있어 별도의 설치는 필요 없고
설정파일만 만들면 됨



<https://gist.github.com/Astro36/211443b279e964af9b2ed05ddec0b1ee>

ClangFormat

```
1 #include <algorithm>
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5
6 int A[1000];
7 int psum[1000];
8
9 int main()
10 {
11     int n, q;
12     cin >> n >> q;
13     for (int i = 1; i <= n; i++)
14     {
15         cin >> A[i];
16     }
17     sort(A, A + n);
18
19     for (int i = 1; i <= n; i++)
20     {
21         psum[i] = psum[i - 1] + A[i];
22     }
23
24     for (int i = 0; i < q; i++)
25     {
26         int L, R;
27         cin >> L >> R;
28         cout << psum[R] - psum[L - 1] << "\n";
29     }
30 }
```

Ctrl+K Ctrl+D



```
1 #include <algorithm>
2 #include <iostream>
3 #include <vector>
4 using namespace std;
5
6 int A[1000];
7 int psum[1000];
8
9 int main() {
10     int n, q;
11     cin >> n >> q;
12     for (int i = 1; i <= n; i++) {
13         cin >> A[i];
14     }
15     sort(A, A + n);
16
17     for (int i = 1; i <= n; i++) {
18         psum[i] = psum[i - 1] + A[i];
19     }
20
21     for (int i = 0; i < q; i++) {
22         int L, R;
23         cin >> L >> R;
24         cout << psum[R] - psum[L - 1] << "\n";
25     }
26 }
27
```


Cppcheck

<https://cppcheck.sourceforge.io/>

Cppcheck
A tool for static C/C++ code analysis

Home | Wiki | Forum | Issues | Developer Info | Online Demo | Project page

[Download](#) | [Features](#) | [News](#) | [Documentation](#) | [Support](#) | [Contribute](#)

Cppcheck is a [static analysis tool](#) for C/C++ code. It provides [unique code analysis](#) to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to have very few false positives. Cppcheck is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects).

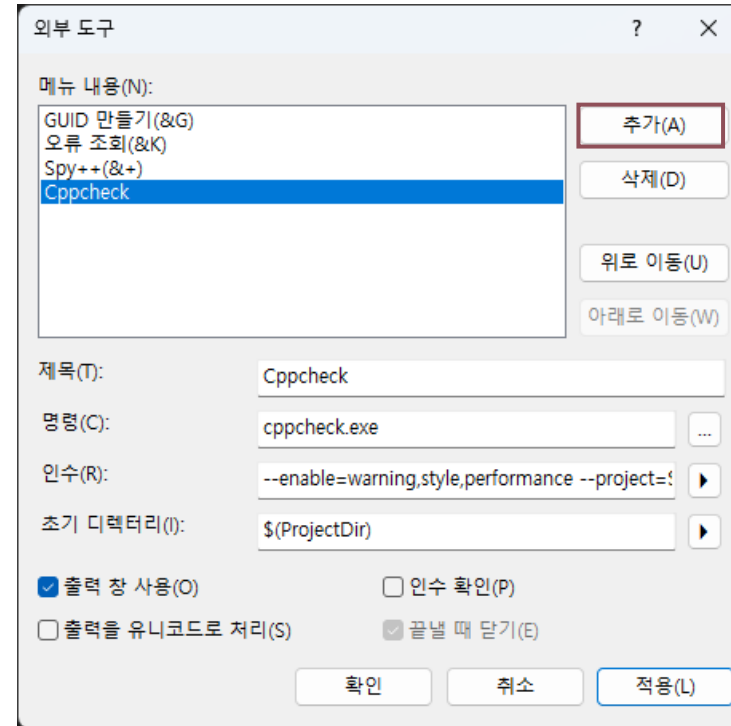
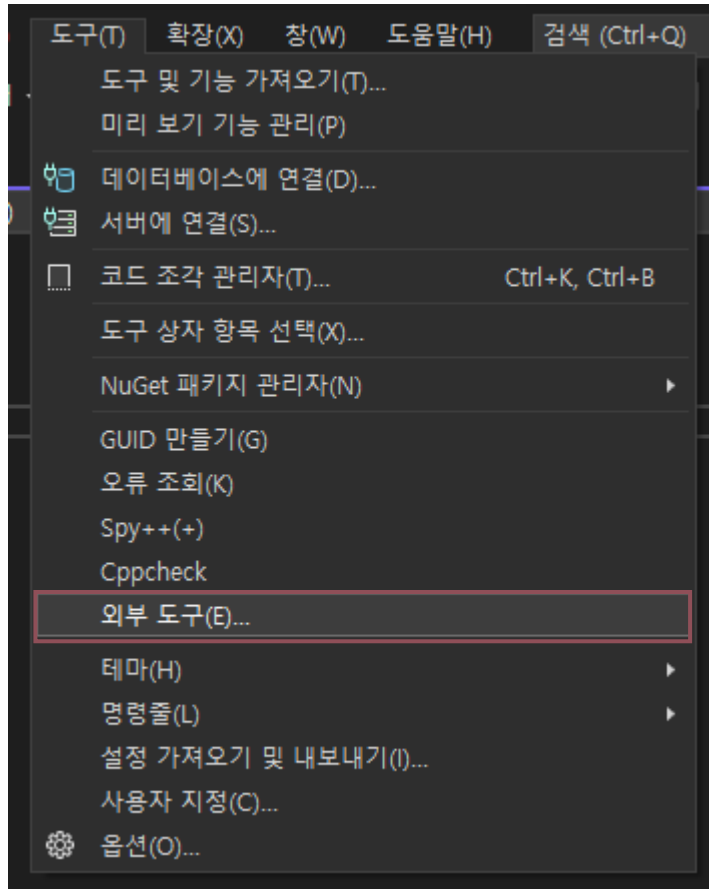
Cppcheck is available both as open-source (this page) and as **Cppcheck Premium** with extended functionality and support. Please visit www.cppchecksolutions.com for more information and purchase options for the commercial version.

Download

Cppcheck 2.9 (open source)

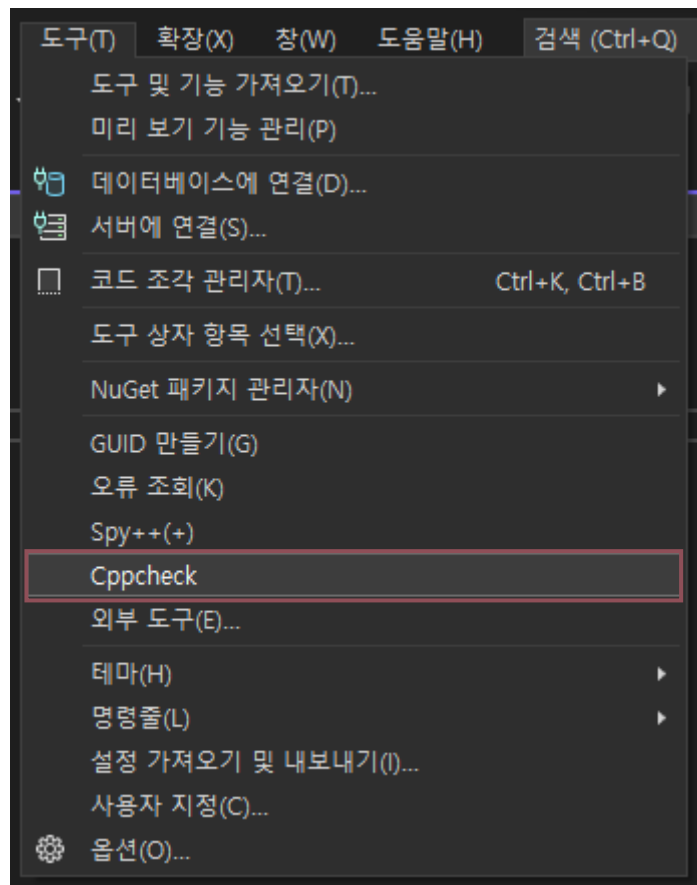
Platform	File
Windows 64-bit (No XP support)	Installer
Source code (.zip)	Archive
Source code (.tar.gz)	Archive

Cppcheck



--enable=warning,style,performance --project=\$(ProjectFileName) -q --template=vs

Cppcheck



```
1 int main() {  
2     int *i = new int;  
3     delete i;  
4     return 0;  
5 }
```

출력 보기 선택(S): Cppcheck

```
ss.cpp(4): error: Memory is allocated but not initialized: i  
ss.cpp(2): style: Variable 'i' is allocated memory that is never used.
```

파일이 없다고 나오면 Ctrl+5로 코드 실행 한 번하고 돌리기

```

1  #include <iostream>
2
3  class Bird {
4  private:
5      int age;
6
7  public:
8      Bird(int age)
9      : age(age) {}
10     virtual ~Bird() = default; 중요
11
12     virtual void fly() const final {
13         // TODO
14     }
15
16     virtual void sing() const {
17         // TODO
18     }
19 };
20
21 class Duck : public Bird {
22 private:
23     int age;
24
25 public:
26     Duck(int age)
27     : Bird(age) {}
28     virtual ~Duck() = default;
29
30     virtual void sing() const override {
31         // TODO
32     }
33 };
34
35 int main() {
36     return 0;
37 }
38

```

C++ class

- 메소드 뒤에 붙은 const는 클래스 객체 속성을 건드리지 않는 함수임을 의미
- virtual은 동적 바인딩되는 메소드임을 의미 (동적 바인딩은 성능 손실 있음, 남용x)
- final은 더 이상 override 불가함을 의미
- override는 메소드가 오버라이드 규칙을 지키는지 컴파일 시간에 검사함

출력

출력 보기 선택(S): Cppcheck

```

ss.cpp(26): warning: Member variable 'Duck::age' is not initialized in the constructor.
ss.cpp(8): style: Class 'Bird' has a constructor with 1 argument that is not explicit.
ss.cpp(26): style: Class 'Duck' has a constructor with 1 argument that is not explicit.

```

C.128: 가상 함수들은 `virtual`, `override`, 혹은 `final` 중 하나만 명시해야 한다

REASON

가독성. 실수를 발견할 수 있다. 명시적으로 `virtual`, `override`, `final` 을 사용하는 것은 함수 자체를 문서화한다. 동시에 컴파일러가 상위 클래스와 하위 클래스의 타입 혹은 이름이 불일치 하는 것을 잡아 낼 수 있도록 돕는다. 하지만 이들을 하나 이상 작성하는 것은 중복적이면서 오류를 발생시킬 수 있다.

하나만 작성하는 것이 단순하고 명확하다:

- `virtual` 는 "새로운 가상 함수"라는 것을 의미한다
- `override` 는 "재정의 될 수 있는(non-`final`) 재정의 함수"라는 것을 의미한다
- `final` 는 "마지막 재정의 함수"라는 것을 의미한다



virtual과 override

`friend` 키워드를 이용하면 외부에서도 `private`, `protected` 로 선언된 멤버변수에 접근할 수 있습니다. 이는 클래스의 캡슐화를 파괴하지만, 연산자를 오버로딩할 때 유용하게 사용됩니다.

```
#include <iostream>

class Book {
    friend std::ostream& operator<<(std::ostream& out, const Book& book);

private:
    std::string title;

public:
    // TODO
};

std::ostream& operator<<(std::ostream& out, const Book& book) {
    out << book.title;
    return out;
}
```

friend 함수

초기화 리스트를 사용해야만 하는 경우

1. 멤버변수가 상수일 때:

```
class A {  
private:  
    const int m;  
  
public:  
    A(int n): m(n) {}  
};
```

만약 `n` 이 상수 리터럴이라면 아래와 같이 쓸 수 있습니다.

```
class A {  
private:  
    const int m = 10;  
  
public:  
    A() {}  
};
```

2. 멤버변수가 참조일 때:

```
class A {  
private:  
    const std::string& m;  
  
public:  
    A(const std::string& n): m(n) {}  
};
```

3. has-a 상속에서 내장 클래스를 초기화시킬 때:

```
class A {  
private:  
    B b;  
  
public:  
    A(): b(1, 2) {}  
};
```

4. is-a 상속에서 부모 클래스를 초기화시킬 때:

```
class B : public A {  
public:  
    B(): A(1, 2) {}  
};
```

```
class A {  
private:  
    int m;  
  
public:  
    A(int n) { // 생성자를 호출하기 전, `m`에 0 할당  
        m = n; // `m`에 `n` 할당  
    }  
};
```

위를 초기화 리스트를 사용해 바꾼다면 한 번의 할당만으로 객체를 초기화할 수 있습니다.

```
class A {  
private:  
    int m;  
  
public:  
    A(int n): m(n) {} // 생성자를 호출하기 전, `m`에 `n` 할당  
};
```

초기화리스트를 이용한 성능 개선

C.46: 단일 인자를 사용하는 생성자는 explicit으로 선언하라

REASON

의도치 않은 변환을 피한다.

EXAMPLE, BAD

```
class String {  
    // ...  
public:  
    explicit String(int); // BAD  
    // ...  
};  
  
String s = 10; // 놀랍게도... 길이가 10인 문자열이 된다
```

explicit 생성자

C.133: protected 데이터를 지양하라

REASON

`protected` 데이터는 복잡성과 에러의 원인이다.
`protected` 데이터는 불변조건의 구문을 복잡하게 만든다.
`protected` 데이터는 상위 클래스에 데이터를 배치함으로써 필연적으로 가상 상속을 처리해야 하는 상황으로 이어질 수 있다.

EXAMPLE, BAD

```
class Shape {  
public:  
    // ... interface functions ...  
protected:  
    // data for use in derived classes:  
    Color fill_color;  
    Color edge_color;  
    Style st;  
};
```

이 예에서 모든 `Shape`의 하위 타입들은 `protected` 데이터를 정확하게 변경해야만 한다. 흔히 볼 수 있으면서 유지보수 문제를 일으키는 주요 원인 중 하나에 해당한다. 클래스 계층구조가 큰 경우, 일관적으로 `protected` 데이터를 사용하는 것은 코드가 양적으로 많고 분산되어 있기 때문에 관리되기 어렵다. 상속되는 데이터를 변경할 수 있는 클래스는 더 늘어날 수 있다: 새로 클래스를 상속받아 `protected` 데이터를 변경하기 시작할 수 있다. 경우에 따라선 클래스들의 전체 집합을 찾는 것이 불가능할 수도 있다. 이로 인해 클래스를 변경하는 것을 실행할 수 없을 수도 있다. `protected` 데이터에는 불변조건을 강요할 수 없다; 전역변수 집합과 같다고 할 수 있다. `protected` 데이터는 코드 규모가 커지면 실제로 전역변수가 된다.

protected는 봉인

상속은 강력하지만 남용하면 안 됨

- C++에서 프로그램 실행 중에 자동으로 돌아가는 기능은 필연적으로 성능 손실을 유발함
- virtual로 정의된 메소드와 소멸자는 virtual을 사용하지 않았을 때보다 느리게 동작
- 상속으로 인해 각 클래스의 의미가 이상해지는 경우도 존재
- 상속 계층이 3단 이상이면 일단 정지! 더 나은 방법이 있는지 생각해보기
- 상속을 통해 코드를 효과적으로 재사용할 수 있는 곳에서만 상속을 이용
- 그 밖의 경우는, Composition(has-a 상속), 인터페이스(Pure virtual class), 템플릿 메타 프로그래밍(TMP) 고려
- 읽어보면 좋은 글: <https://int-i.github.io/java/2022-09-25/oop/>

Bitcoin C++코드

C++스러운 코드는 struct와 class를 이용해 함수간
데이터를 주고 받음 (상속은 그냥 보조해주는 역할)

<https://github.com/bitcoin/bitcoin/blob/master/src/node/coin.cpp>

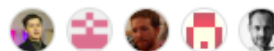
master

bitcoin / src / node / coin.cpp



ryanofsky Add src/node/* code to node:: namespace

5 contributors



26 lines (23 sloc) | 869 Bytes

```
1 // Copyright (c) 2019-2021 The Bitcoin Core developers
2 // Distributed under the MIT software license, see the accompanying
3 // file COPYING or http://www.opensource.org/licenses/mit-license.php.
4
5 #include <node/coin.h>
6
7 #include <node/context.h>
8 #include <txmempool.h>
9 #include <validation.h>
10
11 namespace node {
12 void FindCoins(const NodeContext& node, std::map<COutPoint, Coin>& coins)
13 {
14     assert(node.mempool);
15     assert(node.chainman);
16     LOCK2(cs_main, node.mempool->cs);
17     CCoinsViewCache& chain_view = node.chainman->ActiveChainstate().CoinsTip();
18     CCoinsViewMemPool mempool_view(&chain_view, *node.mempool);
19     for (auto& coin : coins) {
20         if (!mempool_view.GetCoin(coin.first, coin.second)) {
21             // Either the coin is not in the CCoinsViewCache or is spent. Clear it.
22             coin.second.Clear();
23         }
24     }
25 }
26 } // namespace node
```

```

5 #ifndef BITCOIN_NODE_CHAINSTATE_H
6 #define BITCOIN_NODE_CHAINSTATE_H
7
8 #include <util/translation.h>
9 #include <validation.h>
10
11 #include <cstdint>
12 #include <functional>
13 #include <tuple>
14
15 class CTxMemPool;
16
17 namespace node {
18
19 struct CacheSizes;
20
21 struct ChainstateLoadOptions {
22     CTxMemPool* mempool{nullptr};
23     bool block_tree_db_in_memory{false};
24     bool coins_db_in_memory{false};
25     bool reindex{false};
26     bool reindex_chainstate{false};
27     bool prune{false};
28     int64_t check_blocks{DEFAULT_CHECKBLOCKS};
29     int64_t check_level{DEFAULT_CHECKLEVEL};
30     std::function<bool()> check_interrupt;
31     std::function<void()> coins_error_cb;
32 };
33
34 /// Chainstate load status. Simple applications can just check for the success
35 /// case, and treat other cases as errors. More complex applications may want to
36 /// try reindexing in the generic failure case, and pass an interrupt callback
37 /// and exit cleanly in the interrupted case.
38 enum class ChainstateLoadStatus { SUCCESS, FAILURE, FAILURE_INCOMPATIBLE_DB, INTERRUPTED };
39
40 /// Chainstate load status code and optional error string.
41 using ChainstateLoadResult = std::tuple<ChainstateLoadStatus, bilingual_str>;
42
43 /** This sequence can have 4 types of outcomes:
44  *
45  * 1. Success
46  * 2. Shutdown requested
47  * - nothing failed but a shutdown was triggered in the middle of the
48  * sequence
49  * 3. Soft failure
50  * - a failure that might be recovered from with a reindex
51  * 4. Hard failure
52  * - a failure that definitively cannot be recovered from with a reindex
53  *
54  * LoadChainstate returns a (status code, error string) tuple.
55  */
56 ChainstateLoadResult LoadChainstate(ChainstateManager& chainman, const CacheSizes& cache_sizes,
57                                     const ChainstateLoadOptions& options);
58 ChainstateLoadResult VerifyLoadedChainstate(ChainstateManager& chainman, const ChainstateLoadOptions& options);
59 } // namespace node

```

```

8 #include <policy/feerate.h>
9 #include <primitives/transaction.h>
10 #include <util/error.h>
11
12 class CBlockIndex;
13 class CTxMemPool;
14 namespace Consensus {
15 struct Params;
16 }
17
18 namespace node {
19 struct NodeContext;
20
21 /** Maximum fee rate for sendrawtransaction and testmempoolaccept RPC calls.
22  * Also used by the GUI when broadcasting a completed PSBT.
23  * By default, a transaction with a fee rate higher than this will be rejected
24  * by these RPCs and the GUI. This can be overridden with the maxfeerate argument.
25  */
26 static const CFeeRate DEFAULT_MAX_RAW_TX_FEE_RATE{COIN / 10};
27
28 /**
29  * Submit a transaction to the mempool and (optionally) relay it to all P2P peers.
30  *
31  * Mempool submission can be synchronous (will await mempool entry notification
32  * over the CValidationInterface) or asynchronous (will submit and not wait for
33  * notification), depending on the value of wait_callback. wait_callback MUST
34  * NOT be set while cs_main, cs_mempool or cs_wallet are held to avoid
35  * deadlock.
36  *
37  * @param[in] node reference to node context
38  * @param[in] tx the transaction to broadcast
39  * @param[out] err_string reference to std::string to fill with error string if available
40  * @param[in] max_tx_fee reject txs with fees higher than this (if 0, accept any fee)
41  * @param[in] relay flag if both mempool insertion and p2p relay are requested
42  * @param[in] wait_callback wait until callbacks have been processed to avoid stale result due to a sequentially RPC.
43  * return error
44  */
45 [[nodiscard]] TransactionError BroadcastTransaction(NodeContext& node, CTransactionRef tx, std::string& err_string, const CAmount& max_tx_fee, bool relay, bool
46 wait_callback);
47
48 /**
49  * Return transaction with a given hash.
50  * If mempool is provided and block_index is not provided, check it first for the tx.
51  * If -txindex is available, check it next for the tx.
52  * Finally, if block_index is provided, check for tx by reading entire block from disk.
53  *
54  * @param[in] block_index The block to read from disk, or nullptr
55  * @param[in] mempool If provided, check mempool for tx
56  * @param[in] hash The txid
57  * @param[in] consensusParams The params
58  * @param[out] hashBlock The block hash, if the tx was found via -txindex or block_index
59  * @returns The tx if found, otherwise nullptr
60  */
61 CTransactionRef GetTransaction(const CBlockIndex* const block_index, const CTxMemPool* const mempool, const uint256& hash, const Consensus::Params& consensusParams, uint256&
62 hashBlock);
63 } // namespace node

```

struct를 이용한 코드 개선

```
void print_user(std::string name, int age, std::string gender) {  
    std::cout << name << " " << age << " " << gender;  
}
```

```
enum Gender {  
    UNKNOWN,  
    MALE,  
    FEMALE  
};  
  
struct User {  
    std::string name;  
    int age;  
    Gender gender;  
};  
  
void print_user_better(const User& user) {  
    std::cout << user.name << " " << user.age << " " << user.gender;  
}
```

Quiz

오른쪽 헤더파일을 참고하여
프로그램을 완성하시오.
(Cppcheck를 이용할 것)

헤더 파일에 문제가 있다면
헤더파일을 고치시오.

quiz.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>

class Person {
public:
    explicit Person(const std::string &name);
    virtual ~Person() = default;
    virtual const std::string &get_name() const;

private:
    std::string name;
};

class Professor : public Person {
public:
    Professor(const std::string &name, const std::string &department_name);
    const std::string &get_department_name() const;

private:
    std::string department_name; // 소속 학과
};

class Student : public Person {
public:
    Student(const std::string &name, int grade);
    int get_grade() const;

private:
    int grade; // 1학년~4학년
};

class Lecture {
public:
    Lecture(const std::string &name, const Professor &professor, const std::vector<Student> &students);
    void add_student(const Student &student);
    const std::string &get_name() const;
    const Professor &get_professor() const;
    const std::vector<Student> &get_students() const;

private:
    std::string name;
    Professor professor;
    std::vector<Student> students;
};
```

main.c

```
#include <iostream>
#include "quiz.h"

int main() {
    Student kim("김**", 100);
    Student lee("이**", 90);
    Student park("박**", 80);
    Professor choi("최**", "정보통신공학과");

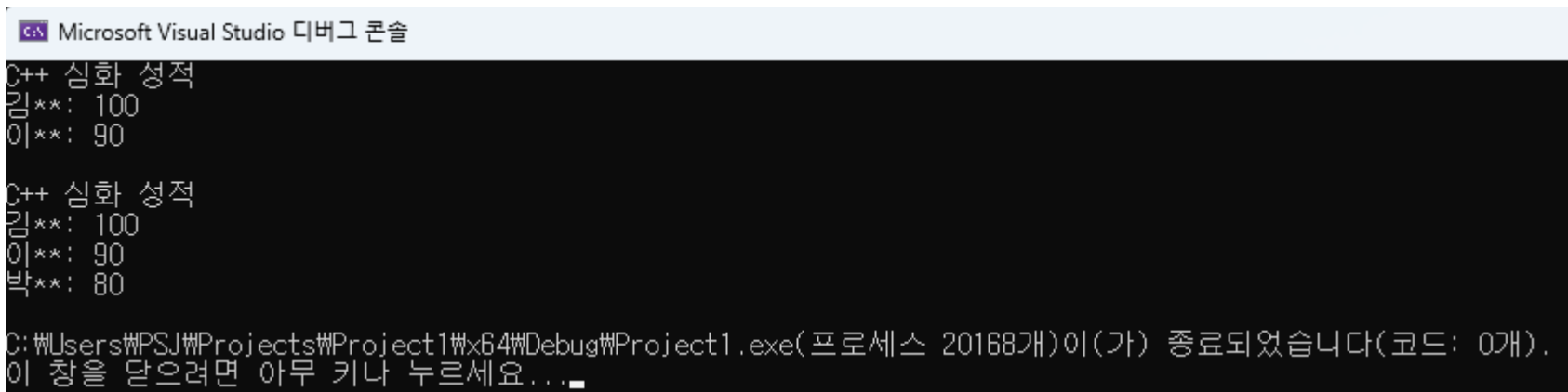
    Lecture cpp("C++ 심화", choi, { kim, lee });

    std::cout << cpp.get_name() << " 성적\n";
    for (const Student &student : cpp.get_students()) {
        std::cout << student.get_name() << ": " << student.get_grade() << '\n';
    }

    cpp.add_student(park); ← push_back 이용 (https://en.cppreference.com/w/cpp/container/vector/push\_back)
    std::cout << '\n';

    std::cout << cpp.get_name() << " 성적\n";
    for (const Student &student : cpp.get_students()) {
        std::cout << student.get_name() << ": " << student.get_grade() << '\n';
    }

    return 0;
}
```



```
Microsoft Visual Studio 디버그 콘솔

C++ 심화 성적
김 **: 100
이 **: 90

C++ 심화 성적
김 **: 100
이 **: 90
박 **: 80

C:\Users\PSJ\Projects\Project1\Debug\Project1.exe(프로세스 20168개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```


Hint

다 풀기 전까지는 되도록이면 페이지를 넘기지 말 것



quiz.cpp

```
#include "quiz.h"

Person::Person(const std::string &name)
: name(name) {

const std::string &Person::get_name() const {
    return name;
}

Professor::Professor(const std::string &name, const std::string &department_name)
: Person(name), department_name(department_name) {

const std::string &Professor::get_department_name() const {
    return department_name;
}

//

int Student::get_grade() const {
    //
}

Lecture::Lecture(const std::string &name, const Professor &professor, const std::vector<Student> &students)
: name(name), professor(professor), students(students) {

void Lecture::add_student(const Student &student) {
    //
}

const std::string &Lecture::get_name() const {
    //
}

const Professor &Lecture::get_professor() const {
    //
}

const std::vector<Student> &Lecture::get_students() const {
    return students;
}
```