



고급프로그래밍

예외처리와 C 사용

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory
Kwangwoon University
(02-940-5625, mhjeong@kw.ac.kr)

Review: 예제 - 스트림 상태 검사

```
#include <iostream>
#include <fstream>
using namespace std;

void showStreamState(ios& stream) {
    cout << "eof() " << stream.eof() << endl;
    cout << "fail() " << stream.fail() << endl;
    cout << "bad() " << stream.bad() << endl;
    cout << "good() " << stream.good() << endl;
}

int main() {
    const char* noExistFile = "c:\\Wtemp\\Wnoexist.txt"; // 존재하지 않는 파일명
    const char* existFile = "c:\\Wtemp\\Wstudent.txt"; // 존재하는 파일명

    ifstream fin(noExistFile); // 존재하지 않는 파일 열기
    if(!fin) { // 열기 실패 검사
        cout << noExistFile << " 열기 오류" << endl;
        showStreamState(fin); // 스트림 상태 출력
    }

    cout << existFile << " 파일 열기" << endl;
    fin.open(existFile);
    showStreamState(fin); // 스트림 상태 출력

    // 스트림을 끝까지 읽고 화면에 출력
    int c;
    while((c=fin.get()) != EOF)
        cout.put((char)c);

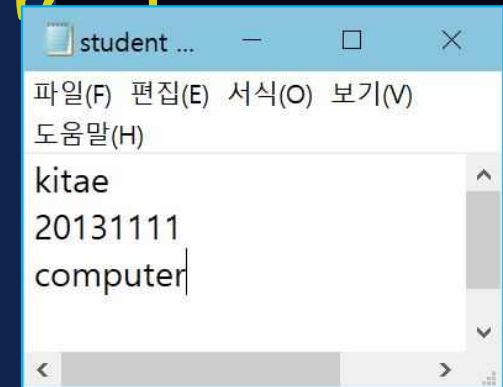
    cout << endl;
    showStreamState(fin); // 스트림 출력

    fin.close();
}
```

존재하지 않는 파일을 열 때, 스트림의 상태가 어떻게 변하는지 알기 위한 시도

정상적인 파일을 열 때, 스트림의 상태가 어떠한지 보기 위한 시도

EOF를 만났을 때, 스트림 상태 출력



c:\\Wtemp\\Wnoexist.txt 열기 오류

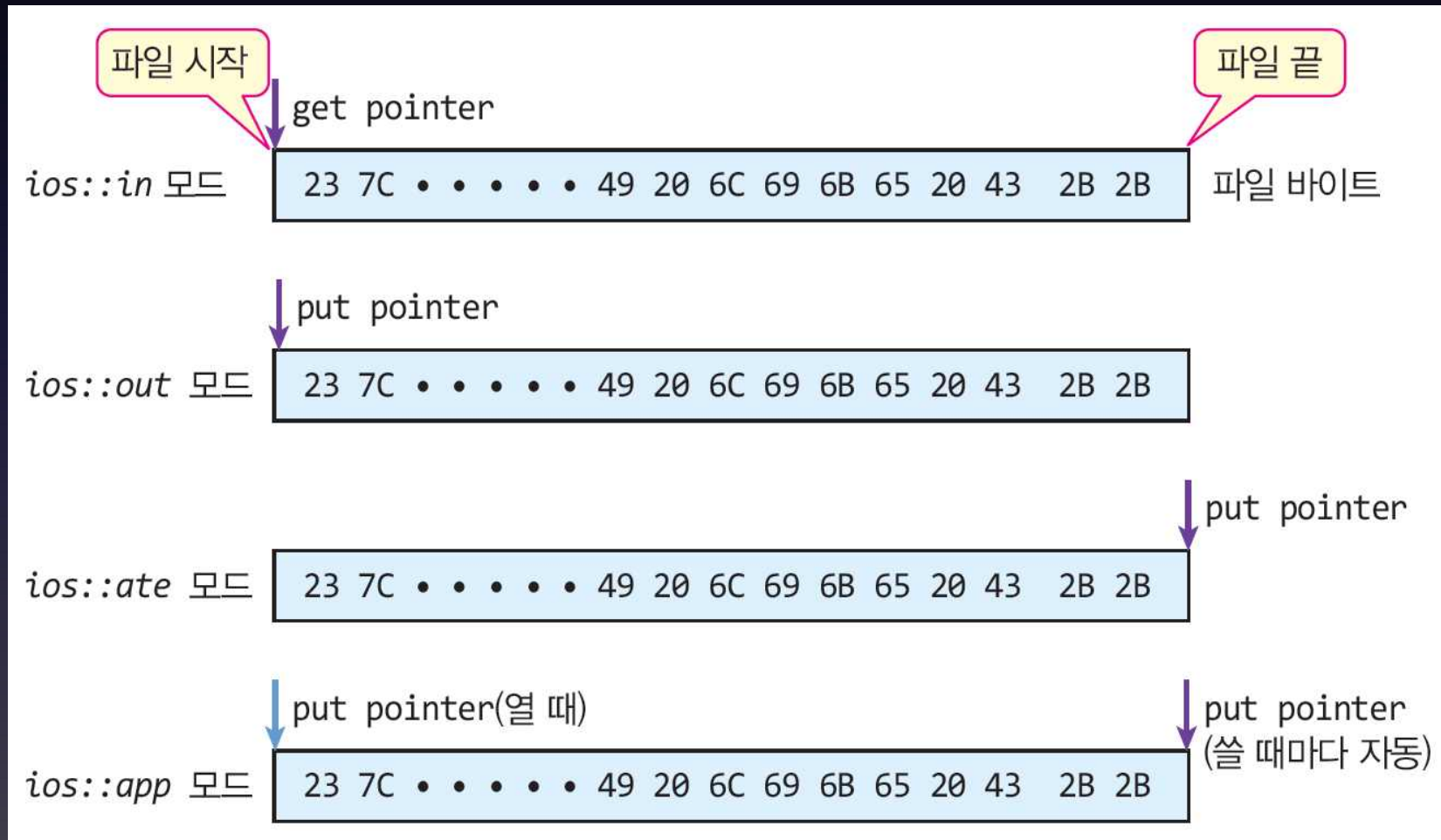
eof() 0
fail() 1
bad() 0
good() 0

c:\\Wtemp\\Wstudent.txt 파일 열기

eof() 0
fail() 0
bad() 0
good() 1
kitae
20131111
computer

eof() 1
fail() 1
bad() 0
good() 0

Review: 파일 모드와 파일 포인터



Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	4
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		5
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력(14), 예외처리 및 C 사용(15)	6	6
10		람다식 , 멀티스레딩		
11		멀티스레딩, 고급문법		
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

오늘의 학습내용

- 예외처리
- C 사용

예외처리

실행 오류의 종류와 원인

■ 오류

- 컴파일 오류

- 문법에 맞지 않는 구문으로 인한 오류

- 실행 오류

- 개발자의 논리 오류
- 예외적으로 발생하는 입력이나 상황에 대한 대처가 없을 때 발생하는 오류
- 실행 오류의 결과
 - 결과가 틀리거나 엉뚱한 코드 실행, 프로그램이 비정상 종료

예제 - 실행 오류

- 밑수와 지수부를 매개 변수로 지수 값을 계산하는 함수 작성

```
#include <iostream>
using namespace std;

int getExp(int base, int exp) { // base의 exp 지수승을 계산하여 리턴
    int value=1;
    for(int n=0; n<exp; n++)
        value = value * base; // base를 exp번 곱하여 지수 값 계산
    return value;
}

int main() {
    int v= getExp(2, 3); // 2의 3승 = 8
    cout << "2의 3승은 " << v << "입니다." << endl;

    int e = getExp(2, -3); // 2의 -3승은 ?
    cout << "2의 -3승은 " << e << "입니다." << endl;
}
```

예상치 못한 음수 입력에
대한 대처가 없음

2의 3승은 8입니다.
2의 -3승은 1입니다.

오답!
2³은 -1이 아니라
1/8

예제 - 오류 처리

```
int getExp(int base, int exp) {
    if(base <= 0 || exp <= 0) {
        return -1; // 오류 리턴
    }
    int value=1;
    for(int n=0; n<exp; n++)
        value = value * base;
    return value;
}

int main() {
    int v=0;
    v = getExp(2, 3);
    if(v != -1)
        cout << "2의 3승은 " << v << "입니다." << endl;
    else
        cout << "오류. 2의 3승은 " << "계산할 수 없습니다." << endl;

    int e=0;
    e = getExp(2, -3); // 2의 -3 승 ? getExp()는 false 리턴
    if(e != -1)
        cout << "2의 -3승은 " << e << "입니다." << endl;
    else
        cout << "오류. 2의 -3승은 " << "계산할 수 없습니다." << endl;
}
```

2의 3승은 8입니다.
오류. 2의 -3승은 계산할 수 없습니다.

예제 - 오류 처리

```
bool getExp(int base, int exp, int &ret) { //  $base^{exp}$  값을 계산하여 ret에 저장
    if(base <= 0 || exp <= 0) {
        return false;
    }
    int value=1;
    for(int n=0; n<exp; n++)
        value = value * base;
    ret = value;
    return true;
}

int main() {
    int v=0;
    if(getExp(2, 3, v)) //  $v = 2^3 = 8$ . getExp()는 true 리턴
        cout << "2의 3승은 " << v << "입니다." << endl;
    else
        cout << " 오류. 2의 3승은 " << "계산할 수 없습니다." << endl;

    int e=0;
    if(getExp(2, -3, e)) //  $2^{-3}$ ? getExp()는 false 리턴
        cout << "2의 -3승은 " << e << "입니다." << endl;
    else
        cout << " 오류. 2의 -3승은 " << "계산할 수 없습니다." << endl;
}
```

음수 입력에 대한 대처
있음

getExp()의 리턴 값의
단일화
- 오류 상태만 표시

참조 매개 변수를 통해
계산 값을 전달하는 정
리된 코드!

2의 3승은 8입니다.
오류. 2의 -3승은 계산할 수 없습니다.

예외

■ 예외란?

- 실행 중, 프로그램 오동작이나 결과에 영향을 미치는 예상치 못한 상황 발생
 - 예) getExp() 함수에 예상치 못하게 사용자가 음수를 입력하여 2^{-3} 을 1로 계산한 경우

■ 예외 처리기

- 예외 발생을 탐지하고 예외를 처리하는 코드
 - 잘못된 결과, 비정상적인 실행, 시스템에 의한 강제 종료를 막음

■ 예외 처리 수준

- 운영체제 수준 예외 처리
 - 운영체제가 예외의 발생을 탐지하여 응용프로그램에게 알려주어 예외에 대처하게 하는 방식
 - 운영체제마다 서로 다르므로, 운영체제나 컴파일러 별로 예외 처리 라이브러리로 작성
 - Java 경우, JVM 혹은 라이브러리에서 탐지한 예외를 자바응용프로그램에게 전달
 - 윈도우 운영체제는 탐지한 예외를 C/C++ 응용프로그램에게 알려줌
 - 운영체제와 컴파일러 의존적인 C++ 문법 사용
- 응용프로그램 수준 예외 처리
 - 사용자의 잘못된 입력이나 없는 파일을 여는 등 응용프로그램 수준에서 발생하는 예외를 자체적으로 탐지하고 처리하는 방법
 - C++ 예외 처리

■ C++ 예외 처리

- C++ 표준의 예외 처리
- 응용프로그램 수준 예외 처리
- 이 책에서 다루는 내용

C++ 예외 처리 기본 형식

■ try-throw-catch

- try { } 블록
 - 예외가 발생할 가능성이 있는 코드를 묶음
- throw 문
 - 발견된 예외를 처리하기 위해, 예외 발생을 알리는 문장
 - try { } 블록 내에서 이루어져야 함
- catch() { } 블록
 - throw에 의해 발생한 예외를 처리하는 코드

```
try { // 예외가 발생할 가능성이 있는 실행문. try { } 블록
.....
예외를 발견한다면 {
    throw XXX; // 예외 발생을 알림. XXX는 예외 값
}
예외를 발견한다면 {
    throw YYY; // 예외 발생을 알림. YYY는 예외 값
}
}
catch(처리할 예외 파라미터 선언) { // catch { } 블록
    예외 처리문
}
catch(처리할 예외 파라미터 선언) { // catch { } 블록
    예외 처리문
}
```

throw와 catch

```
throw 3 ; // int 타입의 값 3을 예외로 던짐
```

...

```
catch( int x ) { // throw 3;이 실행되면 catch() 문 실행. x에 3이 전달
```

...

```
}
```

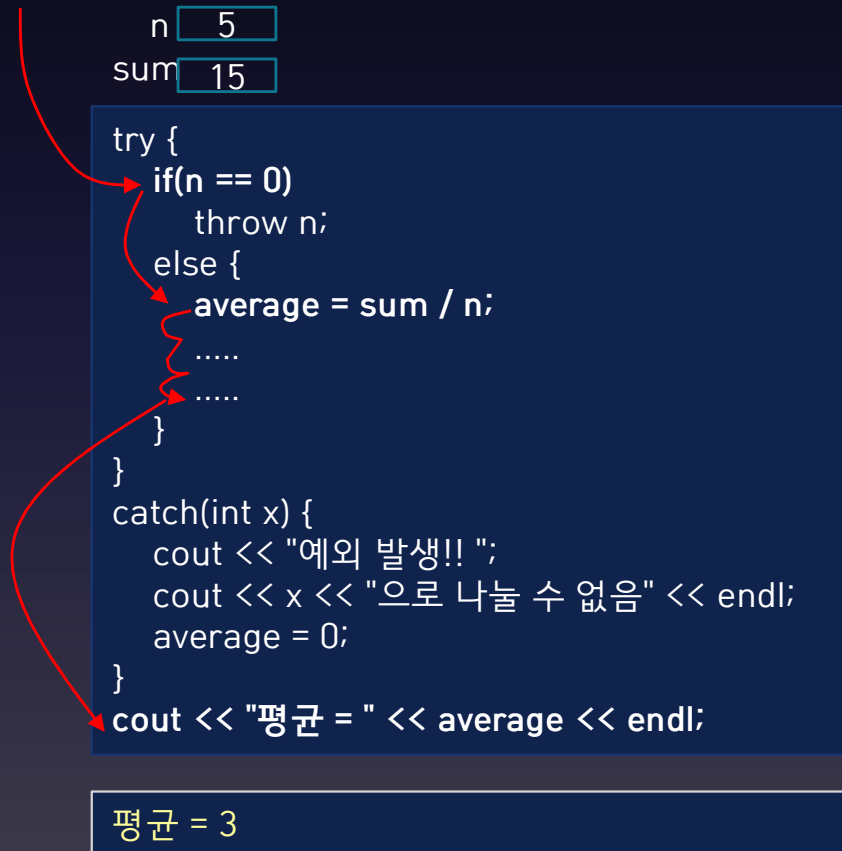
예외 타입

매개변수

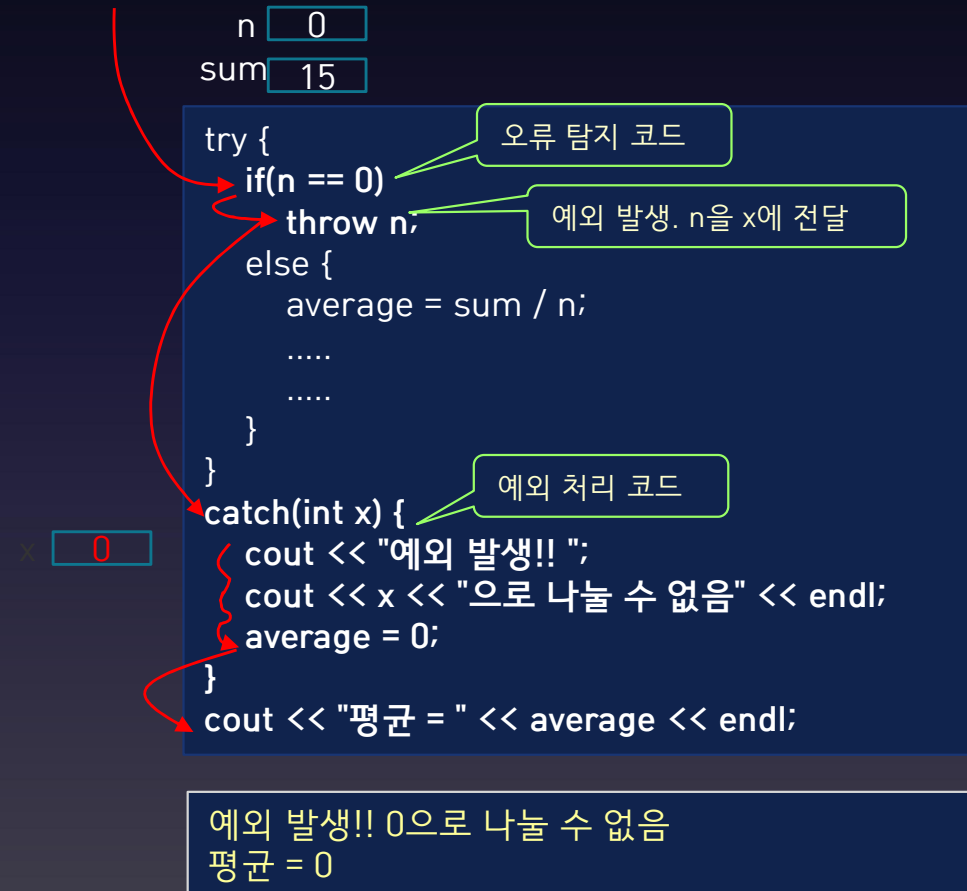
```
try {  
    throw 3.5; // double 타입의 예외 던지기  
}  
catch(double d) { // double 타입 예외 처리. 3.5가 d에 전달됨  
    ...  
}
```

```
try {  
    throw "음수 불가능"; // char* 타입의 예외 던지기  
}  
catch(const char* s) { // const char* 타입의 예외 처리. 예외 값은 "음수 불가능"이 s에 전달됨  
    cout << s; // "음수 불가능" 출력  
}
```

try-throw-catch의 예외 처리 과정



예외가 발생하지 않은 경우



예외가 발생한 경우

예제

- 합과 인원수를 입력 받아 평균을 내는 코드에, 인원수가 0이거나 음수가 입력되는 경우 예외 처리하는 프로그램을 작성하라.

```
int main()
{
    int n, sum, average;

    while(true) {
        cout << "합을 입력하세요>>";
        cin >> sum;
        cout << "인원수를 입력하세요>>";
        cin >> n;

        try {
            if(n <= 0) // 오류 탐지
                throw n; // 예외 발생. catch(int x) 블록으로 점프
            else
                average = sum / n;
        }
        catch(int x) {
            cout << "예외 발생!! " << x << "으로 나눌 수 없음" << endl;
            average = 0;
            cout << endl;
            continue;
        }
        cout << "평균 = " << average << endl << endl; // 평균 출력
    }
}
```

```
합을 입력하세요>>15
인원수를 입력하세요>>5
평균 = 3
```

```
합을 입력하세요>>12
인원수를 입력하세요>>-3
예외 발생!! -3으로 나눌 수 없음
```

```
합을 입력하세요>>25
인원수를 입력하세요>>0
예외 발생!! 0으로 나눌 수 없음
```

```
합을 입력하세요>>
```

throw와 catch의 예

- 하나의 try 블록에 다수의 catch 블록 연결

```
try {  
    ...  
    throw "음수 불가능";  
    ...  
    throw 3;  
    ...  
}  
catch(const char* s) { // 문자열타입 예외 처리. "음수 불가능"이 s에 전달  
    ...  
}  
catch(int x) { // int 타입 예외 처리. 3이 x에 전달됨  
    ...  
}
```

- 함수를 포함하는 try 블록

```
int main() {  
    try {  
        int n = multiply(2, -3);  
        cout << "곱은 " << n << endl;  
    }  
    catch(const char* negative) {  
        cout << "exception happened : " << negative;  
    }  
}
```

exception happened : 음수 불가능

함수 호출

```
int multiply(int x, int y) {  
    if(x < 0 || y < 0)  
        throw "음수 불가능";  
    else  
        return x*y;  
}
```

예외 던지기

예제 - 밑과 지수 수정

```
#include <iostream>
using namespace std;

int getExp(int base, int exp) {
    if(base <= 0 || exp <= 0) {
        throw "음수 사용 불가";
    }
    int value=1;
    for(int n=0; n<exp; n++)
        value = value * base;
    return value; // 계산 결과 리턴
}

int main() {
    int v=0;
    try {
        v = getExp(2, 3); // v = 2의 3승 = 8
        cout << "2의 3승은 " << v << "입니다." << endl;

        v = getExp(2, -3); // 2의 -3 승 ?
        cout << "2의 -3승은 " << v << "입니다." << endl;
    }
    catch(const char *s) {
        cout << s << endl;
    }
}
```

예외 발생

catch 블록으로
바로 점프

2의 3승은 8입니다.
음수 사용 불가

예제

- 문자열을 정수로 변환하는 `stringToInt()` 함수를 작성하라. 정수로 변환할 수 없는 문자열의 경우 예외 처리

catch 블록으로 바로 점프

예외 발생

"123" 은 정수 123로 변환됨
1A3 처리에서 예외 발생!!

```
#include <iostream>
#include <cstring>
using namespace std;

int stringToInt(const char x[]) {
    int sum = 0;
    int len = strlen(x);
    for(int i=0; i<len; i++) {
        if(x[i] >= '0' && x[i] <= '9')
            sum = sum*10 + x[i]-'0';
        else
            throw x; // char* 타입의 예외 발생
    }
    return sum;
}

int main() {
    int n;
    try {
        n = stringToInt("123"); // 문자열을 정수로 변환
        cout << "\"123\" 은 정수 " << n << "로 변환됨" << endl;
        n = stringToInt("1A3"); // 문자열을 정수로 변환
        cout << "\"1A3\" 은 정수 " << n << "로 변환됨" << endl;
    }
    catch(const char* s) {
        cout << s << "처리에서 예외 발생!!" << endl;
        return 0;
    }
}
```

실습

- `sum()`함수는 매개 변수로 주어진 `a`에서 `b`까지의 양의 정수 합을 구한다. 만약, `a`가 `b`보다 크거나, 두 수 중 하나라도 음수이면 예외를 발생시킨다. 아래와 같이 실행되도록 `sum()`함수를 정의하시오.

```
:  
  
int main( ) {  
    try {  
        cout << sum(2,5) << endl; //14 출력  
        cout << sum(-1,5) << endl; // 예외 발생  
        cout << sum(7,3) << endl; // 예외 발생  
    }  
    catch(const char* s) {  
        cout << s << endl;  
    }  
}
```

14

Exception raised !

실습-답

- `sum()` 함수는 매개 변수로 주어진 `a`에서 `b`까지의 양의 정수 합을 구한다. 만약, `a`가 `b`보다 크거나, 두 수 중 하나라도 음수이면 예외를 발생시킨다. 아래와 같이 실행되도록 `sum()` 함수를 정의하시오.

```
int sum(int a, int b)
{
    if(a > b || a < 0 || b < 0)
        throw "Exception Raised !";
    return (b-a+1)*(a+b)/2;
}
```

예외를 발생시키는 함수의 선언

- 예외를 발생시키는 함수는 다음과 같이 선언 가능
 - 함수 원형에 연이어 `throw(예외 타입, 예외 타입, ...)` 선언

```
int max(int x, int y) throw(int) {  
    if(x < 0) throw x;  
    else if(y < 0) throw y;  
    else if(x > y) return x;  
    else return y;  
}
```

모두 int 타입 예외 발생

```
double valueAt(double *p, int index) throw(int, char*) {  
    if(index < 0)  
        throw "index out of bounds exception";  
    else if(p == NULL)  
        throw 0;  
    else  
        return p[index];  
}
```

char* 타입 예외 발생

int 타입 예외 발생

- 장점
 - 프로그램의 작동을 명확히 함
 - 프로그램의 가독성 높임

다중 try { } 블록

- try { } 블록 내에 try { } 블록의 중첩 가능

```
try {  
    ...  
    throw 3;  
    ...  
    try {  
        throw "abc";  
        ...  
        throw 5;  
        ...  
    }  
    catch(int inner) {  
        cout << inner; // 5 출력  
    }  
}  
catch(const char* s) {  
    cout << s; // "abc" 출력  
}  
catch(int outer) {  
    cout << outer; // 3 출력  
}
```

try 블록에 연결된
catch 블록으로 점프

바깥 try 블록에
연결된 catch 블
록으로 점프

throw 사용 시 주의 사항

- throw 문의 위치
 - 항상 try { } 블록 안에서 실행
 - try { } 블록 밖에서 호출
 - 시스템이 abort() 호출, 강제 종료
- 예외를 처리할 catch()가 없으면 프로그램 강제 종료
- catch 블록 내에도 try, catch 가능

```
throw 3; // 프로그램이 비정상 종료된다
try {
    ...
}
catch(int n) {
    ...
}
```

```
try {
    throw "aa"; // char* 타입의 예외를 처리할
                // catch() { } 블록이 없기 때문에
                // 프로그램 종료
}
catch(double p) {
    ...
}
```

```
try {
    throw 3;
}
catch(int x) {
    try {
        throw "aa"; // 아래의 catch(const char* p) { }
                    // 블록에서 처리된다
    }
    catch(const char* p) {
        ...
    }
}
```

예외 클래스 만들기

■ 예외 값의 종류

- 기본 타입의 예외 값
 - 정수, 실수, 문자열 등 비교적 간단한 예외 정보 전달
- 객체 예외 값
 - 예외 값으로 객체를 던질 수 있다.
 - 예외 값으로 사용할 예외 클래스 작성 필요

■ 예외 클래스

- 사용자는 자신 만의 예외 정보를 포함하는 클래스 작성
- `throw`로 객체를 던짐
 - 객체가 복사되어 예외 파라미터에 전달

예제 - 예외 클래스

```
class MyException {
    int lineNo;
    string func, msg;
public:
    MyException(int n, string f, string m) {
        lineNo = n; func = f; msg = m;
    }
    void print() { cout << func << ":" << lineNo << " , "
        << msg << endl; }
};

class DivideByZeroException : public MyException {
public:
    DivideByZeroException(int no, string func, string msg)
        : MyException(no, func, msg) { }
};

class InvalidInputException : public MyException {
public:
    InvalidInputException(int no, string func, string msg)
        : MyException(no, func, msg) { }
};
```

```
int main() {
    int x, y;
    try {
        cout << "나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>>";
        cin >> x >> y;
        if(x < 0 || y < 0)
            throw InvalidInputException(32, "main()", "음수 입력 예외 발생");
        if(y == 0)
            throw DivideByZeroException(34, "main()", "0 나누기 예외 발생");
        cout << (double)x / (double)y;
    }
    catch(DivideByZeroException &e) {
        e.print();
    }
    catch(InvalidInputException &e) {
        e.print();
    }
}
```

나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>>2 5
0.4

나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>>200 -3
main():32 ,음수 입력 예외 발생

나눗셈을 합니다. 두 개의 양의 정수를 입력하세요>>20 0
main():34 ,0 나누기 예외 발생

예제 - 스트림 상태 검사

- 다음은 양의 정수를 입력받아 구구단을 출력하는 프로그램이다. try-throw-catch를 이용하여 예외처리를 수행하시오.

```
int main() {  
    int n;  
    while(true) {  
        cout << "양수 입력 >> ";  
        cin >> n;  
        for(int i=1; i<=9; i++)  
            cout << n << 'x' << i << '=' << n*i << ' ' << endl;  
    }  
}
```

예제 - 스트림 상태 검사

```
int main() {
    int n;
    while(true) {
        cout << "양수 입력 >> ";
        try {
            cin >> n;
            if(cin.fail()) // failbit이 1로 셋되어 있는 경우, 포맷에 맞지 않는 입력 발생
                throw "정수가 아닌 값 입력";
            if(n <= 0 || n > 9)
                throw n;
            for(int i=1; i<=9; i++)
                cout << n << 'x' << i << '=' << n*i << ' ';
            cout << endl;
        }
        catch(int e) {
            cout << "잘못된 입력입니다. 1~9 사이의 정수만 입력하세요" << endl;
        }
        catch(const char *s) {
            cout << "입력 오류가 발생하여 더 이상 입력되지 않습니다. 프로그램을 종료합니다" << endl;
            return 0;
        }
    }
}
```

C 사용

C++ 코드에서 C 코드의 링킹

■ 이름 규칙(naming mangling)

- 컴파일 후 목적 코드에 이름 붙이는 규칙
 - 변수, 함수, 클래스 등의 이름

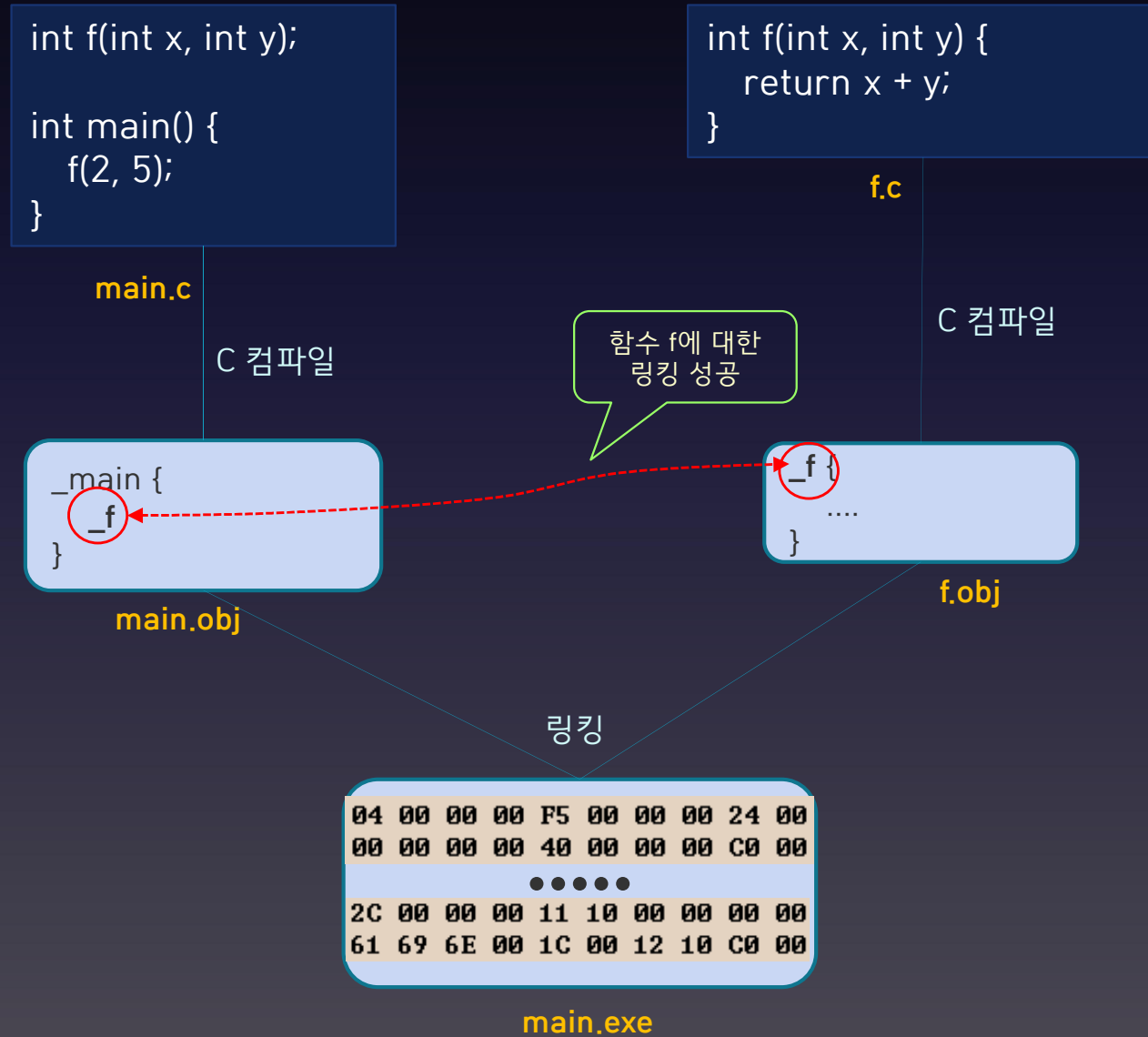
■ C 언어의 이름 규칙

- 이름 앞에 밑줄표시문자(_)를 붙인다.
 - `int f(int x, int y)` ----> `_f`
 - `int main()` -----> `_main`

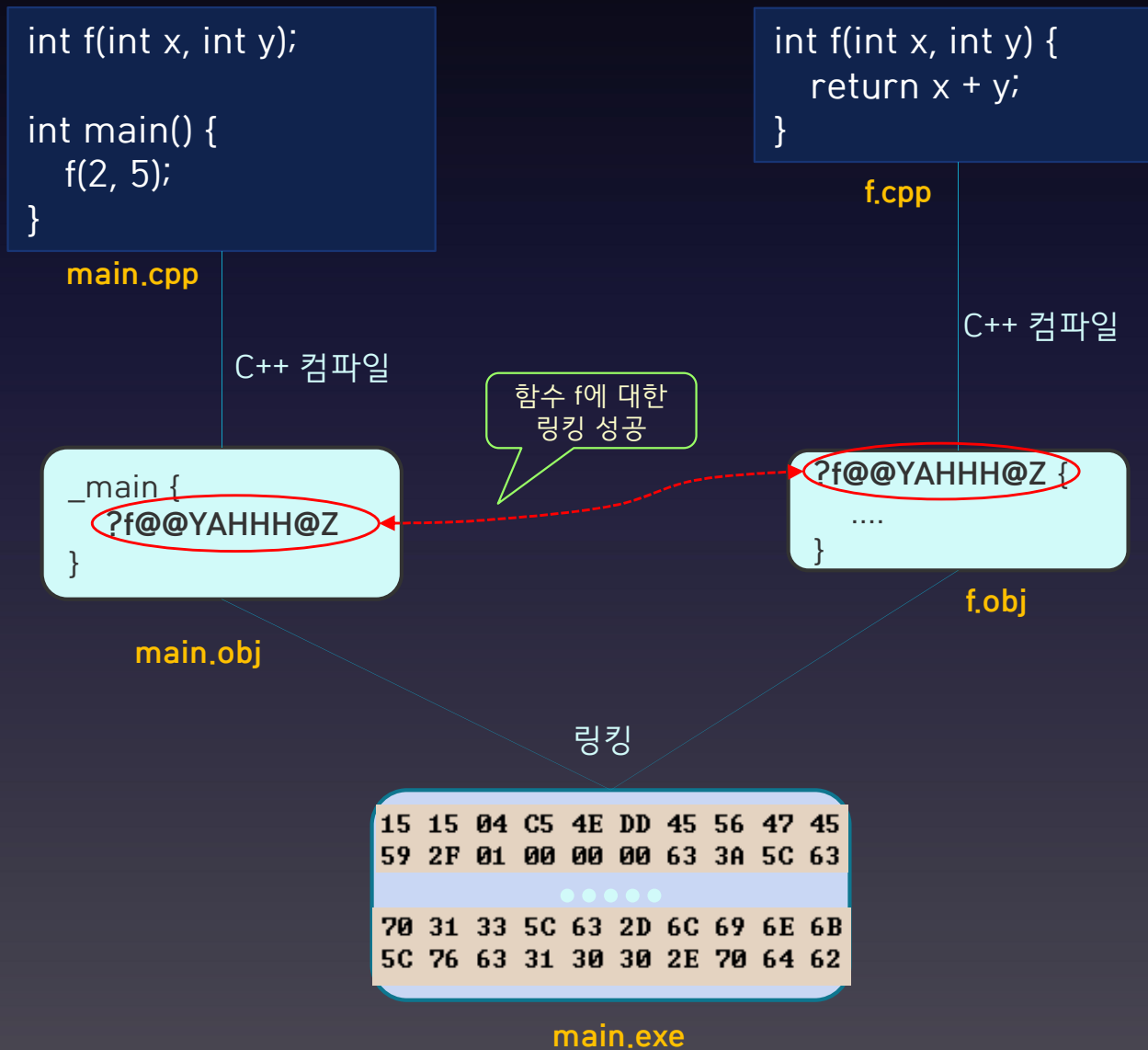
■ C++의 이름 규칙

- 함수의 매개 변수 타입과 개수, 리턴 타입에 따라 복잡한 이름
 - `int f(int x, int y)` ----> `?f@@YAHHH@Z`
 - `int f(int x)` ----> `?f@@YAXH@Z`
 - `int f()` ----> `?f@@YAHXZ`
 - `int main()` ----> `_main`

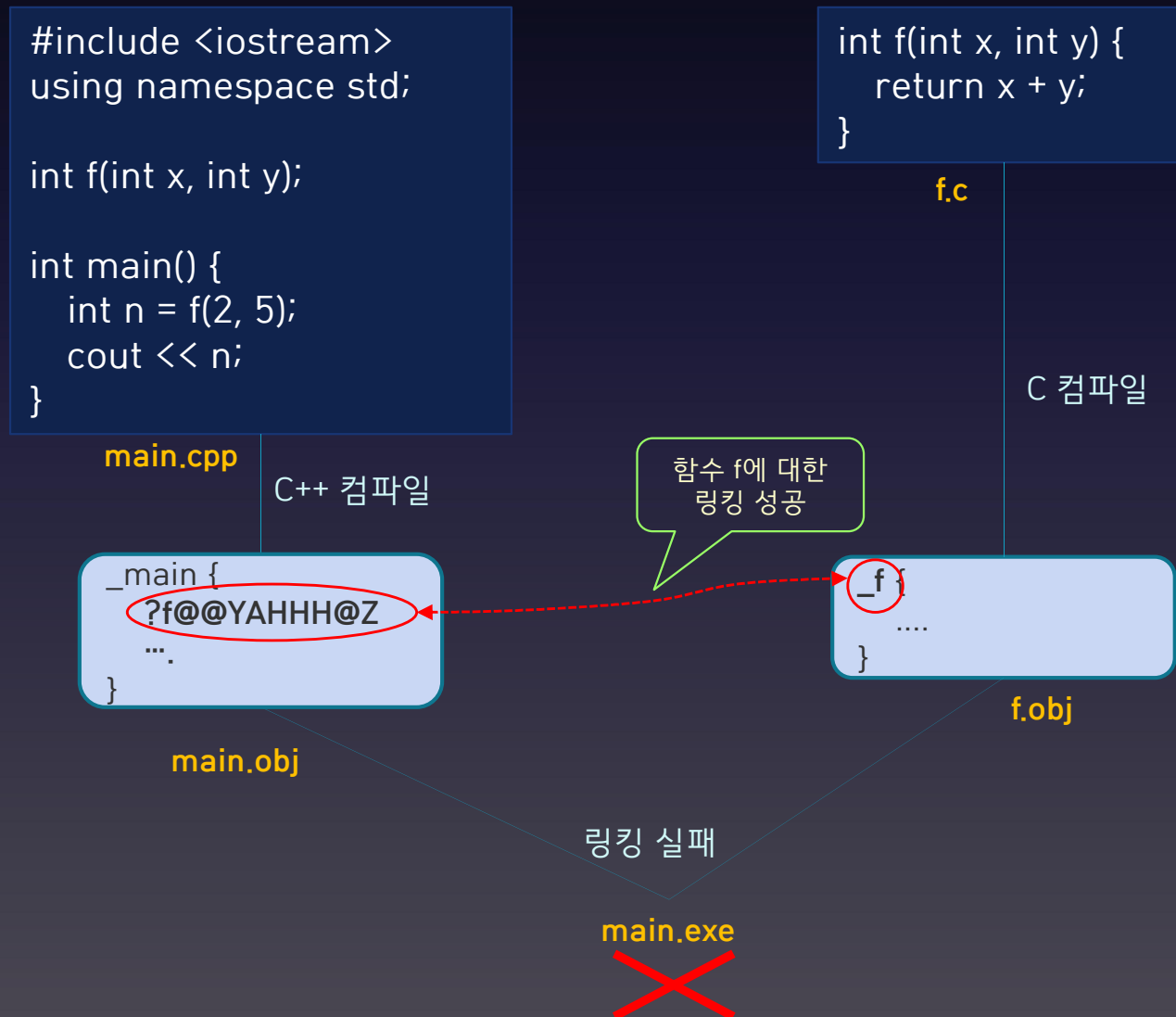
C 프로그램의 컴파일과 링킹



C++ 소스의 컴파일과 링킹



C++에서 C 함수 호출 시 링크 오류



extern "C"

- extern "c"

- C 컴파일러로 컴파일할 것을 지시
 - C 이름 규칙으로 목적 코드를 생성할 것을 지시

- 사용법

- 함수 하나만 선언

```
extern "C" int f(int x, int y);
```

- 여러 함수들 선언

```
extern "C" {  
    int f(int x, int y);  
    void g();  
    char s(int []);  
}
```

- 헤더파일 통째로 선언

```
extern "C" {  
    #include "mycfuction.h"  
}
```

extern "C"를 이용한 링크

```
#include <iostream>
using namespace std;
```

```
int g(int x, int y);
```

```
extern "C" int f(int x, int y);
```

```
int main() {
    cout << f(2, 5) << endl;
    cout << g(2, 5) << endl;
}
```

```
int f(int x, int y) {
    return x + y;
}
```

```
int g(int x, int y) {
    return x - y;
}
```

f.c

g.cpp

7
-3

C 컴파일

C++ 컴파일

main.cpp

C++ 컴파일

함수 f에 대한
링크 성공

함수 g에 대한
링크 성공

```
_main {
    _f
    ?g@@YAHHH@Z
}
```

```
_f {
    ....
}
```

```
?g@@YAHHH@Z {
    ....
}
```

main.obj

f.obj

g.obj

링크 성공

```
15 15 04 C5 4E DD 45 56 47 45
59 2F 01 00 00 00 63 3A 5C 63
.....
70 31 33 5C 63 2D 6C 69 6E 6B
5C 76 63 31 30 30 2E 70 64 62
```

main.exe

실습

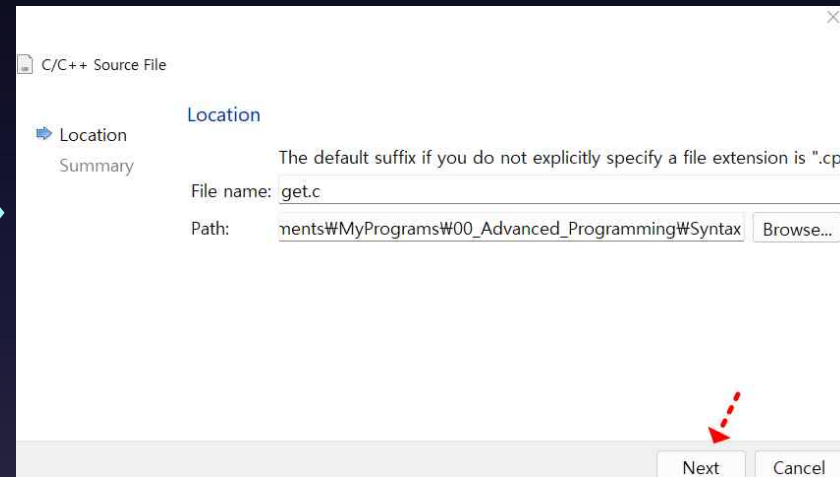
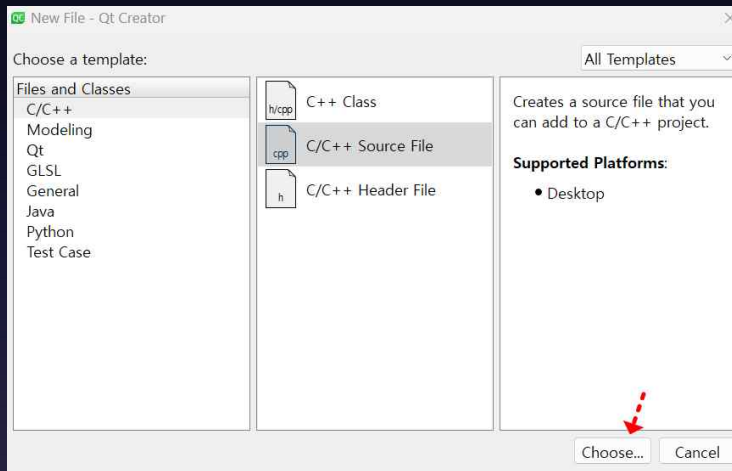
- 아래의 "get.c" 파일을 포함하는 프로젝트를 생성하고 두 수를 입력받아 그 곱을 출력하는 프로그램을 작성하시오.

```
#include <stdio.h>

int get()
{
    int c;
    printf("Enter a Number>> ");
    scanf("%d", &c);
    return c;
}
```

실습

- ① 프로젝트를 생성하고 "get.c" 파일을 추가한다.





- ② 아래와 같이 "main.cpp"을 수정한다.

```
#include <iostream>
using namespace std;

extern "C" int get();

int main()
{
    int n = get();
    int m = get();
    cout << "The multiplication is " << n * m << endl;
}
```

실습

- ③  을 클릭하여 프로젝트 파일을 실행시킨다
- ④ "main.cpp"에서 `extern "C"`을 삭제하고,  을 클릭하여 Build 한다
→ 오류를 확인함

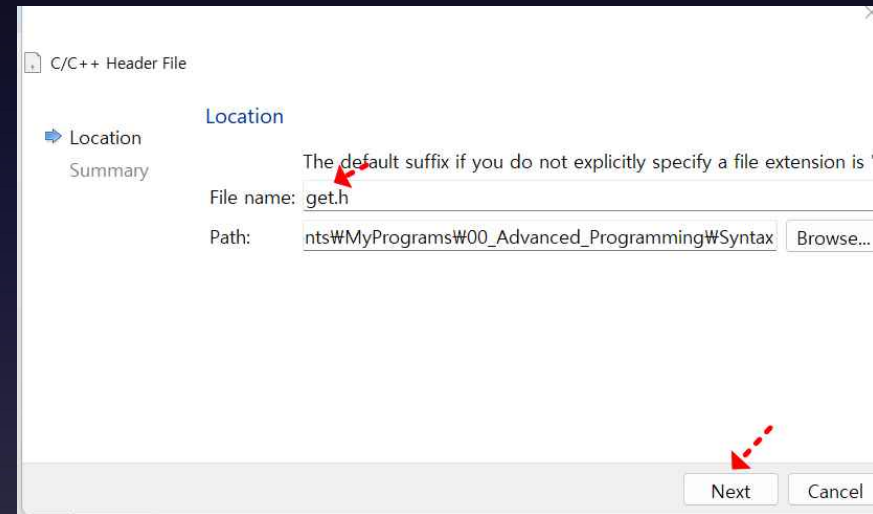
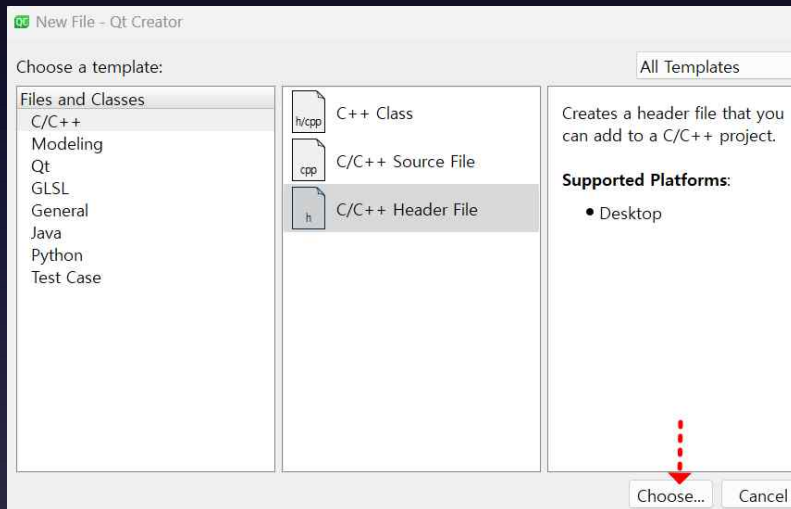
```
#include <iostream>
using namespace std;

int get();

int main()
{
    int n = get();
    int m = get();
    cout << "The multiplication is " << n * m << endl;
}
```

실습

⑤ 프로젝트에 "get.h"을 추가한다



```
Analyze Tools Window Help
aet.h
1  #ifndef GET_H
2  #define GET_H
3
4  int get();
5
6  #endif // GET_H
7
```

실습

⑥ "main.cpp"에서 아래와 같이 수정하고 실행시킨다

```
#include <iostream>
using namespace std;

extern "C"{
    #include "get.h"
}

int main()
{
    int m = get();
    int n = get();

    cout << " The multiplication is " << m*n << endl;
}
```

⑦ "main.cpp"에서 아래와 같이 extern "C"을 삭제하고,



을 클릭하여 Build 한다

→ 오류를 확인함

```
#include <iostream>
using namespace std;

#include "get.h"

int main()
{
    int m = get();
    int n = get();

    cout << " The multiplication is " << m*n << endl;
}
```

