

예외(Exception Handling)

1

- 예외란?
 - ▣ 실행 중, 프로그램 오동작이나 결과에 영향을 미치는 예상치 못한 상황 발생
- 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
 - ▣ 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

기존의 예외처리 방식

예외 처리

- 일반적이지 않은 프로그램의 흐름의 처리
- 에러가 아니다!

```
int main(void)
{
    int a, b;
    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;

    cout<<"a/b의 몫 : "<<a/b<<endl;
    cout<<"a/b의 나머지 : "<<a%b<<endl;
    return 0;
}
```

```
int main(void)
{
    int a, b;

    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;
    if(b==0){
        cout<<"입력오류!"<<endl;
    }
    else {
        cout<<"a/b의 몫 : "<<a/b<<endl;
        cout<<"a/b의 나머지 : "<<a%b<<endl;
    }
    return 0;
}
```

예제 1 예외 상황에 대한 대처가 없는 프로그램 사례

C++의 예외처리 메커니즘

□ try & catch

```
try {  
    /* 예외 발생 예상 지역 */  
}  
  
catch(처리 되어야 할 예외의 종류) {  
    /* 예외를 처리하는 코드가 존재할 위치 */  
}
```

throw ex; // ex를 가리켜 보통은 그냥 “예외”라고 표현을 한다.

C++ 예외 처리 기본 형식

4

□ 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

5

```
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  
}
```

예외 처리

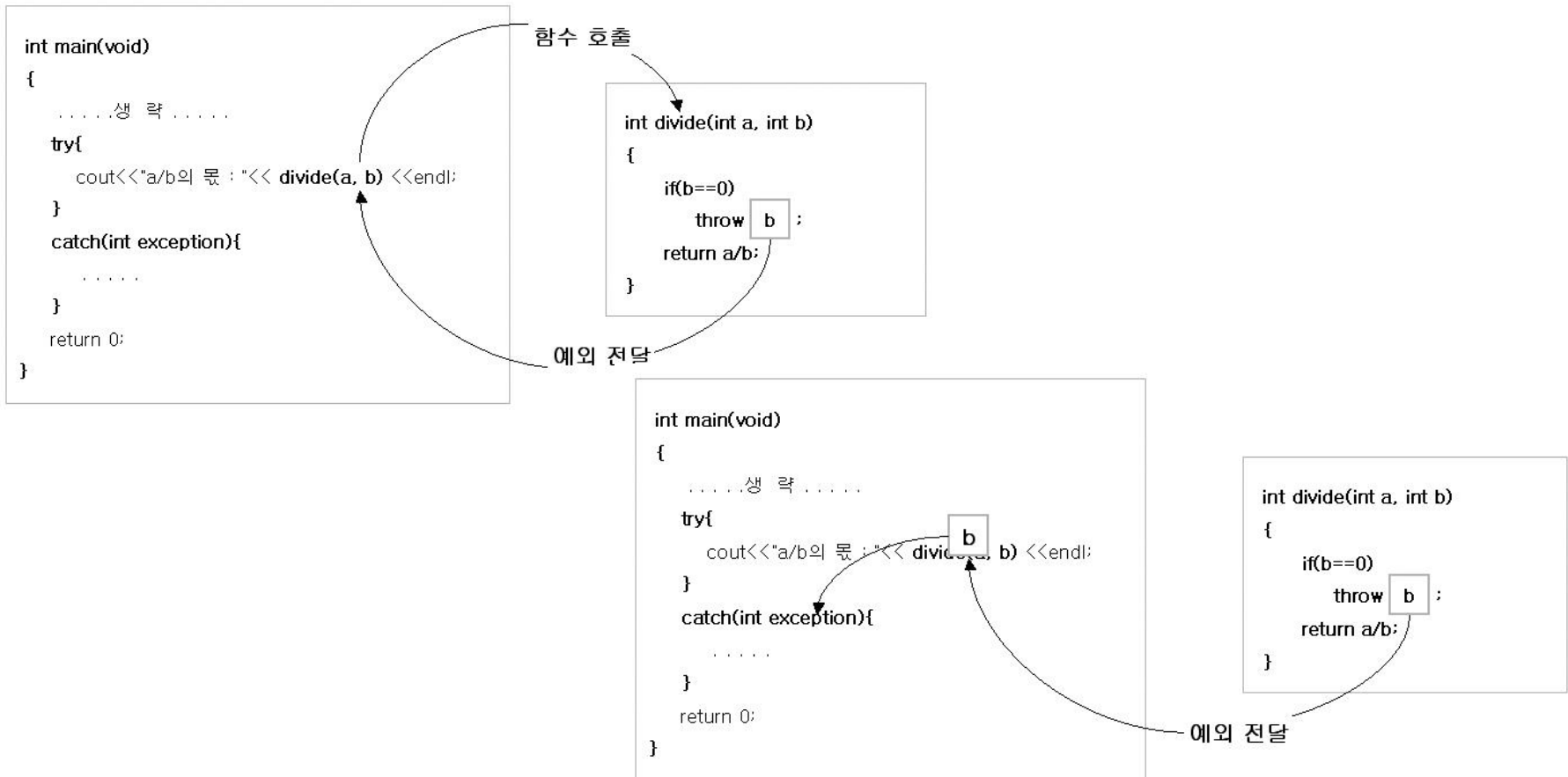
6

변수 `b` 를 이용하여 예외 상황이 발생하였음을 알리고, 이 순간 바로 C++ 예외처리 메커니즘이 작동하여 try블록 다음에 있는 catch블록에 전달된다.



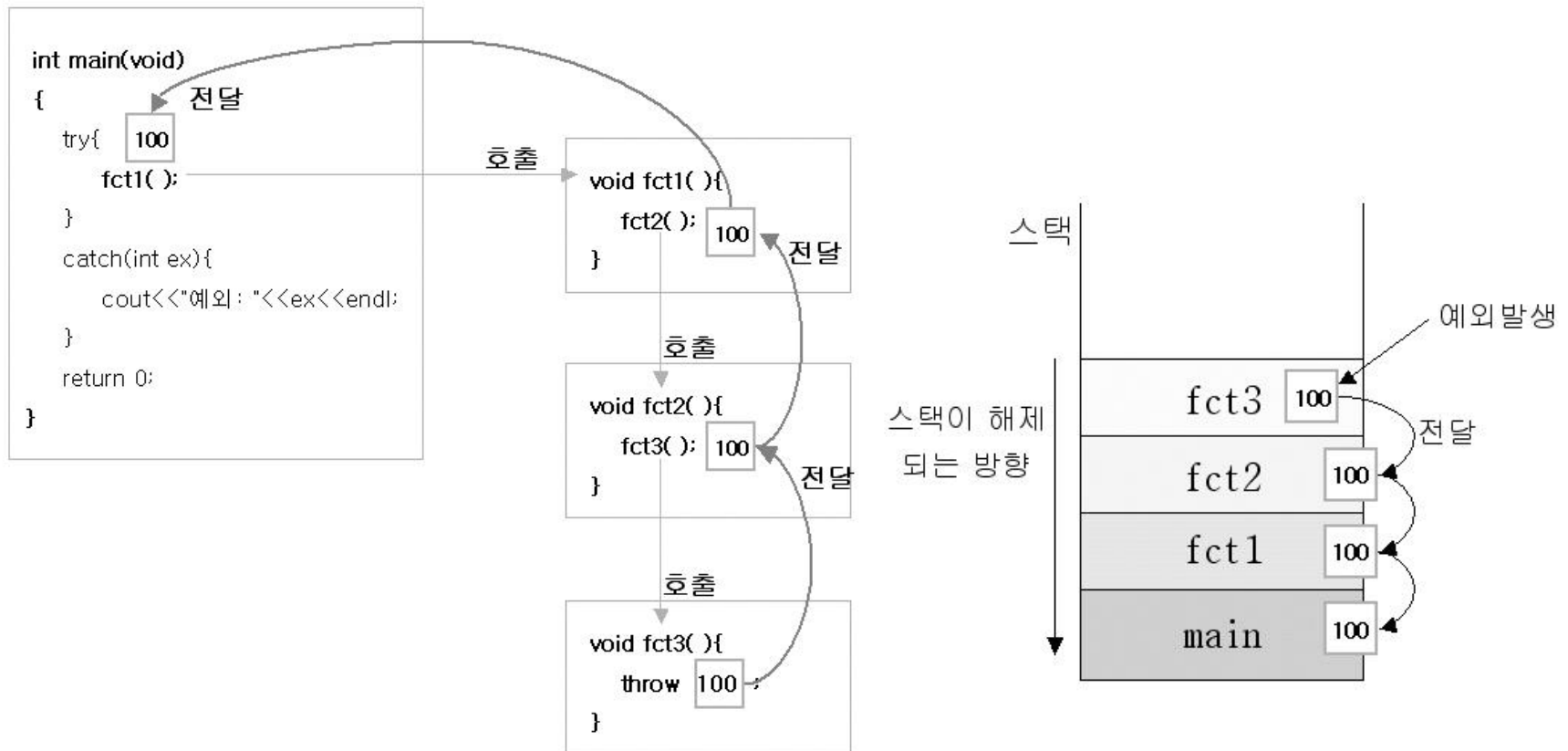
Stack Unwinding(스택 풀기)

함수를 호출했는데 함수에서 오류가 발생하였다. 이때 함수내에서 예외를 처리해주는 (try,catch) 루틴이 없다면 이 함수를 호출한 영역으로 예외는 전달된다. 이를 스택풀기라고 한다.



Stack Unwinding (스택 풀기)

함수내 예외 발생 시 예외처리 하지 않는다면, 예외처리는 호출한 영역으로 예외가 전달된다.



예외 처리 abort() 함수

9

예외가 처리되지 않은 상태에서 예외가 발생하면 프로그램은 abort() 함수가 호출되면서 프로그램은 종료하게 된다. 또한 예외 블록이 있더라도 자료가 일치하지 않은 catch블록은 예외를 처리하지 못한다. 따라서 abort() 함수가 호출된다.

```
int main()
{
    int a, b;

    cout<<"두개의 숫자 입력 : ";
    cin>>a>>b;
    cout<<"a/b의 몫 : "<<divide(a, b)<<endl;

    //예외가 처리되지 않은 상태에서 예외가
    발생, abort(); 호출

    return 0;
}
```

```
int divide(int a, int b)
{
    if(b==0)
        throw b;
    return a/b;
}
```

예외 처리 abort() 함수

10

예외 블록이 있더라도 자료형이 일치하지 않은 catch블록은 예외를 처리하지 못한다. 따라서 abort() 함수가 호출된다.

```
int main(void)
{
    int a, b;

    cout<<"두개의숫자입력: ";
    cin>>a>>b;
    try{
        cout<<"a/b의몫: "<<divide(a, b)<<endl;
    }
    catch(char exception)    //예외 자료형 불 일치
    {
        cout<<exception<<" 입력."<<endl;
        cout<<"입력오류! 다시실행하세요."<<endl;
    }

    return 0;
}
```

```
int divide(int a, int b)
{
    if(b==0)
        throw b; //정수형 오류 핸들러 없으므로
                // abort()호출

    return a/b;
}
```

예외상황을 나타내는 클래스의 설계

- 예외 상황을 나타내는 클래스 & 객체
 - ▣ 예외 클래스, 예외 객체
 - ▣ 일반 클래스, 객체와 다르지 않다.
 - ▣ 예외 상황을 알려주기 위한 용도로 사용
- 예외는 클래스의 객체로 표현되는 것이 일반적
- 예외가 처리되지 않으면
 - `stdlib.h`에 선언되어 있는 `abort` 함수의 호출에 의해 프로그램 종료

예외 클래스 만들기

12

□ 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

▣ throw로 객체를 던짐

- 객체가 복사되어 예외 파라미터에 전달

예외 클래스 만들기

13

예외를 알리기 위해 클래스(예외클래스)를 정의할 수 있다. 이러한 클래스를 **예외클래스라 한다**. 객체를 이용하면 예외상황이 발생한 원인에 대해 정보를 자세히 담을 수 있다.(예외개체)

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

char* account="1001"; //계좌번호
int passwd=100;      //비밀번호
int balance=1000;    //잔액.

class AccountExpt //예외클래스
{
    char acc[10];
    int pass;
public:
    AccountExpt(char* str, int pid)
    {
        strcpy(acc, str);
        pass=pid;
    }
    void What()
    {
        cout<<"계좌번호: "<<acc<<endl;
        cout<<"비밀번호: "<<pass<<endl;
    }
};
```

예제 7 예외 클래스 만들기

14

```
int main()
{
    char acc[10];
    int pid;
    int money;

    try{
        cout<<"계좌번호입력: ";
        cin>>acc;
        cout<<"비밀번호4자리입력: ";
        cin>>pid;
        if(strcmp(account, acc) || passwd!=pid)
            throw AccountExpt(acc, pid); //예외객체 생성
            //개체가생성되면서생성자호출하고, 해당catch를실행

        cout<<"출금액입력: ";
        cin>>money;
        if(balance<money)
            throw money;

        balance-=money;
        cout<<"잔액: "<<balance<<endl;
    }
```

예외 클래스 만들기

15

```
catch(int money)
{
    cout<<"부족금액: "<<money<<endl;
}
catch(AccountExpt& expt)
{
    cout<<"다음입력을다시한번확인하세요"<<endl;
    expt.What();
}

cout<<"End."<<endl;

return 0;
}
```

성능이 중요시 되는 프로그램에서는 C++ 예외 처리 메커니즘을 적용하지 않는게 좋다. Try-catch를 사용하게 되면, 컴파일된 코드에 예외검사 코드가 추가되고, 스택 풀기(unwinding)를 위한 코드가 추가된다. 따라서 실행파일의 크기는 증가되고 속도 또한 저하된다.

Try-catch 속도의 저하가 부담이 된다면 전통적인 방법에 의한 예외처리를 사용하는 것이 좋다.

예외를 나타내는 클래스와 상속관계

16

상속관계에 있는 파생클래스 오류는, 기반클래스 오류 핸들러가 동작한다.
이유는 파생객체는 기반클래스를 상속받았기 때문이다.

```
#include <iostream>
using namespace std;
class ExceptA
{
public:
    void What(){
        cout<<"ExceptA 예외"<<endl;
    }
};

class ExceptB : public ExceptA
{
public:
    void What(){
        cout<<"ExceptB 예외"<<endl;
    }
};

class ExceptC : public ExceptB
{
public:
    void What(){
        cout<<"ExceptC 예외"<<endl;
    }
};
```


예외를 나타내는 클래스와 상속관계

17

```
void ExceptFunction(int ex)
{
    if(ex==1)
        throw ExceptA();
    else if(ex==2)
        throw ExceptB();
    else
        throw ExceptC();
}

int main()
{
    int exID;
    cout<<"발생시킬예외의숫자: ";
    cin>>exID;

    try{
        ExceptFunction(exID);
    }
    //입력 값에 상관 없이 모두 여기서 실행
    //상속관계에 있기 때문에 ExceptB 나 ExceptC의 예외는 ExceptA의 예외도 된다.

    catch(ExceptA& ex){
        cout<<"catch(ExceptA& ex)에의한처리"<<endl;
        ex.What();
    }
    catch(ExceptB& ex)
        //파생클래스 기반클래스로 참조전달(Up Casting)
    {
        cout<<"catch(ExceptB& ex)에의한처리"<<endl;
        ex.What();
    }
    catch(ExceptC& ex){
```

예외를 나타내는 클래스와 상속관계

따라서 이문제의 해결은 반대로 catch를 구성한다. 이유는 상속관계는 역으로 성립하지 않는다. 즉 ExceptC의 예외는 ExceptA의 예외가 아니다.

18

```
int main()
{

    int exID;
    cout<<"발생시킬 예외의 숫자: ";
    cin>>exID;

    try{
        ExceptFunction(exID);
    }
    catch(ExceptC& ex)  // ExceptA 예외는 ExceptC 예외가 아니다
    {
        cout<<"catch(ExceptC& ex)에 의한 처리"<<endl;
        ex.What();
    }
    catch(ExceptB& ex)
    {
        cout<<"catch(ExceptB& ex)에 의한 처리"<<endl;
        ex.What();
    }
    catch(ExceptA& ex)
    {
        cout<<"catch(ExceptA& ex)에 의한 처리"<<endl;
        ex.What();
    }

    return 0;
}
```

new에 의해 전달되는 예외

19

동적메모리 할당이 실패하게되면 bad_alloc 예외가 자동으로 호출된다.

```
int main()
{
    try{
        int i=0;
        while(1)
        {
            cout<<i++<<"번째할당"<<endl;
            //2차원배열포인터변수
            double(*arr)[10000]=new double[10000][10000];
        }
    }
    //동적메모리할당이실패하게되면bad_alloc 예외가자동으로호출된다.
    catch(bad_alloc ex)
    {
        cout<<endl<<"END"<<endl;
    }

    return 0;
}
```

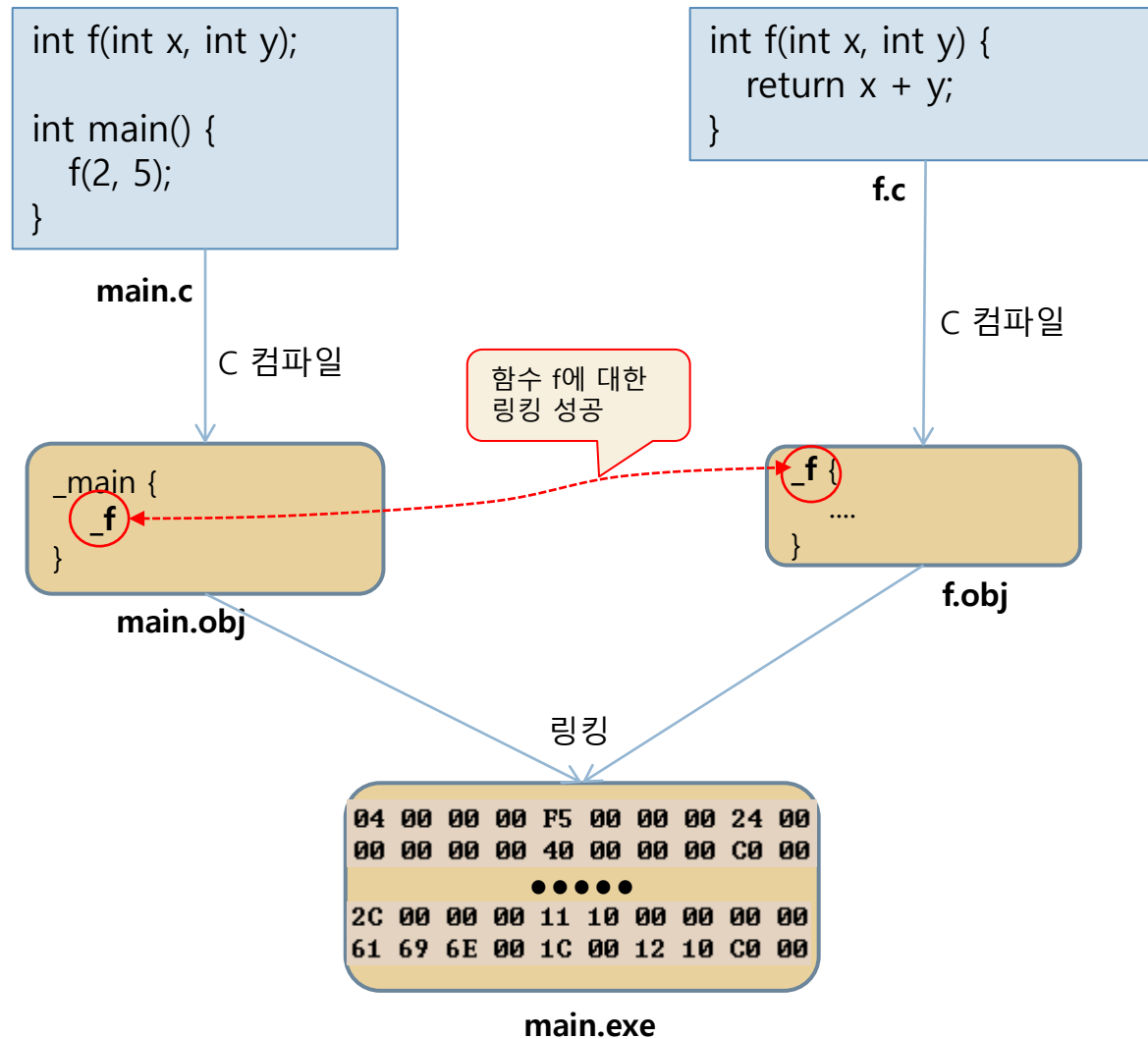
C++ 코드에서 C 코드의 링킹

20

- 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`

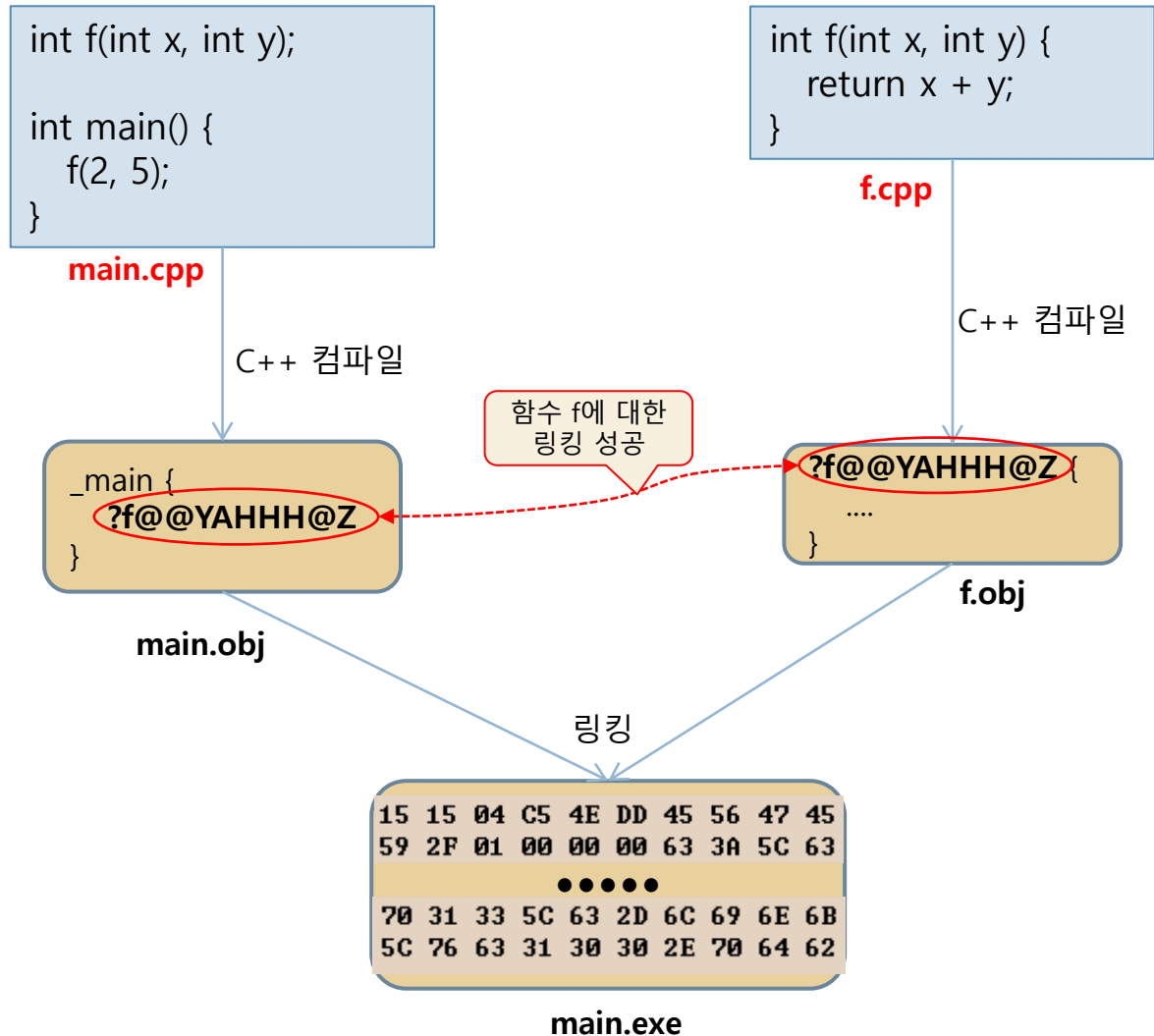
C 프로그램의 컴파일과 링킹

21



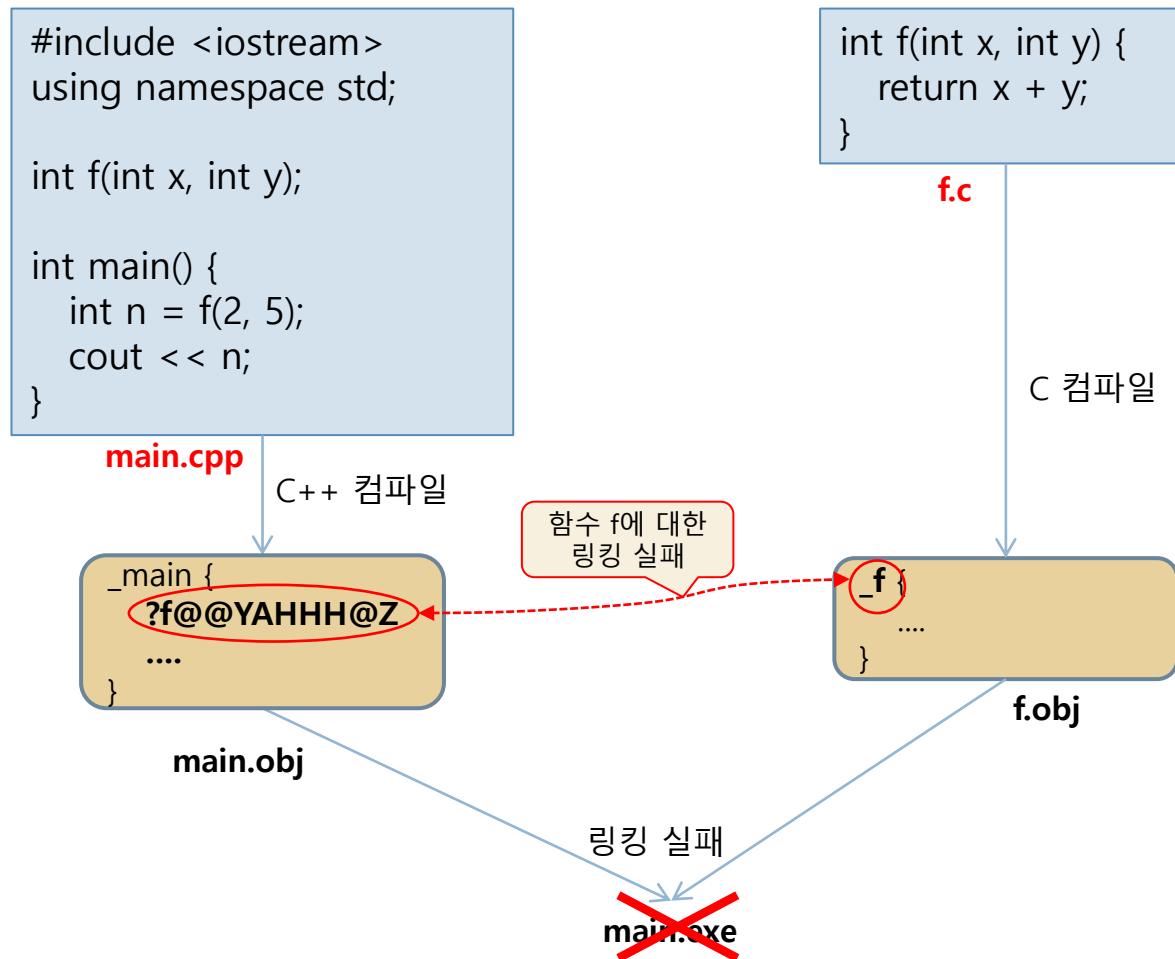
C++ 소스의 컴파일과 링킹

22



C++에서 C 함수 호출 시 링크 오류 발생

23



extern "c"

24

□ 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 이용하여 링크 성공

