

## 14장 예외 처리

- ⌘ 에러 처리
- ⌘ 예외 처리의 개념적 이해
- ⌘ 예외 처리 구문
- ⌘ throw문과 다중 예외 처리 핸들러의 사용
- ⌘ 예외 처리 클래스
- ⌘ throw문의 전달과 응용
- ⌘ new 연산자의 예외 처리
- ⌘ 함수가 전달할 수 있는 예외의 제한

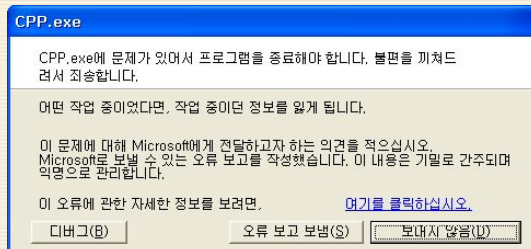
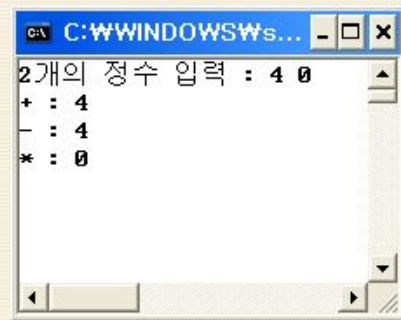
# 1. 에러 처리

## ❏ 에러

- 컴파일 시간 에러 : 주로 문법 관련 에러
- 실행 시간 에러 : 실행 도중 발견되는 에러, 주로 논리 에러
  - 예 : 산술 연산 시 0으로 나누기 에러

## ❏ 에러 처리

- 실행 중 발생할 수 있는 논리적 에러를 예측하고 이를 발견 및 처리하는 코드를 추가하는 것



```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;

    cout << "+ : " << x + y << endl;
    cout << "- : " << x - y << endl;
    cout << "* : " << x * y << endl;
    cout << "/" : " << x / y << endl;

    cout << "사칙 연산 종료" << endl;

    return 0;
}
```

에러 발생  
가능!

# 1. 에러 처리

## ▣ if 문을 이용한 0으로 나누기에 대한 에러 처리

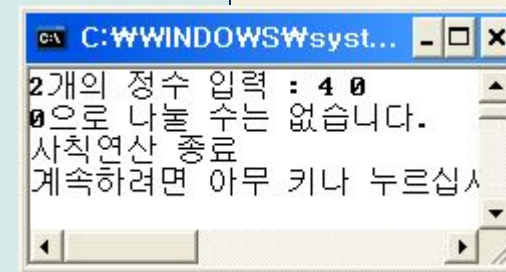
```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;

    if (y == 0) // 에러 처리
        cout << "0으로 나눌 수는 없습니다." << endl;
    else {
        cout << "+ : " << x + y << endl;
        cout << "- : " << x - y << endl;
        cout << "* : " << x * y << endl;
        cout << "/ : " << x / y << endl;
    }

    cout << "사칙연산 종료" << endl;

    return 0;
}
```



## 2. 예외 처리의 개념적 이해

### ❏ 예외 처리

- 프로그램의 정상적인 흐름에 위배되는 예외 상황에 대한 처리
  - 예외 상황 : 문제마다 또는 프로그래머마다 다르게 해석될 수 있음
  - 예1 : 0으로 나누기  
→ 에러
  - 예2 : 본 문제의 경우 x, y 모두 0 이상이어야 한다는 제약조건이 있음

### ❏ 예외 처리의 기본 개념

- 정상적인 수행 흐름과 예외 처리 부분을 분리하는 것

```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;

    if (y == 0)
        goto error;

    cout << "+ : " << x + y << endl;
    cout << "- : " << x - y << endl;
    cout << "* : " << x * y << endl;
    cout << "/" : " << x / y << endl;

    cout << "사칙연산 종료" << endl;

    return 0;

error :
    cout << "0으로 나눌 수는 없습니다." << endl;
    cout << "사칙연산 종료" << endl;

    return 0;
}
```

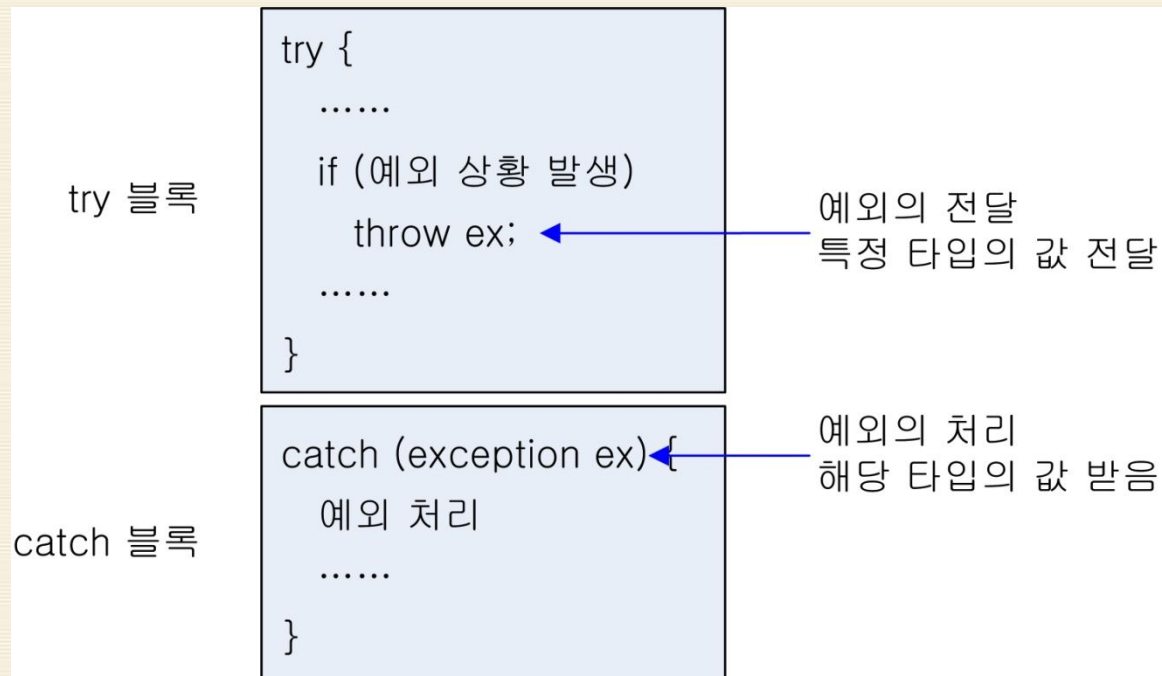
예외 상황 감지  
→ 예외 처리로 이동

예외 처리 부분

### 3. 예외 처리 구문

#### ▣ 예외 처리 구문의 구조 : try, throw, catch 문

- try 블록 : 예외가 발생할 것으로 예상되는 영역
- throw 문 : 예외가 발생할 경우 예외 전달(호출)
- catch 블록 : 예외 처리 핸들러



### 3. 예외 처리 구문

#### ▣ 예외 처리 구문의 사용 예

```
int main(void)
{
    int x, y;

    cout << "2개의 실수 입력 : ";
    cin >> x >> y;

    try {                                // try 블록 : 예외 감시
        if (y == 0)                     // throw문 : 예외 전달
            throw y;

        cout << "+ : " << x + y << endl;
        cout << "- : " << x - y << endl;
        cout << "* : " << x * y << endl;
        cout << "/ : " << x / y << endl;
    }
    catch (int a) {                      // catch 블록 : 예외 처리
        cout << "0으로 나눌 수는 없습니다." << endl;
    }

    cout << "사칙연산 종료" << endl;

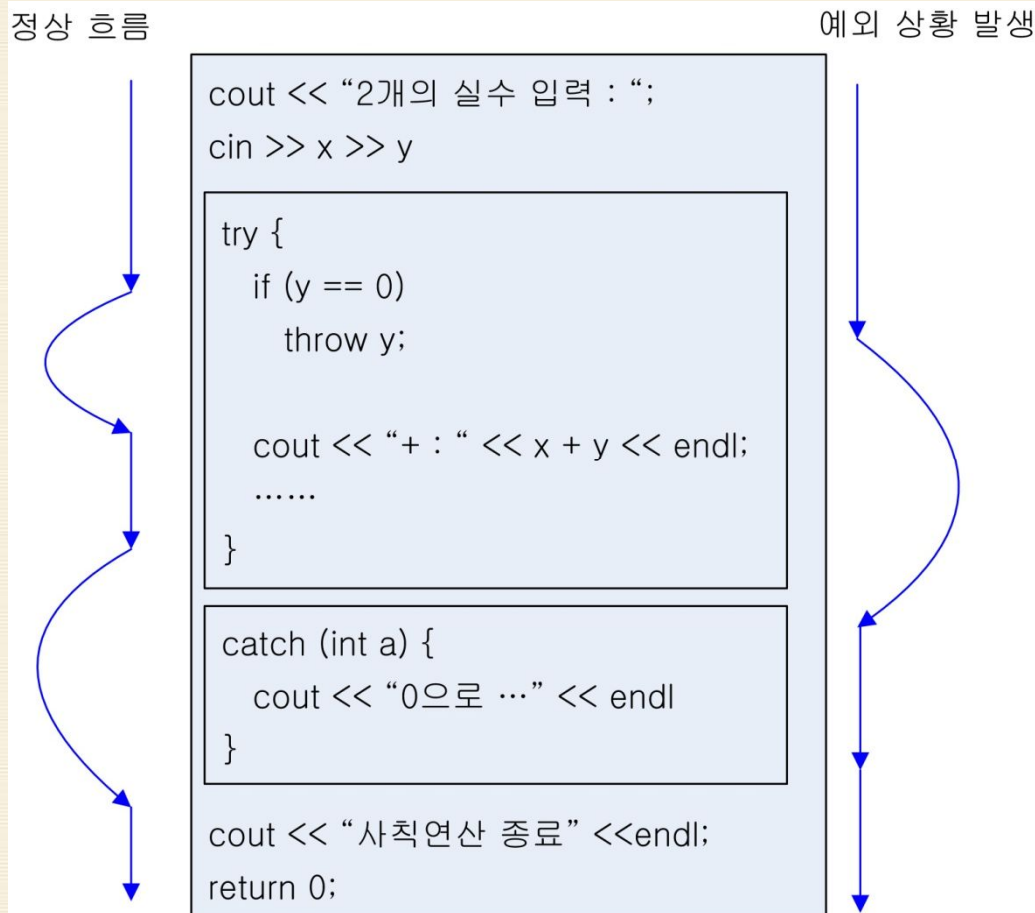
    return 0;
}
```

예외 호출  
→ 함수 호출과 유사



### 3. 예외 처리 구문

#### ▣ 예외 처리 구문의 수행 흐름



**throw** 문 없이 자동으로  
예외를 감지할 수는 없을까?  
없음!

→ 예외 처리 클래스를  
라이브러리로 만든다면  
사용자 측면에서는  
자동으로 감지하는  
것처럼 사용 가능  
→ 5, 6절 참고

## 4. throw문과 다중 예외 처리 핸들러의 사용

### ⌘ throw문을 통한 값의 전달

- 타입 : int, char, char \*, double, 클래스 등 모든 타입 가능
- 값이 전달되지 않을 수도 있음
- 여러 개의 예외 처리 핸들러(catch문) 작성 가능 → 타입에 따라 수행
- 전달된 타입의 값을 처리할 수 있는 예외 처리 핸들러(catch문)가 없다면?  
→ terminate 함수가 수행됨 (VC++의 경우 프로그램 종료)
- 모든 타입의 값을 처리할 수 있는 예외 처리 핸들러 작성 가능  
➢ catch (...) { }

### ⌘ 다중 예외 처리 핸들러 작성 예

```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;
```



## 4. throw문과 다중 예외 처리 핸들러의 사용

### ▣ 다중 예외 처리 핸들러 작성 예 (계속)

```
try {
    if (y == 0)
        throw 1;

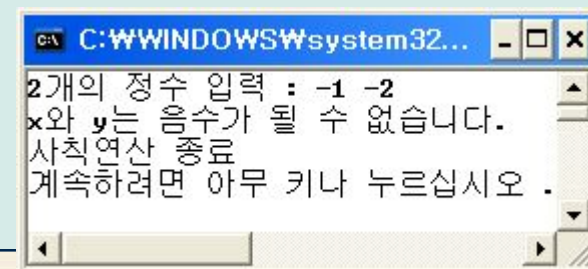
    if (x <= 0 || y <= 0)
        throw 1.0;

    cout << "+ : " << x + y << endl;
    cout << "- : " << x - y << endl;
    cout << "* : " << x * y << endl;
    cout << "/" : " << x / y << endl;
}
catch (int a) {                // 예외 처리 핸들러 : int형
    cout << "0으로 나눌 수는 없습니다." << endl;
}
catch (double a) {             // 예외 처리 핸들러 : double형
    cout << "x와 y는 음수가 될 수 없습니다." << endl;
}
catch (...) {                  // 예외 처리 핸들러 : 모든 타입
    cout << "모든 throw문을 수용할 수 있는 예외처리 핸들러입니다" << endl;
}

cout << "사칙연산 종료" << endl;

return 0;
}
```

\* 주의 : **catch** 블록 선택 방식  
코드 상의 순서대로 부합  
여부 판단  
→ **catch (...)** 블록을 먼저  
위치시킨다면?  
뒤에 있는 블록은 수행  
불가  
→ 사실은 컴파일 에러  
→ 5절 예외 처리 클래스 참고



## 5. 예외 처리 클래스

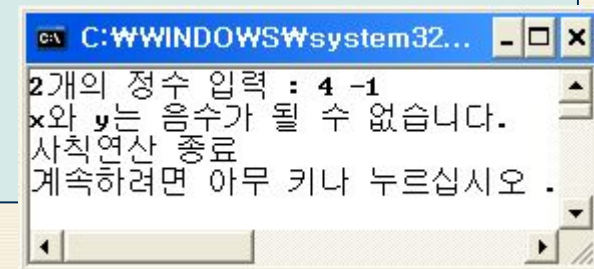
### ⌘ throw 문을 통한 클래스 객체의 전달

```
class CDivideZero {
public :
    void What() { cout << "0으로 나눌 수는 없습니다." << endl; }
};

class CNegativeNumber {
public :
    void What() { cout << "x와 y는 음수가 될 수 없습니다." << endl; }
};

int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;
```



## 5. 예외 처리 클래스

### # throw 문을 통한 클래스 객체의 전달 (계속)

```
try {
    if (y == 0)
        throw CDivideZero();    // CDivideZero 임시 객체 전달

    if (x <= 0 || y <= 0)
        throw CNegativeNumber(); // CNegativeNumber 임시 객체 전달

    cout << "+ : " << x + y << endl;
    cout << "- : " << x - y << endl;
    cout << "* : " << x * y << endl;
    cout << "/" : " << x / y << endl;
}
catch (CDivideZero a) {        // CDivideZero 객체를 값에 의한 전달로 받음
    a.What();
}
catch (CNegativeNumber a) {    // CNegativeNumber 객체 받음
    a.What();
}
catch (...) {
    cout << "모든 throw문을 수용할 수 있는 예외처리 핸들러입니다" << endl;
}

cout << "사칙연산 종료" << endl;
return 0;
```

수행 내용 동일

→ 하나의 예외 처리 핸들러로 수행 가능?

→ **CDivideZero**와 **CNegativeNumber**의  
base 클래스를 만든다면...!

## 5. 예외 처리 클래스

### ✦ CDivideZero, CNegativeNumber의 base 클래스 만들기

#### ■ CMyException

```
class CMyException {    // CDivideZero와 CNegativeNumber의 base 클래스
public :
    virtual void What() = 0; // 추상 클래스로 만듦. base 클래스 역할만.
};

class CDivideZero : public CMyException {
public :
    void What() { cout << "0으로 나눌 수는 없습니다." << endl; }
};

class CNegativeNumber : public CMyException {
public :
    void What() { cout << "x와 y는 음수가 될 수 없습니다." << endl; }
};

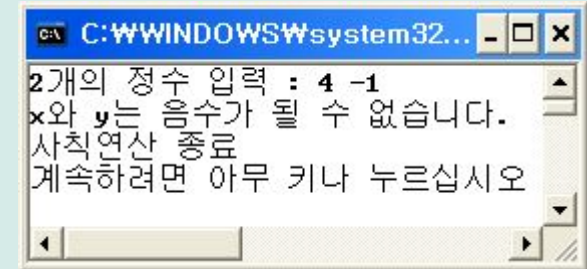
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;
```

## 5. 예외 처리 클래스

### # CDivideZero, CNegativeNumber의 base 클래스 만들기 (계속)

```
try {  
    if (y == 0)  
        throw CDivideZero();  
  
    if (x <= 0 || y <= 0)  
        throw CNegativeNumber();  
  
    cout << "+ : " << x + y << endl;  
    cout << "- : " << x - y << endl;  
    cout << "* : " << x * y << endl;  
    cout << "/" : " << x / y << endl;  
}  
catch (CMyException &a) {    // 이제는 CMyException이 둘 다 받을 수 있음  
    a.What();  
}  
catch (...) {  
    cout << "모든 throw문을 수용할 수 있는 예외처리 핸들러입니다" << endl;  
}  
  
cout << "사칙연산 종료" << endl;  
  
return 0;  
}
```



## 5. 예외 처리 클래스

✦ 다음 프로그램 코드의 문제점은 무엇일까?

- catch 블록 선택 방식에 주의

```
catch (CMyException &a) { // 이제는 CMyException이 둘 다 받을 수 있음
    a.What();
}
catch (CDivideZero &a) { // 0으로 나누기 예외는 별도 처리를 원함
    a.What();
}
catch (...) {
    cout << "모든 throw문을 수용할 수 있는 예외처리 핸들러입니다" << endl;
}
```



## 6. throw문의 전달과 응용

### ※ 중첩 try ~ catch 블록

```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;

    try {
        if (y == 0)
            throw 1;

        try {
            if (x <= 0 || y <= 0)
                throw 1.1;

            cout << "+ : " << x + y << endl;
            cout << "- : " << x - y << endl;
            cout << "* : " << x * y << endl;
            cout << "/" : " << x / y << endl;
        }
        catch (double a) {
            cout << "x와 y는 음수가 될 수 없습니다." << endl;
        }
    }
    catch (int a) {
        cout << "0으로 나눌 수는 없습니다." << endl;
    }

    cout << "사칙연산 종료" << endl;

    return 0;
}
```

해당 try 블록과 match되는 catch문이 값을 수용하지 못한다면 → 외부에 있는 try ~ catch 블록으로 전달됨

함수 내에 throw문을 처리할 수 있는 catch문이 없다면 → 함수를 호출한 함수로 throw문이 전달됨

// int 값 전달

// 중첩 try 문

// double 값 전달

// double 값 수용

// int 값 수용

## 6. throw문의 전달과 응용

### ※ 함수로부터 throw문이 전달되는 예

```
bool CheckNegative(int x, int y)
{
    if (x <= 0 || y <= 0)
        throw 1.1;
    return true;
}
```

예외는 어디로 전달되는가?

```
int main(void)
{
    int x, y;

    cout << "2개의 정수 입력 : ";
    cin >> x >> y;

    try {
        if (y == 0)
            throw 1;

        CheckNegative(x, y); // 함수 호출

        cout << "+ : " << x + y << endl;
        cout << "- : " << x - y << endl;
        cout << "* : " << x * y << endl;
        cout << "/" : " << x / y << endl;
    }
    catch (int a) {
        cout << "0으로 나눌 수는 없습니다." << endl;
    }
    catch (double a) {
        cout << "x와 y는 음수가 될 수 없습니다." << endl;
    }

    cout << "사칙연산 종료" << endl;

    return 0;
}
```

## 6. throw문의 전달과 응용

### ⌘ 예외를 처리하는 클래스 + throw문의 전달

- 라이브러리를 만드는 프로그래머 : 라이브러리의 문제점 인식
  - throw 문 작성 가능, 그러나 예외를 어떻게 처리할지는 의문?
- 라이브러리를 사용하는 프로그래머 : 예외 상황에 대한 처리 책임
  - catch 블록 작성 가능

### ⌘ 다음 예에서 CCalc 클래스와 예외 처리 클래스들을 라이브러리로 생각해 보라!

```
class CMyException {
public :
    virtual void What() = 0;
};

class CDivideZero : public CMyException {
public :
    void What() { cout << "0으로 나눌 수는 없습니다." << endl; }
};

class CNegativeNumber : public CMyException {
public :
    void What() { cout << "x와 y는 음수가 될 수 없습니다." << endl; }
};
```

## 6. throw문의 전달과 응용

```
class CCalc {
private :
    int x, y;


public :
    void Input() {
        cout << "2개의 정수 입력 : ";
        cin >> x >> y;
        if (y == 0)
            throw CDivideZero();
        if (x <= 0 || y <= 0)
            throw CNegativeNumber();
    }
    void Output() {
        cout << "+ : " << x + y << endl;
        cout << "- : " << x - y << endl;
        cout << "* : " << x * y << endl;
        cout << "/ : " << x / y << endl;
    }
};
```

라이브러리를 사용하는 프로그래머  
입장에서는 **throw** 문의 사용없이  
발생 가능한 예외(타입)에 대한  
처리만 신경쓰면 됨

```
int main(void)
{
    try {
        CCalc Calc;
        Calc.Input();
        Calc.Output();
    }
    catch (CMyException &a) {
        a.What();
    }

    cout << "사칙연산 종료" << endl;

    return 0;
}
```



## 7. new 연산자의 예외 처리

- ⌘ new 연산자 : 내부적으로 함수로 수행됨
  - 메모리 할당 실패의 경우 bad\_alloc 예외 발생

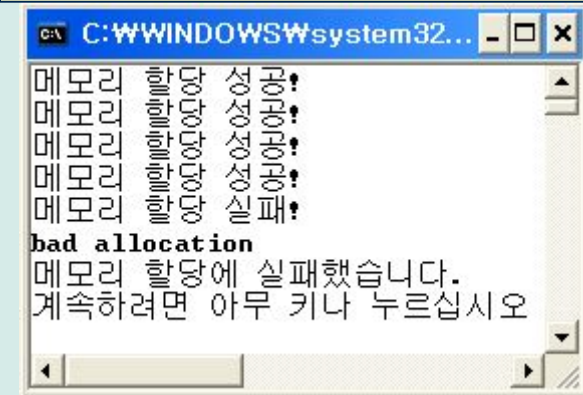
```
#include <iostream>
#include <new>
using namespace std;

int main(void)
{
    int *p;

    try {
        while (1) {
            p = new int[100000000];
            cout << "메모리 할당 성공!" << endl;
            p = NULL;
        }
    }
    catch (bad_alloc &ex) { // 메모리 할당 실패 시 bad_alloc 예외 발생
        cout << "메모리 할당 실패!" << endl;
        cout << ex.what() << endl;
    }

    if (p != NULL) {
        cout << "메모리 할당에 성공했습니다." << endl;
    }
    else {
        cout << "메모리 할당에 실패했습니다." << endl;
    }
    return 0;
}
```

### Release 모드로 실행하는 경우



## 8. 함수가 전달할 수 있는 예외의 제한

### ⌘ 특정 함수 내에서 발생(throw)할 수 있는 예외의 종류 제한

- `bool CheckNegative(int x, int y) throw(int, double);`
  - `CheckNegative` 함수 내에서는 `int`형, `double`형 값에 대한 throw문 발생 가능
- `bool CheckNegative(int x, int y) throw();`
  - `CheckNegative` 함수 내에서는 예외를 발생시킬 수 없음

### ⌘ 지정한 타입 이외의 타입에 대한 예외 발생 시

- `unexpected` 함수 수행

### ⌘ 현재 VC++에서는

- 예외의 제한 문법 및 내용을 인식하고는 있지만 컴파일 시 단순히 경고만 발생 → 실행은 가능