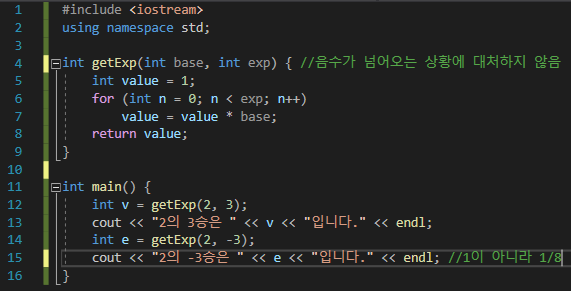
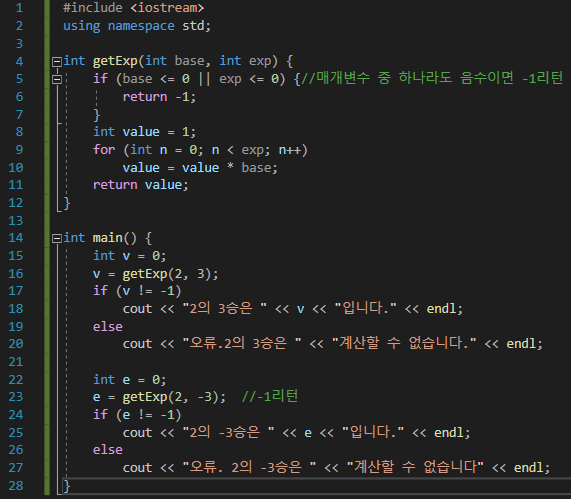
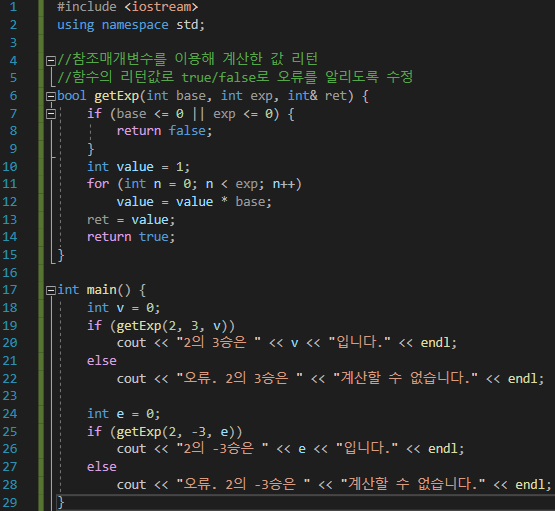
13.1 실행오류와 오류처리

실행오류의 종류와 원인 개발자의 논리가 잘못된 경우

예외에 대한 대책을 준비하지 않는 경우

예외상황에 대한 대처가 없는 코드 사례 예제13-1 

조건문과 리턴값을 이용하는 전형적인 오류 처리 예제13-2 

리턴값과 참조매개변수를 이용한 오류 처리 예제13-3 

13.2 예외와 예외 처리

예외란? 프로그램의 오동작이나 결과에 영향을 미치는 예상치 못한 상황발생

예외가 발생해도 소프트웨어가 오동작하지 않도록 오류탐지와 처리코드를 포함시킴

C++ 예외 처리 운영체제 하드웨어나 시스템 자원을 다루는 중 발생하는 예외를 알려줌

예외 처리 기본 방식, try-throw-catch 형식 try {

예외 발견시, throw{}

예외 발견시, throw{}

}

catch (처리할 예외 피라미터 선언) { 예외처리문 }

catch (처리할 예외 피라미터 선언) { 예외처리문 }

try블록 예외가 발생할 소지가 있는 문장들은 try{ }블록으로 묶고 예외를 처리할 catch(){ }블록을 바로 연결해 서언

예외 탐지시 throw문 실행 🡪 나머지 try{ } 코드들은 실행 안 됨

throw문 try{ }안에서 실행되는 문

예외값을 던짐 던져진 예외값은 catch() { }에서 처리됨

예외값의 타입과 일치하는 catch(){}블록 실행됨

catch(처리할 예외 파라미터 선언) { } () 안에서 예외파라미터 선언

함수 매개변수와 같이 예외타입 매개변수로 선언

예외값의 타입이 예외타입과 일치할 경우, 예외값이 매개변수에 전달되고 catch( ){ } 실행

예외값을 받을 catch(){ }이 선언되지 않았을 경우, 바로 프로그램 종료

예외피라미터는 한 개만 선언 가능

throw와 catch 예외구분 🡨 예외값의 타입

catch(){ } 1개는 1개의 예외타입만 처리

ex) try {

...

throw 3; //int 타입의 값 3을 예외로 던짐

}

catch (int x) {//x에 3전달

...

}

ex) try {

...

throw 3.5; //double 타입의 값 3을 예외로 던짐

}

catch (double x) {//x에 3.5전달

...

}

ex) try {

...

throw "음수 불가능"; //char\* 타입 예외를 던짐

}

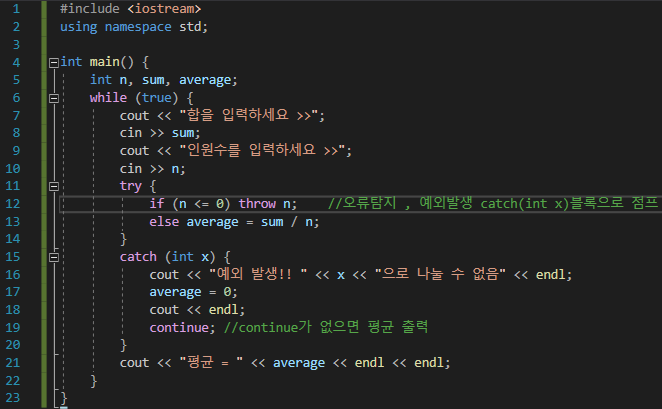
catch (const char\* x) {//x에 전달

...

}

try-throw-catch의 예외 처리 과정 제어흐름 알아보기 예외발생 O try 🡪 catch 🡪 원래 실행흐름

예외발생 X try 🡪 원래 실행흐름

예제13-4 

check time 1. try, throw, catch

2. (1) 예외발생X, 125

2. (2) 예외발생O, 0

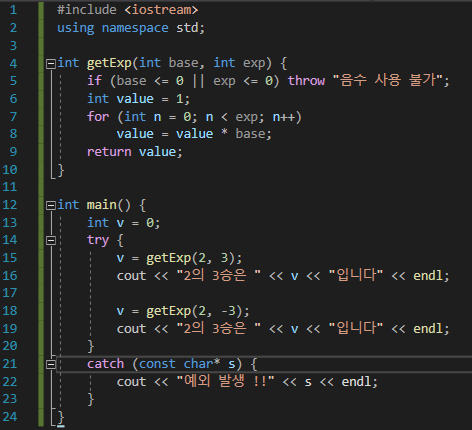
13.3 예외 처리에 대한 자세한 설명

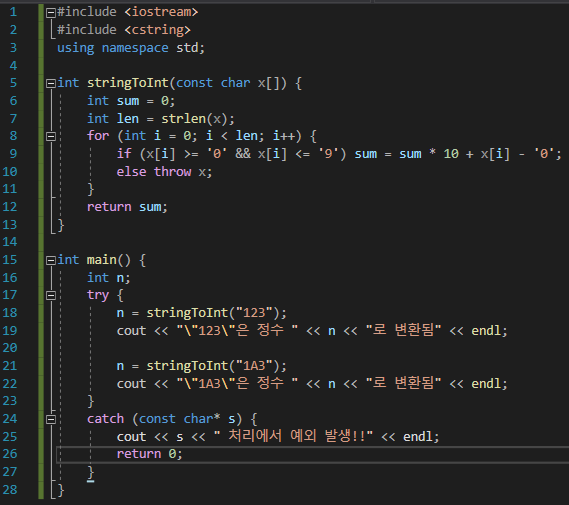
하나의 try{ }에서 다수의 catch(){ } 연결 하나의 try 블록에 여러 개의 catch 블록을 연결해, 여러 타입의 예외를 처리할 수 있음

예외 타입과 일치하는 catch블록이 실행됨

함수를 포함하는 try{ } = try{ }에서 호출한 함수가 예외를 던지는 경우

try{ }에서 호출당하는 함수가 예외를 던지면 catch(){ } 점프

예제13-5 

예제13-6 

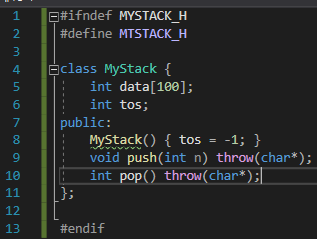
예외를 발생시키는 함수의 선언 throw문을 가지고 있는 함수 🡪 함수선언문에 예외 발생 명시 가능

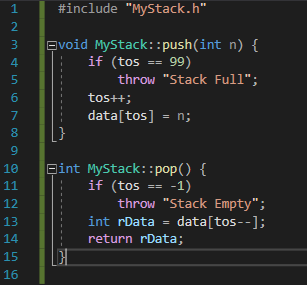
형식 함수반환형 함수이름(매개변수1, 매개변수2) throw(예외타입1, 예외타입2) {…}

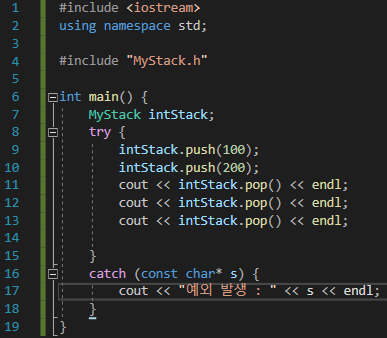
ex) double valueAt(double\* p, int index) throw(int, char\*) {}

장점 프로그램의 작동을 명확히 한다 throw()에 선언되지 않은 예외가 발생하면 프로그램을 중지하도록 컴파일

예외와 관련된 프로그램의 가독성을 높인다

예제13-7 MyStack.h 

MyStack.cpp 

Main.cpp 

어떤 예외타입이든 포착하는 catch(…){} 예외 타입 대신 …를 넣으면, 모든 예외 포착가능 but 반드시 마지막 catch()로만 사용해야 함

중첩 try{ } try{ }내에 다른 try{ } 중첩 가능

안쪽 try{ }에서 던진 예외를 처리할 catch(){ }이 없으면 바깥쪽 try{}과 연결된 catch(){ }에 예외 전달

ex) try {

throw 3; //1실행

try {

throw "abc"; //2실행

...

throw 5; //3실행

}

catch (int inner) { ... } //3

}

catch (const char\* s) { ... } //2

catch (int outer) { ... } //1

throw 사용 시 주의 사항 throw문의 위치 throw문은 항상 try{}안에서 실행되어야 함

그렇지 않을 경우, 컴파일러가 abort() 호출 🡪 프로그램 종료

예외를 처리할 catch(){}이 없으면 프로그램 종료

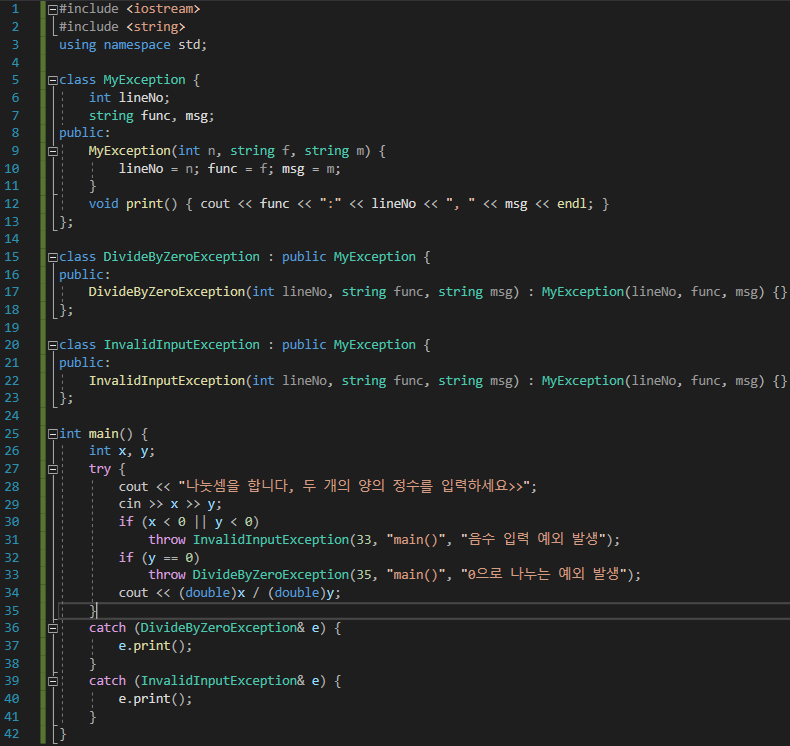
catch(){}내에도 try{}, catch(){} 선언 가능

check time 1. 컴파일 오류 – 2

강제종료 - 3

13.4 예외 클래스 만들기

예외 값으로 클래스를 사용 🡪 catch() { }에 더 많은 정보를 전달할 수 있음

예제13-8 

13.5 C++코드와 C코드의 링킹

C/C++ 컴파일러의 이름 규칙 이름규칙 목적코드를 만들 때, 변수∙함수∙클래스의 이름을 변형하는 규칙

컴파일러마다 다름

C 컴파일러의 이름규칙 함수이름 앞에 \_를 붙임

매개변수 존재, 리턴타입 고려 X 🡨 C언어에서 함수중복이 불가능한 이유

ex) int main() 🡪 \_main

int f() 🡪 \_f

C++컴파일러의 이름규칙 매개변수개수∙타입, 리턴타입 고려 🡪 중복함수들도 구분됨

ex) int f(int x, int y) 🡪 ?f@@YAHHH@Z

int f(int x) 🡪 ?f@@YAXH@Z

int main() 🡪 \_main

main은 항상 \_main

C++ 프로그램에서 C 함수 호출 시 링크 오류가 발생하는 경우 C언어로 작성된 함수를 C++프로그램에서 호출 시 링크 오류 발생 🡨 이름규칙이 다르기 때문

ex) f.c와 main.cpp를 링크할 때, f.c의 이름규칙과 main.cpp의 이름규칙이 다름

f.c int f(int x, int y)

main.cpp int f(int x, int y)

정상적인 링킹, extern “C” C언어의 이름규칙으로 컴파일하도록 지시

ex) f.c int f(int x, int y)

main.cpp extern “C” int f(int x, int y)

묶어서 여러 개의 함수를 동시에 C언어의 이름규칙으로 컴파일하도록 지시 가능

ex) extern “C” {

int f(int x, int y);

void g();

char s(int []);

}

Quiz 1. 1

2. extern “C” { #include “cgraphic.h” }