# 창의적 소프트웨어 설계 실습 문제 13 – hw13-1

**제출 기한**

12월 11일 화 11:59 PM

1. hw13-1(mkdir hw13-1)라는 폴더를 만들고 GitLab에 push
2. hw13-1 디렉토리에 CMakeLists.txt과 각 문제에서 요구하는 파일들을 작성
3. **‘cmake’ 와 ‘make’명령을 수행하여 숙제가 모두 빌드**
4. 최종 버전을 GitLab에 commit
5. **시간과 파일명, 입력과 출력 방식 반드시 지키기**

**본 실습에서는c++에서 exceptions을 다루는 법을 숙지한다.**

* 1. **아래 프로그램에서 exception handling을 하시오. 생성자에서 bad allocation이 발생(n≤0)하면 exception을 throw하고, catch 문에서는 “caught in the main” 을 출력 하도록 하시오. 특히, try 블록 내부에서 생성된 오브젝트는 exception 이 발생하게 되면 소멸자가 호출되므로 소멸자를 잘 구현하고, 또, 어떤 오브젝트들의 생성자와 소멸자가 호출되는지를 잘 확인하시오.**

**class A**

**{**

**public:**

**A(int n)**

**{**

**//implement something here**

**cout << "ID=" << n << ": constructed\n";**

**n\_ID = n;**

**data = new int[n];**

**}**

**~A()**

**{**

**cout << "ID=" << n\_ID << ": destroyed\n";**

**//implement something here**

**}**

**private:**

**int\* data = NULL;**

**int n\_ID;**

**};**

**int main(){**

**try{**

**A a(3);**

**A b(2);**

**{**

**A c(1);**

**A d(0);**

**A e(-1);**

**}**

**}**

**//implement something here**

**return 1;**

**}**

|  |
| --- |
| **예시 출력:**  **ID=3: constructed**  **ID=2: constructed**  **ID=1: constructed**  **ID=1: destroyed**  **ID=2: destroyed**  **ID=3: destroyed**  **caught in the main** |

* 1. **아래 코드와 같이 object ‘a’를 throw 하면, try scope가 끝날 때 ‘a’의 소멸자가 호출되기 때문에, ‘a’는 내부적으로 복사가 되어 사용된다(i.e., “Objects Thrown as Exceptions Are Always Copied”). 다음 프로그램을 수행하였을 때, 예시 출력과 같이 출력이 되도록 프로그램을 수정 하시오 (Hint: object가 제대로 copy되기 위해서는 무엇이 필요할까?)**

**int data\_size = 5;**

**class A{**

**public:**

**A(){**

**data = new int[data\_size];**

**for(int i = 0; i < data\_size; i++)**

**data[i] = i;**

**cout << "constructed\n";**

**}**

**~A()**

**{**

**for(int i = 0; i < data\_size; i++)**

**data[i] = 0;**

**delete[] data;**

**data = NULL;**

**cout << "destroyed\n";**

**}**

**private:**

**int\* data = NULL;**

**friend ostream& operator <<(std::ostream& os, const A& a);**

**};**

**ostream& operator <<(std::ostream& os, const A& a)**

**{**

**for(int i=0; i<data\_size; i++)**

**os << a.data[i] << " ";**

**return os;**

**}**

**int main()**

**{**

**try**

**{**

**A a;**

**cout << a << endl;**

**throw a;**

**}**

**catch(A& a)**

**{**

**cout << "err. handled\n";**

**cout << a << endl;**

**}**

**return 1;**

**}**

|  |
| --- |
| **예시 출력:**  **constructed**  **0 1 2 3 4**  **destroyed**  **err. handled**  **destroyed**  **0 1 2 3 4** |

* 1. **상속 관계가 주어질 때, chained handler(여러 개의 catch 문이 연속하여 나타남) 를 사용하여, 예시와 같이 프로그램이 출력을 하도록 하시오. 단, catch문 내부에서만 출력(cout)을 하도록 하시오.**

**#include <stdlib.h>**

**#include <time.h>**

**class A**

**{**

**};**

**class B : public A**

**{**

**};**

**class C : public B**

**{**

**};**

**int main(){**

**int n;**

**cout << "input num(0~2):";**

**cin >> n;**

**try{**

**if(n == 0)**

**throw new A;**

**else if(n == 1)**

**throw new B;**

**else if (n ==2)**

**throw new C;**

**else**

**throw string(“invalid input”);**

**}**

**//implement here**

**return 1;**

**}**

|  |
| --- |
| **입력 값 n에 따라 결과가 다음과 같다.**  **n=0인 경우 예시 출력: “throw new A has been called”**  **n=1인 경우 예시 출력: “throw new B has been called”**  **n=2인 경우 예시 출력: “throw new C has been called”** |

* 1. **생성자와 소멸자 내부에서 각각 임의의 exception을 발생(throw)시키는 간단한 예제 프로그램을 작성하고, 이 프로그램에서 왜 문제가 발생하는지를 분석하시오.**
  2. **특정 catch handler가 exception을 처리할 수 없을 때, 상위 try 구문의 catch handler에게 rethrow를 할 수 있다. rethrow는 throw를 두 번 하는 것과는 다른데 차이점을 확인해 보자. 먼저 생성자, 복사생성자, 소멸자가 있는 임의의 클래스 A를 설계하고, 다음 프로그램을 실행 시켜보자. rethrow를 하는 경우와 throw를 두 번 하는 경우에 생성자, 복사 생성자, 소멸자가 몇 번 씩 호출 되는지를 확인하여라.**

**int main() {**

**try{**

**try{**

**A a;**

**throw a;**

**}**

**catch(A &a)**

**{**

**#if 1//rethrow, *아래 throw twice구문도 실행시켜 볼 것***

**cout << "rethrow" << endl;**

**throw;**

**#else//throw twice**

**cout << "throw twice" << endl;**

**throw a;**

**#endif**

**}**

**}**

**catch(A &a)**

**{**

**cout << "exception handled" << endl;**

**}**

**return 1;**

**}**