제17장

예외 처리

학습 목표

- 예외 처리의 기본
  - 예외 클래스 정의
  - 여러 가지를 던지고 잡기
  - 예외 명세
- 예외 처리를 위한 프로그래밍 기법
  - 예외를 던지는 시기
  - 예외 클래스 계층구조

소개

- 전형적으로 개발하는 방법:
  - 계획된 것만 작성한다고 가정하고 프로그램을 작성
  - 그리고 나서, 예외적인 상황에 대해 주의
- C++ 예외 처리 기능
  - 예외적인 상황을 처리
  - 흔치 않은 일이 일어났다는 신호를 보내는 매커니즘
  - 다른 곳에 예외를 다루는 코드를 둠

Toy 예제

- 가정 : 우유가 드물게 바닥나는 상황:

```
cout << "Enter number of donuts:";
cin >> donuts;
cout << "Enter number of glasses of milk:";
cin >> milk
dpg = donuts/static_cast<double>(milk);
cout       << donuts << "donuts.\n";
                     << milk << "glasses of milk.\n";
                     << "You have " << dpg
                     << "donuts for each glass of milk.\n";
```

- 기본 코드는 절대 우유가 바닥나지 않는다고 가정함

Toy 예제 if-else

- Notice: 만약 우유가 없다면 → 0으로 나누는 에러!
- 프로그램은 우유가 바닥나는 특수한 상황을 수용해야 함
  - 단순 if-else 구조 이용 가능:

```
if (milk <= 0)
    cout << "Go buy some milk!\n";
else
{…}
```

- Notice: 여기엔 예외 처리가 없다

예외 처리를 가지는 Toy 예제: 디스플레이 17.2 예외 처리를 이용한 것

```
 9        try
10        {
11             cout << "Enter number of donuts:\n";
12             cin >> donuts;
13             cout << "Enter number of glasses of milk:\n";
14             cin >> milk;
15
16             if (milk <= 0)
17                  throw donuts;

18             dpg = donuts / static_cast<double>(milk);
19             cout << donuts << " donuts.\n"
20                  << milk << " glasses of milk.\n"
21                  << "You have " << dpg
22                  << " donuts for each glass of milk.\n";
23        }
24        catch(int e)
25        {
26             cout << e << " donuts, and No Milk!\n"
27                  << "Go buy some milk.\n";
28        }
```

Toy 예제 논의

- try와 catch 키워드 사이의 코드
  - 원본 버전과 같은 코드,

    if 구문을 단순화한 것을 제외하고:

    ```
    if (milk <= 0)
        throw donuts;
    ```
  - 더욱 명백한 코드
  - 만약 우유가 없다면 → 예외적인 무언가를 수행
- "예외적인 무언가"는 catch 키워드 이후에 제공됨

Toy 예제 try-catch

- 예외 처리의 기본 방법은 try-throw-catch
  - 일반적인 상황과 예외적인 상황을 분리해서 제공
- try 블록: 일반적인 상황을 처리
  - 순조롭게 진행될 때 해야 할 코드를 포함
- throw: try 블록 안에서 예외 발생
  - throw 키워드는 다음에 예외 유형이 따름
  - 예외 던지기라 함
- catch 블록: 예외적인 상황을 처리
  - 던져진 "예외를 잡아 " 처리하는 부분
    - try 블록 종료 후 catch 블록으로 제어 전달

```
try
{
        일단_시도할_코드
        if (예외적인 상황 발생)
            throw donuts;
        나머지_코드
}
```

```
catch(int e)
{
    cout << e << " donuts, and no milk!\n";
        << " Go buy some milk.\n";
}
```

catch 블록 및 블록 매개변수

- cath 블록
  - int형 매개변수를 가진 함수 정의처럼 보임!
    - 함수가 아니지만 유사하게 작동
    - 함수 호출과 같이 던짐

- Recall: catch(int e)
  - "e"는 catch-블록 매개변수라 함
    - 각 catch 블록은 하나의 catch-블록 매개변수를 가짐
  - 2가지 일을 수행:
    - 블록이 잡을 수 있는 값이 어떤 형인지 형 이름을 명시한다.
    - 던진 값을 잡았을 때 그 값의 이름을 제공;
      이 값을 이용해서 할 일을 코드로 작성

예외 클래스 정의

- throw 문은 어떤 형이 값도 던질 수 있음
- 예외 클래스
  - 던져질 정보를 가지는 객체를 포함
  - 각 가능한 예외적인 상황을 식별하는 다른 형을 가질 수 있음
  - 그냥 클래스임
    - 어떻게 사용되느냐에 따라 예외 클래스일 수도 아닐 수도

```
class NoMilk
{
public:
        NoMilk() { }
       NoMilk(int howMany) : count(howMany) { }
        int getcount() const { return count; }
private:
        int count;
};
```

```
throw NoMilk(donuts);
```

여러 가지를 던지고 잡기

- try-블록은 여러 형, 다른 형의 예외 값을 던질 수 있음
- 물론, 오직 하나의 예외만 던져짐
  - throw문으로 인해 try-블록이 끝나므로
- 그러나 여러 가지 형이 던져질 수 있음
  - 각 catch 블록은 오직 한 유형만 잡는다
  - try-블록 다음에 여러 개의 catch-블록을 사용
    - 던져진 모든 가능한 예외를 잡기 위함

```
class NegativeNumber
{
public:
    NegativeNumber( ){}
    NegativeNumber(string theMessage): message(theMessage) {}
    string getMessage( ) const { return message; }
private:
    string message;
};

class DivideByZero
{};
```

```
try
{
    cout << "How many pencils do you have?\n";
    cin >> pencils;
    if (pencils < 0)
         throw NegativeNumber("pencils");

    cout << "How many erasers do you have?\n";
    cin >> erasers;
    if (erasers < 0)
        throw NegativeNumber("erasers");

     if (erasers != 0)
        ppe = pencils/static_cast<double>(erasers);
    else
        throw DivideByZero( );

    cout << "Each eraser must last through "
         << ppe << " pencils.\n";
}
catch(NegativeNumber e)
{
   cout << "Cannot have a negative number of "
        << e.getMessage( ) << endl;
}
catch(DivideByZero)
{
   cout << "Do not make any mistakes.\n";
}

cout << "End of program.\n";
return 0;
}
```

잡기

- catch 블록의 순서는 중요
- catch-블록은 try-블록 이후에 순서대로 시도
  - 첫 번째로 일치하는 것을 처리!
- Consider:

  catch (…) { }
  - 디폴트 예외 처리라 함
  - 어떠한 예외라도 잡음
  - 나머지 명시된 예외 이후에 디폴트를 위치하도록 하라!
    - 그렇지 않으면, 다른 것들이 절대 잡을 수 없을 것이다.

사소한 예외 클래스

- Consider:

  class DivideByZero

  { }
- 멤버 변수도 멤버 함수도 없다 (디폴트 생성자 외엔)
- 아무 것도 없지만 이름만으로도 충분
  - 예외 값에 대해 할 일이 없다는 것
  - 단순히 catch 블록에 도달하게 하는데 만 사용
  - catch 블록 매개변수 생략 가능

함수에서 예외 던지기

- 함수가 예외를 던질 수 있음
- 호출자는 서로 다른 반응을 보일 수도 있음
  - 어떤 것은 프로그램을 종료하길 원하고
  - 어떤 것은 무언가 다른 일을 해야 할 수도 있음
- 호출 함수의 try-catch-block에서 예외를 잡는 것이 이치에 맞음
  - try-블록안에서 함수 호출
  - try-블록 이후에 catch-블록에서 처리

함수에서 예외 던지기 예제

- Consider:
  - safeDivide() 함수는 DividebyZero 예외 발생 던진다.
  - 호출자의 catch-블록에서 처리됨

```
    try
    {
       quotient = safeDivide(numerator, denominator);
    }
    catch(DivideByZero)
    {
         cout << "Error: Division by zero!₩n"
              << "Program aborting.₩n";
         exit(0);
    }

    cout << numerator << "/" << denominator
         << " = " << quotient << endl;

    cout << "End of program.₩n";
    return 0;
}


double safeDivide(int top, int bottom) throw (DivideByZero)
{
    if (bottom == 0)
        throw DivideByZero( );

    return top/static_cast<double>(bottom);
}
```

예외 명세

- 함수가 예외를 잡지 않는다면
  - 예외가 던져질 수 있다는 것을 사용자에게 경고해 줘야 함
  - 그러나 잡지는 않을 것이다!

- 이것의 예로 :

  double safeDivide(int top, int bottom) throw (DividebyZero);
  - "예외 명세" 또는 "throw 목록"이라 함
  - 함수 선언과 정의 둘 다 있어야 함
  - 나열한 예외 형은 모두 정상적으로 처리됨
  - 만약 예외 명세가 없다면 → 모든 예외의 형이 열거되어 있는 것과 똑같이 고려됨

throw 목록

- 함수에서 예외가 던져졌지만 예외 명세에 없다면:
  - 에러가 아님 (컴파일 또는 실행 시에)
  - unexpected() 함수가 자동적으로 호출됨
    - 종료하기 위한 기본 행동을 수행. 다르게 작동하도록 수정도 가능.
  - catch 블록을 찾지 못한 것과 같은 결과

- 예외 명세 예제
  - void someFunction() throw(DividebyZero, OtherException);
    //예외 형은 DividebyZero 또는 OtherException
    //정상적으로 다루어짐. 나머지 예외들은 모두 unexpected() 호출
  - void someFunction() throw ();
    // 예외 목록이 비어있다. 그러므로 모든 예외가 unexpected()를 호출
  - void someFunction();
    // 모든 예외가 정상적으로 취급된다

예외를 던지는 시기

- 전형적으로 던지기와 잡기를 분리
  - 분리된 함수에서
- 던지기 함수:
  - 함수 정의에 throw 문을 포함
  - 함수 선언과 정의 둘 다에서 예외 명세에 예외를 열거
- 잡는 함수:
  - 다른 함수 안에서, 심지어는 다른 파일에 있는 다른 함수 안에서

예외의 남용

- 예외는 제어의 흐름을 바꿈
  - 옛날 "goto" 분기문과 유사
  - 무제한적 제어 흐름
- 가급적 사용하지 않아야 함
- 좋은 규칙:
  - "throw"를 하고 싶다면: throw문 없이 프로그램을 어떻게 작성할 수 있는지 생각
  - 합리적인 대안이 있다면 → 그렇게 하라

예외 클래스 계층 구조

- ex.DivideByZero class는 ArithmeticError 예외 클래스로부터 파생
  - ArithmeticError를 위한 모든 catch-블록은 또한 DivideByZero를 잡는다.
  - ArithmeticError가 예외 명세에 있다면, DividebyZero 또한 예외 명세에 있는 것으로 고려됨

이용 가능한 메모리 테스트

- 만약 메모리가 불충분하다면 new 연산자는 bad_alloc 예외를 던짐 :

```
try
{
    NodePtr pointer = new Node;
}
catch (bad_alloc)
{
    cout << "Ran out of memory!";
    // 여기에 다른 일을 할 수 있다
}
```

- <new> 라이브러리에 정의, std 네임스페이스

요약

- 예외 처리는 정상적인 경우와 예외적인 상황을 분리한다.
- try-블록에서 던져진 예외는 catch-블록에서 잡힘
  - 호출이 try-블록에 있는 함수 안에서도 예외를 던질 수 있음
  - try-블록 다음에 하나 이상의 catch-블록이 따라야
  - 처음엔 구체적인 예외를 명세하고 나머지를 열거
- 함수를 분리하여 사용하는 것이 좋음
  - 특별히 호출자가 다르게 처리하는 것을 고려한다면
- 함수에서 예외가 던져지더라도 잡지 않는다면, 예외 명세에 나타내야 함
  - 던져진 예외가 절대 잡히지 않는다면 →프로그램 종료
- 예외를 남용해선 안 된다.
  - 무제한적인 제어 흐름