제 15 강

예외처리

예외처리

1 예외의 개념

2 C++ 언어의 예외처리 체계

3 예외처리 클래스

• 1. 예외의 개념

• (1) 예외

- 예외(exception)란?

- 프로그램 실행 도중에 **비정상적인 사건이 발생**하는 것

   비정상적인 데이터, 자원의 부족 등

- 예외 상황에 대한 적절한 대비를 하지 않으면 정상적인 프로그램 실행이 이루어지지 않음

  ⇨ 프로그램이 안정적으로 동작하도록 하려면 예외의 발생에 대비한 처리를 잘 정의해 놓을 필요가 있음

• (2) 예외가 발생하는 상황의 예 (1/4)

  • 정상적인 처리를 할 수 없는 데이터

```
1  double hmean(double a, double b)  // 조화평균
2  {
3      return 2.0 * a * b / (a + b);
4  }
```

⇨ a == -b인 경우 실행 중 프로그램 비정상 종료

• (2) 예외가 발생하는 상황의 예 (2/4)

- 정상적인 처리를 할 수 없는 데이터

```
1  double hmean(double a, double b)  // 조화평균
2  {
3    if (a == -b) {
4      cout << "나누기를 할 수 없습니다." << endl;
5      exit(EXIT_FAILURE);
6    }
7    return 2.0 * a * b / (a + b);
8  }
```

⇨ a == -b인 경우 이를 알리고, 프로그램 강제 종료

• (2) 예외가 발생하는 상황의 예 (3/4)

- 프로그램이 요청하는 자원을 할당할 수 없는 경우

```
void f() {
  int *p = new(nothrow) int[1000000000];
  ......    // 할당된 메모리의 활용
}
```

⇨ 메모리 할당에 실패한 경우 실행 중 에러가 발생하여 프로그램이 비정상적으로 종료함

• (2) 예외가 발생하는 상황의 예 (4/4)

- 프로그램이 요청하는 자원을 할당할 수 없는 경우

```
void f() {
  int *p = new(nothrow) int[1000000000];
  if (!p) {   // p가 nullptr이면
    cerr << "메모리 할당 오류" << endl;
    exit(EXIT_FAILURE);
  }
  ......
}
```

➪ 메모리 할당이 이루어지지 않은 경우 new(nothrow)는 nullptr를 반환하므로, 이를 이용하여 메모리 할당 오류를 검사함

예외처리

1 예외의 개념

2 C++ 언어의 예외처리 체계

3 예외처리 클래스

• 2. C++ 언어의 예외처리 체계

• (1) C++ 언어의 예외처리 구문

• try 블록, catch 블록, 그리고 throw 문장으로 구성

```
RetType1 someFunction()
{
  try {
    // 예외가 발생할 수 있는 부분
    someDangerousFunction();
  }
  catch (eClass e) {
    // 발생한 예외를 처리하는 부분
    exceptionProcRtn();
  }
  ......
}
```

```
RetType2 someDangerousFunction()
{
  if (예외검출조건)
    throw eObj; // 예외 발생을 알림
  else
    ......   // 정상적인 처리
}
```

• (2) 예외처리의 예 - HMean.cpp (1/2)

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  double hmean(double a, double b)
 5  {
 6    if (a == -b)   // 예외 검출
 7      throw "조화평균을 계산할 수 없음!";
 8    return 2.0*a*b / (a + b);
 9  }
10
11  int main()
12  {
13    double x, y, z;
14    cout << "두 수를 입력하시오 : ";
...   ......
28  }
```

• (2) 예외처리의 예 - HMean.cpp (2/2)

```
11  int main()
12  {
13    double x, y, z;
14    cout << "두 수를 입력하시오
15    while (cin >> x >> y) {
16      try {       // 예외가 발생
17        z = hmean(x, y);
18      }
19      catch (const char* s) {  // 예외처리
20        cout << s << endl;
21        cout << "다른 수를 입력하시오 : ";
22        continue;
23      }
24      cout << "조화평균 = " << z << endl;
25      cout << "다음 두 수를 입력하시오 (Q는 종료) : ";
26    }
27    return 0;
28  }
```

```
double hmean(double a, double b)
{
  if (a == -b)    // 예외 검출
    throw "조화평균을 계산할 수 없음!";
  return 2.0*a*b / (a + b);
}
```

• (3) 예외 유형별 처리

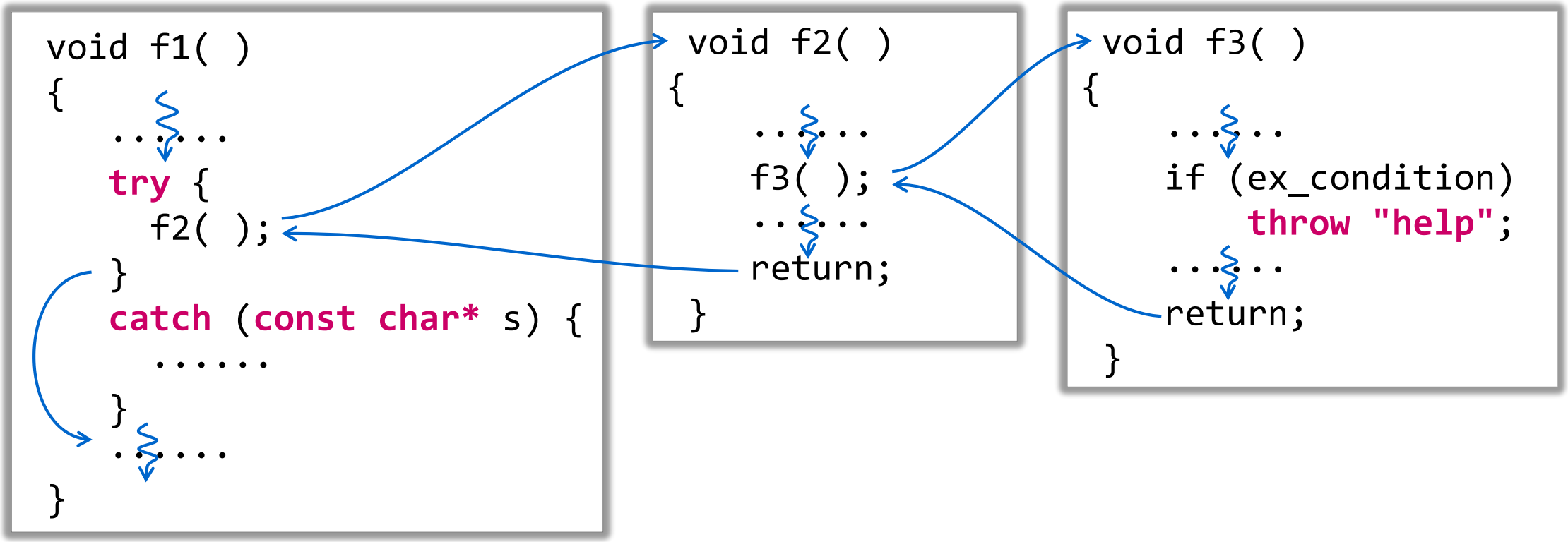
• 하나의 try 블록에 여러 개의 catch 블록 사용

```
try {
   ......  // 예외가 발생할 가능성이 있는 함수 호출
}
catch (eClass1 e) {
   ...... // 예외처리 블록1
}
catch (eClass2 e) {
   ...... // 예외처리 블록2
}
catch (...) {
   ...... // 그 외의 모든 예외 처리
}
// 다음 문장
```

throw된 예외 객체의 자료형에 맞는 매개변수가 선언된 catch 블록에서 예외를 처리함

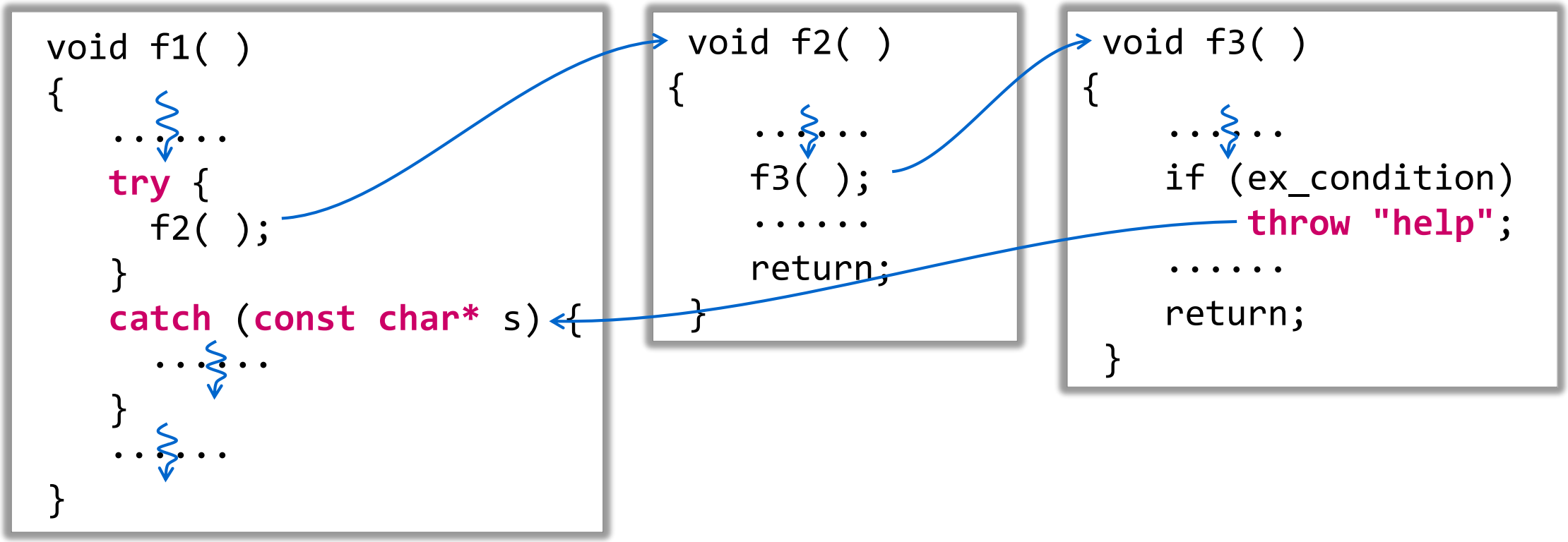
• (4) 제어의 전달 (1/2)

  • 정상적인 제어의 흐름



• (4) 제어의 전달 (2/2)

- 예외가 발생한 경우 제어의 흐름



• (5) 예외처리에 따른 자원 관리 문제 (1/5)

- 자원 소실이 가능한 상황

```
void f()
{
    int *p = new int[1000];
    for (int i=0 ; i < 1000 ; i++)
        p[i] = i;
    ......
    if (ex_condition)
        throw "exception";
    ......
    delete[] p;
}
```

⇨ throw에 명령이 실행되면
나머지 문장들은 실행되지 않음

예외를 처리할 catch블록으로 복귀할 때 f()가 호출될 때까지 거쳐온 함수들의 지역변수들은 정상적인 소멸 과정을 거침

• (5) 예외처리에 따른 자원 관리 문제 (2/5)

• 스마트 포인터의 활용

- **unique_ptr : 할당된 메모리를 한 개의 포인터만 가리킬 수 있음**
  - 다른 unique_ptr에 대입할 수 없으며, 이동 대입만 할 수 있음
  - unique_ptr가 제거되거나 nullptr를 대입하면 가리키고 있던 메모리를 반납함
- **shared_ptr : 할당된 메모리를 여러 개의 포인터로 가리킬 수 있음**
  - 다른 shared_ptr에 대입 및 이동 대입 가능
  - 포인터가 제거되거나 nullptr을 대입하는 등의 처리로 그 메모리를 가리키는 shard_ptr이 더 이상 없으면 메모리를 반납함

• (5) 예외처리에 따른 자원 관리 문제 (3/5)

- 스마트 포인터의 활용 예

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

int main() {
  unique_ptr<int> p1{ new int };
  unique_ptr<int> p2;
  *p1 = 10;
  cout << *p1 << endl;
  p2 = move(p1);  // p2 = p1;은 불가
  cout << *p2 << endl;
  p2 = nullptr;   // 가리키고 있던 메모리는 해제됨
  return 0;
}
```

• (5) 예외처리에 따른 자원 관리 문제 (4/5)

• 예외 처리로 인한 자원 소실 방지

- unique_ptr 활용

```
#include  <memory>
void f()
{
    std::unique_ptr<int[]> p { new int[1000] };
    for (int i=0 ; i < 1000 ; i++)
        p[i] = i;
    ......
    if (ex_condition)
        throw "exception";
    ......
}
```

• (5) 예외처리에 따른 자원 관리 문제 (5/5)

• 예외처리로 인한 자원 소실 방지

- vector 활용

```
#include  <vector>
void f()
{
    std::vector<int> p(1000);
    for (int i=0 ; i < 1000 ; i++)
        p[i] = i;
    ......
    if (ex_condition)
        throw "exception";
    ......
}
```

• (6) noexcept 지정자

• noexcept 함수 지정

- 함수가 예외를 일으키지 않음을 지정

예

```
template <typename T>
T max(const vector<T>& v) noexcept {
    auto p = v.begin();
    T m = *p++;
    for (; p != v.end(); p++)
        if (m < *p)  m = *p;
    return m;
}
```

예외처리

1 예외의 개념

2 C++ 언어의 예외처리 체계

3 예외처리 클래스

- 3. 예외처리 클래스
- (1) 클래스에서 예외처리 활용하기
  - 예외처리 클래스의 활용

클래스 설계시 예외처리 기능을 포함시킴으로써 객체에서 예외가 발생하였을 때 그 위치나 원인 등의 식별을 용이하게 할 수 있음

클래스 선언문 내에 예외처리 담당 클래스를 선언하여 활용함

• (2) 클래스의 예외처리 활용 예 - IntArray1.h (1/3)

```
...  ......
 6   const int DefaultSize = 10;
 7   class Array {
 8     int  *buf;
 9     int  size;
10   public:
11     Array(int s = DefaultSize);
12     virtual ~Array() { delete[] buf; }
13     int& operator[](int offset);
14     const int& operator[](int offset) const;
15     int getsize() const { return size; }
16     friend ostream& operator<<(ostream&, Array&);
17     class BadIndex {};   // exception class
18   };
```

• (2) 클래스의 예외처리 활용 예 - IntArray1.cpp (2/3)

```
...  ......
 3   #include  "IntArray1.h"
 4   using  namespace    std;
 5
 6   Array::Array(int s) : size(s)   // 생성자
 7   {
 8     buf = new int[s];
 9     memset(buf, 0, sizeof(int)*s);
10   }
11
12   int& Array::operator[](int offset)
13   {
14     if (offset < 0 || offset >= size)  // 예외조건 검사
15       throw BadIndex();          // 예외객체 생성 및 전달
16     return buf[offset];
17   }
...  ......
```

• (2) 클래스의 예외처리 활용 예 - IA1Main.cpp (3/3)

```
1   #include  <iostream>
2   #include  "IntArray1.h"
3   using  namespace  std;
4
5   int main()
6   {
7     Array arr(10);
8     try {
9       for (int i = 0; i <= 10; i++)
10        arr[i] = i;
11    }
12    catch (Array::BadIndex e) {
13      cerr << "인덱스 범위 오류" << endl;
14    }
15    cout << arr << endl;
16    return 0;
17  }
```

• (3) 예외 객체의 멤버를 통한 예외 정보 전달 (1/2)

IntArray1.h

```
class Array {
  ......
public:
  ......
  class BadIndex {
  public:
    int wrongIndex;
    BadIndex(int n)
      : wrongIndex(n) {}
  };
};
```

IntArray1.cpp

```
int& Array::operator[](int offset) {
  if (offset < 0 || offset >= size)
    throw BadIndex(offset);
  return buf[offset];
}
```

• (3) 예외 객체의 멤버를 통한 예외 정보 전달 (2/2)

IntArray1.h

```
class Array {
  ......
public:
  ......
  class BadIndex {
  public:
    int wrongIndex;
    BadIndex(int n)
      : wrongIndex(n) {}
  };
};
```

IA1Main.cpp

```
int main()
{
  ......
  try {
    ......
  }
  catch (Array::BadIndex e) {
    cout << "인덱스 범위 오류 --> "
         << e.wrongIndex << endl;
  }
};
```

• (4) exception 클래스 (1/2)

• exception 클래스란?

- C++ 언어에서 예외를 처리하기 위해 예외처리 담당 클래스의 기초 클래스로 제공하는 클래스
- 헤더파일 <exception>을 소스파일에 포함시킴
- 가상함수 what()을 멤버함수로 가지고 있음
  - 예외의 종류를 char* 형태로 반환함
  - exception의 파생 클래스에서 재정의하여 사용함

• (4) exception 클래스 (2/2)

• 예 : IntArray1.h 수정

```
#include <exception>
......
class Array {
  ......
public:
  ......
  class BadIndex : public exception {
  public:
    int wrongIndex;
    BadIndex(int n) : wrongIndex(n), exception() {}
    const char* what() const
        { return "Array Exception::"; }
  };
};
```

• (5) 예외 객체의 다시 던지기 (1/2)

• catch 블록에서 처리를 완결할 수 없는 예외의 전달

- 현 단계의 catch 블록에서 처리를 완결할 수 없는 예외에 대한 후속 처리를 할 수 있게 예외 객체를 다시 throw할 수 있음

```
class ExceptionClass
      : public exception {
  ......
};

int f(int a) {
  if (a < 0)
    throw ExceptionClass();
  ......
};
```

```
int g(int b) {
  try {
    f(b);
    ......
  }
  catch (ExceptionClass e) {
    ...... // 현 단계의 예외처리
    throw;
  }
}
```

• (5) 예외 객체의 다시 던지기 (2/2)

• catch 블록에서 처리를 완결할 수 없는 예외의 전달

- 현 단계의 catch 블록에서 처리를 완결할 수 없는 예외에 대한 후속 처리를 할 수 있게 예외 객체를 다시 throw할 수 있음

```
class ExceptionClass
      : public exception {
  ......
};

int f(int a) {
  if (a < 0)
    throw ExceptionClass();
  ......
};
```

```
int g(int b) {
  try {
    f(b);
    ......
  }
  catch (ExceptionClass e) {
    ...... // 후속 예외처리
  }
}
```

```
int h(int c) {
  try {
    g(c);
    ......
  }
  catch (ExceptionClass e) {
    ...... // 후속 예외처리
  }
}
```

한 학기 동안
수고 많이 하셨습니다.