제6장 멤버 함수 포인터와 예외처리

1. 멤버 함수 포인터

클래스의 멤버함수를 가리키는 포인터를 멤버함수 포인터라 한다.
멤버함수 포인터를 이해하기 위해서 함수 포인터의 기본 문법부터 차근차근 정리해보자.

▶ 함수 포인터

함수명는 자신의 주소 상수를 의미한다.
함수 포인터 변수를 이용해 간접적으로 해당 함수 호출이 가능하다. ← 동적 바인딩

▶ 함수 포인터의 타입 재정의

함수의 타입은 표기가 복잡하므로 typedef로 선언 한 후 이용 권장한다.
타입 재정의는 typedef나 using 명령어를 사용한다.

```
int add(int a, int b) { return a + b; }     //int (*)(int, int)
int sub(int a, int b) { return a - b; }     //int (*)(int, int)
typedef int func_2(int, int);
func_2 *pfn;
…
pfn = add;
pfn(10, 20);   //(*pfn)(10, 20) → add(10, 20)
pfn = sub;
pfn(10, 20);   //(*pfn)(10, 20) → sub(10, 20)
```

```
using func_2 = int(*)(int, int);
func_2 pfn;
…
switch(op_code) {
case '+' :
        pfn = add;
        break;
case '-' :
        pfn = sub;
        break;
}
pfn(10, 20);
```

[예제6-1] 함수포인터의 사용

```
1:
2:   #include <iostream>
3:   using namespace std;
4:   #include <cstdlib>
5:
6:   #define ARRAY_SIZE(a) (sizeof(a) / sizeof(a)[0])
7:   #define ARRAY_ITEM_SIZE(a) (sizeof(a[0]))
8:   int compare(const int* one, const int* two)
9:   {
10:         return (*one < *two) ? -1 : 1;
11:  }
12:  int main()
13:  {
14:         int a[] = { 20, 50, 30, 70, 40 };
15:
16:         qsort(a, ARRAY_SIZE(a), ARRAY_ITEM_SIZE(a),
17:                 reinterpret_cast<int(*)(const void*, const void*)>(compare));
18:
19:         for (auto value : a) {
20:                 cout << value << ' ';
21:         }
22:         cout << endl;
23:
24:         return 0;
25:  }
```

▶ 클래스 멤버 함수 포인터

클래스의 멤버함수를 가리키는 포인터를 의미.

객체의 멤버변수와 멤버함수는 객체를 통해서만 사용할 수 있으므로 멤버 함수 포인터 역시 객체를 이용해서 호출해주어야 함

- 멤버 포인터 변수 선언
  - 일반적인 함수 포인터 선언과 비슷함
  - 클래스 이름과 범위연산자(scope operator) 추가

- 멤버 포인터 변수에 멤버 주소 대입
  - 일반적인 함수 포인터에서 주소연산자 및 클래스 이름, 범위연산자(scope operator) 추가

```
class Log {
    string _msg;
public:
    Log(string msg) : _msg(msg) {}
    void show(void) const { … }
};
…
typedef void (Log::*Fptr)(void) const;
Fptr mp;       //void (Log::*mp)(void) const;
mp = &Log::show
```

• 멤버 포인터를 이용한 간접 접근
  - 반드시 인스턴스 필요
  - 인스턴스 타입에 따라 멤버 역 참조 연산자 (.* 또는 ->*) 선택
  - 인스턴스와 멤버 역 참조 연산자, 멤버 포인터를 괄호로 묶음

```
class Log {
    string _msg;
public:
    Log(string msg) : _msg(msg) {}
    void show(void) const { … }
};
…
typedef void (Log::*Fptr)(void) const;
// using Fptr = void(Log::*)(void) const;

Log log1("Stack");
Log *log2 = new Log("Heap");
Fptr pfn = &Log::show;
(log1.*pfn)();
(log2->*pfn)();
delete log2;
```

[예제6-2] 함수포인터의 사용

```
1:  #include <iostream>
2:  using namespace std;
3:  class Obj
4:  {
5:  public:
6:      void Foo(void) { cout << __FUNCTION__ << endl; }
7:  };
8:
9:  int main()
10: {
11:     Obj o;   //o는 Obj 타입 객체 선언
12:
13:     void (Obj::*pmFunc)(void); //Obj 클래스의 void(*)(void) 멤버에 대한 포인터 변수
14:     pmFunc = &Obj::Foo; //멤버 함수 포인터에 멤버 함수 주소 대입
15:
16:     (o.*pmFunc)(); // o객체와 멤버 함수 포인터를 이용해 Obj::Foo() 호출
17:
18:     return 0;
19: }
```

▶ 클래스 정적 멤버 함수 포인터

- 클래스의 정적 멤버는 인스턴스 없이 접근 가능
  - 일반 전역 변수나 함수와 비슷함
  - 클래스 이름과 범위연산자(scope operator) 사용

```
class Utile {
public:
    static void log(string msg) { … }
};
void (*pfn)(string msg);  //함수의 포인터 변수 선언과 동일
pfn = Utile::log;            //클래스 이름 포함
pfn("Static Member Call");
```

[예제6-4] 정적 멤버 포인터의 예

```
1:  #include <iostream>
2:  using namespace std;
3:
4:  #define ARRAY_SIZE(a) (sizeof(a) / sizeof(a)[0])
5:  #define ARRAY_ITEM_SIZE(a) (sizeof(a[0]))
6:
7:  class Obj2 {
8:     using COMP_FUNC_PTR = int (Obj2::*)(const int*, const int*);
9:  public:
10:    Obj2() {
11:           pThis = this; //객체가 생성될 때 정적 변수인 pThis에 객체 주소 대입
12:           pFunc = &Obj2::_compare;
13:    }
14:    //정적 멤버 함수에서 일반 멤버 함수를 호출하기 위해 객체 주소를 저장한 pThis 이용
15:    static int compare(const int* one, const int* two) {
16:          return (pThis->*pFunc)(one, two);
17:    }
18:
19: private:
20:    int _compare(const int* one, const int* two){ return (*one<*two) ? -1:1; }
21:
22:    static Obj2* pThis;
23:    static COMP_FUNC_PTR pFunc;
24: };
25:
26: Obj2* Obj2::pThis;                          //정적 변수는 외부 정의 필요
27: Obj2::COMP_FUNC_PTR Obj2::pFunc;
28:
29: int main()
30: {
31:    // 호출 테스트
32:    Obj2 o2;
```

```
33:     int ia[] = { 20, 50, 30, 70, 40, 60 };
34:     using comp_pfn_t = int(*)(const void*, const void*);
35:
36:     //정적 멤버 함수는 함수와 동일하게 사용 가능
37:     qsort(ia, ARRAY_SIZE(ia), sizeof(ia[0]),
38:               reinterpret_cast<comp_pfn_t>(Obj2::compare));
39:
40:     for (auto a : ia) {
41:             cout << a << ' ';
42:     }
43:
44:     return 0;
45: }
```

2. 예외 처리의 이해

◈ 예외(exception)

- 프로그램 실행 도중에 일어나는 비정상적인 상황
- 컴파일 에러는 포함되지 않음

◈ 예외(exception)의 종류

- 하드웨어, 소프트웨어 문제
- 사용자의 입력 실수 : 존재하지 않는 파일 열기, 숫자 입력 공간에 문자 입력 등
- 처리될 수 없는 연산 : 0으로 나누기

◈ 예외처리를 위한 명령어

- try블록 : 예외처리 구문은 예외상황이 발생할 가능성이 있는 코드를 묶어 줌
- catch블록 : 예외를 처리하는 핸들러
- throw구문 : 전달하는 명령어

◈ 예외처리 규칙

- 예외는 함수 범위를 넘어서 전달될 수 있음
- 함수 안에서 발생한 예외 처리 과정
  ① 함수 안에서 throw 문을 통한 예외 발생
  ② 함수 안에서 발생한 예외와 일치하는 catch블록이 있는지 검색
  ③ 함수 안에서 일치하는 catch문이 없으면 해당 함수를 호출한 곳에서 일치하는 catch블록이 있는지 검색
  ④ 일치하는 catch블록이 있을 때까지 ③번 과정 반복
  ⑤ main()함수까지 찾아갔는데도 일치하는 catch블록을 찾을 수 없으면 프로그램 비정상 종료

[예제6-1] try, catch, throw를 사용한 예외처리

```
1:  #include <iostream>
2:  using namespace std;
3:  #include <stdlib.h>
4:  int mod(int x, int y);
5:  int main()
6:  {
7:          int a, b, res;
8:
9:          cout << "두 정수 입력 : ";
10:         cin >> a >> b;
11:         try{
12:             res=mod(a, b);
13:         }
14:         catch(char *s){
15:             cout << s << endl;
16:             exit(1);
17:         }
18:         cout << a << "를 " << b << "로 나눈 몫 : " << res << endl;
19:
20:         return 0;
21: }
22:
23: int mod(int x, int y)
24: {
25:         if(y==0){
26:             throw "0으로 나눌 수 없습니다.";
27:         }
28:         return x/y;
29: }
```

실행결과

```
두 정수 입력 : 7 0(엔터)
0으로 나눌 수 없습니다.
```

[예제6-2] 서로 다른 예외형의 처리

```
1:   #include <iostream>
2:   using namespace std;
3   int mod1(int x, int y);   // a를 b로 나눈 몫
4   int mod2(int x, int y);   // 100을 a+b로 나눈 몫

5:   int main()
6:   {
7:         int a, b, res1, res2;
8:
9:         while(1){
10:              cout << "두 정수 입력 : ";
11:              cin >> a >> b;
12:              try{
13:                   res1=mod1(a, b);
14:                   res2=mod2(a, b);
15:                   cout << "try블록의 끝..." << endl;
16:              }
17:              catch(char *s){
18:                   cout << s << endl;
19:                   cout << "프로그램 종료..." << endl;
20:                   break;
21:              }
22:              catch(int n){
23:                   cout << "나누는 값 : " << n << endl;
24:                   cout << "다시 입력합니다..." << endl;
25:                   continue;
26:              }
27:
28:              cout << "res1 : " << res1 << endl;
29:              cout << "res2 : " << res2 << endl;
30:        }
31:        return 0;
32: }
33:
34: int mod1(int x, int y)
35: {
36:        if(y==0) throw "0으로 나눌 수 없다!";
37:        return x/y;
38: }
39:
40: int mod2(int x, int y)
41: {
42:        if(x==-y) throw 0;
43:        return 100/(x+y);
44: }
```

실행결과

```
두 정수 입력 : 10 3 (엔터)
try블록의 끝...
res1 : 3
res2 : 7

두 정수 입력 : 5 -5 (엔터)
나누는 값 : 0
다시 입력합니다...

두 정수 입력 : 10 0 (엔터)
0으로 나눌 수 없다!
프로그램 종료...
```

◈ try블록과 throw구문의 위치 문제

• 정상적인 리턴 과정을 거치지 못하는 중간단계의 함수들은 그들이 사용하던 객체의 소멸자를 자동으로 호출하여 할당 받은 메모리를 반환하는데 이를 스택 풀기라고 한다.

[예제6-3] 스택 풀기(stack unwinding)

```
1:  #include <iostream>
2:  using namespace std;
3:  #include <stdlib.h>
4:  #include <string.h>
5:
5:  class Test{
6:  private:
7:          char str[80];
8:  public:
9:          Test(char *s) { strcpy(str, s); }
10:         ~Test(){ cout << str << "객체의 소멸자 호출.." << endl; }
11: };
12:
13: int mod1(int x, int y);
14: int mod2(int x, int y);
15:
16: int main()
17: {
18:         int a, b, res;
19:
20:         cout << "두 정수 입력 : ";
21:         cin >> a >> b;
22:         try{
23:             res=mod2(a, b);
24:         }
25:         catch(char *s){
26:             cout << s << endl;
```

```
27:             exit(1);
28:         }
29:         cout << "몫의 두 배 : " << res << endl;
30:
31:         return 0;
32: }
33:
34: int mod2(int x, int y)
35: {
36:         Test ob1("mod2");
37:         int r;
38:
39:         r=2*mod1(x, y);
40:         return r;
41: }
42:
43: int mod1(int x, int y)
44: {
45:         Test ob2("mod1");
46:         if(y==0){
47:             throw "0으로 나눌 수 없습니다.";
48:         }
49:         return x/y;
50: }
```

실행결과

```
두 정수 입력 : 5 0
mod1객체의 소멸자 호출..
mod2객체의 소멸자 호출..
0으로 나눌 수 없습니다.
```

3. 예외 처리 클래스의 활용

[예제6-4]

```
1:  #include <iostream>
2:  using namespace std;
3:  #include <string.h>
4:
5;  class CException // 예외처리 클래스
6:  {
7:  private :
8:          char * excepFileName;
9:          int line;
10: public:
11:         CException(char * s, int l)
12:         {
13:                 excepFileName = new char[strlen(s) + 1];
14:                 strcpy(excepFileName, s);
15:                 line = l;
16:         }
17:         CException(const CException &r)
18:         {
19:                 excepFileName = new char[strlen(r.excepFileName) + 1];
20:                 strcpy(excepFileName, r.excepFileName);
21:                 line = r.line;
22:         }
23:         ~CException()
24:         {
25:                 delete[] excepFileName;
26:         }
27:         char* getName()
28:         {
29:                 return excepFileName;
30:         }
31:         int getLine()
32:         {
33:                 return line;
34:         }
35: };
36: int mod(int x, int y);
37: int   main()
38: {
39:         int a, b, res;
40:         cout << "두 정수 입력 : ";
41:         cin >> a >> b;
42:         try
43:         {
44:                 res=mod(a, b);
45:
46:                 cout << "This will be never printed" << endl;
47:         }
48:         catch(CException & e)
49:         {
50:                 cout << "Excepton " << e.getName() << " at line "
51:                         << e.getLine() << endl;
52:                 exit(1);
53:         }
54:         cout << a << "를 " << b << "로 나눈 몫 : " << res << endl;
55:         return 0;
56: }
```

```
57: int mod(int x, int y)
58: {
59:         if(y==0)
60:         {
61:           // 예외처리 오브젝트 생성
62:           throw CException("testMain.cpp파일", __LINE__);
63:         }
64:         return x/y;
65: }
```

◈ 예외 클래스 상속

• exception 클래스간의 상속 관계를 고려하여 예외 처리

• 파생클래스의 예외처리부를 기본클래스의 예외처리부 보다 먼저 작성해야 함

: 기본클래스의 예외처리부를 먼저 작성한 경우 파생클래스로 넘어가지 않음

[예제6-5]

```
1:  #include<iostream>
2:  using namespace std;
3:  class BaseException
4:  {
5:  public:
6:        void ShowExceptionMessage()
7:        {
8:              cout<<"BaseException exception₩n";
9:       }
10: };
11: class DerivedException : public BaseException
12: {
13: public:
14:        void ShowExceptionMessage()
15:        {
16:              cout<<"DerivedException exception₩n";
17:        }
18: };
19: class Theater
20: {
21: public:
22:       void BuyTicket(int age)
23:       {
24:            if(age>18)
25:                  cout << "관람가능합니다.₩n";
26:            else
27:                  throw DerivedException();
28:            cout << "결제에 성공했습니다.₩n";
29:       }
30: };
31: int main()
32: {
33:     int age;
34:     cout << "나이를입력하세요.";
35:     cin >> age;
36:     Theater t;
37:     try
38:     {
39:           t.BuyTicket(age);
40:     }
41:
42:     /* catch(BaseException ex) //잘못된 동작
43:     {
44:           ex.ShowExceptionMessage();
45:     }
46:     catch(DerivedException ex)
47:     {
48:           ex.ShowExceptionMessage();
49:     } */
50:     catch(DerivedException ex)
51:     {
52:           ex.ShowExceptionMessage();
53:     }
54:     catch(BaseException ex)
55:     {
56:           ex.ShowExceptionMessage();
57:     }
58:     return 0;
59: }
```

제7장 파일 I/O

◈ C++의 File I/O

C++에서 File 입출력은 객체를 이용해서 처리한다.

1. fstream 헤더 파일을 포함한다.
2. 출력 스트림을 관리하기 위해 ofstream 객체를 생성하여 사용한다.
3. 입력 스트림을 관리하기 위해서는 ifstream 객체를 생성하여 사용한다.
4. 입출력 함수는 cin, cout에서 사용하던 멤버함수들을 그대로 사용하면 된다.

예제 1) C++ 파일 입출력 예제

```
#include<fstream>
#include<iostream>
using namespace std;
#include<windows.h>

int main()
{
   ifstream fin;
   ofstream fout("c:₩₩data₩₩res.txt"); // 새로 쓰기 위해 open
   // 추가 하기 위한 오픈은 fout("c:₩₩data₩₩res.txt", ios_base::app);

   char filename[100];
   char name[20];
   char addr[100];
   double height;

   cout << "오픈 할 파일명을 입력하시오 : ";
   cin >> filename;

   fin.open(filename); // 읽기 위해 open
   if(fin.fail())  // 오픈 에러 체크
```

```
    {
        cout << filename << "file open error!!!" << endl;
        system("pause");
        return 0;
    }
    while( 1 )
    {
        fin.getline(name,sizeof(name));
        if(fin.eof()) { break; }  // file 끝 체크
        fin.getline(addr, sizeof(addr));
        fin>>height;
        fin.get(); // 숫자정보를 읽은 뒤에 남아있는 개행문자('₩n')을 읽어들임
        cout << name <<" "<< addr <<" "<< height << endl; // 모니터에 출력
    }

    fin.clear();  // eof 상태를 clear해주어야 파일을 다시 읽을 수 있게 됨
    fin.seekg(0); // file의 읽을 위치를 맨 처음으로 되돌려 놓음

    while( 1 )
    {
        fin.getline(name,sizeof(name));
        if(fin.eof()) { break; }  // file 끝 체크
        fin.getline(addr, sizeof(addr));
        fin>>height;
        fin.get();
        fout <<name<<" "<<addr<<" "<< height<<endl; // res.txt파일에 출력
    }
    fin.close();  // 파일 닫기
    fout.close(); // 파일 닫기
    return 0;
}
```