7-1. 템플릿(Template)

♲ 템플릿 (template)

- 동일한 기능을 하는 함수가 반환형이나 인자의 자료형만이 달라지는 경우 템플릿을 이용하여 함수를 작성한 후 인자나 반환형에 따라 함수를 호출할 수 있다.
- 템플릿은 데이터 타입을 인자로 전달할 수 있으며, 이를 통해 함수의 일반화를 가능하도록 한다.
- 객체 지향 프로그래밍의 다형성을 지원하며 함수나 클래스를 매번 정의하지 않고 한 번의 정의로 공유할 수 있기 때문에 효율적인 프로그래밍이 가능하다.

ex1

```
void Swap(int &i, int &j) {
	int t = i;
	i = j
	j = t;
};
```

ex2

```
void Swap(long &i, long &j) {
	long t = i;
	i = j
	j = t;
};
```

ex3

```
// #define SwapType long
typedef long SwapType;
void Swap(SwapType &i, SwapType &j) {
	SwapType t = i;
	i = j
	j = t;
};
```

ex4

```
void Swap(int &i, int &j) {
	int t = i;
	i = j
	j = t;
};
void Swap(long &i, long &j) {
	int t = i;
	i = j
	j = t;
};
```

템플릿의 선언방법

```
template <type_list [인수_list] >
```

위의 예를 템플릿을 이용해 선언한 코드

```
template<class SwapType>
void Swap(SwapType &i, SwapType &j) {
      SwapType t = i;
      i = j
      j = t;
};
```

※ Swap()은 일반 함수가 아니라 템플릿함수이다. 템플릿 함수는 함수가 아니라 함수를 찍어내는 틀일뿐이다.

Ex1) 템플릿 함수를 이용한 swap()프로그램

```
#include <iostream.h>

template <class SwapType>
void Swap(SwapType &i, SwapType &j) {
      SwapType t = i;
  i = j;
  j = t;
}   //Swap

int main()  {
      int i = 2,  j = 3;
  Swap(i, j);
      cout << "i = " << i << ", j = " << j << endl;

      return 0;
}
```

실 행 결 과	결 과 분 석
i = 3 , j = 2 Press any key to continue...	main의 Swap()는 템플릿함수 Swap()를 호출한 것이 아니라 컴파일 시간에 생성된 Swap(int &, int &)를 호출한 것이다. 또한 template는 형의 리스트를 정할 수 있다.

Ex2) 템플릿의 파라미터로 변수를 지정하는 예제 (에러발생)

```
#include <iostream.h>
#include <string.h>

template <class T, class U, int i>
void func(T x) {
    U buffer[i];
    strcpy(buffer, x);
    cout << "buffer = " << buffer << endl;
}

int main() {
    func("Hello world"); //컴파일 시 에러 발생
                return 0;
}
```

Ex3) 2번 예제를 수정한 프로그램

```
#include <iostream.h>
#include <string.h>

template <class T, class U, int i>
void func(T x) {
    U buffer[i];
    strcpy(buffer, x);
    cout << "buffer = " << buffer << endl;
}

int main() {
    func <char *, char, 20> ("Hello world");
                return 0;
}
```

실 행 결 과	결 과 분 석
buffer = Hello world Press any key to continue...	함수 func가 문법처럼 호출되면 T, U와 I가 각각 char *, char, 20으로 결정된다. 템플릿의 파라미터이므로 < > 안에 명시한다.

Ex4) 템플릿 swap() 함수 예제 프로그램

```
#include<iostream.h>

template <class DATATYPE>
void swap(DATATYPE &a, DATATYPE &b)  {
       DATATYPE t;
       t = a;
       a = b;
       b = t;
}

void main()  {
       int a = 10, b = 20;
       swap(a, b);
       cout << "₩n a = " << a << ", b = " << b << endl;
       double c = 10.5, d = 20.7;
        swap(c, d);
       cout << "₩n c = " << c << ", d = " << d << endl;
 }
```

실 행 결 과	결 과 분 석
a = 20, b = 10 c = 10.5, d = 20.7 Press any key to continue...	

Ex5) 템플릿함수 MIN()함수를 정의한 프로그램

```
#include<iostream.h>
template <class DATATYPE>
DATATYPE MIN(DATATYPE a, DATATYPE b)  {
       return (a<b)? a:b;
}

void main()  {
       cout << MIN(37, 19) << endl;  // int MIN(int, int)형 함수 생성 및 호출
       cout << MIN(5.182, 3.942);
                                    // double MIN(double, double)형 함수 생성 및 호출
}
```

실 행 결 과	결 과 분 석
19 3.942 Press any key to continue...	

Ex7) 템플릿의 형변환 예제 프로그램

```
#include <iostream.h>

template <class Type>
Type MIN(Type a, Type b)  {
       return (a < b)? a:b;
}

void main()  {
       int n = MIN(54, 22);  // int MIN(int, int) 함수 생성 및 호출
       cout << "MIN(54, 22) : " << n << endl;
       double d = MIN(double(n), 8.3);
                                    //double MIN(double, double)함수 생성 및 호출
       cout << "MIN(" << n << ", 8.3) : " << d << endl;
}
```

실 행 결 과	결 과 분 석
MIN(54, 22) : 22 MIN(22, 8.3) : 8.3 Press any key to continue...	정수형과 실수형의 인자를 가지는 함수가 호출될 때는 정수형 자료를 명시적으로 형 변환해 템플릿 함수를 호출한다.

Ex8) 템플릿함수의 오버로딩 예제 프로그램

```
#include <iostream.h>

template <class Type>   // sum() 템플릿함수1 정의
Type sum(Type a, Type b)  {
       return a+b;
}

template <class Type>  // sum() 템플릿함수2 정의
Type sum(Type *array, int size)  {
       Type tot = 0;
       for(int i=0; i<size; i++)
              tot += array[i];
       return tot;
}

void main()  {
       int arr[10];
       for(int i=0; i<10; i++) {  arr[i] = i*i;  }
       float x = 5.5, y = 19.2;
       cout << "sum 1: " << sum(x, y) << endl;  // 템플릿함수1 생성 및 호출
       cout << "sum 2: " << sum(arr, 10) << endl;  // 템플릿함수2 생성 및 호출
}
```

실 행 결 과	결 과 분 석
sum 1 : 24.7 sum 2 : 285 Press any key to continue...	단순자료형인 sum(x, y) 템플릿 함수와 배열과 정수를 가지는 sum(arr, size)간의 중복정의 예제이다

Ex9) 템플릿 함수와 일반함수의 오버로딩 예제 프로그램

```
#include <iostream.h>
#include <string.h>

template <class Type>
Type MIN(Type a, Type b)  {
       return (a < b)? a:b;
}

char *MIN(char *str1, char *str2)  {
       return (strcmp(str1, str2) < 0) ? str1 : str2;
}

void main()  {
       int n = MIN(34, 19);  // int MIN(int, int) 함수 생성 및 호출
       char *ptr = MIN("alarm", "phone");
                              // 일반함수 char *MIN(char *, char *)호출
       cout << " n = " << n << endl;
       cout << " ptr = " << ptr << endl;
}
```

실 행 결 과	결 과 분 석
n = 19 ptr = alarm Press any key to continue...	문자열 비교를 위해서 strcmp() 함수가 필요하게 됨으로 문자열 비교를 위한 독자적인 함수를 오버로딩해야 한다

Ex10) 정수형 배열과 실수형 배열을 오름차순 정렬하는 템플릿 함수 정의

```
#include <iostream.h>
#include <iomanip.h>

const int MAX = 10;

template <class Type>
void sort(Type digits[])  {
   Type temp;
   for(int i=0; i<MAX-1; i++) {
       for(int j=i+1; j<MAX; j++) {
           if(digits[i] > digits[j]) {
               temp = digits[i];
               digits[i] = digits[j];
               digits[j] = temp;
           }
      }
   }
   cout << "Wn The sorted values : Wn";
   for(int j=0; j<10; j++)
                cout << setw(7) << digits[j];
        cout << endl;
}

int main()  {
   static int arr1[] = {6, 5, 8, 87, 65, 30, 45, 23, 68, 17};
   static double arr2[] = {9.92, 1.45, 2.39, 5.78, 0.56, 4.10, 4.07, 6.80, 1.49, 3.82};

   sort(arr1); // 정수형 함수 호출
   sort(arr2); // 실수형 함수 호출
        return 0;
}
```

실 행 결 과

```
 The sorted values :
      5       6       8       17      23      30      45      65      68      87

 The sorted values :
    0.56    1.45    1.47    2.39    3.82    4.07     4.1    5.78     6.8    9.92

Press any key to continue...
```

템플릿 클래스(Template Class)란?

- 서로 다른 두 개 이상의 클래스가 유사한 내용으로 중복되어 정의되어 있을 경우
- 실제 클래스의 정의는 템플릿 클래스를 정의하는 부분만 존재한다.
- 클래스의 객체를 생성 시에 템플릿 인자를 지정함으로서 템플릿 클래스의 객체를 생성할 수 있다.
- 생성된 객체는 일반적인 클래스로부터 생성된 객체와 동일하다.
- 자료형에 제한을 받지 않는 객체를 생성할 수 있다.
- 컴파일 시간이 길어지는 단점이 있다.

Ex11) 템플릿 클래스를 이용한 스택 프로그램

```
#include <iostream.h>
template<class Type>
class Stack {
        Type *data;
        int size, sp;
   public:
        Stack(int s) {
           data = new Type[size = s];
           sp = -1;
       }
        void Push(Type d);
        Type Pop();
};   // Stack

template<class Type>
void Stack<Type>::Push(Type d) {
       ++sp;
       data[sp] = d;
}   // Push()

template<class Type>
Type Stack<Type>::Pop() {
       return data[sp--];
}   // Pop()

void main() {
       Stack<char> Cha(10);
        Stack<int> Int(10);

       Cha.Push('A');  Cha.Push('a');
        Int.Push(100);  Int.Push(1);
       cout << "Cha.Pop() : " << Cha.Pop() << endl;
       cout << "Int.Pop() : " << Int.Pop() << endl;
}
```

실 행 결 과	결 과 분 석
Cha.Pop() : a Int.Pop() : 1 Press any key to continue...	템플릿 클래스의 멤버 함수를 클래스외부에서 정의할 때는 클래스의 이름 뒤에 형이 무었인지를 명시한다

Ex14) 템플릿 클래스 사용예제 프로그램 -1

```
#include <iostream.h>

template <class Type>
class Counter {
                Type value;
        public:
                Counter(Type n)   { value = n;  }
                Counter()         { value = 0;  }
                Type val()        { return value;  }
                void operator++() { ++value;  }
                void operator--() { --value;  }
};

int main()  {
        Counter <int>     icnt;
        Counter <double>  dcnt(7.12);
        Counter <char>    ccnt('K');

        ++icnt;  --dcnt;  ++ccnt;
        cout << "++icnt : " << icnt.val() << endl;
        cout << "--dcnt : " << dcnt.val() << endl;
        cout << "++ccnt : " << ccnt.val() << endl;

        return 0;
}
```

실 행 결 과	결 과 분 석
++icnt : 1 --dcnt : 6.12 ++ccnt : L Press any key to continue...	객체 선언시 일반 클래스를 이용한 선언과 마찬가지로 생성자를 호출함으로 생성자의 실인수를 기술할 수 있다.

Ex15) 템플릿 클래스 사용예제 프로그램 -2

```
#include <iostream.h>

template <class SS, class KK>
class Size {
                int s, k;
        public:
                Size()  {
                        s = sizeof(SS);
                        k = sizeof(KK);
                }
                void print() {
                        cout << "s = " << s << ", k = " << k << endl;
                }
};

int main()  {
        Size <char, long int>  ob1;
        Size <float, double>   ob2;

        cout << "Wn char, long int : "; ob1.print();
        cout << "Wn float, double : "; ob2.print();

        return 0;
}
```

실 행 결 과	결 과 분 석
char, long int : s = 1, k = 4 float, double : s = 4, k = 8 Press any key to continue...	ob 객체 선언시 기술된 자료형 <char, long, int>는 각각 SS에는 char, KK에는 long int 가 순서대로 치환되어 객체를 선언한다

Ex16) 배열의 범위조사가 가능한 템플릿 클래스의 예제

```
#include <iostream.h>
#include <stdlib.h>
template <class Type>
class Array  {
               Type *p;   // 배열의 시작 주소
               int size;  // 배열의 크기
       public:
               Array(int num);   // 생성자
               Type &operator [](int i);   // [ ] 연산자 중복
};

template <class Type>
Array<Type>::Array(int num)  {  // 템플릿 클래스 생성자 정의
       p = new Type[num];
       if(!p)  {
               cout << "₩n 메모리 할당 에러!1 ₩n";
               exit(1);
       }
       size = num;
}

template <class Type>
Type &Array<Type>::operator [](int i)  {
       if(i<0 || i>=size)  {   // 배열의 범위 검사
               cout << "₩n 배열의 범위 참조 에러!! ₩n";
               exit(1);
       }
       return p[i];   // p[i]의 참조자 반환
}

void main()  {
       Array <int> n(10);  // int형 Array 클래스 생성
       Array <double> d(10);  // double형 Array 클래스 생성

       for(int i=0; i<10; i++)  {
               n[i] = 2*i;  // 연산자 함수 [ ]호출, 참조자에 값 대입
               d[i] = 0.2*i;
       }

       for(i=0; i<11; i++)  {
               cout << n[i] << ":";  // 연산자 함수 [ ]호출
               cout << d[i] << "  ";
       }
       cout << endl;
}
```

실 행 결 과

```
0:0  2:0.2  4:0.4  6:0.6  8:0.8  10:1  12:1.2  14:1.4  16:1.6  18:1.8
 배열의 범위 참조 에러!!
Press any key to continue..0
```

Ex17) 최대값, 최소값을 계산하는 템플릿 클래스

```
#include <iostream.h>
#include <string.h>

template <class Type>
class Cmp  {
                Type a;
                Type b;
        public:
                Cmp(Type x, Type y)  {  a = x;   b = y;  }
                Type max() { return a>b? a:b;  }
                Type min() { return a<b? a:b;  }
                // char *형, max(), min() 멤버 함수 중복
                char *max(char *str1, char *str2)  {
                        return strcmp(str1, str2)>0? str1 : str2;
                }
                char *min(char *str1, char *str2);
};

char *Cmp<char *>::min(char *str1, char *str2)  {
        return  strcmp(str1, str2)<0? str1 : str2;
}

int main()  {
        Cmp <int> ob1(31, 87);
        Cmp <char *> ob2("C++", "Programming");

        cout << ob1.min() << endl;
        cout << ob2.max("C++", "Programming") << endl;

        return 0;
}
```

실 행 결 과

```
31
Programming
Press any key to continue...
```

Ex18) 문자열을 저장하기 위한 템플릿 클래스 정의

```
#include <iostream.h>
#include <stdlib.h>

template <class Type>
class Array {
    Type *str;
    int min, max;
    void del()  {
        if(str) delete []str;
        str = 0;    // 널 포인터 값 저장
    }
    void set(const Array &OB)  {
        min = OB.min;
        max = OB.max;
        if(!OB.str)
            str = 0;
        else {
            str = new Type[max - min + 1 ];
            for(int i=0; i<max-min; i++)
                str[i] = OB.str[i];
        }
    }
public:
    Array()  {   // 디폴트 생성자
        str = 0;
        min = max = 0;
    }
    Array(int low, int high)  { // 인수를 갖는 생성자
        if(high < low)  {
            int temp = low;
            low = high;
            high = temp;
        }
        const int len = high-low+1;
        str = new Type[len];
        min = low;
        max = high;
    }
    Type & operator [](int i)  {  // [ ]연산자의 오버로딩
        if(i<min || i>max) {
            cerr << "첨자 범위 오류!! Wn";
            exit(1);
        }
        return str[i];
    }
    ~Array()  {      // 소멸자
        del();
    }
```

```
    Array & operator = (const Array & OB)   {  // 할당연산자의 오버로딩
        if(this != &OB) {
            del();
            set(OB);
        }
        return *this;
    }
    Array(const Array & OB)  {  // 복사생성자
        set(OB);
    }
   void print()  {
        for(int i=0; i<26; i++)
            cout << str[i] << " ";
        cout << endl;
    }
};

void main()  {
       Array <char> ob1(0, 26);   // 인수를 갖는 생성자의 호출
  for(int i=0; i<26; i++) {
              ob1[i] = 'A' + i;     //  [ ]연산자 오버로딩에 의한 객체의 초기화
  }

  Array <char> ob2 = ob1;  // 복사생성자를 이용한 객체의 초기화
  Array <char> ob3;    // 디폴트 생성자 호출
  ob3 = ob1;        // 할당연산자 오버로딩에 의한 객체의 초기화

  ob1.print();
  ob2.print();
  ob3.print();

}
```

실 행 결 과

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Press any key to continue...
```

7-2. 예외처리(exception handling)

오류 없는 프로그램을 작성하기는 매우 어렵다. 또한 오류의 원인을 찾아내는 것도 쉬운 일이 아니다. 프로그램을 작성하다보면 다음과 같은 일이 발생한다.

① 예기치 않은 잘못된 변수의 값
② 잘못된 자료의 입력
③ 네트워크의 예고 없는 끊김
④ 메모리 부족으로 인한 메모리 할당에러
⑤ 출력하려는 포인터의 값이 0인 경우
⑥ 0으로 나누기 등

재난 방지를 위해 abort()나 exit()등 비정상 종료 함수를 수동으로 사용하는 방법도 있지만 치밀한 주의력을 필요로 한다.

※ fault와 error

fault : 원하는 대로 동작하지 않는 것(동작 명세대로 동작하지 않는 것)
error : 이상하게 동작하는 것(동작 명세대로 동작하게 만들 프로그램의 부분적인 결함

예외처리의 원리

어떠한 코드를 실행했을 때 발생할 수 있는 모든 오류에 대해 처리하는 코드를 미리 만들어 두어 오류를 방지하는 것이다. 즉, 오류가 발생하면 미리 만들어 두었던 코드로 제어를 이동시켜 처리하게 하는 것이다.

C++에서 지원하는 구조적 예외 처리 (catch handler)

```
try, catch, throw
```

※ 구조적 예외처리 구문은 borland C++ 4.5이상에서 지원한다

예외처리문의 사용형식

```
try {
        예외가 발생하는 문(들);
        throw 예외;
}
catch (매개변수) {
        예외처리1
}
catch (매개변수) {
        예외처리2
}
        .
        .
        .
```

① try와 catch문은 반드시 인접하여 존재해야한다.
② 예외가 발생하면 throw문에 의해 해당하는 에러 사항과 일치하는 catch 문으로 예외상황을 넘겨 처리한다.
③ catch문은 여러개 존재할 수 있다.
④ catch문의 결정은 throw에서 넘어온 예외와 매개변수의 자료형이 서로 일치된 예외 상황에 따른다.
⑤ 예외가 발생하지 않으면 catch문도 실행되지 않는다.
⑥ throw문은 예외상황 발생하기 전까지 할당된 모든 메모리를 자동으로 제거한 후 예외상황을 던지는 문으로도 사용된다
⑦ 전체 프로그램을 예외상황으로 평가하고자 한다면 try문안에 main()함수 전체 코드를 포함시킬 수 있다.

Ex1) 문제가 일어날 만한 부분을 제어문으로 해결한 프로그램

```
#include <iostream.h>

int main() {
	int i=5, j=2, k=0;

	cout << "--> begin <--₩n" << endl;

	if (j==100 || j==0)
		cout << j << " : Error occured" << endl;
	else
		cout << i/j << endl;

	if (k==100 || k==0)
		cout << k << " : Error occured" << endl;
	else
		cout << i/k << endl;

	cout << "₩n--> end <--" << endl;

	return 0;
}
```

실 행 결 과

```
--> begin <--

2
0 : Error occured

--> end <--

Press any key to continue...
```

Ex2) 예외처리에 관한 간단한 프로그램

```
#include <iostream.h>

int main() {
    int i=5, j=2, k=0;

                cout << "--> begin <--₩n" << endl;

                try {
                        if(j == 100 || j == 0)
                                throw j;
                        cout << i/j << endl;

                        if (k==100 || k==0)
                                throw k;
                        cout << i/k << endl;
    }
                catch(int e) {
                        cout << e << " : Error occured" << endl;
                }

                cout << "₩n--> end <--" << endl;
                return 0;
}
```

실 행 결 과

```
--> begin <--
2
0 : Error occured
--> end <--

Press any key to continue...
```

Ex3) 예외처리가 비정상적으로 처리된 경우의 프로그램

```
#include <iostream.h>

int main() {
	int i=5, j=2, k=0;

	cout << "--> begin <--₩n" << endl;
	try {
		if (j==100 || j==0)
			throw j;
		cout << i/j << endl;

		if (k==100 || k==0)
			throw k;
		cout << i/k << endl;
	}
	catch(char e) {
		cout << e << " : Error occured" << endl;
	}
	cout << "₩n--> end <--" << endl;
	return 0;
}
```

Ex4) 예외처리 구문형식을 지키지 않아 컴파일 되지 않는 프로그램

```
#include <iostream.h>
void main() {
	int i=5, j=2, k=0;
	cout << "--> begin <--₩n" << endl;
	try {
		if (j==100 || j==0)
			throw j;
		cout << i/j << endl;
		if (k==100 || k==0)
			throw k;
		cout << i/k << endl;
	}
	cout << "₩n--> middle <--₩n" << endl;
	catch(int e) {		//try와 catch를 아무도 방해해서는 안 된다.
		cout << e << ": Error occured" << endl;
	}
	cout << "₩n--> end <--" << endl;
}
```

Ex5) catch 블록을 여러개 연속하여 선언한 프로그램

```
#include <iostream.h>

int main() {
	int i=5, j=2, k=0;

	cout << "--> begin <--₩n" << endl;
	try {
		if (j==100 || j==0)
			throw "예외 : j가 100 또는 0 ₩n";
		cout << i/j << endl;

		if (k==100 || k==0)
			throw "예외 : k가 100 또는 0 ₩n";
		cout << i/k << endl;
		cout << "₩n try 블록 끝" << endl;
	}
	catch(char e) {
		cout << "char catch" << endl;
	}
	catch(int e) {  // catch(char e)에 실패한 경우
		cout << "int catch" << endl;
	}
	 catch(char* e) {  // catch(int e)에 실패한 경우
		cout << e << endl;
	}
	cout << "₩n--> end <--" << endl;
	return 0;
}
```

실 행 결 과	결 과 분 석
--> begin <-- 2 예외 : k가 100 또는 0 --> end <-- Press any key to continue...	‘예외 : k가 100 또는 0’ fmf 출력한 블록은 3번째 catch(char* e) 블록이다

Ex6) 블록외부의 함수에서의 에러를 처리한 경우

```
#include <iostream.h>

void Func(int test) {
       cout << " Inside Func, test is : " << test << "₩n";
  if(test) throw test;
}

int main() {
  cout << "--> Begin <-- ₩n";

  try {   // try 블록 시작
              cout << "₩n Inside try block... ₩n";
    Func(0);
    Func(1);
    Func(2);
  }
  catch(int i) {  // 오류 잡기
    cout << " Caught an exception : value is ";
    cout << i << "₩n";
  }
  cout << "₩n --> End <--" << endl;
  return 0;
}
```

실 행 결 과	결 과 분 석
--> Begin <-- Inside try block... Inside Func, test is : 0 Inside Func, test is : 1 Caught an exception : value is 1 --> End <-- Press any key to continue...	try블록에서 호출된 함수 내에 존재하기만 한다면 try블록 외부의 문장에서도 예외를 옮길 수 있다.

Ex7) 조화평균 구하는 프로그램 (try 블록 외부에서의 예외상황 발생)

```
#include <iostream.h>
double hmean(double a, double b);
void main()  {
       double x, y, z;
       cout << "₩n 두 수를 입력하십시오 : ";
       while(cin >> x >> y) {
              try {
                     z = hmean(x, y);
              }
              catch(char *s)  {
                     cout << s << endl;
                     cout << " 새로운 두 수를 입력하십시오 : ";
                     continue;
              }             // 예외 핸들러의 끝
              cout << " ₩n ==> 두 수 " << x << ", "<< y << "의 조화 평균 : " << z << endl;
              cout << "₩n다음 두 수를 입력하십시오.[종료하려면 q]:";
       }
       cout << "₩n 종료~!" << endl;
}

double hmean(double a, double b)  {
       if(a == b)
              throw "₩n 잘못된 hmean() 전달인자 : a==b는 허용되지 않습니다. ₩n";
         return 2.0*a*b/(a+b);         // 조화평균 구하기
}
```

실 행 결 과	결 과 분 석
두 수를 입력하십시오 : 2 3 ==> 두 수 2, 3 의 조화 평균 : 2.4 다음 두 수를 입력하십시오.[종료하려면 q]: 5.3 8.90 ==> 두 수 5.3, 8.90 의 조화 평균 : 6.64366 다음 두 수를 입력하십시오.[종료하려면 q]: 10 10 잘못된 hmean() 전달인자 : a==b는 허용되지 않습니다. 새로운 두 수를 입력하십시오 : 4 100 ==> 두 수 5.3, 8.90 의 조화 평균 : 7.69231 다음 두 수를 입력하십시오.[종료하려면 q]: q 종료~! Press any key to continue...	에러는 try블록 외부에 정의된 함수 hmean()에서 발생하고 그것을 다시 호출함수인 main()에서 예외를 받아 처리한다.

Ex8) 예외상황을 특정 함수 내에서만 발생하게 지정한 프로그램

```
#include<iostream.h>

void Err_test(int n) {
	try {
		if(n > 0)
			throw "Positive";
		else if(n == 0)
			throw "Zero";
		else
			throw "Negative";
	}
	catch(char * msg) {
		cout << "Message : " << msg << endl;
  }
}

int main()  {
	Err_test(-10);
	Err_test(0);
	Err_test(10);

	return 0;
}
```

실 행 결 과

```
Message : Negative
Message : Zero
Message : Positive

Press any key to continue...
```

Ex9) 모든 예외를 처리하는 프로그램

```
#include <iostream.h>

int main() {
	int i=5, j=2, k=0;

	cout << "--> begin <--₩n" << endl;
	try {
		if (j==100 || j==0)
			throw "예외 : j가 100 또는 0 ₩n";
		cout << i/j << endl;

		//if (k==100 || k==0)
		//	throw "예외 : k가 100 또는 0 ₩n";
		cout << i/k << endl;
		cout << "₩n try 블록 끝" << endl;
	}
	catch(char e) {
		cout << "char catch" << endl;
	}
	catch(int e) {  // catch(char e)에 실패한 경우
		cout << "int catch" << endl;
	}
	catch(...) {
    cout << "I will catch anything!" << endl;
  }
	cout << "₩n--> end <--" << endl;
	return 0;
}
```

실 행 결 과	결 과 분 석
--> begin <-- 2 I will catch anything! --> end <-- Press any key to continue...	catch(...)는 어떠한 자료형이라도 예외를 받아 처리할 수 있다.

Ex10) 모든 예외를 처리하는 프로그램 -2

```
#include <iostream.h>

void Xhandler(int test) {
  try {
              switch(test)  {
                      case 0 : throw test;     // 정수형 옮기기
                      case 1 : throw 'A';            // 문자형 옮기기
                      case 2 : throw 123.45;         // double형 옮기기
                      case 3 : throw "C++ 프로그램"; // 문자열 옮기기
                      default : throw test;
              }
  }
  catch(...) {    // 모든 예외 잡기
    cout << " Caught : " << test << "Wn";
  }
}

int main() {
  Xhandler(0);
  Xhandler(1);
  Xhandler(2);
  Xhandler(3);
  Xhandler(99);
  return 0;
}
```

실 행 결 과

```
 Caught : 0
 Caught : 1
 Caught : 2
 Caught : 3
 Caught : 99

Press any key to continue...
```