C++ Template Programming

Project

2017년 2차 과정 (2017.6.12 ~ 7/9)

2017.06.19

~

2017.07.09

❑ 과제 배점

20점 만점

[참고] Quiz : 25문항 * 2 = 50점, Final Exam : 20 * 1.5 = 30점, Final Project : 20점.

❑ 과제 설명

총 5개의 프로그래밍 과제 입니다. 5개의 문제 중 3개이상만 해결해서 제출 하시면 됩니다.

1개 해결시 : 10점

2개 해결시 : 15점

3개 이상 해결시 : 20점 만점

❑ 제출 방법

"이름_사번" 으로 폴더를 만들고, 폴더안에

과제설명.txt 을 만들고

(1) 몇 번 과제를 해결했는지 적어 주세요..

(2) 어떤 컴파일러를 사용했는지를 적어 주세요

해결한 과제만, "**과제1.cpp, 과제2.cpp, 과제3.cpp** …" 으로 소스를 만들어 주세요. 실행파일은 필요 없습니다. 소스만 포함해 주시면 됩니다. **소스 파일의 이름 규칙을 꼭 지켜 주세요(대소문자 포함)**

폴더를 포함해서 통째로 압축해서 LGE MOOC "Final Project"에서 "select file" 버튼을 눌러서 압축파일을 올려 주세요

압축파일명은 반드시 영문이어야 합니다. 시스템 특성성 파일명은 한글을 사용할수 없습니다.

❑ 제출 기한

2017년 7월 9일(일요일) 24:00 까지

과제 1. N1 ~ N2 사이의 합을 구하는 Sum을 만드세요

```
int main()
{
    int     n1 = 5, n2 = 10;
    short   s1 = 5, s2 = 10;
    double  d1 = 3.4, d2 = 1.2;

    Sum(n1, n2); // ok
    Sum(s1, s2); // ok
    Sum(d1, d2); // error가 나와야 합니다.
}
```

조건 1. 모든 정수 타입에는 동작해야 하지만 실수 일 경우는 컴파일 에러가 나와야 합니다.
조건 2. static_assert() 버전과 enable_if 버전으로 각각 작성해 보세요

[참고 자료] - 20 강 enable_if 동영상 참고

과제 2. 인자로 전달된 컨테이너가 가진 모든 요소의 평균을 구하는 average()를 만드세요

```
int main()
{
    int x[10]{ 1,2,3,4,5,6,7,8,9,10 };
    vector<int> v{ 1,2,3,4,5,6,7,8,9,10 };


    cout << average(x) << endl;
    cout << average(v) << endl;
}
```

배열과 STL 컨테이너 모두에 대해서 동작해야 합니다.

과제 3. 특정 클래스안에 resize 멤버 함수가 있는지 조사하는 has_resize<> traits를 만들어 보세요.

```
template<typename T> void foo(T& c)
{
    if (has_resize<T>::value)
        cout << "T has resize" << endl;
    else
        cout << "T has not resize" << endl;
}
int main()
{
    vector<int>    v  = { 1,2,3,4,5 };
    array<int, 10> ar = { 1,2,3,4,5 };
    foo(v);     // vector는 resize 함수를 가지고 있습니다.
    foo(ar);    // array 는 resize 함수가 없습니다.
}
```

[참고 자료] - 20강 동영상 중 is_abstract<> 만들기 동영상을 참고 하세요

과제 4. printf() 함수의 전통적인 문제점은 인자의 개수를 잘못 보내도 에러(컴파일 또는 실행시)가 나오지 않는 다는 점입니다. 가변 인자 함수 템플릿을 사용해서 printf()의 문제를 해결해 보세요

```
int main()
{
    printf("%d\n", 1, 2, 3); // 인자가 너무 많습니다.
    printf("%d%d\n", 1);     // 인자가 부족 합니다.

    // cpp_print를 만들어 보세요
    try
    {
        cpp_print("%d\n", 1, 2, 3); // 예외가 나오게 하세요
        cpp_print("%d%d\n", 1);     // 예외가 나오게 하세요
    }
    catch (...)
    {
        cout << "예외 발생" << endl;
    }
}
```

[참고 자료] - 23강 동영상 참고

단, "%f"등은 처리하지 말고, "%d" 처리 할 수 있으면 됩니다.

과제 5. TypeList 만들기

1~3의 TypeList 설명을 잘 읽고, 과제를 해결해 보세요. 과제는 마지막 부분에 설명됩니다.

1. TypeList 소개

13강 동영상에서 만든 couple 이나, STL의 **pair는 서로 다른 타입 2개의 값을 보관하는 구조체** 입니다.

```
template<typename T, typename U> struct pair
{
    T first;
    U second;
};
```

하지만, 아래 TypeList는 **값이 아닌 타입만** 보관 합니다.

```
template <class T, class U> struct Typelist
{
    typedef T Head;
    typedef U Tail;
};
```

TypeList는 타입을 2개 보관하지만, 템플릿은 인자로 자신을 받을 수 있으므로 TypeList는 2개 뿐 아니라 몇 개의 타입도 보관할 수 있습니다. 그리고 TypeList 의 끝을 나타내는 NullType도 도입했습니다.

```
struct NullType {};

template <class T, class U> struct Typelist
{
    typedef T Head;
    typedef U Tail;
};
TypeList<int, NullType> t1; // 한개의 타입을 보관합니다.
TypeList<int, TypeList<double, NullType>> t2; // 2개의 타입을 보관합니다.
TypeList<int, TypeList<double, TypeList<char, NullType>>> t3; // 3개의 타입 보관.
```

2. TypeList 활용

인자가 하나인 템플릿에는 타입을 하나만 보낼 수 있습니다. 하지만, TypeList를 사용하면 인자가 한 개인 템플릿에도 여러 개의 타입을 보낼 수 있습니다. 따라서 가변인자 템플릿이 없어도 다양한 용도로 활용 할수 있습니다.

```
template<typename T> class xtuple // xtuple은 인자가 하나밖에 없습니다.
{
};

int main()
{
    xtuple<int> t1; // ok

    // xtuple<int, char> t2; // error. xtuple에는 타입을 하나만 보낼수 있습니다.

    xtuple<TypeList<int, TypeList<char, NullType>>> t2; // ok.. TypeList를 사용하면
                                                        // 타입을 몇개 라도 보낼수 있습니다.
}
```

3. TypeList 연산

- **MakeTypeList**

TypeList를 바로 사용하면 재귀적 모양 때문에 좀 불변 합니다. TypeList를 선형적으로 만들수 있도록 도와주는 MakeTypeList를 도입하면 간단하게 TypeList를 만들 수 있습니다.

```
template <  typename T1 = NullType, typename T2 = NullType, typename T3 = NullType,
            typename T4 = NullType, typename T5 = NullType, typename T6 = NullType,
            typename T7 = NullType, typename T8 = NullType, typename T9 = NullType,
            typename T10 = NullType, typename T11 = NullType, typename T12 = NullType,
            typename T13 = NullType, typename T14 = NullType, typename T15 = NullType,
            typename T16 = NullType, typename T17 = NullType, typename T18 = NullType >
struct MakeTypelist
{
private:
    typedef typename MakeTypelist<T2, T3, T4, T5, T6, T7, T8, T9, T10,T11, T12, T13,
                                  T14, T15, T16, T17, T18 >::Result TailResult;
public:
```

```
    typedef Typelist<T1, TailResult> Result;
};
template<> struct MakeTypelist<>
{
    typedef NullType Result;
};
int main()
{
    // myType : TypeList<int, TypeList<char, TypeList<double, NullType>>

    typedef MakeTypelist<int, char, double>::Result myType;

    cout << typeid(myType).name() << endl;
}
```

- **TypeList 의 개수 구하기 - Length**

아래의 Length template을 사용하면 TypeList가 보관하는 타입의 개수를 구할 수 있습니다.

```
template <class TList> struct Length;

template <> struct Length<NullType>
{
    enum { value = 0 };
};
template <class T, class U> struct Length< Typelist<T, U> >
{
    enum { value = 1 + Length<U>::value };
};
int main()
{
    typedef MakeTypelist<int, char, double>::Result myType;

    cout << Length<myType>::value << endl; //3
}
```

- **TypeList에 추가하기**

Linked List의 기본은 값을 추가 할수 있어야 합니다. TypeList도 Type을 저장 하는 list 이므로 기존의 TypeList의 끝에 새로운 Type 또는 TypeList를 추가하는 Append를 생각해 봅시다.

```
template <class TList, class T> struct Append;

template <> struct Append<NullType, NullType>
{
    typedef NullType Result;
};
template <class T> struct Append<NullType, T>
{
    typedef Typelist<T, NullType> Result;
};
template <class Head, class Tail>
struct Append<NullType, Typelist<Head, Tail> >
{
    typedef Typelist<Head, Tail> Result;
};

template <class Head, class Tail, class T>
struct Append<Typelist<Head, Tail>, T>
{
    typedef Typelist<Head,
        typename Append<Tail, T>::Result>
        Result;
};
int main()
{
    typedef MakeTypelist<int, char, double>::Result myType;

    // myType 끝에 short를 추가한 타입을 만듭니다.
    typedef Append<myType, short>::Result myType2;

    cout << typeid(myType2).name() << endl;

    // myType2  2개를 결합한 타입을 만듭니다.
    typedef Append<myType2, myType2>::Result myType3;

    cout << Length<myType3>::value << endl; // 8
}
```

1~3의 코드를 하나의 파일로 만들어서 테스트해보세요.. 그리고, 다음장의 과제를 해결해 보세요.

과제. 아래 코드가 수행되도록 TypeAt, IndexOf, Erase, Replace 를 만들어 보세요

```
int main()
{
    typedef MakeTypelist<int, char, double, short>::Result myType;


    // (1) TypeAt : N번째 타입이 무슨 타입인지 알아내는 템플릿 입니다.
    TypeAt<myType, 1>::Result  n; // n은 char 타입이어야 합니다.


    // (2) IndexOf : 특정 타입이 몇번째 있는지 찾아 내는 템플릿입니다.
    cout << IndexOf<myType, double>::value << endl; // 2가 나와야 합니다.


    // (3) Erase : 특정 타입을 제거한 새로운 TypeList 만들기
    //최초 발견된것 하나만 제거
    typedef Erase<myType, char>::Result myType2;

    cout << typeid(myType2).name() << endl; // int, double, short 의 TypeList입니다.


    // (4) Replace : 특정 타입을 다른 타입으로 교체한 TypeList 만들기
    typedef Replace<myType, char, float>::Result myType3;
    cout << typeid(myType3).name() << endl; // int, float, double, short
}
```