# 창의적 소프트웨어 설계 실습 문제 12 – hw12-2

**제출 기한**

12월 10일 월 11:59 PM

1. hw12-2(mkdir hw12-2)라는 폴더를 만들고 GitLab에 push
2. hw12-2 디렉토리에 CMakeLists.txt과 각 문제에서 요구하는 파일들을 작성
3. **‘cmake’ 와 ‘make’명령을 수행하여 숙제가 모두 빌드**
4. 최종 버전을 GitLab에 commit
5. **시간과 파일명, 입력과 출력 방식 반드시 지키기**

## <단순연결리스트>

class template을 이용하여 단순연결리스트(singly lined list)를 작성해보자. 단순연결리스트는 다음과 같이 동작한다.

1. 먼저, 연결리스트는 다음의 templated Node 클래스 오브젝트들을 저장한다.

**template** <**class** T>

**class** **Node**

{

**public**:

**T** data;

**Node**<**T**>\* next;

};

1. templated 를 Node클래스를 저장하는 class template, ‘Lis’t를 다음과 같이 구현하시오.

template <class T>

class List {

private:

Node<T> \*head;

public:

List() : head(NULL) {};

~List();//free resources

List(T\* arr, int n\_nodes);//create a list with n\_nodes

//List &operator=(const List &rhs);//assignment operation

void insert\_at(int idx, T& data);

void remove\_at(int idx, T& data);

void pop\_back();

void push\_back(T& val);

void pop\_front();

void push\_front(T& val);

ostream& operator<<(ostream& out, list& rhs);//리스트 내 노드값 출력

};

1. 구현한 리스트를 이용하여 다음의 예제 코드가 잘 동작하는지를 확인하시오.

int main(){

int array[5] = {12, 7, 9, 21, 13 };

List<int> li(array, 5);

cout<<li; ***//12,7,9,21,13***

li.pop\_back();

cout<<li; ***//12,7,9,21***

li.push\_back(15);

cout<<li; ***//12,7,9,21,15***

li.pop\_front();

cout<<li; ***//7,9,21,15***

li.push\_front(8);

cout<<li; ***//8,7,9,21,15***

li.insert\_at(4, 19);

cout<<li; ***//8,7,9,21,19,15***

li.remove\_at(1);

cout<<li; ***//8,9,21,19,15***

return 1;

}

## <스마트 포인터>

C++에서 제공하는 스마트 포인터(smart pointer)를 사용하면 메모리 누수 문제로부터 해방될 수 있다. 본 실습에서는 shared pointer(shared\_ptr) 개념을 이해하고, 이를 바탕으로 사용이 끝난 메모리를 자동으로 해제해주는 class template, My\_sharepd\_ptr를 구현해 보도록 한다.

**여러가지 스마트 포인터 중, shared\_ptr클래스는 그림 1과 같이, shared\_ptr클래스 오브젝트들(obj1,…,obj4)의 두 멤버 변수, m\_obj와 count가 동일한 물리 공간 Mem1과 Mem2를 각각 참조한다. 즉, 여러 개의 오브젝트들이 Mem1의 내용을 공유하고, 현재 Mem1을 참조하고 있는 오브젝트의 개수가 Mem2에 기록되어 있다. 가령, 첫번째 오브젝트 obj1이 생성될 때, Mem1이 할당되고, Mem2에는 숫자 ‘1’ 이 기록 된다. 다음으로, 4개의 오브젝트, obj1, obj2, obj3, obj4 가Mem1을 동시에 참조하는 경우, Mem2에는 숫자 4가 기록 된다. 최종적으로Mem1과 Mem2를 참조하는 유일한 obj가 소멸되는 순간 Mem1, Mem2도 함께 반환된다.**

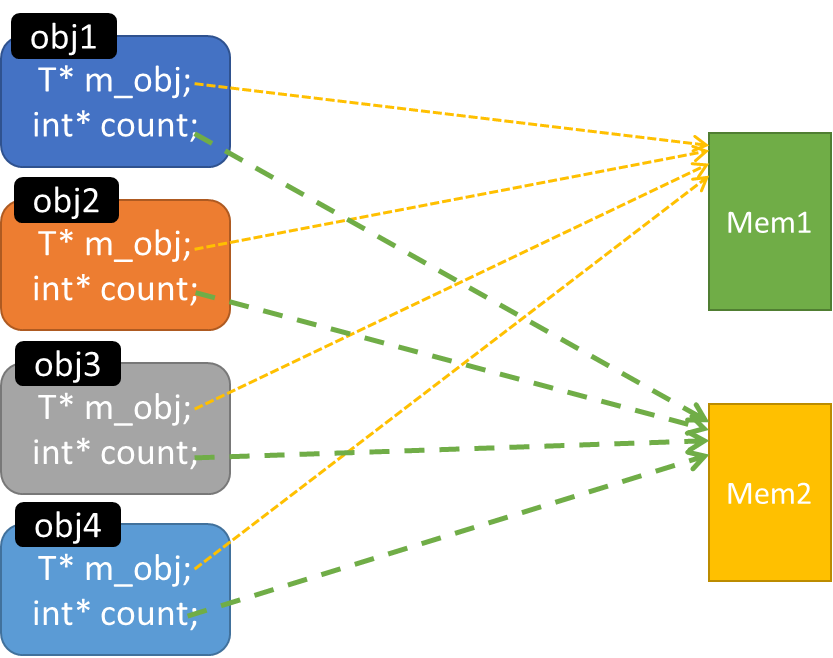


그림1

**My\_shared\_ptr를 사용하는 프로그램은 다음 표1과 같이 동작한다.**

|  |
| --- |
| **My\_shared\_prt: 여러 오브젝트가 특정 메모리를 동시에 참조할 수 있지만, 실제 메모리의 할당과 해제는 단 1회만 수행된다. 특히, getCount() 멤버 함수는 같은 리소스를 참조 중인 오브젝트의 개수를 리턴하고, 참조자의 수가 0이 되는 순간 할당된 리소스를 반환한다.** |
| {  **My\_shared\_ptr**<**int**> ptr01(new **int**(5));// int형My\_shared\_ptr인 ptr01을 선언하고 초기화함.  cout<<ptr01.getCount()<<endl;// 리소스 참조자 수=1  {  **My\_shared\_ptr**<**int**> ptr02(ptr01);// 복사 생성자를 이용한 초기화, ptr02=ptr01  cout<<ptr01.getCount()<<endl;// 리소스 참조자 수=2  cout<<ptr02.getCount()<<endl;// 리소스 참조자 수=2  }  cout<<ptr01.getCount()<<endl;// ptr02가 사라지며 참조자 수 감소, 참조자 수=1  }//prt01이 사라짐에 따라 참조자 수=0이 되고, new int(5)에 할당된 메모리가 마침내 해제 |

표 1

실습 내용: 아래와 같이 **My\_shapred\_ptr**를 클래스 템플릿을 이용하여 작성하시오.

template <class T>

class **My\_shared\_ptr**

{

private:

T\* m\_obj = NULL;

int\* count = NULL;

public:

**My\_shared\_ptr**();

**My\_shared\_ptr**(**T**\* obj);

**My\_shared\_ptr**(**const** **My\_shared\_ptr**& rhs);

~**My\_shared\_ptr**(){

//…

if(getCount() == 0) cout << “everything destroyed”

}

**My\_shared\_ptr**<**T**>& operator=(**const My\_shared\_ptr**<**T**>& rhs)

{

//…

**return** \*this;

}

**const T**\* get\_m\_obj(){ **return** m\_obj;}

int getCount();

void increase();

void decrease();

};

* **My\_shared\_ptr**는멤버 변수 **T**\* m\_obj와 **int**\* count 를 가진다.
* 디폴트 디폴트 생성자 (no parameters) 는 m\_obj와 count를 NULL로 초기화한다. Generic type **T**의 포인터를 받는 생성자, **My\_shared\_ptr** (**T**\* obj)에서는 m\_obj = obj, \*count=1로 초기화 한다.
* \*count 값을 1씩 증가시키는increase(), 1씩 감소시키는 decrease(), 그리고 \*count를 리턴하는 멤버함수 getCount()를 작성한다.
* **My\_shared\_ptr**의 복사 생성자 ( **My\_shared\_ptr**(const My\_shared\_ptr& rhs) )를 정의한다. **My\_shared\_ptr**의 복사 생성자는 m\_obj = obj로 주소 값만 복사한다.이 때, count와\*count를 잘 처리할 것)
* 소멸자 ~**My\_shared\_ptr**() 는 \*count = 0 이 되는 경우에 모든 리소스를 반환한다. 그렇지 않은 경우도 잘 처리할 것
* assignment operator (**My\_shared\_ptr<T>**& operator=(const **My\_shared\_ptr<T>**& rhs), 예제 코드 참조)를 정의 하시오. (hint: count = NULL, \*count =1, \*count>1 인 경우, 각각 다르게 처리해야 함)
* 다음 예제 코드(main 함수)를 수행하여 실제 출력 값과 예제 코드에 표기된 출력 값이 같은 지 확인한다.

**int** main()

{

{

**My\_shared\_ptr**<**T**> a;

cout<<a.getCount()<<endl;

{

**My\_shared\_ptr** <**T**> b(new T(5));

cout<<a.getCount()<<endl;

cout<<b.getCount()<<endl;

if(a.get\_m\_obj() == b.get\_m\_obj())

cout << "resource shared" << endl;

a = b;//assignment operation

cout<<a.getCount()<<endl;

cout<<b.getCount()<<endl;

if(a.get\_m\_obj() == b.get\_m\_obj())

cout << "resource shared" << endl;

}

cout<<a.getCount()<<endl;

}

return 1;

}

|  |
| --- |
| *Output:*  0  0  1  2  2  resource shared  1  everything destroyed |