MyBoolVector

2020203071 이강우

2023/10/12

목차

1. 이번 과제의 주제 및 핵심 키워드

이번 과제의 주제는 STL 에 있는 vector<bool>과 비슷한 기능을 하는 class 를 구현하는 것이었다. 그리하여 이것을 구현하기 위해서는 class 의 생성자와 소멸자에 대한 이해, 멤버변수의 초기화 그리고 연산자 오버로드의 구현이 이번 과제의 핵심 키워드였다고 생각한다. 이 키워드들을 활용하여 이번 과제를 해결해 나갈 수 있었다

2.MyBoolVector.h

```h
#pragma once
#include <iostream>
#include <algorithm>
#include <cassert>

class MyBoolVector {
public:
    MyBoolVector() :data(nullptr), capa(0), used(0) {};
    //precondition :
    //postcondition : 빈 상태로 MyBoolVector 객체 생성
    ~MyBoolVector();
    //precondition : MyBoolVector 객체 존재
    //postcondition : MyBoolVector 객체 동적 할당 해제
    MyBoolVector(const MyBoolVector& v);
    //precondition : deep copy 할 MyBoolVector v 존재
    //postcondition : 객체 v 복사 후 객체 생성

    MyBoolVector& operator =(const MyBoolVector &v);
    //precondition : deep copy 할 MyBoolVector v 존재
    //postcondition : deep copy 하여 복사이동

    MyBoolVector operator +=(const MyBoolVector& v);
    //precondition : 두 백터 객체 존재
    //postcondition : 왼쪽 벡터에 오른쪽 벡터 추가
    bool operator[](size_t index) const;
    //precondition : index가 범위안에 존재해야함
    //postcondition : 특정 data값 얻음
    MyBoolVector operator +(const MyBoolVector& v1);
    //precondition : v1과 v2의 used 값이 일치해야함
    //postcondition : v1과 v2 데이터의 XOR값 얻음
    MyBoolVector& operator-();
    //precondition : 백터 객체 존재
    //postcondition : 저장된 data의 반대값 반환
    bool operator==(const MyBoolVector& v1);
    //precondition : v1과 v2의 used 값이 일치해야함
    //postcondition : v1과 v2의 모든 data 값이 일치할 시 true 반환
    bool operator!=(const MyBoolVector& v1);
    //precondition : v1과 v2의 used 값이 일치해야함
    //postcondition : v1과 v2의 data 값이 불일치시 true 반환

    size_t size() const;
    //precondition : 백터 객체 존재
    //postcondition : 정수 used값 반환
    size_t capacity() const;
    //precondition : 백터 객체 존재
    //postcondition : 정수 used값 반환
    size_t capacity() const;
    //precondition : 백터 객체 존재
    //postcondition : 정수 capa값 반환
    void pop_back();
    //precondition : used 크기 1이상
    //postcondition : used 크기가 1이상이면 마지막 data값을 지우고, 아니면 바뀌지 않음
    void push_back(bool x);
    //precondition : 벡터 객체 존재
    //postcondition : bool x 값 추가
    void reserve(size_t n);
    //precondition : capcity용량보다 n이 커야함
    //postcondition : used보다 n이 크면 정수 n 만큼 capa 할당 아니면 그대로 반환
    bool is_empty() const;
    //precondition : 백터 객체 존재
    //postcondition : 사이즈가 0이면 true반환 아니면 false 반환
    void clear();
    //precondition : 백터 객체 존재
    //postcondition : data값과 used, capa 0으로 초기화

private:
    bool* data;
    int used;
    int capa;
};
```

MyBoolVector.h 에 class MyBoolVector 를 정의하고 그에 대한 멤버변수와 멤버함수를 선언하였다. public 영역에는 멤버함수들을 선언하였고 private 영역에는 멤버 변수들을 선언하였다.
멤버함수들은 위의 사진처럼 그에 대한 precondition 과 postcondition 을 명시하였다.

3.MyBoolVector.cpp

MyBoolVector.cpp에는 MyBoolVector.h에 선언하였던 멤버함수, 연산자, 생성자 그리고 소멸자를 정의하였다. 그 중 멤버함수, 연산자 그리고 생성자를 각각 하나씩 예를 들어 대표로 코드를 설명하겠다.

copy constructor

```
MyBoolVector::MyBoolVector(const MyBoolVector& v) {
    used = v.used;
    capa = v.capa;
    data = new bool[v.capa];
    for(int i=0;i<v.size();i++)
        data[i] = v.data[i];
}
```

생성자 중 copy constructor를 대표로 설명하면, 위 사진처럼 used와 capa에 v의 used의 값과 capa의 값을 대입하였고 포인터 변수 data에 v.capa 만큼 동적할당 해주고 v.data 값을 data에 대입하여 copy constructor를 구현하였다.

operator +=

```
MyBoolVector MyBoolVector::operator +=(const MyBoolVector& v)
{
    reserve(used + v.used);
    for (int i = 0;i < v.size();i++)
        data[used + i] = v.data[i];
    used += v.used;
    return *this;
}
```

연산자는 +=을 대표로 설명하겠다. 연산자 +=은 연산자 오른쪽의 벡터 객체를 왼쪽의 백터 객체에다가 추가하는 기능을 한다. 위 코드를 설명하면 reserve() 함수로 used+v.used의 크기만큼 capa를 할당 받고 왼쪽 벡터의 used 번째 data에서부터 v.data의 값을 할당 받는다. 그리고 used의 크기를 v.used만큼 더 키워주고 *this를 반환해준다. 이 과정을 거치면 +=의 기능이 구현된다.

reserve()

```
void MyBoolVector:: reserve(size_t n)
{
    if (n <= capa)
        return;
    bool* p= new bool[n];
    for (int i=0;i <size();i++)
         p[i] = data[i];
    delete[] data;
    data = p;
    capa = n;
}
```

멤버함수를 대표로 reserve()를 소개하겠다. 이 함수는 벡터의 capa를 n 만큼 하는 함수로 벡터의 용량을 바꿔주는 함수이다. 그래서 위 코드를 보면 n이 capa보다 클 때 유의미한 작동을 하게 된다. 코드를 살펴보면 만약 n<=capa 이면 바로 종료하게 함수를 종료하게 되고 그렇지 않으면 새로운 bool 포인터 p를 크기 n만큼 동적할당하고 그 p의 값에다가 data의 값을 대입한다. 그리고 포인터 data를 동적해제하고 data에 p값을 대입하고 capa에 n을 대입한다. 그러면 함수의 동작이 완료될 것이다.

4.main.c

main.c에는 테스트 케이스를 구현해 놓았다.

```
int main() {
    //default constructor, is_empty() 테스트
    MyBoolVector v0, v1;
    assert(v1.size() == 0);
    assert(v1.capacity() == 0);
    assert(v1.is_empty() == true);

    //push_back(), size() 테스트
    v1.push_back(true);
    v1.push_back(false);
    assert(v1.size() == 2);

    //copy assignment 테스트
    v0.push_back(false);
    v0 = v1;
    assert(v0 == v1);

    //capacity() 테스트
    assert(v1.capacity() >= 2);

    //operator[] 테스트
    assert(v1[0] == true);
    assert(v1[1] == false);

    //pop_back() 테스트
    v1.pop_back();
    assert(v1.size() == 1);
    assert(v1[0] == true);

    //clear() 테스트
    v1.clear();
    assert(v1.is_empty()==true);
    assert(v1.capacity() == 0);

    //copy constructor, operator== 테스트
    MyBoolVector v2;
    v2.push_back(true);
    v2.push_back(false);
```

```
    MyBoolVector v3(v2);
    assert(v3 == v2);

    //operator+ 테스트
    MyBoolVector v4, v5;
    v4.push_back(true);
    v5.push_back(false);
    MyBoolVector result;
    result = (v4 + v5);
    assert(result[0] == 1);

    //operator- 테스트
    MyBoolVector v6,v7;
    v6.push_back(true);
    v7=-v6;
    assert(v7[0] == false);

    //operator!= 테스트
    MyBoolVector v8,v9;
    v8.push_back(true);
    v9.push_back(false);
    assert(v8 != v9 );

    //operator += 테스트
    v8 += v9;
    assert(v8[0] == true);
    assert(v8[1] == false);

    //copy assignment의 chaining assignment 기능 테스트
    v7 = v8 = v9;
    assert(v7 == v9);
    assert(v8 == v9);

    cout << "모든 테스트 통과!₩n";
    return 0;
}
```

main.c에서 assert()를 통하여 모든 멤버함수와 연산자가 정상작동 되는지 확인하는 테스트 케이스를 만들었고, 디버깅을 통해 모든 멤버함수 및 멤버변수가 정상작동 하는지 확인하였다.