# C++ 과제4

MyVector Class 만들기

로봇게임단

19기 예비단원

이원준

# 1. 코드

- 헤더 파일

```
#ifndef MYVECTOR_H
#define MYVECTOR_H

#include <stdexcept>

template<typename T>
class MyVector
{
private:

    T* data;
    int size;
    int cap;

    void double_cap()
```

```
    {
        cap *=2;
        T* new_array = new T(cap);

        for(int i =0; i < size ; ++i)
        {
            new_array[i] = data[i];
        }
        delete []data;
        data = new_array;
    }

public:

    MyVector();
    ~MyVector();

    void My_push_back(T element);
    T My_at(int index) const;
    T My_begin() const;
    T My_end() const;
    bool My_empty() const;
    void My_erase(int index);
    void My_insert(int index, T element);
    int My_size() const;
```

```
    T& operator [](int index) const;

    // T& operator = (int index);
    // void operator + (int index);
};

template<typename T>
MyVector<T>::MyVector() : data(nullptr) , size(0), cap(1) {data = new T(cap);};

template<typename T>
MyVector<T>:: ~MyVector()
{
    delete[] data;
}

template<typename T>
void MyVector<T>::My_push_back(T element)
{
    if(cap > size)
    {
        data[size] = element;
        ++size;
    }
    else
    {
        double_cap();
```

```
            data[size] = element;
            ++size;
        }
}

template<typename T>
T MyVector<T>::My_at(int index) const
{
        if (index < 0 || index >= size)
        {
            throw std::out_of_range("Index out of range");
        }
        return data[index];
}

template<typename T>
T MyVector<T>::My_begin() const
{
        if (My_empty())
        {
            throw std::out_of_range("Vector is empty");
        }
        return data[0];
}

template<typename T>
```

```
T MyVector<T>::My_end() const
{
	if (My_empty())
	{
		throw std::out_of_range("Vector is empty");
	}
	return data[size - 1];
}

template<typename T>
bool MyVector<T>::My_empty() const
{
	return (size == 0);
}

template<typename T>
void MyVector<T>::My_erase(int index)
{
	if (index < 0 || index >= size)
	{
		throw std::out_of_range("Index out of range");
	}
	for (int i = index; i < size - 1; ++i)
	{
		data[i] = data[i + 1];
	}
```

```
	--size;
}

template<typename T>
void MyVector<T>::My_insert(int index, T element)
{
	if (index < 0 || index > size)
	{
		throw std::out_of_range("Index out of range");
	}

	if(cap > size)
	{
		for (int i = size; i > index; --i)
		{
			data[i] = data[i - 1];
		}
		data[index] = element;
		++size;
	}
	else
	{
		double_cap();

		for (int i = size; i > index; --i)
		{
```

```
                data[i] = data[i - 1];
            }
            data[index] = element;
            ++size;
        }
}

template<typename T>
int MyVector<T>::My_size() const
{
    return size;
}

template<typename T>
T& MyVector<T>::operator [](int index) const
{
    if (index < 0 || index >= size)
    {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

// template<typename T>
// T& MyVector<T>::operator = (int index)
// {
```

```
//         if (index < 0 || index >= size)
//         {
//                 throw std::out_of_range("Index out of range");
//         }
//         return data[index];
// }

// template<typename T>
// void MyVector<T>::operator + (int index,int other_index)
// {

// }

#endif // MYVECTOR_H
```

- 메인 코드

```
#include <iostream>
#include "MyVector.h"

using namespace std;

void choose_type_function();
template<typename T>
void test_myvector();

int main()
```

```
{
	choose_type_function();
}

void choose_type_function()
{
	while(1)
	{
		string option;

		cout << "choose vector's type" << endl;
		cin >> option;

		if(option == "int")
		{
			test_myvector<int>();
			break;
		}
		else if(option == "double")
		{
			test_myvector<double>();
			break;
		}
		else if(option == "char")
		{
			test_myvector<char>();
```

```
            break;
        }
        else
        {
            cout << "retry" << endl;
        }
    }
}

template<typename T>
void test_myvector()
{
    string new_vector;

    cout << "name new vector: " << endl;
    cin >> new_vector;

    MyVector<T> vec;

    string command;
    T element;
    int index;

    while(1)
    {
        cout << "command: " << endl;
```

```
cin >> command;


if(command == "pushback")
{
	cout << "element: " << endl;
	cin >> element;


	vec.My_push_back(element);
	cout << "pushback done" << endl;
}
else if(command == "at")
{
	cout << "Enter index to find: " << endl;
	cin >> index;
	cout << vec.My_at(index) << endl;
	cout << "at done" << endl;
}
else if(command == "begin")
{
	cout << "First element: " << vec.My_begin() << endl;
	cout << "begin done" << endl;
}
else if(command == "end")
{
	cout << "Last element: " << vec.My_end() << endl;
	cout << "end done" << endl;
```

```
        }
        else if(command == "empty")
        {
            cout << (vec.My_empty() ? "Vector is empty." : "Vector is not empty.") << endl;
            cout << "empty done" << endl;
        }
        else if(command == "erase")
        {

            cout << "Enter index to erase: " << endl;
            cin >> index;
            vec.My_erase(index);
            cout << "erase done" << endl;
        }
        else if(command == "insert")
        {

            cout << "Enter index to insert: " << endl;
            cin >> index;
            cout << "Enter element to insert: " << endl;
            cin >> element;
            vec.My_insert(index, element);
            cout << "insert done" << endl;
        }
        else if(command == "size")
```

```
            {
                cout << "Size of vector: " << vec.My_size() << endl;
                cout << "size done" << endl;
            }
            else if(command == "exit")
            {
                break;
            }
            else
            {
                cout << "retry" << endl;
            }
        }
    }
```

## 2. 코드 설명

### 2-1 헤더 파일 설명

- Class 선언의 private으로 선언된 멤버들을 설명하겠다

```
template<typename T>
class MyVector
{
private:

    T* data;
    int size;
    int cap;

    void double_cap( )
    {
        cap *=2;
        T* new_array = new T(cap);

        for(int i =0; i < size ; ++i)
        {
            new_array[i] = data[i];
        }
        delete []data;
        data = new_array;
    }
```

- myvector에서 정보를 동적할당된 배열의 형태로 저장하게 될 건데, 이때 data는 배열을 만드는데 사용되는 정수형 포인터이다.

- 벡터의 특성인 정보의 크기에 맞게 유동적으로 메모리를 할당하는 작업을 구현하기 위해 나머지 3개의 멤버가 사용된다
- 정수형 멤버 size와 cap은 각각 현재 배열의 크기와 동적할당된 벡터의 최대 크기를 의미한다.
- 벡터는 두 멤버의 대소비교를 통해 전체 벡터의 크기를 유동적으로 바꿀 수 있고 이 작업을 담당하는 멤버 함수가 double_cap() 함수이다.

```
public:

    MyVector();
    ~MyVector();

    void My_push_back(T element);
    T My_at(int index) const;
    T My_begin() const;
    T My_end() const;
    bool My_empty() const;
    void My_erase(int index);
    void My_insert(int index, T element);
    int My_size() const;

    T& operator [](int index) const;

    // T& operator = (int index);
    // void operator + (int index);
};
```

- 이제 public으로 선언된 멤버를 확인해보자
- 생성자를 통해 변수형 멤버를 초기화한다. 이때 cap을 1로 초기화 하는데 이는 벡터의 크기를 유동적으로 조절할 때 분해능을 높이기 위한 초반 세팅이다. 다음 사진을 참고하자
- ```
  template<typename T>
  MyVector<T>::MyVector() : data(nullptr) , size(0), cap(1) {data = new T(cap);};
  ```
- 벡터의 크기는 항상 cap을 기준으로 설정하고, double_cap() 멤버 함수를 통해 메모리 크기를 2배로 확장한다. 따라서 초반 값을 1로 세팅했을 때, 벡터의 메모리 크기는 1,2,4,8,16 배로 커질 것이고 이러한 세팅은 효율적인 메모리 관리는 돕는다.
- 나머지 멤버는 ppt 가이드 라인에 따라 추가했다
- 8개의 멤버 함수와 1개의 연산자를 구현하는 데 성공했고, 마지막 2개의 연산자 오버로딩은 실패했다. 이에 대해서는 후술하겠다.

- 일부 멤버의 재정의 코드에 대해 추가 설명하겠다.
  ```

```cpp
template<typename T>
void MyVector<T>::My_insert(int index, T element)
{
    if (index < 0 || index > size)
    {
        throw std::out_of_range("Index out of range");
    }

    if(cap > size)
    {
        for (int i = size; i > index; --i)
        {
            data[i] = data[i - 1];
        }
        data[index] = element;
        ++size;
    }
    else
    {
        double_cap();

        for (int i = size; i > index; --i)
        {
            data[i] = data[i - 1];
        }
        data[index] = element;
        ++size;
    }
}
```

- 인서트 함수를 실행시키면 2번의 검사가 진행되는데, 선택한 인덱스가 미리 정의된 인덱스 값 사이인지를 검사하고 새로 벡터를 인서트할 때 오버플로우가 발생하는 지를 점검한다.

```cpp
template<typename T>
T& MyVector<T>::operator [](int index) const
{
    if (index < 0 || index >= size)
    {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}
```

- 배열 연산자를 오버로딩은 해당 인덱스가 미리 정의된 인덱스 내부인지를 검사하고 리턴을 진행한다.

## 2-2. 메인 함수 설명

- 메인 함수는 크게 2개의 함수로 구성되고 이를 통해 벡터 헤더파일을 디버깅하고 테스트할 수 있도록 구성했다.

```
void choose_type_function( )
{
    while(1)
    {
        string option;

        cout << "choose vector's type" << endl;
        cin >> option;

        if(option == "int")
        {
            test_myvector<int>( );
            break;
        }
        else if(option == "double")
        {
            test_myvector<double>( );
            break;
        }
        else if(option == "char")
        {
            test_myvector<char>( );
            break;
        }
        else
        {
            cout << "retry" << endl;
        }
    }
}
```

- 이 함수는 벡터의 변수형을 선택할 수 있도록 구성된 choose_type_function() 함수이다.
- 메인문을 실행시키면 가장 먼저 돌아가는 함수로 벡터를 선언하고 테스트할 수 있는 test_myvector() 함수를 호출한다.

```
template<typename T>
void test_myvector()
{
    string new_vector;

    cout << "name new vector: " << endl;
    cin >> new_vector;

    MyVector<T> vec;

    string command;
    T element;
    int index;

    while(1)
    {
        cout << "command: " << endl;
        cin >> command;

        if(command == "pushback")
        {
            cout << "element: " << endl;
            cin >> element;

            vec.My_push_back(element);
            cout << "pushback done" << endl;
        }
        else if(command == "at")
        {
            cout << "Enter index to find: " << endl;
            cin >> index;
            cout << vec.My_at(index) << endl;
            cout << "at done" << endl;
        }
        else if(command == "begin")
        {
            cout << "first element: " << vec.My_begin() << endl;
            cout << "begin done" << endl;
        }
        else if(command == "end")
        {
            cout << "Last element: " << vec.My_end() << endl;
            cout << "end done" << endl;
        }
        else if(command == "empty")
        {
            cout << (vec.My_empty() ? "Vector is empty." : "Vector is not empty.") << endl;
            cout << "empty done" << endl;
        }
        else if(command == "erase")
        {

            cout << "Enter index to erase: " << endl;
            cin >> index;
            vec.My_erase(index);
            cout << "erase done" << endl;
        }
        else if(command == "insert")
        {

            cout << "Enter index to insert: " << endl;
            cin >> index;
            cout << "Enter element to insert: " << endl;
            cin >> element;
            vec.My_insert(index, element);
            cout << "insert done" << endl;
        }
        else if(command == "size")
        {
            cout << "size of vector: " << vec.My_size() << endl;
            cout << "size done" << endl;
        }
        else if(command == "exit")
        {
            break;
        }
        else
        {
            cout << "retry" << endl;
        }
    }
}
```

- 이 함수는 test_myvector() 함수로 템플릿으로 선언되어 첫번째 함수에서 변수형을 읽어 올 수 있다. 이를 바탕으로 벡터를 선언하고 명령어를 입력 받아 벡터 테스트를 진행한다.

## 3. 느낀점 및 고찰

- 마지막 2개의 연산자' = '와 '+ '를 구현하지 못했다. 먼저 두 연산자에 대한 이해도가 부족한 상태로 코드를 구상했다. 그렇기에 나머지 조건을 완성하고 마지막에 두 연산자를 추가하겠다는 계획을 세웠다. 하지만 나머지 9개의 멤버와 달리 나머지 두 개의 연산자는 2개 이상의 벡터를 생성하는 작업이 필수적이었고 이 부분을 구현하는 데 어려움이 있었다.
- 복수 개의 벡터를 생성하는데는 크게 2가지 방법으로 접근을 했다. 먼저 STL 자료구조의 map을 활용하여 벡터 테스트 함수 코드의 벡터 선언 부분에서 미리 지정된 변수의 이름을 변경하는 방법이다.  이런 방법을 채택하면 여러 개의 벡터를 쉽게 만들 수 있을 뿐만 아니라 사용자가 직접 벡터의 이름을 설정할 수 있다. 하지만 이러한 방법은 난이도가 높고 여러가지 버그를 발생시켜 디버깅을 하던 중 포기하기로 결정했다.
- 2번째 방법은 처음부터 여러 개의 벡터를 선언하고 시작하는 방법이다, 이러한 방법은 각각의 벡터마다 새로운 함수를 만들어야 하는 부담이 있고 좋은 방법이 아니라고 생각해서 포기하게 되었다.
- 개인적으로 쉽지 않았던 과제인 만큼 얻어 가는게 많다고 느꼈다.