<HW4 과제 보고서>

1. 과제 목표

: MyVector class 구현

2. 과제 진행 과정

2 - 1. 멤버변수 및 생성자 만들기

```
template<typename T>

class MyVector
{

private:
    T*vector_stack;
    T*temp_arr;
    int length;
    int capacity;
    int idx;

public:
    MyVector(const int& temp=10)
    {
        vector_stack = new T[temp];
        temp_arr = new T[temp];

        length = 0;
        capacity = temp;
    };
```

생성자와 멤버 변수는 위와 같이 설정했다. vector_stack 와 temp_arr 는 동적할당 배열로 선언했다. 만약 배열의 크기와 벡터안에 있는 요소의 크기가 같아지면 자동으로 늘어나게 하기 위해서이다.

2 - 2. 소멸자 만들기

```
~MyVector()
{
    delete[] vector_stack;
    delete[] temp_arr;
}
```

소멸자에서는 메로리할당을 해제해 메모리 누수가 일어나지 않도록 구성했다.

2 – 3. push_back 함수 만들기

```
void push_back(const T& data)
{
    if(length >= capacity)
    {
        T*tmep_arr = new T[2*capacity];

        capacity *= 2;

        for (int i = 0; i < length; ++i)
        {
            temp_arr[i] = vector_stack[i];
        }

        delete [] vector_stack;

        vector_stack = temp_arr;
    }

    vector_stack[length] = data;

    length++;
}
```

위 함수는 벡터의 마지막에 원하는 값을 넣을 수 있는 기능을 담당하고 있고, 만약 값을 넣었을 때 벡터의 크기보다 요소의 크기가 더 클경우 벡터의 크기를 2배 해줌으로써 요소의 크기가 벡터의 크기를 넘지 않도록 구성했다. 중간에 한변 배열을 복사해놓고 삭제한 후 재할당하는 과정을 거친다. 이러한 과정들이 끝나면 length 값을 ++해서 배열의 크기를 업데이트 한다.

2 – 4. at, begin, end, size, empty 함수 만들기

```
int at(int index)
{
    IndexError(index);

    return vector_stack[index];
}

int begin()
{
    return vector_stack[0];
}

int end()
{
    return vector_stack[length];
};

int size()
{
    return length;
}


bool empty()
{
    if(length != 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

위 사진에 보이는 것처럼 at 함수는 index 위치에 대한 값을 리턴하고. Begin 함수는 벡터의 첫번째에 위치한 값을 리턴하고, end 함수는 벡터의 끝에 있는 함수를 리턴한다. 마지막으로 empty 함수는 벡터가 비어있는지 여부를 판단해 bool 형식으로 리턴한다.

2 – 4. erase 함수 만들기

```
void erase(int index)
{
    if (index < 0 || index >= length)
    {
        std::cout << "index out of range";
    }

    else if(length < 1)
    {
        std::cout << "No data exists to be erased" << std::endl;
    }
    else
    {
        for(int i = index; i<= index-1; i++)
        {
            vector_stack[i] = vector_stack[i + 1];
        }
    }

    length--;
}
```

위 사진처럼 인덱스 값이 범위를 초과하지 않을 경우에 for 문을 돌려서 인덱스 부분에 왼쪽으로 한칸씩 이동해서 값을 덮어씌운다.

2 – 4. insert 함수 만들기

```
void insert(int index, const T& data)
{

    if (index < 0 || index >= length)
    {
        std::cout << "index out of range";
    }

    else if(length >= capacity)
    {
        T*tmep_arr = new T[2*capacity];

        capacity *= 2;

        for (int i = 0; i < length; ++i)
        {
            temp_arr[i] = vector_stack[i];
        }

        delete [] vector_stack;

        vector_stack = temp_arr;
    }

    // 배열 temp_arr에 저장
    for(int i = 0; i <= length; i++)
    {
        temp_arr[i] = vector_stack[i];
    }

    temp_arr[index] = data;

    for(int i = index; i <= length; i++)
    {
        vector_stack[i+1] = temp_arr[i];
    }

    length++;
}
```

앞서 설명한 것처럼 요소의 크기가 벡터의 크기와 같아지면 벡터의 크기가 두배가 된다. 우선 벡터를 빈공간에 저장해두고, 원하는 위치에 데이터를 삽입한다. 그 다음에 원래 벡터에서 index 를 기준으로 오른쪽 값을 temp_arr 에 저장되어있는 값으로 바꾼다.

2 – 4. 복사기능 구현

```
MyVector<T>& operator=(const MyVector<T>& other)
{
    if (this == &other)
    {
        return *this;
    }

    delete[] vector_stack;

    capacity = other.capacity;
    length = other.length;
    vector_stack = new T[capacity];

    for (int i = 0; i < length; ++i)
    {
        vector_stack[i] = other.vector_stack[i];
    }

    return *this;
}
```

위 사진처럼 연산자 오버로딩을 통해 복사하는 기능을 만들었다. if 문을 이용해 현재 객체와 대입할 객체가 동일할 경우 아무 작업 없이 반환하고, 기존의 메모리 할당을 해제한다 그 후에 새로운 메모리를 할당하여 데이터를 복사하고, 마지막에 현재 객체를 반환한다.

2 – 5. 벡터끼리 더하는 기능 구현

```
MyVector<T> operator+(const MyVector<T>& other)
    {
        MyVector<T> Addition_between_vectors(this->length + other.length);
        for (int i = 0; i < this->length; i++)
        {
            Addition_between_vectors.push_back(this->vector_stack[i]);
        }
        for (int i = 0; i < other.length; i++)
        {
            Addition_between_vectors.push_back(other.vector_stack[i]);
        }

        return Addition_between_vectors;
    }
```

일단 두 벡터의 요소들을 모두 담을 수 있는 새로운 벡터를 생성한다. 그 후에 현재 벡터의 요소와 다른 벡터의 요소를 각각 추가한 결과를 반환한다.

배운점 및 고찰

: 그냥 머리로 생각했을 때는 구조가 잘그려지지 않고 어떻게 시작할지 몰라 막막했었는데, 막상 시작하고 나니 생각했던것만큼 어렵지는 않았던 과제였다. 하지만 헤더파일과 구현 파일을 따로 만들었을 경우에 발생하는 오류를 해결하지 못해 그냥 헤더파일에 구현했던점이 상당히 아쉬웠다. 과제를 끝마치고 난 후에 내가 만든 코드들을 디버깅 해보면 코드를 짤때는 완벽하다고 생각했던 것들이 수정이 필요한 부분이 많다는 것을 느꼈고, 이를 통해 처음에 짤때부터 내가 짜고있는 코드가 올바른 방향으로 가고있는지 확인하는 과정이 필요하다고 느꼈다. 또한 여러 자료들을 찾아보면서 하나의 기능을 구현하는데에도 엄청나게 여러가지 방식으로 접근할 수 있다는 것을 여실히 느끼게 되어 코드를 짜기 전에 여러 알고리즘을 찾아보고 코드를 짜면 더 효율적일 것 같다는 생각을 하게 되었다.