**2023203093 Homework\_01**

**: STL <vector>와 유사한 MyStringVector 헤더를 만들기**

**컴파일 환경: Visual Studio 2022**

**Step 1. vector class 분석**

<vector>의 동작 원리를 고민하고, **어떤 변수와 함수가 멤버로 들어갈 수 있을 지 분석**하는 세션이다.

우선 <vector>는 크기가 늘었다가 줄었다가 하는 배열이다. **본인이 필요한 만큼만 공간이 증가하고, 필요가 없다면 공간을 감소시킬 수 있다.**

그렇다면 어떻게 vector를 늘렸다 줄였다 할 수 있을까? -> 문제에서 조건으로 제시했다.

최대 담을 수 있는 개수를 나타내는 capacity와 실제로 담긴 개수인 size 변수들로 메모리의 현재 상태를 관리한다. **size == capacity가 되면 더 큰 capacity를 가진 공간으로 모든 요소를 이동(복사) 시킨다.**

**Step 2. MyStringVector implementation**

이제 함수와 연산자를 하나하나 구성해보자.

1. **private 변수**

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

* 1. string\* data
  2. size\_t size
  3. size\_t capacity

가 private 변수로 들어갈 수 있다.

data는 **저장한 문자열들**, size는 **실제 문자열이 들어간 개수**, capacity는 **문자열이 최대로 들어갈 수 있는 개수를** 의미한다.

1. **생성자 (constructor) & 소멸자 (destructor)**

생성자는 크게 3가지 경우로 나눌 수 있다.

* 1. 기본 생성자
  2. capacity를 사전에 정하고 시작하는 생성자
  3. 기존의 vector를 그대로 복사하고 시작하는 생성자

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

기본 생성자는 (**들어간 string이 없고**), (들어간 string이 없으므로 **size도 0**) 이다. capacity는 0 또는 1로 설정 할 수 있는데, 이후 **통일성을 위해 0**으로 하는 게 좋겠다.

// Precondition: sufficient memory is available

// Postcondition: capacity is set to 0, size is set to 0, data is set to nullptr

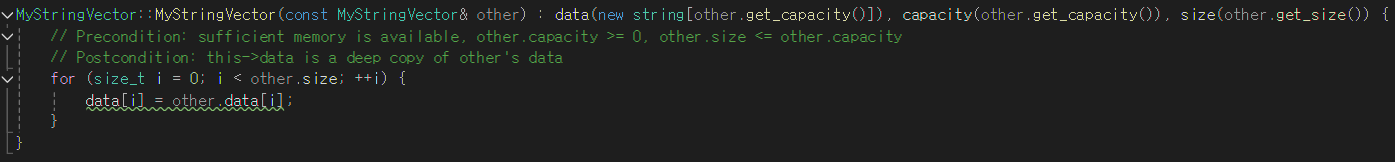
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

capacity를 사전에 정하고 시작하는 생성자는 입력한 **capacity만큼의 공간을 가진 배열을 만들어야 한다**. (data = new string[capacity]) 다음으로 capacity를 입력한 capacity로 바꿔줘야 한다. this->capacity = capacity. (여담으로, 입력 변수명과 private 변수 명이 다르더라도 this->capacity를 사용하면 초기화가 가능하다.)

// Precondition: sufficient memory is available, capacity >= 0

// Postcondition: capacity is set to the given value, size is set to 0, data is allocated



기존 MyStringVector를 그대로 복사하고 시작하는 생성자는 = 연산자를 이용해야 한다. 구현은 하단에서 할 예정이다. 여기서는 =연산자가 구현이 완료가 되었다고 생각한다. 우선 capacity, size를 other의 멤버 함수 **getter로 가져와 this의 capacity, size를 초기화한다**. 기존 벡터를 그대로 복사 하므로 기존 데이터를 끌어다 쓴다. (입력 vector의 size) 만큼 **data[index] = other.data[i];** 해 주어야 한다. 하단에 caution이 뜨는 이유는 잘못된 데이터를 읽고 있기 때문인데, 이는 test file에서 [] operator의 잘못된 인덱싱 접근을 테스트 하기 때문이다. (신경 쓸 필요 없다.)

// Precondition: sufficient memory is available, other.capacity >= 0, other.size <= other.capacity

// Postcondition: this->data is a deep copy of other's data

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

소멸자는 단순하게 데이터 동적 할당을 제거하면 된다. **delete[] data;**

capacity와 size를 초기화 하지 않는 이유는 clear()와 다르게 **더 이상 MyStringVector를 사용하지 않기 때문이다**.

// Precondition: constructor has been called

// Postcondition: memory allocated for data is released

1. **memory management와 관련된 함수**

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

reserve(size\_t new\_capacity)는 **new\_capacity까지 vector의 수용 크기를 증가**시킨다. **기존 capacity보다 작은 경우에만 더 큰 수용 공간이 필요하므로** new\_capacity > capacity 일 때만 새 배열로 이동한다.

// Precondition: new\_capacity >= size, vector is valid, if data is nullptr capacity is 0

// Postcondition: capacity is increased to new\_capacity, data is moved to a larger block of memory, size is unchanged

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

shrink\_to\_fit()은 **size와 capacity의 갭을 없앤다**. size == 0인 경우 배열을 지워버리고, 그 외의 경우 (size != 0)에는 **size == capacity**인 새 배열을 만든다.

// Precondition: vector is valid if size > 0

// Postcondition: capacity is reduced to the current size, or deallocated if size is 0, size is unchanged

1. **data management와 관련된 함수**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

push\_back(const string& value)는 **string value의 값을 다음 data에 집어넣는 함수이다**. 우선, size – capacity 관계가 유효한지 확인해야 한다. 즉, 집어넣을 공간이 있어야 한다. 이는 **size < capacity**를 의미한다. size >= capacity인 경우 reserve(new\_capacity); new\_capacity는 capacity == 0이면 1, capacity != 0 이면 기존 capacity의 2배로 한다. -> **reserve(capacity == 0 ? 1 : capacity \* 2)**; 다음에는 data[size] = value;로 값을 저장한다. 이후 size를 ++해야 하는데, **이를 합쳐서 data[size++] = value;**로 사용하였다. 이렇게 하면 data[size] = value; 를 처리한 뒤 size가 1 증가해 **자동으로 다음 인덱스 세팅**이 된다.

// Precondition: vector is valid, sufficient memory is available

// Postcondition: value is added to the vector, size is incremented, reserve is called if necessary

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

pop\_back()은 매번 delete 메모리 하는 과정 대신 **단순히 size만 줄여 안에 들어있는 값을 garbage 값으로 취급**하겠다는 얘기이다. 어차피 push\_back 또는 = 연산자 등등 **모든 값 추가 과정에서 값을 대입하기 때문에 가능하다**. 주의할 점은, **size == 0인 경우 뽑아낼 string이 없기 때문에, error를 throw** 해야 한다.

// Precondition: size > 0

// Postcondition: the last element is removed, size is decremented

1. **getter() & setter()**

OOP 원칙을 지키기 위한 함수들이다.

getter는 **private 변수를 return**하기만 하면 된다.

HW1.pdf의 요구 사항중 size\_t capacity() const, size\_t size() const 함수들이 여기에 해당한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

get\_size() const ->

// Precondition: vector is valid

// Postcondition: returns the number of elements in the vector, vector is unchanged

get\_capacity() const ->

// Precondition: vector is valid

// Postcondition: returns the capacity of the vector, vector is unchanged

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

setter는 **private 변수들을 입력한 new\_priviate 변수들로 초기화**한다.

set\_size(size\_t new\_size) ->

// Precondition: new\_size <= capacity

// Postcondition: size is updated to new\_size

set\_capacity() ->

// Precondition: new\_capacity >= size

// Postcondition: capacity is updated to new\_capacity

주의할 점은, set\_size()의 인자 size\_t new\_size가 현재 capacity보다 작은 경우는 현재 설정한 최대 용량을 넘기므로 **out\_of\_range error를 throw**한다.

마찬가지로 set\_capacity()의 인자 size\_t new\_capacity가 현재 size보다 작을 경우 최대 용량보다 담을 게 많은 상황이 된다. **out\_of\_range error를 throw**한다.

1. **data status와 관련된 함수**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

is\_empty() 는 bool type으로, 배열의 원소가 하나도 없는지 묻고 있다. 이는 size 변수와 연관이 되어있으므로 **return (size == 0)**; 을 한다.

// Precondition: vector is valid

// Postcondition: returns true if size is 0, otherwise false, vector is unchanged

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

clear()는 **vector의 모든 요소를 없애버리는 함수**이다. delete[] data;로 배열을 삭 제하고, data = nullptr;로 초기화한다. 다음으로 size, capacity를 0으로 한다.

// Precondition: vector is valid

// Postcondition: all elements are removed, size and capacity are set to 0

1. **operator overloading**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

operator[]은 **[] 안에 index를 넣으면 index + 1 번째에 해당하는 원소를 반환**한다. 이때, 입력 index가 size보다 크거나 같으면 이에 대응하는 원소가 없으므로 **out\_of\_range error를 throw**한다.

여기서 중요한 점은 operator[]의 경우 **string&을 리턴하는 함수와 const string&을 리턴하는 함수를 따로 만들어야 한다**는 점인데, 이는 데이터 보호와 수정 가능성을 제어하기 용이하기 때문이다.

예를 들어, MyStringVector vec; vec[0] = “Hello, World!”; 라고 해 보자. 이때 operator[]은 **비-const로 선언된 객체에 대해 호출**이 되고, 반환된 참조를 통해 해당 vector의 원소를 수정이 가능하다.

반대로, MyStringVector vec; vec[0] = “Hello, World!”; std::cout << vec[0]; 라고 해 보자. 이때 operator[]은 **const로 선언된 객체에 대해 호출**이 되고, 반환된 참조를 통해 읽기 작업만 가능하다.

이와 같은 이유로 const에 대한 함수와 비-const에 대한 함수가 모두 필요하다.

// Precondition: index < size

// Postcondition: returns reference to the element at position index

텍스트, 소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

operator=은 **RHS의 값을 LHS의 자리에 복사**하는 함수이다. 주의할 점은, LHS에 위치한 것과 RHS에 위치한 것이 달라야 한다는 점이다. 즉, 대입하려고 하는 게 서로 다른 녀석이어야 한다는 건데, 처음 보았을 때는 무슨 차이인가 싶을 수 있다. 어차피 대입을 실행하고 리턴하면 내용이 동일할텐데 굳이 오류 처리를 하는 이유가 뭘까? -> 바로 **메모리 할당과 해제의 작업에서 비정상적인 동작이 발생** 할 수 있기 때문이다.

텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

예를 들어, 대입 연산자가 다음과 같은 방식으로 구현이 되어있다 가정 해 보자. 이때 **vec = vec 같은 자기 대입**이 일어나면 다음과 같은 문제가 발생할 수 있다.

1. 기존 메모리 해제: delete[] data로 기존 메모리가 해제된다. 이때 rhs의 data 역시 this와 동일한 객체를 가리키므로, **rhs.data가 가리키는 데이터도 삭제**된다.
2. 새 메모리 할당과 복사: 그 다음 rhs.data로부터 복사하려고 하지만, 이미 그 **데이터는 삭제된 상태**이므로 비정상적인 동작이 발생할 수 있다.

위와 같은 문제를 해결하기 위해 가장 먼저 **this와 rhs가 동일한지 확인**한다.

// Precondition: other is a valid MyStringVector object, does not point to this, sufficient memory is available

// Postcondition: this vector is a deep copy of other, freeing any memory previously allocated, if this == &other, no change

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

operator+=은 **입력한 RHS를 현재 vector의 맨 끝에 이어 붙이는 함수**이다.

capacity가 부족하다면 **reserve()를 이용해 충분한 공간을 먼저 만들어주고**, 다음으로 push\_back()을 이용해 RHS의 모든 요소들을 뒤에 추가한다. 마지막으로 size를 업데이트한다.

**Step 3. HW1\_2023203093.cpp implementation**

이제 오류 체크 파일을 만들어보자**. try – catch 구문을 활용해 에러가 생기는 부분을 처리 한다.** 그리고 모든 에러가 생길 수 있는 부분을 **errTest class**로 묶어 정리를 한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testDefaultConstructor()**

* default constructor를 테스트 하는 함수이다.

vec의 size, capacity가 0으로 초기화 되어 있으면 정상, 아니라면 error를 반환한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testCapacityConstructor(size\_t capacity)**

* capacity를 입력하는 constructor를 테스트 하는 함수이다.

vec의 size가 0, capacity가 입력한 capacity로 초기화 되어 있으면 정상, 아니라면 error를 반환한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **testCopyConstructor(const myStringVector& original)**

* 이전 vector와 똑같은 상태에서 시작하는 constructor를 테스트 하는 함수이다.

“copy”라는 이름의 MyStringVector에 입력한 original MyStringVector를 생성한다. 이후 copy와 original의 size, capacity, element 를 비교한다. 다르면 제대로 된 복사가 이루어지지 않았음을 알 수 있다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testShrinkToFit(MyStringVector& vec)**

* shrink\_to\_fit()을 테스트 하는 함수이다.

vec를 shrink\_to\_fit() 한 뒤 capacity와 size가 동일한지 확인한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testReserve(MyStringVector& vec, size\_t new\_capacity)**

* new\_capacity 보다 크거나 같게 vec의 capacity가 설정이 되었는지를 확인한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testPushBack(MyStringVector& vec, const string& value)**

* vec.push\_back(value)를 한 뒤 vec의 마지막에 value가 저장이 되어있고, size가 1 증가 하였는지 확인한다.

텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

1. **testPopBack(MyStringVector& vec)**

* vec의 size가 0보다 크고, vec.pop\_back()을 한 뒤 vec의 size가 1 감소 하였는지 확인한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **testIsEmpty(MyStringVector& vec)**

* vec이 비었고 size가 0인 경우 또는 vec이 안 비었고 size가 0이 아닌 경우 두 가지를 테스트 한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **testClear(MyStringVector& vec)**

* vec을 clear() 한 뒤 size와 capacity가 0인지 확인한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testSizeGetterSetter(MyStringVector& vec, size\_t new\_size)**

* 우선 new\_size가 현재 capacity를 초과하는지 검사한다. 그리고 set\_size(new\_size)를 한 뒤 vec.get\_size() == new\_size인지 검사한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **testCapacityGetterSetter(MyStringVector& vec, size\_t new\_capacity)**

* set\_capacity(new\_capacity)를 한 뒤 vec.get\_capacity() == new\_capacity인지 검사한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. **testAssignmentOperator(MyStringVector& lhs, const MyStringVector rhs)**

* lhs = rhs를 한 뒤 size, capacity, element가 동일한지 모두 검사한다.

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. **testOperatorIndex(MyStringVector& vec, size\_t index)**

* 우선 index가 유효 범위인지 확인한다. index < vec.get\_size(), 다음으로 복사가 잘 되는지 확인한다. string value = vec[index];

main() 에서는 **compile test, output test, additional test를 하는 3구역**으로 나누었다. compile test에선 각 함수가 compile하면서 제대로 된 상황에서 **정상 작동** 하는지, error를 반환해야 하는 상황에서 **error를 반환** 하는지를 테스트 하며, passed 또는 failed: 원인 의 형식으로 작동을 확인한다. output test 구역에서는 멤버 함수들을 실행 후 변해야 하는 값들이 **제대로 변하는지**를 테스트 하는 구역이다. 예를 들어, pop\_back()을 수행 후 size가 변하였는지 확인해야 한다. additional test는 **chaining assignment, printing error message**를 테스팅 하는 구역이다.

1. **compile test**

상단 test~ 함수들을 한 번씩 작동 시켜 보는 함수이며, 로직에 차이는 없다. 실행 결과 화면만 보여주고 생략하겠다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. **output test**

각 멤버 함수들을 실행 시킨 뒤 바뀌어야 하는 값들이 제대로 바뀌었는지 테스트 해 본다. 테스트 범위는 **헤더에 있는 모든 함수들**이다.

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



스크린샷, 텍스트이(가) 표시된 사진

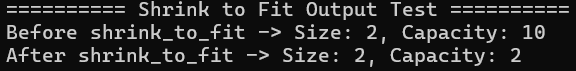
자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

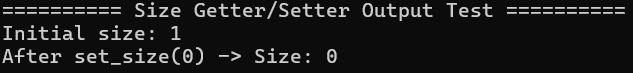
자동 생성된 설명

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

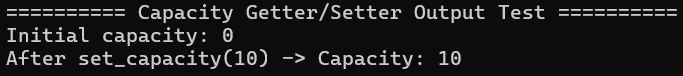
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 블랙이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* 다음과 같이 완벽히 작동함을 알 수 있다.

1. **additional test**

HW1.pdf 에 있는 함수의 특별한 요구사항을 테스트 하는 구역이다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* 요구사항 자체가 범위를 벗어난 index 입력시 error 메세지를 출력하고 프로그램 종료이다. 원본 파일에는 주석처리를 해 놓았고, 이번엔 결과를 확인하기 위해 주석을 빼고 프로그램을 컴파일 한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**Step 4. 느낀점**

1. **자료구조의 기본기**: 자료구조는 무궁무진한 활용이 있습니다. 이론적인 내용을 실제 코드로 구현 해 볼 때 프로그램의 동작 방식에 대한 생각이 명확해지고 기초 지식이 탄탄해짐을 느꼈습니다.
2. **예상보다 어려운 구현 난이도**: 간단해보이는 자료구조도 실제로 구현하면 생각보다 복잡하다는 것을 느꼈습니다. 특히 그저 간단하게 요소만 추가 삭제 하면 되는 것 아닌가? 했던 vector에 사실 배열의 size, capacity 관리법과 꽉 차면 더 큰 배열을 할당해 이동 시킨다는 숨겨진 원리에 대해 알게 되었습니다. 비록 디버깅 하는 데 많은 시간을 먹고 좌절감도 많이 느꼈지만, 성취감을 느꼈습니다.
3. **메모리 관리**: 정적 배열과 동적 배열의 가장 큰 차이는 “메모리 이득” 이라고 생각하기 때문에, 메모리 관리와 최적화에 가장 중점을 두고 코드를 짰습니다. 이를 고민하는 과정에서 메모리 누수에 대한 개념과 중요성을 체감할 수 있었습니다.

**이상 마치겠습니다.**

**2023203093 김수오**