Data Structure HW#1 Report

2016706018 박건희

모든 테스트는 visual studio 2019 버전에서 진행되었습니다.

먼저 선언부를 살펴보고 테스트코드 실행을 하면서 구현부를 살펴보는 방식으로 진행되었다.

헤더파일에는 동적할당을 위한 cstdlib

배열의 복사를 위한 algorithm,

버그 발생을 방지하기 위한 cassert

위 3가지 헤더파일이 선언되었다.

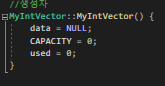
Private에 data의 포인터와 capacity, used가 선언되었다.

Used는 MyIntVector의 size와 동일하다.

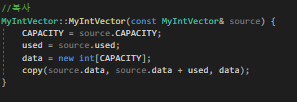
public에는 생성자, 소멸자, 각종 멤버함수들이 선언되었다.

생성자는 default와 deep copy로 2가지를 구현하였다.

Precondition과 postcondition에서 precondition엔 입력이 있는 경우와 MyIntVector객체에서 특정 조건이 있는 경우 표시하였고 postcondition에 함수의 결과를 서술하였다.

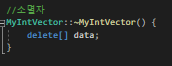


Default 생성자는 아무것도 안 들어있는 벡터이므로 data에 메모리를 할당하지 않고, capacity와 size가 0이다.



Deep copy를 위하여 data의 메모리를 새로 할당한 후 copy함수를 통하여 source의 data 배열을 새로 할당된 data에 복사하였다.

소멸자는 객체가 소멸시 동적할당 되었던 data배열을 반환하도록 구현하였다.



그 다음 main문에 만들어 놓은, 테스트코드를 살펴보면 원하는 갯수만큼 입력받은 후 객체를 동적할당하여 배열 형태로 관리하기 쉽게 하도록 하였다.

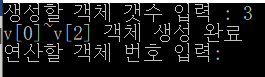


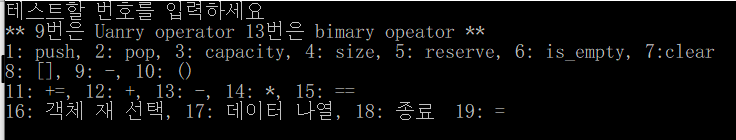
다음과 같이 객체 개수를 입력하면



위와 같은 코드를 통해 원하는 개수만큼 객체를 만들 수 있다.

3개를 생성 후 MyIntVector의 기능들을 살펴보자.





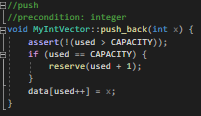
객체 생성 후 테스트코드에서 객체 번호를 선택 후 입력 받는다.

보고서 작성시 위 테스트코드의 16,17,18을 제외한 과정을 순서대로 진행하였다.

**1. Push**

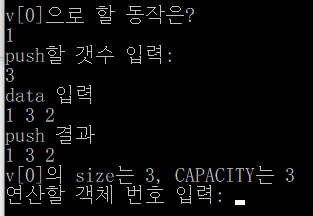
//precondition: 정수 x입력

//postcondtion: x를 data에 push하고 size를 한 칸 늘리고 capacity가 모자란 경우 한 칸 늘린다.



Push를 하면 위 멤버함수에 의해 먼저 size가 capacity보다 큰 버그가 있는지 확인을 한 후 만약 size가 capacity와 같으면 reserve 함수를 통해 capacity를 하나 늘려준다.

그 후 데이터를 저장한다.



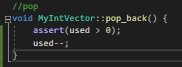
v[0], v[1], v[2] 각각의 객체에서 먼저 v[0]에 (1, 3, 2) 3개를 push하자.

Data에 1,3,2의 값이 저장되고, size는 3, capacity는 3이 된다.

**2. Pop**

//precondition: push되있는, 즉 size가 0보다 큰 객체에서 실행

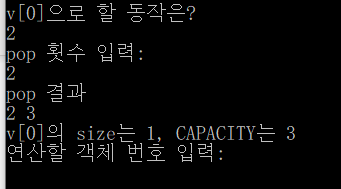
//postcondtion: 데이터를 빼내어 size를 한 칸 줄인다. capacity는 유지된다.

****

Pop을 실행시 size가 하나 줄게된다. 데이터 메모리는 사실상 남아있게 되기 때문에 capacity는 유지된다.

Pop을 2번 실행시 데이터가 2,3 순서대로 빠져나오게 된다.

그리고 데이터는 1, size는1 capacity는 3이 된다.



**3,4. Capacity & size**

Capacity():

//precondition:

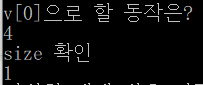
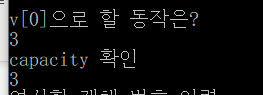
//postcondtion: capacity 리턴

Size():

//precondition:

//postcondtion: size 리턴

Capacity와 size확인시





위 멤버함수에 의해

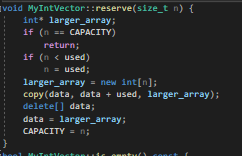
다음과 같이 capacity와 size가 각각 3,1이 리턴된 것을 확인하였다.

**5. reserve**

//precondition: 0이상의 정수 n 입력

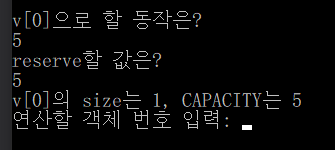
//postcondtion: capacity 조절

Reserve를 통해 capacity를 늘려보고 줄여보자.

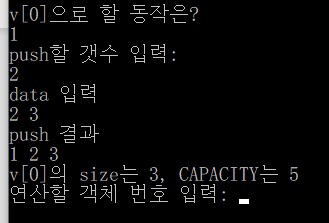


위 코드에 의해 입력받은 값이 size보다 작거나 같게 되면 capacity가 size와 동일하게 되고  
입력 받은 값이 더 크면 capacity가 그에 맞게 조절된다.

이때 동적할당을 통해 새로운 capacity만큼 공간을 만들고 기존 데이터를 복사하는 copy함수를 활용하였다. 그 후 기존 메모리공간을 반환 하였다.

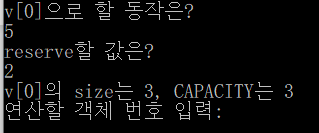


기존 capacity가 3에서 5로 늘어난 것을 확인하였다.



테스트를 위해 2,3을 push하였다. Size=3, capacity=5가 된 것을 확인하였다.

이때 reserve(2)를 하게 되면 입력된 capacity가 size보다 작게되고 이 경우에는 capacity를 현재 size에 맞게 맞춰준다.



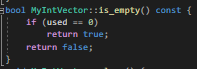
**6. is\_empty**

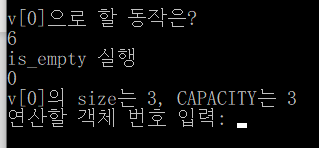
//precondition:

//postcondtion: used가 0인 경우 true, 아닌 경우 false 리턴

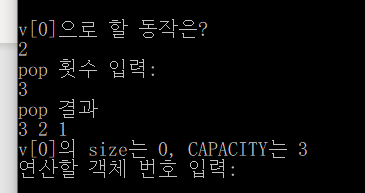
Size가 0일 때 1, 아니면 0을 리턴하는 함수이다.

Capacity는 고려하지 않는다.

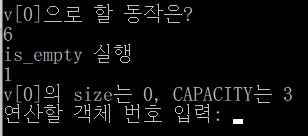




Size가 3이기 때문에 0이 리턴된 것을 확인하였다.



Pop을 3번하여 size를 0으로 맞춘 후 테스트를 진행하였다.

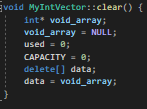


Size가 0인 경우 1이 리턴된 것을 확인하였다.

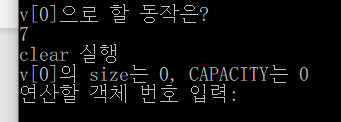
**7. clear**

//precondition:

//postcondtion: data 반환 및 size, capacity 0으로 초기화



초기화를 위해 size와 capacity를 0으로 맞추고 기존 데이터를 동적할당 해제 및 데이터가 없기 때문에 NULL값으로 포인터를 대체하였다.



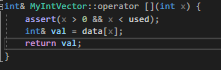
V[0]이 초기화 되었다.

Unary operator

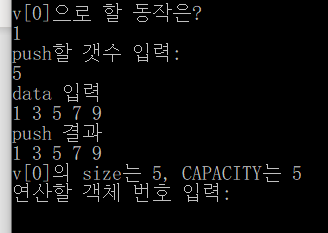
**8. []**

//precondition: MyIntVector 객체

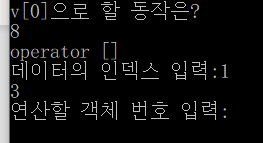
//postcondtion: 입력받은 객체를 deepcopy 후 레퍼런스 return



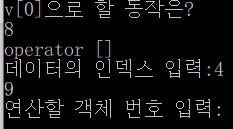
data의 인덱스를 입력 후 레퍼런스가 리턴되는 함수를 구현하였다.



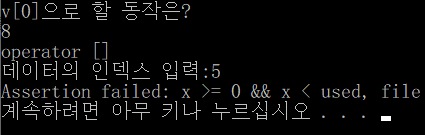
Test를 위해 clear된 v[0]에 1,3,5,7,9를 push하였다.

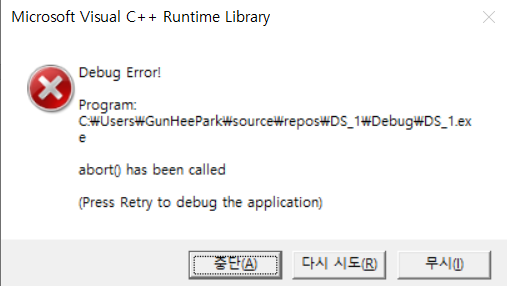


V[0][1]이 실행되어 v[0]의 1번째 값인 3이 리턴되었다.

4

또 테스트를 위해 index가 4인 경우는 9가 리턴된 것을 확인하였다. 이 때 인덱스보다 큰 값을 입력시 강제종료 되도록 함수를 구현하였다.

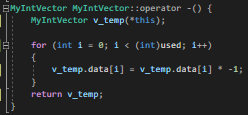




**9. –(Unary)**

//precondition:

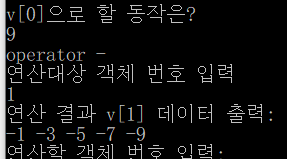
//postcondtion: 데이터의 부호가 반대로 된 MyIntVector 객체 반환



위와 같이 Unary operator이므로 자기자신을 복사한 v\_temp 객체를 생성 후, 데이터를 전부 부호를 바꿔준 후 v\_temp를 리턴하였다.

테스트에 앞서 위 과정에서 코드가 종료되었기 때문에 3개의 객체를 만든 후 v[0]에 1,3,5,7,9의 데이터를 push하고 v[1]=-v[0]를 실행하였다.

실행 결과 v[1]에 제대로 -1,-3,-5,-7,-9의 data가 입력된 것을 확인하였다.

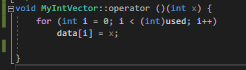


**10. ()**

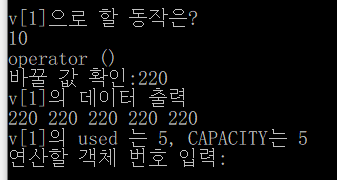
//precondition: 정수 x

//postcondtion: 정수 x로 객체의 모든 데이터로 변환

모든 데이터를 입력 값으로 바꾸는 함수이다.



위와 같은 코드를 통해 정수 x를 입력받으면 모든 저장된 데이터를 x로 변환한다.

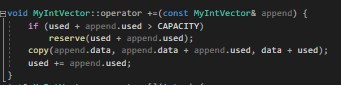


위와 같이 v[1]에서 -1,-3,-5,-7,-9가 저장된 데이터를 220으로 바꿔주었다.

**11. +=**

//precondition: MyIntVector 객체

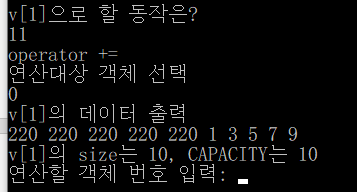
//postcondtion: append의 data만큼을 기존 데이터에 붙여주고 사이즈를 그 만큼 늘려줌, capacity는 모자란 경우 할당



배열을 붙여주는 연산자이다.

Capacity가 확보되지 않은 경우 operand간의 size를 더하여 capacity를 할당해준다.

그 후 copy함수를 통해 기존 데이터의 배열에 대상 배열을 복사한 후, size를 그에 맞게 늘려준다.



V[1]+=v[0]의 실행 결과이다.

V[1]에 1,3,5,7,9의 데이터가 늘어났다.

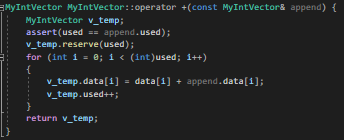
Size와 capacity가 개수에 맞게 조절된 것도 확인하였다.

**Vector 연산**

**12. +**

//precondition: size가 동일한 MyIntVector 객체

//postcondtion: 두 operand 데이터간 벡터 +연산 된 yIntVector 객체 반환

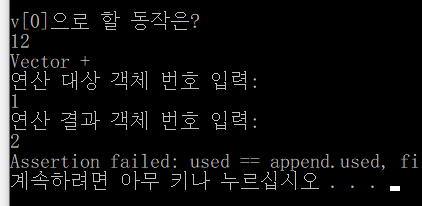


위와 같은 코드를 통해 size가 같은지를 먼저 확인한다.

Size가 다르면 강제종료 되도록 코드를 구현하였다.

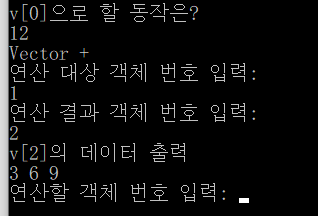
테스트코드상에서는 v[2]=v[0]+v[1]을 실행하였다.

이때 v[0]은 size가 5이고, v[1]은 size가 10이기 때문에 강제종료되었다.



테스트를 위해 size를 동일하게 v[0]와 v[1]에 1,2,3 및 2,4,6를 push 한 후

V[2]=v[0]+v[1]을 실행하였다.

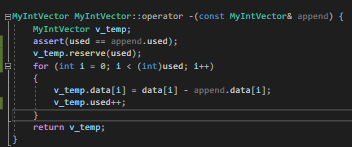


실행 결과 v[2]에 3,6,9로 벡터의 +연산이 된 것을 확인하였다.

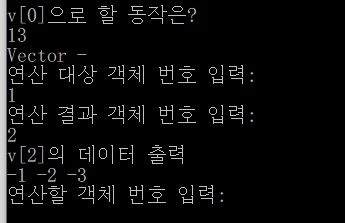
**13. – (Binary)**

//precondition: size가 동일한 MyIntVector 객체

//postcondtion: 두 operand 데이터간 벡터 -연산 된 MyIntVector 객체 반환



-연산 또한 +연산과 부호빼고 전부 같다.

’

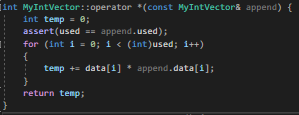
V[2]=v[0]-v[1]을 실행하였다.

(1,2,3)-(2,4,6)이 진행 되어 (-1,-2,-3)이 v[2]에 저장되었다.

**14. \***

//precondition: size가 동일한 MyIntVector 객체

//postcondtion: 두 operand간 data의 벡터 내적 후 결과 값 반환

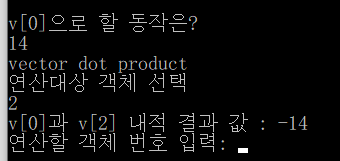


벡터의 내적 연산이다. 위와 같이 size가 같은지 확인한 후에 operand간 곱하고 그 값을 temp에 더하는 과정을 size만큼 반복 후 리턴한다.

V[0]=(1,2,3)이고 v[2]=(-1,-2,-3)이기 때문에 내적을 진행시

1\*-1 + 2\*-2 + 3\*-3 = -14의 결과가 나올 것이다.

테스트코드를 통해 v[0]\*v[2]을 실행시

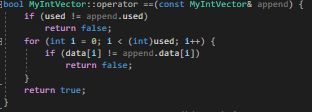


-14가 출력되는 것을 확인하였다.

**15. ==**

//precondition: MyIntVector 객체

//postcondtion: 동일한 데이터가 들어있으면 True 아니면 false 반환, 이 때 capacity는 고려하지 않는다.

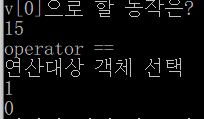


먼저 사이즈 비교 후에 다르면 0을 리턴,

사이즈가 같은 경우 데이터간에 size만큼 반복하여 비교하면서 데이터가 다른 경우 0을 리턴하며 size만큼 비교 후에도 데이터가 전부 같으면 그때 1을 리턴한다.

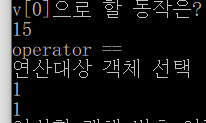
이때 capacity는 코드와 같이 고려하지 않았다.

먼저 v[0]==v[1]을 테스트한 결과 0이 리턴된 것을 확인하였다.



테스트를 위해 v[0]과 v[1]을 모두 연산자()를 이용해서 입력된 데이터를 3으로 바꾼 후

V[0]==v[1]을 실행하였다.



그 결과 v[0]==v[1]이 1이 리턴된 것을 확인하였다.

**16,17,18**

테스트코드에서 16,17,18은

객체 재 선택, 객체에 저장된 데이터 나열, 프로그램 종료이다.

종료시에 중요한 부분이 하나 있다.

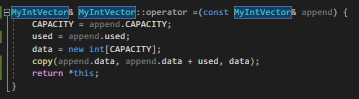


테스트시 사용된 방법이 객체를 동적할당받는 방법이었기 때문에 동적할당된 객체들을 반환하였다.

**19. =**

//precondition: MyIntVector 객체

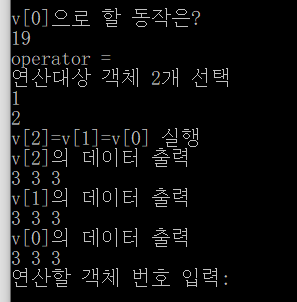
//postcondtion: 입력받은 객체를 deepcopy 후 레퍼런스 return



Append를 deepcopy하여 레퍼런스를 리턴한다.

Chaining assignment까지 확인을 위해

V[2]=v[1]=v[0]을 실행하였다.



실행 결과 v[1]과 v[2]가 모두 v[0]과 동일하게 되었다.

**고찰**

위 테스트들을 통해 실제 Int Vector를 구현 및 벡터 연산들을 추가하여 소스코드를 제작했다.

코딩시 디버깅과 과제의 조건을 충족하느라 시간이 많이 소모되었다. 또한 실제 int vector와 기능을 비교한 결과 많은 부분들이 빠져 있는 것 또한 확인하였다. 추가적인 기능을 구현하였다가 지운 기능들도 있었는데 size가 다른 경우 vector연산과 생성자를 만들 때 default가 아닌 값을 받아서 capacity를 조절하는 경우이다. 이러한 부분들이 과제 조건에 추가되어도 괜찮다고 생각하였다.

테스트 코드에서 객체 자체를 동적할당하는 과정을 진행하였다. 그 후 스위치 케이스 문법을 통해 테스트 번호를 입력받도록 코드를 구현하였다. 테스트코드를 제작하면서 객체의 버그들을 찾아내었다. 또한 테스트코드에도 버그들이 있었고 이 부분들을 수정하는데 꽤 많은 시간이 소요되었다.