Data sturcture HW1 2016726028 이민재

예시출력을 위한 사전 함수들은 다음과 같다.

.capacity, .size

void printVector(string name ,MyIntVector& v) { // 벡터 정보를 출력

cout << endl;

cout << "MyIntVector " << name << endl;

cout << "capacity : " << v.capacity() << endl;

cout << "size : " << v.size() << endl << endl;

}

void printElem(string name, MyIntVector& v) { //벡터 원소들을 출력

cout << name <<"[0] : " << v[0] << endl;

cout << name <<"[1] : " << v[1] << endl;

cout << name << "[2] : " << v[2] << endl << endl;

}

Default Constructor

MyIntVector(size\_t init\_capacity = Default\_Capacity); //default constructor

//precondition: init\_capacity(integer) should be more than 0

//postcondition: init\_capacity - > v\_capacity, if init\_capacity N/A.. Default\_Capacity -> v\_capacity

//set the size of array (data), allocate into HEAP, initialize v\_used = 0; v\_capacity= ..

MyIntVector::MyIntVector(size\_t init\_capacity)

{

data = new int[init\_capacity];

v\_capacity = init\_capacity; //총 용량

v\_used = 0; // 사용량

}

Example )

cout << ">>Test Default constructor" << endl << endl;

cout << "MyIntVector v1;" << endl;

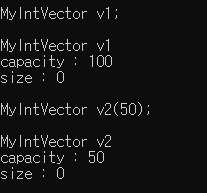
MyIntVector v1;

printVector("v1", v1);

cout << "MyIntVector v2(50);" << endl;

MyIntVector v2(50);

printVector("v2", v2);



기본생성자로 생성한 경우 capacity 100 으로. (50)처럼 사이즈를 지정해준 경우 capacity 50으로 vector 컨테이너가 생성되는 것을 확인할 수 있다.

Push\_Back

void push\_back(int x); // puch back func

//precnd: factor should be integer

//postcnd: if v\_capacity is full , double current capacity using reserve func

//Adds a new element at the end of the vector, after its current last element. The content of this new element is initialized to a copy of x.

void MyIntVector::push\_back(int x)

{

if (v\_used == v\_capacity) // 사용량 한계치 도달하면 capacity를 두배 하여 용량을 확보한다.

reserve(v\_capacity \* 2);

data[v\_used++] = x; //인덱스 1씩늘려가며 뒤에 삽입

}

Example)

cout << ">>push back(1),(2),(3) into v2" << endl << endl;

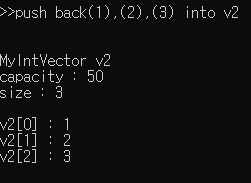
v2.push\_back(1);

v2.push\_back(2);

v2.push\_back(3);

printVector("v2", v2);

printElem("v2", v2);



빈 컨테이너인 v2에 push\_back 함수로 1, 2, 3을 넣었다. 화면 결과처럼 순서대로 원소들이 삽입된것을 확인할 수 있다.

Copy Constructor, Assignment operator (=) for deep copy

MyIntVector(const MyIntVector& v); //Copy constructor for deep copy

//precondition: target Vector should be different with original Vector (copy v1 to v2, not in v1)

//postcondition: make same structure with v(copy of v)

//v1 data(original) - > v2 data (target) , initialize with same value

MyIntVector& operator=(const MyIntVector& v); //Assignment operator (=) for deep copy

//precnd: obj should be MyIntVector Type

//postcnd: Original vector's data turn into target vector's data.

// v1 = v1 should return v1

MyIntVector::MyIntVector(const MyIntVector& v)

{

data = new int[v.v\_capacity]; //복사할 대상의 벡터로 size, capacity 정보를 복사

v\_capacity = v.v\_capacity;

v\_used = v.v\_used;

std::copy(v.data, v.data + v\_used, data); // data를 copy 사용하여 복사

}

Example)

MyIntVector& MyIntVector::operator=(const MyIntVector& v)

{

if (this == &v) return \*this;

if (v\_capacity != v.v\_capacity)

{

delete[] data; //기존 data 할당을 없애고 복사된 vector 정보들로 덮어씌움

data = new int[v.v\_capacity];

v\_capacity = v.v\_capacity;

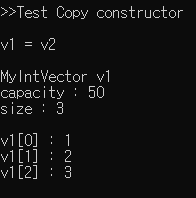
}

v\_used = v.v\_used;

std::copy(v.data, v.data + v\_used, data);

return \*this;

}



cout << ">>Test Copy constructor" << endl << endl;

cout << "v1 = v2" << endl;

v1 = v2;

printVector("v1", v1);

printElem("v1", v1);

= operator 를 이용하여 v1 = v2 의 형식으로 v2의 값을 v1으로복사 하였다. 결과값으로 capcity가 기존 100에서 50으로 element 들이 1, 2, 3으로 v2의 값이 그대로 복사됨을 알 수 있다.

Destructor

~MyIntVector(); //Destructor

//preconditon: None

//postcondition: delete vector data & return HEAP data resource

MyIntVector::~MyIntVector()

{

delete[] data; //할당된 영역을 해제하고 data를 삭제함.

}

Operator +=

void operator+=(const MyIntVector& v); //Operator: += : Appends RHS object to LHS one

//precnd: obj should be MyIntVector Type

//postcnd: put target vector(v2) into original vector(v1)'s back

//capacity of new v1 is 'v\_used of v1 + v\_used of v2' if v\_capcity is smaller than that

void MyIntVector::operator+=(const MyIntVector& v)

{

if (v\_capacity < v\_used + v.v\_used) // 총 용량이 각 사용량 합보다 부족하다면

reserve(v\_used + v.v\_used); // 백터들의 사용량의 합으로 새로운 사용량 산출

std::copy(v.data, v.data + v.v\_used, data + v\_used); //copy 로 대상 컨테이너 뒤에 삽입

v\_used += v.v\_used; //사용량 증가 저장.

}

Example)

cout << ">>Test += Operator" << endl << endl;

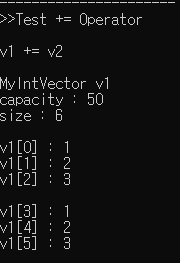
cout << "v1 += v2" << endl;

v1 += v2;

printVector("v1", v1);

printElem("v1", v1);

printElem2("v1", v1);



V1 += v2 로 v1 컨테이너의 사용된 data 뒤에 이어서 v2의 정보를 삽입하는 것을 확인할 수 있다. Size는 기존 3에서 6으로 v1[3] , [4], [5] 각각 1, 2, 3 으로 저장되었다.

* Operator [], void clear( ), bool is\_empty( ) const;

int& operator[](size\_t index); // Operator[] :

//precnd: index should be in '0<= index < v\_used'

//postcnd: return reference of data with index

//If the requested position is out of range, Print some messages(error) and terminate the program.

int& MyIntVector::operator[](size\_t index)

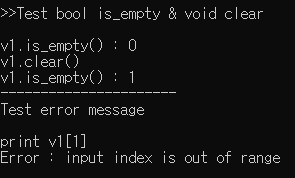
{

if (!(index >= 0 && index < v\_used))

throw std::out\_of\_range("input index is out of range");

return data[index];

}

Example)

cout << ">>Test bool is\_empty & void clear" << endl << endl;

cout << "v1.is\_empty() : " << v1.is\_empty() << endl;

cout << "v1.clear()" << endl;

v1.clear();

cout << "v1.is\_empty() : " << v1.is\_empty() << endl;

cout << "----------------------" << endl;

cout << "Test error message" << endl << endl;

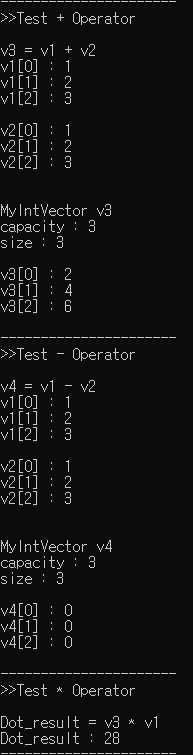
cout << "print v1[1]" << endl;

cout << v1[1] << endl;

[]안의 인덱스를 기반으로 data 레퍼런스를 리턴하는 []오퍼레이터는 범위 밖의 인덱스가 입력되면 에러 메시지를 출력한다. 위 결과는 clear로 내부 내용이 제거된 컨테이너의 [1]값을 참조하다 에러메시지가 출력된 상황이다.

Operator + , - , \*

Example)



cout << ">>Test + Operator" << endl << endl;

MyIntVector v3;

cout << "v3 = v1 + v2" << endl;

v3 = v1 + v2;

printElem("v1", v1);

printElem("v2", v2);

printVector("v3", v3);

printElem("v3", v3);

cout << "----------------------" << endl;

cout << ">>Test - Operator" << endl << endl;

MyIntVector v4(50);

cout << "v4 = v1 - v2" << endl;

v4 = v1-v2;

printElem("v1", v1);

printElem("v2", v2);

printVector("v4", v4);

printElem("v4", v4);

cout << "----------------------" << endl;

cout << ">>Test \* Operator" << endl << endl;

int Dot\_result;

cout << "Dot\_result = v3 \* v1" << endl;

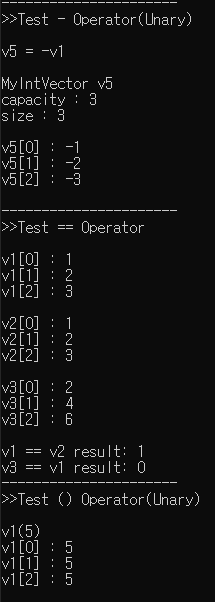
cout << "Dot\_result : " << v3 \* v1 << endl;

cout << "----------------------" << endl;

각 binary operator의 연산 결과를 출력해보았다. V3은 v1과 v2의 합으로 각 2, 4, 6이 출력되었고, v4는 v1 과 v2 의 차로 각 0, 0, 0이 출력되었다 dotproduct 값인 dot\_result 는 v3\*v1을 하여 28이 출력됨을 알 수 있다.

Operator –(Unary) , ==, ()

Example)



cout << ">>Test - Operator(Unary)" << endl << endl;

MyIntVector v5;

cout << "v5 = -v1" << endl;

v5 = -v1;

printVector("v5", v5);

printElem("v5", v5);

cout << "----------------------" << endl;

cout << ">>Test == Operator" << endl << endl;

printElem("v1", v1);

printElem("v2", v2);

printElem("v3", v3);

cout << "v1 == v2 result: " << (v1==v2) << endl;

cout << "v3 == v1 result: " << (v3 == v1) << endl;

cout << "----------------------" << endl;

cout << ">>Test () Operator(Unary)" << endl << endl;

cout << "v1(5)" << endl;

v1(5);

printElem("v1", v1);

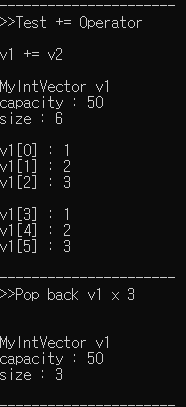
Unary 연산자를 적용한 v5 는 -v1의 값으로 각 원소들이 -n 형태로 저장됨을 확인 할 수있다.

== 연산자의 경우 v1 과 v2, v1 과 v3 의 값을 대조하여 서로 같으면 1을 다르면 0을 리턴하고 이를 결과값으로 확인하였다.

()연산자의 경우 v1(5)로 사용된 모든 data 레퍼런스값을 5로 초기화함을 결과값으로 확인할 수있다.

PoP\_Back

Example)



cout << ">>Test += Operator" << endl << endl;

cout << "v1 += v2" << endl;

v1 += v2;

printVector("v1", v1);

printElem("v1", v1);

printElem2("v1", v1);

cout << "----------------------" << endl;

cout << ">>Pop back v1 x 3" << endl << endl;

v1.pop\_back();

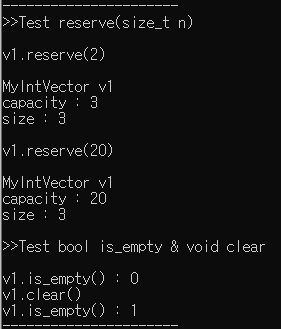
v1.pop\_back();

v1.pop\_back();

printVector("v1", v1);

+= 연산자에서 이어지는 내용으로 += 에 의해 6 까지 늘어난 size 의 v1 컨테이너에서 popback 을 3번 사용하여 뒤의 3, 4, 5번 인덱스의 원소들을 제거하고 다시 정보를 출력하여 size가 3이 되되는 것 결과값으로 확인 할 수있다.

Reserve

reserve() 사용으로 capacity 가 변화하는 것을 확인 가능하다.

printElem("v1", v1);

cout << "----------------------" << endl;

cout << ">>Test reserve(size\_t n)" << endl << endl;

cout << "v1.reserve(2)" << endl;

v1.reserve(2);

printVector("v1", v1);

cout << "v1.reserve(20)" << endl;

v1.reserve(20);

printVector("v1", v1);