**EE 205 Lab 6**

**Vectors and lists**

**Objectives**

The objectives of this lab are to get familiar and gain experience with vectors and lists.

.

**Task 1**

I made two simple file to get familiar with the vectors: one with our own implementation of class vector, and another with standard template library (STL) vector container. Run the files and experiment.

Homemade vector class has structure like this:

however many elements (size)

size

// MyVector.cpp : homemade vector class

//

#include "stdafx.h" //NECESSARY FOR VISUAL STUDIO. On Unix, comment out this line

#include <iostream>

using namespace std;

class vector {

private:

int sz; // the size

double\* elem; // a pointer to the elements

public:

vector(int s) // constructor: allocates/acquires memory

:sz(s), elem(new double[s])

{

}

~vector() // destructor: de-allocates/releases memory

{

delete[] elem;

}

double get(int n) const { return elem[n]; } // access: read

void set(int n, double v) { elem[n] = v; } // access: write

int size() const { return sz; } // the current size

};

int main()

{

vector v(10);

for (int i = 0; i<v.size(); ++i) {

v.set(i, i);

cout << v.get(i) << ' ';

}

return 0;

}

The vector we created looks like this:

1.0

2.0

3.0

4.0

5.0

6.0

7.0

8.0

0.0

9.0

10

// Vectors.cpp : demonstrates use of vector standard template library container

//

#include "stdafx.h" //NECESSARY FOR VISUAL STUDIO. On Unix, comment out this line

#include <iostream>

#include <vector>

using namespace std;

void display(const vector<int> &v);

void displaywithiterator(vector<int> &v);

int main()

{

vector<int> v1; //empty vector of ints

vector<int> v2(3); //3 ints

vector<int> v3(4, 10); //4 ints, of value 10

vector<int> v4(v3); //copy constructor

v1.push\_back(1);

v1.push\_back(2);

v1.push\_back(3);

cout << "v1 should display 1 2 3: \t ";

display(v1);

cout << "v2 should display 3 ints: \t";

display(v2);

v3.pop\_back();

v3.push\_back(5);

cout << "v3 should display 3 ints of 10, then 5: \t ";

display(v3);

cout << "v4 should display v3 before changes: \t";

display(v4);

vector<int>::iterator it;

//iterator is like an index into vector

//find element with value 5, and delete it from vector

it = find(v3.begin(), v3.end(), 5);

if (it != v3.end()) {

v3.erase(it);

}

cout << "deleted 5 in v3: \t";

displaywithiterator(v3);

return 0;

}

void display(const vector<int> &v) {

int i;

for (i = 0; i < v.size(); i++) {

cout << v[i] << " ";

}

cout << "\n";

}

void displaywithiterator(vector<int> &v) {

vector<int>::iterator it;

for (it = v.begin(); it != v.end(); it++) {

cout << \*it << " ";

}

cout << "\n";

}

Demonstrate to the TA that you can make a new STL vector with values 10, 20, 30 and 40 inside it.

Optional: explore multidimensional vectors. There are many ways to go about it, but seems that vector<vector<int>> v is the simplest.

// VectorVector.cpp : how about multidimensional vectors?

//

#include "stdafx.h"

#include <iostream>

#include <string>

#include <vector>

#define N 3

#define M 5

using namespace std;

void displayvector(const vector<int> &v) {

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << "\n";

}

void displayvectorbyp(vector<int> \*v, int size) {

for (int i = 0; i < size; i++)

cout << (\*v)[i] << " ";

cout << "\n";

}

int main()

{

//vector<int><int> vm;

//VECTOR OF VECTORS

cout << "vector of vectors\n";

vector<vector<int>> vm(N, vector<int> (M));

vm[1].push\_back(1);

vm[2].push\_back(2);

for (int i = 0; i < N; i++) {

displayvector(vm[i]);

}

//ARRAY OF VECTORS

//really cheesy. not a good idea

cout << "too cheesy: array of vectors\n";

vector<int> va[N];

for (int i = 0; i < N; i++) {

vector<int> v(M,44);

va[i] = v;

displayvector(va[i]);

}

//ARRAY OF POINTERS TO VECTOR

//not a good idea. I was thinking of strings, but here it is too cumbersome

cout << "too much work: array of pointers to vector\n";

vector<int>\* pvm[N];

for (int i = 0; i < N; i++) {

vector<int> v(M);

pvm[i] = &v;

pvm[i]->push\_back(10);

displayvectorbyp(pvm[i], M + 1);

}

//ARRAY OF POINTERS TO STRING:

//GOOD IDEA WITH STRINGS

cout << "good idea: array of pointers to string\n";

string\* sa [N];

for (int i = 0; i < N; i++) {

string x = "something \n"; //read lines from file, e.g. getline(x)

sa[i] = &x;

cout << \*(sa[i]);

}

cout << "\n";

cout << "good idea: array of pointers to list. Stay tuned\n";

return 0;

}

**Task 2**

Finish the list [slides given in class](https://dl.dropboxusercontent.com/u/1895560/UH-PUBLIC/EE%20205%20C%2B%2B/PPTs/list.ppt). Write code for SinglyLinkedList class and DoublyLinkedList class, with all the functions we did not do. (We designed only insert and erase for doubly linked list).

* insert: insert element n before element p
* add: insert element n after element p
* erase: delete element
* find: search for a given value
* advance: get nth element

class SinglyLinkedList {

public:

insert, delete, find, …

private:

class Link { … } //Since the slides have a list of strings, let us continue with that

Link\* head

Link\* tail; int size; //one or the other is sufficient. size is probably better

}

At the end, you should have these files: singlylinked.cpp and singlylinked.h, doublylinked.cpp and doublylinked.h and driver.cpp. Your driver will very simply test the functions just to see if they work.

Use this code I wrote as a starting point. I put in some stubs, but not all of them.

// Lists.cpp : singly and doubly linked lists

/\*

Test cases for the list :

Empty list

List with one element

List with some elements

Test cases for each element :

Is it the first element, i.e.does it have predecessor ?

Is it the last element, i.e.does it have successor ?

Is it the only element ?

\*/

//

#include "stdafx.h"

#include <iostream>

#include <int>

using namespace std;

struct SLink { //singly linked node

int value;

string name;

double bal;

SLink\* succ;

SLink(const int& v, SLink\* s = 0)

:value(v), succ(s) {}

};

struct DLink { //doubly linked node

int value;

DLink\* pred;

DLink\* succ;

DLink(const int& v, DLink\* p = 0, DLink\* s = 0)

:value(v), pred(p), succ(s) {}

};

class SinglyLinkedList {

public:

SinglyLinkedList() { head = NULL; tail=NULL; }

SLink\* insert(SLink\* n, SLink\* p); //insert n before p, return pointer to n

SLink\* add(SLink\* p, SLink\* n); //insert n after p, return pointer to n

SLink\* erase(SLink\* p); //erase node at p

SLink\* find(SLink\* p, const int& s); //find node with specified content

SLink\* advance(SLink\* p, int n);

void append(SLink \*p); //add a list node at the end

void print\_all(SLink\* p);

SLink\* getHead() { return head; }

SLink\* setHead(SLink \*p) { head = p; }

private:

//class SLink { //… } //could also be struct Link { }

SLink\* head;

SLink\* tail; int size; //one or the other is sufficient

};

class DoublyLinkedList {

public:

DoublyLinkedList(){ head = NULL; tail = NULL; }

DLink\* insert(DLink\* n, DLink\* p);

DLink\* add(DLink\* p, DLink\* n);

DLink\* erase(DLink\* p);

DLink\* find(DLink\* p, const int& s);

DLink\* advance(DLink\* p, int n);

void append(DLink \*p);

void print\_all(DLink\* p);

DLink\* getHead() { return head; }

DLink\* setHead(DLink \*p) { head = p; }

private:

DLink\* head;

DLink\* tail; int size; //one or the other is sufficient

};

//stub

void SinglyLinkedList::append(SLink \*p) {

cout << "append\n";

return;

}

void DoublyLinkedList::append(DLink \*p) {

//insert(p, tail);

if (head == NULL) {

head = p;

p->pred = NULL;

}

if (tail != NULL) {

tail->succ = p;

p->pred = tail;

}

p->succ = NULL;

tail = p;

return;

}

//stub

SLink\* SinglyLinkedList::insert(SLink\* n, SLink \*p) { //insert n before p

cout << "insert \n";

SLink s("something");

return &s;

}

DLink\* DoublyLinkedList::insert(DLink\* n, DLink \*p) { //insert n before p

if (n == 0) return p;

if (p == 0) return n;

n->succ = p;

if (p->pred)

p->pred->succ = n;

n->pred = p->pred;

p->pred = n;

return n;

}

//stub

SLink\* SinglyLinkedList::erase(SLink \*p) {

cout << "Derase \n";

SLink s("something");

return &s;

}

DLink\* DoublyLinkedList::erase(DLink \*p) { //version 3; the version in Stroustrup book

if (p == 0) return 0;

if (p->succ) {

p->succ->pred = p->pred;

if (p == head) head = p->succ;

}

if (p->pred) {

p->pred->succ = p->succ;

if (p == tail) tail = p -> pred;

}

return p->succ;

}

void DoublyLinkedList::print\_all(DLink\* p) {

while (p) {

cout << p->value;

if (p = p->succ) cout << ", ";

}

cout << "\n";

}

void SinglyLinkedList::print\_all(SLink\* p) {

while (p) {

cout << p->value;

if (p = p->succ) cout << ", ";

}

cout << "\n";

}

int main()

{

SinglyLinkedList s;

DoublyLinkedList d;

DLink d1("d1");

DLink d2("d2");

DLink d3("d3");

SLink s1("s1");

SLink s2("s2");

SLink s3("s3");

d.append(&d1);

d.append(&d2);

d.append(&d3);

d.print\_all(d.getHead());

d.erase(d.getHead());

d.print\_all(d.getHead());

//DLink \*p = d.find(d.getHead(), "d2");

//DLink \*p2 = d.insert(p, &d1);

return 0;

}

**Task 3**

Continue working with Task 1 and create a dictionary. Ask the user to enter an int and what they want to do with it.

1 means: insert into the list in ascending order (this is just a find: Find the element larger than the number and then insert before that element).

2 means: delete this element from the list (so you will first have to find it). Think about the search algorithm.

Test singly linked list in driver dictionary.cpp.

# Grading

* Put all your files into EE205/Labs/Lab6 directory.
* Copy and paste your codes into Laulima submit window or however your TA wants you to submit. Attach the tar file too.

Demonstrate to the TA that you have completed the code, for 2 programs you wrote. Total number of points is 70 (task 2 is 40 pts). You must finish the lab and show it to the TA ideally by the end of this lab and at most by the end of the lab 7 .