**C++ Part II (INFO1-CE9265) Spring 2015 – Final**

Clement Chan

**Question 1 – (35 pts)**

**Create a Singly Linked List that can create at runtime the following scenario**

**Linked\_List\_Implement.cpp**

#include<iostream>

#include<cstring>

class List;

//Implementing the class node

class Node{

private:

std::string data;

Node \*next;

public:

friend class List;

Node():data("No Fruit Yet"),next(0){}; //Default constructor

Node(std::string the\_data): data(the\_data), next(0){}; //Parameterized constructor

};

//Implementing the class List

class List{

private:

int N;

Node \*head;

public:

List() : head(NULL){}; // Constructor

~List(){ //Destructor

std::cout<<"Initiating List Destructor ... " <<std::endl;

Node \*q = head;

int count = 0;

while(q -> next != NULL){

Node \*temp = q;

q = q -> next;

delete temp;

count ++;

std::cout<<"Deleted " << count << " Node" << std::endl;

}

if(q -> next == NULL){

delete q;

count ++;

std::cout<<"Deleted " << count << " Node" << std::endl;

}

};

//Initialize list and write down some fruits name

void Define\_node(int number){

int i = 0;

N = number;

while(i<N){

Node \*temp = new Node();

temp -> next = head;

head = temp;

i++;

}

}

void write\_data(std::string thefruit, int pos){

int i=0;

Node \*p = head;

while(i < pos){

p = p -> next;

i++;

}

p -> data = thefruit;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Inserting and erasing\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Insert From Front

void insert\_front(std::string fruit){

if(head == NULL){

Node \*temp = new Node(fruit);

head = temp;

}

else{

Node \*temp = new Node(fruit);

temp -> next = head;

head = temp;

}

}

//Insert From Back

void insert\_back(std::string fruit){

Node \*q = head;

while(q -> next != NULL){

q = q-> next; // Move towards the last node and point towards next

}

Node \*temp = new Node(fruit);

q -> next = temp;

}

//Insert at location x

void insert\_loc(std::string fruit, int pos){

Node \*q = head;

int count = 0;

char flag = 'g';

while(q -> next != NULL && count < pos){

q = q-> next; // Move towards the last node and point towards next

count++;

}

if(q->next == NULL && count < pos){

throw flag ;

}

else{

Node \*temp = new Node(fruit);

temp -> next = q -> next;

q -> next = temp;

}

}

//Erase the Node

void remove\_loc(int pos){

Node \*q = head;

int count = 0;

char flag = 'g';

while(q -> next != NULL && count < pos){

q = q-> next; // Move towards the last node and point towards next

count++;

}

if(q->next == NULL && count < pos){

throw flag ;

}

else{

Node \*temp = new Node(q->next->data);

temp ->next = q -> next -> next;

q -> next = temp -> next;

delete temp;

}

}

//Search Function

int search(std::string fruit){

Node \*q = head;

int count = 1;

char flag = 'g';

while(q -> next != NULL){

if(q->data != fruit){

q = q-> next;

count ++;

}

else if (q -> data == fruit){

return count;

}

}

if(q->next == NULL){

throw flag;

}

}

//Displaying List

void DisplayList(){

std::cout<< "The List of Fruit is -- " << std::endl;

Node \*q = head;

std::cout<< q->data << std::endl;

while(q -> next != NULL){

q = q -> next;

std::cout<<q->data<<std::endl;

}

std::cout<< " " << std::endl;

}

};

int main(){

List List\_fruit;

List\_fruit.Define\_node(5);

List\_fruit.write\_data("Pineapple",0);

List\_fruit.write\_data("Blueberry",1);

List\_fruit.write\_data("Pinkberry",2);

List\_fruit.write\_data("Laichi",3);

List\_fruit.write\_data("Sukini",4);

//Displaying initial list

List\_fruit.DisplayList();

List\_fruit.insert\_front("apple");

List\_fruit.insert\_front("banana");

List\_fruit.insert\_front("grapes");

List\_fruit.insert\_back("pears");

List\_fruit.insert\_back("mango");

List\_fruit.insert\_back("watermelon");

//Inserting between node location

try{

List\_fruit.insert\_loc("strawberry",3);

}

catch(char a){

std::cout << "Position of entry exceeds List Size" << std::endl;

std::cout << " " << std::endl;

}

//Displaying extended list

List\_fruit.DisplayList();

//Inserting a position that is not possible to test try and catch

try{

List\_fruit.insert\_loc("Orange",10);

}

catch(char a){

std::cout << "Position of entry exceeds List Size" << std::endl;

std::cout << " " << std::endl;

}

//Use the search function defined

try{

std::cout<<"Searching for Apple ... " << std::endl;

std::cout<<"The position for apple is: " << List\_fruit.search("apple") <<

std::endl;

std::cout << " " << std::endl;

}

catch(char a){

std::cout << "No such fruit -- try again" << std::endl;

std::cout << " " << std::endl;

}

//try a position that is impossible

try{

std::cout<<"Searching for Kiwi ... " << std::endl;

std::cout<<List\_fruit.search("Kiwi")<<std::endl;

std::cout << " " << std::endl;

}

catch(char a){

std::cout << "No such fruit -- try again" << std::endl;

std::cout << " " << std::endl;

}

//Erasing between node location

try{

List\_fruit.remove\_loc(2);

}

catch(char a){

std::cout << "Position of entry exceeds List Size" << std::endl;

std::cout << " " << std::endl;

}

//Displaying Erased list

List\_fruit.DisplayList();

return 0;

}

**Output**



**Question 2 – (35 pts)**

**(35 pts) Write a C++ program that creates a text editor using the STL list container.**

#include <iostream>

#include <fstream>

#include <list>

#include <cstring>

using namespace std;

void print\_choices(){

cout<<" " <<endl;

cout<< "Please select the following keys for editing ... " <<endl;

cout<< " 'I' for Insert. 'A' For Append. 'M' to move line. 'D' to Delete. 'F' to find lines with specific word. " <<endl;

}

//Print function that will be used

void print\_list(list<string> &the\_list){

cout<<" " <<endl;

cout<<"Here are the lines that are inputted into the system : " << endl;

int linecount = 1;

for(list<string>::iterator iter=the\_list.begin(); iter != the\_list.end(); iter ++){

cout<< \*iter << "----- line " << linecount << endl;

linecount++;

}

cout<<" " <<endl;

cout<<" " <<endl;

}

//Print function that will be used

void print\_list\_ga(list<string> &the\_gettysburg\_address){

int linecount = 1;

string temp1;

cout<< " " <<endl;

cout<< "Here are the lines that contains 'dead' word in the Gettysburg Address : " << endl;

cout<< " " <<endl;

list<string>::iterator iter;

for(iter = the\_gettysburg\_address.begin(); iter != the\_gettysburg\_address.end(); iter++){

temp1 = \*iter;

if(temp1.find("dead") != string::npos){

cout << "Line " << linecount << " " << \*iter << endl;

cout<< " " <<endl;

}

linecount++;

}

}

//Insert function to be used for the list

void insert\_list(list<string> &the\_list){

string temp;

int position;

cout<<"Enter the position you want to insert the new text line : "<<endl;

cin >> position;

cout<<"Enter textline that you want to insert at the position : "<<endl;

cin >> temp;

getline(cin,temp);

list<string>::iterator iter1 = the\_list.begin();

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

++iter1;

}

the\_list.insert(iter1, temp); //insert into the iterated position

}

//Append function to be used for the list

void append\_list(list<string> &the\_list){

string temp2;

cout<<" " <<endl;

cout<<"Please enter the sentence that you want to append to the last of the list. " <<endl;

cin >> temp2;

getline(cin, temp2);

list<string>::iterator iter;

for(list<string>::iterator iter2 = the\_list.begin(); iter2 != the\_list.end(); ++iter2){

iter = iter2;

}

the\_list.insert(iter, temp2);

}

//Move function to be used for the list

void Move\_list(list<string> &the\_list){

int position;

int final\_position;

cout<<" " <<endl;

cout<<"Enter the sentence's position that you want move from: " <<endl;

cin >> position;

cout<<"Enter the final position that you want to move to: " <<endl;

cin >>final\_position;

list<string>::iterator iter = the\_list.begin();

list<string>::iterator iter2 = the\_list.begin();

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

iter++;

}

for(int i=0; i<final\_position-1; i++){

iter2++;

}

if(position > the\_list.size() || final\_position > the\_list.size()){

throw 'c';

}else{

the\_list.insert(iter2, \*iter);

the\_list.erase(iter);

}

}

//Delete the line at a specific position

void Delete\_list(list<string> &the\_list){

int position;

int final\_position;

cout<<" " <<endl;

cout<<"Enter the sentence's position that you want delete from: " <<endl;

cin >> position;

list<string>::iterator iter = the\_list.begin();

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

iter++;

}

if(position > the\_list.size()){

throw 'c';

}else{

the\_list.erase(iter);

}

}

//Find lines containing target word dead

void Find\_list(list<string> &the\_gettysburg\_address){

string temp, temp1, term;

int linecount = 1;

ifstream address("Gettysburg\_Address.txt");

while(address.good()){

getline(address, temp);

the\_gettysburg\_address.push\_back(temp);

}

address.close();

list<string>::iterator iter;

print\_list\_ga(the\_gettysburg\_address);

}

//Define compare case so that the sort does not have Case differences

bool compare(const std::string& first, const std::string& second)

{

unsigned int i=0;

while ( (i<first.length()) && (i<second.length()) )

{

if (tolower(first[i])<tolower(second[i])) return true;

else if (tolower(first[i])>tolower(second[i])) return false;

++i;

}

return ( first.length() < second.length() );

}

int main(){

list<string> txteditor;

list<string> gettysburg\_address;

string temp, end("#");

char ans, ans1;

cout<<" " <<endl;

cout << "Please enter the lines you want to input in : " << endl;

while(txteditor.size() < 40 && temp.find(end)){

getline(cin,temp);

txteditor.push\_back(temp);

}

/\*Start performing edits\*/

do{

print\_list(txteditor);

print\_choices();

cin >> ans;

switch (ans)

{

{case 'I':

insert\_list(txteditor);

break;

}

{case 'A':

append\_list(txteditor);

break;

}

{case 'M':

try{

Move\_list(txteditor);

}

catch(char c){

cout << "The position exceeds list length.. " <<endl;

}

break;

}

{case 'D':

try{

Delete\_list(txteditor);

}

catch(char c){

cout << "The position exceeds list length.. " <<endl;

}

break;

}

{case 'F':

Find\_list(gettysburg\_address);

break;

}

}

cout<<"Quit ? (Press Q to quit, Any Key to continue)" << endl;

cin >> ans1;

}while(ans1 != 'Q' || ans1 != 'q');

//Print out the list of final edited text

print\_list(txteditor);

print\_list\_ga(gettysburg\_address);

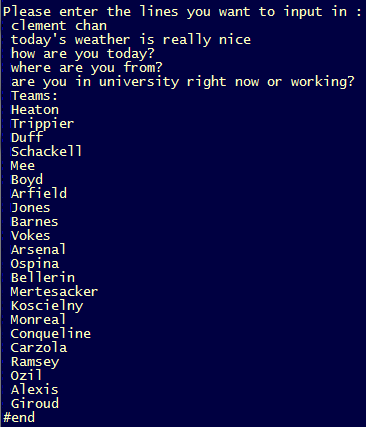
txteditor.sort(compare);

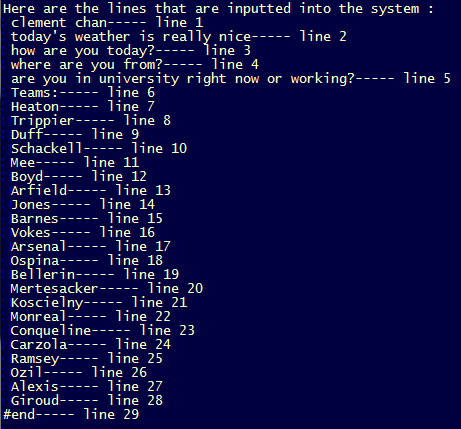
print\_list(txteditor);

return 0;

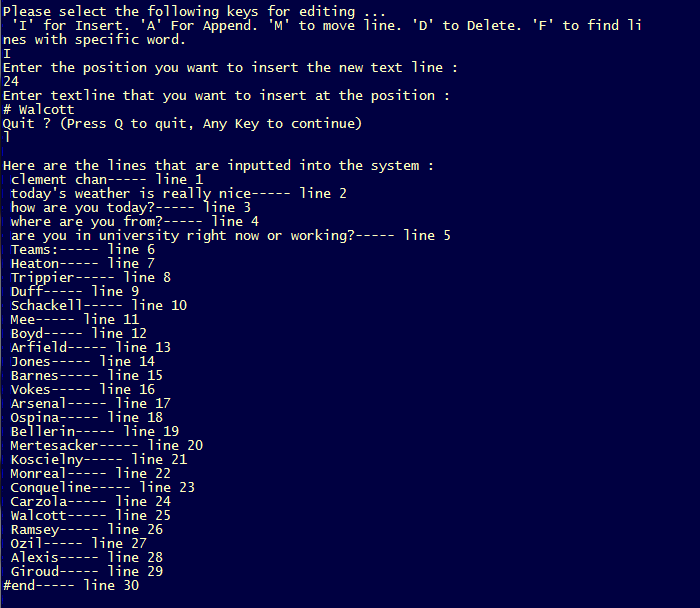
}

**Output**

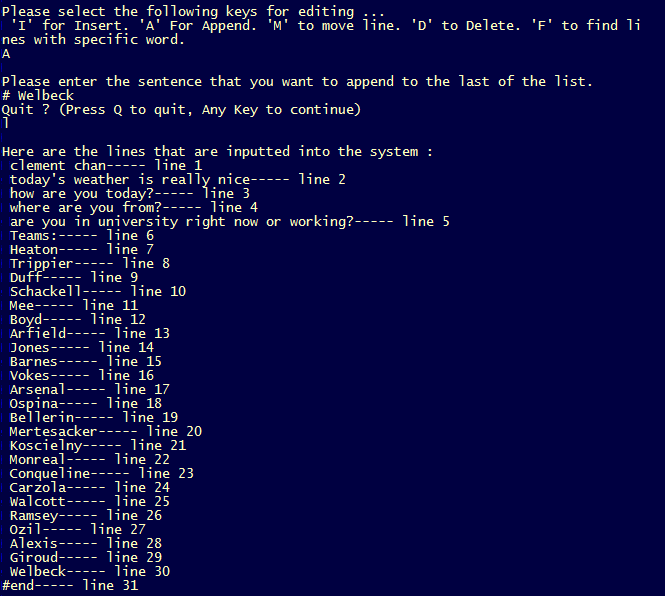
****

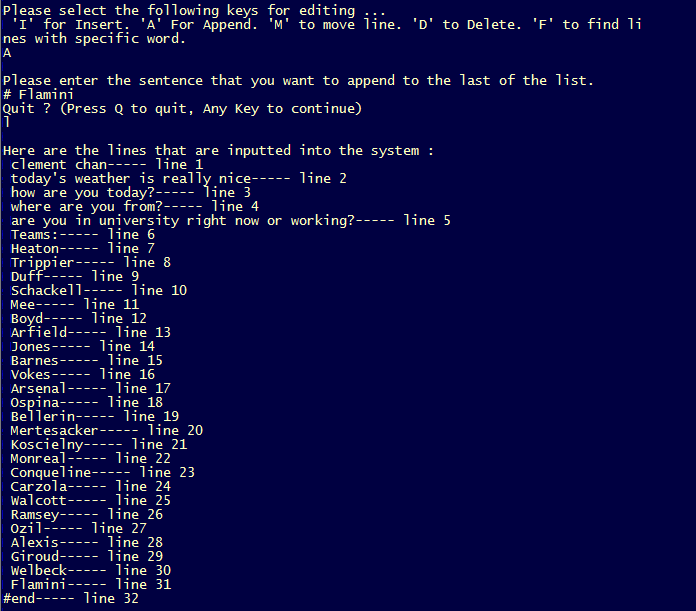
****

**Testing “Insert” function—**

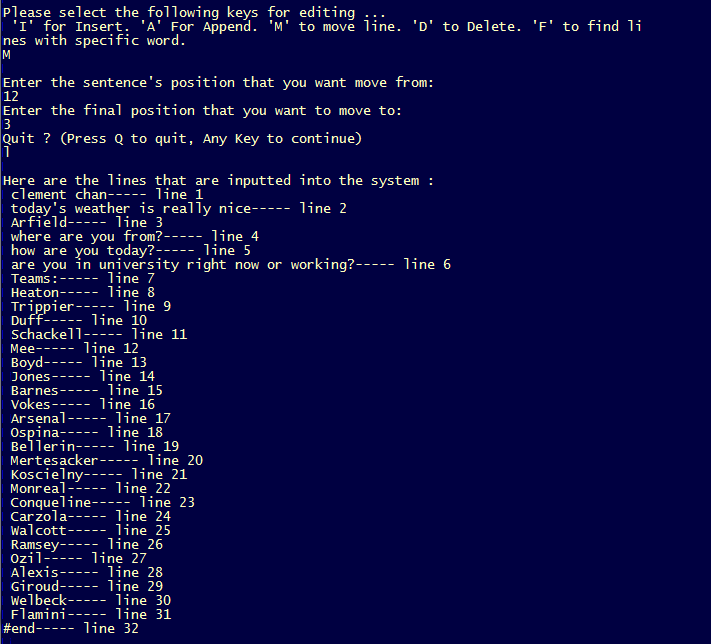
****

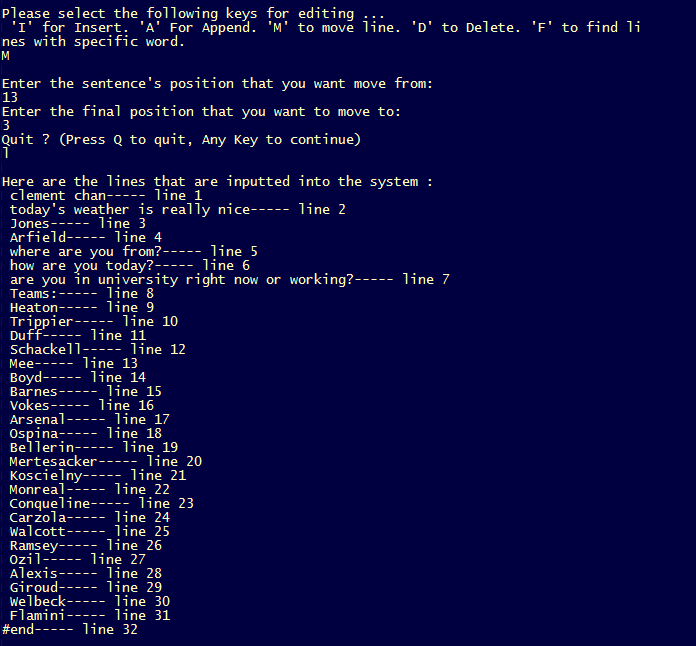
**Testing “Append” function—**

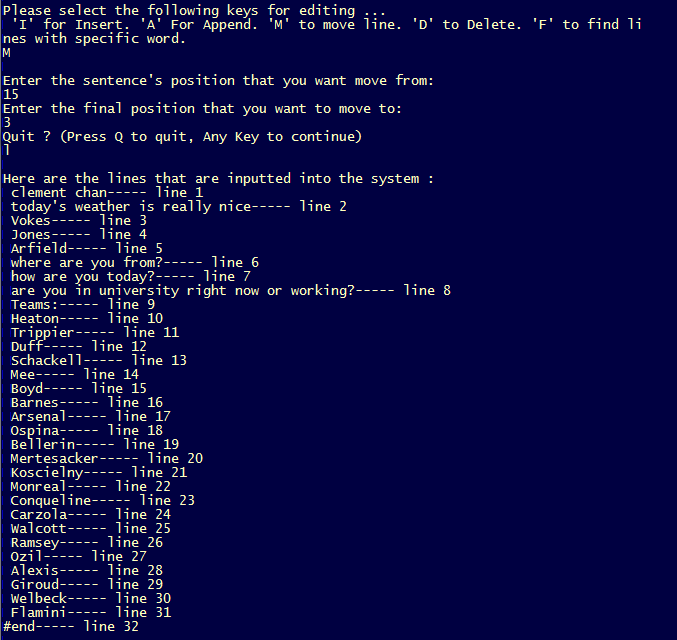
****

****

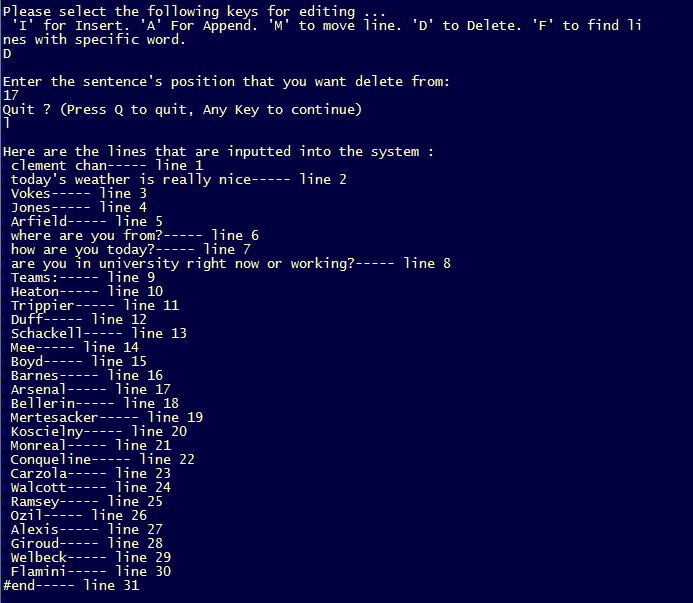
**Testing “Move” function—**

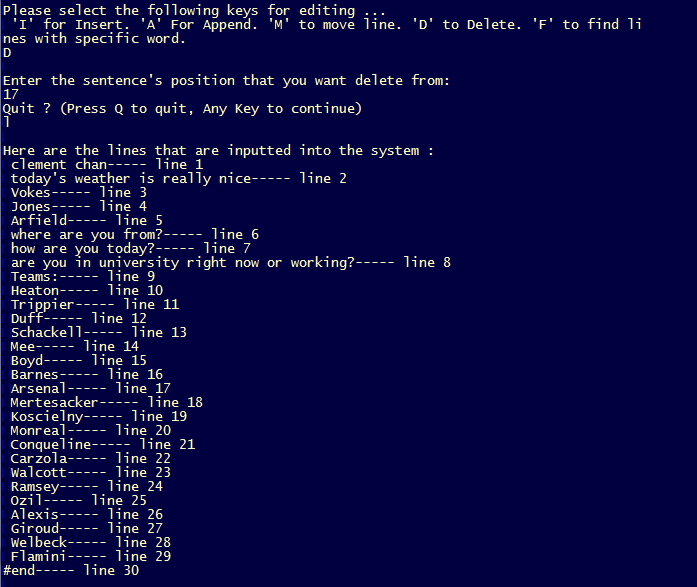
****

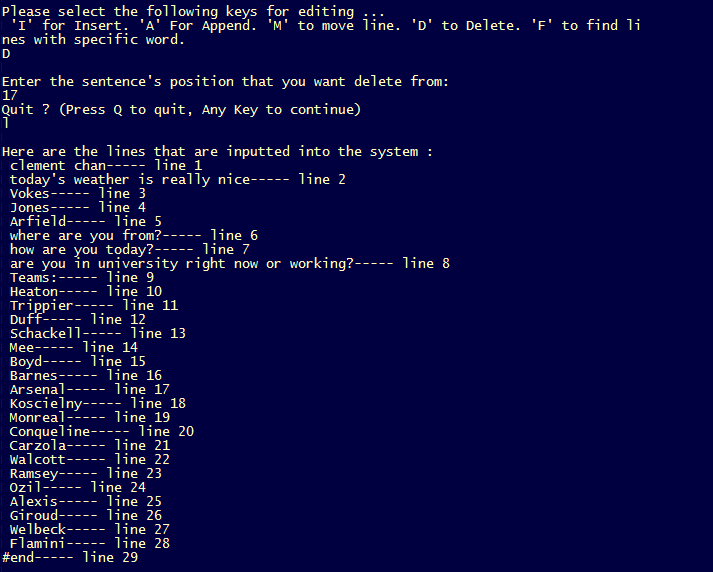
****

****

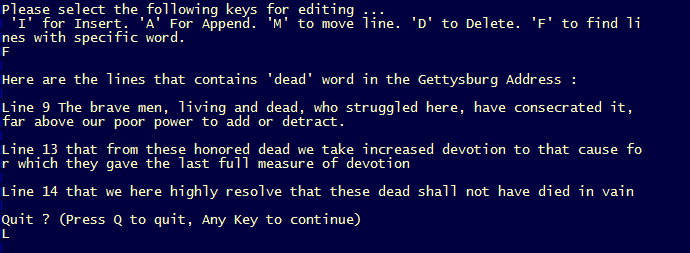
**Testing “Delete” function—**

****

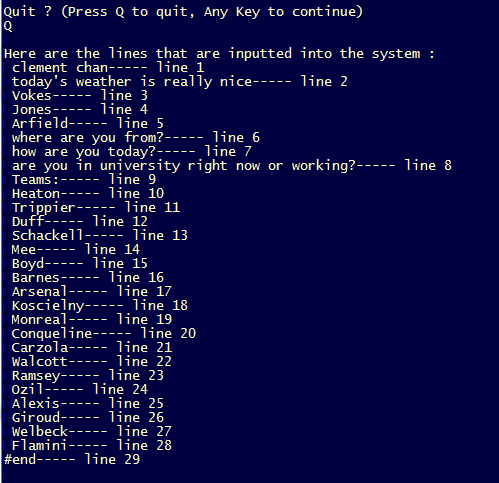
****

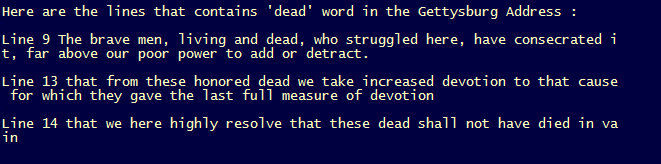
****

**Testing “Find” function—**

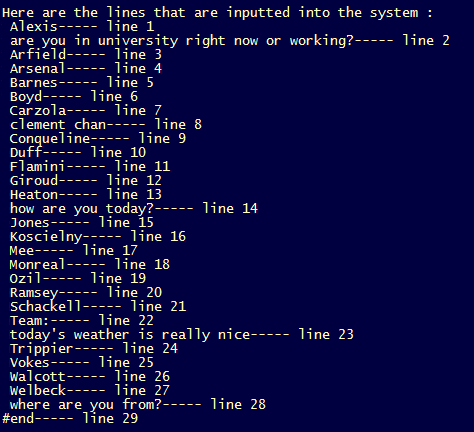
****

**Final Output and Sorted Result**

****

****

**Sorted Result**

****

**Question 3 – (30 pts)**

**Write a C++ program that multiplies two integer matrices, A and B, and outputs result onto a matrix C.**

**Matrix\_Mult.cpp**

#include <iostream>

#include <vector>

using namespace std;

//Improved method using class to do the multiplication

class Matrix{

private:

int row;

int column;

int \*\*Mat;

public:

//Default Constructor

Matrix() : row(0), column(0), Mat(0){}

//Parameterized Constructor

Matrix(int the\_row, int the\_column) : row(the\_row), column(the\_column){

cout<<"Initializaing Constructor ... " <<endl;

Mat = new int\*[row];

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

Mat[i] = new int[column];

}

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

for(int j=0; j<column; j++){

Mat[i][j] = 0;

}

}

}

//Parameterized Constructor with Matrix passing by

Matrix(int \*\*the\_Mat, int the\_row, int the\_column) : row(the\_row), column(the\_column){

Mat = new int\*[row];

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

Mat[i] = new int[column];

}

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

for(int j=0; j<column; j++){

Mat[i][j] = 0;

Mat[i][j] = the\_Mat[i][j];

}

}

}

//Destructor

~Matrix(){

cout << "Initializing Destructor ... " << endl;

delete[] Mat;

};

//Assignment operator

Matrix &operator=(const Matrix &Mt){

delete[] Mat;

Mat = new int\*[row];

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

Mat[i] = new int[column];

}

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

for(int j=0; j<column; j++){

Mat[i][j] = 0;

Mat[i][j] = Mt.getMat(i,j);

}

}

}

//Setters Getters and Operator

int getMat(int row, int column)const{

return Mat[row][column];

}

//Overloading \* to multiply the vectors together (combining all the functions)

friend Matrix operator \*(const vector<vector<int> > &the\_A, const vector<vector<int> > &the\_B){

if(the\_A[1].size() == the\_B.size()){

int \*\*M; //Define the new matrix

M = new int\*[the\_A.size()];

for(int i=0; i<the\_A.size(); i++){

M[i] = new int[the\_B[1].size()];

}

for(int i=0; i<the\_A.size(); i++){

for(int j=0;j<the\_B[1].size(); j++){

M[i][j] = 0;

}

}

for(int i=0; i<the\_A.size(); i++){

for(int j=0; j < the\_B[1].size(); j++){

int C\_mult = 0;

for(int k = 0; k < the\_A[1].size(); k++){

C\_mult += the\_A[i][k] \* the\_B[k][j];

}

M[i][j] = C\_mult;

}

}

return Matrix(M,the\_A.size(), the\_B[1].size());

}

else{

cout << "The dimension is not compatible ... " << endl;

exit(1);

}

}

};

//Building functions without using the class

bool Check\_Dimensions(vector<vector<int> > the\_A, vector<vector<int> > the\_B){

if(the\_A[1].size() == the\_B.size()){

return true;

}

else{

cout << "The dimension is not compatible ... " << endl;

return false;

}

}

int Multiply(vector<vector<int> > the\_A, vector<vector<int> > the\_B, int A\_row, int B\_col, int A\_col\_B\_row){

int C = 0;

for(int j = 0; j < A\_col\_B\_row; j++){

C += the\_A[A\_row][j] \* the\_B[j][B\_col];

}

return C;

}

int main(){

//Define the dimension of rows and columns

int row = 5;

int column = 4;

vector<vector<int> > A(row, vector<int>(column,0)); // 5-by-4 matrix

vector<vector<int> > B(column, vector<int>(row,0)); // 4-by-5 matrix

vector<vector<int> > C(row, vector<int>(row,0)); // 5-by-5 matrix

//Initializing A

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

for(int j=0; j <column; j++){

A[i][j] = 0;

A[i][j] = i + 1;

}

}

//Initializing B

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

for(int j=0; j <row; j++){

B[i][j] = 0;

B[i][j] = j + 1;

}

}

int k = Check\_Dimensions(A,B);

if(k == 1){

//Initializing C

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

for(int j=0; j <row; j++){

C[i][j] = 0;

C[i][j] = Multiply(A,B,i,j,column);

}

}

cout<<" " << endl;

cout<<"Matrix A: " << endl;

//Printing out A Matrix

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

for(int j=0; j<column; j++){

if(j != column-1)

cout << A[i][j] << " ";

else

cout << A[i][j] << endl;

}

}

cout<<" " <<endl;

cout<<"Matrix B: " << endl;

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

for(int j=0; j<row; j++){

if(j != row-1)

cout << B[i][j] << " ";

else

cout << B[i][j] << endl;

}

}

cout<<" " <<endl;

cout<<"Matrix C: " << endl;

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

for(int j=0; j<row; j++){

if(j != row-1)

cout << C[i][j] << " ";

else

cout << C[i][j] << endl;

}

}

}

else{

//If not compatible, then exit program.

exit(1);

}

cout<<" " <<endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Improved Method with operator \* overloaded \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Improved method using classes and operator overloading

Matrix CMat(row,row);

CMat = A \* B;

cout<<" " <<endl;

cout<<"Matrix C using assignment operator (A\*B): " << endl;

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

for(int j=0; j<row; j++){

if(j != row-1)

cout << CMat.getMat(i,j) << " ";

else

cout << CMat.getMat(i,j) << endl;

}

}

Matrix CMat1(column,column);

CMat1 = B \* A;

cout<<" " <<endl;

cout<<"Matrix C using assignment operator (B\*A): " << endl;

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

for(int j=0; j<column; j++){

if(j != column-1)

cout << CMat1.getMat(i,j) << " ";

else

cout << CMat1.getMat(i,j) << endl;

}

}

cout<<" " <<endl;

cout<<"Matrix D with incompatible size: " << endl;

//Testing a non working example

Matrix DMat(row,row);

DMat = A \* A;

return 0;

}

**Output**

