#include <iostream>

#include <sstream>

#include <vector>

#include <string>

#include "Q.h"

#include <boost/tokenizer.hpp>

using namespace boost;

using namespace std;

void menu(){

cout << "\n--- Lab\_11C Queue Test Menu ---\n"

<< " N - to bulk create New Queue\n"

<< " D - to Display\n"

<< " E - is Empty?\n"

<< " A - to Add ~ push Front \n"

<< " a - to append ~ push Back\n"

<< " R - to Remove ~ pop Front\n"

<< " r - to remove ~ Pop Back\n"

<< " G - to Get by subscription\n"

<< " P - to Put by subscription\n"

<< " L - Length of Queue?\n"

<< " C - to Clear\n"

<< " Q - to Q this program\n"

<< " H - this menu\n";

}

int main(){

bool stay = true;

cout << "\nInstanciate an object of Queue\n";

Q<string> strQ;

string choice;

string str;

string sArr[] = {"apple","banna","cherry","durian","mango"};

vector<string> inputS(sArr,end(sArr));

for(auto item:inputS) strQ.push\_front( item );

menu();

while(stay){

cout << "\n Enter your command: ";

stay = true;

cin >> choice;

cin.ignore();

int pos;

string input;

string token;

int item;

stringstream sst;

stringstream ss;

int Queue\_size;

if(choice.size() == 1){

char ch = choice[0];

vector<int>dump;

string value;

switch(ch){

case 'D':

case 'd':{

cout <<" Queue: " << strQ.toString();

break;

}

case 'r':{

string temp;

if( 0 < strQ.getSize()) {

strQ.pop\_back(temp);

cout << "\n removed " << temp <<"\n";

}

else cout << " queue is empty already";

cout << strQ.toString();

break;

}

case 'R':{

string temp;

if( 0 < strQ.getSize()) {

strQ.pop\_front(temp);

cout << "\n removed " << temp <<"\n";

}

else cout << " queue is empty already";

cout << strQ.toString() << "\n";

break;

}

case 'A':{

string n;

cout << "\n Enter a phrase: ";

cin >> n;

strQ.push\_front(n);

cout << "You have entered: "<<n;

cout << "\n Queue: "<< strQ.toString() << "\n";

break;

}

case 'a':{

string n;

cout << "\n Enter a phrase: ";

cin >> n;

strQ.push\_back(n);

cout << "You have entered: "<<n;

cout << "\n Queue: "<< strQ.toString() <<"\n";

break;

}

case 'l':

case 'L':{

cout << "\n Number of Enteries in Queue: "<<strQ.getSize() << endl;

break;

}

case 'e':

case 'E':{

if(!strQ.isEmpty()) {

cout << "Queue is not empty. \n";

}

else{ cout << "Queue is empty. \n"; }

break;

}

case 'c':

case 'C':{

cout << " Queue is cleared. ";

strQ.clear();

break;

}

case 'n':

case 'N':{

if(!strQ.isEmpty()) strQ.clear();

string str;

cout << "Enter a line of comma (,) delmited data set: ";

getline (cin, str); // user input -> string

ss << str; // string -> stream

char\_separator<char> sep(",");

tokenizer<char\_separator<char>> tokens(str, sep);

for (const auto& t : tokens) {

strQ.push\_front(t);

}

break;

}

case 'q':

case 'Q':{

stay = false;

break;

}

case 'g':

case 'G':{

int i;

cout << "\n Enter Item index: ";

cin >> i;

cout << "geting q["<<i<<"] ... ";

if(i>strQ.getSize()){

cout << "\n Exception" << i << " invalid subscription.";

break;

}

else

{

cout << " contains : " <<strQ[i-1];

break;

}

}

case'p':

case'P':{

int i;

string p;

cout << "\n Enter Item index: ";

cin >> i ;

cout << "\n Enter a phrase: ";

cin >> p ;

cout << "\n You have entered: " << p;

if(i>strQ.getSize() || i < 1){

cout << "\n Exception: " << i << " is out of range!";

}else{

strQ[i-1] = p ;

}

break;

}

}

}

}

}

#ifndef Q\_H

#define Q\_H

#include <iostream>

#include <string>

template <class T>

class Q {

private:

struct Node {

T data;

Node \*next;

};

Node \*front, \*rear, \*p, \*pp; // pp is previous p

int size;

public:

class OutOfRangeSubscription {

private:

int value;

public:

OutOfRangeSubscription(int sub) {value = sub;}

int getValue() const { return value; }

};

Q() : front(nullptr), rear(nullptr), size(0) {};

~Q() { clear(); };

bool isEmpty() {

if(front == nullptr) return true;

return false;

};

int getSize() const { return size; };

void clear() {

p = front;

while(p) {

delete(p);

p = p->next;

size--;

}

size =0;

front=rear=pp=p=nullptr;

};

std::string toString() {

std::string s = "";

p = front;

while(p) {

std::stringstream ss;

ss << p->data;

s += ss.str();

p = p->next;

s += (p)?", ":"";

}

return s;

};

void push\_front(T item) {

p = new Node;

p->next = nullptr;

p->data = item;

if(!front) {

front = rear = p;

} else {

p->next = front;

front = p;

}

size++;

};

void push\_back(T item) {

p = new Node;

p->next = nullptr;

p->data = item;

if(!front) // empty Q

front = rear = p;

else {

rear->next = p;

rear = p;

}

size++;

};

bool pop\_front( T &item ) {

if( size <= 0 ) return false;

else {

item = front->data;

delete(front);

if( front==rear )

front = rear = pp = p = nullptr;

else

front = front->next;

}

size--;

}

bool pop\_back( T &item ) {

if( size <= 0 ) return false;

else if(front != rear) {

p = front;

while(p->next != rear) { // this item is not last one!

p = p->next;

}

item = rear->data;

size--;

delete(rear);

rear = p;

rear->next = nullptr;

} else {

item = rear->data;

delete(rear);

front = rear = pp = p = nullptr;

size = 0;

}

return true;

};

T& operator[](const int &sub){

if( 0<= sub && sub < getSize() ) {

p = front;

int count = 0;

while(p) {

if(count == sub)

return p->data;

p = p->next;

count++;

}

}

else

throw OutOfRangeSubscription(sub);

};

};

#endif

