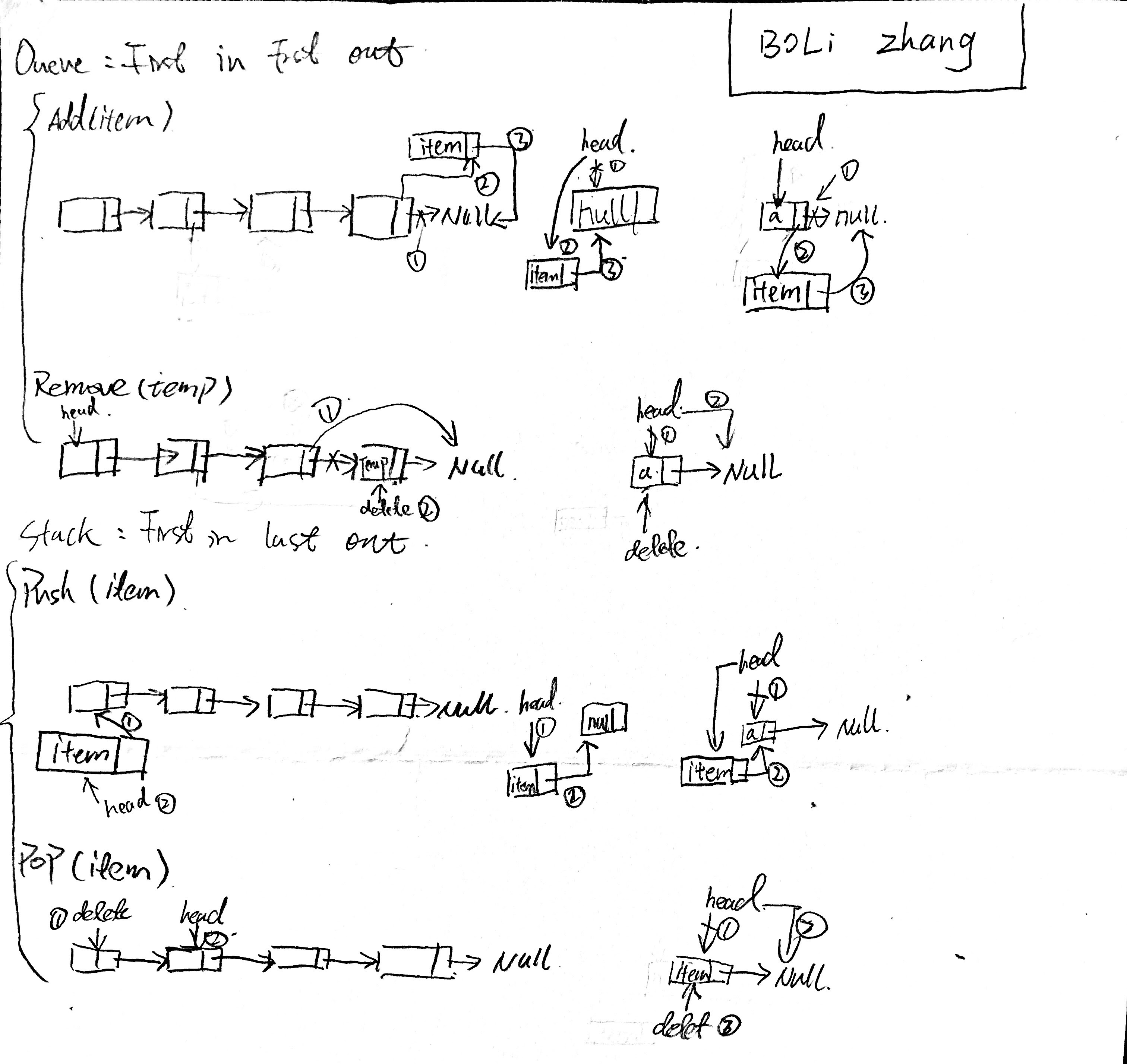
//



// main.cpp

// lab11b

//

// Created by Jeff on 10/24/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include "IntLinkedQueue.h"

using namespace std;

string del = ",";

void menu(){

cout <<" --- Lab\_11B LinkedQueue Test Menu --- \n"

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

<<"D - to Display \n"

<<"E - is Empty? \n"

<<"A - to Add \n"

<<"R - to Remove \n"

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

<<"C - to Clear \n"

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

<<"H - this menu \n"<<endl;

}

int main()

{

bool stay = true;

// create one instance for each of the test classes

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

string catchVar; // To hold values popped off the Queue

IntLinkedQueue q;

cout << "\nFirst, test with hard-wired data sets!\n"

<< "For example, \nyou may use one set of tokens for all test classes: \n";

string choice; // user input for choices

string str; // user input for list, delimiter, ...

int s[] = {1, 3, 5, 7, 9};

vector<int> input(s, end(s));

// show the initial item list

int input\_size = input.size();

for (auto item : input ) {

cout << item << ((input\_size <= 1) ? " " : ", ");

input\_size--;

}

cout << endl;

// initialize the Queue class instances

for (auto i: input) q.add( i );

cout << "\nUse menu to test a Queue class instance.\n";

menu();

// main menu while

while(stay) { // main menu while starts

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) { // main menu switch starts

case 'N':

case 'n':

{

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

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

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

ss << input; // string -> stream

while ( getline(ss, token, ',') ) { // stream -> string token

stringstream sst(token);

sst >> item;

q.add(item);

}

break;

}

case 'D':

case 'd':

{

cout << q.toString();

break;

}

case 'E':

case 'e':

{

if(!q.isEmpty()) {

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

}

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

break;

}

case 'H':

case 'h':

{

menu();

break;

}

case 'R':

case 'r':

{

int temp;

if( 0 < q.getSize()) q.remove(temp);

else cout << " queue is empty already";

cout << q.toString();

break;

}

case 'c':

case 'C':

{

q.clear();

break;

}

case 'l':

case 'L':

{

cout << q.getSize() << endl;

break;

}

case 'a':

case 'A':

{

int n;

cout << "Enter a number: ";

cin >> n;

q.add(n);

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

break;

}

// default:{

// cout << "invaild \n ";

// }

// }

}

}

}cout << " program end " ;

}

//

// IntLinkedQueue.h

// lab11b

//

// Created by Jeff on 10/24/16.

// Copyright © 2016 Jeff zhang. All rights reserved.

//

#ifndef IntLinkedQueue\_h

#define IntLinkedQueue\_h

#include<iostream>

#include<string>

using namespace std;

class IntLinkedQueue{

private:

struct Node{

int data;

Node \*next;

};

Node \*front, \*rear,\*p,\*pp;

int size;

public:

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

~IntLinkedQueue(){

clear();

};

bool isEmpty(){

if(front == nullptr){

return true;

}

else

return false;

}

int getSize()const{

return size;

};

void clear(){

p = front;

while(p){

pp = p;

p = p -> next;

delete (pp);

size--;

}

front = rear = p = pp = nullptr;

size = 0;

}

string toString(){

std::string s = "";

p = front;

while(p){

stringstream ss;

ss << p->data;

s += ss.str();

p = p->next;

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

}

return s;

}

void add(int n){

p = new Node;

p->next = nullptr;

p->data = n;

if(!front){

front = rear = p;

}

else {

p ->next = front;

front = p;

}

size++;

}

void remove (int &n){

if(front != rear){

p = front ;

while(p->next!=rear){

p = p -> next;

}

n = p ->data;

size --;

delete(rear);

rear = p ;

rear -> next = nullptr;

}

else{

// n = p ->data;

delete (rear);

size = 0 ;

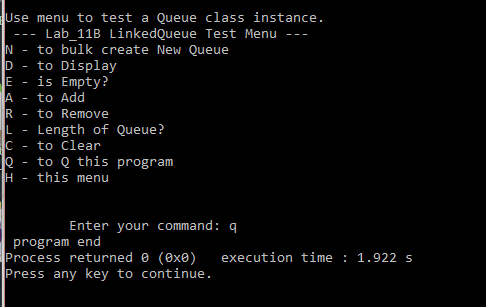
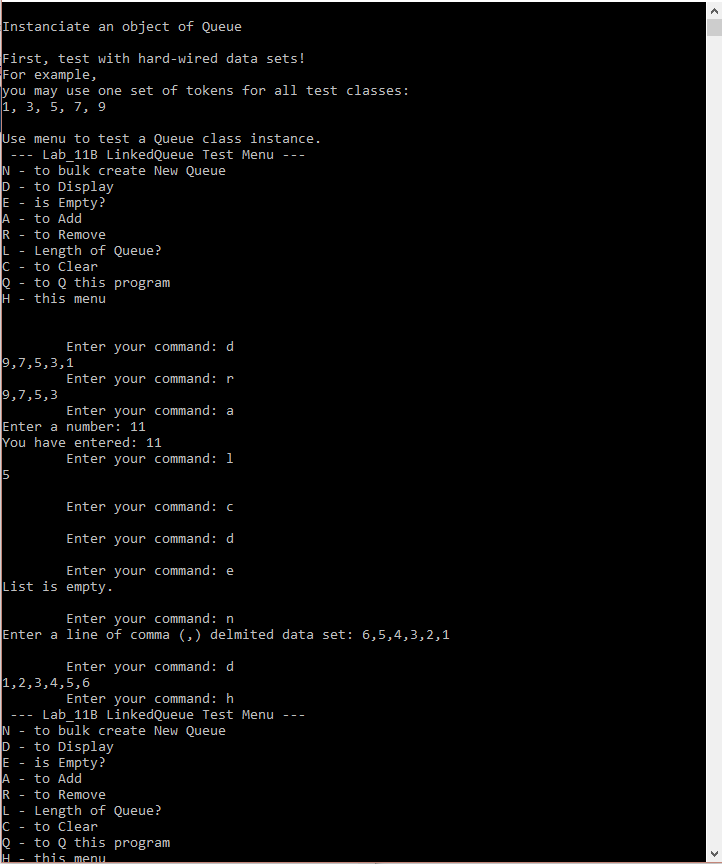
front = rear = p = pp = nullptr;

}

}

};

#endif /\* IntLinkedQueue\_h \*/



#include <iostream>

#include <sstream>

#include <vector>

#include <string>

#include "Q.h"

using namespace std;

int main() {

string catchS;

Q<string> strQ;

string sArr[] = {"using", "queue", "may", "be", "easier"};

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

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

cout << "\nstrQ: " << strQ.toString();

if( strQ.pop\_front(catchS) )

cout << "\n pop\_front() => " << catchS ;

else "\n pop\_front() => Empty strQ!";

if( strQ.pop\_back(catchS) )

cout << "\n pop\_back() => " << catchS ;

else "\n pop\_back() => Empty strQ!";

strQ.push\_front("Donald");

cout << "\n push\_front() => Donald";

strQ.push\_back("Duck");

cout << "\n push\_back() => Duck";

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

cout << "\n strQ[2] is " << strQ[2];

strQ[2] = "President";

cout << "\n strQ[2] set as 'President'\n strQ[2] is now " << strQ[2];

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

int catchI;

Q<int> intQ;

int iArr[] = {1 ,3, 5, 7, 9};

vector<int> inputI(iArr, end(iArr));

for (auto item: inputI) intQ.push\_front( item );

cout << "\n\nintQ: " << intQ.toString();

if( intQ.pop\_front(catchI) )

cout << "\n pop\_front() => " << catchI ;

else "\n pop\_front() => Empty intQ!";

if( intQ.pop\_back(catchI) )

cout << "\n pop\_back() => " << catchI ;

else "\n pop\_back() => Empty intQ!";

intQ.push\_front(11);

cout << "\n push\_front() => 11";

intQ.push\_back(22);

cout << "\n push\_back() => 22";

cout << "\n intQ: " << intQ.toString();

cout << "\n intQ[2] is " << intQ[2];

intQ[2] = 2;

cout << "\n intQ[2] = 2; \n intQ[2] is now " << intQ[2];

cout << "\n intQ: " << intQ.toString();

}

#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 )

return front = rear = pp = p = nullptr;

else

return 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

