**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-Ориентированное Программирование»**

Тема: Вектор.

|  |  |  |
| --- | --- | --- |
| Студент гр. 6382 |  | Швайко Т.А. |
| Преподаватель |  | Филатов А. Ю. |

Санкт-Петербург

2018

**Цель работы.**

Изучить принцип работы stl vector на основе двусвязного линейного списка, реализовав основной функционал итератора.

**Постановка задачи.**

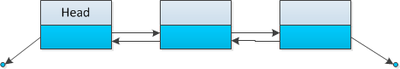
Реализовать собственный двунаправленный список (добавить в начало, в конец и по номеру, удалить из начала, из конца или по номеру) с внешним итератором (begin, end, operator \*, operator ->, operator + (int), operator - (int)).

**Основные теоретические положения.**

**Двусвязный список.**

Линейный двусвязного список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий и предыдущий элемент. Последний элемент списка указывает на NULL.

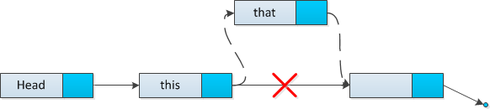
Простейшая реализация списка. В узлах хранятся данные и указатель на следующий и предыдущий элемент в списке.



Операции на списке

Очевиден случай, когда необходимо добавить элемент (newHead) в голову списка. Установим в этом элементе ссылку на старую голову, и обновим указатель на голову.

function insert(Node thatElement):





Удаление элемента после заданного (thisElement) происходит следующим образом: изменим ссылку на следующий элемент на следующий за удаляемым, затем удалим нужный объект.

**Итератор.**

Главное предназначение итераторов заключается в предоставлении возможности пользователю обращаться к любому элементу контейнера при сокрытии внутренней структуры контейнера от пользователя. Это позволяет контейнеру хранить элементы любым способом при допустимости работы пользователя с ним как с простой последовательностью или списком. Проектирование класса итератора обычно тесно связано с соответствующим классом контейнера. Обычно контейнер предоставляет методы создания итераторов.

Итератор похож на указатель своими основными операциями: он указывает на отдельный элемент коллекции объектов (предоставляет доступ к элементу) и содержит функции для перехода к другому элементу списка (следующему или предыдущему). Контейнер, который реализует поддержку итераторов, должен предоставлять первый элемент списка, а также возможность проверить, перебраны ли все элементы контейнера (является ли итератор конечным). В зависимости от используемого языка и цели, итераторы могут поддерживать дополнительные операции или определять различные варианты поведения.

**Спецификация программы:**

Создает вектор на основе двусвязного списка с возможностью перемещения с помощью итератора.

**Входные данные:**

Не необходимы.

**Выходные данные:**

На экран будет выведен вектор с измененными полями данных на 1.

**Реализация.**

Программа разбита на 2 заголовочных файл и 1 исполняемый файла:

Заголовочные файлы:

**“Iterator.hpp”**

template <typename T>

class Iterator {

friend class List<T>;

Elem<T> \*el;

public:

//Iterator(List<T> \*l): lst(l), el(l->head){};

//Iterator(List<T> \*l): el(l->head){};

Iterator(Elem<T> \*e): el(e){};

Iterator &operator++ (); //especial for prefix

Iterator &operator-- ();

void operator++ (int); //for postfix

void operator-- (int);

Elem<T> &operator\* ();

Elem<T> \*operator-> ();

bool operator!= (Iterator const& r);

Iterator &operator= (const Iterator &r);

};

|  |
| --- |
|  |

**“My\_vctor.hpp”**

template <typename T>

class Elem {

friend class List<T>;

friend class Iterator<T>;

private:

T info;

Elem \*prev;

Elem \*next;

public:

Elem(T inf) : info(inf) {};

T& get\_Info() {return info; }

};

//--------------CLASS-ELEM-----------------

//--------------CLASS-LIST-----------------

template <typename T>

class List {

friend class Iterator<T>;

private:

Elem<T> \*head;

Elem<T> \*tail;

public:

List():head(nullptr), tail(nullptr){}; //Constructor

~List(); //Destructor

bool is\_Empty();

void Print\_List();

void Add\_After(T data, size\_t k);

void Add\_to\_Head(T data);

void Add\_to\_Tail(T data);

size\_t Quantity();

size\_t Quantity(Elem<T>);

Elem<T> \*Init(T data);

Elem<T> \*get\_Elem(size\_t index);

T operator[] (size\_t index);

List<T> &operator= (const List<T> &r);

Iterator<T> begin(){ return Iterator<T>(head); }

Iterator<T> end() { return Iterator<T>(nullptr); }

};

Функция bool is\_Empty() определяет, пуст ли список. Если указатель на голову списка указывает на NULL, значит список пуст. Возвращает true or false.

Функция size\_t Quantity() определяет размер списка. Принимает на вход указатель на голову списка, выполняет проверку на пустоту списка, пока указатель не NULL переходит к следующему указателю, прибавляя в счетчике. Возвращает длину списка.

Функция Elem<T> \*get\_Elem(size\_t index) возвращает указатель текущего узла списка.

Функция void Add\_After(T data, size\_t k) вставляет новый элемент с список после текущего элемента. Принимает на вход индекс и значение, которое будет хранить узел. Создает новый узел с новым значением, переприсваивает указатели, если же список пуст, меняет значения указателя на голову списка.

Функция int erase() удаляет элемент из списка. Принимает на вход указатель на узел списка, который нужно удалить. Переприсваивает указатели, очищает память при помощи функции free(). Если узел головной, то меняем значение указателя на голову.

Деструктор ~List() удаляет весь список. Последовательно очищает память при помощи функции delete().

Функция void Print\_List() печатает весь список.

**Выводы.**

В данной лабораторной работе были освоены основы работы stl vector на примере двусвязного списка с итератором.

**Приложение A.**

**Main.cpp:**

// lab 3 stack logical expression.cpp : Defines the entry point for the console application.

//

// Программа клиент вычисляет арифметическое выражение, заданное в постфиксной форме

#include "stdafx.h"

#include <iostream>

#include <fstream>

#include <string>

#include <stdio.h>

#include <cstdlib>

#include <windows.h>

#include "st\_interf1.h"

using namespace std;

using namespace st\_modul1;

void menu(){

cout << "1 - Enter from file" << endl;

cout << "2 - Enter from console" << endl;

cout << "3 - Exit" << endl;

}

bool true\_or\_false(char a){

return a == '1';

}

void result(char a){

if (true\_or\_false(a) )

cout << "Результат = true " << endl;

else

cout << "Результат = false " << endl;

}

void er\_brek\_open(){

cout << "There must be open breket - (" << endl;

exit(1);

}

void er\_brek\_clouse(){

cout << "There must be clouse breket - )" << endl;

exit(1);

}

void error(){

cout << "There must be 0 or 1" << endl;

exit(1);

}

char not(char a){

if (a == '0')

return '1';

if (a == '1')

return '0';

}

char or (char a, char b){

if (a == '0' && b == '0')

return '0';

else return '1';

}

char and(char a, char b){

if (a == '1' && b == '1')

return '1';

else return '0';

}

char number(char a){

if ( a == '0' )

return '0';

if (a == '1')

return '1';

return 'e'; // e - not number!

}

char operation(char a){

if (a == '|')

return '|';

if (a == '&')

return '&';

return 'e';

}

void calc\_log\_expr(char \*a, size\_t n, Stack s){

char op, x, y, z, res;

for (size\_t i = 0; i < n; i++) {

while (a[i] != ')' && i < n) { //stored symbols in stack

s.push(a[i]);

i++;

}

if (n > 1) {

x = s.pop2(); // get value of operand

if (number(x) == 'e') // if x isn't 0 or 1 then error

error();

cout << "\tpop -> " << x << endl;

op = s.pop2(); // get symbol of operation

cout << "\tpop -> " << op << endl;

if (op == '!') { // check all operation in expression

res = not(x);

}

else if (op == '&') {

y = s.pop2(); //get element for calculating expression

if (number(y) == 'e') // if x isn't 0 or 1 then error

error();

cout << "\tpop -> " << y << endl;

res = and (x, y);

}

else if (op == '|') {

y = s.pop2();

if (number(y) == 'e') // if x isn't 0 or 1 then error

error();

cout << "\tpop -> " << y << endl;

res = or (x, y);

}

else { // didn't right expression

cout << "There must be operation" << endl;

exit(1);

}

z = s.pop2();

if (z != '(') // error

er\_brek\_open();

if (a[i] != ')') // if a[i-1] isn't clouse breket then error

er\_brek\_clouse();

s.push(res);

}

}

result(s.pop2());

}

int main()

{

char a[100];

// это вставка для правильной кодировки русских букв

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

ifstream fin("postfix.txt");

fin >> noskipws; // включить манипулятор!

if (!fin){

cout << "File not open for reading!\n";

return 1;

}

int n1 = 100;

int n = 0;

int sw;

bool flag = true;

Stack s;

while (flag) {

menu();

cin >> sw;

switch (sw) {

case 1:

while (n<n1 && fin >> a[n]) n++;

break;

case 2:

cin >> a;

n = strlen(a);

break;

case 3:

flag = false;

exit(1);

break;

default:

cout << "Error choyce!" << endl;

exit(1);

}

//вывод строки

for (int i = 0; i < n; i++) cout << a[i];

cout << endl;

cout << "Calculate!" << endl;

calc\_log\_expr(a, n, s);

s.destroy();

}

system("pause");

return (0);

}

**St\_impl1.cpp:**

#include "stdafx.h"

#include <iostream>

#include <cstdlib>

#include "st\_interf1.h"

using namespace std;

namespace st\_modul1

{

struct Stack::node {//

base \*hd;

node \*tl;

// constructor

node()

{

hd = NULL; tl = NULL;

}

};// end node

//-------------------------------------

base Stack::top(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }

else return \*topOfStack->hd;

}

//-------------------------------------

void Stack::pop(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{

node \*oldTop = topOfStack;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

}

}

//-------------------------------------

base Stack::pop2(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{

node \*oldTop = topOfStack;

base r = \*topOfStack->hd;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

return r;

}

}

//-------------------------------------

void Stack::push(const base &x)

{

node \*p;

p = topOfStack;

topOfStack = new node;

if (topOfStack != NULL) {

topOfStack->hd = new base;

\*topOfStack->hd = x;

cout << "push -> " << x << endl; // Demo

topOfStack->tl = p;

}

else { cerr << "Memory not enough\n"; exit(1); }

}

//-------------------------------------

bool Stack::isNull(void)

{

return (topOfStack == NULL);

}

//-------------------------------------

void Stack::destroy(void)

{

while (topOfStack != NULL) {

pop();

}

}

} // end of namespace st\_modul1

**St\_interf1.hpp:**

#pragma once

// интерфейс АТД "Стек" (ссылочная реализация в динамической памяти)

namespace st\_modul1

{

//-------------------------------------

typedef char base;

class Stack {

private:

struct node;

/\* определение структуры будет дано в другом файле (продолжении namespace st\_modul) - в файле Implementation,

а здесь достаточно объявления "struct node;"

\*/

node \*topOfStack;

public:

Stack()

{

topOfStack = NULL;

};

base top(void);

void pop(void);

base pop2(void);

void push(const base &x);

bool isNull(void);

void destroy(void);

};

}