Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу “Логика и основа алгоритмизации в ИЗ”

на тему “ Динамические списки”

Выполнили студенты группы 21ВВ3:

Головинов Владислав

Пронин Илья

Приняли:

Митрохин М.А., Юрова О.В.

Пенза 2022

**Название**

Динамические списки.

**Цель работы:** выполнить лабораторные указания 1 реализуя приоритетную очередь, путем добавления элемента в список в соответствии с приоритетом объекта. Создать структуры данных для лабораторных заданий 2 – 3, реализовав для них определенные функции.

**Лабораторное задание:**

1. Реализовать приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта (т.е. объект с большим приоритетом становится перед объектом с меньшим приоритетом).
2. \* На основе приведенного кода реализуйте структуру данных *Очередь*.
3. \* На основе приведенного кода реализуйте структуру данных *Стек*.

**Задание 1:**

**Листинг**

#include <iostream>

#include <time.h>

#include <locale>

#include <queue>

using namespace std;

struct pri\_queue

{

int num;

int priority;

pri\_queue\* next = nullptr;

};

void insert(pri\_queue\*& main, pri\_queue\* new\_el) {

pri\_queue\* tmp = main;

pri\_queue\* left = main;

while (tmp != nullptr && new\_el->priority < tmp->priority) {

left = tmp;

tmp = tmp->next;

}

if (tmp != left) {

left->next = new\_el;

}

else main = new\_el;

new\_el->next = tmp;

}

int main()

{

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int count;

cout << "Введите кол-во данных: ";

cin >> count;

if (count <= 1) count = 2;

pri\_queue\* head = nullptr;

while(count){

pri\_queue\* tmp = new pri\_queue;

cout << "Введите число: ";

cin >> tmp->num;

cout << "Введите приоритет: ";

cin >> tmp->priority;

insert(head, tmp);

count--;

}

pri\_queue\* tmp = head;

while (tmp != nullptr) {

cout << tmp->num << endl;

tmp = tmp->next;

}

return 0;

}

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

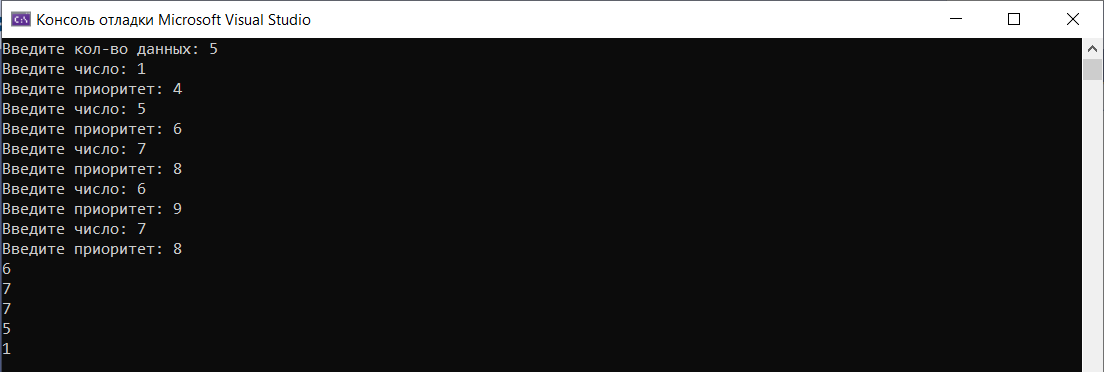


Рис. 1

Протокол трассировки программы показан на рисунке 2.

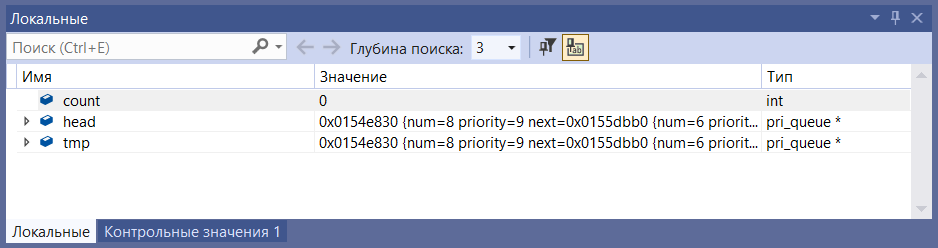


Рис. 2

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1, совпал с результатами трассировки на рисунке 2.

**Задание 2:**

**Листинг**

//List

template <typename T>

class L\_ptr;

template <typename T>

class List {

public:

List\* next = nullptr;

T number;

// конструктор

List() : number(0), next(nullptr) { };

List(int number) : number(number), next(nullptr) { };

List(const List& a) {

this->number = a.number;

this->next = a.next;

};

// булевые операторы

bool operator == (List& a) { return number == a.number; };

bool operator >= (List& a) { return number >= a.number; };

bool operator <= (List& a) { return number <= a.number; };

bool operator < (List& a) { return number < a.number; };

bool operator > (List& a) { return number > a.number; };

bool operator == (int a) { return number == a; };

bool operator >= (int a) { return number >= a; };

bool operator <= (int a) { return number <= a; };

bool operator < (int a) { return number < a; };

bool operator > (int a) { return number > a; };

//

T& operator = (List\* a) { //

this->number = a->number;

return this->number;

};

operator int() { return this->number; };

~List() { delete next; next = nullptr; }

friend L\_ptr<T>;

};

// L\_ptr.h

#include "list.cpp"

#include <iostream>

template <typename T>

class L\_ptr {

protected:

int size;

List<T>\* head = nullptr;

List<T>\* end = nullptr;

List<T>\* \_\_index\_\_(int count);

public:

L\_ptr() : size(0) { };

L\_ptr(int size) : size(size) { };

int operator [] (int a);

const List<T>\* Get\_head() { return head; };

List<T>\* top();

void pop();

// находим элемент по данным

List<T>\* find(T data);

// удаление узла

void Del\_vertex(int count);

// меняем местами два узла

void Swap(int top1, int top2);

// проход по списку

int at(int count);

// получить длину списка

int lenght();

// вывести все на экран

void Display();

};

// L\_ptr.cpp

#include "L\_ptr.h"

template<typename T>

void L\_ptr<T>::Display(){

List<T>\* x = head;

while (x != nullptr) {

std::cout << x->number << std::endl;

x = x->next;

}

}

template<typename T>

int L\_ptr<T>::at(int count) {

if (count < 0 || count > size) return -1;

else {

return this->operator [](count);

}

}

template<typename T>

List<T>\* L\_ptr<T>::\_\_index\_\_(int count) {

if (count < 0 || count >= size) nullptr;

int id = 0;

List<T>\* tmp = head;

while (id + 1 != count) {

tmp = tmp->next;

id++;

}

return tmp;

}

template<typename T>

void L\_ptr<T>::Del\_vertex(int count) {

size--;

if (count == 0) {

List<T>\* tmp = head;

head = head->next;

tmp = nullptr;

}

else {

List<T>\* tmp = \_\_index\_\_(count);

tmp->next = tmp->next->next;

}

};

template<typename T>

int L\_ptr<T>::lenght() { return size; };

template<typename T>

List<T>\* top() {

if (this->size <= 0) {

std::cout << "Данных нет" << std::endl;

\_sleep(3000);

abort();

}

return this->head;

};

template<typename T>

void L\_ptr<T>::pop() {

if (this->size <= 0) {

std::cout << "Данных нет" << std::endl;

\_sleep(3000);

abort();

}

List<T>\* tmp = this->head;

this->head = this->head->next;

tmp->next = nullptr;

delete tmp;

};

template<typename T>

List<T>\* L\_ptr<T>::find(T data) {

if (size == 0) {

abort();

}

List<T>\* tmp = this->head;

while (tmp->number != data) {

if (tmp->next == nullptr) {

return this->head;

}

tmp = tmp->next;

}

return tmp;

}

template<typename T>

int L\_ptr<T>::operator [] (int a) {

List<T>\* x = head;

if (size == 0) {

std::cout << "Лист пуст" << std::endl;

return 0;

}

while (a != 0) {

x = x->next;

a--;

if (a != 0 && x == nullptr || a != 0 && x->next == nullptr || a == 0 && x == nullptr) {

std::cout << "Ошибка скобок" << std::endl;

return 0;

}

}

return x->number;

}

template<typename T>

void L\_ptr<T>::Swap(int top1, int top2) {

List<T>\* head\_tmp1 = nullptr;

List<T>\* head\_tmp2 = nullptr;

List<T>\* tmp1 = nullptr, \* tmp2 = nullptr;

List<T>\* x = head;

if (top1 == 0) {

tmp1 = head;

head\_tmp1 = head;

}

if (top2 >= size) {

std::cout << "Ошибка скобок" << std::endl;

return;

}

while (top2 != 0) {

if (top1 == 1) { head\_tmp1 = x; tmp1 = x->next; }

if (top2 == 1) { head\_tmp2 = x; tmp2 = x->next; }

x = x->next;

top1--; top2--;

}

int g;

tmp2 = tmp1;

tmp1 = x;

//

g = tmp2->number;

tmp2->number = tmp1->number;

tmp1->number = g;

}

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

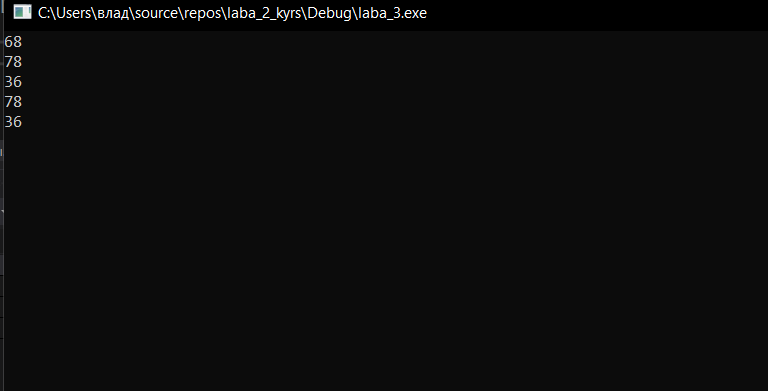
****

Рис. 1

Протокол трассировки программы показан на рисунке 2.

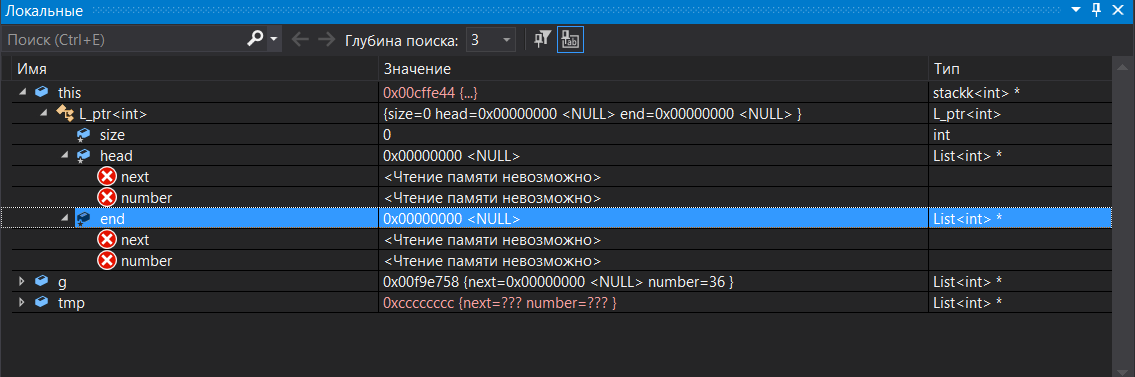


Рис. 2

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1, совпал с результатами трассировки на рисунке 2.

Вывод: В результате выполнения второго задания мы научились реализовывать структуру данных – очередь.

**Задание 3:**

**Листинг**

//List

template <typename T>

class L\_ptr;

template <typename T>

class List {

public:

List\* next = nullptr;

T number;

// конструктор

List() : number(0), next(nullptr) { };

List(int number) : number(number), next(nullptr) { };

List(const List& a) {

this->number = a.number;

this->next = a.next;

};

// булевые операторы

bool operator == (List& a) { return number == a.number; };

bool operator >= (List& a) { return number >= a.number; };

bool operator <= (List& a) { return number <= a.number; };

bool operator < (List& a) { return number < a.number; };

bool operator > (List& a) { return number > a.number; };

bool operator == (int a) { return number == a; };

bool operator >= (int a) { return number >= a; };

bool operator <= (int a) { return number <= a; };

bool operator < (int a) { return number < a; };

bool operator > (int a) { return number > a; };

//

T& operator = (List\* a) { //

this->number = a->number;

return this->number;

};

operator int() { return this->number; };

~List() { delete next; next = nullptr; }

friend L\_ptr<T>;

};

// L\_ptr.h

#include "list.cpp"

#include <iostream>

template <typename T>

class L\_ptr {

protected:

int size;

List<T>\* head = nullptr;

List<T>\* end = nullptr;

List<T>\* \_\_index\_\_(int count);

public:

L\_ptr() : size(0) { };

L\_ptr(int size) : size(size) { };

int operator [] (int a);

const List<T>\* Get\_head() { return head; };

List<T>\* top();

void pop();

// находим элемент по данным

List<T>\* find(T data);

// удаление узла

void Del\_vertex(int count);

// меняем местами два узла

void Swap(int top1, int top2);

// проход по списку

int at(int count);

// получить длину списка

int lenght();

// вывести все на экран

void Display();

};

// L\_ptr.cpp

#include "L\_ptr.h"

template<typename T>

void L\_ptr<T>::Display(){

List<T>\* x = head;

while (x != nullptr) {

std::cout << x->number << std::endl;

x = x->next;

}

}

template<typename T>

int L\_ptr<T>::at(int count) {

if (count < 0 || count > size) return -1;

else {

return this->operator [](count);

}

}

template<typename T>

List<T>\* L\_ptr<T>::\_\_index\_\_(int count) {

if (count < 0 || count >= size) nullptr;

int id = 0;

List<T>\* tmp = head;

while (id + 1 != count) {

tmp = tmp->next;

id++;

}

return tmp;

}

template<typename T>

void L\_ptr<T>::Del\_vertex(int count) {

size--;

if (count == 0) {

List<T>\* tmp = head;

head = head->next;

tmp = nullptr;

}

else {

List<T>\* tmp = \_\_index\_\_(count);

tmp->next = tmp->next->next;

}

};

template<typename T>

int L\_ptr<T>::lenght() { return size; };

template<typename T>

List<T>\* top() {

if (this->size <= 0) {

std::cout << "Данных нет" << std::endl;

\_sleep(3000);

abort();

}

return this->head;

};

template<typename T>

void L\_ptr<T>::pop() {

if (this->size <= 0) {

std::cout << "Данных нет" << std::endl;

\_sleep(3000);

abort();

}

List<T>\* tmp = this->head;

this->head = this->head->next;

tmp->next = nullptr;

delete tmp;

};

template<typename T>

List<T>\* L\_ptr<T>::find(T data) {

if (size == 0) {

abort();

}

List<T>\* tmp = this->head;

while (tmp->number != data) {

if (tmp->next == nullptr) {

return this->head;

}

tmp = tmp->next;

}

return tmp;

}

template<typename T>

int L\_ptr<T>::operator [] (int a) {

List<T>\* x = head;

if (size == 0) {

std::cout << "Лист пуст" << std::endl;

return 0;

}

while (a != 0) {

x = x->next;

a--;

if (a != 0 && x == nullptr || a != 0 && x->next == nullptr || a == 0 && x == nullptr) {

std::cout << "Ошибка скобок" << std::endl;

return 0;

}

}

return x->number;

}

template<typename T>

void L\_ptr<T>::Swap(int top1, int top2) {

List<T>\* head\_tmp1 = nullptr;

List<T>\* head\_tmp2 = nullptr;

List<T>\* tmp1 = nullptr, \* tmp2 = nullptr;

List<T>\* x = head;

if (top1 == 0) {

tmp1 = head;

head\_tmp1 = head;

}

if (top2 >= size) {

std::cout << "Ошибка скобок" << std::endl;

return;

}

while (top2 != 0) {

if (top1 == 1) { head\_tmp1 = x; tmp1 = x->next; }

if (top2 == 1) { head\_tmp2 = x; tmp2 = x->next; }

x = x->next;

top1--; top2--;

}

int g;

tmp2 = tmp1;

tmp1 = x;

//

g = tmp2->number;

tmp2->number = tmp1->number;

tmp1->number = g;

}

template <typename T>

class queuee : public L\_ptr<T> {

public:

queuee() : L\_ptr<T>() { };

void Push(List<T>\* g) {

List<T>\* tmp = new List<T>;

\*tmp = g;

if (this->size == 0) {

this->end = tmp;

this->head = tmp;

}

else {

this->end->next = tmp;

this->end = tmp;

}

this->size++;

};

};

int main()

{

setlocale(LC\_ALL, "rus");

srand(static\_cast<unsigned int>(time(NULL)));

stackk<int> a;

queuee<int> b;

List<int>\* tmp1 = new List<int>;

List<int>\* tmp2 = new List<int>;

List<int>\* tmp3 = new List<int>;

tmp1->number = rand() % 101;

tmp2->number = rand() % 101;

tmp3->number = rand() % 101;

b.Push(tmp1);

b.Push(tmp2);

b.Push(tmp3);

a.Push(tmp1);

a.Push(tmp2);

a.Push(tmp3);

cout << a[0] << endl;

cout << a[1] << endl;

cout << a[2] << endl;

a.Display();

return 0;

}

**Результаты работы программы:**

Результаты работы программы показан на рисунке 1.

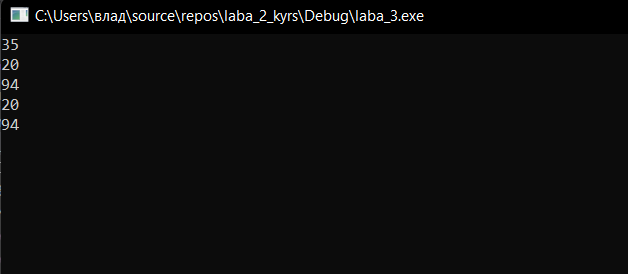
****

Рис. 1

Протокол трассировки программы показан на рисунке 2.

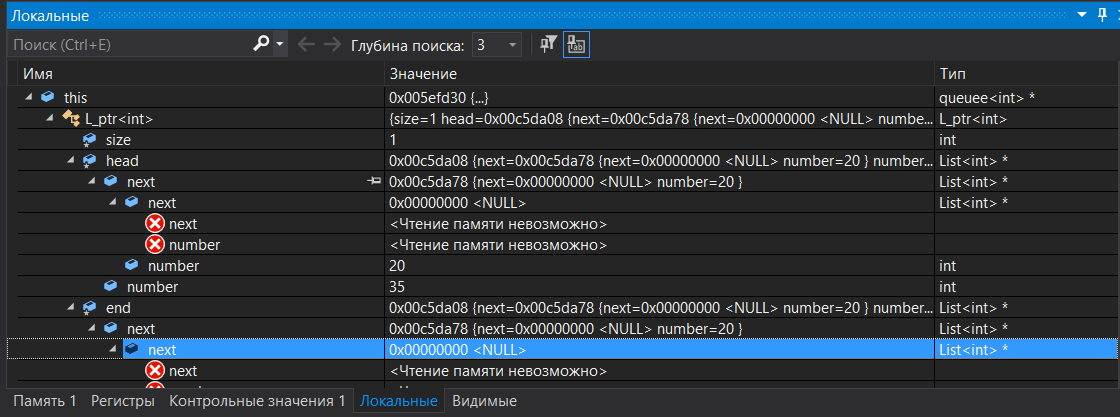


Рис. 2

Результат получился правильный, все действия совершены верно.

Результат работы программы, показанный на рисунке 1, совпал с результатами трассировки на рисунке 2.

Вывод: В результате выполнения второго задания мы научились реализовывать структуру данных – стек и поняли как реализовано добавление элементов в него.

**Вывод:** Мы выполнили лабораторные указания 1-3 реализуя приоритетную очередь, путем добавления элемента в список в соответствии с приоритетом объекта.