Министерство науки и высшего 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:

Савосин Владислав

Юсеев Руслан

Димитренко Никита

Приняли:

Митрохин М.А. , Юрова О.В.

Пенза 2022

**Название:**

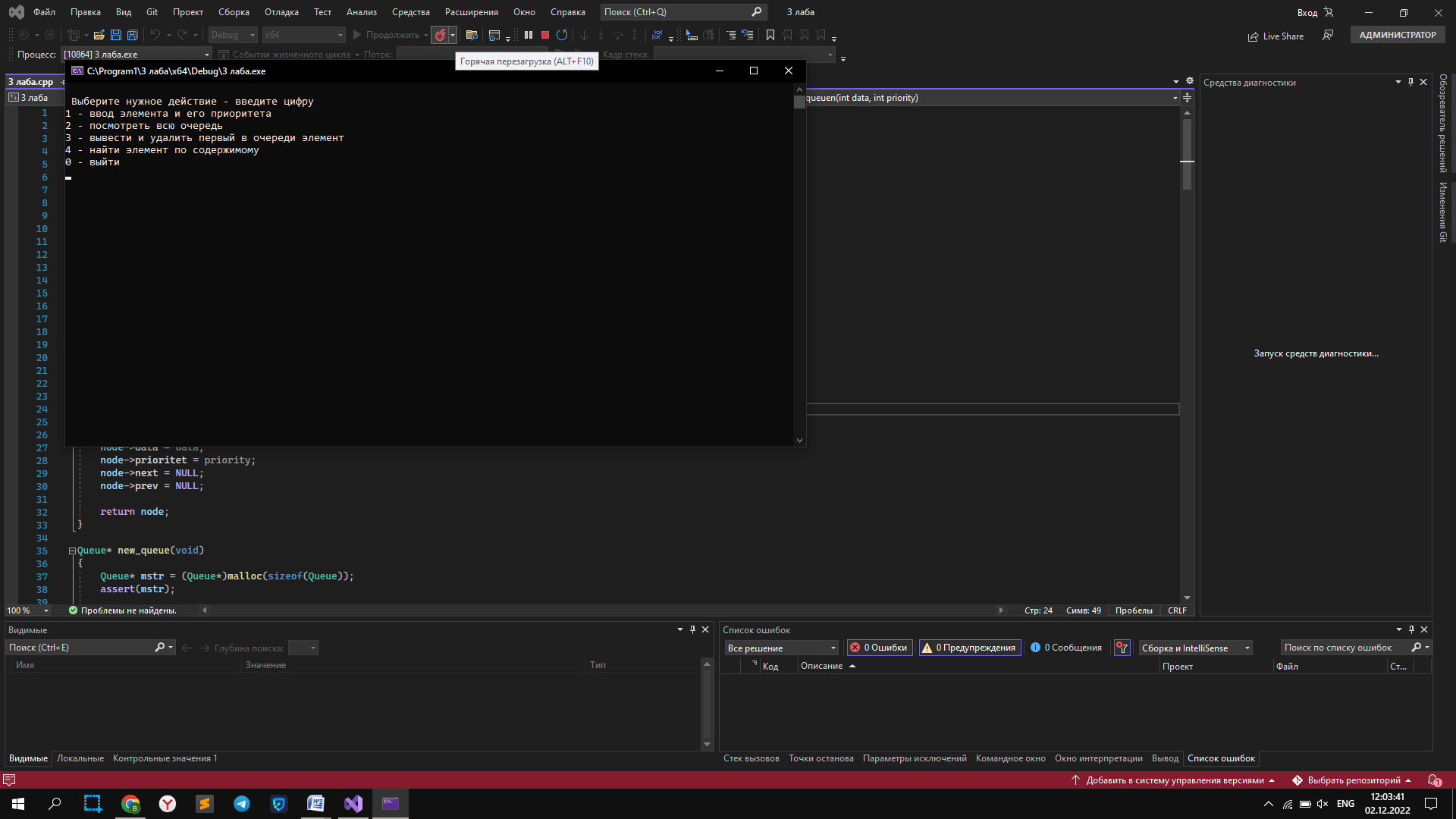
Динамические списки

**Цель работы:** Оценить время выполнения программы, выполняя лабораторные указания 1-2.

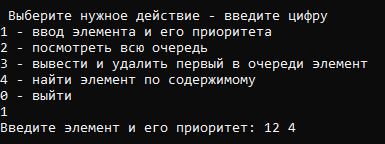
**Лабораторное задание:**

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).

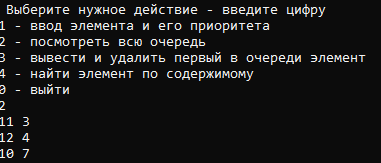
Навигационное меню приложения :



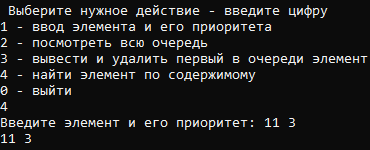
Вводим элементы, которые будут добавлены в порядке возрастания заданного приоритета:

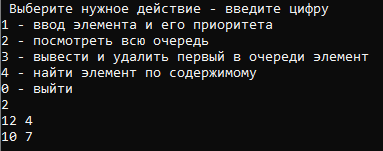


Существует возможность просмотреть всю очередь:



Находим элемент по содержимому:



Выводим и удаляем первый элемент в очереди, а затем просматриваем её: 

#include <iostream>

#include <conio.h>

#include <assert.h>

//using namespace std;

struct MyStr

{

int data;

MyStr\* next;

MyStr\* prev;

int prioritet;

};

struct Queue

{

MyStr\* head;

MyStr\* tail;

};

static MyStr\* queuen(int data, int priority) {

MyStr\* node = (MyStr\*)malloc(sizeof(MyStr));

assert(node);

node->data = data;

node->prioritet = priority;

node->next = NULL;

node->prev = NULL;

return node;

}

Queue\* new\_queue(void)

{

Queue\* mstr = (Queue\*)malloc(sizeof(Queue));

assert(mstr);

mstr->head = NULL;

mstr->tail = NULL;

return mstr;

}

int pq\_is\_empty(Queue\* pq) {

return (pq->head == NULL);

}

void q\_push(Queue\* pq, int data, int priority) {

MyStr\* node = queuen(data, priority);

if (pq\_is\_empty(pq)) {

pq->head = pq->tail = node;

}

else {

MyStr\* last = pq->tail;

while (last && last->prioritet > priority) {

last = last->prev; //последний=предыдущий

}

while (last && last->prioritet == priority && last->data < data) //это уже сортировка по значению

last = last->prev;

if (!last) { //если получился элемент с наивысшим приоритетом

node->next = pq->head; //указатель на следующую переменную=начало всей очереди

pq->head->prev = node;

pq->head = node;

}

else if (last == pq->tail) { //если получился с наименьшим

node->prev = pq->tail; //указатель на предыдуций элемент=конец всей очереди

pq->tail->next = node;

pq->tail = node;

}

else { //элемент между наивысш и наименьш приоритетами

node->prev = last; //предыдущий=последний найденный приоритет

node->next = last->next; //следующий=следующий после последнего найденного

last->next->prev = node;

last->next = node;

}

}

}

unsigned pq\_size(Queue\* pq) { //размер очереди

unsigned size = 0;

MyStr\* node;

for (node = pq->head; node; node = node->next)

++size;

return size;

}

void q\_find(Queue\* pq, int data, int prioritet) {

MyStr\* last = pq->head;

while (last && last != pq->tail)

{

if (last->data == data && last->prioritet == prioritet)

{

std::cout << (int)last->data << " " << (int)last->prioritet << std::endl;

}

last = last->next;

}

if (pq->tail->data == data && pq->tail->prioritet == prioritet)

{

std::cout << (int)pq->tail->data << " " << (int)pq->tail->prioritet << std::endl;

}

}

int qdata(Queue\* pq)

{

assert(!pq\_is\_empty(pq));

return pq->head->data;

}

int qprioritet(Queue\* pq)

{

assert(!pq\_is\_empty(pq));

return pq->head->prioritet;

}

void q\_pop(Queue\* pq)

{

assert(!pq\_is\_empty(pq));

if (pq->head == pq->tail) {

free(pq->head);

pq->head = pq->tail = NULL;

}

else {

MyStr\* s = pq->head->next;

s->prev = NULL;

free(pq->head);

pq->head = s;

}

}

void pq\_print(Queue\* pq) {

MyStr\* last = pq->head;

while (last && last != pq->tail)

{

std::cout << (int)last->data << " " << (int)last->prioritet << std::endl;

last = last->next;

}

std::cout << (int)pq->tail->data << " " << (int)pq->tail->prioritet << std::endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int num, pr;

Queue\* q = new\_queue(); //создание очереди

bool flag = true;

while (flag)

{

std::cout << "\n Выберите нужное действие - введите цифру" << std::endl;

std::cout << "1 - ввод элемента и его приоритета" << std::endl;

std::cout << "2 - посмотреть всю очередь" << std::endl;

std::cout << "3 - вывести и удалить первый в очереди элемент" << std::endl;

std::cout << "4 - найти элемент по содержимому" << std::endl;

std::cout << "0 - выйти" << std::endl;

int n;

std::cin >> n;

switch (n)

{

case 1:

{

std::cout << "Введите элемент и его приоритет: ";

std::cin >> num >> pr;

q\_push(q, num, pr); //добавление элемента

break;

}

case 2:

{

if (pq\_size(q) == 0)

{

std::cout << "Очередь пуста!" << std::endl;

break;

}

pq\_print(q);

break;

}

case 3:

if (pq\_size(q) == 0)

{

std::cout << "Очередь пуста!" << std::endl;

break;

}

std::cout << qdata(q) << " " << qprioritet(q) << std::endl;

q\_pop(q);

break;

case 4:

if (pq\_size(q) == 0)

{

std::cout << "Очередь пуста!" << std::endl;

break;

}

std::cout << "Введите элемент и его приоритет: ";

std::cin >> num >> pr;

q\_find(q, num, pr);

break;

case 0:

exit(1);

default:

std::cout << "неправильное число!" << std::endl;

break;

}

}

system("pause");

return 0;

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