Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

По лабораторной работе №3

По курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

**Выполнили**

**студенты группы 23ВВВ2:**

Пырков Д. А.

Родионов А. А.

**Приняли:**

Митрохин М. А.

Юрова О. В.

Пенза 2024

**Цель работы:** Изучить и практически реализовать основные структуры данных на базе динамических списков. Освоить принципы работы и реализовать приоритетную очередь с использованием динамического списка, обеспечивая вставку элементов в соответствии с их приоритетом. Изучить и реализовать структуры данных «Очередь» и «Стек» на основе динамического списка, с использованием предоставленного кода.

**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).

**\*Задание 2:**

На основе приведенного кода реализуйте структуру данных Очередь.

**\*Задание 3:**

На основе приведенного кода реализуйте структуру данных Стек.

**Ход работы**

**Задание 1**

**Описание программы**

Программа демонстрирует базовую реализацию приоритетной очереди и предоставляет простой интерфейс для работы с ней. Программа устроена следующим образом:

1. Структура данных:

* Определена структура Node, которая представляет элемент очереди. Она содержит данные (data), приоритет (priority) и указатель на следующий узел (next).

1. Основные функции:
   * createNode: Создает новый узел с заданными данными и приоритетом.
   * enqueue: Добавляет новый элемент в очередь, сохраняя порядок по приоритету (элементы с высшим приоритетом в начале).
   * deleteByPriority: Удаляет элемент с наивысшим приоритетом (первый в списке).
   * deleteByContent: Удаляет элемент по заданному значению данных.
   * printList: Выводит содержимое очереди.
2. Главная функция (main):
   * Реализует интерфейс пользователя через бесконечный цикл с меню.
   * Пользователь может выбирать различные действия:
3. Добавить элемент
4. Удалить элемент с наивысшим приоритетом
5. Удалить элемент по содержимому
6. Показать очередь
7. Выйти из программы
8. Особенности реализации:
   * Очередь поддерживает порядок по приоритету: элементы с высшим приоритетом располагаются ближе к началу списка.
   * При добавлении нового элемента (enqueue) он вставляется в правильную позицию согласно его приоритету.
   * Удаление по приоритету всегда удаляет первый элемент (с наивысшим приоритетом).
   * Удаление по содержимому ищет элемент по всему списку и удаляет первое совпадение.
9. Обработка ошибок:
   * Программа проверяет на пустоту очереди перед операциями удаления.
   * При попытке удалить несуществующий элемент выводится соответствующее сообщение.

Листинг программы смотрите в Приложении А.

**Результат работы программы**

На рисунках 1 и 2 приведён результат работы программы №1.

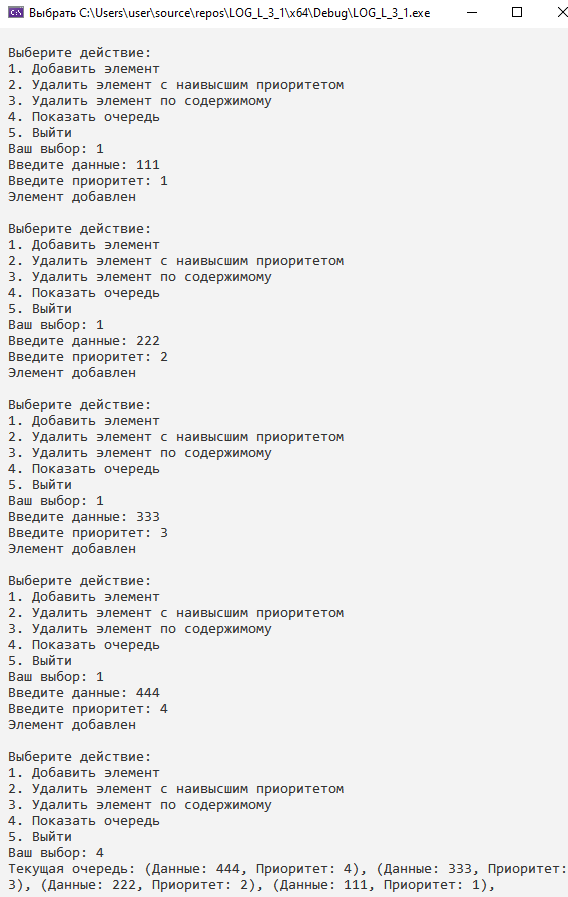


Рисунок 1 – результат работы программы №1 (часть 1)

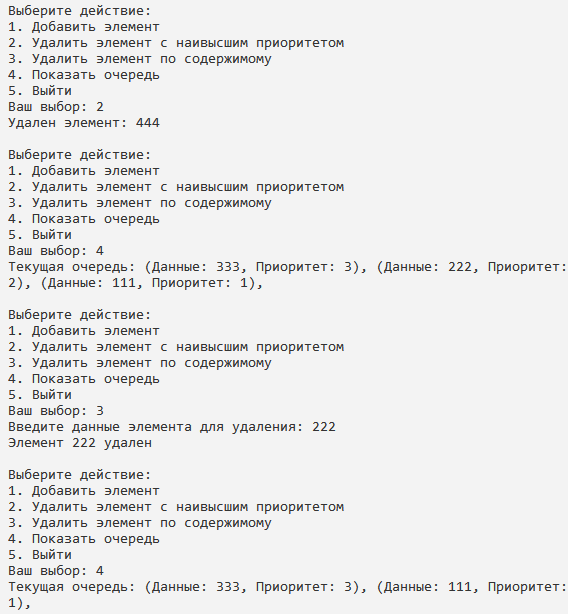


Рисунок 2 – результат работы программы №1 (часть 2)

**\*Задание 2**

**Описание программы**

Программа реализует структуру данных «Очередь» с использованием связного списка. Это базовая реализация очереди, позволяющая добавлять, удалять элементы и просматривать содержимое очереди. Программа работает следующим образом:

1. Структура node определяет элемент очереди, содержащий информационное поле (строку) и указатель на следующий элемент.
2. Глобальные переменные head и tail указывают на начало и конец очереди, size хранит количество элементов.
3. Функции:
   * get\_struct(): создает новый элемент, запрашивая у пользователя ввод;
   * enqueue(): добавляет элемент в конец очереди;
   * dequeue(): удаляет элемент из начала очереди;
   * display(): выводит содержимое очереди;
   * front(): возвращает первый элемент очереди;
   * is\_empty(): проверяет, пуста ли очередь.
4. В main() реализовано меню, позволяющее пользователю выбирать действия:
   * Добавить элемент
   * Удалить элемент
   * Показать содержимое
   * Показать первый элемент
   * Выйти
5. При добавлении элемента (enqueue()):
   * Если очередь пуста, новый элемент становится и головой, и хвостом.
   * Иначе, новый элемент добавляется в конец и становится новым хвостом.
6. При удалении элемента (dequeue()):
   * Удаляется первый элемент (голова).
   * Второй элемент становится новой головой.
   * Если удален последний элемент, обнуляется указатель на хвост.
7. Программа использует динамическое выделение памяти для создания новых элементов и освобождает память при удалении элементов.

Листинг программы смотрите в Приложении В.

**Результат работы программы**

На рисунке 3 приведён результат работы программы №2.

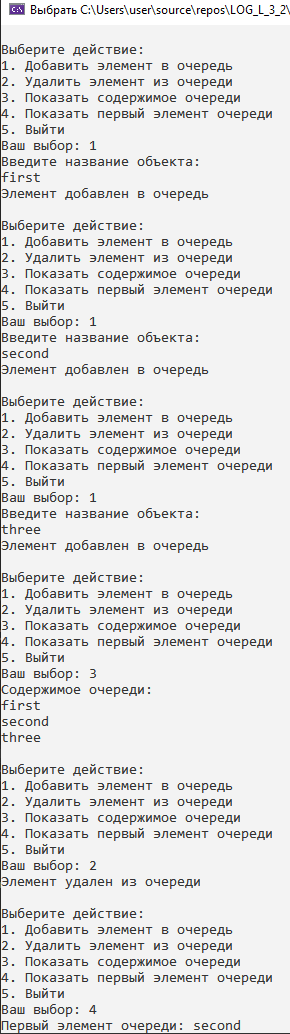


Рисунок 3 – результат работы программы №2

**\*Задание 3**

**Описание программы**

Программа реализует структуру данных «Стек» с использованием связного списка. Она демонстрирует базовую реализацию стека и позволяет пользователю взаимодействовать с ним через консольный интерфейс. Программа работает следующим образом:

1. Определяется структура node, которая представляет элемент стека. Она содержит строку inf для хранения данных и указатель next на следующий элемент.
2. Глобальные переменные head (указатель на вершину стека) и size (размер стека) инициализируются.
3. Функция get\_struct() создает новый узел, запрашивает у пользователя данные и возвращает указатель на созданный узел.
4. Функция push() добавляет новый элемент на вершину стека.
5. Функция pop() удаляет верхний элемент стека.
6. Функция display() выводит содержимое стека.
7. Функция front() возвращает указатель на верхний элемент стека.
8. Функция is\_empty() проверяет, пуст ли стек.
9. В функции main() реализовано меню, которое позволяет пользователю выбирать действия:
   * Добавить элемент в стек
   * Удалить элемент из стека
   * Показать содержимое стека
   * Показать верхний элемент стека
   * Выйти из программы
10. Программа работает в бесконечном цикле, пока пользователь не выберет опцию выхода.
11. Программа использует динамическое выделение памяти для создания новых узлов и освобождает память при удалении элементов.

Листинг программы смотрите в Приложении С.

**Результат работы программы**

На рисунке 4 приведён результат работы программы №3.

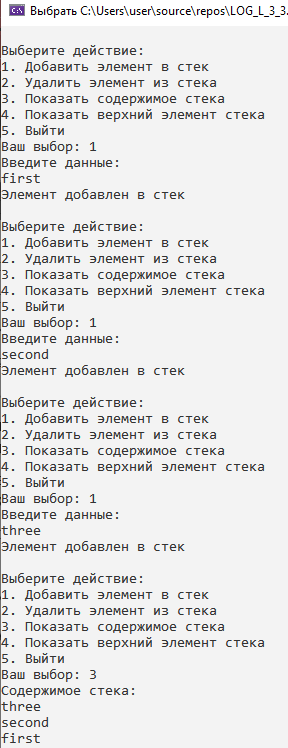
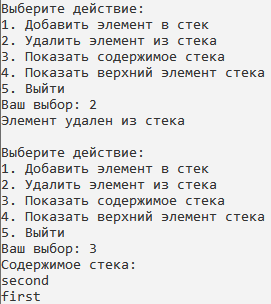


Рисунок 4 – результат работы программы №3

**Выводы**

Были изучены и практически реализованы основные структуры данных на базе динамических списков. Была написана программа, реализующая приоритетную очередь с использованием динамического списка, обеспечивая вставку элементов в соответствии с их приоритетом. В рамках дополнительного задания были написаны 2 программы, представляющие собой структуры данных «Очередь» и «Стек» на основе динамического списка, с использованием предоставленного кода.

**Приложение А**

**Листинг программы для задания 1**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

typedef struct Node {

int data;

int priority;

struct Node\* next;

} Node;

Node\* createNode(int data, int priority) {

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->data = data;

newNode->priority = priority;

newNode->next = NULL;

return newNode;

}

void enqueue(Node\*\* head, int data, int priority) {

Node\* newNode = createNode(data, priority);

if (\*head == NULL || priority > (\*head)->priority) {

newNode->next = \*head;

\*head = newNode;

}

else {

Node\* current = \*head;

while (current->next != NULL && current->next->priority >= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

}

int deleteByPriority(Node\*\* head) {

setlocale(LC\_ALL, "Russian");

if (\*head == NULL) {

printf(" Очередь пуста\n");

return -1;

}

Node\* temp = \*head;

int data = temp->data;

\*head = (\*head)->next;

free(temp);

return data;

}

void deleteByContent(Node\*\* head, int data) {

setlocale(LC\_ALL, "Russian");

if (\*head == NULL) {

printf(" Очередь пуста\n");

return;

}

if ((\*head)->data == data) {

Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

printf(" Элемент %d удален\n", data);

return;

}

Node\* current = \*head;

Node\* prev = NULL;

while (current != NULL && current->data != data) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf(" Элемент %d не найден\n", data);

return;

}

prev->next = current->next;

free(current);

printf(" Элемент %d удален\n", data);

}

void printList(Node\* head) {

setlocale(LC\_ALL, "Russian");

if (head == NULL) {

printf(" Список пуст\n");

return;

}

Node\* current = head;

while (current != NULL) {

printf("(Данные: %d, Приоритет: %d), ", current->data, current->priority);

current = current->next;

}

printf("\n");

}

int main() {

setlocale(LC\_ALL, "Russian");

Node\* head = NULL;

int choice, data, priority;

while (1) {

printf("\n Выберите действие:\n");

printf(" 1. Добавить элемент\n");

printf(" 2. Удалить элемент с наивысшим приоритетом\n");

printf(" 3. Удалить элемент по содержимому\n");

printf(" 4. Показать очередь\n");

printf(" 5. Выйти\n");

printf(" Ваш выбор: ");

scanf(" %d", &choice);

switch (choice) {

case 1:

printf(" Введите данные: ");

scanf(" %d", &data);

printf(" Введите приоритет: ");

scanf(" %d", &priority);

enqueue(&head, data, priority);

printf(" Элемент добавлен\n");

break;

case 2:

data = deleteByPriority(&head);

if (data != -1) {

printf(" Удален элемент: %d\n", data);

}

break;

case 3:

printf(" Введите данные элемента для удаления: ");

scanf(" %d", &data);

deleteByContent(&head, data);

break;

case 4:

printf(" Текущая очередь: ");

printList(head);

break;

case 5:

printf(" Программа завершила свою работу.\n");

exit(0);

default:

printf(" Неверный выбор. Попробуйте снова.\n");

}

}

return 0;

}

**Приложение B**

**Листинг программы для задания 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node {

char inf[256];

struct node\* next;

};

struct node\* head = NULL, \* tail = NULL;

int size = 0;

void enqueue(void);

void dequeue(void);

void display(void);

struct node\* front(void);

int is\_empty(void);

struct node\* get\_struct(void) {

setlocale(LC\_ALL, "Russian");

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

printf("Введите название объекта: \n");

scanf("%s", s);

if (\*s == 0)

{

printf("Запись не была произведена\n");

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p;

}

void enqueue(void) {

setlocale(LC\_ALL, "Russian");

struct node\* p = get\_struct();

if (p == NULL) return;

if (is\_empty()) {

head = tail = p;

}

else {

tail->next = p;

tail = p;

}

size++;

printf("Элемент добавлен в очередь\n");

}

void dequeue(void) {

setlocale(LC\_ALL, "Russian");

if (is\_empty())

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = head;

head = head->next;

free(temp);

size--;

if (size == 0)

{

tail = NULL;

}

printf("Элемент удален из очереди\n");

}

void display(void) {

setlocale(LC\_ALL, "Russian");

if (is\_empty())

{

printf("Очередь пуста\n");

return;

}

struct node\* current = head;

printf("Содержимое очереди:\n");

while (current)

{

printf("%s\n", current->inf);

current = current->next;

}

}

struct node\* front(void)

{

setlocale(LC\_ALL, "Russian");

if (is\_empty())

{

printf("Очередь пуста\n");

return NULL;

}

return head;

}

int is\_empty(void)

{

return size == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

int choice;

while (1)

{

printf("\nВыберите действие:\n");

printf("1. Добавить элемент в очередь\n");

printf("2. Удалить элемент из очереди\n");

printf("3. Показать содержимое очереди\n");

printf("4. Показать первый элемент очереди\n");

printf("5. Выйти\n");

printf("Ваш выбор: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

{

struct node\* first = front();

if (first)

printf("Первый элемент очереди: %s\n", first->inf);

}

break;

case 5:

exit(0);

default:

printf("Неверный выбор. Попробуйте снова.\n");

}

}

return 0;

}

**Приложение С**

**Листинг программы для задания 3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node {

char inf[256];

struct node\* next;

};

struct node\* head = NULL;

int size = 0;

void push(void);

void pop(void);

void display(void);

struct node\* front(void);

int is\_empty(void);

struct node\* get\_struct(void) {

setlocale(LC\_ALL, "Russian");

struct node\* p = NULL;

char s[256];

if ((p = (struct node\*)malloc(sizeof(struct node))) == NULL)

{

printf("Ошибка при распределении памяти\n");

exit(1);

}

printf("Введите данные: \n");

scanf("%s", s);

if (\*s == 0)

{

printf("Запись не была произведена\n");

return NULL;

}

strcpy(p->inf, s);

p->next = head;

return p;

}

void push(void) {

setlocale(LC\_ALL, "Russian");

struct node\* p = get\_struct();

head = p;

if (p == NULL) return;

if (is\_empty())

{

head = p;

}

size++;

printf("Элемент добавлен в стек\n");

}

void pop(void) {

setlocale(LC\_ALL, "Russian");

if (is\_empty()) {

printf("Стек пуст\n");

return;

}

struct node\* temp = head;

head = head->next;

free(temp);

size--;

if (size == 0) {

head = NULL;

}

printf("Элемент удален из стека\n");

}

void display(void) {

setlocale(LC\_ALL, "Russian");

if (is\_empty()) {

printf("Стек пуст\n");

return;

}

struct node\* current = head;

printf("Содержимое стека:\n");

while (current)

{

printf("%s\n", current->inf);

current = current->next;

}

}

struct node\* front(void)

{

setlocale(LC\_ALL, "Russian");

if (is\_empty())

{

printf("Стек пуст\n");

return NULL;

}

return head;

}

int is\_empty(void) {

return size == 0;

}

int main() {

setlocale(LC\_ALL, "Russian");

int choice;

while (1) {

printf("\nВыберите действие:\n");

printf("1. Добавить элемент в стек\n");

printf("2. Удалить элемент из стека\n");

printf("3. Показать содержимое стека\n");

printf("4. Показать верхний элемент стека\n");

printf("5. Выйти\n");

printf("Ваш выбор: ");

if (!scanf("%d", &choice)) return 1;

switch (choice)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

{

struct node\* first = front();

if (first)

printf("Верхний элемент стека: %s\n", first->inf);

}

break;

case 5:

exit(0);

default:

printf("Неверный выбор. Попробуйте снова.\n");

}

}

return 0;

}