Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 3

по дисциплине: "Логика и ОА в ИЗ"

на тему: "Динамические списки"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза, 2023

**Общие сведения.**

**Цель:** Научиться создавать Приоритетную Очередь. Научиться создавать структуру данных – Стек. Научиться создавать структуру данных – Очередь.

Список представляет собой последовательность элементов определенного типа. Простейший тип списка – линейный, когда для каждого из элементов, кроме последнего, имеется следующий, и для каждого, кроме первого имеется предыдущий элемент.

Возможна реализация списков посредством массивов или динамическая реализация.

Динамические списки относятся к динамическим структурам и используются, когда размер данных заранее неизвестен. Созданием динамических данных должна заниматься сама программа во время своего исполнения, этим достигается эффективное распределение памяти, но снижается эффективность доступа к элементам.

Динамические структуры данных отличаются от статических двумя основными свойствами:

1) в них нельзя обеспечить хранение в заголовке всей информации о структуре, поэтому каждый элемент должен содержать информацию, логически связывающую его с другими элементами структуры;

2) для них зачастую не удобно использовать единый массив смежных элементов памяти, поэтому необходимо предусматривать ту или иную схему динамического управления памятью.

Для обращения к динамическим данным применяют указатели.

Набор операций над списком будет включать добавление и удаление элементов, поиск элементов списка.

Различают односвязные, двусвязные и циклические списки.

В простейшем случае каждый элемент содержит всего одну ссылку на следующий элемент, такой список называется односвязным.

В простейшем случае для создания элемента списка используется структура, в которой объединяются полезная информация и ссылка на следующий элемент списка:

struct node

{

char inf[256]; // полезная информация

struct node \*next; // ссылка на следующий элемент

};

Обращение к списку и его элементам осуществляется посредством указателей:

struct node \*head = NULL, \*last = NULL, \*f = NULL; // указатели на первый и последний элементы списка

int dlinna = 0;

Для списка реализованы функции создания, добавления, удаления элемента, просмотра списка, нахождения нужного элемента списка:

// Функции добавления элемента, просмотра списка

void spstore(void), review(void), del(char \*name);

char find\_el[256];

struct node \*find(char \*name); // функция нахождения элемента

struct node \*get\_struct(void); // функция создания элемента

struct node \*get\_struct(void)

{

struct node \*p = NULL;

char s[256];

if ((p = (node\*)malloc(sizeof(struct node))) == NULL) // выделяем память под новый элемент списка

{

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

exit(1);

}

printf("Введите название объекта: \n"); // вводим данные

scanf("%s", s);

if (\*s == 0)

{

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

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента (в конец)\*/

void spstore(void)

{

struct node \*p = NULL;

p = get\_struct();

if (head == NULL && p != NULL) // если списка нет, то устанавливаем голову списка

{

head = p;

last = p;

}

else if (head != NULL && p != NULL) // список уже есть, то вставляем в конец

{

last->next = p;

last = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node \*struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

return;

}

/\* Поиск элемента по содержимому. \*/

struct node \*find(char \*name)

{

struct node \*struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

if (strcmp(name, struc->inf) == 0)

{

return struc;

}

struc = struc->next;

}

printf("Элемент не найден\n");

return NULL;

}

/\* Удаление элемента по содержимому. \*/

void del(char \*name)

{

struct node \*struc = head; // указатель, проходящий по списку установлен на начало списка

struct node \*prev;// указатель на предшествующий удаляемому элемент

int flag = 0; // индикатор отсутствия удаляемого элемента в списке

if (head == NULL) // если голова списка равна NULL, то список пуст

{

printf("Список пуст\n");

return;

}

if (strcmp(name, struc->inf) == 0) // если удаляемый элемент - первый

{

flag = 1;

head = struc->next; // установливаем голову на следующий элемент

free(struc); // удаляем первый элемент

struc = head; // устанавливаем указатель для продолжения поиска

}

else

{

prev = struc;

struc = struc->next;

}

while (struc) // проход по списку и поиск удаляемого элемента

{

if (strcmp(name, struc->inf) == 0) // если нашли, то

{

flag = 1; // выставляем индикатор

if (struc->next) // если найденный элемент не последний в списке

{

prev->next = struc->next; // меняем указатели

free(struc); // удаляем элемент

struc = prev->next; // устанавливаем указатель для продолжения поиска

}

else // если найденный элемент последний в списке

{

prev->next = NULL; // обнуляем указатель предшествующего элемента

free(struc); // удаляем элемент

return;

}

}

else / если не нашли, то

{

prev = struc; // устанавливаем указатели для продолжения поиска

struc = struc->next;

}

}

if (flag == 0) // если флаг = 0, значит нужный элемент не найден

{

printf("Элемент не найден\n");

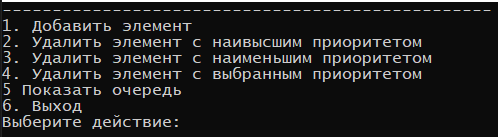
return;

}

}

**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).



На фото представлено меню программы, реализующей приоритетную очередь.

В данной программе можно: [1] Добавить элемент; [2] Удалить элемент с наивысшим приоритетом; [3] Удалить элемент с наименьшим приоритетом; [4] Удалить элемент с выбранным приоритетом; [5] Показать очередь; [6] Выйти.

Попробуем добавить пару элементов:





Как можно увидеть. Программа выводит числа, в зависимости от их приоритета.

Попробуем [2] Удалить элемент с наивысшим приоритетом:



Программа удалила число 22 с приоритетом 11.

Попробуем [3] Удалить элемент с наименьшим приоритетом:



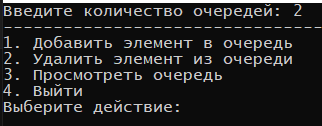
Программа удалила число 22 с приоритетом -5.

Попробуем [4] Удалить элемент с выбранным приоритетом = 10:

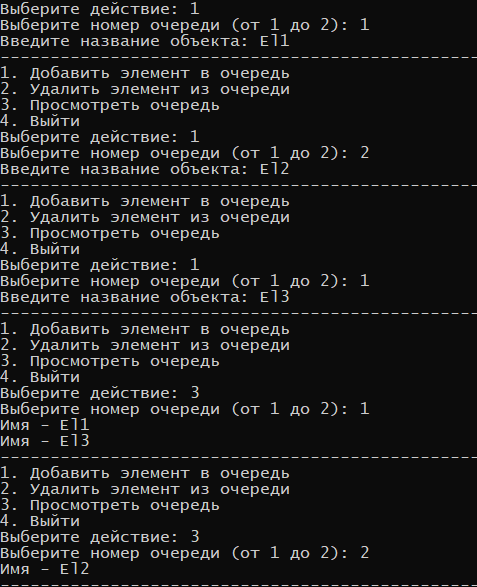


Программа удалила число 22 с приоритетом 10.

1. \* На основе приведенного кода реализуйте структуру данных *Очередь*.

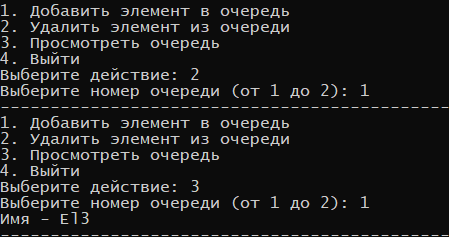


На фото представлено меню программы, реализующей структуру данных *Очередь*. Изначально можно выбрать количество очередей, после добавлять элементы в очереди и удалять их; Смотреть очередь.



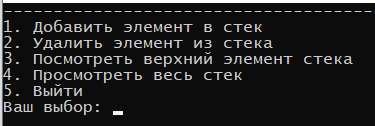
Мы добавили элементы El1 и El3 в 1 очередь и элемент El2 во вторую очередь и вывели на экран.

После мы удалили элемент из 1 очереди, и как можно заметить, удалился Элемент El1 – Структура работает правильно.

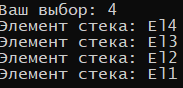


1. \* На основе приведенного кода реализуйте структуру данных *Стек*.

На фото представлено меню программы, реализующей структуру данных *Очередь*. Можно: Добавить элемент в стек; Удалить элемент из стека; Посмотреть верхний элемент стека; Посмотреть весь стек; Выйти.



Добавим пару элементов в стек.



Как можно заметить – Последний введенный элемент El4 находится в самом начале стека.

Попробуем удалить элемент из стека.





Как можно заметить – удалился последний элемент стека El4. Структура реализована правильно.**Вывод:**

Научились создавать Приоритетную Очередь. Научились создавать структуру данных – Стек. Научились создавать структуру данных – Очередь.

**Листинг**

**Приложение А (Код программы 1)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio>

#include <locale.h>

// Структура Node для хранения элементов очереди

struct Node {

int data; // Данные

int priority; // Приоритет

Node\* next; // Указатель на следующий элемент в очереди

};

// Класс PriorityQueue для реализации очереди с приоритетом

class PriorityQueue {

private:

Node\* head; // Указатель на голову (самый высокий приоритет)

public:

// Конструктор для инициализации head как nullptr

PriorityQueue() : head(nullptr) {}

// Функция добавления элемента в очередь

void push(int data, int priority) {

Node\* temp = new Node(); // Создание нового узла

temp->data = data; // Заполнение данных

temp->priority = priority; // Заполнение приоритета

temp->next = nullptr; // Установка следующего указателя как nullptr

// Вставка нового узла в соответствующее место по приоритету

if (head == nullptr || head->priority < priority) {

temp->next = head;

head = temp;

}

else {

Node\* current = head; // Текущий узел для итерации

// Ищем подходящее место для вставки

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

current = current->next;

}

// Вставляем новый узел

temp->next = current->next;

current->next = temp;

}

}

// Удаление элемента с наивысшим приоритетом

int popHighestPriority() {

if (head == nullptr) {

printf("Очередь пуста.\n");

return -1;

}

Node\* temp = head; // Временный узел для хранения головы

head = head->next; // Перемещение головы на следующий узел

int data = temp->data; // Извлечение данных из узла

delete temp; // Удаление временного узла

return data; // Возврат данных

}

// Удаление элемента с наименьшим приоритетом

int popLowestPriority() {

if (head == nullptr) {

printf("Очередь пуста.\n");

return -1;

}

Node\* current = head; // Текущий узел для итерации

Node\* prev = nullptr; // Предыдущий узел для итерации

// Итерация до последнего узла

while (current->next != nullptr) {

prev = current;

current = current->next;

}

// Удаление последнего узла

if (prev == nullptr) {

head = nullptr;

}

else {

prev->next = nullptr;

}

int data = current->data; // Извлечение данных из узла

delete current; // Удаление узла

return data; // Возврат данных

}

// Удаление элемента с заданным приоритетом

int popSpecificPriority(int targetPriority) {

Node\* current = head; // Текущий узел для итерации

Node\* prev = nullptr; // Предыдущий узел для итерации

// Итерация по всем узлам для поиска узла с заданным приоритетом

while (current != nullptr) {

if (current->priority == targetPriority) {

if (prev == nullptr) {

head = current->next;

}

else {

prev->next = current->next;

}

int data = current->data;

delete current;

return data;

}

prev = current;

current = current->next;

}

printf("Элемент с приоритетом %d не найден.\n", targetPriority);

return -1;

}

// Отображение всех элементов очереди

void display() {

Node\* current = head; // Текущий узел для итерации

while (current != nullptr) {

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

current = current->next;

}

}

};

int main() {

setlocale(LC\_ALL, "RUS");

PriorityQueue pq; // Объект очереди с приоритетом

int choice; // Выбор пользователя

int data; // Данные для добавления или удаления

int priority; // Приоритет для добавления

// Основной цикл

while (true) {

printf("-------------------------------------------------\n");

printf("1. Добавить элемент\n2. Удалить элемент с наивысшим приоритетом\n3. Удалить элемент с наименьшим приоритетом\n4. Удалить элемент с выбранным приоритетом\n5 Показать очередь\n6. Выход\n");

printf("Выберите действие: ");

scanf("%d", &choice);

// Обработка выбора пользователя

switch (choice) {

case 1:

printf("Введите число: ");

scanf("%d", &data);

printf("Введите приоритет: ");

scanf("%d", &priority);

pq.push(data, priority);

break;

case 2:

data = pq.popHighestPriority();

if (data != -1) {

printf("Удаленный элемент с наивысшим приоритетом: %d\n", data);

}

break;

case 3:

data = pq.popLowestPriority();

if (data != -1) {

printf("Удаленный элемент с наименьшим приоритетом: %d\n", data);

}

break;

case 4:

printf("Введите приоритет: ");

scanf("%d", &priority);

data = pq.popSpecificPriority(priority);

if (data != -1) {

printf("Удаленный элемент с заданным приоритетом: %d\n", data);

}

break;

case 5:

pq.display();

break;

case 6:

return 0;

default:

printf("Неверный выбор. Пожалуйста, попробуйте снова.\n");

}

}

return 0;

}

**Приложение Б (Код программы 2)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

// Определение структуры для хранения элементов очереди

struct Node {

char inf[256]; // Информационное поле (название объекта)

struct Node\* next; // Указатель на следующий элемент в очереди

};

int main() {

setlocale(LC\_ALL, "RUS"); // Установка русской локализации для консоли

int numQueues; // Количество очередей

printf("Введите количество очередей: ");

scanf("%d", &numQueues);

// Выделение памяти для массива указателей на головы очередей

struct Node\*\* heads = (struct Node\*\*)malloc(numQueues \* sizeof(struct Node\*));

// Выделение памяти для массива указателей на хвосты очередей

struct Node\*\* lasts = (struct Node\*\*)malloc(numQueues \* sizeof(struct Node\*));

// Инициализация указателей на головы и хвосты всех очередей как NULL

for (int i = 0; i < numQueues; ++i) {

heads[i] = NULL;

lasts[i] = NULL;

}

// Главный цикл программы

while (1) {

int choice, queueNum; // choice для хранения выбранного действия, queueNum для номера очереди

printf("-------------------------------------------------\n");

printf("1. Добавить элемент в очередь\n2. Удалить элемент из очереди\n3. Просмотреть очередь\n4. Выйти\n");

printf("Выберите действие: ");

scanf("%d", &choice);

// Условие для выхода из цикла

if (choice == 4) break;

printf("Выберите номер очереди (от 1 до %d): ", numQueues);

scanf("%d", &queueNum);

queueNum--; // Переход к индексации с 0

// Проверка корректности номера очереди

if (queueNum < 0 || queueNum >= numQueues) {

printf("Неверный номер очереди.\n");

continue;

}

// Добавление элемента в очередь

if (choice == 1) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node)); // Выделение памяти для нового узла

printf("Введите название объекта: ");

scanf("%s", newNode->inf); // Считывание информации

newNode->next = NULL; // Установка указателя next как NULL

// Если очередь пуста, добавить элемент как первый и последний

if (heads[queueNum] == NULL && lasts[queueNum] == NULL) {

heads[queueNum] = lasts[queueNum] = newNode;

}

else {

// Если не пуста, добавить элемент в конец и обновить указатель на последний элемент

lasts[queueNum]->next = newNode;

lasts[queueNum] = newNode;

}

}

else if (choice == 2) {

// Удаление элемента из очереди

if (heads[queueNum] == NULL) { // Если очередь пуста, вывод сообщения

printf("Очередь пуста.\n");

continue;

}

struct Node\* temp = heads[queueNum]; // Временный указатель для удаления

heads[queueNum] = heads[queueNum]->next; // Сдвиг головы на следующий элемент

free(temp); // Освобождение памяти

}

else if (choice == 3) {

// Просмотр содержимого очереди

struct Node\* current = heads[queueNum]; // Начало с головы

if (current == NULL) { // Если очередь пуста

printf("Очередь пуста.\n");

continue;

}

// Проход по всей очереди и вывод каждого элемента

while (current) {

printf("Имя - %s\n", current->inf);

current = current->next;

}

}

}

// Освобождение памяти, выделенной для массивов heads и lasts

free(heads);

free(lasts);

return 0;

}

**Приложение С (Код программы 3)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <cstdio> // Подключение библиотеки для работы с вводом-выводом

#include <cstdlib> // Подключение библиотеки для работы с памятью и преобразованием типов

#include <cstring> // Подключение библиотеки для работы со строками

#include <locale.h> // Подключение библиотеки для работы с локалью

// Определение структуры Node для узлов стека

struct Node {

char inf[256]; // Массив для хранения данных

Node\* next; // Указатель на следующий узел стека

};

Node\* head = nullptr; // Указатель на вершину стека

// Функция для создания нового узла стека

Node\* createNode() {

Node\* newNode = new Node; // Выделяем память под новый узел

char s[256];

printf("Введите название элемента: ");

scanf("%s", s);

// Проверка на пустую строку

if (strlen(s) == 0) {

printf("Элемент не добавлен\n");

return nullptr;

}

// Копирование данных в новый узел

strcpy(newNode->inf, s);

newNode->next = nullptr;

return newNode;

}

// Функция для добавления узла на вершину стека

void push() {

Node\* newNode = createNode();

if (newNode == nullptr) return;

newNode->next = head;

head = newNode;

}

// Функция для удаления узла с вершины стека

void pop() {

if (head == nullptr) {

printf("Стек пуст\n");

return;

}

Node\* temp = head;

head = head->next;

delete temp;

}

// Функция для просмотра данных на вершине стека

void peek() {

if (head == nullptr) {

printf("Стек пуст\n");

return;

}

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

}

// Функция для просмотра всех узлов стека

void review() {

Node\* temp = head;

if (head == nullptr) {

printf("Стек пуст\n");

return;

}

while (temp) {

printf("Элемент стека: %s\n", temp->inf);

temp = temp->next;

}

}

// Главная функция программы

int main() {

setlocale(LC\_ALL, "RUS"); // Установка русской локали

int choice; // Переменная для хранения выбора пользователя

// Цикл для работы с меню

do {

printf("-------------------------------------------------\n");

printf("1. Добавить элемент в стек \n2. Удалить элемент из стека\n3. Посмотреть верхний элемент стека\n4. Просмотреть весь стек\n5. Выйти\n");

printf("Ваш выбор: ");

scanf("%d", &choice);

// Обработка выбора пользователя

switch (choice) {

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peek();

break;

case 4:

review();

break;

case 5:

printf("Выход\n");

break;

default:

printf("Неправильный выбор. Попробуйте еще раз.\n");

break;

}

} while (choice != 5); // Цикл продолжается, пока не выбрана опция "Выйти"

return 0;

}