Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Динамические списки»

Выполнили

студенты группы 22ВВВ1:

Ганин И.Р.

Курушин Я.С.

Приняли:

К.т.н, доцент Акифьев И. В.

К.т.н, доцент Юрова О. В.

**Цель работы**

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

**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.

**Теоретическая часть**

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 неудобно 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.

**Практическая часть**

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).

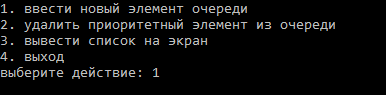


Рисунок 1.1 - интерфейс



Рисунок 1.2 – добавили элемент с приоритетом 1



Рисунок 1.3 – добавили элемент с приоритетом 5



Рисунок 1.4 – добавили элемент с приоритетом 3

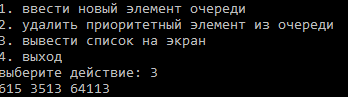


Рисунок 1.5 – вывели элементы с разным приоритетом

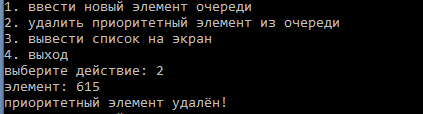


Рисунок 1.6 – удалили элемент с приоритетом 1

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

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

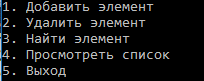


Рисунок 2.1 – интерфейс для работы с очередью

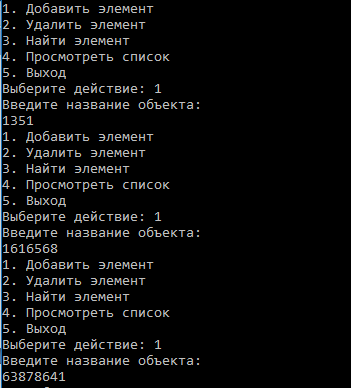


Рисунок 2.2 – добавили 3 элемента

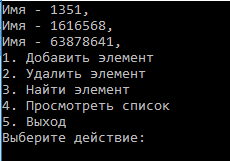


Рисунок 2.3 – вывели очередь

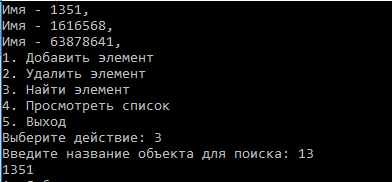


Рисунок 2.4– нашли элемент, содержащий «13»

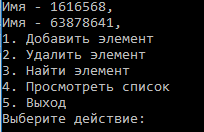


Рисунок 2.5 – удалили первый элемент

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

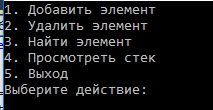


Рисунок 3.1 – интерфейс для работы со стеком

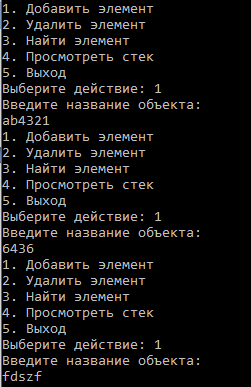


Рисунок 3.2 – добавили 3 элемента

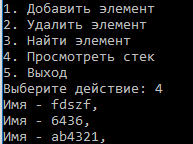


Рисунок 3.3 – посмотрели стек

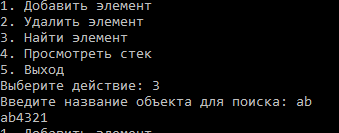


Рисунок 3.4 – нашли элемент, содержащий «ab»

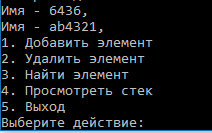


Рисунок 3.5 – удалили последний элемент

**Вывод:** Научились создавать Приоритетную Очередь. Научились создавать структуру данных – Стек. Научились создавать структуру данных – Очередь.

**Приложение А**

**Листинг**

Файл Source.cpp

#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));

if (newnode == NULL) {

printf("не удалось выделить память!\n");

return NULL;

}

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) {

\*head = newnode;

return;

}

if (priority < (\*head)->priority) {

newnode->next = \*head;

\*head = newnode;

return;

}

node\* current = \*head;

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

current = current->next;

}

newnode->next = current->next;

current->next = newnode;

}

int dequeue(node\*\* head) {

if (\*head == NULL) {

printf("список пуст!\n");

return -1;

}

node\* temp = \*head;

int data = temp->data;

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

free(temp);

return data;

}

void main() {

node\* head = NULL;

int choice, priority, data;

setlocale(LC\_ALL, "rus");

do {

printf("1. ввести новый элемент очереди\n");

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

printf("3. вывести список на экран\n");

printf("4. выход\n");

printf("выберите действие: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("введите информацию в элемент списка: ");

scanf("%d", &data);

printf("введите приоритет: ");

scanf("%d", &priority);

enqueue(&head, data, priority);

break;

case 2:

data = dequeue(&head);

printf("элемент: %d", data);

printf("\nприоритетный элемент удалён!\n");

break;

case 3:

while (head != NULL) {

data = dequeue(&head);

printf("%d ", data);

}

printf("\n");

break;

case 4:

system("cls");

printf("до свидания!\n");

break;

default:

system("cls");

printf("некоректный выбор!\n");

break;

}

} while (choice != 4);

}

Файл Source1.cpp

#define \_crt\_secure\_no\_warnings

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

typedef struct node {

char inf[256]; // информация

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

} node;

// функции добавления элемента, просмотра списка

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

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

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

node\* get\_struct(void) {

node\* p = null;

char s[256];

if ((p = (node\*)malloc(sizeof(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(node\*\* head, node\*\* last) {

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;

}

}

\* просмотр содержимого списка. \*/

void review(node\*\* head) {

node\* struc = \*head;

if (\*head == null)

{

printf("список пуст\n");

}

while (struc != null)

{

printf("имя - %s, \n", struc->inf);

struc = struc->next;

}

}

\* поиск элемента по содержимому. \*/

node\* find(node\*\* head, char\* name) {

node\* struc = \*head;

int k = 0;

if (head == null)

{

printf("список пуст\n");

}

while (struc)

{

if (strstr(struc->inf, name) != null) {

printf("%s ", struc->inf);

k++;

}

struc = struc->next;

}

if (k == 0) {

printf("ни одного подходящего элемеента не найдено.");

}

printf("\n");

return null;

}

// удаление элемента из начала очереди

void dequeue(node\*\* head) {

if (\*head == null) {

printf("очередь пуста\n");

return;

}

node\* temp = \*head;

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

free(temp);

}

void main() {

node\* head = null, \*last = null, \*result = null; // указатели на первый и последний элементы списка

int choice;

char name[256];

setlocale(lc\_all, "rus");

do {

printf("1. добавить элемент\n");

printf("2. удалить элемент\n");

printf("3. найти элемент\n");

printf("4. просмотреть список\n");

printf("5. выход\n");

printf("выберите действие: ");

scanf("%d", &choice);

switch (choice) {

case 1:

spstore(&head, &last);

break;

case 2:

dequeue(&head);

printf("первый элемент удалён!\n");

break;

case 3:

printf("введите название объекта для поиска: ");

scanf("%s", &name);

find(&head, name);

break;

case 4:

system("cls");

review(&head);

break;

case 5:

system("cls");

printf("до свидание!\n");

break;

default:

system("cls");

printf("некоректный выбор!");

break;

}

} while (choice != 5);

}

Файл Source2.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

typedef struct node {

char inf[256]; // информация

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

} node;

// Функции добавления элемента, просмотра стека

void push(node\*\* top), review(node\*\* top), pop(node\*\* top);

node\* get\_struct(void) {

node\* p = NULL;

char s[256];

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

{

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

}

/\* Добавление элемента в стек \*/

void push(node\*\* top) {

node\* p = NULL;

p = get\_struct();

if (p != NULL) {

p->next = \*top;

\*top = p;

}

}

/\* Просмотр содержимого стека \*/

void review(node\*\* top) {

node\* struc = \*top;

if (\*top == NULL)

{

printf("Стек пуст\n");

}

while (struc != NULL)

{

printf("Имя - %s, \n", struc->inf);

struc = struc->next;

}

}

/\* Поиск элемента по содержимому. \*/

node\* find(node\*\* top, char\* name) {

node\* struc = \*top;

int k = 0;

if (\*top == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

if (strstr(struc->inf, name) != NULL) {

printf("%s ", struc->inf);

k++;

}

struc = struc->next;

}

if (k == 0) {

printf("Ни одного подходящего элемеента не найдено.");

}

printf("\n");

return NULL;

}

/\* Удаление элемента из стека \*/

void pop(node\*\* top) {

if (\*top == NULL) {

printf("Стек пуст\n");

return;

}

node\* temp = \*top;

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

free(temp);

}

void main() {

node\* top = NULL; // указатель на верхний элемент стека

int choice;

char name[256];

setlocale(LC\_ALL, "RUS");

do {

printf("1. Добавить элемент\n");

printf("2. Удалить элемент\n");

printf("3. Найти элемент\n");

printf("4. Просмотреть стек\n");

printf("5. Выход\n");

printf("Выберите действие: ");

scanf("%d", &choice);

switch (choice) {

case 1:

push(&top);

break;

case 2:

pop(&top);

printf("Верхний элемент удалён!\n");

break;

case 3:

printf("Введите название объекта для поиска: ");

scanf("%s", &name);

find(&top, name);

break;

case 4:

// system("cls");

review(&top);

break;

case 5:

system("cls");

printf("До свидания!\n");

break;

default:

system("cls");

printf("Некоректный выбор!");

break;

}

} while (choice != 5);

}