 Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему «Динамические списки»

Выполнили:

студенты группы 22ВВВ2

Перфилов А.В.

Приняли:

доцент, профессор Митрохин М.А.

к.э.н., доцент Акифьев И.В.

Пенза 2023

**Название**

Динамические списки

**Цель работы**

Научиться реализовывать динамические списки

**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*.

**Листинг**

**Задание 1:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale>

struct node {

char inf[256];

int priority;

struct node\* next;

};

struct node\* head = NULL;

struct node\* get\_struct(int priority)

{

struct node\* p = NULL;

char s[256];

p = (node\*)malloc(sizeof(struct node));

if ((p == 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;

}

p->priority = priority;

strcpy(p->inf, s);

p->next = NULL;

return p; // возвращаем указатель на созданный элемент

}

/\* Последовательное добавление в список элемента \*/

void spstore(int priority)

{

struct node\* p = NULL;

p = get\_struct(priority);

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

p->next = head;

head = p;

}

else {

struct node\* curr = head;

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

curr = curr->next;

}

p->next = curr->next;

curr->next = p;

}

return;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

}

while (struc)

{

printf("Приоритет: %d Имя: %s, \n",struc->priority, 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

return;

}

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;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

int pr;

for (int i = 0; i < 5; i++) {

printf("Введите приоритет элемента:\n");

scanf("%d", &pr);

spstore(pr);

}

review();

char s[256] = {1};

printf("Введите удаляемый элемент: \n");

scanf("%s", s);

del(s);

review();

return 0;

}}

**Задание 2:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale>

struct node {

char inf[256];

struct node\* next;

};

struct node\* head = NULL;

struct node\* get\_struct()

{

struct node\* p = NULL;

char s[256];

p = (node\*)malloc(sizeof(struct node));

if ((p == 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

}

/\* добавление в список элемента \*/

void spstore()

{

struct node\* p = NULL;

p = get\_struct();

if (head == NULL) {

p->next = head;

head = p;

}

else {

struct node\* curr = head;

head = p;

p->next = curr;

}

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;

}

/\* Удаление элемента по содержимому. \*/

void del()

{

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

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

{

printf("Список пуст\n");

return;

}

else

{

head = head->next;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

for (int i = 0; i < 7; i++) {

spstore();

}

printf("Перед удалением\n");

review();

del();

printf("После удаления\n");

review();

return 0;

}

**Задание 3:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale>

struct node {

char inf[256];

struct node\* next;

};

struct node\* head = NULL;

struct node\* last = NULL;

struct node\* get\_struct()

{

struct node\* p = NULL;

char s[256];

p = (node\*)malloc(sizeof(struct node));

if ((p == 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

}

/\* добавление в список элемента \*/

void spstore()

{

struct node\* p = NULL;

p = get\_struct();

if (head == NULL) {

head = p;

last = head;

}

else {

struct node\* curr = head;

head = p;

p->next = curr;

}

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;

}

/\* Удаление элемента по содержимому. \*/

void del()

{

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

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

{

printf("Список пуст\n");

return;

}

else

{

struct node\* curr = head;

while (curr->next->next != NULL) {

curr = curr->next;

}

curr->next = NULL;

last = curr;

}

}

int main() {

setlocale(LC\_ALL, "Rus");

for (int i = 0; i < 7; i++) {

spstore();

}

printf("Перед удалением\n");

review();

del();

printf("После удаления\n");

review();

return 0;

}

**Результат работы программы**

Результат работы программы показан на рисунках 1, 2 и 3.

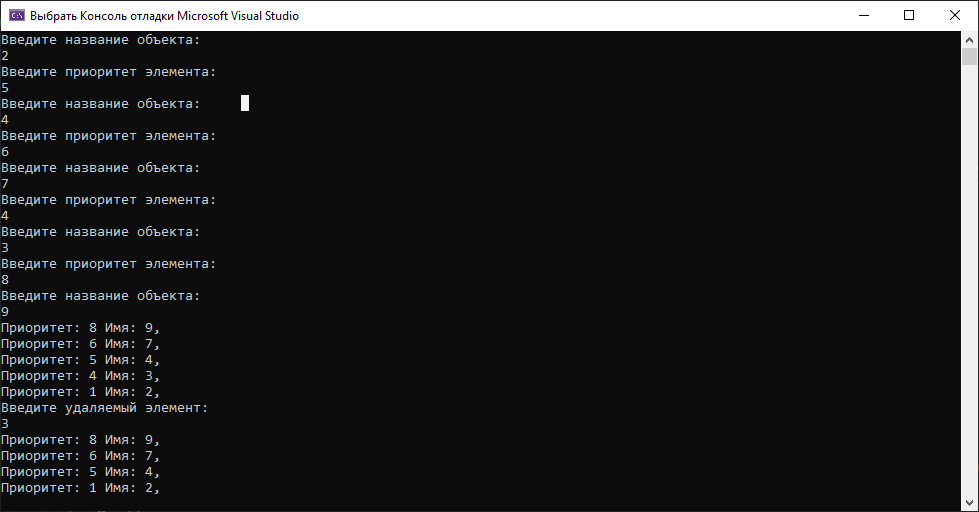


Рис. 1



Рис.2

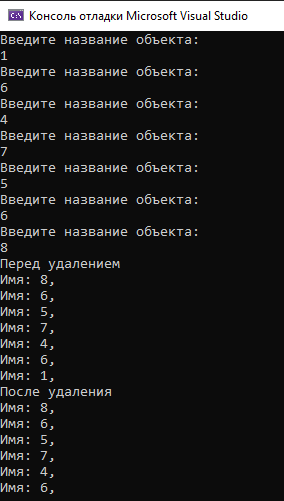


Рис.3

**Вывод:**

В ходе выполнения лабораторной работы мы научились оценивать время выполнения программы и анализировать алгоритмы сортировки.

Ссылка на репозиторий: https://github.com/chif45/laba3\_logica