Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №3  
по дисциплине: «Динамические списки»

Выполнили студенты группы 22ВВВ2:  
Демин М. С.

Сергунов М. Р.

Амиров И. Р.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Динамические списки

**Цель работы**

Разобраться с основными моментами при работе с динамическими списками

**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**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

typedef struct node {

char inf[256]; // полезная информация

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

} node;

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

int dlinna = 0;

// 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 = calloc(1, sizeof(struct node))) == NULL) {

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;

}

void spstore2(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 начало

p->next = head;

head = 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 = NULL; // 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;

}

}

/\*

\* 3333

\* 3

\* 3

\* 3333

\* 3

\* 3

\* 3333

\*/

typedef struct PQobj {

node\* inf; // полезная информация

int prior;

} PQobj;

void qp() {

printf("нечестная очередь:\n");

int i2 = 15;

int q\_size = 0;

PQobj\* q = NULL;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

q\_size++;

q = realloc(q, sizeof(PQobj) \* q\_size);

node\* inf = get\_struct();

int prior = 0;

printf("Введите приоритет:\n");

scanf("%d", &prior);

q[q\_size - 1].inf = inf;

q[q\_size - 1].prior = prior;

for (int i3 = 0; i3 < q\_size; i3++) {

for (int i4 = 0; i4 < q\_size; i4++) {

if (i3 == i4)

continue;

PQobj n1 = q[i3];

PQobj n2 = q[i4];

if (n1.prior > n2.prior) {

q[i3] = n2;

q[i4] = n1;

}

}

}

}

else if (act == 2) {

for (int i3 = 0; i3 < q\_size; i3++) {

PQobj n1 = q[i3];

printf("%s -- приоритет %d\n", n1.inf->inf, n1.prior);

}

}

else if (act == 3) {

q\_size--;

memmove(&q[0], &q[1], sizeof(q) \* q\_size);

}

else {

printf("нет такого действия\n");

}

}

for (int i = 0; i < q\_size; i++) {

free(q[i].inf);

}

free(q);

}

void q() {

printf("очередь:\n");

int i2 = 15;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

spstore();

}

else if (act == 2) {

review();

}

else if (act == 3) {

if (head) {

printf("Удален: %s\n", head->inf);

del(head->inf);

}

else {

printf("Нечего удалять\n");

}

}

else {

printf("нет такого действия\n");

}

}

printf("\n");

}

void stack() {

printf("стек:\n");

int i2 = 15;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

spstore2();

}

else if (act == 2) {

review();

}

else if (act == 3) {

if (head) {

printf("Удален: %s\n", head->inf);

del(head->inf);

}

else {

printf("Нечего удалять\n");

}

}

else {

printf("нет такого действия\n");

}

}

printf("\n");

}

int main() {

setlocale(LC\_ALL, "rus");

setbuf(stdout, NULL);

qp();

q();

stack();

}

**Задание 1:**

Реализовали приоритетную очередь, путём добавления элемента в список в соответствии с приоритетом объекта. Сделали отдельной функцией

void qp() {

printf("нечестная очередь:\n");

int i2 = 15;

int q\_size = 0;

PQobj\* q = NULL;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

q\_size++;

q = realloc(q, sizeof(PQobj) \* q\_size);

node\* inf = get\_struct();

int prior = 0;

printf("Введите приоритет:\n");

scanf("%d", &prior);

q[q\_size - 1].inf = inf;

q[q\_size - 1].prior = prior;

for (int i3 = 0; i3 < q\_size; i3++) {

for (int i4 = 0; i4 < q\_size; i4++) {

if (i3 == i4)

continue;

PQobj n1 = q[i3];

PQobj n2 = q[i4];

if (n1.prior > n2.prior) {

q[i3] = n2;

q[i4] = n1;

}

}

}

}

else if (act == 2) {

for (int i3 = 0; i3 < q\_size; i3++) {

PQobj n1 = q[i3];

printf("%s -- приоритет %d\n", n1.inf->inf, n1.prior);

}

}

else if (act == 3) {

q\_size--;

memmove(&q[0], &q[1], sizeof(q) \* q\_size);

}

else {

printf("нет такого действия\n");

}

}

for (int i = 0; i < q\_size; i++) {

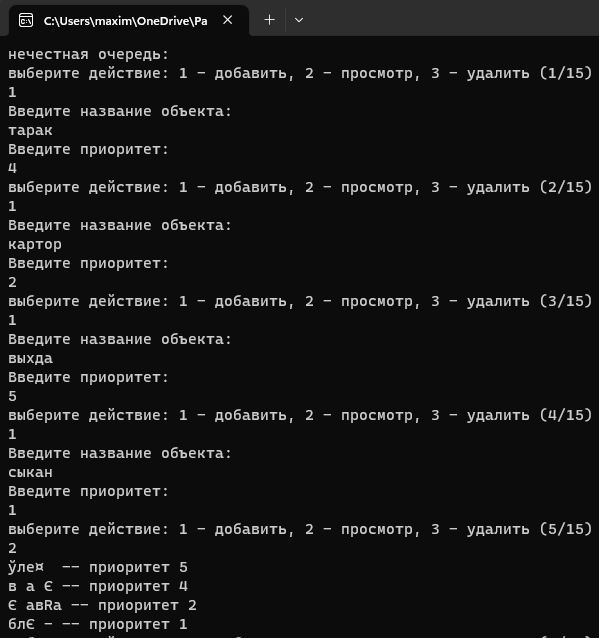
free(q[i].inf);

}

free(q);

}

Всё это в консоли выглядит так:



Задание 2. На основе приведенного кода реализовали структуру данных очередь в отдельной функции:

void q() {

printf("очередь:\n");

int i2 = 15;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

spstore();

}

else if (act == 2) {

review();

}

else if (act == 3) {

if (head) {

printf("Удален: %s\n", head->inf);

del(head->inf);

}

else {

printf("Нечего удалять\n");

}

}

else {

printf("нет такого действия\n");

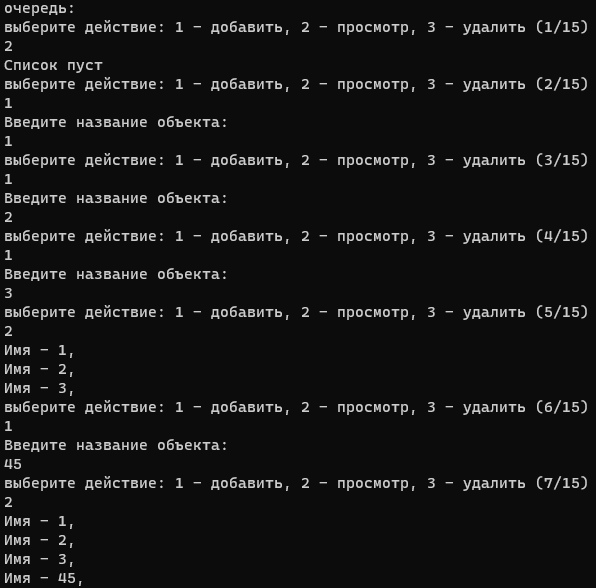
}

}

printf("\n");

}

В консоли выглядит:



Задание 3. На основе приведенного кода реализовали структуру данных стек:

void stack() {

printf("стек:\n");

int i2 = 15;

for (int i = 1; i <= i2; i++) {

printf("выберите действие: 1 - добавить, 2 - просмотр, 3 - удалить (%d/%d)\n", i, i2);

int act = 0;

scanf("%d", &act);

if (act == 1) {

spstore2();

}

else if (act == 2) {

review();

}

else if (act == 3) {

if (head) {

printf("Удален: %s\n", head->inf);

del(head->inf);

}

else {

printf("Нечего удалять\n");

}

}

else {

printf("нет такого действия\n");

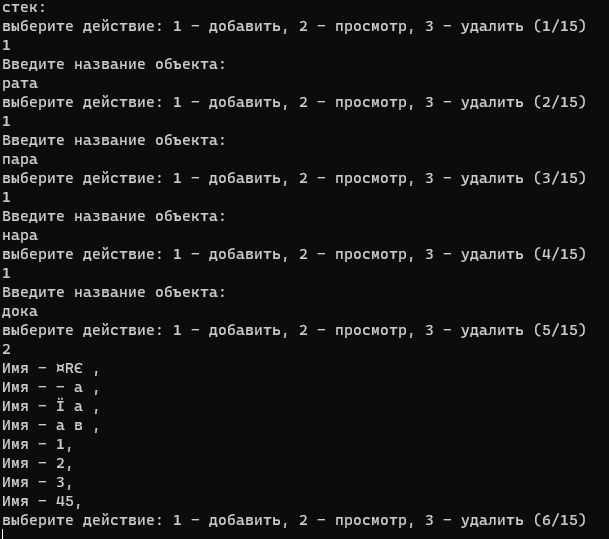
}

}

printf("\n");

}

Вывод в консоли:



**Вывод**

Разобрались с основными моментами при работе с динамическими списками