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»

Выполнил:

студент группы 23ВВВ4

Соснин Глеб

Проверил:

доцент, Юрова О. В.

Пенза, 2024

**Цель** – научиться создавать динамические списки, ознакомиться с функциями динамического списка.

**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).

**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

struct node

{

char inf[256];

int priority; // приоритет объекта

struct node\* next;

};

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

/\* Функция создания элемента \*/

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

int pr;

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);

printf("Введите приоритет объекта (целое число): \n");

scanf("%d", &pr);

if (\*s == 0)

{

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

free(p);

return NULL;

}

strcpy(p->inf, s);

p->priority = pr;

p->next = NULL;

return p;

}

/\* Функция добавления элемента в приоритетную очередь \*/

void spstore(void)

{

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

if (p == NULL)

return;

if (head == NULL)

{

head = p;

return;

}

if (p->priority > head->priority)

{

p->next = head;

head = p;

return;

}

struct node\* curr = head;

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

{

curr = curr->next;

}

p->next = curr->next;

curr->next = p;

}

/\* Просмотр содержимого списка. \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Список пуст\n");

return;

}

while (struc)

{

printf("Имя - %s, Приоритет - %d\n", struc->inf, struc->priority);

struc = struc->next;

}

}

/\* Удаление элемента с наивысшим приоритетом \*/

void del\_top\_priority(void)

{

if (head == NULL)

{

printf("Список пуст\n");

return;

}

struct node\* temp = head;

head = head->next;

free(temp);

printf("Элемент с наивысшим приоритетом удален.\n");

}

int main()

{

setlocale(LC\_ALL, "Russian");

int choice;

while (1)

{

printf("\n1. Добавить элемент\n2. Просмотреть очередь\n3. Удалить элемент с наивысшим приоритетом\n4. Выход\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

spstore();

break;

case 2:

review();

break;

case 3:

del\_top\_priority();

break;

case 4:

exit(0);

default:

printf("Неверный выбор.\n");

}

}

return 0;

}

Приоритетная очередь добавляет элементы в соответствии с приоритетом — объект с большим приоритетом становится перед объектом с меньшим приоритетом. Для этого нужно добавить поле приоритета в структуру *node* и изменить функцию добавления элемента так, чтобы она вставляла новый элемент на нужное место.

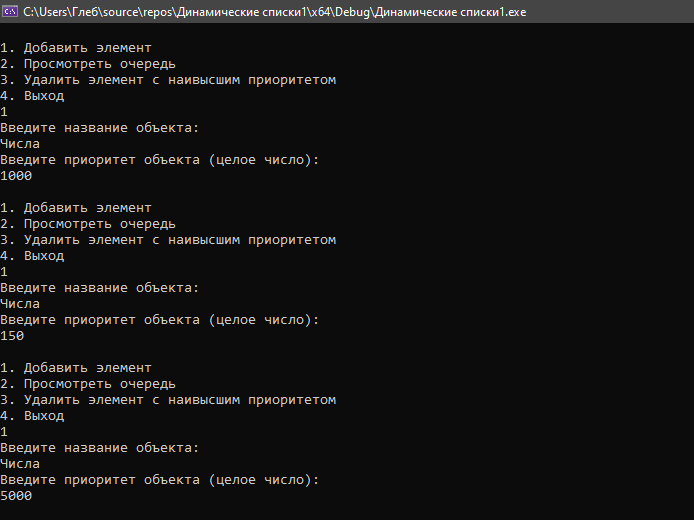


Рисунок 1 – Составление списка

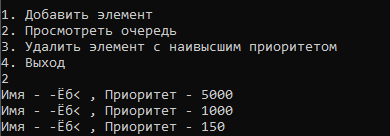


Рисунок 2 – Просмотр содержимого списка с приоритетной очередью

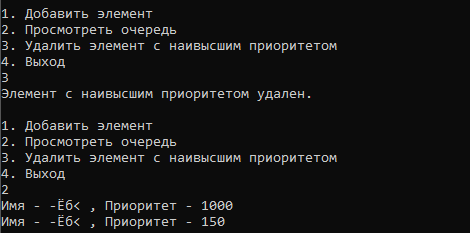


Рисунок 3 – Удаление элемента из списка

**Задание 2: \*** На основе приведенного кода реализуйте структуру данных *Очередь*.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <clocale>

// Определение структуры узла

struct node

{

char inf[256];

struct node\* next;

};

// Указатели на начало и конец очереди

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

struct node\* get\_struct(void)

{

struct node\* p = NULL;

char s[256];

if ((p = (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;

}

/\* Добавление элемента в конец очереди \*/

void enqueue(void)

{

struct node\* p = get\_struct(); // создаем новый элемент

if (head == NULL && p != NULL)

{

head = p;

last = p;

}

else if (head != NULL && p != NULL)

{

last->next = p;

last = p;

}

return;

}

/\* Удаление элемента из начала очереди \*/

void dequeue(void)

{

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

struct node\* temp = head; // временный указатель для удаления элемента

printf("Удаление элемента: %s\n", head->inf);

head = head->next;

free(temp);

if (head == NULL)

{

last = NULL;

}

}

/\* Просмотр содержимого очереди \*/

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Очередь пуста\n");

return;

}

while (struc)

{

printf("Имя - %s\n", struc->inf);

struc = struc->next;

}

}

int main(void)

{

setlocale(LC\_ALL, "Russian");

int choice;

char name[256];

while (1)

{

printf("\nМеню:\n");

printf("1. Добавить элемент \n");

printf("2. Удалить элемент \n");

printf("3. Просмотр очереди \n");

printf("4. Выйти\n");

printf("Выберите операцию: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

review();

break;

case 4:

exit(0);

default:

printf("Неверный выбор\n");

}

}

return 0;

}

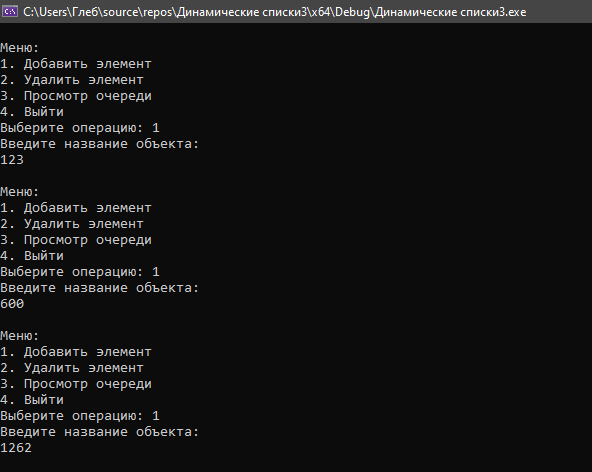


Рисунок 4 – Добавление элементов в Очередь

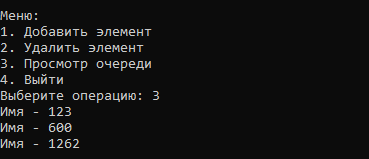


Рисунок 5 – Просмотр элементов в Очереди

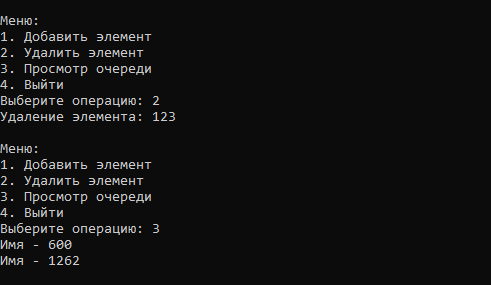


Рисунок 6 – Удаление элемента из Очереди

1. *Enqueue* (добавление элемента в конец):

Мы создаем новый элемент с помощью функции *get\_struct*() и добавляем его в конец очереди. Если очередь пуста (то есть *head* == *NULL*), то новый элемент становится как головой, так и концом очереди.

1. *Dequeue* (удаление элемента с начала):

Если очередь пуста, выводится сообщение "Очередь пуста". Иначе удаляется элемент с начала очереди (голова), а указатель *head* перемещается на следующий элемент.

1. *Review* (просмотр элементов очереди):

Проходим по очереди и выводим содержимое каждого элемента.

**Задание 3: \*** На основе приведенного кода реализуйте структуру данных *Стек*.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <clocale>

struct node

{

char inf[256];

struct node\* next;

};

struct node\* head = NULL;

/\* Функция создания элемента \*/

struct node\* get\_struct(void)

{

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");

free(p);

return NULL;

}

strcpy(p->inf, s);

p->next = NULL;

return p;

}

/\* Функция добавления элемента в стек \*/

void push(void)

{

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

if (p == NULL)

return;

p->next = head;

head = p;

}

void review(void)

{

struct node\* struc = head;

if (head == NULL)

{

printf("Стек пуст\n");

return;

}

while (struc)

{

printf("Имя - %s\n", struc->inf);

struc = struc->next;

}

}

/\* Функция удаления элемента из стека \*/

void pop(void)

{

if (head == NULL)

{

printf("Стек пуст\n");

return;

}

struct node\* temp = head;

head = head->next;

free(temp);

printf("Элемент удален\n");

}

int main()

{

setlocale(LC\_ALL, "Russian");

int choice;

while (1)

{

printf("\n1. Добавить элемент в стек \n2. Просмотреть стек\n3. Удалить элемент из стека\n4. Выход\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

push();

break;

case 2:

review();

break;

case 3:

pop();

break;

case 4:

exit(0);

default:

printf("Неверный выбор.\n");

}

}

return 0;

}

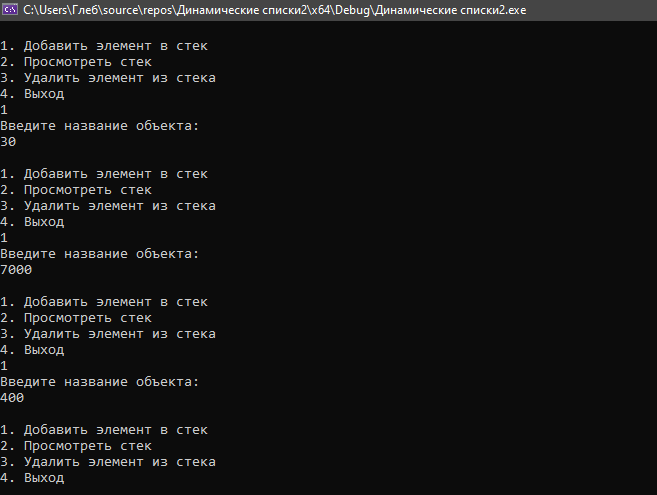


Рисунок 7 – Создание списка Стека

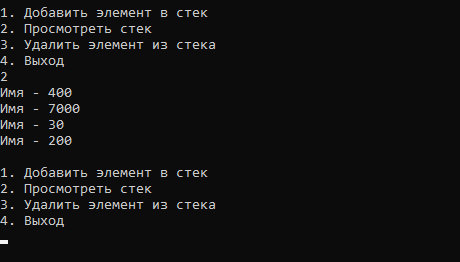


Рисунок 8 – Просмотр элементов Стека

Каждый новый элемент добавляется в начало списка (*push*()).

Удаление элемента происходит с головы списка, что соответствует принципу стека LIFO (*pop*()).

Функция *review*() выводит текущий стек на экран.

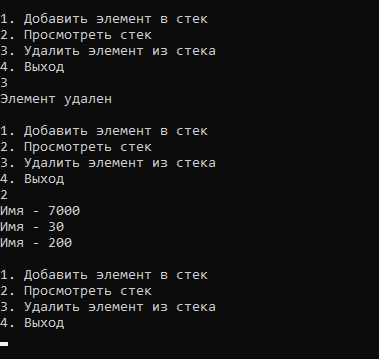


Рисунок 9 – Удаление элемента из Стека

**Вывод** - научились создавать динамические списки, ознакомились с функциями динамического списка.