Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра системного анализа и телекоммуникаций

**ПРОВЕРИЛ** **ВЫПОЛНИЛ**

Лапшин В. С. Мосейкин П.А.

« » апреля 2024 г. « » апреля 2024 г.

**ЛАБОРАТОРНАЯ РАБОТА №2**

по дисциплине

«Алгоритмизация и программирование»

на тему

«Динамическое распределение памяти, списки»

Оглавление

[Техническое задание 3](#_Toc162090996)

[Результаты работы программы 5](#_Toc162090997)

[Вывод 7](#_Toc162090998)

[Листинг программы 8](#_Toc162090999)

# 

# Техническое задание

**Цель работы**

Целью данной лабораторной работы является освоение студентами способов представления и обработки данных в виде списков.

**Задание**

Написать программы для работы с односвязным и двусвязным списками в соответствии с выданным вариантом задания. Предусмотреть в программах следующие функции:

1. Включить новый элемент в конец списка.

2. Включить новый элемент на заданное пользователем место списка.

3. Включить новый элемент после элемента с заданной информационной частью.

4. Включить новый элемент перед элементом с заданной информационной частью.

5. Включить новый элемент в середину списка.

6. Исключить элемент из середины списка.

7. Исключить элемент с заданной информационной частью.

8. Исключить элемент из конца списка.

9. Исключить элемент из заданного пользователем места списка.

10. Исключить элемент из головы списка.

Программы должны уметь обрабатывать структуру данных в соответствии с вариантом заданий. Варианты заданий:

1. Библиографическая карточка каталога библиотеки.

2. Карточка студента и его успеваемости деканата.

3. Структура учета автомобилей в ГАИ.

4. Учет материальных ценностей бухгалтерии.

5. Телефонная книга/справочник.

6. Домашний бюджет. Список покупок.

7. Расписание общественного транспорта.

8. Книга рецептов.

9. Адресная книга.

10. Ежедневник - деловой блокнот.

В соответствии с кодировочной таблицей, мой вариант - №9.

# Результаты работы программы

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 - результат работы кода с двусвязными списками

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – результат работы кода с односвязным списком

# Вывод

Проделав данную лабораторную работу, я узнал:

1) как управлять динамическим выделением памяти

2) что такое односвязные и двусвязные списки и их реализацию в языке Си

# Листинг программы

Односвязные списки:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

struct Data

{

char city[64];

char street[64];

char index[16];

char fio[256];

Data\* next;

};

Data\* create(const char\* city, const char\* street, const char\* index, const char\* fio)

{

Data\* new\_data = (Data\*)malloc(sizeof(Data));

if (new\_data)

{

strcpy(new\_data->city, city);

strcpy(new\_data->street, street);

strcpy(new\_data->index, index);

strcpy(new\_data->fio, fio);

new\_data->next = 0;

}

return new\_data;

}

void print(const Data\* data)

{

printf("Улица: %s; Индекс: %s; Адрес ячейки: %p; Следующая: %p\n", data->street, data->index, data, data->next);

}

void print\_list(Data\* head)

{

if (head != 0)

{

printf("Начало списка \n");

Data\* temp = head;

while (temp != 0)

{

print(temp);

temp = temp->next;

}

printf("\nКонец списка \n");

printf("/////////////////////\n");

}

else

printf("Список пуст\n");

}

void append(Data\*\* head, Data\* new\_data)

{

if (\*head != 0)

{

Data\* last = \*head;

while (last->next != 0)

last = last->next;

last->next = new\_data;

}

else

\*head = new\_data;

}

void add\_by\_pos(Data\*\* head, const unsigned pos, Data\* new\_data)

{

if (\*head)

{

if (pos == 0)

{

new\_data->next = \*head;

\*head = new\_data;

}

else

{

Data\* last = \*head;

unsigned cur\_pos = 0;

while (last->next != 0 && cur\_pos < pos - 1)

{

last = last->next;

cur\_pos = cur\_pos + 1;

};

if (cur\_pos == pos - 1)

{

new\_data->next = last->next;

last->next = new\_data;

}

else

printf("Указанного индекса не существует\n");

}

}

else if (pos == 0)

\*head = new\_data;

else

printf("Указанного индекса не существует\n");

}

void add\_by\_number\_after(Data\*\* head, const char\* index, Data\* new\_data)

{

if (\*head != 0)

{

Data\* temp = \*head;

while (temp != 0)

{

if (strcmp(temp->index, index) == 0)

{

new\_data->next = temp->next;

temp->next = new\_data;

break;

}

else

temp = temp->next;

}

}

else

printf("Список пуст, добавление по указанному индексу невозможно\n");

}

void add\_by\_number\_before(Data\*\* head, const char\* index, Data\* new\_data)

{

if (\*head != 0)

{

if (strcmp((\*head)->index, index) == 0)

{

new\_data->next = \*head;

\*head = new\_data;

}

else

{

Data\* temp = \*head;

while (temp->next != 0)

{

if (strcmp(temp->next->index, index) == 0)

{

new\_data->next = temp->next;

temp->next = new\_data;

break;

}

else

temp = temp->next;

}

}

}

else

printf("Ошибка, список пуст\n");

}

void add\_in\_center(Data\*\* head, Data\* new\_data)

{

if (\*head != 0)

{

Data\* mid = \*head;

Data\* end = mid->next;

while (end != 0 && end->next != 0)

{

mid = mid->next;

end = end->next->next;

}

new\_data->next = mid->next;

mid->next = new\_data;

}

else

\*head = new\_data;

}

void delete\_from\_center(Data\*\* head)

{

if (\*head != 0)

{

Data\* mid = \*head;

Data\* end = \*head;

Data\* prev = 0;

while (end != 0 && end->next != 0)

{

prev = mid;

mid = mid->next;

end = end->next->next;

}

if (prev != 0)

prev->next = mid->next;

else

\*head = 0;

free(mid);

}

else

printf("Ошибка, список пуст\n");

}

void delete\_by\_number(Data\*\* head, const char\* index)

{

if (\*head != 0)

{

Data\* temp = \*head;

if (strcmp(temp->index, index) == 0)

{

\*head = temp->next;

free(temp);

}

else

{

while (temp->next != 0)

{

if (strcmp(temp->next->index, index) == 0)

{

Data\* del = temp->next;

temp->next = temp->next->next;

free(del);

break;

}

else

temp = temp->next;

}

}

}

else

printf("Ошибка, список пуст\n");

}

void delete\_last(Data\*\* head)

{

if (\*head != 0)

{

Data\* temp = \*head;

if (temp->next == 0)

{

free(temp);

\*head = 0;

}

else

{

while (temp->next->next != 0)

temp = temp->next;

free(temp->next);

temp->next = 0;

}

}

else

printf("Error: List is empty!\n");

}

void delete\_by\_pos(Data\*\* head, const unsigned pos)

{

if (\*head != 0)

{

Data\* temp = \*head;

if (pos == 0)

{

\*head = temp->next;

free(temp);

}

else

{

unsigned cur\_pos = 0;

while (temp != 0 && cur\_pos < pos - 1)

{

temp = temp->next;

cur\_pos = cur\_pos + 1;

}

if (cur\_pos == pos - 1 && temp != 0 && temp->next != 0)

{

Data\* del = temp->next;

temp->next = temp->next->next;

free(del);

}

else

printf("Указанного индекса не существует\n");

}

}

else

printf("Ошибка: список пуст\n");

}

void delete\_first(Data\*\* head)

{

if (\*head != 0)

{

Data\* temp = \*head;

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

free(temp);

}

else

printf("Ошибка: список пуст\n");

}

int main()

{

setlocale(LC\_ALL, "ru");

Data\* new\_data\_0 = create("Taganrog", "Svobody", "324567", "Petrov Petr Petrovich");

Data\* new\_data\_1 = create("Rostov", "Pervomayskaya", "783367", "Vasilyev Vasiliy Vasilyevich");

Data\* new\_data\_2 = create("Moscow", "Sadovaya", "729812", "Ivanov Ivan Ivanovich");

Data\* new\_data\_3 = create("Zernograd", "Oktyabrskaya", "347740", "Alexeev Alexey Alexeyevich");

Data\* new\_data\_4 = create("Volgodonsk", "Mira", "340912", "Antonov Anton Antonovich");

Data\* head = 0;

printf("\n");

print\_list(head);

printf("После добавления в конец списка:\n");

append(&head, new\_data\_0);

print\_list(head);

printf("После добавления элемента по индексу 1:\n");

add\_by\_pos(&head, 1, new\_data\_1);

print\_list(head);

printf("Добавление элемента после элемента с заданной информационной частью(\"324567\"):\n");

add\_by\_number\_after(&head, "324567", new\_data\_2);

print\_list(head);

printf("Добавление элемента перед элементом с заданной информационной частью(\"783367\"):\n");

add\_by\_number\_before(&head, "783367", new\_data\_3);

print\_list(head);

printf("Добавление элемента в середину списка:\n");

add\_in\_center(&head, new\_data\_4);

print\_list(head);

printf("Исключение элемента из середины списка:\n");

delete\_from\_center(&head);

print\_list(head);

printf("Исключение элемента с заданной информационной частью(\"729812\"):\n");

delete\_by\_number(&head, "729812");

print\_list(head);

printf("Исключение элемента из конца списка:\n");

delete\_last(&head);

print\_list(head);

printf("Исключение элемента по заданному пользователем индексу (1):\n");

delete\_by\_pos(&head, 1);

print\_list(head);

printf("Исключение элемента из начала списка:\n");

delete\_first(&head);

print\_list(head);

return 0;

}

Двусвязные списки:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

struct Data

{

char city[64];

char street[64];

char index[16];

char fio[256];

Data\* prev;

Data\* next;

};

struct DoubleList

{

Data\* head;

Data\* end;

};

DoubleList create\_list()

{

DoubleList list;

list.head = 0;

list.end = 0;

return list;

}

Data\* create(const char\* city, const char\* street, const char\* index, const char\* fio)

{

Data\* new\_data = (Data\*)malloc(sizeof(Data));

if (new\_data != 0)

{

strcpy(new\_data->city, city);

strcpy(new\_data->street, street);

strcpy(new\_data->index, index);

strcpy(new\_data->fio, fio);

new\_data->prev = 0;

new\_data->next = 0;

}

return new\_data;

}

void print(const Data\* data)

{

printf("Улица: %s; Индекс: %s; Адрес ячейки: %p; Предыдущая: %p; Следующая: %p\n", data->street, data->index, data, data->prev, data->next);

}

void print\_list(DoubleList\* list)

{

if (list->head)

{

printf("Начало списка %p\n", list->head);

Data\* temp = list->head;

while (temp != 0)

{

print(temp);

temp = temp->next;

}

printf("\nКонец списка %p\n", list->end);

printf("/////////////////////\n");

}

else

printf("Список пуст\n");

}

void append(DoubleList\* list, Data\* new\_data)

{

if (list->head != 0)

{

list->end->next = new\_data;

new\_data->prev = list->end;

list->end = new\_data;

}

else

{

list->head = new\_data;

list->end = new\_data;

}

}

void add\_by\_pos(DoubleList\* list, const unsigned pos, Data\* new\_data)

{

if (list->head != 0)

{

if (pos == 0)

{

new\_data->next = list->head;

list->head->prev = new\_data;

list->head = new\_data;

}

else

{

Data\* temp = list->head;

unsigned cur\_pos = 0;

while (temp->next != 0 && cur\_pos < pos - 1)

{

temp = temp->next;

cur\_pos++;

}

if (cur\_pos == pos - 1)

{

new\_data->next = temp->next;

new\_data->prev = temp;

temp->next = new\_data;

if (new\_data->next != 0)

new\_data->next->prev = new\_data;

else

list->end = new\_data;

}

else

printf("Указанного индекса не существует\n");

}

}

else if (pos == 0)

{

list->head = new\_data;

list->end = new\_data;

}

else

printf("Список пуст, добавление по индексу %d невозможно\n", pos);

}

void add\_by\_number\_after(DoubleList\* list, const char\* index, Data\* new\_data)

{

if (list->head != 0)

{

Data\* temp = list->head;

while (temp != 0)

{

if (strcmp(temp->index, index) == 0)

{

new\_data->next = temp->next;

new\_data->prev = temp;

temp->next = new\_data;

if (new\_data->next != 0)

new\_data->next->prev = new\_data;

else

list->end = new\_data;

break;

}

else

temp = temp->next;

}

}

else

printf("Ошибка, список пуст\n");

}

void add\_by\_number\_before(DoubleList\* list, const char\* index, Data\* new\_data)

{

if (list->head != 0)

{

Data\* temp = list->head;

while (temp != 0)

{

if (strcmp(temp->index, index) == 0)

{

new\_data->next = temp;

new\_data->prev = temp->prev;

temp->prev = new\_data;

if (new\_data->prev != 0)

new\_data->prev->next = new\_data;

else

list->head = new\_data;

break;

}

else

temp = temp->next;

}

}

else

printf("Ошибка, список пуст\n");

}

void add\_in\_center(DoubleList\* list, Data\* new\_data)

{

if (list->head != 0)

{

Data\* mid = list->head;

Data\* end = mid->next;

while (end != 0 && end->next != 0)

{

mid = mid->next;

end = end->next->next;

}

new\_data->next = mid->next;

new\_data->prev = mid;

mid->next = new\_data;

if (new\_data->next != 0)

new\_data->next->prev = new\_data;

else

list->end = new\_data;

}

else

{

list->head = new\_data;

list->end = new\_data;

}

}

void delete\_from\_center(DoubleList\* list)

{

if (list->head != 0)

{

Data\* mid = list->head;

Data\* end = mid->next;

while (end != 0 && end->next != 0)

{

mid = mid->next;

end = end->next->next;

}

if (mid->prev != 0)

mid->prev->next = mid->next;

else

list->head = mid->next;

if (mid->next != 0)

mid->next->prev = mid->prev;

else

list->end = mid->prev;

free(mid);

}

else

printf("Ошибка, список пуст\n");

}

void delete\_by\_number(DoubleList\* list, const char\* index)

{

if (list->head)

{

Data\* temp = list->head;

while (temp != 0)

{

if (strcmp(temp->index, index) == 0)

{

if (temp->prev != 0)

temp->prev->next = temp->next;

else

list->head = temp->next;

if (temp->next != 0)

temp->next->prev = temp->prev;

else

list->end = temp->prev;

free(temp);

break;

}

else

temp = temp->next;

}

}

else

printf("Ошибка, список пуст\n");

}

void delete\_last(DoubleList\* list)

{

if (list->end != 0)

{

Data\* temp = list->end;

if (temp->prev != 0)

{

list->end = temp->prev;

list->end->next = 0;

}

else

list->head = list->end = 0;

free(temp);

}

else

printf("Ошибка, список пуст\n");

}

void delete\_by\_pos(DoubleList\* list, const unsigned pos)

{

if (list->head != 0)

{

Data\* current = list->head;

unsigned cur\_pos = 0;

while (current != 0 && cur\_pos < pos)

{

current = current->next;

cur\_pos++;

}

if (current && cur\_pos == pos)

{

if (current->prev != 0)

current->prev->next = current->next;

else

list->head = current->next;

if (current->next != 0)

current->next->prev = current->prev;

else

list->end = current->prev;

free(current);

}

else

printf("Удаление по индексу %d невозможно, элементов недостаточно\n", pos);

}

else

printf("Удаление по индексу %d невозможно, элементов недостаточно\n", pos);

}

void delete\_first(DoubleList\* list)

{

Data\* temp = list->head;

list->head = temp->next;

if (temp->next != 0)

temp->next->prev = 0;

else

list->end = 0;

free(temp);

}

int main()

{

setlocale(LC\_ALL, "RU");

Data\* new\_data\_0 = create("Taganrog", "Svobody", "324567", "Petrov Petr Petrovich");

Data\* new\_data\_1 = create("Rostov", "Pervomayskaya", "783367", "Vasilyev Vasiliy Vasilyevich");

Data\* new\_data\_2 = create("Moscow", "Sadovaya", "729812", "Ivanov Ivan Ivanovich");

Data\* new\_data\_3 = create("Zernograd", "Oktyabrskaya", "347740", "Alexeev Alexey Alexeyevich");

Data\* new\_data\_4 = create("Volgodonsk", "Mira", "340912", "Antonov Anton Antonovich");

DoubleList list = create\_list();

printf("\n");

print\_list(&list);

printf("После добавления в конец списка:\n");

append(&list, new\_data\_0);

print\_list(&list);

printf("После добавления элемента по индексу 1:\n");

add\_by\_pos(&list, 1, new\_data\_1);

print\_list(&list);

printf("Добавление элемента после элемента с заданной информационной частью(\"324567\"):\n");

add\_by\_number\_after(&list, "324567", new\_data\_2);

print\_list(&list);

printf("Добавление элемента перед элементом с заданной информационной частью(\"783367\"):\n");

add\_by\_number\_before(&list, "783367", new\_data\_3);

print\_list(&list);

printf("Добавление элемента в середину списка:\n");

add\_in\_center(&list, new\_data\_4);

print\_list(&list);

printf("Исключение элемента из середины списка:\n");

delete\_from\_center(&list);

print\_list(&list);

printf("Исключение элемента с заданной информационной частью(\"729812\"):\n");

delete\_by\_number(&list, "729812");

print\_list(&list);

printf("Исключение элемента из конца списка:\n");

delete\_last(&list);

print\_list(&list);

printf("Исключение элемента по заданному пользователем индексу (1):\n");

delete\_by\_pos(&list, 1);

print\_list(&list);

printf("Исключение элемента из начала списка:\n");

delete\_first(&list);

print\_list(&list);

return 0;

}