Министерство науки и образования РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

им. В.И. Ульянова (Ленина)»

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Отчет по лабораторной работе № 4**

**по дисциплине «Программирование» на**

**тему:** **«Стек и очередь в языке C/C++»**

|  |  |
| --- | --- |
| Выполнил студент гр. 9308 | Дементьев Д.П. |
|  |  |
| Проверил к.т.н., доцент | Сискович Т.И |

Санкт-Петербург, 2020

Оглавление

**[Цель 4](#_Toc14866)**

**[1. Задание 4](#_Toc4700)**

**[2. Уточнение задания 4](#_Toc11405)**

**[4. Краткое описание алгоритма 9](#_Toc27464)**

**[5. Структура вызова функций 10](#_Toc10743)**

**[6. Функции 10](#_Toc10554)**

[6.1 Функция main 11](#_Toc31684)

[6.2 Функция get\_string 13](#_Toc18012)

[6.3 Функция get\_int 14](#_Toc18013)

[6.4 Функция get\_float 15](#_Toc29608)

[6.5 Функция get\_subject 16](#_Toc15824)

[6.6 Функция add\_item 17](#_Toc7028)

[6.7 Функция make\_head 18](#_Toc30628)

[6.8 Функция create\_node 19](#_Toc18285)

[6.9 Функция add\_first 20](#_Toc8185)

[6.10 Функция add\_last 21](#_Toc8145)

[6.11 Функция remove\_node 22](#_Toc8260)

[6.12 Функция copy\_node 23](#_Toc2474)

[6.13 Функция clean\_node 24](#_Toc3368)

[6.14 Функция clean\_list 25](#_Toc968)

[6.15 Функция print\_tutors 26](#_Toc7182)

[6.16 Функция get\_item 27](#_Toc8318)

[6.17 Функция Menu 28](#_Toc9816)

**[8. Текст программы 29](#_Toc10874)**

**[8. Пример работы программы 42](#_Toc12143)**

[8.1. Вывод программы 43](#_Toc512)

[8.2 Результат работы программы 45](#_Toc4331)

**[9. Заключение 46](#_Toc4230)**

Цель

Целью лабораторной работы является приобретение практических навыков в реализации стека и очереди на языке программирования C/C++.

# 1. Задание

Написать программу, выполняющую работу со стеком и очередью, содержащими характеристики объектов выбранной предметной области, размер которых заранее неизвестен. Реализовать действия со списком в режиме стека/очереди: добавление и удаление элемента, согласно режиму работы(стек или очередь), просмотр списка имеющихся элементов, удаление элемента, согласно режиму работы.

# 2. Уточнение задания

При выполнении задания необходимо учитывать:

1. Вводимый текст не может быть пустым.
2. Максимальная длина вводимой строки – 24 символа.
3. Пользователь может выбрать работу со стеком или очередью перед выполнением программы.
4. При смене режима работы список не очищается, а только меняется режим добавления/удаления элементов (например, добавление в начало списка для стека и в конец списка - для очереди).
5. Предметная область - база репетиторов, в которой есть такие критерии как имя, цена за час, предмет преподавания, квалификация, рейтинг.

Для данной предметной области были выбраны следующие поля структур:

Для структуры tutor:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| name | char\* | Имя преподавателя |
| subject | char\* | Предмет преподавания |
| qual | char | Квалификация преподавателя |
| price | int | Цена за час занятия |
| rating | float | Рейтинг преподавателя |

Для структуры Node:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| next | tutor\_elem\* | Указатель на следующую структуру |
| info | tutor | Вложенная структура с информацией об объекте |
| prev | tutor\_elem\* | Указатель на предыдущую структуру |

Для структуры Head:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Тип** | **Назначение** |
| count | int | Количество узлов в списке |
| first | Node\* | Указатель на адрес первого элемента списка |
| last | Node\* | Указатель на адрес последнего элемент в списке |

Поле subject определяется введенным пользователем числом от 1 до 5, обозначающим номер предмета:

1. Maths

2. Computer science

3. English

4. Russian

5. Physics

Меню программы должно иметь следующую иерархию:

1 - Режим очереди

* 1. - Добавление элемента
  2. - Вывод данных
  3. - Просмотр элемента
  4. - Удаление элемента

1.0 - Вернуться в главное меню

2 - Режим стека

2.1 - Добавление элемента

2.2 - Вывод данных

2.3 - Просмотр элемента

2.4 - Удаление элемента

2.0 - Вернуться в главное меню

3 - Справка

0 - Выход

# 3. Контрольные примеры

Контрольные примеры приведены в таблице 1.

Таблица 1. Контрольные примеры

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Поля структуры | | | | | Очередь | Стек |
| - Удалить элемент | - Удалить элемент  - Удалить элемент |
| Имя | Предмет | Цена | Рейтинг | Квал. | **Вывод структур** | |
| 1 | Alex | Maths | 900 | 5 | A | 2, 3 | 1 |
| 2 | Vlad | Maths | 350 | 3 | C |
| 3 | Liza | Physics | 500 | 4 | A |

**4. Краткое описание алгоритма**

Начало программы.

Шаг №1. Вывод меню на экран.

Шаг №2. Выбор пользователем одного из пунктов меню.

Шаг №3. Переход к пункту, выбранным пользователем:

1 - Режим очереди

2 - Режим стека

3 - Справка

0 - Выход

Шаг №4. Если пользователь не захотел выйти - переход к шагу 1, иначе конец программы.

Конец.

# 5. Структура вызова функций

Рисунок 1. Структура вызова функций

# struct_main_1 (1) (2)6. Функции

## 6.1 Функция main

**Назначение:**

Является точкой входа в программу.

**Описание переменных:**

Описание переменных приведено в таблице 2.

Таблица 2. Описание переменных функции main

|  |  |  |
| --- | --- | --- |
| Имя переменной | Тип | Назначение |
| Q3 | int | Номер пункта подменю |
| Q | int | Номер пункта меню |
| c | char | Символ, вводимый с клавиатуры |
| p | Node\* | Проверка выделения памяти для входных данных |
| output | int | Проверка выделения памяти для выходных данных |
| HEAD | Head\* | Указатель на голову списка |
| NEW\_HEAD | Head\* | Указатель на новую голову списка |
| first | int | Номер первого элемента |
| second | int | Номер второго элемента |
| Mode | int | Номер режима работы программы |

**Схема алгоритма:**

Схема алгоритма для функции main представлена на рисунке 2.

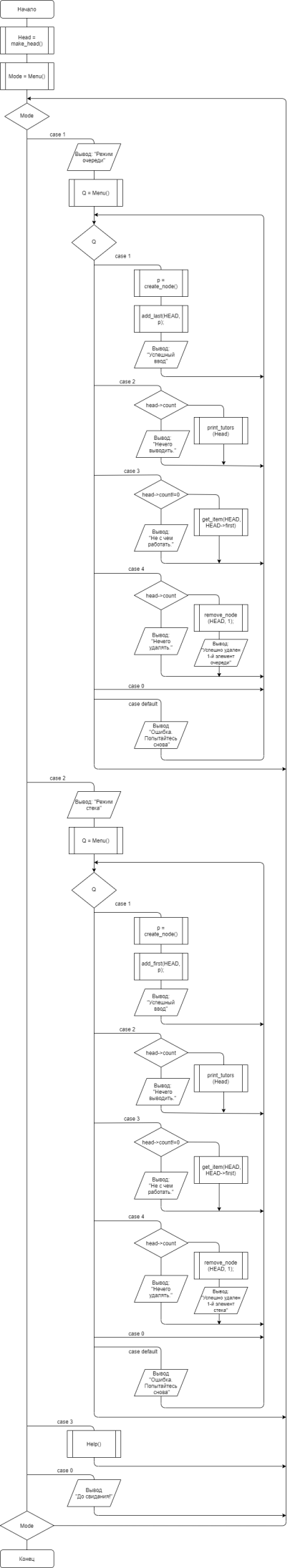


Рисунок 2. Схема алгоритма

## 6.2 Функция get\_string

**Назначение:**

Функция ввода строки.

**Прототип:**

char \*get\_string();

**Пример вызова:**

list->name = get\_string();

**Описание переменных:**

Описание переменных приведено в таблице 3.

Таблица 3. Описание переменных функции get\_string

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | c | char | Символ строки |
| string | сhar\* | Адрес начала строки |
| i | int | Индекс символа строки |

## 6.3 Функция get\_int

**Назначение:**

Функция ввода числа типа int.

**Прототип:**

int get\_int(void);

**Пример вызова:**

list->price = get\_int();

**Описание переменных:**

Описание переменных приведено в таблице 4.

Таблица 4. Описание переменных функции get\_int

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | line | char | Буфер |
| curChar | сhar | Переменная для определения корректности ввода |
| temp | int | Проверка успешной записи числа |
| result | int | Сформированное значение типа int |

## 6.4 Функция get\_float

**Назначение:**

Функция ввода числа типа float.

**Прототип:**

int get\_float(void);

**Пример вызова:**

list->rating = get\_float();

**Описание переменных:**

Описание переменных приведено в таблице 5.

Таблица 5. Описание переменных функции get\_float

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | line | char | Буфер |
| curChar | char | Переменная для определения корректности ввода |
| temp | int | Проверка успешной записи числа |
| result | float | Сформированное значение типа float |

## 6.5 Функция get\_subject

**Назначение:**

Функция для выбора категории пользователем.

**Прототип:**

сhar \*get\_subject(void);

**Пример вызова:**

List->subject = get\_subject();

**Описание переменных:**

Описание переменных приведено в таблице 6.

Таблица 6. Описание переменных функции get\_subject

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | choice | char\* | Указатель на выбранный пользователем предмет |
| c | char | Проверка на конец файла |
| q | int | Номер выбранного предмета |
| i | int | Счетчик цикла |

## 6.6 Функция add\_item

**Назначение:**

Функция добавления новой структуры.

**Прототип:**

void \*add\_item(tutor\*);

**Пример вызова:**

add\_item(data + ++(\*count) -1);

**Описание переменных:**

Описание переменных приведено в таблице 7.

Таблица 7. Описание переменных функции add\_item

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | list | tutor\* | Указатель на новую структуру |
| Локальная | с | char | Проверка конца строки |

## 6.7 Функция make\_head

**Назначение:**

Инициализация головы cписка.

**Прототип:**

Head \*make\_head();

**Пример вызова:**

NEW\_HEAD = make\_head();

**Описание переменных:**

Описание переменных приведено в таблице 8.

Таблица 8. Описание переменных функции make\_head

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | ph | Head\* | Указатель на голову списка |

## 6.8 Функция create\_node

**Назначение:**

Создание элемента списка.

**Прототип:**

Node \*create\_node()

**Пример вызова:**

p = create\_node();

**Описание переменных:**

Описание переменных приведено в таблице 9.

Таблица 9. Описание переменных функции сreate\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | new\_node | Node\* | Указатель на адрес элемента списка |

**6.9 Функция add\_first**

**Назначение:**

Добавить элемент в начало списка.

**Прототип:**

void \*add\_first(Head\*, Node\*);

**Пример вызова:**

add\_first(my\_head, new\_node);

**Описание переменных:**

Описание переменных приведено в таблице 10.

Таблица 10. Описание переменных функции add\_first

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | new\_node | Node\* | Указатель на новый узел |
| my\_head | Head\* | Указатель на адрес головы списка |

**6.10 Функция add\_last**

**Назначение:**

Добавить элемент в конец списка.

**Прототип:**

void \*add\_last(Head\*, Node\*);

**Пример вызова:**

add\_last(my\_head, new\_node);

**Описание переменных:**

Описание переменных приведено в таблице 11.

Таблица 11. Описание переменных функции add\_last

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | new\_node | Node\* | Указатель на новый узел |
| my\_head | Head\* | Указатель на адрес головы списка |

**6.11 Функция remove\_node**

**Назначение:**

Удалить выбранный элемент из списка.

**Прототип:**

void remove\_node(Head\*)

**Пример вызова:**

remove\_node(HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 12.

Таблица 12. Описание переменных функции remove\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | my\_head | Head\* | Указатель на голову |
| Локальная | i | int | Счетчик для цикла |
| pos | int | Номер позиции элемента, который необходимо удалить |
| c | char | Символ, вводимый пользователем с клавиатуры |
| p | Node\* | Указатель на узел |
| buff | Node\* | Буфер |

**6.12 Функция copy\_node**

**Назначение:**

Копирование элемента.

**Прототип:**

Node \*copy\_node(Node\*);

**Пример вызова:**

add\_last(NEW\_HEAD, copy\_node(p));

**Описание переменных:**

Описание переменных приведено в таблице 13.

Таблица 13. Описание переменных функции copy\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | NODE | Node\* | Указатель на адрес элемента списка |
| Локальная | p | Node\* | Указатель на узел |

**6.13 Функция clean\_node**

**Назначение:**

Очистка памяти для одной записи.

**Прототип:**

Node \*clean\_node(Node\*);

**Пример вызова:**

clean\_node(p);

**Описание переменных:**

Описание переменных приведено в таблице 14.

Таблица 14. Описание переменных функции clean\_node

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | node | Node\* | Указатель на адрес удаляемого элемента |

**6.14 Функция clean\_list**

**Назначение:**

Очистка памяти под список.

**Прототип:**

Head \*clean\_list(Head\*);

**Пример вызова:**

NEW\_HEAD = clean\_list(NEW\_HEAD);

**Описание переменных:**

Описание переменных приведено в таблице 15.

Таблица 15. Описание переменных функции clean\_list

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову |
| Локальная | p | Node\* | Адрес первого элемента списка |
| temp | Node\* | Временная переменная |
| i | int | Счетчик для цикла |

## 6.15 Функция print\_tutors

**Назначение:**

Печать списка в виде таблицы.

**Прототип:**

void print\_tutors(tutor\*, int);

**Пример вызова:**

print\_tutors(data, k);

**Описание переменных:**

Описание переменных приведено в таблице 16.

Таблица 16. Описание переменных функции print\_tutors

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | i | int | Счетчик для цикла |
| p | Node\* | Хранит текущий элемент списка |
| Формальная | my\_head | Head\* | Указатель на голову |

**6.16 Функция get\_item**

**Назначение:**

Просмотр элемента.

**Прототип:**

void get\_item(Head \*HEAD, Node \*p);

**Пример вызова:**

get\_item(HEAD, HEAD->first);

**Описание переменных:**

Описание переменных приведено в таблице 17.

Таблица 17. Описание переменных функции get\_item

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Формальная | HEAD | Head\* | Указатель на голову списка |
| p | Node\* | Указатель на узел |

## 6.17 Функция Menu

**Назначение:**

Функция вывода меню.

**Прототип:**

int Menu();

**Пример вызова:**

Q **=** Menu();

**Описание переменных:**

Описание переменных приведено в таблице 18.

Таблица 18. Описание переменных функции Menu

|  |  |  |  |
| --- | --- | --- | --- |
| Вид переменной | Имя переменной | Тип | Назначение |
| Локальная | Q | int | Номер пункта меню, выбираемый пользователем |

# 8. Текст программы

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <malloc.h>

#include <conio.h>

#include <string.h>

#include <windows.h>

#define MAXLEN 24

const char \*subjects[] = {"Maths" , "Computer science", "English", "Russian", "Physics"};

typedef struct

{

// Описание полей

char \*name; // Имя

char \*subject; // Предмет

int price; // Цена за час

char qual; // Квалификация

float rating; // рейтинг преподавателя

} tutor; // Информация о репетиторе

typedef struct tutor\_elem

{

tutor info;

struct tutor\_elem \*prev;

struct tutor\_elem \*next;

} Node; // Элемент списка

typedef struct

{

int count;

Node \*first;

Node \*last;

} Head; // Голова двусвязного списка

// Получение валидных значений //

char \*get\_string(); // Получение строки

int get\_int(); // Получение целого числа

float get\_float(); // ПОлучение числа с плавающей запятой

char \*get\_subject(); // Получение учебного предмета

// Работа со списком //

void \*add\_item(tutor \*list); // Заполнение информационных полей у репетитора

Head \*make\_head(); // Инициализация головы списка

Node \*create\_node(); // Создать элемент списка

void \*add\_first(Head \*my\_head, Node \*new\_node); // Добавить элемент в начало списка

void \*add\_last(Head \*my\_head, Node \*new\_node); // Добавить элемент в конец списка

void remove\_node(Head \*my\_head, int pos); // Удаление элемента

Node \*copy\_node(Node \*NODE); // Копирование элемента

void print\_tutors(Head \*my\_head); // Печать списка в виде таблицы

Node \*clean\_node(Node \*); // Очистка памяти для одной записи

void get\_item(Head \*HEAD, Node \*p); // Вывод элемента и его удаление

Head \*clean\_list(Head \*HEAD); // Очистить память под список

// Интерфейс //

int Menu(); // Вывод меню и выбор его пункта

void Help(); // Справка

int main()

{

Head \*HEAD = NULL, \*NEW\_HEAD = NULL;

Node \*p = NULL;

HEAD = make\_head();

int Mode, Q, Q3,output = 0;

int first, second;

char c = 0;

do

{

Mode = Menu(-1);

switch (Mode)

{

case 1: // Queue

do {

puts("QUEUE MODE");

Q = Menu(0);

switch (Q)

{

case 1: //input

p = create\_node();

add\_last(HEAD, p);

printf("Successful input.\n");

break;

case 2: //output

if (HEAD->count)

print\_tutors(HEAD);

else

printf("No input to print!\n");

break;

case 3:

if (HEAD->count != 0)

{

get\_item(HEAD, HEAD->first);

}

else

printf("No input to actions!\n");

break;

case 4: //process

if (HEAD->count)

{

remove\_node(HEAD, 1);

puts("First in the queue was deleted successfully");

}

else

printf("No input to remove!\n");

break;

case 0:

break;

default:

printf("Error! Try again!\n");

break;

}

system("pause");

} while (Q);

break;

case 2: // Stack

do {

puts("STACK MODE");

Q = Menu(0);

switch (Q)

{

case 1: //input

p = create\_node();

add\_first(HEAD, p);

printf("Successful input.\n");

break;

case 2: //output

if (HEAD->count)

print\_tutors(HEAD);

else

printf("No input to print!\n");

break;

case 3:

if (HEAD->count != 0)

{

get\_item(HEAD, HEAD->first);

}

else

printf("No input to actions!\n");

break;

case 4: //process

if (HEAD->count)

{

remove\_node(HEAD, 1);

puts("First item successfully deleted");

}

else

printf("No input to remove!\n");

break;

case 0:

break;

default:

printf("Error! Try again!\n");

break;

}

system("pause");

} while (Q);

break;

case 3:

Help();

case 0: //exit

printf("Good buy, see you soon!\n");

break;

}

} while (Mode);

// Очистка всего мусора

if (HEAD)

HEAD = clean\_list(HEAD);

return 0;

}

char \*get\_string() // Функция получения строки

{

char c, \*string = (char\*)malloc(MAXLEN\*sizeof(char));

int i=0;

do

{

while ((c=getchar()) != '\n' && i < MAXLEN-1) string[i++] = c;

string[i] = '\0';

if (i < 1) printf("Error. You entered empty string. Please, try again.\n");

} while (i < 1);

if (i >= MAXLEN-1)

while ((c = getchar()) != '\n' && c != EOF);

return string;

}

int get\_int() // Функция получения целого числа

{

char line[MAXLEN]; //buffer

char curChar = ' ';

int temp, result;

do {

line[MAXLEN - 1] = '\n';

fgets(line, MAXLEN - 1, stdin);

temp = sscanf(line, "%d%c", &result, &curChar);

temp = !temp || temp < 0 || (curChar != '\n' && curChar != ' ');

if (temp)

printf("Error reading number. Please, try again.\n");

} while (temp); //not a number actually

if (line[MAXLEN - 1] != '\n') //clear all iput

while ((curChar = getchar()) != '\n' && curChar != EOF);

return result;

}

float get\_float() // Функция получения числа с плавающей запятой

{

char line[MAXLEN]; //buffer

char curChar = ' ';

int temp;

float result;

do {

line[MAXLEN - 1] = '\n';

fgets(line, MAXLEN - 1, stdin);

temp = sscanf(line, "%f%c", &result, &curChar);

temp = !temp || temp < 0 || (curChar != '\n' && curChar != ' ');

if (temp)

printf("Error reading number. Please, try again.\n");

} while (temp); //not a number actually

if (line[MAXLEN - 1] != '\n') //clear all iput

while ((curChar = getchar()) != '\n' && curChar != EOF);

return result;

}

char \*get\_subject() // Функция выбора Учебного предмета из заданных заранее

{

char \*choice = NULL, c;

int q, i;

for (i=0; i<(int)sizeof(subjects)/sizeof(char\*); i++)

printf("%d. %s\n", i+1, subjects[i]);

do

{

scanf("%d", &q);

if (q <= 0 || q > (int)sizeof(subjects)/sizeof(char\*))

printf("This item doesn't exist. Try again.\n");

} while (q <= 0 || q > (int)sizeof(subjects)/sizeof(char\*));

while ((c = getchar()) != '\n' && c != EOF);

printf("Your chosen %s\n", choice = subjects[q-1]);

return choice;

}

void \*add\_item(tutor \*list) // Добавить элемент в список

{

system("cls");

char c;

list->name = (char\*)malloc(MAXLEN\*sizeof(char));

list->subject = (char\*)malloc(MAXLEN\*sizeof(char));

if (list->subject && list->name)

{

printf("Enter tutor's name: \n");

list->name = get\_string();

printf("Enter academic subject: \n");

list->subject = get\_subject();

do

{

printf("Enter price per 1 hour(RUB): \n");

list->price = get\_int();

} while (list->price < 0);

printf("Enter qualifications: \n");

scanf("%c", &(list->qual));

while ((c=getchar()) != '\n' && c != EOF);

do

{

printf("Enter tutor's rating[From 0 to 5]: \n");

list->rating = get\_float();

} while (list->rating <= 0 || list->rating > 5);

}

}

Head \*make\_head() // Инициализация головы

{

Head \*ph=NULL;

ph=(Head\*)malloc(sizeof(Head));

if(ph!=NULL)

{

ph->count=0;

ph->first=NULL;

ph->last=NULL;

}

return ph;

}

Node \*create\_node() // Создать заполненную запись

{

Node \*new\_node=NULL;

new\_node = (Node\*)malloc(sizeof(Node));

if(new\_node)

{

add\_item(&(new\_node->info));

}

new\_node->prev = NULL;

new\_node->next = NULL;

return new\_node;

}

void \*add\_first(Head \*my\_head, Node \*new\_node) // Добавить новую запись первой в список

{

if(my\_head&&new\_node)

{

if (!(my\_head->count))

my\_head->last = new\_node;

else

{

(my\_head->first)->prev = new\_node;

new\_node->next = my\_head->first;

}

my\_head->first = new\_node;

my\_head->count++;

}

}

void \*add\_last(Head \*my\_head, Node \*new\_node) // Добавить новую запись последней в список

{

if(my\_head&&new\_node)

{

if (!(my\_head->count))

my\_head->first = new\_node;

else

{

(my\_head->last)->next = new\_node;

new\_node->prev = my\_head->last;

}

my\_head->last = new\_node;

my\_head->count++;

}

}

void insert(Head \*my\_head, Node \*new\_node) // Вставка в любое место

{

int i, pos;

Node \*p;

do

{

printf("What position? [From 1 to %d]\n", my\_head->count+1);

pos = get\_int();

} while (pos < 1 || pos > my\_head->count+1);

if(my\_head&&new\_node)

{

if (!(my\_head->count)) // Проверка на пустой список

{

my\_head->first = new\_node;

my\_head->last = new\_node;

my\_head->count++;

}

else if (pos > 0 && pos < my\_head->count+2)

{

if (pos==1)

add\_first(my\_head, new\_node);

else if (pos == my\_head->count + 1)

add\_last(my\_head, new\_node);

else {

p = my\_head->first;

for (i=1; i<pos-1; i++) { // Проматываем до позиции перед вставляемым элементом

p = p->next;

}

new\_node->prev = p;

new\_node->next = p->next;

(p->next)->prev = new\_node;

p->next = new\_node;

my\_head->count++;

}

}

}

}

void remove\_node(Head \*my\_head, int pos)

{

Node \*p;

int i;

char c;

p = my\_head->first;

if (my\_head->count > 1)

{

for (i=1; i<pos; i++) // Проматываем до удаляемого элемента

{

p = p->next;

}

if (pos==1)

{

my\_head->first = p->next;

(p->next)->prev = NULL;

}

else if (pos==my\_head->count)

{

my\_head->last = p->prev;

(p->prev)->next = NULL;

}

else

{

(p->prev)->next = p->next;

(p->next)->prev = p->prev;

}

}

else

{

my\_head->first = NULL;

my\_head->last = NULL;

}

my\_head->count--;

clean\_node(p);

}

Node \*copy\_node(Node \*NODE)

{

int i;

Node \*p=NULL;

p = (Node\*)malloc(sizeof(Node));

(p->info).name = (char\*)malloc(MAXLEN\*sizeof(char));

(p->info).subject = (char\*)malloc(MAXLEN\*sizeof(char));

if((p->info).subject!=NULL && (p->info).name!=NULL)

{

strcpy((p->info).name, (NODE->info).name);

strcpy((p->info).subject, (NODE->info).name);

(p->info).price = (NODE->info).price;

(p->info).qual = (NODE->info).qual;

(p->info).rating = (NODE->info).rating;

}

return p;

}

void get\_item(Head \*HEAD, Node \*p)

{

printf(" %s\n %s teacher\n Price per hour(RUB): %d\n Qualification: %c\n Rating: %.2f\n", (p->info).name, (p->info).subject, (p->info).price, (p->info).qual, (p->info).rating);

if (p->next == NULL)

remove\_node(HEAD, HEAD->count);

else

remove\_node(HEAD, 1);

}

void print\_tutors(Head \*my\_head)

{

int i;

Node \*p;

printf("| Tutor`s name | Subject |Price per hour(RUB)|Qualification| Rating |\n");

printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\n");

p = my\_head->first;

for (i = 0; i < my\_head->count; i++)

{

printf("|%22.22s|%16.16s|%19d|%13c|%8.2f|\n", (p->info).name, (p->info).subject, (p->info).price, (p->info).qual, (p->info).rating);

printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_|\n");

p = p->next;

}

}

// Очистка для конкретной записи

Node \*clean\_node(Node \*node)

{

free((node->info).subject);

free((node->info).name);

(node->info).subject = NULL;

(node->info).name = NULL;

node->next = NULL;

node->prev = NULL;

free(node);

return NULL;

}

// Очистка всего мусора

Head \*clean\_list(Head \*HEAD)

{

Node \*p, \*temp;

int i;

p = HEAD->first;

for (i = 0; i < HEAD->count; i++)

{

free((p->info).subject);

free((p->info).name);

temp = p;

p = p->next;

free(temp->next);

free(temp->prev);

}

free(HEAD->first);

free(HEAD->last);

return NULL;

}

int Menu(int q)

{

int Q; // Выбор пользователя

system("cls");

switch(q)

{

case -1:

puts("MENU");

puts("1 - Queue Mode");

puts("2 - Stack Mode");

puts("3 - Help");

puts("0 - Exit");

break;

case 0:

puts("1 - Add item");

puts("2 - Output data");

puts("3 - Get item");

puts("4 - Delete item");

puts("0 - To the Main menu");

break;

}

printf("Select menu item - ");

scanf("%d", &Q);

printf("\n");

fflush(stdin);

return Q;

}

void Help()

{

system("cls");

printf("\tHELP\n");

puts(" To work, you need to select the mode: queue or stack.");

puts(" You can add an item (1 menu item), see the entire list (2 menu item),");

puts(" view information about an item and delete it (3 menu item),");

puts(" and also delete the item without viewing the content (4 menu item).");

puts(" To exit the program, select the Exit menu item.\n");

system("pause");

}

# 8. Пример работы программы

## Исходные данные

Таблица 19. Исходные данные

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Поля структуры | | | | | Очередь | Стек |
| - Удалить элемент | - Удалить элемент  - Удалить элемент |
| Имя | Предмет | Цена | Рейтинг | Квал. | **Вывод структур** | |
| 1 | Alex | Maths | 900 | 5 | A | 2, 3 | 1 |
| 2 | Vlad | Maths | 350 | 3 | C |
| 3 | Liza | Physics | 500 | 4 | A |

## 8.1. Вывод программы

Для очереди:

Рисунок 3. Пример работы программы

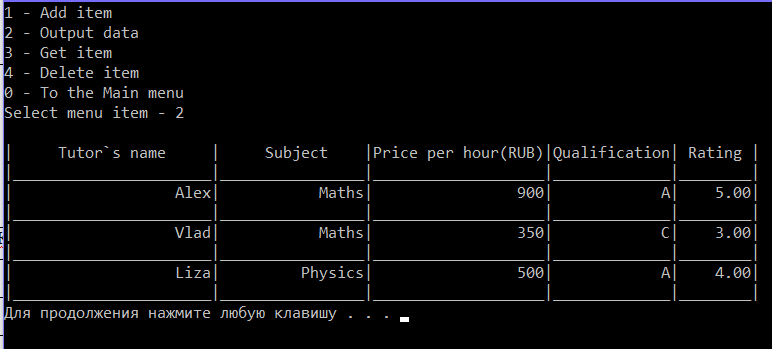


Рисунок 4. Пример работы программы

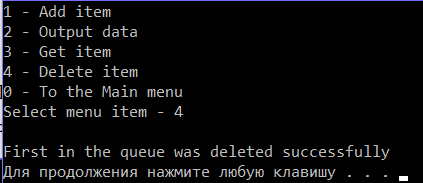
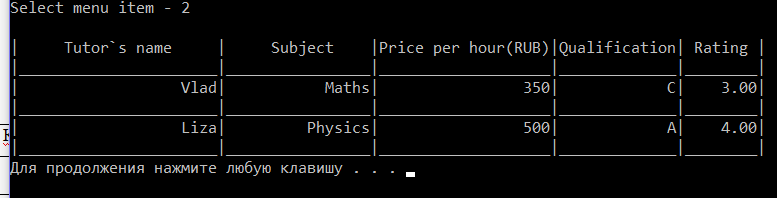


Рисунок 5. Пример работы программы



Для стека:

Рисунок 6. Пример работы программы

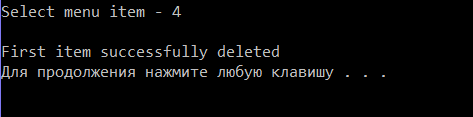
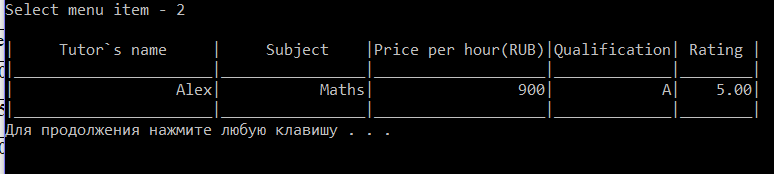


Рисунок 7. Пример работы программы



## 8.2 Результат работы программы

При выполнении программы получены результаты, совпадающие со значениями, приведенными в Таблице 19. Ошибки не обнаружены. Пример протокола выполнения программы приведены на рисунках 3-7.

# 9. Заключение

При выполнении лабораторной работы получены практические навыки в реализации стека и очереди на языке программирования C/C++.