МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе № 13
по дисциплине «Программирование»
Тема: **БИТОВЫЕ ПОЛЯ В СТРУКТУРАХ.**

Студент гр. 9305 _____Ивашкин В.В.

Преподаватель _____Перязева Ю.В,

Санкт-Петербург

2020

Содержание

Введение

Получить практические навыки в разработке алгоритма и написании программы на языке Си для знакомства с синтаксисом и битовыми полями в структурах, а также правилами написания кода на языке Си.

Задание

Написать программу учета сдачи зачетов при помощи битовых полей.
Структура содержит поля: фамилия, группа, зачеты (битовое поле, 4 бита).
Предусмотреть вывод списков сдавших все зачеты и должников по группам и в алфавитном порядке.

Исходные данные вводятся из файла с возможностью добавления при вводе с клавиатуры.

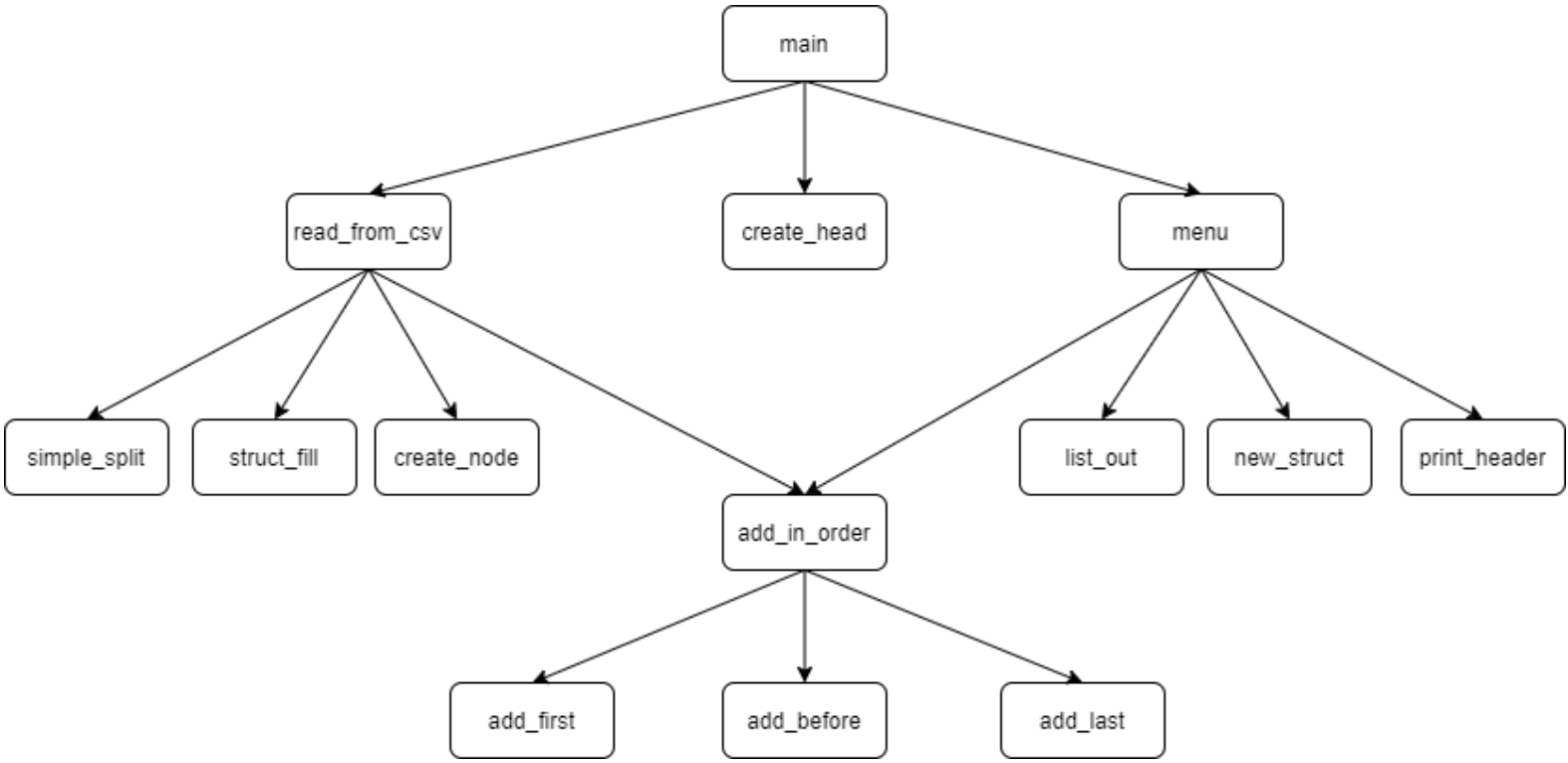
Постановка задачи и описание решения

Дан файл, содержащий информацию о студентах: фамилия, группа и 4 зачёта(сдал/не сдал). Необходимо написать программу, которая выводит списки тех, кто сдал все экзамены и тех, кто не сдал. Списки должны быть отсортированы в алфавитном порядке по группам. Предусмотреть возможность новых данных с клавиатуры.

Сначала считывается каждая строка и с помощью функции simple_split каждый пункт данных сохраняется в массив строк. Массив строк сохраняется в структуру с помощью функции struct_fill. С помощью функции create_node создаётся первый элемент списка. Остальные данные добавляются с помощью функции add_in_order, которая добавляет данные в то место, где они должны стоять в алфавитном порядке по группам.

Далее с помощью меню пользователь может вывести весь список студентов, список тех, кто сдал всё и список тех, кто не сдал, добавить нового студента.

Схема вызовов функций



Описание структур

1) struct marks

Имя поля	Тип поля	Назначение
m1	unsigned char	1 зачёт, 1 бит
m2	unsigned char	1 зачёт, 1 бит
m3	unsigned char	1 зачёт, 1 бит
m4	unsigned char	1 зачёт, 1 бит

2) Объединение total

Имя поля	Тип поля	Назначение
pass	unsigned	4 бита, содержащие зачёты
marks	struct marks	Зачёты по отдельности

3) students

Имя поля	Тип поля	Назначение
name	char*	Имя студента
group	int	Группа
resut	union total	Зачёты

4) struct node

Имя поля	Тип поля	Назначение
data	studs*	Информация, хранящаяся в элементе
next	struct node*	Указатель на следующий элемент
prev	struct node*	Указатель на предыдущий элемент

5) struct head

Имя поля	Тип поля	Назначение
cnt	int	Количество элементов в списке
first	struct node*	Указатель на первый элемент списка
last	struct node*	Указатель на последний элемент списка

Функции

1. Функция main

Описание:

Является точкой входа в программу. Вызывает функции create_head, read_from_csv, menu.

Прототип:

int main()

Пример вызова:

main()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	head2	head*	Голова списка

Возвращаемое значение:
0, если работа программы завершена корректно.

2. Функция simple_split

Описание:

Функция разделяет строки по заданному разделителю. Подсчитывается количество разделителей, по количеству выделяется память двумерного массива и заполняется отдельными элементами.

Прототип:

char **simple_split(char *str, int length, char sep)

Пример вызова:

s2=simple_split(s1,slen,sep)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char*	Строка, которую необходимо разделить
Формальный аргумент	length	int	Длина строки
Формальный аргумент	sep	char	Символ-разделитель
Локальная	str_array	char**	Массив строк
Локальная	i	int	Счётчик цикла
Локальная	j	int	Счётчик цикла
Локальная	k	int	Счётчик цикла
Локальная	m	int	Счётчик строки в массиве строк
Локальная	key	int	Проверка выделения памяти
Локальная	count	int	Проверка выделения памяти

Возвращаемое значение:
Массив строк

3. Функция ClearStringArray

Описание:

Функция очистки памяти для массива строк.

Прототип:

void ClearStringArray(char **str, int n)

Пример вызова:

ClearStringArray(str_array,count)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char**	Массив строк

Формальный аргумент	n	int	Количество строк
Локальная	i	int	Счётчик цикла

4. Функция struct_fill

Описание:

Функция заполнения структуры данными массива строк. Создаёт структуру и по порядку сохраняет данные в неё.

Прототип:

studs *struct_fill(char **str)

Пример вызова:

s3 = struct_fill(s2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	str	char**	Массив строк
Локальная	str0	studs*	Структура для заполнения

Возвращаемое значение:
Заполненная структура.

5. Функция new_struct

Описание:

Создаёт новую структуру, в которую пользователь вводит данные

Прототип:

studs *new_struct()

Пример вызова:

str0 = new_struct()

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Локальная	str0	studs	Новая структура
Локальная	mark	int	Оценка

Возвращаемое значение:
Заполненная структура.

6. Функция create_node

Описание:

Функция создаёт новый узел списка. Отводится память под новую запись, устанавливаются значения полей, переданные в аргументах, ссылка на следующий элемент устанавливается в NULL.

Прототип:

void create_node(studs *data, head *head)

Пример вызова:

create_node(s3,head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	data	studs*	Структура, хранящаяся в узле
Формальный аргумент	head	head*	Указатель на голову
Локальная	temp	node*	Узел списка

7. Функция list_out

Описание:

Функция вывода списка. Выводит данные узла до тех пор, пока указатель не равен NULL

Прототип:

void list_out(node **head)

Пример вызова:

list_out(&head)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову списка
Локальная	p	node*	Указатель на текущий узел

8. Функция add_before

Описание:

Функция добавляет структуру перед элементом

Прототип:

void add_before(head *head, node *nownode, studs *data)

Пример вызова:

add_before(head2,temp,data)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	head*	Указатель на голову списка
Формальный аргумент	data	studs*	Структура
Формальный аргумент	nownode	node*	Узел, перед которым надо добавить структуру
Локальная	temp	node*	Временный узел

9. Функции add_first/add_last

Описание:

Функции добавляют структуру в начало/конец двусвязного списка.

Прототип:

void add_first(head *head, studs *data)

void add_last(head *head, studs *data);

Пример вызова:

add_first(head,str0)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head	node**	Указатель на голову списка
Формальный аргумент	data	studs*	Структура
Локальная	temp	node*	Временный узел

10. Функция menu

Описание:

Функция выводит меню.

Прототип:

void menu(head *head2)

Пример вызова:

menu(head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head2	head*	Указатель на голову для взаимодействия со списком

Локальная	choice	int	Выбор пользователя
Локальная	str0	studs*	Новая структура

11. Функции free_head/free_node

Описание:

Функции отчищают голову и элементы списка.

Прототип:

void free_head(head *q)

void free_node(node *temp)

Пример вызова:

free_head(head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	q	head*	Указатель на голову списка
Формальный аргумент	temp	node*	Указатель на элемент списка

12. Функция free_list

Описание:

Функции отчищают список или только элементы списка.

Прототип:

void free_list(head *q)

Пример вызова:

free_list(head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	q	head*	Указатель на голову списка
Локальная	temp	node*	Указатель на элемент списка
Локальная	p	node*	Указатель на элемент списка
Локальная	i	int	Счётчик цикла
Локальная	n	int	Количество элементов

13. Функция read_from_csv

Описание:

Функция считывает данные из файла и сохраняет их в список.

Прототип:

void read_from_csv(head *head2)

Пример вызова:

read_from_csv(head2)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head2	head*	Указатель на голову списка
Локальная	slen	int	Длина строки
Локальная	i	int	Счётчик цикла
Локальная	number_of_lines	int	Количество строк в файле
Локальная	s2	char**	Строка с разделённой по символу-разделителю информацией
Локальная	s3	studs*	Информация об 1 студенте
Локальная	s1[256]	char	Строка
Локальная	sep	char	Символ-разделитель
Локальная	df	FILE*	Поток

14. Функция add_in_order

Описание:

Функции добавляют структуру в список по алфавиту по группам.

Прототип:

void add_in_order(head *head2, studs *data);

Пример вызова:

add_in_order(head2,s3)

Описание переменных:

Вид переменной	Имя переменной	Тип	Назначение
Формальный аргумент	head2	head*	Указатель на голову списка
Формальный аргумент	data	studs*	Структура
Локальная	temp	node*	Временный узел

Контрольные примеры

Исходные данные:

students-data – Блокнот

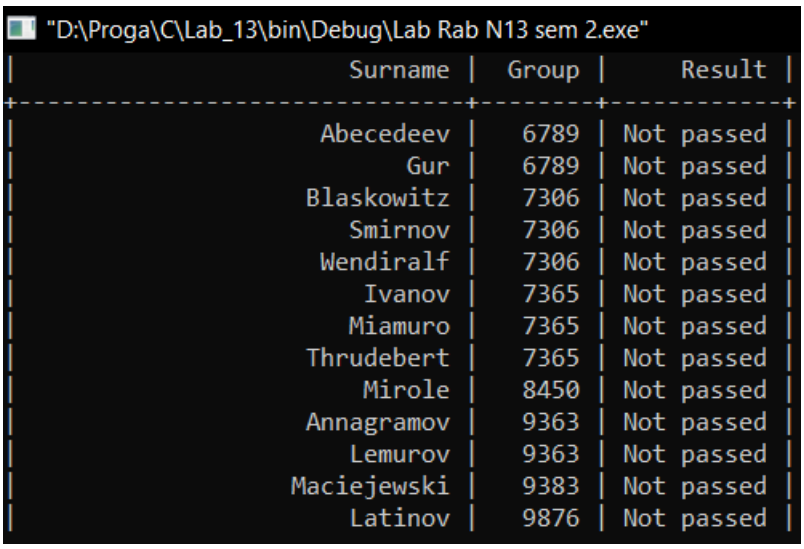
Файл Правка Формат Вид Справка

```
Ivanov;7365;1;0;0;1;
Smirnov;7306;1;0;1;1;
Stepikov;8450;1;1;1;1;
Lemurov;9363;0;1;1;1;
Timofeev;9876;1;1;1;1;
Abecedeev;6789;1;0;0;0;
Miamuro;7365;1;1;0;1;
Generatorov;8450;1;1;1;1;
Mirole;8450;1;0;0;1;
Annagramov;9363;0;1;1;0;
Wandaer;7365;1;1;1;1;
Kirillov;9363;1;1;1;1;
Latinov;9876;0;1;0;1;
Wendiralf;7306;0;1;1;1;
Elmok;7306;1;1;1;1;
Genewi;7365;1;1;1;1;
Gur;6789;0;1;1;1;
Arnoot;6789;1;1;1;1;
Thrudebert;7365;1;0;0;0;
Blaskowitz;7306;0;0;1;1;
Chrzanowski;8450;1;1;1;1;
Maciejewski;9383;0;0;0;0;
Brzeczyszczykiewicz;9876;1;1;1;1;
Karandashulikov;6789;1;1;1;1;
Letov;7365;1;1;1;1;
Structurkin;8450;1;1;1;1;
```

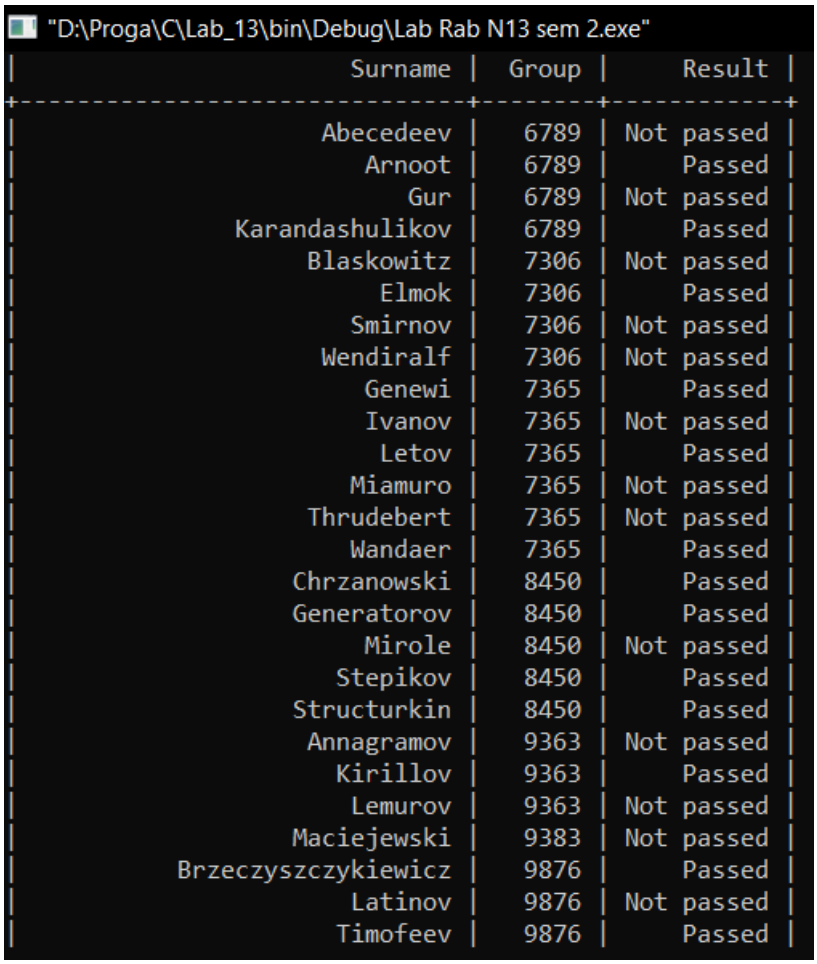
Работа программы:

"D:\Proga\C\Lab_13\bin\Debug\Lab Rab N13 sem 2.exe"

Surname	Group	Result
Arnoot	6789	Passed
Karandashulikov	6789	Passed
Elmok	7306	Passed
Genewi	7365	Passed
Letov	7365	Passed
Wandaer	7365	Passed
Chrzanowski	8450	Passed
Generatorov	8450	Passed
Stepikov	8450	Passed
Structurkin	8450	Passed
Kirillov	9363	Passed
Brzeczyszczykiewicz	9876	Passed
Timofeev	9876	Passed



Surname	Group	Result
Abecedeev	6789	Not passed
Gur	6789	Not passed
Blaskowitz	7306	Not passed
Smirnov	7306	Not passed
Wendiralf	7306	Not passed
Ivanov	7365	Not passed
Miamuro	7365	Not passed
Thrudebert	7365	Not passed
Mirole	8450	Not passed
Annagramov	9363	Not passed
Lemurov	9363	Not passed
Maciejewski	9383	Not passed
Latinov	9876	Not passed



Surname	Group	Result
Abecedeev	6789	Not passed
Arnoot	6789	Passed
Gur	6789	Not passed
Karandashulikov	6789	Passed
Blaskowitz	7306	Not passed
Elmok	7306	Passed
Smirnov	7306	Not passed
Wendiralf	7306	Not passed
Genewi	7365	Passed
Ivanov	7365	Not passed
Letov	7365	Passed
Miamuro	7365	Not passed
Thrudebert	7365	Not passed
Wandaer	7365	Passed
Chrzanowski	8450	Passed
Generatorov	8450	Passed
Mirole	8450	Not passed
Stepikov	8450	Passed
Structurkin	8450	Passed
Annagramov	9363	Not passed
Kirillov	9363	Passed
Lemurov	9363	Not passed
Maciejewski	9383	Not passed
Brzeczyszczykiewicz	9876	Passed
Latinov	9876	Not passed
Timofeev	9876	Passed

Текст программы

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "structs.h"
#include "read_from_file.h"
#include "adding.h"
#include "list.h"
#include "clean_list.h"



void menu(head *head2);


int main()
{
    head *head2 = NULL;

    head2 = create_head();
    read_from_csv(head2);


    menu(head2);

    //Clearing memory
    if((head2->cnt) == 1) free_head(head2);
    else if((head2->cnt) > 1) free_list(head2);
    else if ((head2->cnt) == 0) free(head2);


    return 0;
}


void menu(head *head2)
{
    int choice;
    studs *str0=NULL;

    do
    {
        printf("| |         Menu        |\n");
        printf("+-+---------------------+\n");
        printf("|1| - All students      |\n");
        printf("|2| - Passed all exams  |\n");
        printf("|3| - Didn't pass exams |\n");
```

```
        printf("|4| - Add information   |\n");
        printf("|0| - Exit              |\n");
        printf("Your choice: ");
        scanf("%d", &choice);
        getchar();

        switch(choice)
        {
        case 1:
            system("cls");
            print_header();
            list_out(head2,0); //All
            break;
        case 2:
            system("cls");
            print_header();
            list_out(head2,1); //Passed
            break;
        case 3:
            system("cls");
            print_header();
            list_out(head2,-1); //Not passed
            break;
        case 4:
            system("cls");
            printf("Adding info:\n");
            str0 = new_struct();
            add_in_order(head2,str0);
            system("cls");
            printf("Information added\n\n");
            break;
        case 0:
            system("cls");
            break;
        default:
            system("cls");
            puts("Incorrect input!");
            break;
        }

    } while(choice!=0);
}
```

read_from_file.h

```
#ifndef READ_FROM_FILE_H_INCLUDED
#define READ_FROM_FILE_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "structs.h"
#include "adding.h"
#include "list.h"


void read_from_csv(head *head2);
char **simple_split(char *str, int length, char sep);
studs *struct_fill(char **str);



#endif // READ_FROM_FILE_H_INCLUDED
```

read_from_file.c

```
#include "read_from_file.h"


void read_from_csv(head *head2)
{
    int slen,i,number_of_lines;
    char **s2=NULL;
    studs *s3=NULL;
    char s1[256];
    char sep;
    FILE *df;

    sep=';';

    df=fopen("students-data.csv","r");
    if(df!=NULL)
    {
        number_of_lines=0;
        while((fgets(s1,256,df))!=NULL) number_of_lines++;
        rewind(df);
        fgets(s1,256,df);
        slen=strlen(s1);
        s1[slen-1]='\0';
        slen=strlen(s1);

        s2=simple_split(s1,slen,sep);
        s3 = struct_fill(s2);
        create_node(s3,head2); //Creating first node

      //Splitting other strings, creating//
      //structures and making list      //
        for(i=0;i<number_of_lines-1;i++)
        {
            fgets(s1,256,df);
            slen=strlen(s1);
```

```
            s1[slen-1]='\0';
            slen=strlen(s1);

            s2=simple_split(s1,slen,sep);
            s3 = struct_fill(s2);
            add_in_order(head2,s3);
        }
        if (fclose(df)==EOF) printf ("Error with closing
file!");
    }
    else puts("File not found!");
}


char **simple_split(char *str, int length, char sep)
{
    char **str_array=NULL;
    int i,j,k,m;
    int key,count;
    for(j=0,m=0;j<length;j++)
    {
        if(str[j]==sep) m++;
    }

    key=0;
    str_array=(char**)malloc((m+1)*sizeof(char*));
    if(str_array!=NULL)
    {
        for(i=0,count=0;i<=m;i++,count++)
        {
            str_array[i]=(char*)malloc(length*sizeof(char));
            if(str_array[i]!=NULL) key=1;
            else
            {
                key=0;
                i=m;
            }
        }
        if(key)
        {
            k=0;
            m=0;
            for(j=0;j<length;j++)
            {
                if(str[j]!=sep) str_array[m][j-k]=str[j];
                else
                {
                    str_array[m][j-k]='\0';
                    k=j+1;
                    m++;
                }
```

```
            }
        }
        else
        {

            for(i=0;i<count;i++)
            {
                free(str_array[i]);
                str_array[i]=NULL;
            }
            free(str_array);
            str_array = NULL;
        }
     }
     return str_array;
}


studs *struct_fill(char **str)
{
    studs *str0=NULL;

    str0=(studs*)malloc(sizeof(studs));
    if(str0!=NULL)
    {
        str0->name=str[0];
        str0->group=atoi(str[1]);
        str0->result.marks.m1=atoi(str[2]);
        str0->result.marks.m2=atoi(str[3]);
        str0->result.marks.m3=atoi(str[4]);
        str0->result.marks.m4=atoi(str[5]);
    }
    return str0;
}
```

adding.h

```
#ifndef ADDING_H_INCLUDED
#define ADDING_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "structs.h"

void add_in_order(head *head2, studs *data);
void add_first(head *head, studs *data);
void add_last(head *head, studs *data);
void add_before(head *head, node *nownode, studs *data);
studs *new_struct();

#endif // ADDING_H_INCLUDED
```

adding.c

```
#include "adding.h"



void add_in_order(head *head2, studs *data)
{
    node *temp=NULL;

    temp = head2->first;
    if (data->group < temp->data->group)/*--------------------*/
add_first(head2,data);
    else if ((strcmp(data->name,temp->data->name) <= 0) &&
            (data->group == temp->data->group))/*-------------*/
add_first(head2,data);
    else
    {
        while (temp!=NULL &&
              (data->group > temp->data->group))/*------------*/
temp = temp->next;
        while (temp!=NULL &&
              (strcmp(data->name,temp->data->name) > 0) &&
              (data->group==temp->data->group))/*-------------*/
temp = temp->next;
        if (temp!=NULL) add_before(head2,temp,data);
        else add_last(head2,data);
    }
}


void add_first(head *head, studs *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    temp->data = (studs*)malloc(sizeof(studs));
    temp->prev = NULL;
    temp->next = head->first;
    head->first->prev = temp;
    head->first = temp;
    head->cnt += 1;
    temp->data = data;
}


void add_last(head *head, studs *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
```

```
    temp->data = (studs*)malloc(sizeof(studs));
    temp->next = NULL;
    temp->prev = head->last;
    head->last->next = temp;
    head->last = temp;
    head->cnt += 1;
    temp->data = data;
}


void add_before(head *head, node *nownode, studs *data)
{
    node *temp;

    temp = (node *)malloc(sizeof(node));
    temp->data = (studs*)malloc(sizeof(studs));
    temp->prev = nownode->prev;
    temp->next = nownode;
    nownode->prev->next = temp;
    nownode->prev = temp;
    temp->data = data;
    head->cnt += 1;

}



studs *new_struct()
{
    studs *str0=NULL;
    int mark;

    str0=(studs*)malloc(sizeof(studs));
    str0 -> name = (char*)malloc(sizeof(char)*32);
    if(str0!=NULL)
    {
        printf("Enter name: ");
        fgets((*str0).name,64,stdin);
        printf("Enter group: ");
        scanf("%d",&(*str0).group);
        printf("Enter marks\n[1] - Passed\n[0] - Not passed\n");
        do
        {
            printf("Enter 1'st mark: ");
            scanf("%d",&mark);
            if(mark!=0&&mark!=1) printf("Incorrect input! Needs
0 or 1!\n");
        }while (mark!=0&&mark!=1);
        str0->result.marks.m1 = mark;
        do
        {
```

```
            printf("Enter 2'nd mark: ");
            scanf("%d",&mark);
            if(mark!=0&&mark!=1) printf("Incorrect input! Needs
0 or 1!\n");
        }while (mark!=0&&mark!=1);
        str0->result.marks.m2 = mark;
        do
        {
            printf("Enter 3'rd mark: ");
            scanf("%d",&mark);
            if(mark!=0&&mark!=1) printf("Incorrect input! Needs
0 or 1!\n");
        }while (mark!=0&&mark!=1);
        str0->result.marks.m3 = mark;
        do
        {
            printf("Enter 4'th mark: ");
            scanf("%d",&mark);
            if(mark!=0&&mark!=1) printf("Incorrect input! Needs
0 or 1!\n");
        }while (mark!=0&&mark!=1);
        str0->result.marks.m4 = mark;
        str0->name[strlen(str0->name)-1]='\0';
    }
    return str0;
}
```

list.h

```
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
#include <stdio.h>
#include <stdlib.h>
#include "structs.h"


head *create_head();
void create_node(studs *data, head *head);
void print_header();
void list_out(head *head, int ch);
int is_pass(node *p);


#endif // LIST_H_INCLUDED
```

list.c

```
#include "list.h"


head *create_head()
{
    head *ph=NULL;

    ph=(head*)malloc(sizeof(head));
    if(ph)
    {
        ph->cnt=0;
        ph->first=NULL;
        ph->last=NULL;
    }
    return ph;
}



void create_node(studs *data, head *head)
{
      node *temp=NULL;
      temp = (node *)malloc(sizeof(node));
      temp->data = (studs*)malloc(sizeof(studs));

      temp->next = NULL;
      temp->prev = NULL;
      head->first = temp;
      head->last = temp;
      head->cnt += 1;
      temp->data = data;
}


void print_header()
{
    printf("|%30s |%7s |%11s |\n","Surname","Group","Result");
    printf("+--------------------------------+--------
+------------+\n");
}


void list_out(head *head, int ch)
{
    node *p;
      //int i;

    p = head->first;
```

```
    switch(ch)
    {
    case 0:
        while(p != NULL)
        {
            printf("|%30s |%7d |",
            p->data->name,
            p->data->group);
            if (is_pass(p)) printf ("%11s |\n","Passed");
            else printf ("%11s |\n","Not passed");
            p = p->next;
        }
        break;
    case 1:
        while(p != NULL)
        {
            if (is_pass(p))
            {
                printf("|%30s |%7d |%11s |\n",
                p->data->name,
                p->data->group,"Passed");
            }
            p = p->next;
        }
        break;
    case -1:
        while(p != NULL)
        {
            if (!is_pass(p))
            {
                printf("|%30s |%7d |%11s |\n",
                p->data->name,
                p->data->group,"Not passed");
            }
            p = p->next;
        }
        break;
    default:
        puts("UNCKNOWN ERROR WITH OUTPUT");
        break;
    }
    printf("\n");
}


int is_pass(node *p)
{
    int chk;

    if (p->data->result.marks.m1==1&&
        p->data->result.marks.m2==1&&
```

```
        p->data->result.marks.m3==1&&
        p->data->result.marks.m4==1) chk = 1;
    else chk = 0;

    return chk;
}
```

clean_list.h

```
#ifndef CLEAN_LIST_H_INCLUDED
#define CLEAN_LIST_H_INCLUDED
#include <stdlib.h>
#include "structs.h"


void free_head(head *q);
void free_node(node *temp);
void free_list(head *q);

#endif // CLEAN_LIST_H_INCLUDED
```

clean_list.c

```
#include "clean_list.h"


void free_head(head *q){
    free_node(q->first);
    q->first = NULL;
    q->last = NULL;
    free(q);
}


void free_node(node *temp){
    if((temp->next) != NULL) temp->next = NULL;


    free(temp->data->name);
    free(temp->data);
    free(temp);
}


void free_list(head *q){

    node *temp = NULL;
    node *p = NULL;
    int i,n;
```

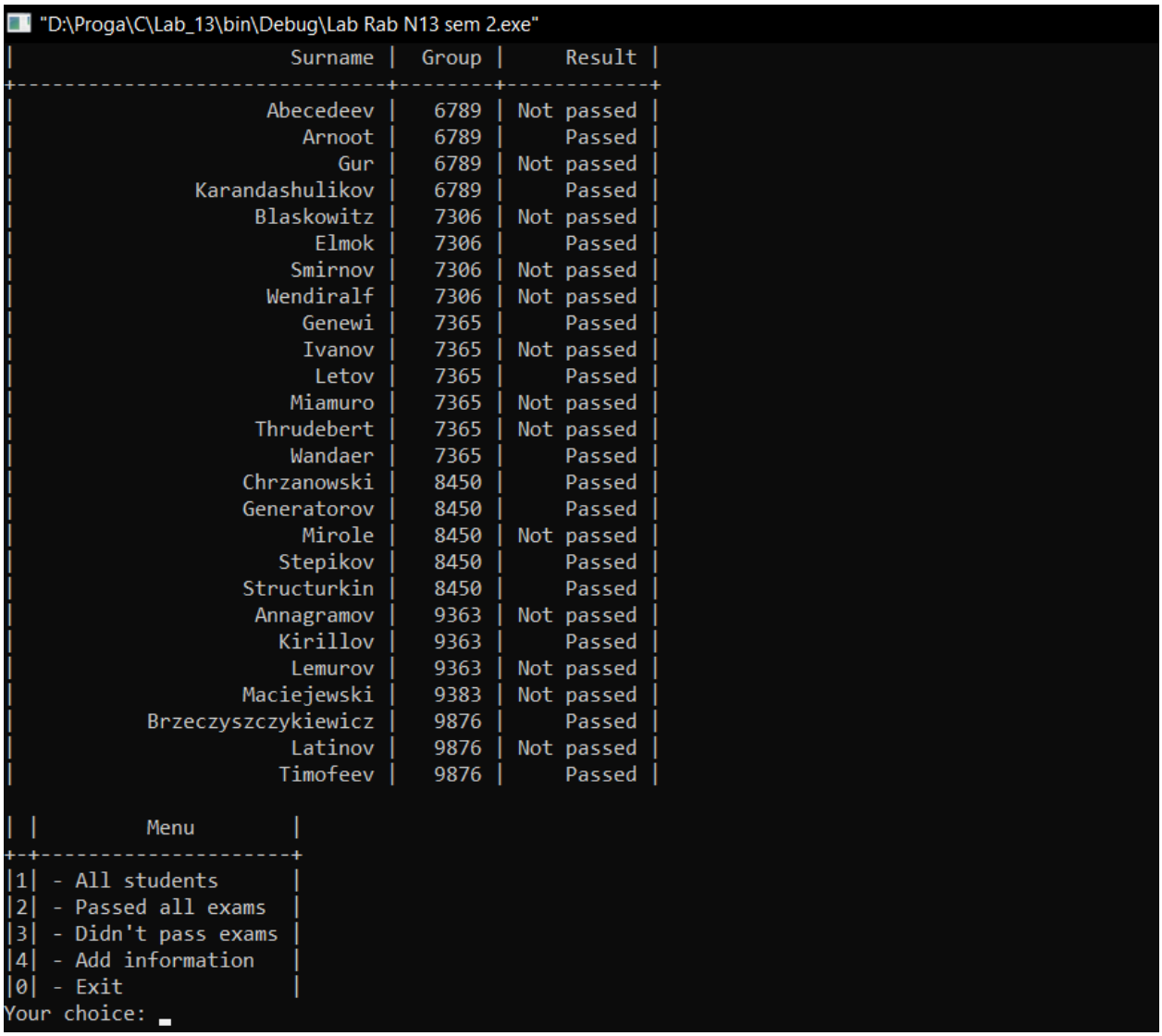
```
    temp = q->first->next;
    n = q->cnt;
    free_head(q);

    //while((temp->next) != NULL){
    for(i=0;i<n-2;i++){
        p = temp;
        temp = temp -> next;
        free_node(p);
    }

    free_node(temp);
}
```

structs.h

```
#ifndef STRUCTS_H_INCLUDED
#define STRUCTS_H_INCLUDED



union total             //Union in 1 byte
{
    unsigned pass;      //==7 if all passed
    struct marks        //Four exams 1-passed,0-didn't
    {
        unsigned char m1:1;
        unsigned char m2:1;
        unsigned char m3:1;
        unsigned char m4:1;
    }marks;
};


struct students
{
    char *name;
    int group;
    union total result;
};
typedef struct students studs;


struct node{

    studs *data;
    struct node *next;
    struct node *prev;
};
typedef struct node node;
```

```
struct head {
    int cnt;
    struct node *first;
    struct node *last;
};
typedef struct head head;


#endif // STRUCTS_H_INCLUDED
```

Пример работы программы



Заключение

Выводы:

При выполнении лабораторной работы были получены практические навыки в разработке алгоритма и написании программы на языке Си. Были получены основные знания о битовых полях.