**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема:   
Генерация и обработка файлов в формате csv.

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Путьков Д.А. |
| Преподаватель |  | Берленко Т.А, |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Путьков Дмитрий Александрович | | |
| Группа 6304 | | |
| Тема работы: Генерация и обработка файлов в формате csv. | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций * Работа с репозиторием * Примеры работы программы * Заключение * Пример работы программы * Список использованных источников * Приложение: Исходный код программы | | |
| Объем пояснительной записки:  33 страницы. | | |
| Дата выдачи задания: 12.03.2017 | | |
| Дата сдачи реферата: 20.04.2017 | | |
| Дата защиты реферата: 20.04.2017 | | |
| Студент |  | Путьков Д.А. |
| Преподаватель |  | Берленко Т.А. |

# **Аннотация**

В данной работе был создан проект на языке программирования C, который позволяет генерировать тестовые данные(csv-таблицы) и видоизменять их согласно заданию. Для функционирования проекта были созданы:

1. Генератор тестов
2. Эталонное решение поставленной задачи
3. Скрипт вызова генератора и сверки эталонного решения с решением студента.

Были созданы и описаны необходимые функции, позволяющие считывать и изменять данные генерируемых таблиц, а также создавать новые. Помимо этого, была проведена работа над оптимизацией исходного кода программы для ускорения ее быстродействия и оптимального использования памяти и ресурсов. Приведено примеры работы программы в виде таблиц формата csv и полное описание исходного кода.

**Содержание**

[**Аннотация** 3](#_Toc483930669)

[**Введение** 5](#_Toc483930670)

[Цель работы 5](#_Toc483930671)

[**Формулировка задачи** 5](#_Toc483930672)

[Содержимое чекера 5](#_Toc483930673)

[Общая постановка задачи 6](#_Toc483930674)

[Параметры 7](#_Toc483930675)

[**Решение задачи** 7](#_Toc483930676)

[Программа генерации тестовых данных 7](#_Toc483930677)

[Создание структуры данных 7](#_Toc483930678)

[Функция создания номеров групп 8](#_Toc483930679)

[Функция создания слова 8](#_Toc483930680)

[Функция создания структуры для одного студента. 8](#_Toc483930682)

[Функция создания списка студентов 9](#_Toc483930683)

[Функция создания первой таблицы 10](#_Toc483930684)

[Функция генерации оценок студента 10](#_Toc483930685)

[Функция создания второй таблицы 11](#_Toc483930686)

[Функция генерации команд для проверяемой программы. 11](#_Toc483930687)

[**Эталонное решение** 12](#_Toc483930688)

[Структура Student 12](#_Toc483930689)

[Функция создания структуры одного студента 13](#_Toc483930690)

[Функция чтения первой таблицы 13](#_Toc483930691)

[Функция нахождения средней оценки студента за экзамен 14](#_Toc483930692)

[Функция чтения второй таблицы 15](#_Toc483930693)

[Функция сортировки номеров групп по возрастанию 16](#_Toc483930694)

[Функция вывода данных в новые таблицы 17](#_Toc483930695)

[Функция создания новых таблиц 17](#_Toc483930696)

[Функция вывода количества отличников 19](#_Toc483930697)

[MainFunction 19](#_Toc483930698)

[Makefile 20](#_Toc483930699)

[Makefile для генерации тестовых данных 21](#_Toc483930700)

[Makefile для проверки 21](#_Toc483930701)

[Скрипт на bash 21](#_Toc483930702)

[**Заключение** 22](#_Toc483930703)

[Пример использования программы 23](#_Toc483930704)

[Список использованных источников 25](#_Toc483930705)

[Исходный код 25](#_Toc483930706)

# **Введение**

## Цель работы

Целью данной работы является создание проекта, позволяющего проверить, корректно ли работает решение поставленной задачи; генерация и обработка файлов в формате csv. Написание программы генератора тестов, эталонного решения задачи и скрипта, связывающих эти программы с проверяемой, оценка ее работы.

**Формулировка задачи**

Для достижения поставленной цели требуется решить следующие задачи:

* Создание эталонного решения - корректное решение задачи (исходный код).
* Создание генератора тестов- данные, на которых и будет тестироваться пользовательское решение.
* Создание чекера - набора файлов с исходным кодом и скрипт, написанный на bash. Последний сравнивает результаты эталонного и пользовательского решений и дает однозначный ответ, верно ли пользователь решил задачу.

### Содержимое чекера

#### generate.c

Генерация тестовых данных.

#### refsol.c

Эталонное решение задачи.

#### Makefile

Цели: удаление объектных и исполняемых файлов, компиляция файлов с исходным кодом.

#### run\_solution.sh

* запуск генерации тестовых данных
* запуск эталонного решения с данным набором тестовых данных
* запуск пользовательского решения с данным набором тестовых данных
* сравнение результатов и вывод ответа, верное ли решение

### Общая постановка задачи

Обработка файла в формате csv (таблица).

На обработку программе подаются две таблицы. Первая таблица содержит информацию о ФИО студента, его Github аккаунте, email'е и номере группы. Вторая таблица содержит фамилию, имя, количество баллов за каждое из пяти заданий экзамена. Требуется реализовать программу, которая:

1. находит общее количество баллов за экзамен для каждого студента;
2. составляет несколько новых таблиц (по количеству номеров групп), в которых содержится ФИО, Github аккаунт, email, общий балл за экзамен. Номер группы должен быть в начале каждой таблицы, таблицы разделяются двумя символами перевода строки. Номер группы - целое число. Таблицы должны следовать в порядке возрастания групп;
3. находит количество студентов, которые получили больше 90% от максимального балла и выводит результат на консоль;
4. сохраняет результат в новом файле.

(число здесь идентифицирует номер команды)

### Параметры

Программа получает параметры из входного потока. Параметры:

* input\_file\_1 - csv файл
* input\_file\_2 - csv файл
* commands

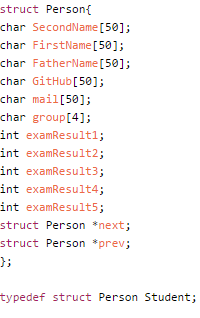
commands - числовой массив неизвестной длины, который хранит в себе последовательность функций обработки входного файла. Массив заканчивается числом 4 - функцией сохранения результата в новом файле.

**Решение задачи**

## Программа генерации тестовых данных

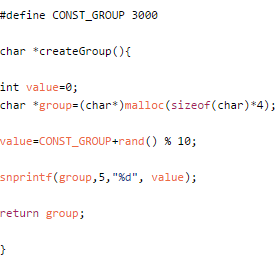
### Создание структуры данных

С помощью типа *struct* объявим новый сложный тип *Student*, для упорядочения генерируемых данных.



### Функция создания номеров групп

Выделим память для номера группы элемента, сгенерируем его значение при помощи функции *rand,* присвоим это значение переменной типа *char \** при помощи функции *snprintf* и вернем его значение.



### Функция создания слова

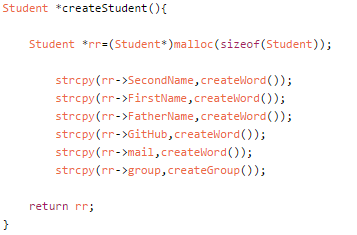
Создадим сначала массив элементов типа char, переменные которого будут представлять из себя буквы латинского алфавита. Далее сгенерируем длину слова при помощи функции *rand*. Выделим память под генерируемое слово, и для каждого из элементов слова будем генерировать букву из нашего алфавита.

Функция возвращает созданное слово.

### 

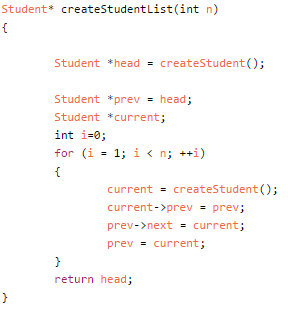
### Функция создания структуры для одного студента.

Создаем указатель на структуру, после чего в каждую ячейку структуры записываем сгенерируемое значение, путем вызова функций *createWord* (для слов) и *createGroup*(для номера группы). Функция возвращает указатель на созданную структуру.



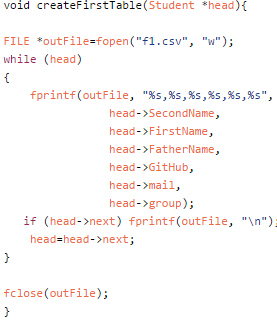
### Функция создания списка студентов

Функция принимает в качестве аргумента указатель на новый элемент *(head).* Присваивая указателям *current* и *prev* соответствующие значения, создаем двусвязный линейный список. Функция возвращает указатель на первую структуру в списке.



### Функция создания первой таблицы

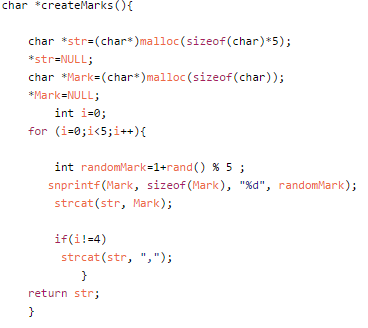
Создадим файл для записи в него данных *FILE \*outFile=fopen("f1.csv", "w");* При помощи функции *fprintf* будем записывать значения текущего элемента списка, от первого и до последнего.



### Функция генерации оценок студента

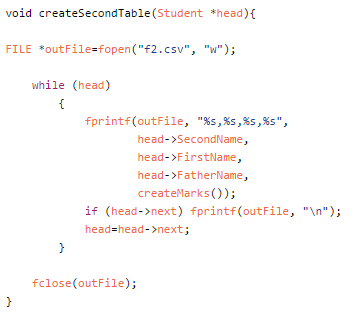
Выделим память для строки символов длиной пять(по количеству оценок).

Создадим цикл, в котором будут генерироваться 5 оценок от 1 до 5 баллов соответственно. При помощи функции *snprintf* эти значения будут приведены к типу char, после чего произойдет конкатенация элемента с запятой, и полученный результат функция вернет.  
 Схематично это выглядит так : ( [1;5]+’ , ’+[1;5]+’ , ’+..+[1;5])



### Функция создания второй таблицы

Данная функция является почти полным аналогом функции создания первой таблицы.

**

### Функция генерации команд для проверяемой программы.

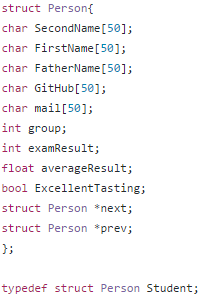
Мы собираемся передать не одну, а несколько совокупностей команд для тестируемой программы. Для этого запустим цикл в цикле и при помощи описанной выше функции *rand* создадим массив команд. Использование цикла в цикле позволит нам сэкономить память, не утруждая себя выделением памяти под двумерный массив.



# **Эталонное решение**

### Структура Student

Для более удобного использования данных, полученных из таблиц, в этой программе мы так же будем использовать двусвязные списки из структур.



### Функция создания структуры одного студента

В эту функцию передается указатель на одну строку из таблицы. Создадим указатель на структуру, выделим память под массив строк. При помощи функции *strtok* разобьем переданную в функцию строку на слова, каждое из них присвоим двумерному массиву, после чего передадим полученные строки в поля структуры. Функция возвращает указатель на структуру.



### Функция чтения первой таблицы

В эту функцию передается файл, с которого считываются строки.

Выделяем память под массив элементов и массив строк. В этой же функции будет параллельно создаваться двусвязный список.

1. Считывание из файла строк до *\n* при помощи функции *fgets*
2. Присваивание получившейся строки в двумерный массив.
3. Создание списка, использования функции *createStudent* с передачей в нее текущей строки
4. Работа с указателями, благодаря чему и создается двусвязный список.
5. Освобождение памяти.
6. Функция возвращает указатель на первый элемент списка.



### Функция нахождения средней оценки студента за экзамен

В функцию передается указатель на текущий элемент списка, а также строка, в которой находятся оценки студента.

Используем сравнение фамилии студента, чтобы удостовериться, что это данные именно того студента. Фамилию получим из строки при помощи функции *strtok*.

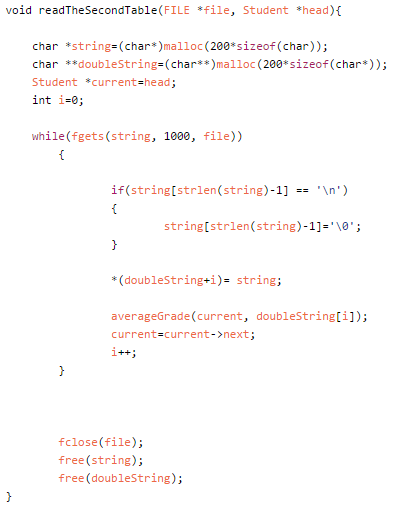
После того, как мы удостоверились в правильности данных, разобьем строку на лексемы, при помощи функции *atoi*  приведем полученные оценки к типу *int* и найдем сумму оценок. Присвоим это значение полю *examResult.*



### Функция чтения второй таблицы

Функция принимает файл, из которого будут считываться данные и указатель на первый элемент уже созданного списка.

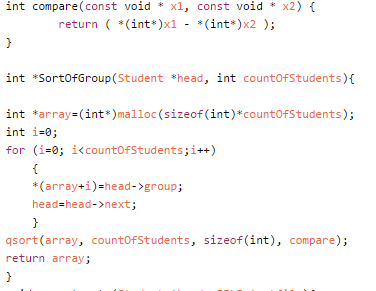
Аналогично функции для первой таблицы, мы считываем данные, добавляя только вызов вышеописанной функции считывания оценок и приведения их к типу int.



### Функция сортировки номеров групп по возрастанию

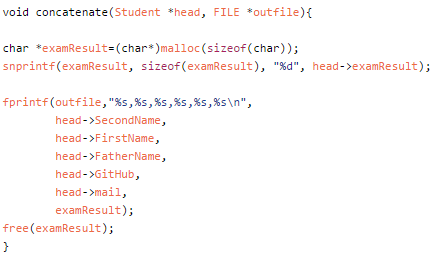
В этой функции реализуется метод быстрой сортировки *qsort.* Функция принимает на вход количество студентов, создает цикл,в котором функция *qsort* будет работать с полем структуры *group*, и в массив, под который мы выделили память, записывает номера групп.

Функция возвращает массив.



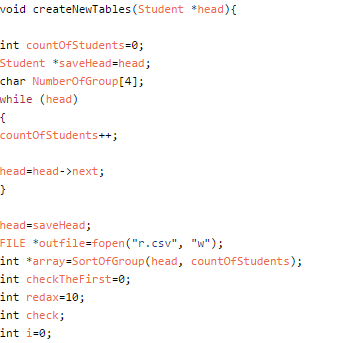
### Функция вывода данных в новые таблицы

Функция принимает на вход указатель на первый элемент списка и файл, в который будут выводиться новые таблицы. Функция является аналогом функции вывода таблиц в *generate.*



### Функция создания новых таблиц

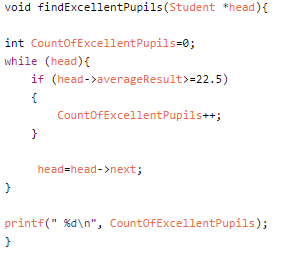
Функция принимает на вход указатель на первый элемент списка. При помощи цикла *while* считывается количество студентов. Далее создается файл, в который будет записана таблица, и в новом цикле будут выводиться данные студентов групп(при помощи вызова функции *concatenate*), идущих по возрастанию. Между студентами будет ставиться *\n*, между номерами групп : *\n\n*. Для этого сделана соответствующая проверка.





### Функция вывода количества отличников

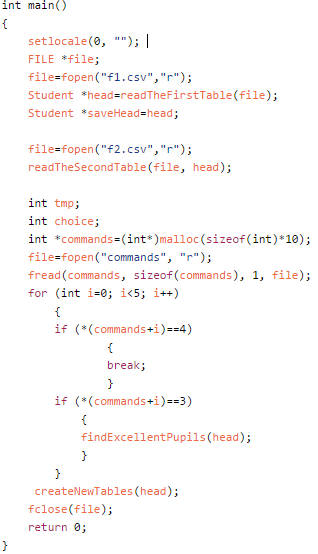
В функцию передается указатель на первый элемент списка, после чего идет сравнение оценки студента с 22.5(25\*90%) . количество студентов выводится на экран.



## MainFunction

В функции main подключена русский алфавит на случай, если таблицы будут на русском. Объявляются переменные с таблицами для чтения, вызов функций чтения таблиц.

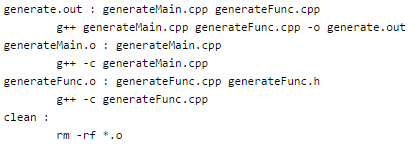
После этого начинает выполняться массив команд, считываемый из отдельного файла *commands*. В зависимости от значения элемента массива вызывается нужная функция.



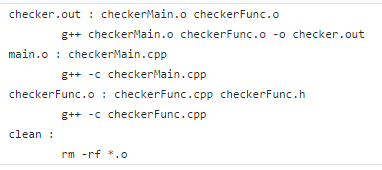
## Makefile

Создадим поочерёдно объектные файлы функций и главной функции, после скомпилируем их вместе, также предусмотрена функция очистки после сборки проекта.

### Makefile для генерации тестовых данных



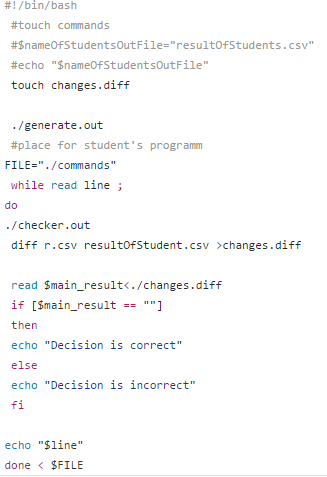
### Makefile для проверки



## Скрипт на bash

В скрипте при помощи команды touch создается файл *changes.diff*, в который будет записываться разница между выводом тестируемой задачи и правильным выводом. Этот файл при корректном решении пуст.

После этого вызывается исполняемый файл *generate.out*-генерация тестов. Затем используется алгоритм считывания из файла построчно: для каждой строки из *commands* будет вызываться *checker.out*, затем тестируемая программа. После этого идет сравнение файлов, в которых записаны выводы программ, и сравнение записывается в *changes.diff.*   
Далее идет проверка на пустой файл. Если пустой-тест пройден.



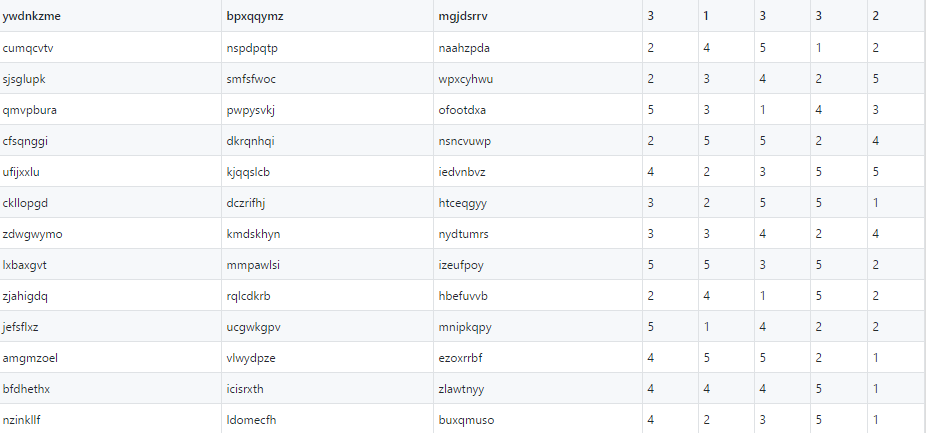
# **Заключение**

Выполнив данную курсовую работу, были получены навыки работы с такими функциями как *atoi, strtok, fnprintf, rand*; была оптимизирована и улучшена работа с вводом и выводом файлов, работа со структурами, циклами и списками. Были освоены навыки считывания данных из файла с различными расширениями. Стала понятной работа чекеров на платформе stepik. Получены навыки написания скриптов на *bash*, перенаправление вывода и вызов исполняемых файлов как последовательно, так и параллельно.

### Пример использования программы

На вход программе проверки подаются две таблицы, которые изначально были сгенерированы.





По завершении работы программы, ответ будет записан в файл, студенты будут отсортированы по группам в порядке возрастания. Таблица будет иметь вид :  


### Список использованных источников

* 1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб: Издательство «Невский Диалект», 2001. 352 с.
  2. UNIX. Программное окружение / Керниган Б., Пайк Р. СПб: Символ Плюс, 2003. 416с.

### Исходный код

**Исходный код Ref\_solution**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <locale.h>

#include <stdbool.h>

struct Person{

char SecondName[50];

char FirstName[50];

char FatherName[50];

char GitHub[50];

char mail[50];

int group;

int examResult;

float averageResult;

bool ExcellentTasting;

struct Person \*next;

struct Person \*prev;

};

typedef struct Person Student;

Student \*createStudent(char \*smallString){

Student \*rr=(Student\*)malloc(sizeof(Student));

char \*\*str2=(char\*\*)malloc(100\*sizeof(char\*));

int i=0;

smallString=strtok(smallString, ",");

while (smallString!=NULL){

\*(str2+i)=smallString;

i++;

smallString=strtok(NULL, ",");

}

strcpy(rr->SecondName, \*(str2));

strcpy(rr->FirstName, \*(str2+1));

strcpy(rr->FatherName, \*(str2+2));

strcpy(rr->GitHub, \*(str2+3));

strcpy(rr->mail, \*(str2+4));

rr->group=atoi(\*(str2+5));

rr->next = NULL;

rr->prev = NULL;

free(str2);

return rr;

}

Student \*readTheFirstTable(FILE \*file){

char \*string=(char\*)malloc(200\*sizeof(char));

char \*\*doubleString=(char\*\*)malloc(200\*sizeof(char\*));

Student \*head;

Student \*prev;

Student \*current;

int i=0;

while(fgets(string, 1000, file))

{

if(string[strlen(string)-1] == '\n')

{

string[strlen(string)-1]='\0';

}

\*(doubleString+i)= string;

if (i==0){head=createStudent(doubleString[0]); prev= head;}

else

{

current = createStudent(doubleString[i]);

current->prev = prev;

prev->next = current;

prev = current;

}

i++;

}

free(string);

free(doubleString);

fclose(file);

return head;

}

void averageGrade(Student \*current, char \*doubleString){

doubleString=strtok(doubleString, ",");

int i=1;

while ((\*(current->SecondName)!=\*doubleString) || (!current))

current=current->next;

if(!current) return;

while (doubleString!=NULL){

if(i<3)

{

doubleString=strtok(NULL,","); }

else

{

current->examResult+=atoi(doubleString);

doubleString=strtok(NULL,",");

}

i+=1;

}

current->averageResult=current->examResult;

}

void readTheSecondTable(FILE \*file, Student \*head){

char \*string=(char\*)malloc(200\*sizeof(char)); // ñîçäàåì äâóìåðíûé ìàññèâ, êîòîðûé áóäåò ñ÷èòûâàòü íàøè äàííûå

char \*\*doubleString=(char\*\*)malloc(200\*sizeof(char\*));

Student \*current=head;

int i=0;

while(fgets(string, 1000, file))

{

if(string[strlen(string)-1] == '\n')//ïðîâåðÿåì ÿâëÿåòñÿ ëè ïîñëåäíèé ýëåìåíò â ñòðîêå ñèìâîëîì å¸ îêîí÷àíèÿ

{

string[strlen(string)-1]='\0';

}

\*(doubleString+i)= string;

averageGrade(current, doubleString[i]);

current=current->next;

i++;

}

fclose(file);

free(string);

free(doubleString);

}

int compare(const void \* x1, const void \* x2) {

return ( \*(int\*)x1 - \*(int\*)x2 );

}

int \*SortOfGroup(Student \*head, int countOfStudents){

int \*array=(int\*)malloc(sizeof(int)\*countOfStudents);

int i=0;

for (i=0; i<countOfStudents;i++)

{

\*(array+i)=head->group;

head=head->next;

}

qsort(array, countOfStudents, sizeof(int), compare);

return array;

}

void concatenate(Student \*head, FILE \*outfile){

char \*examResult=(char\*)malloc(sizeof(char));

snprintf(examResult, sizeof(examResult), "%d", head->examResult);

fprintf(outfile,"%s,%s,%s,%s,%s,%s\n",

head->SecondName,

head->FirstName,

head->FatherName,

head->GitHub,

head->mail,

examResult);

free(examResult);

}

void createNewTables(Student \*head){

int countOfStudents=0;

Student \*saveHead=head;

char NumberOfGroup[4];

while (head)

{

countOfStudents++;

head=head->next;

}

head=saveHead;

FILE \*outfile=fopen("r.csv", "w");

int \*array=SortOfGroup(head, countOfStudents);

int checkTheFirst=0;

int redax=10;

int check;

int i=0;

for (i=0; i<countOfStudents; i++){

head=saveHead;

if (check== \*(array+i))

continue;

while (head){

if (head->group== \*(array+i)){

check=head->group;

if (checkTheFirst==0)

{

snprintf(NumberOfGroup, 5, "%d", head->group);

fprintf(outfile,"%s\n", NumberOfGroup);

checkTheFirst=1;

}

concatenate(head, outfile);

}

head=head->next;

}

checkTheFirst=0;

if (i+1!=countOfStudents)

fprintf(outfile,"\n\n");

}

fclose(outfile);

}

void findExcellentPupils(Student \*head){

int CountOfExcellentPupils=0;

while (head){

if (head->averageResult>=22.5)

{

CountOfExcellentPupils++;

}

head=head->next;

}

printf(" %d\n", CountOfExcellentPupils);

}

int main()

{

setlocale(0, "");

FILE \*file;

file=fopen("f1.csv","r");

Student \*head=readTheFirstTable(file);

Student \*saveHead=head;

file=fopen("f2.csv","r");

readTheSecondTable(file, head);

int tmp;

int choice;

int \*commands=(int\*)malloc(sizeof(int)\*10);

file=fopen("commands", "r");

fread(commands, sizeof(commands), 1, file);

for (int i=0; i<5; i++)

{

if (\*(commands+i)==4)

{

break;

}

if (\*(commands+i)==3)

{

findExcellentPupils(head);

}

}

createNewTables(head);

fclose(file);

return 0;

}

**Исходный код программы, генерирующей данные**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <stdbool.h>

#define CONST\_GROUP 3000

char \*createGroup(){

int value=0;

char \*group=(char\*)malloc(sizeof(char)\*4);

value=CONST\_GROUP+rand() % 10;

snprintf(group,5,"%d", value);

return group;

}

struct Person{

char SecondName[50];

char FirstName[50];

char FatherName[50];

char GitHub[50];

char mail[50];

char group[4];

int examResult1;

int examResult2;

int examResult3;

int examResult4;

int examResult5;

struct Person \*next;

struct Person \*prev;

};

typedef struct Person Student;

char \*createWord(){

char alphabet[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};

int randomLengthOfWord=0;

randomLengthOfWord=1+ rand() %14;

char \*words=(char\*)malloc(sizeof(char)\*randomLengthOfWord);

int i=0;

for (i=0; i<8; i++){

int randomAlphabet= rand() % 26;

\*(words+i)=alphabet[randomAlphabet];

}

return words;

}

Student \*createStudent(){

Student \*rr=(Student\*)malloc(sizeof(Student));

strcpy(rr->SecondName,createWord());

strcpy(rr->FirstName,createWord());

strcpy(rr->FatherName,createWord());

strcpy(rr->GitHub,createWord());

strcpy(rr->mail,createWord());

strcpy(rr->group,createGroup());

return rr;

}

Student\* createStudentList(int n)

{

Student \*head = createStudent();

Student \*prev = head;

Student \*current;

int i=0;

for (i = 1; i < n; ++i)

{

current = createStudent();

current->prev = prev;

prev->next = current;

prev = current;

}

return head;

}

void createFirstTable(Student \*head){

FILE \*outFile=fopen("f1.csv", "w");

while (head)

{

fprintf(outFile, "%s,%s,%s,%s,%s,%s",

head->SecondName,

head->FirstName,

head->FatherName,

head->GitHub,

head->mail,

head->group);

if (head->next) fprintf(outFile, "\n");

head=head->next;

}

fclose(outFile);

}

char \*createMarks(){

char \*str=(char\*)malloc(sizeof(char)\*5);

\*str=NULL;

char \*Mark=(char\*)malloc(sizeof(char));

\*Mark=NULL;

int i=0;

for (i=0;i<5;i++){

int randomMark=1+rand() % 5 ;

snprintf(Mark, sizeof(Mark), "%d", randomMark);

strcat(str, Mark);

if(i!=4)

strcat(str, ",");

}

return str;

}

void createSecondTable(Student \*head){

FILE \*outFile=fopen("f2.csv", "w");

while (head)

{

fprintf(outFile, "%s,%s,%s,%s",

head->SecondName,

head->FirstName,

head->FatherName,

createMarks());

if (head->next) fprintf(outFile, "\n");

head=head->next;

}

fclose(outFile);

}

void createCommands(void){

int countOfCommands= 1+rand() % 5;

int \*commands=(int\*)malloc(sizeof(int)\*countOfCommands);

FILE \*file=fopen("commands", "w");

for (int j=0; j<countOfCommands; j++)

{

for (int i=0; i<countOfCommands-1; i++)

{

\*(commands+i)=1+ rand() % 4;

}

\*(commands+countOfCommands)=4;

fwrite(commands,sizeof(commands),1, file);

if (countOfCommands-1!=j)

fprintf(file, "\n");

}

fclose(file);

free(commands);

}

int main()

{

char alphabet[26]={'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z'};

srand( (unsigned)time( NULL ) );

int countOfStud=0;

countOfStud=rand() %60;

Student \*head=createStudentList(countOfStud);

createFirstTable(head);

createSecondTable(head);

createCommands();

}