

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Программирование»**  
**Тема: Программа для работы с csv файлами (таблицами)**

Студент гр. 6304

\_\_\_\_\_

Рыбин А.С.

Преподаватель

\_\_\_\_\_

Берленко Т.А.

Санкт-Петербург

2017

# ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Рыбин А.С.

Группа 6304

Тема работы: Программа для работы с csv файлами (таблицами)

Исходные данные:

Курсовой проект должен включать программу генерации csv файлов (таблиц) и команд для работы с ними, эталонное решение, и скрипт для сравнения результатов их работы с пользовательским решением.

Содержание пояснительной записки:

- Аннотация;
- Содержание;
- Введение;
- Описание основных функций;
- Работа с make;
- Работа с репозиторием;
- Примеры работы программы;
- Заключение;
- Приложение (листинг программы);

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 12.03.2017

Дата сдачи реферата: 31.05.2017

Дата защиты реферата: 31.05.2017

Студент \_\_\_\_\_

Рыбин А.С.

Преподаватель \_\_\_\_\_

Берленко Т.А.

## **АННОТАЦИЯ**

Требуется создать проект на языке Си, который включает в себя следующие подпрограммы: генератор исходных данных (csv таблицы и файл с командами для работы с ними), эталонное решение, скрипт, выполняющий сборку проекта вместе с пользовательским решением и сравнения результатов их работы на некотором количестве наборов команд на сгенерированных данных. Далее в пояснительной записке представлены результаты работы программы на сгенерированных тестовых данных в виде скриншотов, а также исходный код.

## **SUMMARY**

The creating project on ANSI language, that including generator of input data (two csv tables and file with some sets of commands), standard solution and bash script, that compiles the project containing user solution and checks results their work on some generated sets of commands. Results of working program on generated test data are shown as screenshots. Source code is also included.

# СОДЕРЖАНИЕ

1 . ВВЕДЕНИЕ.....	5
1.1 <b>Цель работы</b> .....	5
1.2 <b>Формулировка задания</b> .....	5
2 . ОСНОВНЫЕ ФУНКЦИИ.....	7
2.1. <b>Пользовательские структуры данных и определения</b> .....	7
2.2. <b>Функции генератора</b> .....	8
2.3. <b>Функции эталонного решения</b> .....	9
3. РАБОТА С МАКЕ.....	12
4. РАБОТА С РЕПОЗИТОРИЕМ.....	13
5. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ.....	14
5.1. <b>Генерация тестовых данных</b> .....	14
5.2. <b>Эталонное решение</b> .....	15
5.3. <b>Работа скрипта</b> .....	17
6. ЗАКЛЮЧЕНИЕ.....	19
7. ПРИЛОЖЕНИЕ.....	20
7.1 <b>Generate</b> .....	20
7.1.1. <i>Generate_Header.h</i> .....	20
7.1.2. <i>Generate_Fun.cpp</i> .....	22
7.1.3. <i>Generate.cpp</i> .....	23
7.2. <b>Refsol</b> .....	24
7.2.1. <i>Refsol_Header.h</i> .....	24
7.2.2. <i>Refsol_Fun.cpp</i> .....	26
7.2.3. <i>Refsol.cpp</i> .....	30
7.3. <b>Run_sh, makefile</b> .....	32
7.3.1. <i>Run.sh</i> .....	32
7.3.2. <i>Makefile</i> .....	33

# 1. ВВЕДЕНИЕ

## 1.1 Цель работы

Написание набора программ, позволяющего проверить работу пользовательского решения на случайно сгенерированных данных с помощью эталонного решения.

Для достижения цели должны быть решены следующие задачи:

- реализация подпрограмм генератор,
- образцовое решение,
- скрипт для их компиляции и сравнения пользовательского и эталонного результатов работы.

## 1.2 Формулировка задания

Обработка файла в формате csv (таблица).

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

1. удаляет повторяющиеся строки в таблицах;
2. составляет несколько новых таблиц (по количеству номеров групп), в которых содержится ФИО, Github аккаунт, email, оценка. Номер группы должен быть в начале каждой таблицы, таблицы разделяются двумя символами перевода строки;
3. находит количество студентов, которые получили максимальный балл и выводит результат на консоль;
4. находит количество студентов, которые написали хуже, чем 60% от максимального балла и выводит результат на консоль;
5. 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 5 - функцией сохранения результата в новом файле.

В случае, если программа получила некорректные параметры, то:

- операционной системе возвращается ненулевой код возврата (`return` в `main`)
- не создается выходного файла
- выводится сообщение об ошибке “Fail with <имя параметра>”.

## 2. ОСНОВНЫЕ ФУНКЦИИ

### 1. Пользовательские структуры данных и определения

```
typedef struct Table
{
    char name[MAX_LENGTH_TOKEN]
    char surname[MAX_LENGTH_TOKEN]
    char patronymic[MAX_LENGTH_TOKEN];
    char GitHub_account[MAX_LENGTH_TOKEN];
    char Email[MAX_LENGTH_TOKEN];
    int group;
    int exam_result;
    int flag;
} Table;
```

Данная пользовательская структура данных определена для того, чтобы при генерации таблиц, каждую сгенерированную строку записывать в одну переменную, а позже получать доступ к каждому полю. Также для последующего использования в эталонном решении для удобного чтения таблиц по строкам и последующего дробления по разделителям и записи в каждое поле структуры.

```
typedef struct Vector
{
    int len;
    Table vector_ptr[MAX_LINES];
} Vector;
```

Данная пользовательская структура данных определена в качестве итератора по массиву структур типа Table. Она содержит длину массива и указателей на первый элемент.

MAX\_LENGTH\_TOKEN – максимальная длина генерируемой лексемы

MIN\_LENGTH\_TOKEN - минимальная длина генерируемой лексемы

MAX\_LINES – максимальное количество строк в генерируемых таблицах

MIN\_LINES - минимальное количество строк в генерируемых таблицах

MAX\_LENGTH\_ARRAY\_COMMANDS – максимальная длина набора команд

MAX\_GROUP – максимальное значение генерируемой группы

MAX\_EXAM\_RESULT – максимальное значение генерируемой оценки за экзамен

INPUT1 – input\_file\_1.csv

INPUT2 – input\_file\_2.csv

COMMANDS – commands.txt

RESULT – results.csv

## 2. Функции генератора

```
void GenerateToken(char* random_token)
{
    int len = MIN_LENGTH_TOKEN + rand() % (MAX_LENGTH_TOKEN - MIN_LENGTH_TOKEN);

    for (int i = 0; i < len; i++)
        random_token[i] = (char)(97 + rand() % 25);
    random_token[len] = '\0';
}
```

Данная функция генерирует случайную лексему длины не менее MIN\_LENGTH\_TOKEN и не более MAX\_LENGTH\_TOKEN и записывает по переданному в функцию указателю.

```
void PutRandomCommands(FILE* commands_list)
{
    int len = rand() % MAX_LENGTH_ARRAY_COMMANDS;

    for (int i = 0; i < len; i++)
        fprintf(commands_list, "%d ", -1 + rand() % 8);

    fprintf(commands_list, "\n");
}
```

Данная функция генерирует набор случайных команд длины от 0 до MAX\_LENGTH\_ARRAY\_COMMANDS, причём команды имеют значения от -1 до 6, при этом идёт запись данного набора в файл, указатель на который передан в качестве аргумента функции.

```
Vector GenerateRandomVector()
{
    Vector info;
    memset(info.vector_ptr, 0, sizeof(info.vector_ptr));

    info.len = MIN_LINES + rand() % (MAX_LINES - MIN_LINES);

    for (int i = 0; i < info.len; i++)
    {
        GenerateToken(info.vector_ptr[i].Email);
        GenerateToken(info.vector_ptr[i].GitHub_account);
        GenerateToken(info.vector_ptr[i].name);
        GenerateToken(info.vector_ptr[i].surname);
        GenerateToken(info.vector_ptr[i].patronymic);
        info.vector_ptr[i].group = 1 + rand() % MAX_GROUP;
        info.vector_ptr[i].exam_result = 1 + rand() % MAX_EXAM_RESULT;
    }
    return info;
}
```

Данная функция генерирует массив структур типа Table, в полях каждого элемента массива записываются случайные лексемы функцией GenerateToken. В поля, содержащие группу и результат экзамена записываются случайные значения от 1 до соответственно MAX\_GROUP и MAX\_EXAM\_RESULT.



## 2.1. Функции эталонного решения

```
void ReadTable1(FILE* stream, Vector* data)
{
    int count = 0;
    char buf[MAX_LENGTH_TOKEN * 5 + 10];
    memset(data->vector_ptr, 0, sizeof(data->vector_ptr));

    /* Skip Header */
    fscanf(stream, "%*s");
    if (feof(stream))
    {
        printf("Error reading first table");
        data->len = -1;
        return;
    }

    while (count < MAX_LINES)
    {
        fscanf(stream, "%s", buf);
        if (feof(stream))
        {
            data->len = count;
            return;
        }

        if (!ParseString(&(data->vector_ptr[count]), buf, 0))
        {
            printf("Error reading first table");
            data->len = -1;
            return;
        }
        count++;
    }
}
```

Данная функция читает из файла с первой таблицей, указатель на который передан в качестве аргумента. Данные записываются в массив структур типа Table с помощью итератора Vector. Чтение происходит построчно, а разбиение данных осуществляется функцией ParseString с параметром 0, что значит разбиение для первой таблицы. Аналогично работает функция чтения из второго файла со второй таблицей, за исключением того, что функция ParseString вызывается с параметром 1, что значит разбиение для второй таблицы. Так же происходит проверка на корректность заполнения файла данными.

```
void CombineTables(Vector* data1, Vector* data2, Vector* data)
{
    if (data1->len < 0 || data2->len < 0 || data1->len != data2->len)
    {
        data->len = -1;
        return;
    }

    memset(data->vector_ptr, 0, sizeof(data->vector_ptr));
    data->len = data1->len;

    for (int i = 0; i < data->len; i++)
    {
        if (strcmp(data1->vector_ptr[i].name, data2->vector_ptr[i].name) != 0
            || strcmp(data1->vector_ptr[i].surname, data2->vector_ptr[i].surname) != 0)
        {
            return;
        }
    }
}
```

```

    {
        data->len = -1;
        return ;
    }

    strcpy(data->vector_ptr[i].name,data1->vector_ptr[i].name);
    strcpy(data->vector_ptr[i].surname,data1->vector_ptr[i].surname);
    strcpy(data->vector_ptr[i].patronymic,data1->vector_ptr[i].patronymic);
    strcpy(data->vector_ptr[i].GitHub_account,data1->vector_ptr[i].GitHub_account);
    strcpy(data->vector_ptr[i].Email,data1->vector_ptr[i].Email);
    data->vector_ptr[i].group = data1->vector_ptr[i].group;
    data->vector_ptr[i].exam_result = data2->vector_ptr[i].exam_result;
}

```

Данная функция объединяет данные полученные из первой и второй таблицы в один массив для более удобного доступа. При это происходит проверка на корректность заполнения первого и второго массивов соответственно с первой и второй таблицами.

```

void CheckBadResults(Vector* data)
{
    int count = 0;
    for (int i = 0; i < data->len; i++)
    {
        if ((data->vector_ptr[i].exam_result <= (int)(0.6 * MAX_EXAM_RESULT))
            && (data->vector_ptr[i].flag != 1))
            count++;
    }
    printf("Students with mark less then 60 percents: %d\n", count);
}

void CheckGoodResults(Vector* data)
{
    int count = 0;
    for (int i = 0; i < data->len; i++)
    {
        if ((data->vector_ptr[i].exam_result == MAX_EXAM_RESULT)
            && (data->vector_ptr[i].flag != 1))
            count++;
    }
    printf("Students with max mark: %d\n", count);
}

```

Данные функции подсчитывают и выводят на экран количество элементов, у которых поле exam\_result отвечает соответственно меньше или равно чем  $0.6 * \text{MAX\_EXAM\_RESULT}$  и равно  $\text{MAX\_EXAM\_RESULT}$ .

```

void CheckRepeats(Vector* data)
{
    for (int i = 0; i < data->len - 1; i++)
        if(memcmp(&(data->vector_ptr[i]),
            &(data->vector_ptr[i + 1]),sizeof(Table)) == 0)
            data->vector_ptr[i].flag = 1;
}

```

Данная функция сравнивает поочерёдно сравнивает два соседних элемента передаваемого массива и в случае совпадения ставит в поле flag 1, что означает повторение. Требуется передавать отсортированный массив.

```

void CheckByGroup(Vector* data)
{
    for (int i = 0; i < data->len; i++)
    {
        if (data->vector_ptr[i].group != data->vector_ptr[i + 1].group)
        {
            if (data->vector_ptr[i].flag == 1)    // If repeats then flag saves
                continue;
            data->vector_ptr[i].flag = -1;
        }
        else if (data->vector_ptr[i + 1].flag == 1)
            //If next is repeat then current student is last
            data->vector_ptr[i].flag = -1;
    }
}

```

Данная функция сравнивает поочерёдно два соседних элемента переданного массива и в случае несовпадения групп ставит в поле flag значение -1, что означает последний в группе.

```

void SaveResults(Vector* data)
{
    FILE* result = fopen(RESULT, "w");
    fprintf(result, RESULT_HEADER);

    for (int i = 0; i < data->len; i++)
    {
        if (data->vector_ptr[i].flag == 1)
            continue;

        fprintf(result, "%d,%s,%s,%s,%s,%s,%d\n",
            data->vector_ptr[i].group,
            data->vector_ptr[i].name,
            data->vector_ptr[i].surname,
            data->vector_ptr[i].patronymic,
            data->vector_ptr[i].GitHub_account,
            data->vector_ptr[i].Email,
            data->vector_ptr[i].exam_result);

        if (data->vector_ptr[i].flag == -1)
            fprintf(result, "\n\n");
    }

    fclose(result);
}

```

Данная функция сохраняет результат работы программы в новом файле заданном RESULT. При этом проверяется значение поля flag у каждого элемента: в случае 1(повторения) элемент пропускается, а в случае -1(последний в группе) после данного элемента печатается два символа перевода строки.

### 3. РАБОТА С MAKE

- Создадим главную цель *main*: со следующими **зависимостями**: *Generate*, *Refsol*, *Usersol.cpp*, где *Usersol.cpp* файла, содержащий пользовательское решение.
- Перечень инструкций для данных целей будет: **компиляция генератора** *gcc Generate.cpp Generate\_Fun.cpp -o Generator*, **компиляция эталонного решения** *gcc Refsol.cpp Refsol\_Fun.cpp -o Refsol*, **компиляция пользовательского решения** *gcc Usersol.cpp -o Usersol*, **удаление файла с пользовательским решением из папки с исходным кодом чекера** *rm Usersol.cpp*
- Если пользовательское решение **никак не определено**, то происходит создание файла *Usersol.cpp* путём копирования *Refsol.cpp*: *touch Usersol.cpp, cat Refsol.cpp > Usersol.cpp*. И последующая компиляция вместе с *Refsol\_Fun.cpp*: *gcc Usersol.cpp Refsol\_Fun.cpp*.

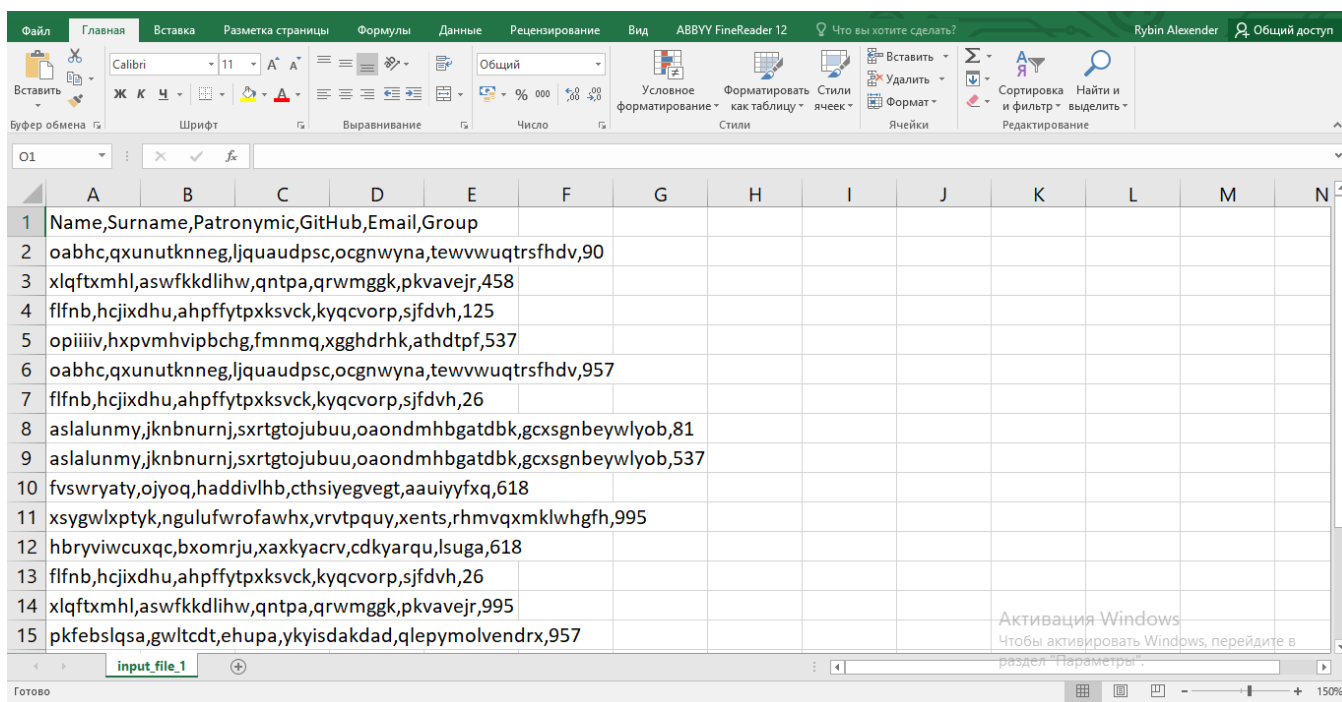
#### 4. РАБОТА С РЕПОЗИТОРИЕМ

- Выполним настройку репозитория: *git config user.name <name>*, *git config user.email <email>*
- Выполним клонирование репозитория: *git clone <URL>*
- Создадим новые ветки для каждой части проекта: *git checkout -b <branch\_name>*
- Добавим в рабочую область каждой ветки соответствующие файлы подпроектов: *git add \*.cpp \*.h*
- Создадим новый слепок состояния на каждой ветке соответственно: *git commit -m <name>*
- Отправим изменения на удаленный сервер: *git push origin*
- С каждой ветки выполним запрос на слияние с веткой *master*

## 5. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

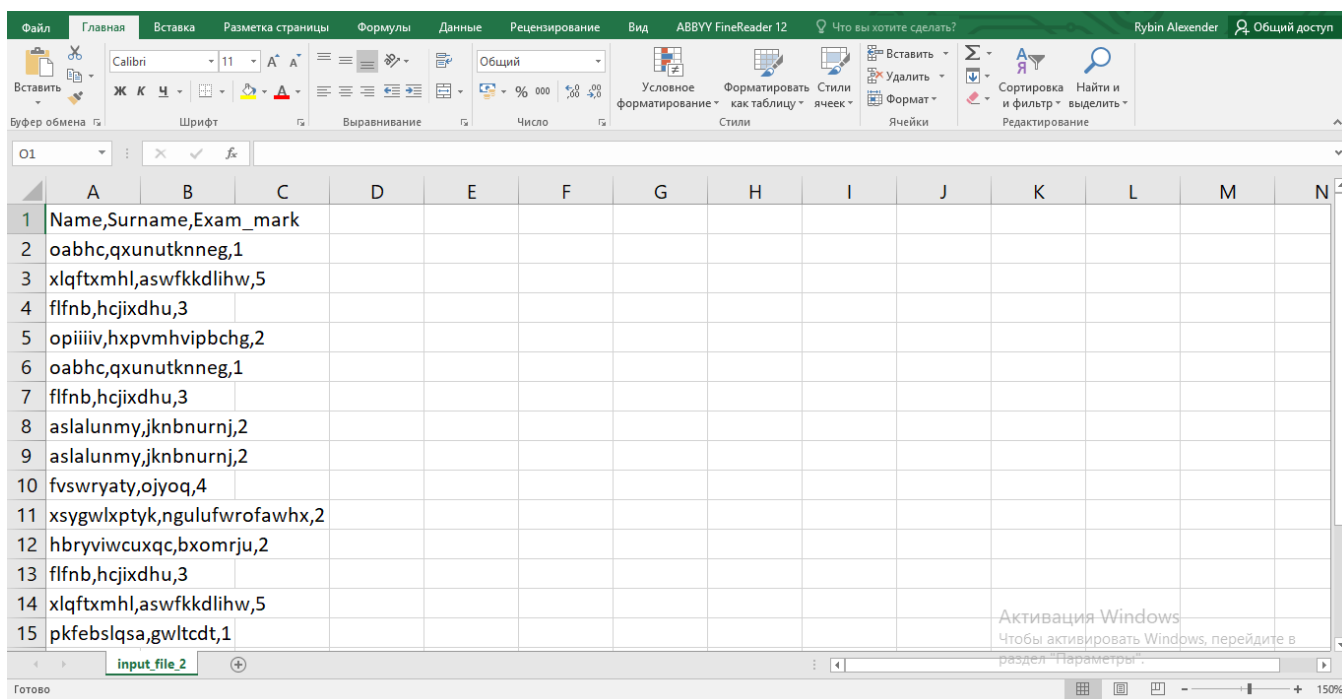
### 5.1. Генерация тестовых данных

Пример генерации двух csv таблиц и файла с набором команд.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Name,Surname,Patronymic,GitHub,Email,Group													
2	oabhc,qxunutknneg,ljquaudpsc,ocgnwyna,tewvwuqtrsfdhv,90													
3	xlqftxmhl,aswfkdkdlihw,qntpa,qrwmggk,pkvavejr,458													
4	flfnb,hcjixdhu,ahpffypxksvck,kyqcvorp,sjfdvh,125													
5	opiiiiiv,hxpvmhvipbchg,fmnmq,xgghdrhk,athdtpf,537													
6	oabhc,qxunutknneg,ljquaudpsc,ocgnwyna,tewvwuqtrsfdhv,957													
7	flfnb,hcjixdhu,ahpffypxksvck,kyqcvorp,sjfdvh,26													
8	aslalunmy,jknbnurnj,sxrtgtojubuu,oaondmhbgatdbk,gcxsgnbeywlyob,81													
9	aslalunmy,jknbnurnj,sxrtgtojubuu,oaondmhbgatdbk,gcxsgnbeywlyob,537													
10	fvswryaty,ojoyq,haddivlhb,cthsiyegvegt,aauiyyfxq,618													
11	xsyglwlpkyk,ngulufwrofawhx,vrvtpquy,xents,rhmvqxmklwhgfh,995													
12	hbryviwcuxqc,bxomrju,xaxkyacrv,cdkyarqu,lsuga,618													
13	flfnb,hcjixdhu,ahpffypxksvck,kyqcvorp,sjfdvh,26													
14	xlqftxmhl,aswfkdkdlihw,qntpa,qrwmggk,pkvavejr,995													
15	pkfebslqsa,gwltcdt,ehupa,ykysidakdad,qlpeymolvendr,957													

Рис 1. Input\_file\_1.csv



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Name,Surname,Exam_mark													
2	oabhc,qxunutknneg,1													
3	xlqftxmhl,aswfkdkdlihw,5													
4	flfnb,hcjixdhu,3													
5	opiiiiiv,hxpvmhvipbchg,2													
6	oabhc,qxunutknneg,1													
7	flfnb,hcjixdhu,3													
8	aslalunmy,jknbnurnj,2													
9	aslalunmy,jknbnurnj,2													
10	fvswryaty,ojoyq,4													
11	xsyglwlpkyk,ngulufwrofawhx,2													
12	hbryviwcuxqc,bxomrju,2													
13	flfnb,hcjixdhu,3													
14	xlqftxmhl,aswfkdkdlihw,5													
15	pkfebslqsa,gwltcdt,1													

Рис 2. Input\_file\_2.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	-1	1	1	1	-1	-1	4						
2	-1	1	0	3	4	4	1	2	5				
3	4	3	3										
4	1	3	2	5	4	1	3	5					
5	2	1	4	1	5								
6	1	4	2	5	3	4	1	2					
7	1	3	5	-1	4	0	0	3	4				
8	-1	3	0	2	2								
9	0												
10	1	-1	1	4	1	2	-1	4					
11	5	3	5										
12	1	3	3	2	0								
13	1	3	3	2	0								
14	1	3	3	2	0								

Рис 3. Commands.txt

## 5.2. Эталонное решение

Работа эталонного решения на сгенерированных ранее наборах команд.

```

Терминал - vfibyrf007@vfibyrf007-HP-ProBook-430-G3: ~
Файл Правка Вид Терминал Вкладки Справка
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol -1 -1 -1 3 1 1 -1 -1 4
Fail with command 1
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol -1 1 0 3 4 4 1 2 5
Fail with command 1
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 4 3 3
Fail with last command
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1 3 2 5 4 1 3 5
Students with mark less than 60 percents: 10
Students with max mark: 2
Students with mark less than 60 percents: 10
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 2 1 4 1 5
Students with max mark: 2
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1 4 2 5 3 4 1 2
Fail with last command
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1 3 5 -1 4 0 0 3 4
Fail with command 4
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol -1 3 0 2 2
Fail with command 1
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 0
Too few commands
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1 -1 1 4 1 2 -1 4
Fail with command 2
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol
Too few commands
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 5 3 5
Students with mark less than 60 percents: 11
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1
Too few commands
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~$ ./RefSol 1 3 3 2 0
Fail with command 5

```

Рис 4. Результаты работы эталонного решения

Прим. Кроме как для наборов <1 3 2 5 4 1 3 5>, <5 3 5> не создавалось выходных файлов

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Group,Name,Surname,Patronymic,GitHub,Email,ExamResult														
2	26,flfmb,hcjxdhu,ahpfftypxksvck,kyqcvorp,sjfdvh,3														
3															
4															
5	81,aslalunmy,jknbnurnj,sxrtgtjubuu,oaondmhbgatdbk,gcxsgnbeywlyob,2														
6															
7															
8	90,oabhc,qxunutkneg,ljquaudpsc,ocgnwyna,tewwwuqtrsfhdv,1														
9															
10															
11	125,flfmb,hcjxdhu,ahpfftypxksvck,kyqcvorp,sjfdvh,3														
12															
13															
14	458,xlqftxmhl,aswfkdlhw,qntpa,qrwmggk,pkvavejr,5														
15															
16															

Рис 5. Выходной файл для набора команд <1 3 2 5 4 1 3 5>

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
17	537,opliiv,hxpmhvipbchg,fnmq,xgghdrhk,athdtpf,2														
18	537,aslalunmy,jknbnurnj,sxrtgtjubuu,oaondmhbgatdbk,gcxsgnbeywlyob,2														
19															
20															
21	618,fvswryaty,ojyq,haddivlb,cthsiyegvegt,auiyfxq,4														
22	618,hbryviwcuxqc,bxomrju,xakycrv,cdkyarqu,lsuga,2														
23															
24															
25	957,oabhc,qxunutkneg,ljquaudpsc,ocgnwyna,tewwwuqtrsfhdv,1														
26	957,pkfebslqsa,gwltcdt,ehupa,ykyisdakdad,qlpymolvendrx,1														
27															
28															
29	995,xsygwlxptyk,ngulufwrofawhx,vrtpqy,xents,rhmvmxmkllwhgfh,2														
30	995,xlqftxmhl,aswfkdlhw,qntpa,qrwmggk,pkvavejr,5														
31															
32															

Рис 6. Выходной файл для набора команд <1 3 2 5 4 1 3 5> (продолжение таблицы)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Group,Name,Surname,Patronymic,GitHub,Email,ExamResult														
2	26,flfmb,hcjixdhu,ahpffitypxksvck,kyqcvorp,sjfdvh,3														
3															
4															
5	81,aslalunmy,jknburnj,sxrtgtjubuu,oaondmhbgatdbk,gcxsgnbeywlyob,2														
6															
7															
8	90,oabhc,qxunutkneg,ljquaudpsc,ocgnwyna,tewwwuqtrsfdv,1														
9															
10															
11	125,flfmb,hcjixdhu,ahpffitypxksvck,kyqcvorp,sjfdvh,3														
12															
13															
14	458,xlqftxmhl,aswfkldlihw,qntpa,qrwmggk,pkvavejr,5														
15															
16															

Рис 8. Выходной файл для набора команд <2 1 4 1 5>

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
15															
16															
17	537,opiiiv,hxpvmhvipbchg,fmnmq,xgghdrhk,athdtpf,2														
18	537,aslalunmy,jknburnj,sxrtgtjubuu,oaondmhbgatdbk,gcxsgnbeywlyob,2														
19															
20															
21	618,fvswryaty,ojoyq,haddivlhb,cthsiyegvegt,aauiyfxq,4														
22	618,hbryviwcuxqc,bxomrju,xakyaacr,cdkyarqu,lsuga,2														
23															
24															
25	957,oabhc,qxunutkneg,ljquaudpsc,ocgnwyna,tewwwuqtrsfdv,1														
26	957,pkfebslqsa,gwltcdt,ehupa,ykyisdakdad,qlpymolvendrx,1														
27															
28															
29	995,xsygwlxptyk,ngulufwrofawhx,vrvtquy,xents,rhmvmxmkllwhgfh,2														
30	995,xlqftxmhl,aswfkldlihw,qntpa,qrwmggk,pkvavejr,5														

Рис 9. Выходной файл для набора команд <2 1 4 1 5> (продолжение таблицы)

### 5.3. Работа скрипта

Проверяется работа скрипта, который включает в себя сборку проекта вместе с пользовательским решением и сравнение его работы с образцовым на сгенерированных данных.

```
vfibyrf007@vfibyrf007-HP-ProBook-430-G3:~/YandexDisk/Study/Prog/Course_Work$ ./run.sh
touch Usersol.cpp
cat Refsol.cpp > Usersol.cpp
gcc Generate.cpp Generate_Fun.cpp -o Generator
gcc Refsol.cpp Refsol_Fun.cpp -o Refsol
gcc Usersol.cpp Refsol_Fun.cpp -o Usersol
rm Usersol
Successful
```

Рис 10. Работа скрипта с идентичным пользовательским решением

```

vfbyrf007@vfbyrf007-HP-ProBook-430-G3:~/YandexDisk/Study/Prog/Course_Work$ ./run.sh
gcc Generate.cpp Generate_Fun.cpp -o Generator
gcc Refsol.cpp Refsol_Fun.cpp -o Refsol
gcc Usersol.cpp Refsol_Fun.cpp -o Usersol
rm Usersol.cpp
Fail test 2
Correct output: Too few commands
Your output: Fail with command 1

```

*Рис 11. Работа скрипта с «испорченным» пользовательским решением*

```

vfbyrf007@vfbyrf007-HP-ProBook-430-G3:~/YandexDisk/Study/Prog/Course_Work$ ./run.sh
gcc Generate.cpp Generate_Fun.cpp -o Generator
gcc Refsol.cpp Refsol_Fun.cpp -o Refsol
gcc Usersol.cpp Refsol_Fun.cpp -o Usersol
rm Usersol.cpp
Fail test 4
Correct output: Fail with last command
Your output: Students with max mark: 4

```

*Рис 12. Работа скрипта с «испорченным» пользовательским решением*

```

vfbyrf007@vfbyrf007-HP-ProBook-430-G3:~/YandexDisk/Study/Prog/Course_Work$ ./run.sh
gcc Generate.cpp Generate_Fun.cpp -o Generator
gcc Refsol.cpp Refsol_Fun.cpp -o Refsol
gcc Usersol.cpp Refsol_Fun.cpp -o Usersol
rm Usersol.cpp
Fail test 1
Correct output: Fail with command 2
Your output: Fail

```

*Рис 13. Работа скрипта с «испорченным» пользовательским решением*

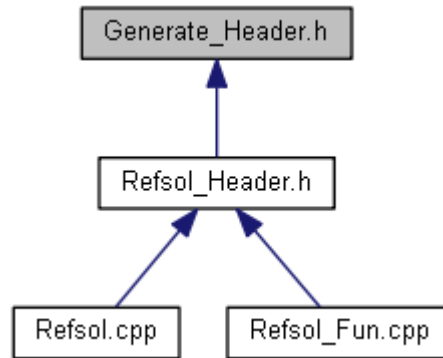
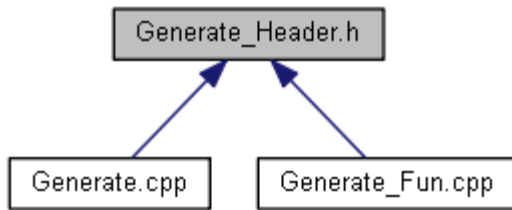
## 6. ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта, были закреплены на практике знания о структурах, файлах, массивах, функциях ввода/вывода, сценариях bash и make файлах. Стандартными средствами языка Си воплощён весь требуемый функционал. Все задачи, поставленные во введении, реализованы полностью. В связи с чем, считаю, что цель данной работы выполнена.

## 7. ПРИЛОЖЕНИЕ

### 7.1 Generate

#### 7.1.1. Generate\_Header.h



```
/*!  
\file  
\brief Generate Header  
\author Rybin Aleksandr 1 course 2 half  
\date 21.05.2017  
\version 3.0  
*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <string.h>  
  
#define MAX_LENGTH_ARRAY_COMMANDS 10                                //!< Max length of generating  
array of commands  
#define MAX_LENGTH_TOKEN 15                                         //!<< Max length of gen-  
erating random token  
#define MIN_LENGTH_TOKEN 5                                           //!<< Min length  
of generating random token  
#define MAX_GROUP 1000                                              //!<< Max  
generating group number  
#define MAX_EXAM_RESULT 5                                           //!<< Max generating  
exam result  
#define MAX_LINES 15                                               //!<< Max lines in generating ta-  
bles  
#define MIN_LINES 5                                                 //!<< Min lines in gener-  
ating tables  
#define HEADER1 "Name,Surname,Patronymic,GitHub,Email,Group\n"      //!<< Header for first table  
#define HEADER2 "Name,Surname,Exam_mark\n"                          //!<< Header for  
second table  
#define INPUT1 "input_file_1.csv"                                    //!<< First table  
#define INPUT2 "input_file_2.csv"                                    //!<< Second table  
#define COMMANDS "commands.txt"                                     //!<< File with commands  
  
/*!  
\struct  
\brief Struct Table for generating data, or read data from tables  
*/  
typedef struct Table  
{  
    char name[MAX_LENGTH_TOKEN];                                     //!<< Name of student  
    char surname[MAX_LENGTH_TOKEN];                                //!<< Surame of student  
    char patronymic[MAX_LENGTH_TOKEN];                             //!<< Patronymic of student  
    char GitHub_account[MAX_LENGTH_TOKEN];                         //!<< GitHub account of student  
    char Email[MAX_LENGTH_TOKEN];                                  //!<< Student's email  
    int group;                                                      //!<< Student's group  
    int exam_result;                                                //!<< Mark for exam
```

```

        int flag;                                //!< If repeats 1, if last in group (-
1), default 0
}Table;

/*!
\brief Struct Array for return information about vector
*/
typedef struct Vector
{
    int len;                                    //!< Length of vector
    Table vector_ptr[MAX_LINES];              //!< Pointer to struct Table array
}Vector;

/*!
\brief Generates vector of integers
\param[in] commands_list Pointer to file to write commands
\return void
\ingroup Generate
*/
void PutRandomCommands(FILE* commands_list);

/*!
\brief Generates random token
\param[in] random_token Pointer to string to write random token
\return void
\ingroup Generate
*/
void GenerateToken(char* random_token);

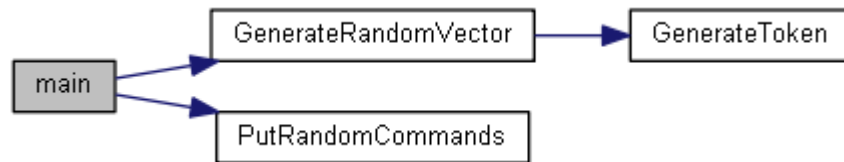
/*!
\brief Generates random vector of fields
\return info Sturct Vector whith information about generated vector
\ingroup Generate
*/
Vector GenerateRandomVector();

```

### 7.1.2. Generate\_Fun.cpp

```
/*!  
\file  
\brief Generate Functions  
  
\author Rybin Aleksandr 1 course 2 half  
\date 21.05.2017  
\version 3.0  
*/  
  
#include "Generate_Header.h"  
  
void PutRandomCommands(FILE* commands_list)  
{  
    int len = rand() % MAX_LENGTH_ARRAY_COMMANDS;  
  
    for (int i = 0; i < len; i++)  
        fprintf(commands_list, "%d ", -1 + rand() % 8);  
  
    fprintf(commands_list, "\n");  
}  
  
void GenerateToken(char* random_token)  
{  
    int len = MIN_LENGTH_TOKEN + rand() % (MAX_LENGTH_TOKEN - MIN_LENGTH_TOKEN);  
  
    for (int i = 0; i < len ; i++)  
        random_token[i] = (char)(97 + rand() % 25);  
    random_token[len] = '\0';  
}  
  
Vector GenerateRandomVector()  
{  
    Vector info;  
    memset(info.vector_ptr, 0, sizeof(info.vector_ptr));  
  
    info.len = MIN_LINES + rand() % (MAX_LINES - MIN_LINES);  
  
    for (int i = 0; i < info.len; i++)  
    {  
        GenerateToken(info.vector_ptr[i].Email);  
        GenerateToken(info.vector_ptr[i].GitHub_account);  
        GenerateToken(info.vector_ptr[i].name);  
        GenerateToken(info.vector_ptr[i].surname);  
        GenerateToken(info.vector_ptr[i].patronymic);  
        info.vector_ptr[i].group = 1 + rand() % MAX_GROUP;  
        info.vector_ptr[i].exam_result = 1 + rand() % MAX_EXAM_RESULT;  
    }  
    return info;  
}
```

### 7.1.3. Generate.cpp



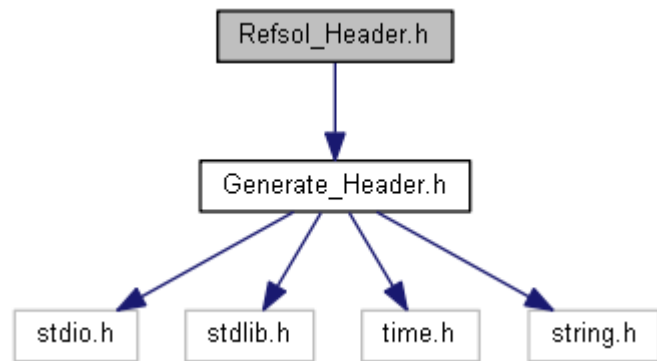
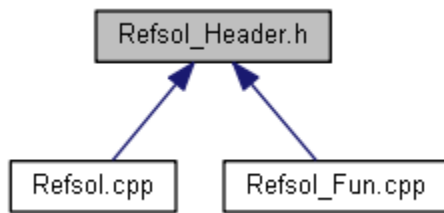
```
/*!  
\file  
\brief Generating data for checker
```

This file contains code for generate 2 random tables (csv format) and commands\_file.txt

```
\author Rybin Aleksandr 1 course 2 half  
\date 21.05.2017  
\version 3.0  
*/  
#include "Generate_Header.h"  
int main()  
{  
    // /* Initialize a seed for rand() */  
    srand(time(NULL));  
  
    /* Create new files */  
    FILE* Table1 = fopen(INPUT1, "w");  
    FILE* Table2 = fopen(INPUT2, "w");  
    FILE* Commands = fopen(COMMANDS, "w");  
  
    /* Create table header */  
    fprintf(Table1, HEADER1);  
    fprintf(Table2, HEADER2);  
  
    Vector random = GenerateRandomVector();  
  
    /* Write commands file as much as lines in generated table */  
    for (int i = 0; i < random.len; i++)  
        PutRandomCommands(Commands);  
  
    /* Write files */  
    for (int i = 0; i < random.len; i++)  
    {  
        /* Take random element from vector */  
        int index = rand() % random.len;  
        fprintf(Table1, "%s,%s,%s,%s,%s",  
                random.vector_ptr[index].name,  
                random.vector_ptr[index].surname,  
                random.vector_ptr[index].patronymic,  
                random.vector_ptr[index].GitHub_account,  
                random.vector_ptr[index].Email);  
  
        fprintf(Table2, "%s,%s,%d\n",  
                random.vector_ptr[index].name,  
                random.vector_ptr[index].surname,  
                random.vector_ptr[index].exam_result);  
        index = rand() % random.len;  
        fprintf(Table1, ",%d\n", random.vector_ptr[index].group);  
    }  
    fclose(Table1);  
    fclose(Table2);  
    fclose(Commands);  
    return 0;  
}
```

## 7.2. Refsol

### 7.2.1. Refsol\_Header.h



```
/*!  
\file  
\brief Refsol header file  
\author Rybin Aleksandr 1 course 2 half  
\date 22.05.2017  
\version 3.0  
*/  
  
#include "Generate_Header.h"  
  
#define RESULT "results.csv" //!< New file with results of program  
#define RESULT_HEADER "Group,Name,Surname,Patronymic,GitHub,Email,ExamResult\n" //!< Header for  
file with results  
#define CHECK_REPEATS 1 //!< Command 1  
#define CHECK_BY_GROUP 2 //!< Command 2  
#define CHECK_BAD_RESULTS 3 //!< Command 3  
#define CHECK_GOOD_RESULTS 4 //!< Command 4  
#define SAVE_RESULTS 5 //!< Command 5  
  
/*!  
\brief Compare two objects of struct Table  
\param[in] a first ptr  
\param[in] a second ptr  
\return a->group - b->group  
\ingroup Refsol  
*/  
int CompareTable(const Table* a, const Table* b);  
  
/*!  
\brief Parse string  
\param[in] obj pointer to object to write result  
\param[in] buf string to parse  
\param[in] param <0> if parse sting to first table or <1> to second table  
\return result of parsing  
\ingroup Refsol  
*/  
int ParseString(Table* obj, char* buf, int param);  
  
/*!  
\brief Read input_file_1  
\param[in] stream file with table1  
\param[in] data pointer to object to read data  
\return void  
\ingroup Refsol  
*/  
void ReadTable1(FILE* stream, Vector* data);  
  
/*!
```

```

\brief Read input_file_2
\param[in] stream file with table2
\param[in] data pointer to object to read data
\return void
\ingroup Refsol
*/
void ReadTable2(FILE* stream, Vector* data);

/*!
\brief Combine data from first & second files
\param[in] Data1 pointer to object with first table
\param[in] Data2 pointer to object with second table
\param[in] data pointer to object to combine data
\return void
\ingroup Refsol
*/
void CombineTables(Vector* data1, Vector* data2, Vector* data);

/*!
\brief Count students with mark less then 60% of max
\param[in] data struct Vector with info
\return void
\ingroup Refsol
*/
void CheckBadResults(Vector* data);

/*!
\brief Count students with max mark
\param[in] data struct Vector with info
\return void
\ingroup Refsol
*/
void CheckGoodResults(Vector* data);

/*!
\brief Check repeats in sorted readed data
\param[in] data struct Vector with data
\return void
\ingroup Refsol
*/
void CheckRepeats(Vector* data);

/*!
\brief Check data by group and marks last stident in group
\param[in] data struct Vector with data
\return void
\ingroup Refsol
*/
void CheckByGroup(Vector* data);

/*!
\brief Save results in new file results.txt
\param[in] data struct Vector with data
\return void
\ingroup Refsol
*/
void SaveResults(Vector* data);

```



### 7.2.2. Refsol\_Fun.cpp

```
/*!  
\file  
\brief Refsol Functions  
\author Rybin Aleksandr 1 course 2 half  
\date 22.05.2017  
\version 3.0  
*/  
  
#include "Refsol_Header.h"  
  
int CompareTable(const Table* a, const Table* b)  
{  
    return (a->group - b->group);  
}  
  
void ReadTable1(FILE* stream, Vector* data)  
{  
    int count = 0;  
    char buf[MAX_LENGTH_TOKEN * 5 + 10]; // Five fields and integer(INT_MAX have 10 signs)  
  
    memset(data->vector_ptr, 0, sizeof(data->vector_ptr));  
  
    /* Skip Header,*/  
    fscanf(stream, "%*s");  
    if (feof(stream))  
    {  
        printf("Error reading first table");  
        data->len = -1;  
        return ;  
    }  
  
    while (count < MAX_LINES)  
    {  
        fscanf(stream, "%s", buf);  
        if (feof(stream))  
        {  
            data->len = count;  
            return ;  
        }  
  
        if (!ParseString(&(data->vector_ptr[count]), buf, 0))  
        {  
            printf("Error reading first table");  
            data->len = -1;  
            return ;  
        }  
        count++;  
    }  
}  
  
void ReadTable2(FILE* stream, Vector* data)  
{  
    int count = 0;  
    char buf[MAX_LENGTH_TOKEN * 2 + 10]; // Five fields and integer(INT_MAX have 10 signs)  
  
    memset(data->vector_ptr, 0, sizeof(data->vector_ptr));  
  
    /* Skip Header,*/  
    fscanf(stream, "%*s");  
    if (feof(stream))  
    {  
        printf("Error reading second table");  
        data->len = -1;  
        return ;  
    }  
}
```

```

while (count < MAX_LINES)
{
    fscanf(stream,"%s",buf);
    if(!feof(stream))
    {
        data->len = count;
        return ;
    }

    if(!ParseString(&(data->vector_ptr[count]), buf , 1))
    {
        printf("Error reading second table");
        data->len = -1;
        return ;
    }
    count++;
}

void CombineTables(Vector* data1,Vector* data2,Vector* data)
{
    if(data1->len < 0 || data2->len < 0 || data1->len != data2->len)
    {
        data->len = -1;
        return ;
    }

    memset(data->vector_ptr, 0 ,sizeof(data->vector_ptr));
    data->len = data1->len;

    for(int i = 0; i < data->len; i++)
    {
        if(strcmp(data1->vector_ptr[i].name,data2->vector_ptr[i].name) != 0
            || strcmp(data1->vector_ptr[i].surname,data2->vector_ptr[i].surname) != 0)
        {
            data->len = -1;
            return ;
        }

        strcpy(data->vector_ptr[i].name,data1->vector_ptr[i].name);
        strcpy(data->vector_ptr[i].surname,data1->vector_ptr[i].surname);
        strcpy(data->vector_ptr[i].patronymic,data1->vector_ptr[i].patronymic);
        strcpy(data->vector_ptr[i].GitHub_account,data1->vector_ptr[i].GitHub_account);
        strcpy(data->vector_ptr[i].Email,data1->vector_ptr[i].Email);
        data->vector_ptr[i].group = data1->vector_ptr[i].group;
        data->vector_ptr[i].exam_result = data2->vector_ptr[i].exam_result;
    }
}

void CheckBadResults(Vector* data)
{
    int count = 0;
    for (int i = 0; i < data->len; i++)
    {
        if (((data->vector_ptr[i].exam_result <= (int)(0.6 * MAX_EXAM_RESULT))
            && (data->vector_ptr[i].flag != 1))
            count++;
    }
    printf("Students with mark less then 60 percents: %d\n", count);
}

void CheckGoodResults(Vector* data)
{
    int count = 0;
    for (int i = 0; i < data->len; i++)
    {
        if (((data->vector_ptr[i].exam_result == MAX_EXAM_RESULT)
            && (data->vector_ptr[i].flag != 1))
            count++;
    }
}

```

```

    }
    printf("Students with max mark: %d\n", count);
}

void CheckRepeats(Vector* data)
{
    for (int i = 0; i < data->len - 1; i++)
        if(memcmp(&(data->vector_ptr[i]),
                  &(data->vector_ptr[i + 1]),sizeof(Table)) == 0)
            data->vector_ptr[i].flag = 1;
}

void CheckByGroup(Vector* data)
{
    for (int i = 0; i < data->len; i++)
    {
        if (data->vector_ptr[i].group != data->vector_ptr[i + 1].group)
        {
            if (data->vector_ptr[i].flag == 1)    // If repeats then flag saves
                continue;
            data->vector_ptr[i].flag = -1;
        }
        else if (data->vector_ptr[i + 1].flag == 1)
            //If next is repeat then current student is last
            data->vector_ptr[i].flag = -1;
    }
}

void SaveResults(Vector* data)
{
    FILE* result = fopen(RESULT, "w");
    fprintf(result, RESULT_HEADER);

    for (int i = 0; i < data->len; i++)
    {
        if (data->vector_ptr[i].flag == 1)
            continue;

        fprintf(result, "%d,%s,%s,%s,%s,%s,%d\n",
                data->vector_ptr[i].group,
                data->vector_ptr[i].name,
                data->vector_ptr[i].surname,
                data->vector_ptr[i].patronymic,
                data->vector_ptr[i].GitHub_account,
                data->vector_ptr[i].Email,
                data->vector_ptr[i].exam_result);

        if (data->vector_ptr[i].flag == -1)
            fprintf(result, "\n\n");
    }

    fclose(result);
}

int ParseString(Table* obj,char* buf,int param)
{
    char* buf_ptr = strtok(buf,"");
    if(!buf_ptr) return -1;
    strcpy(obj->name,buf_ptr);

    buf_ptr = strtok(NULL,"");
    if(!buf_ptr) return -1;
    strcpy(obj->surname,buf_ptr);

    if(param == 0);
    else
    {
        buf_ptr = strtok(NULL,"");
        if(!buf_ptr) return -1;
    }
}

```

```

        obj->exam_result = atoi(buf_ptr);

        buf_ptr = strtok(NULL, ",");
        if(buf_ptr) return -1; // if something extra int buf

        return 1;
    }

    buf_ptr = strtok(NULL, ",");
    if(!buf_ptr) return -1;
    strcpy(obj->patronymic, buf_ptr);

    buf_ptr = strtok(NULL, ",");
    if(!buf_ptr) return -1;
    strcpy(obj->GitHub_account, buf_ptr);

    buf_ptr = strtok(NULL, ",");
    if(!buf_ptr) return -1;
    strcpy(obj->Email, buf_ptr);

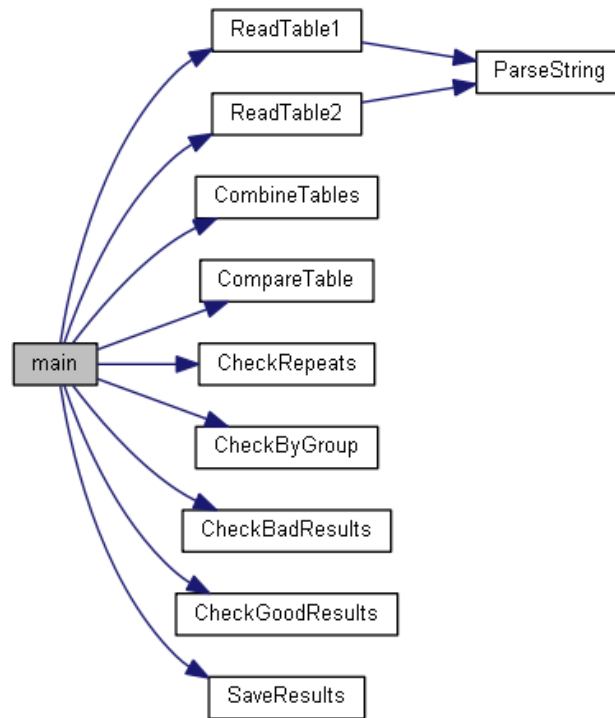
    buf_ptr = strtok(NULL, ",");
    if(!buf_ptr) return -1;
    obj->group = atoi(buf_ptr);

    buf_ptr = strtok(NULL, ",");
    if(buf_ptr) return -1; // if something extra int buf

    return 1;
}

```

### 7.2.3. Refsol.cpp



```
/*!  
\file  
\brief Solving checker task
```

This file contains code for checker task on right way for compare with user solve.

```
\author Rybin Aleksandr 1 course 2 half  
\date 22.05.2017  
\version 2.0
```

```
*/  
#include "Refsol_Header.h"  
int main(int argc, char* argv[])  
{  
    /* Open files */  
    FILE* Table1 = fopen(INPUT1, "r");  
    FILE* Table2 = fopen(INPUT2, "r");  
  
    if (Table1 == NULL)  
    {  
        printf("Fail with %s\n", INPUT1);  
        return 1;  
    }  
    else if (Table2 == NULL)  
    {  
        printf("Fail with %s\n", INPUT2);  
        return 1;  
    }  
  
    /* Read commands */  
    if (argc < 3)  
    {  
        printf("Too few commands\n");  
        return 1;  
    }  
    int* commands = (int*)malloc(sizeof(int) * (argc - 1));  
    for (int i = 1; i < argc; i++)  
    {
```

```

        commands[i - 1] = atoi(argv[i]);
        if ((commands[i - 1] < 1) || // Commands must be {1,2,3,4,5}
            (commands[i - 1] > 5))
        {
            printf("Fail with command %d\n", i);
            return 1;
        }
    }
    if (!(commands[argc - 2] == SAVE_RESULTS)) // Last command always 5 - "save results"
    {
        printf("Fail with last command\n");
        return 1;
    }

    Vector Data1, Data2, Data;
    ReadTable1(Table1, &Data1);
    ReadTable2(Table2, &Data2);
    CombineTables(&Data1, &Data2, &Data);
    if (Data1.len < 1 || Data2.len < 1 || Data.len < 1)
    {
        printf("Fail with data\n");
        return 1;
    }

    qsort(Data.vector_ptr, Data.len, sizeof(Table), (int (*)(const void*, const void*))CompareTable);

    /* Work with commands */
    for (int i = 0; i < argc - 1; i++)
    {
        switch (commands[i])
        {
            case CHECK_REPEATS:
            {
                CheckRepeats(&Data);
                break;
            }
            case CHECK_BY_GROUP:
            {
                CheckByGroup(&Data);
                break;
            }
            case CHECK_BAD_RESULTS:
            {
                CheckBadResults(&Data);
                break;
            }
            case CHECK_GOOD_RESULTS:
            {
                CheckGoodResults(&Data);
                break;
            }
            case SAVE_RESULTS:
            {
                SaveResults(&Data);
                break;
            }
            default:
                break;
        }
    }
    free(commands);
    fclose(Table1);
    fclose(Table2);
return 0;
}

```

## 7.3. Run\_sh, makefile

### 7.3.1. Run.sh

```
#!/bin/bash
```

```
make  
./Generator
```

```
count=0  
bad_result=1
```

```
cat commands.txt|while read line  
do
```

```
    31refsol_stdout=`./Refsol $line`  
    if [[ -r results.csv ]]  
    then  
        31refsol_fileout=`cat results.csv`  
    fi  
  
    31usersol_stdout=`./Usersol $line`  
    if [[ -r results.csv ]]  
    then  
        31usersol_fileout=`cat results.csv`  
    fi  
  
    if [[ "$refsol_stdout" != "$usersol_stdout" ||  
        -n "$refsol_fileout" && -n "$usersol_fileout" && "$refsol_fileout" != "$usersol_fileout" ]]  
    then  
        count=$(( $count + 1 ))  
        echo "Fail test $count "  
        echo "Correct output: $refsol_stdout"  
        echo "Your output: $usersol_stdout"  
  
        rm Generator Refsol Usersol input_file_1.csv input_file_2.csv commands.txt  
        if [[ -e results.csv ]]  
        then  
            rm results.csv  
        fi  
  
        exit 1  
    fi
```

```
    count=$(( $count + 1 ))  
done
```

```
if [[ $? -eq $bad_result ]]  
then  
    exit 1  
fi
```

```
rm Generator Refsol Usersol input_file_1.csv input_file_2.csv commands.txt  
if [[ -e results.csv ]]  
then  
    rm results.csv  
fi
```

```
echo «Successful»  
exit 0
```

### 7.3.2. *Makefile*

**Main:** Generate Refsol Usersol.cpp  
gcc Usersol.cpp -o Usersol  
rm Usersol.cpp

**Generate:** Generate.cpp Generate\_Fun.cpp Generate\_Header.h  
gcc Generate.cpp Generate\_Fun.cpp -o Generator

**Refsol:** Refsol.cpp Refsol\_Fun.cpp Refsol\_Header.h Generate\_Header.h  
gcc Refsol.cpp Refsol\_Fun.cpp -o Refsol