Firing range C++ project

File management and database functionality in C++

By: Kacper Klimkowski

Table of contents:

1) Introduction

This is an object oriented C++ project utilizing text files and functions made to mimic a database and functionality of an MySQL as well as provide a basic command line interface for interaction with said database.

2) Turning on the program: classes, database loading and a command line interface

The “class.h” file in the “sub” subdirectory of the project directory defines the classes used to simulate various tables of the database. These classes and their components are as follows:

Caliber

- Id
- Name
- Price

```
class Caliber {
    public:
        int id;
        string name;
        float price;

        Caliber(int id, string name, float price)
        {
            this->id = id;
            this->name = name;
            this->price = price;
        }
```

Caliber class

Client

- Id
- First name
- Last name
- PESEL
- Phone directional
- Phone number

```
class Client {
    public:
        int id;
        string first_name;
        string last_name;
        string pesel;
        string phone_dir;
        string phone_num;

        Client(int id, string first_name, string last_name,
                string pesel, string phone_dir, string phone_num)
        {
            this->id = id;
            this->first_name = first_name;
            this->last_name = last_name;
            this->pesel = pesel;
            this->phone_dir = phone_dir;
            this->phone_num = phone_num;
        }
```

Client class

Gun

- Id
- Name
- Caliber id
- Manufacturer id

```
class Gun {
    public:
        int id;
        string name;
        int caliber_id;
        int manufacturer_id;

        Gun(int id, string name, int caliber_id, int manufacturer_id)
        {
            this->id = id;
            this->name = name;
            this->caliber_id = caliber_id;
            this->manufacturer_id = manufacturer_id;
        }
```

Gun class

Manufacturer

- Id
- Name

```
class Manufacturer {
    public:
        int id;
        string name;

        Manufacturer(int id, string name)
        {
            this->id = id;
            this->name = name;
        }
```

Manufacturer class

Visit

- Id
- Client id
- Gun id
- Amount shot
- Accuracy
- Date
- Time

```
class Visit: public Forced_class_that_only_exists_to_tick_a_checkbox {
    public:
        int id;
        int client_id;
        int gun_id;
        int amount_shot;
        float accuracy;
        string date;
        string time;

        Visit(int id, int client_id, int gun_id, int amount_shot, float accuracy,
              string date, string time)
        {
            this->id = id;
            this->client_id = client_id;
            this->gun_id = gun_id;
            this->amount_shot = amount_shot;
            this->accuracy = accuracy;
            this->date = date;
            this->time = time;
        }
```

Visit class

Upon launching the program vectors storing the different classes are created and populated with data from the respective text files:

```
//Load database to vectors to simulate tables
string path;

vector<Caliber> caliber;
path = "./db/caliber.txt";
read_file_to_caliber_vec(path.c_str(), caliber);

vector<Client> client;
path = "./db/client.txt";
read_file_to_client_vec(path.c_str(), client);

vector<Gun> gun;
path = "./db/gun.txt";
read_file_to_gun_vec(path.c_str(), gun);

vector<Manufacturer> manufacturer;
path = "./db/manufacturer.txt";
read_file_to_manufacturer_vec(path.c_str(), manufacturer);

vector<Visit> visit;
path = "./db/visit.txt";
read_file_to_visit_vec(path.c_str(), visit);
```

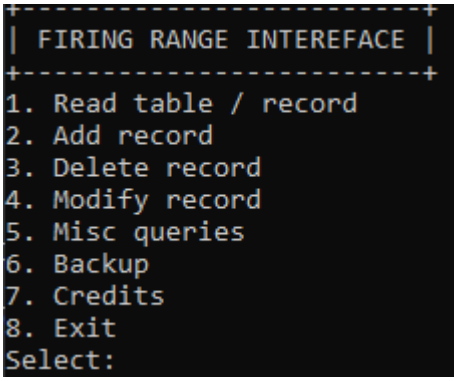
Table vectors creation and data population

These functions are included in the “db_load.h” in the “sub” subdirectory of the project directory. They all follow the same template when loading the data into vectors, i.e.:

```
void read_file_to_caliber_vec(string path, vector<Caliber> &caliber)
{
    fstream file;
    file.open(path, ios::in);
    if (!file.good())
    {
        cout<< "Error reading file";
        caliber.clear();
        file.close();
    }
    else
    {
        while (!file.eof())
        {
            int id;
            file>> id;

            string name;
            file>> name;

            float price;
            file>> price;

            Caliber temp(id, name, price);
            caliber.push_back(temp);
        }
        file.close();
    }
}
```

Function loading text file into a vector of the Caliber class

Now that the data is loaded into the program, the interface will appear:



Interface landing screen

From here you can select what you want the program to do. The functionalities are as follows: reading tables (everything from all tables to a specified record from a specified tables based on given criteria), adding, deleting and modifying records, miscellaneous queries (which supports only a single command), managing backups (creating and loading), a brief credits page (with thanks to contributing parties) and exit.

The interface functions on a series of nested switch(case) statements

```
cout<< endl;
cout<<
    "+-------------------------+" << endl <<
    "| FIRING RANGE INTEREFACE |" << endl <<
    "+-------------------------+" << endl;
int choice;
cout<< "1. Read table / record" << endl; // done
cout<< "2. Add record" << endl; // done
cout<< "3. Delete record" << endl; // done
cout<< "4. Modify record" << endl; // done
cout<< "5. Misc queries" << endl;
cout<< "6. Backup" << endl; // done
cout<< "7. Credits" << endl; // done
cout<< "8. Exit" << endl; // done
cout<< "Select: "; cin>> choice;
cout<< endl << "+-------------------------+" << endl << endl;

switch(choice)
{
    case 1: // read table / record
    {
```

Snippet of the interface code

3) Reading tables and records

Upon selecting “1” on the main interface the user is given options as to what tables or records they want to search for:

```
1. Read all tables
2. Read specified table
3. Read specified record
4. Read joined tables
5. Go back
Select:
```

3.1) Read all tables

Upon selecting “1” all tables will be displayed

```
Caliber
id      name                    price
1       5.56x45_NATO******      4.5
2       7.62x51_NATO******      8
3       7.62x39**********       3
4       7.62x54R*********       7
5       5.7x28mm_FN*******      5
6       4.6x30_HK********       5.5
7       9x19_Parabellum***      2.5
8       9x18_Makarov******      2
9       .30-06_Springfield      10
10      .45_ACP**********       3.5
11      8mm_Mauser********      10
13      .40_S&W**********       20
----------------
Client
id      first_name      last_name       pesel           phone_dir       phone_num
1       Cezary****      Krawczyk**      00291571018     +48             124651423
2       Juliusz***      Cezar*****      97082950097     +48             325641741
3       California      Friday****      95071986791     +48             225437432
4       Daniel****      Makowski**      56040821196     +48             357416872
5       Florian***      Kowalski**      91050666771     +48             542136982
6       Ludwik****      Jawrowski*      98012844794     +48             236874159
7       Kacper****      Klimkowski      22010116894     +48             867409243
----------------
Gun
id      name                    caliber_id      manufacturer_id
1       M4_Carbine**********    1               4
2       SCAR-L**************    1               5
3       SCAR-H**************    2               5
4       MP7A2***************    6               2
5       UMP45***************    10              2
6       UMP9****************    7               2
7       Five-seveN**********    5               5
8       P90*****************    5               5
9       M1903_Springfield****   9               1
10      Makarov_PMM*********    8               3
11      Makarov_PB**********    8               3
12      Mosin-Nagant_M1891/30   4               6
13      Karabiner_98k********   11              7
14      MP-443_Grach*********   7               3
15      USP.45_Elite*********   10              2
```

Print of “Read all tables” option, pt.1

```
16      Glock_17***********     7               9
17      Glock_18C**********    7               9
18      P226R**************    7               8
19      MPX****************    7               8
20      MPX****************    7               8
22      MPX****************    7               8
----------------
Manufacturer
id      name
1       Springfield_Armory******
2       Heckler_&_Koch*********
3       Kalashnikov_Concern*****
4       Daniel_Defense*********
5       FN_Herstal*************
6       Imperial_Tula_Arms_Plant
7       Mauser*****************
8       SIG_Sauer**************
9       GLOCK******************
----------------
Visit
id      client_id       gun_id  amount_shot     accuracy        date            time
1       7               7       45              0.94            08.11.2022      12:00:00
2       3               18      5               0               08.11.2022      08:00:00
3       2               9       25              0.97            09.11.2022      08:00:00
4       4               1       33              0.6             08.11.2022      10:15:00
5       5               17      200             0.29            11.11.2022      20:30:00
6       6               13      14              0.99            13.11.2022      11:30:00
7       1               16      18              0.8             21.11.2022      12:30:00
8       1               10      16              0.78            21.11.2022      12:30:00
9       1               18      24              0.4             21.11.2022      12:30:00
10      1               15      8               0               21.11.2022      12:30:00
11      7               9       30              0.93            22.11.2022      21:15:00
12      7               13      30              0.96            22.11.2022      21:15:00
13      7               12      30              0.82            22.11.2022      21:15:00
14      2               4       80              0.93            01.12.2022      06:30:00
15      2               3       55              0.89            01.12.2022      06:30:00
16      2               2       90              0.99            01.12.2022      12:00:00
17      2               8       200             0.13            02.12.2022      12:00:00
----------------
```

Print of "Read all tables" option, pt.1

The code responsible is as followed:

```
case 1: // read all tables
{
    cout<< "-----------------" << endl;
    cout<< "Caliber" << endl;
    print_table(caliber);
    cout<< "-----------------" << endl;
    cout<< "Client" << endl;
    print_table(client);
    cout<< "-----------------" << endl;
    cout<< "Gun" << endl;
    print_table(gun);
    cout<< "-----------------" << endl;
    cout<< "Manufacturer" << endl;
    print_table(manufacturer);
    cout<< "-----------------" << endl;
    cout<< "Visit" << endl;
    print_table(visit);
    cout<< "-----------------" << endl;
} break;
```

Switch(case) statement when choosing to read all tables

```
template<typename vec_type>
void print_table(vec_type vec)
{
    vec[0].print_header();
    cout<< endl;
    for (auto x: vec)
    {
        x.print_class();
        cout<< endl;
    }
}
```

Universal print_table function

Every class possesses functions that print their headers and contents, respectively named print_header and print_class

```
void print_header()
{
    cout<< "id\tname\t\t\tcaliber_id\tmanufacturer_id";
}
void print_class()
{
    cout<< id << "\t" << name << "\t" << caliber_id << "\t\t" << manufacturer_id;
}
```

Example of print_header and print_class functions from Gun class

3.2) Read specified table

Upon selecting “2” the user will be prompted to specify the table they want to display

```
1. Caliber
2. Client
3. Gun
4. Manufacturer
5. Visit
6. Go back
Select: 4

+-------------------------+


-----------------
id      name
1       Springfield_Armory******
2       Heckler_&_Koch**********
3       Kalashnikov_Concern*****
4       Daniel_Defense**********
5       FN_Herstal**************
6       Imperial_Tula_Arms_Plant
7       Mauser******************
8       SIG_Sauer***************
9       GLOCK*******************
-----------------
```

Selecting to display the “Manufacturer” table

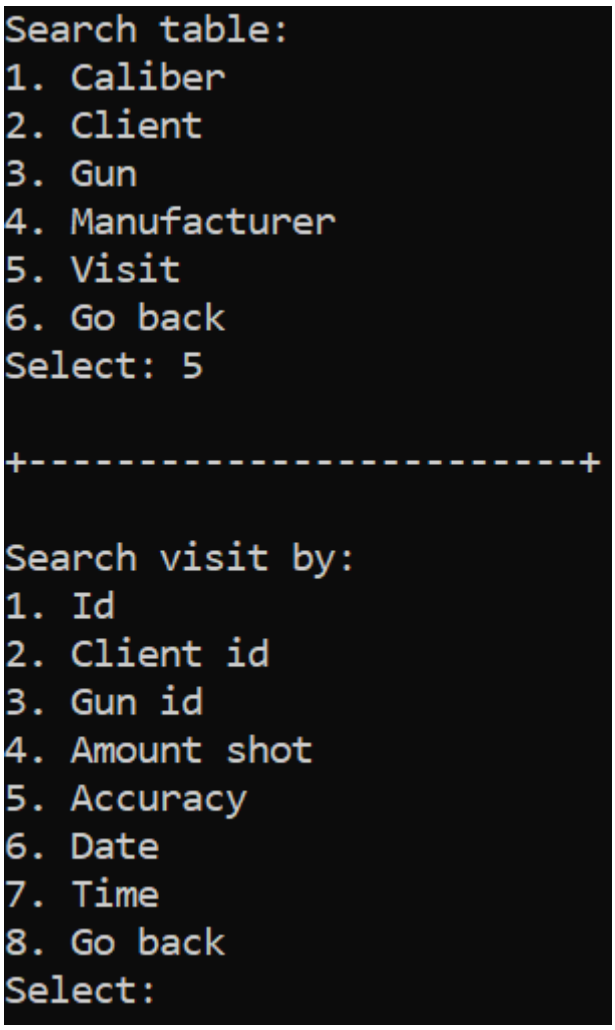
The code in this section is the same as in section 3.1, but simply separated into different cases

```
case 1: // read caliber
{
    cout<< "-----------------" << endl;
    print_table(caliber);
    cout<< "-----------------" << endl;
} break;

case 2: // read client
{
    cout<< "-----------------" << endl;
    print_table(client);
    cout<< "-----------------" << endl;
} break;
```

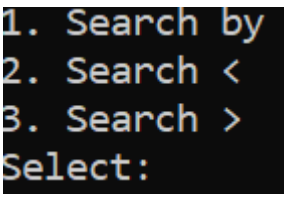
Example of separation of print_table statements

3.3) Read specified record

Upon selecting “3” the user will be prompted to specify the table of the record they want to search for, the column they want to search the record by and if supported, whether they want to search for records with greater (>), lower (<) or equal(=) of the specified criteria (i.e. searching Visit by amount shot allows for those options, whereas searching by gun id doesn’t).



Selecting to search for a record in the “Visit” table



If criteria allow

```
1. Search by
2. Search <
3. Search >
Select: 2
Search by amount shot: 75
id      client_id       gun_id  amount_shot     accuracy        date            time
1       7               7       45              0.94            08.11.2022      12:00:00
2       3               18      5               0               08.11.2022      08:00:00
3       2               9       25              0.97            09.11.2022      08:00:00
4       4               1       33              0.6             08.11.2022      10:15:00
6       6               13      14              0.99            13.11.2022      11:30:00
7       1               16      18              0.8             21.11.2022      12:30:00
8       1               10      16              0.78            21.11.2022      12:30:00
9       1               18      24              0.4             21.11.2022      12:30:00
10      1               15      8               0               21.11.2022      12:30:00
11      7               9       30              0.93            22.11.2022      21:15:00
12      7               13      30              0.96            22.11.2022      21:15:00
13      7               12      30              0.82            22.11.2022      21:15:00
15      2               3       55              0.89            01.12.2022      06:30:00
```

Searching Visit for records with less than (<) 75 shots

```
void print_visit_record_amount_shot(vector<Visit> visit, int target_symbol, int target_value)
{
    visit[0].print_header();
    cout<< endl;

    for (auto x: visit)
    {
        switch(target_symbol)
        {
            case 1:
            {
                if (x.amount_shot == target_value)
                {
                    x.print_class();
                    cout<< endl;
                }
            } break;

            case 2:
            {
                if (x.amount_shot < target_value)
                {
                    x.print_class();
                    cout<< endl;
                }
            } break;

            case 3:
            {
                if (x.amount_shot > target_value)
                {
                    x.print_class();
                    cout<< endl;
                }
            } break;
        }
    }
}
```

Function responsible for displaying the specified records from Visit when searching by amount shot

3.4) Read joined tables

Upon selecting “4” the user will be prompted to select which joined table they want to select. This section imitates MySQL’s inner join function by joining tables together based on foreign keys. The 2 tables possessing foreign keys are Gun and Visit.

Gun id	Name	Caliber	Manufacturer	Price
1	M4_Carbine**********	5.56x45_NATO******	Daniel_Defense**********	4.5
2	SCAR-L***********	5.56x45_NATO******	FN_Herstal**********	4.5
3	SCAR-H***********	7.62x51_NATO******	FN_Herstal**********	8
4	MP7A2***********	4.6x30_HK*********	Heckler_&_Koch**********	5.5
5	UMP45***********	.45_ACP**********	Heckler_&_Koch**********	3.5
6	UMP9************	9x19_Parabellum***	Heckler_&_Koch**********	2.5
7	Five-seveN**********	5.7x28mm_FN*******	FN_Herstal**********	5
8	P90**************	5.7x28mm_FN*******	FN_Herstal**********	5
9	M1903_Springfield****	.30-06_Springfield	Springfield_Armory******	10
10	Makarov_PMM**********	9x18_Makarov******	Kalashnikov_Concern*****	2
11	Makarov_PB**********	9x18_Makarov******	Kalashnikov_Concern*****	2
12	Mosin-Nagant_M1891/30	7.62x54R**********	Imperial_Tula_Arms_Plant	7
13	Karabiner_98k********	8mm_Mauser********	Mauser**********	10
14	MP-443_Grach********	9x19_Parabellum***	Kalashnikov_Concern*****	2.5
15	USP.45_Elite*********	.45_ACP**********	Heckler_&_Koch**********	3.5
16	Glock_17************	9x19_Parabellum***	GLOCK**********	2.5
17	Glock_18C***********	9x19_Parabellum***	GLOCK**********	2.5
18	P226R***********	9x19_Parabellum***	SIG_Sauer**********	2.5
19	MPX**************	9x19_Parabellum***	SIG_Sauer**********	2.5
20	MPX**************	9x19_Parabellum***	SIG_Sauer**********	2.5
22	MPX**************	9x19_Parabellum***	SIG_Sauer**********	2.5

Gun table inner joined with Caliber and Manufacturer tables

```
void print_joined_tables_gun(vector<Gun> gun, vector<Manufacturer> manufacturer, vector<Caliber> caliber)
{
    float price;
    cout<< "Gun id\tName\t\t\tCaliber\t\t\tManufacturer\t\t\tPrice" << endl;
    for (auto x: gun)
    {
        cout<< x.id << "\t" << x.name << "\t";
        for (auto y: caliber)
        {
            if (x.caliber_id == y.id)
            {
                cout<< y.name << "\t";
                price = y.price;
                break;
            }
        }
        for (auto z: manufacturer)
        {
            if (x.manufacturer_id == z.id)
            {
                cout<< z.name << "\t";
                break;
            }
        }
        cout<< price << endl;
    }
}
```

Code responsible for inner join of above example

4) Adding records

Upon selecting “2” on the main interface the user is given options as to which table they want to add a record to. From there the user will be prompted to enter the appropriate information in the correct format. The program checks for the correctness of the information the user is inputting and making sure that it’s in the correct format, if it’s not, the user is prompted to reenter information until it’s written in the correct format.

```
Input can't be empty
Name (BACK to exit):  .45-70 Government

Input can't have empty spaces, use '_' instead
Name (BACK to exit):  .45-70_Government


Price can't be negative
Price (BACK to exit): 12,5

Price must be a number
Price (BACK to exit): 12.5

id        name                          price
14        .45-70_Government*            12.5
Record added
```

Adding a new record to the Caliber table

```
case 1: // add record to caliber
{
    string name = "";
    while(!string_correct_caliber(caliber, name, "name"))
    {
        cout<< "Name (BACK to exit): ";
        cin.sync();
        getline(cin, name);
        cout<< endl;
        if (name == "BACK") return true;
    }
    cout<< endl;

    string price_str = "-1";
    while (!price_correct(price_str))
    {
        cout<< "Price (BACK to exit): ";
        cin.sync();
        getline(cin, price_str);
        cout<< endl;
        if (price_str == "BACK") return true;
    }

    name = normalize_string_caliber(caliber, name, "name");
    float price = stof(price_str);
    int id = caliber[caliber.size()-1].id + 1;
    add_record_caliber(id, name, price, caliber);
} break;
```

Code implementation for inputting data into Caliber table

```
bool string_correct_client(vector<Client> client, string str, string var)
{
    if (str == "")
    {
        cout<< "Input can't be empty" << endl;
        return false;
    }
    if (str[0] < 'A' || str[0] > 'Z')
    {
        cout<< "Input must start with a capital letter" << endl;
        return false;
    }
    for (int i=0; i<str.length(); i++)
    {
        if (str[i] == ' ')
        {
            cout<< "Input can't have empty spaces, use '_' instead" << endl;
            return false;
        }
        if (i>0 && (str[i] < 'a' || str[i] > 'z'))
        {
            cout<< "Only the first letter of the input is to be a capital letter" << endl;
            return false;
        }
    }
```

Criteria to check when inputting first and last name for Client table

```
bool string_correct_visit_date(vector<Visit> visit, string str)
{
    if (str.length() != 10)
    {
        cout<< "Date must be 10 symbols long" << endl;
        return false;
    }
    if (str[2] != '.' || str[5] != '.')
    {
        cout<< "Incorrect date format, format is DD.MM.YYYY" << endl;
        return false;
    }
    vector<string> date = split_str(str, '.');
    string day = date[0];
    string month = date[1];
    string year = date[2];
    if (year < "2022")
    {
        cout<< "The firing range wasn't open before 2022" << endl;
        return false;
    }
    if (day < "01") // if day is below 1, false
    {
        cout<< "Day can't go below 1" << endl;
        return false;
    }
    if (month < "01") // if month is below 1, false
    {
        cout<< "Month can't go below 1" << endl;
        return false;
    }
    if (month > "12") // if month is above 12, false
    {
        cout<< "Month can't go above 12" << endl;
        return false;
    }
    if (month == "02") // check for leap year
    {
        int year_int = (int(year[0])-'0')*1000 + (int(year[1])-'0')*100 + (int(year[2])-'0')*10 + int(year[3])-'0';
        cout<< endl << endl << year_int << endl << endl;
        if ((year_int % 4 == 0 && year_int % 100 == 0 && year_int % 400 == 0) || (year_int % 4 == 0 && year_int % 100 != 0)) // leap year
```

Part of criteria to check when inputting the date for Visit table

Additionally, certain inputs (such as first name for Client table) need to be normalized before being entered into the database (the length difference between given string and max length of string from the given column must be filled with “*”, which is done via the use of the appropriate normalize_string function

```
/*
    // Returns max length of given Caliber column
*/
int string_max_length_caliber(vector<Caliber> caliber, string var)
{
    int max_length = 0;
    if (var == "name")
    {
        for (auto x: caliber)
        {
            if (x.name.length() > max_length)
            {
                max_length = x.name.length();
            }
        }
    }
    return max_length;
}

/*
    // Normalizes given string for database entry into Caliber table
*/
string normalize_string_caliber(vector<Caliber> caliber, string str, string var)
{
    int max_length = string_max_length_caliber(caliber, var);
    while (str.length() < max_length)
    {
        str += '*';
    }
    return str;
}
```

String normalization function for the inputted name of caliber

The id in chosen automatically (1 greater than the id in the last record, which due to the way data is stored, always means 1 greater than the previous max id number). The inputted data is now ready to be added to the database. The code responsible for this follows the same formula.

```
void add_record_gun(int id, string name, int caliber_id, int manufacturer_id, vector<Gun> &gun, string path="./db/gun.txt")
{
    fstream file;
    file.open(path.c_str(), ios::app);
    file<< "\n" << id << "\t" << name << "\t" << caliber_id << "\t" << manufacturer_id;
    file.close();

    Gun temp(id, name, caliber_id, manufacturer_id);
    gun.push_back(temp);

    gun[0].print_header();
    cout<< "\n" << id << "\t" << name << "\t" << caliber_id << "\t\t" << manufacturer_id << endl;
    cout<< "Record added" << endl;
}
```

Code adding data into the Gun table

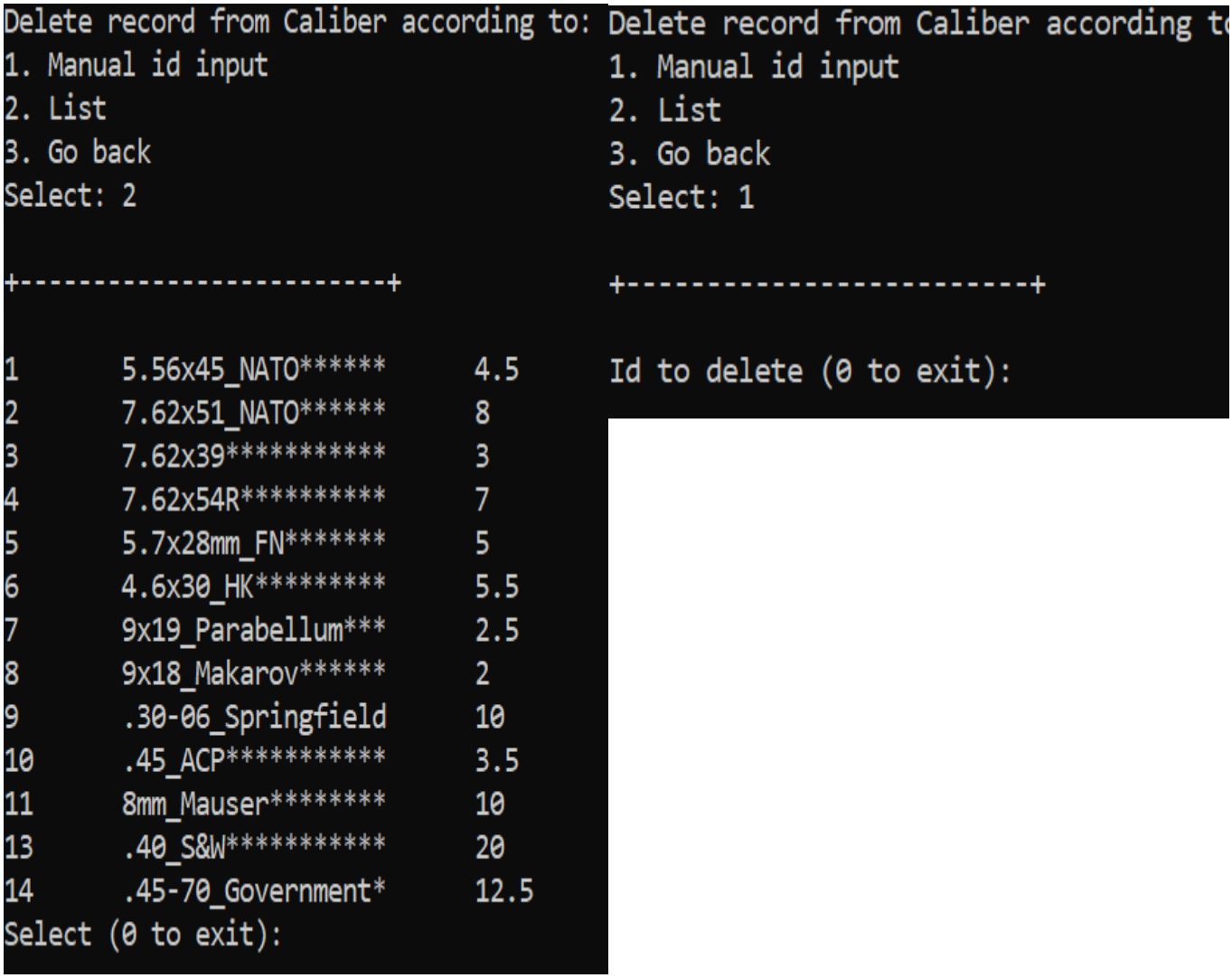
id	name	price
1	5.56x45_NATO******	4.5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R*********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20

id	name	price
1	5.56x45_NATO******	4.5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R*********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20
14	.45-70_Government*	12.5

Before and after adding a new record to the Caliber table

5) Deleting records

Upon selecting “3” on the main interface the user is given options as to which table they want to delete a record from. From there the user will be prompted to either manually enter the id of the record they want deleted or to select it from a list.



Manual and list inputs for Caliber table

```
case 1: // delete record caliber manual id
{
    int target_id;
    cout<< "Id to delete (0 to exit): "; cin>> target_id;
    if (target_id == 0) return true;
    delete_record_caliber(target_id, caliber, gun);
} break;

case 2: // delete record caliber list
{
    int target_id;
    for (auto x: caliber)
    {
        x.print_class();
        cout<< endl;
    }
    cout<< "Select (0 to exit): "; cin>> target_id;
    if (target_id == 0) return true;
    delete_record_caliber(target_id, caliber, gun);
} break;
```

Code for id selection for caliber Table

The code for deleting a record follows the same template for all tables.

```
void delete_record_caliber(int target_id, vector<Caliber> &caliber, vector<Gun> gun)
{
    /*
        // Check if caliber is foreign key to existing gun
        // Terminate operation if yes
        // Continue if not
    */
    for (auto x: gun)
    {
        if (x.caliber_id == target_id)
        {
            cout<< "Can't delete caliber that's a foreign key to existing gun" << endl;
            return;
        }
    }

    /*
        // Redo caliber vector without deleted record
    */
    fstream file;
    string path = "./db/caliber.txt";
    file.open(path.c_str(), ios::in);
    caliber.clear();
    while (!file.eof())
    {
        int id;
        file>> id;

        string name;
        file>> name;

        float price;
        file>> price;

        if (id != target_id)
        {
            Caliber temp(id, name, price);
            caliber.push_back(temp);
        }
    }
```

Code for deleting a record from caliber database pt. 1

```
    file.close();

    /*
        // Output redone vector into file
    */
    file.open(path.c_str(), ios::out);
    int counter = 1;
    for (auto x: caliber)
    {
        if (counter++ != caliber.size())
        {
            file<< x.id << "\t" << x.name << "\t" << x.price << endl;
        }
        else
        {
            file<< x.id << "\t" << x.name << "\t" << x.price;
        }
    }
    file.close();

    cout<< "Record with id " << target_id << " deleted" << endl;
}
```

Code for deleting a record from caliber database pt. 2

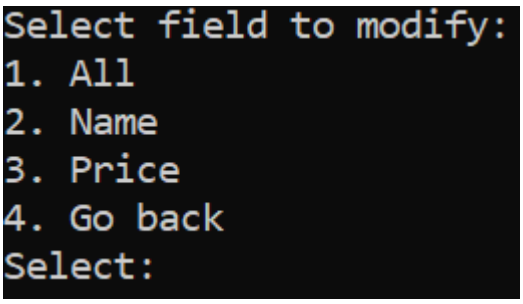
```
Select (0 to exit): 14
Record with id 14 deleted
```

```
id      name                    price
1       5.56x45_NATO******      4.5
2       7.62x51_NATO******      8
3       7.62x39**********       3
4       7.62x54R*********       7
5       5.7x28mm_FN*******      5
6       4.6x30_HK********       5.5
7       9x19_Parabellum***      2.5
8       9x18_Makarov******      2
9       .30-06_Springfield      10
10      .45_ACP**********       3.5
11      8mm_Mauser*******       10
13      .40_S&W**********       20
----------------
```

Deleting record with id 14 (added previously in the documentation) from Caliber table

6) Modifying records

Upon selecting “4” on the main interface the user is given options as to which table they want to modify a record from. From there the user will be prompted to either manually enter the id of the record they want to modify or to select it from a list. Then the user will be asked what parts of the record they want modified (an individual element or the entire record)



Select which part to modify from Caliber table

This leads the user to input the data in the same way as talked about in section 4).

Record modification follows the same template for all tables:

Save the record that’s being modified

```
string name;
float price;
for (auto x: caliber)
{
    if (x.id == target_id)
    {
        name = x.name;
        price = x.price;
        break;
    }
}
```

Input data that needs to be modified (i.e. price) and overwrite part of the previously saved record

```
case 3: // modify field caliber price
{
    string price_str = "-1";
    while (!price_correct(price_str))
    {
        cout<< "Price (BACK to exit): ";
        cin.sync();
        getline(cin, price_str);
        cout<< endl;
        if (price_str == "BACK") return true;
    }

    price = stof(price_str);
    modify_record_caliber(target_id, name, price, caliber);
}
```

Modify the record

```
id       name                    price
OLD: 1   5.56x45_NATO******      4.5
NEW: 1   5.56x45_NATO******      5
Returning...
```

```
-----------------
id       name                    price
1        5.56x45_NATO******      5
2        7.62x51_NATO******      8
3        7.62x39**********       3
4        7.62x54R*********       7
5        5.7x28mm_FN*******      5
6        4.6x30_HK********       5.5
7        9x19_Parabellum***      2.5
8        9x18_Makarov******      2
9        .30-06_Springfield      10
10       .45_ACP**********       3.5
11       8mm_Mauser*******       10
13       .40_S&W**********       20
```

```
void modify_record_caliber(int target_id, string name, float price, vector<Caliber> &caliber)
{
    fstream file;
    string path = "./db/caliber.txt";
    file.open(path.c_str(), ios::out);
    for (int i=0; i<caliber.size(); i++)
    {
        if (caliber[i].id == target_id)
        {
            caliber[i].print_header();
            cout<< endl << "OLD: "; caliber[i].print_class();
            caliber[i].name = name;
            caliber[i].price = price;
            cout<< endl << "NEW: "; caliber[i].print_class();
            cout<< endl;
        }
        file<< "\n" << caliber[i].id << "\t" << caliber[i].name << "\t" << caliber[i].price;
    }
    file.close();
}
```

Code for modifying Caliber table

7) Miscellaneous queries

Upon selecting “5” on the main interface the user is given options as to which miscellaneous query they want to run. Currently there is only one query as this section was meant to be used for the required functions as per the assignment that weren’t implemented anywhere else in the code, namely: class inheritance, polymorphism (operator overloading), abstract and inner classes.

The only available query in this section is “Add shots to visit”, which uses an overloaded + operator to increase the amount of shots and changes the accuracy in a user selected visit as well as the Visit class itself inheriting from a class which has an abstract overloaded + operator in an inner class.

```
Misc. queries
1. Add shots to visit
2. Go back
Select:
```

```
Add shots to Visit according to:
1. Manual id input
2. List
3. Go back
Select: 1
```

```
Id to modify (0 to exit): 1
Added shots can't be negative
Added shots (-1 to exit): 5

Accuracy must be between 0 and 1 (i.e. 0.5 for 50%)
Precision of 2 symbols after period
Accuracy (-1 to exit): 0.80
```

The added shots and accuracy are combined into one float number (i.e. 5.8) which is then added to the appropriate Visit record via the following code:

```
if (visit[i].id == target_id)
{
    visit[i]+num;
}
```

```
void operator+(float num)
{
    int added_shots = int(num);
    float with_accuracy = num - added_shots;

    int old_score = amount_shot * accuracy;
    int added_score = added_shots * with_accuracy;
    amount_shot += added_shots;
    float acc = (old_score + added_score) / float(amount_shot);
    accuracy = ceil(acc * 100) / 100;
}
```

id	client_id	gun_id	amount_shot	accuracy	date	time
1	7	7	45	0.94	08.11.2022	12:00:00

id	client_id	gun_id	amount_shot	accuracy	date	time
1	7	7	50	0.92	08.11.2022	12:00:00

The results

The query itself uses a combination of previously talked about in section 4) checking of format corrects when inputting data

The requirements for the assignment talked about in the introduction of this section were achieved as such:

```
class Forced_class_that_only_exists_to_tick_a_checkbox {
    public:
        class Forced_class_that_only_exists_to_tick_a_checkbox_inner {
            virtual void operator+(float num) = 0;
        };
};
```

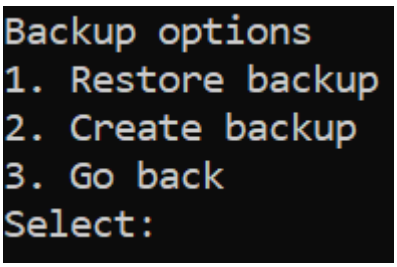
An aptly named class

```
class Visit: public Forced_class_that_only_exists_to_tick_a_checkbox {
```

Visit class inheriting from aptly named class

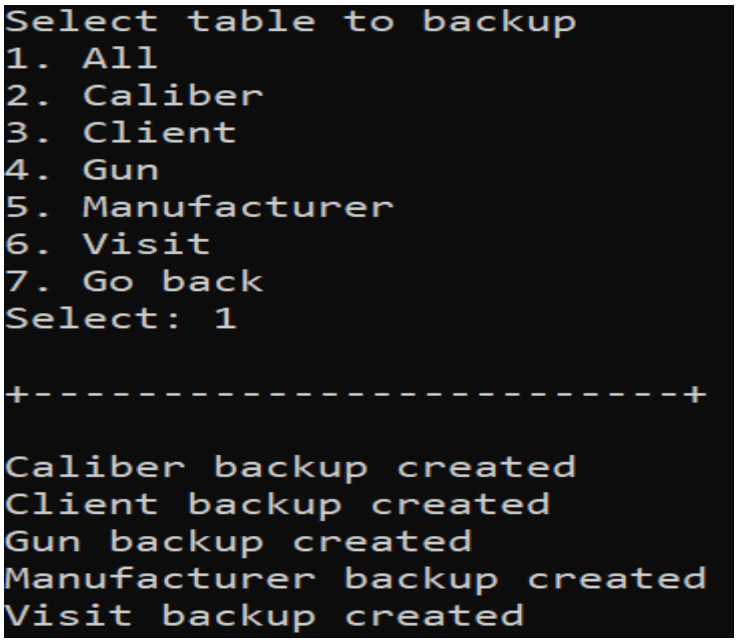
8) Creating and restoring backups

Upon selecting “6” on the main interface the user is given options as what to do with backups. All backups are stored in a “backup” subdirectory of the “db” subdirectory of the main project directory, where they are further split into “user” and “original” subdirectories. Original backups cannot be modified via the program (only by hand, but that is not recommended).



8.1) Creating backups

When choosing to create a backup the user can do so only for the selected table or for all



Backing up individual tables is just a split version of backing up all tables, just like talked about in section 3.2). All functions creating backups use the same template.

```
void backup_create_caliber(vector<Caliber> caliber)
{
    fstream file;
    string path = "./db/backup/user/caliber.txt";
    file.open(path.c_str(), ios::out);
    for (auto x: caliber)
    {
        file<< "\n" << x.id << "\t" << x.name << "\t" << x.price;
    }
    file.close();
    cout<< "Caliber backup created";
}
```

Code to create backup of Caliber table

8.2) Restoring backups

When choosing to restore backups, the user is given an option to either restore user created backups or original ones.

```
Backup options
1. Restore backup
2. Create backup
3. Go back
Select: 1

+-------------------------+

Restore backup from:
1. Original backup
2. User created backup
3. Go back
Select:
```

Either way, the user is then presented with an option to choose whether they want to restore all tables, or only the select one, just like in section 8.1).

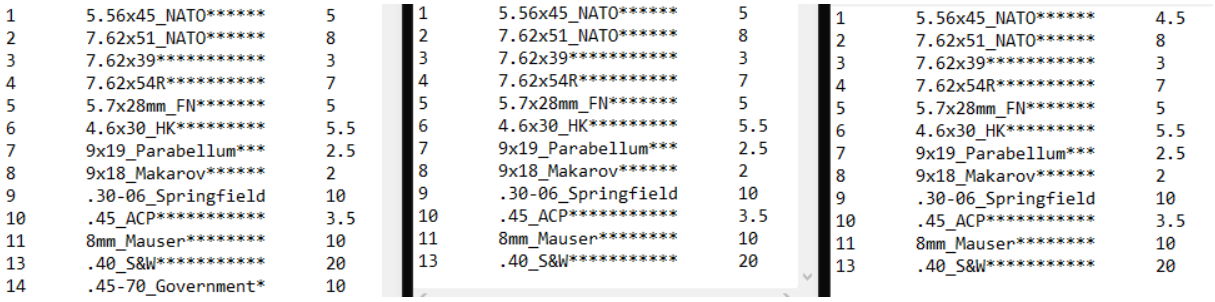
```
Select table to restore from user backup
1. All
2. Caliber
3. Client
4. Gun
5. Manufacturer
6. Visit
7. Go back
Select: 1

+-------------------------+

Caliber restored from user backup
Client restored from user backup
Gun restored from user backup
Manufacturer restored from user backup
Visit restored from user backup
```

Restoring individual tables is just a split version of backing up all tables, just like talked about in section 3.2). All functions creating restoring tables use the same template.

```
void backup_restore_caliber(vector<Caliber> &caliber, string source)
{
    string path = "./db/backup/" + source + "/caliber.txt";
    caliber.clear();
    read_file_to_caliber_vec(path.c_str(), caliber);

    fstream file;
    path = "./db/caliber.txt";
    file.open(path.c_str(), ios::out);
    for (auto x: caliber)
    {
        file<< "\n" << x.id << "\t" << x.name << "\t" << x.price;
    }
    file.close();
    cout<< "Caliber restored from " << source << " backup";
}
```

Code to restore selected backup for Caliber table

Backup demonstration:

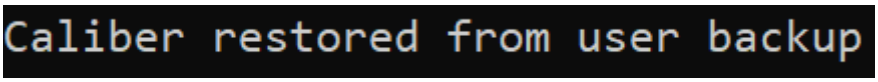


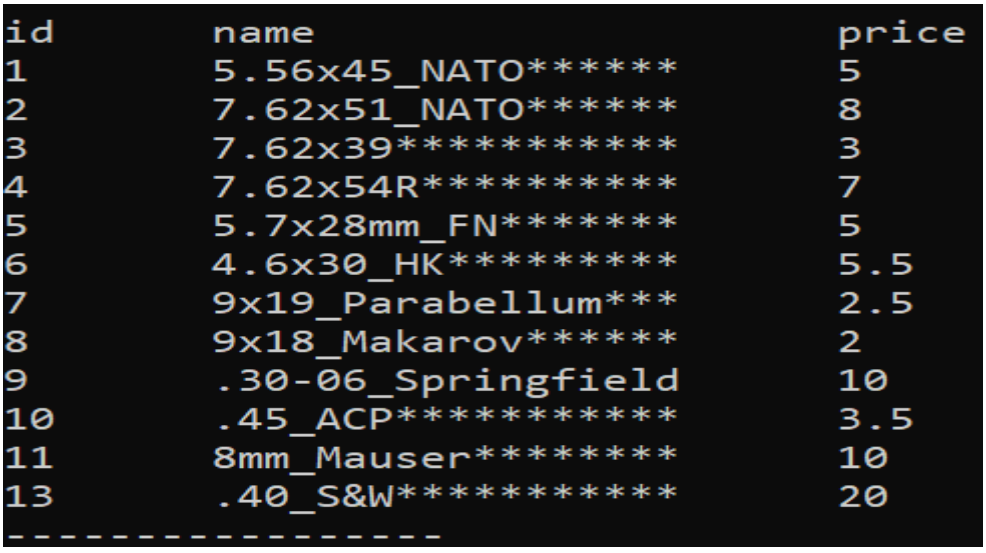
1	5.56x45_NATO******	5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R**********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK*********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20
14	.45-70_Government*	10

1	5.56x45_NATO******	5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R**********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK*********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20

1	5.56x45_NATO******	4.5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R**********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK*********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20

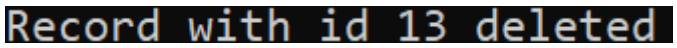
Text files of Caliber table, respectively: active table, user backup, original backup





id	name	price
1	5.56x45_NATO******	5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R**********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK*********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10
13	.40_S&W**********	20

State of active Caliber table after restoring user backup



Deleting caliber record with id 13

1	5.56x45_NATO******	5
2	7.62x51_NATO******	8
3	7.62x39**********	3
4	7.62x54R**********	7
5	5.7x28mm_FN*******	5
6	4.6x30_HK*********	5.5
7	9x19_Parabellum***	2.5
8	9x18_Makarov******	2
9	.30-06_Springfield	10
10	.45_ACP**********	3.5
11	8mm_Mauser********	10

State of user created Caliber backup

```
Caliber restored from original backup
```

```
id      name                        price
1       5.56x45_NATO******          4.5
2       7.62x51_NATO******          8
3       7.62x39**********          3
4       7.62x54R*********          7
5       5.7x28mm_FN*******          5
6       4.6x30_HK*********          5.5
7       9x19_Parabellum***          2.5
8       9x18_Makarov******          2
9       .30-06_Springfield          10
10      .45_ACP**********          3.5
11      8mm_Mauser********          10
13      .40_S&W**********          20
-----------------
```

State of active table after restoring original backup

9) Credits

```
Author: Kacper Klimkowski
Speical thanks to:
        Stack Overflow forum
        cplusplus.com
```

All there is to it

10) Afterword and statistics

Total lines of coded (including a few commented lines): 4933

Total time spent code: ~45 hours

Total time spent writing documentations: ~8 hours

Time completed before deadline: ~9 hours

Things going according to plan: ~20% (~90% once I stopped trying to do everything the smart and efficient way)

Finally understood why my compiler refused to recognize outside libraries?: No (this is a cry for help)

Number of crashes due to too deeply rooted recursion because the interface originally worked on recursively calling itself when going back instead of the current system where it's a bool function in a loop: 0

```
bool active = true;
while (active)
{
    active = interface(caliber, client, gun, manufacturer, visit);
}
```

Number of times Word annoyed me: ~20 (surprisingly little)

Years of life lost due to MySQL not working: at least 2

Chances that this documentation is up to scratch: ~60%

Expected grade: 4- (closer to being just acceptable rather than great, but otherwise known as potentially somewhat redeemable)

These rambling: too long

Bonus: Screenshot in the main project directory on GitHub