Main

```
#include <iostream>


#include "Sa/Helper/UI.h"


/* run this program using the console pauser or add your own getch, system("pause") or input loop */

int main(int argc, char** argv) {

	Sa::Helper::UI ui;


	ui.Initialize();


	return 0;

}
```

Helper/File

```
#ifndef Helper_File

#define Helper_File


#include <fstream>

#include <iostream>

#include <string>

#include <vector>
```

```
using namespace std;


namespace Sa{
            namespace Helper{
                        class File{
                                    public:
                                                File(string fileName){
                                                            _fileName = fileName;
                                                }


                                                void Append(string line){
                                                            fstream appendFileToWorkWith;


                                                            appendFileToWorkWith.open(_fileName.c_str(), ios::in
| ios::out | ios::app);


                                                // If file does not exist, Create new file
                                                if (!appendFileToWorkWith)
                                                {
                                                appendFileToWorkWith.open(_fileName.c_str(),  fstream::in |
fstream::out | fstream::trunc);
                                                            appendFileToWorkWith << line << endl;
                                                            appendFileToWorkWith.close();
                                                }
                                                else
```

```
                {
                        appendFileToWorkWith << line << endl;
                cout<<"\n";
                appendFileToWorkWith.close();
            }
                }

                vector<string> GetLines(){
                        ifstream file(_fileName.c_str());
                        string line;
                        vector<string> lines;

                        while(getline(file, line))
                                lines.push_back(line);

                        return lines;
                }

                void Remove(){
                        Append("");
                        remove(_fileName.c_str());
                }
        private:
                string _fileName;
```

```
		};
	}
}

#endif
```

Helper/String

```
#ifndef Helper_String
#define Helper_String

#include <string>
#include <vector>

using namespace std;

namespace Sa{
	namespace Helper{
		class String{
			public:
				vector<string> Split(string input, char seperator){
					vector<string> parts;
					int index = 0;
					int count = 0;
					for(int i = 0; i < input.length(); i++)

					{

						if (input[i] == '|'){

							parts.push_back(input.substr(index,
count));

							index += count + 1;

							count = 0;


							continue;

						}


						count++;

					}

					if (index < input.length()){

						parts.push_back(input.substr(index,
input.length() - 1));

					}


					return parts;

				}

		};

	}

}


#endif
```

Helper/UI

```
#ifndef Helper_UI

#define Helper_UI


#include <iostream>

#include <conio.h>

#include <windows.h>

#include <string>

#include <stdlib.h>

#include "../Model/Student.h"

#include "../Bll/StudentManager.h"

#include "../Bll/CourseManager.h";

#include "../Bll/StudentPointManager.h";


using namespace std;


namespace Sa{

	namespace Helper{

		class UI{

			public:

				Initialize(){

					MainForm();
```

```
                HandleMainFormMenu();
        }

private:
        void MainForm(){
                Clear();
                TopMargin();

                cout << "\t\t1) New student" << endl << endl;
                cout << "\t\t2) New point" << endl << endl;
                cout << "\t\t3) Report" << endl << endl;
                cout << "\t\t9) Factory reset (!!!)" << endl << endl;
                cout << "\t\t0) Exit" << endl << endl;
        }

        void HandleMainFormMenu(){
                char ch;
                do {
                        ch = getch();

                        switch (ch){
                                case '1':
                                        NewStudentForm();
                                        break;
                                        case '2':
                                            NewStudentPointForm();
                                            break;
                                        case '3':
                                            ReportForm();
                                            break;
                                        case '9':
                                            Sa::Helper::File file =
Sa::Helper::File("student");
                                            file.Remove();

                                            file =
Sa::Helper::File("student_point");
                                            file.Remove();

                                            break;
                                    }
                            } while (ch != '0');
                    }

                    void NewStudentForm(){
                            Sa::Model::Student student;
                            Sa::Bll::StudentManager studentManager;

                            Clear();
```

```
                                        cout << endl << endl << "\t\t*** To cancel process,
enter -1 ***";

                                        TopMargin();


                                        cout << "\t\tStudent Id: ";

                                        cin >> student.Id;

                                        cout << endl << endl;


                                        if (student.Id == "-1"){

                                                MainForm();

                                                return;

                                        }


                                        if (studentManager.IsDuplicate(student)){

                                                cout << "\t\tStudent id exists. try again ...";

                                                getch();


                                                MainForm();


                                                return;

                                        }


                                        cout << "\t\tFull name: ";

                                        cin >> student.Name;

                                        cout << endl << endl;
```

```
        if (student.Name == "-1"){
                MainForm();
                return;
        }

        Clear();
        TopMargin();

        studentManager.Insert(student);

        cout << "\t\tStudent inserted successfully!";

        getch();

        MainForm();
}

void CourseSelectMenu(){
        Sa::Bll::CourseManager courseManager;

        Clear();
        TopMargin();
```

```
				cout << "\t\tSelect course: " << endl << endl;


				vector<Sa::Model::Course> courses = 
courseManager.Get();
				for (int i = 0; i < courses.size(); i++)
					cout << "\t\t" << courses[i].Id << ") " 
<< courses[i].Name << endl << endl;


				cout << "\t\t0) Cancel";
			}


			void NewStudentPointForm(){
				Sa::Bll::CourseManager courseManager;


				CourseSelectMenu();


				char ch;
				do {
					ch = getch();


					string courseId(1, ch);
					if (courseManager.IsValid(courseId)){
						NewStudentPointStep2Form(courseId);
						break;
					}
```

```
		} while (ch != '0');

		MainForm();
	}

	void NewStudentPointStep2Form(string courseId){
		Sa::Bll::CourseManager courseManager;
		Sa::Bll::StudentManager studentManager;
		Sa::Bll::StudentPointManager studentPointManager;

		do{
			string studentId;
			string point;

			Clear();
			cout << endl << endl << "\t\t*** To cancel
process, enter -1 ***";
			TopMargin();

			cout << "\t\tStudent Id: ";
			cin >> studentId;
			if (studentId == "-1")
				break;

			Sa::Model::StudentPoint studentPoint;
```

```
studentPoint.CourseId = courseId;

studentPoint.StudentId = studentId;


if
(studentPointManager.IsDuplicate(studentPoint)){

        cout << "\t\tYou've entered point to
this student id before.";

        getch();

        continue;

}


if (!studentManager.IsValid(studentId)){

        cout << "\t\tStudent id is not valid! Try
again ..." << endl;

        getch();


        continue;

}


Sa::Model::Student student =
studentManager.GetById(studentId);

cout << "\t\t" << student.Name << endl << endl
<< "\t\tPoint: ";

cin >> point;

if (studentId == "-1")

        break;
```

```
				Clear();
				TopMargin();

				studentPoint.Point = point;

				studentPointManager.Insert(studentPoint);

				cout << "\t\tPoint inserted successfully! try
another one ...";

				getch();
			} while (true);
		}

		void ReportForm(){
			Sa::Bll::CourseManager courseManager;

			CourseSelectMenu();

			char ch;
			do {
				ch = getch();

				string courseId(1, ch);
```

```
                        if (courseManager.IsValid(courseId)){
                                ReportStep2Form(courseId);
                                break;
                        }
                } while (ch != '0');

                MainForm();
        }

        void ReportStep2Form(string courseId){
                Sa::Bll::CourseManager courseManager;
                Sa::Bll::StudentManager studentManager;
                Sa::Bll::StudentPointManager studentPointManager;

                Clear();

                cout
<< "============================================================" << endl;
                cout << courseManager.GetById(courseId).Name
<< endl;
                cout
<< "============================================================" << endl;
                gotoxy(0, 3);
                cout << "Student Id";
                gotoxy(20, 3);
```

```
                        cout << "Full name";
                        gotoxy(40, 3);
                        cout << "Point";
                        gotoxy(0, 4);

                        vector<Sa::Model::StudentPoint> studentPoints =
studentPointManager.Get();

                        double passedSum = 0;
                        double notPassedSum = 0;
                        int passedCount = 0;
                        int notPassedCount = 0;
                        for(int i = 0; i < studentPoints.size(); i++){
                                Sa::Model::StudentPoint studentPoint =
studentPoints[i];
                                Sa::Model::Course course =
courseManager.GetById(studentPoint.CourseId);

                                if (course.Id != courseId)
                                        continue;

                                Sa::Model::Student student =
studentManager.GetById(studentPoint.StudentId);

                                gotoxy(0, 4 + i);
                                cout << student.Id;
```

```
                                        gotoxy(20, 4 + i);

                                        cout << student.Name;

                                        gotoxy(40, 4 + i);

                                        cout << studentPoint.Point;


                                        double doublePoint;

                                    sscanf ( studentPoint.Point.c_str(), "%lf" ,
&doublePoint);


                                        if (doublePoint > 12){

                                             passedSum += doublePoint;

                                             passedCount++;

                                        }

                                        else{

                                             notPassedSum += doublePoint;

                                             notPassedCount++;

                                        }

                                   }


                                   if (studentPoints.size() > 0){

                                        cout << endl
<< "============================================================" << endl;

                                        if (passedCount > 0)

                                             cout << "Passed average: "
<< passedSum / passedCount << endl;
```

```
                        if (notPassedCount > 0)
                                cout << "Not passed average: "
<< notPassedSum / notPassedCount << endl;
                }

                getch();

                MainForm();
        }

        void TopMargin(){
                cout << endl << endl << endl << endl << endl << endl;
        }

        void Clear()
        {
        #if defined _WIN32
          system("cls");
          //clrscr(); // including header file : conio.h
        #elif defined (__LINUX__) || defined(__gnu_linux__) ||
defined(__linux__)
          system("clear");
          //std::cout<< u8"\033[2J\033[1;1H"; //Using ANSI Escape
Sequences
        #elif defined (__APPLE__)
          system("clear");
        #endif
        }

        void gotoxy( int column, int line )
        {
                COORD coord;
                coord.X = column;
                coord.Y = line;
                SetConsoleCursorPosition(
                        GetStdHandle( STD_OUTPUT_HANDLE ),
                        coord
                );
        }

        int wherex()
        {
                CONSOLE_SCREEN_BUFFER_INFO csbi;
                if (!
GetConsoleScreenBufferInfo(GetStdHandle( STD_OUTPUT_HANDLE ),&csbi))
                return -1;
                return csbi.dwCursorPosition.X;
        }
```

```
                                        int wherey()
                                        {
                                                  CONSOLE_SCREEN_BUFFER_INFO csbi;
                                                  if (!
GetConsoleScreenBufferInfo(GetStdHandle( STD_OUTPUT_HANDLE ), &csbi))
                                                  return -1;
                                                  return csbi.dwCursorPosition.Y;
                                        }
                    };
          }
}

#endif
```

Model/Course

```
#ifndef Model_Course
#define Model_Course

#include <string>

using namespace std;

namespace Sa{
          namespace Model{
                struct Course {
                        string Id;
                        string Name;

                        Course(){
                                Id = "";
                                Name = "";
                        }
                };
        }
}

#endif
```

Model/Student

```
#ifndef Model_Student
#define Model_Student

#include <string>

using namespace std;

namespace Sa{
        namespace Model{

                struct Student{

                        string Id;

                        string Name;


                        Student(){

                                Id = "";

                                Name = "";

                        }

                };

        }

}


#endif
```

Model/StudentPoint

```
#ifndef Model_StudentPoint

#define Model_StudentPoint


#include <string>


using namespace std;
```

```
namespace Sa{
	namespace Model{
		struct StudentPoint {
			string CourseId;
			string StudentId;
			string Point;

			StudentPoint(){
				CourseId = "";
				StudentId = "";
				Point = "";
			}
		};
	}
}

#endif
```

Bll/CourseManager

```
#ifndef Bll_CourseManager
#define Bll_CourseManager
```

```
#include "../Model/Course.h"


namespace Sa{
	namespace Bll{
		class CourseManager{
			public:
				vector<Sa::Model::Course> Get(){
					vector<Sa::Model::Course> result;

					Sa::Model::Course c1;
					c1.Id = "1";
					c1.Name = "Math";
					result.push_back(c1);

					c1.Id = "2";
					c1.Name = "Algorithms";
					result.push_back(c1);

					c1.Id = "3";
					c1.Name = "Data Structure";
					result.push_back(c1);

					c1.Id = "4";
					c1.Name = "Statistics";
		result.push_back(c1);


		return result;
	}


	Sa::Model::Course GetById(string courseId){
		Sa::Model::Course result;
		vector<Sa::Model::Course> courses = Get();


		for(int i = 0; i < courses.size(); i++){
			if (courses[i].Id == courseId)
			{
				result = courses[i];
				break;
			}
		}


		return result;
	}


	bool IsValid(string courseId){
		Sa::Model::Course course = GetById(courseId);


		if (course.Id == "")
```

```
                                        return false;

                                return true;
                        }
                };
        }
}

#endif
```

Bll/StudentManager

```
#ifndef Bll_StudentManager
#define Bll_StudentManager

#include "../Model/Student.h"
#include "../Helper/File.h"
#include "../Helper/String.h"
#include <vector>

namespace Sa{
        namespace Bll{
                class StudentManager{
```

```
                public:
                        void Insert(Sa::Model::Student student){
                                Sa::Helper::File db = Sa::Helper::File(dbName);

                                db.Append(string(student.Id) + "|" +
string(student.Name));
                        }

                        vector<Sa::Model::Student> Get(){
                                Sa::Helper::File db = Sa::Helper::File(dbName);

                                vector<Sa::Model::Student> result;

                                vector<string> lines = db.GetLines();

                                for(int i = 0; i < lines.size(); i++)
                                {
                                        Sa::Helper::String stringHelper;
                                        Sa::Model::Student student;
                                        string line = lines[i];
                                        vector<string> parts = stringHelper.Split(line,
'|');

                                        student.Id = parts[0];
                                        student.Name = parts[1];
                                        result.push_back(student);
                                }

                                return result;
                        }

                        Sa::Model::Student GetById(string studentId){
                                Sa::Model::Student result;
                                vector<Sa::Model::Student> students = Get();

                                for(int i = 0; i < students.size(); i++){
                                        if (students[i].Id == studentId)
                                        {
                                                result = students[i];
                                                break;
                                        }
                                }

                                return result;
                        }

                        bool IsValid(string studentId){
                                Sa::Model::Student student = GetById(studentId);
```

```
                if (student.Id == "")
                    return false;

                return true;
            }

            bool IsDuplicate(Sa::Model::Student student){
                vector<Sa::Model::Student> students = Get();
                bool isDuplicate = false;

                for(int i = 0; i < students.size(); i++){
                    if (student.Id.compare(students[i].Id) == 0){
                        isDuplicate = true;
                        break;
                    }
                }

                return isDuplicate;
            }
        private:
            const string dbName = "student";
    };
}
```

```
}

#endif
```

Bll/StudentPointManager

```
#ifndef Bll_StudentPointManager
#define Bll_StudentPointManager

#include <string>
#include "../Model/StudentPoint.h"

using namespace std;

namespace Sa{
	namespace Bll{
		class StudentPointManager{
			public:
				void Insert(Sa::Model::StudentPoint studentPoint){
					Sa::Helper::File db = Sa::Helper::File(dbName);

					db.Append(string(studentPoint.CourseId) + "|" +
string(studentPoint.StudentId) + "|" + studentPoint.Point);
				}
```

```
vector<Sa::Model::StudentPoint> Get(){
        Sa::Helper::File db = Sa::Helper::File(dbName);

        vector<Sa::Model::StudentPoint> result;

        vector<string> lines = db.GetLines();

        for(int i = 0; i < lines.size(); i++)
        {
                Sa::Helper::String stringHelper;
                Sa::Model::StudentPoint studentPoint;
                string line = lines[i];
                vector<string> parts = stringHelper.Split(line,
'|');

                studentPoint.CourseId = parts[0];
                studentPoint.StudentId = parts[1];
                studentPoint.Point = parts[2];

                result.push_back(studentPoint);
        }

        return result;
}
```

```
                    bool IsDuplicate(Sa::Model::StudentPoint studentPoint){

                        vector<Sa::Model::StudentPoint> studentPoints = Get();

                        bool isDuplicate = false;


                        for(int i = 0; i < studentPoints.size(); i++){

                            if
(studentPoint.CourseId.compare(studentPoints[i].CourseId) == 0 &&
studentPoint.StudentId.compare(studentPoints[i].StudentId) == 0){

                                isDuplicate = true;

                                break;

                            }

                        }


                        return isDuplicate;

                    }


                private:

                    const string dbName = "student_point";

            };

    }

}


#endif
```