Выполнила: Белоусова Е., ИП-911

Задача

Написать программу для манипуляции данными на основе рекурсивных структур, реализовать функции вставки, удаления и навигации для списка; реализовать сериализацию списка. (Например: электронный журнал успеваемости студентов.)

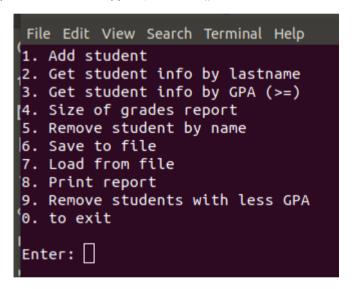
Описание работы программы

Журнал успеваемости в программе реализован классом GradesReport.

Он содержит вложенную структуру Student для информации об отдельном студенте, методы для сериализации и десериализации, печать информации о студенте; также GradesReport имеет указатель на список типа List, состоящий из объектов структуры Student.

Класс List является шаблонным односвязным списком, содержит рекурсивный класс Node (содержит поле данных и указатель на следующий элемент), разнообразные методы для работы со списком (удаление узлов, вставка узлов, получение последнего, получение размера списка, проверка на пустоту).

В начале работы программа вызывает функцию menu():

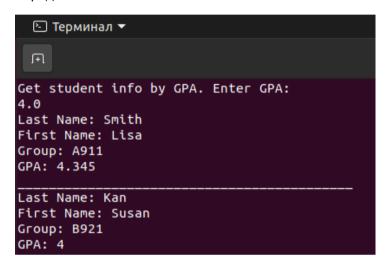


При добавлении студента пользователю предлагается ввести имя, фамилию, группу и средний балл. Далее все эти данные заполняют поля объекта, объект добавляется в конец списка.

Для получения информации о студенте пользователю необходимо ввести фамилию студента. В цикле перебирается список, получая по порядку элементы и сравнивая фамилии. В случае совпадения печатается информация.

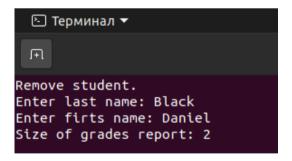


Для получения информации о студентах, у которых средний балл выше определенного значения, пользователю необходимо ввести это некоторое значение. Программа по порядку получает объекты и сравнивает средний балл.



Пользователь может получить информацию о размере журнала. Класс List имеет приватную переменную, в которой отслеживает размер списка, также имеет метод для получения размера списка.

Пользователь может удалить студента по его имени и фамилии.



Также студента можно удалить, если его средний балл ниже определенного.

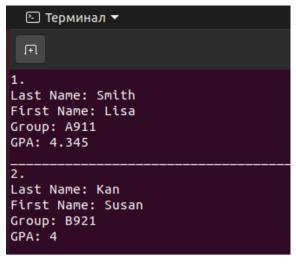
```
File Edit View Search Terminal Help

Enter GPA:

4.0

1 report(s) have been deleted (GPA below: 4).
```

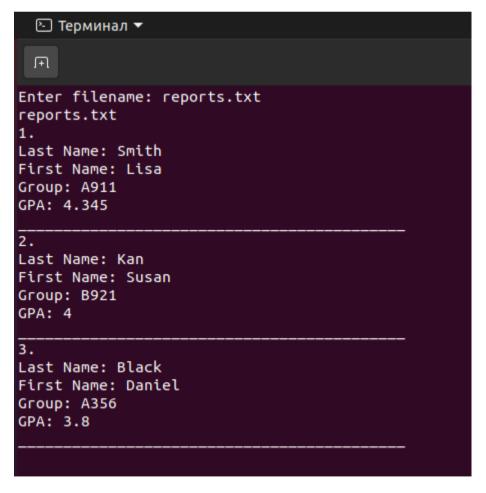
Можно распечатать весь журнал.



Пользователь может сохранить свой файл. Тогда данные о студентах будут храниться в виде серий, разделяемых ';':



Также можно загрузить свой файл для работы программы, но только если он будет записан в таком же формате. После загрузки информации из файла, выведется весь журнал успеваемости:



Листинг

//list.h

#pragma once

```
#include <iostream>
#include <cstring>
#include <sstream>
#include <exception>
template <typename T>
class List
        class Node
        public:
                T* data;
                Node* next{ nullptr };
                Node(): data(), next{ nullptr }{};
                Node(T* _data) : data(_data), next(nullptr) {}
                Node(T* _data, Node* _next) : data(_data), next(_next) {};
                ~Node() {delete next;};
        };
        Node* _head = nullptr;
        size t size = 0;
public:
        bool isEmpty();
        size_t getSize();
        void pushBack(T* data);
        Node* getLast();
        void pushFront(T* data);
        void popFront();
        void popBack();
        void deleteNode(T* data);
        T* getObjByIndex(int idx);
        void deserialize(std::string str);
        std::string serialize();
        void deleteAt(int idx);
        ~List()
        {
                for(size_t i = 0; i < _size; i++)
                        popFront();
                delete _head;
        }
};
template <typename T>
bool List<T>::isEmpty()
{
        return !(0 < _size);
}
template <typename T>
size_t List<T>::getSize()
{
        return _size;
}
```

```
template <typename T>
void List<T>::pushBack(T* data)
       Node* last = getLast();
       Node* tmp = new Node(data);
       if (last)
       {
               last->next = tmp;
       }
       else
       {
               _head = tmp;
       }
       _size++;
}
template <typename T>
typename List<T>::Node* List<T>::getLast()
       if (!_head)
       {
               return nullptr;
       Node* tmp = _head;
       while (tmp->next)
               tmp = tmp->next;
       return tmp;
}
template <typename T>
void List<T>::pushFront(T* data)
{
       Node* tmp = new Node(data, _head);
       _head = tmp;
       _size++;
       delete tmp;
}
template <typename T>
void List<T>::popFront()
       if (isEmpty())
       {
               return;
       Node* tmp = _head;
       _head = tmp->next;
       _size--;
       delete tmp;
}
template <typename T>
void List<T>::popBack()
```

```
if (isEmpty())
       {
               return;
       }
       Node* last = getLast();
       if (last == _head)
       {
               popFront();
       }
       Node* tmp = head;
       while (tmp->next != last)
       {
               tmp = tmp->next;
       }
       tmp->next = nullptr;
        size--;
       delete last;
//GradesReport.h
#pragma once
#include <cstring>
#include <iostream>
#include "List.h"
class GradesReport
{
       struct Student
               std::string lastName;
               std::string firstName;
               std::string group;
               double GPA = 0;
               virtual std::string Serialize();
               virtual void Deserialize(char* str);
               void PrintStudentInfo();
               ~Student(){};
       };
       List<Student>* list = new List<Student>();
public:
       void Add(std::string lstname, std::string fstname, std::string grp, double gpa);
       void RemoveByName(std::string Istname, std::string fstname);
       int RemoveByLessGPA(double lessThGPA);
       void GetStudentInfoByLastName(std::string lstname);
       void GetStudentInfoByGPA(double GPA);
       int NumberOfStudents();
       void Save(std::string path);
       void Load(std::string path);
       void Print();
};
//GradesReport.cpp
#include "GradesReport.h"
#include <sstream>
#include <fstream>
#include <cstdio>
```

```
std::string GradesReport::Student::Serialize()
       std::stringstream ss;
       ss << lastName << "|" << GPA;
       return ss.str();
void GradesReport::Student::Deserialize(char* str)
       lastName = strtok(str, "|");
       firstName = strtok(NULL, "|");
       group = strtok(NULL, "|");
       std::string _gpa = strtok(NULL, "");
       GPA = atof(_gpa.c_str());
}
void GradesReport::Student::PrintStudentInfo()
{
       std::cout << "Last Name: " << lastName << std::endl;
       std::cout << "First Name: " << firstName << std::endl;
       std::cout << "Group: " << group << std::endl;</pre>
       std::cout << "GPA: " << GPA << std::endl;
       std::cout << "
                                                                         " << std::endl;
}
void GradesReport::Add(std::string lstname, std::string fstname, std::string grp, double gpa)
       Student* tmp = new Student();
       tmp->lastName = lstname;
       tmp->firstName = fstname;
       tmp->group = grp;
       tmp->GPA = gpa;
       list->pushBack(tmp);
}
void GradesReport::RemoveByName(std::string Istname, std::string fstname)
       if (list->getSize() <= 0)
       {
               return;
       for (int i = 0; i < (int)list->getSize(); i++)
       {
               if (lstname == list->getObjByIndex(i)->lastName)
               {
                       if (fstname == list->getObjByIndex(i)->firstName)
                       {
                               list->deleteAt(i);
                               return;
                       }
               }
       }
}
```

```
int GradesReport::RemoveByLessGPA(double lessThGPA)
        int count = 0;
        if (list->getSize() <= 0)
                return 0;
        int sizeOfRep = (int)list->getSize();
        for (int i = 0; i < sizeOfRep; i++)
                if (lessThGPA > list->getObjByIndex(i)->GPA)
                        list->deleteAt(i);
                        count++;
                        sizeOfRep--;
                }
        }
        return count;
}
void GradesReport::GetStudentInfoByLastName(std::string lstname)
        if (list->getSize() <= 0)
        {
                return;
        for (int i = 0; i < (int)list->getSize(); i++)
                if (lstname == list->getObjByIndex(i)->lastName)
                {
                        list->getObjByIndex(i)->PrintStudentInfo();
                        return;
        }
}
void GradesReport::GetStudentInfoByGPA(double GPA)
        if (list->getSize() <= 0)
        {
                return;
        for (int i = 0; i < (int)list->getSize(); i++)
                if (GPA <= list->getObjByIndex(i)->GPA)
                        list->getObjByIndex(i)->PrintStudentInfo();
        }
}
int GradesReport::NumberOfStudents()
        return (int)list->getSize();
}
```

```
void GradesReport::Save(std::string path)
        std::ofstream outf(path);
        if (!outf)
        {
                std::cerr << "This file can't be open!" << std::endl;
                exit(1);
        outf << list->serialize();
}
void GradesReport::Load(std::string path)
        std::ifstream inf(path, std::ios::binary);
        if (!inf)
        {
                std::cerr << "This file can't be open!" << std::endl;
                exit(1);
        inf.seekg(0, std::ios::end);
        size_t filesize = inf.tellg();
        inf.seekg(0);
        std::string dataFromFile(filesize, ' ');
        inf.read(&dataFromFile[0], filesize);
        list->deserialize(dataFromFile);
}
void GradesReport::Print()
        if (list->getSize() <= 0)
        {
                return;
        for (int i = 0; i < (int)list->getSize(); i++)
        {
                std::cout << i + 1 << ". " << std::endl;
                list->getObjByIndex(i)->PrintStudentInfo();
//osLab1.cpp
#include <cstdlib>
#include <cstdio>
#include <stdio.h>
#include <iostream>
#include <cstring>
#include <sstream>
#include <fstream>
#include "GradesReport.h"
using namespace std;
GradesReport* report = new GradesReport();
void menu()
        system("clear");
        while(1)
```

```
//system("clear");
cout << "1. Add student" << endl;</pre>
cout << "2. Get student info by lastname" << endl;
cout << "3. Get student info by GPA (>=)" << endl;
cout << "4. Size of grades report" << endl;
cout << "5. Remove student by name" << endl;
cout << "6. Save to file" << endl;
cout << "7. Load from file" << endl;
cout << "8. Print report" << endl;
cout << "9. Remove students with less GPA" << endl;
cout << "0. to exit" << endl;
cout << endl << "Enter: ";
int n;
cin >> n;
system("clear");
if (n == 0)
{
        delete report;
        break;
}
if (n == 1)
{
        system("clear");
        string tLN, tFN, tG;
        double gpa = 0;
        cout << "Enter lastname: ";</pre>
        cin >> tLN;
        cout << "Enter firstname: ";
        cin >> tFN;
        cout << "Enter group: ";
        cin >> tG;
        cout << "Enter gpa: ";
        cin >> gpa;
        report -> Add(tLN, tFN, tG, gpa);
        cout << endl << endl;
}
if (n == 2)
        system("clear");
        string tLN;
        cout << "Get student info by lastname. Enter lastname: " << endl;</pre>
        cin >> tLN;
        cout << endl;
        report -> GetStudentInfoByLastName(tLN);
        cout << endl << endl;
}
if (n == 3)
        system("clear");
        cout << "Get student info by GPA. Enter GPA: " << endl;
        double gpa = 0;
        cin >> gpa;
        report->GetStudentInfoByGPA(gpa);
        cout << endl << endl;
if (n == 4)
```

```
{
                        system("clear");
                        cout << "Size of grades report: " << report->NumberOfStudents() << endl <<
endl;
                if (n == 5)
                {
                        system("clear");
                        string tLN, tFN;
                        cout << "Remove student. " << endl;
                        cout << "Enter last name: ";
                        cin >> tLN;
                        cout << "Enter firts name: ";</pre>
                        cin >> tFN;
                        report->RemoveByName(tLN, tFN);
                        cout << "Size of grades report: " << report->NumberOfStudents() << endl <<
endl;
                if (n == 6)
                        system("clear");
                        cout << "Enter filename: ";
                        string filename;
                        cin >> filename;
                        cout<<filename<<endl;
                        if (report->NumberOfStudents() > 0) report -> Save(filename);
                        cout << "Saved!" << endl << endl;</pre>
                if (n == 7)
                {
                        system("clear");
                        cout << "Enter filename: ";
                        string filename;
                        cin >> filename;
                        cout<<filename<<endl;
                        if (report->NumberOfStudents() > 0)
                        {
                                delete report;
                                report = new GradesReport();
                        report -> Load(filename);
                        report -> Print();
                        cout << endl << endl;
                if (n == 8)
                {
                        system("clear");
                        report->Print();
                        cout << endl << endl;
                }
                if (n == 9)
                        system("clear");
                        double gpa = 0;
                        cout << "Enter GPA: " << endl;
                        cin >> gpa;
```

```
cout << report->RemoveByLessGPA(gpa) << " report(s) have been deleted (GPA
below: " << gpa << ")." << endl << endl;
               }
       }
}
int main()
       // GradesReport* report = new GradesReport();
       // report->Add("Smith", "Lisa", "A911", 4.345);
       // report->Add("Kan", "Daniel", "A911", 3.8);
       // report->Add("Kan", "Susan", "B921", 4.0);
       // report->Print();
       // cout << "========" << endl;
       // cout << "Get student info by lastname: " << endl;</pre>
       // report->GetStudentInfoByLastName("Smith");
       // cout << "========" << endl;
       // cout << "Get student info by GPA: " << endl;
       // report->GetStudentInfoByGPA(4.0);
       // cout << "========" << endl;
       // cout << "Size of grades report: " << report->NumberOfStudents() << endl;</pre>
       // cout << report->RemoveByLessGPA(4.0) << " report(s) have been deleted (GPA below: " << 4.0
<< ")." << endl;
       // cout << "========= << endl;
       // report->Add("Black", "Michael", "A913", 4.2);
       // cout << "Size of grades report: " << report->NumberOfStudents() << endl;</pre>
       // report->RemoveByName("Black", "Michael");
       // cout << "========" << endl;
       // cout << "Size of grades report: " << report->NumberOfStudents() << endl;</pre>
       // report->Save("reports.txt");
       // cout << "========" << endl;
       // GradesReport* reportTMP = new GradesReport();
       // reportTMP->Load("reports.txt");
       // reportTMP->Print();
       // delete reportTMP;
       // delete report;
       menu();
}
```