**3-nd assignment, Makhmutov Eldar, Balkiyaeva Amira**

**(Please turn on viewer mode to web-site for better experience, right bottom corner in Word-App)**

**Github: https://github.com/Sintelaris/adv-assignment-4**

**1 task.**

**Class Course has been created (header only):**

using namespace std;  
  
class Course {  
private:  
  
 string name;  
 int students;  
 Lecturer lecturer;  
public:  
  
 Course(){};  
 Course(string name, int students, Lecturer lecturer);  
  
 const string &getName() const;  
  
 void setName(const string &name);  
  
 int getStudents() const;  
  
 void setStudents(int students);  
  
 const Lecturer &getLecturer() const;  
  
 void setLecturer(const Lecturer &lecturer);  
  
 friend ostream &operator<<( ostream &output, const Course &course );  
  
 friend istream &operator>>( istream &input, Course &course );  
  
 friend bool operator==(const Course &lv, const Course &rv);  
};

**Activities have been created:**

**IEEE:**

using namespace std;  
  
class IEEE {  
private:  
 string confName;  
 string confLocation;  
 set<string> topicsOfPresentations = {"Computer Algorithm", "Computer Design", "Industrial Automation", "Intrusion Detection System",  
 "IoT Internet of Things"};  
  
public:  
 IEEE(){}  
  
 IEEE(const string &confName, const string &confLocation, const set<string> &topicsOfPresentations);  
  
 const string &getConfName() const;  
  
 void setConfName(const string &confName);  
  
 const string &getConfLocation() const;  
  
 void setConfLocation(const string &confLocation);  
  
 const set<string> &getTopicsOfPresentations() const;  
  
 void setTopicsOfPresentations(const set<string> &topicsOfPresentations);  
  
 friend ostream &operator<<( ostream &output, const IEEE &ieee );  
  
 friend istream &operator>>( istream &input, IEEE &ieee );  
  
};

**ACM:**

using namespace std;  
  
class ACM {  
private:  
 vector<string> topicOfTheDay = {"Models of computation", "Formal languages and automata theory", "Computational complexity and cryptography",  
 "Theory and algorithms for application domains", "Randomness, geometry and discrete structures"};  
 map<string,string> location;  
 int numOfPresentations;  
public:  
 ACM();  
  
 void addTopicOfTheDay(const string&);  
  
 const string & getTopicOfTheDay() const;  
  
 const map<string, string> &getLocation() const;  
  
 void addLocation(const string&, const string&);  
  
 int getNumOfPresentations() const;  
  
 void setNumOfPresentations(int numOfPresentations);  
  
 const int random(int number) const;  
  
 friend ostream &operator<<( ostream &output, const ACM &acm );  
  
 friend istream &operator>>( istream &input, ACM &acm );  
  
};

**Templated class Student has been created (in main.cpp):**

template<typename T>  
class Student{  
private:  
 string name;  
 int id;  
 int age;  
 vector<Course> courses;  
 T activities;  
  
public:  
  
 Student() {}  
  
 const string &getName() const {  
 return name;  
 }  
  
 void setName(const string &name) {  
 Student::name = name;  
 }  
  
 int getId() const {  
 return id;  
 }  
  
 void setId(int id) {  
 Student::id = id;  
 }  
  
 int getAge() const {  
 return age;  
 }  
  
 void setAge(int age) {  
 Student::age = age;  
 }  
  
 const vector<Course> &getCourses() const {  
 return courses;  
 }  
  
 void setCourses(const vector<Course> &courses) {  
 Student::courses = courses;  
 }  
  
 const set<T> &getActivities() const {  
 return activities;  
 }  
  
 void setActivities(const set<T> &activities) {  
 Student::activities = activities;  
 }  
  
 friend ostream &operator<<( ostream &output, const Student<T> &student ) {  
 output << "\nName\t\t" << "ID\t\t\t" << "Age\t\t" << "Activities\t\t" << "Courses";  
 output << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";  
 string act = typeid(T).name();  
 output << "\n" << student.getName() << "\t\t" << student.getId() << "\t\t" << student.getAge() << "\t\t" << act.substr(1,4) << "\t\t";  
 for (auto it = student.courses.begin(); it != student.courses.end(); ++it) {  
 output << "\t" << it->getName() << "\n\t\t\t\t\t\t\t\t\t\t\t";  
 }  
 output << "\n";  
 return output;  
 }  
  
 friend istream &operator>>( istream &input, Student<T> &student ) {  
 cout << "Input ID, name, and age" << endl;  
 input >> student.id >> student.name >> student.age;  
 return input;  
 }  
  
 friend bool operator==(const Student<T> &lv, const Student<T> &rv){  
 if (lv.name == rv.name && lv.id == rv.id && lv.age == rv.age && lv.getCourses() == rv.getCourses()){  
 return true;  
 }  
 else return false;  
 }  
};

**Stream operators have has been overloaded:**

friend ostream &operator<<( ostream &output, const Student<T> &student ) {  
 output << "\nName\t\t" << "ID\t\t\t" << "Age\t\t" << "Activities\t\t" << "Courses";  
 output << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_";  
 string act = typeid(T).name();  
 output << "\n" << student.getName() << "\t\t" << student.getId() << "\t\t" << student.getAge() << "\t\t" << act.substr(1,4) << "\t\t";  
 for (auto it = student.courses.begin(); it != student.courses.end(); ++it) {  
 output << "\t" << it->getName() << "\n\t\t\t\t\t\t\t\t\t\t\t";  
 }  
 output << "\n";  
 return output;  
 }  
  
 friend istream &operator>>( istream &input, Student<T> &student ) {  
 cout << "Input ID, name, and age" << endl;  
 input >> student.id >> student.name >> student.age;  
 return input;  
 }

**Necessary operators have been overladed so std::count(), std::find(), std::for\_each(), std::count\_if() are usable:**

vector<Student<ACM>> acmstudents = {student, student2, student3};  
vector<Student<IEEE>> ieeestudents = {student1, student4, student5};  
  
  
cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;  
cout << "Count() " << count(acmstudents.begin(), acmstudents.end(), student) << endl;  
  
  
vector<Student<ACM>>::iterator it = find(acmstudents.begin(), acmstudents.end(), student);  
if (it != acmstudents.end()){  
 cout << "Find():\n";  
 cout << \*it;  
}  
  
cout << "For\_each() " << for\_each(acmstudents.begin(), acmstudents.end(), [](Student<ACM> &x){ x.setName("Clever");})<<endl;  
  
cout << acmstudents[0] << endl;  
cout << "Count\_if() " <<count\_if(acmstudents.begin(), acmstudents.end(), [](Student<ACM> &x){return x.getName() == "Clever";})<<endl;