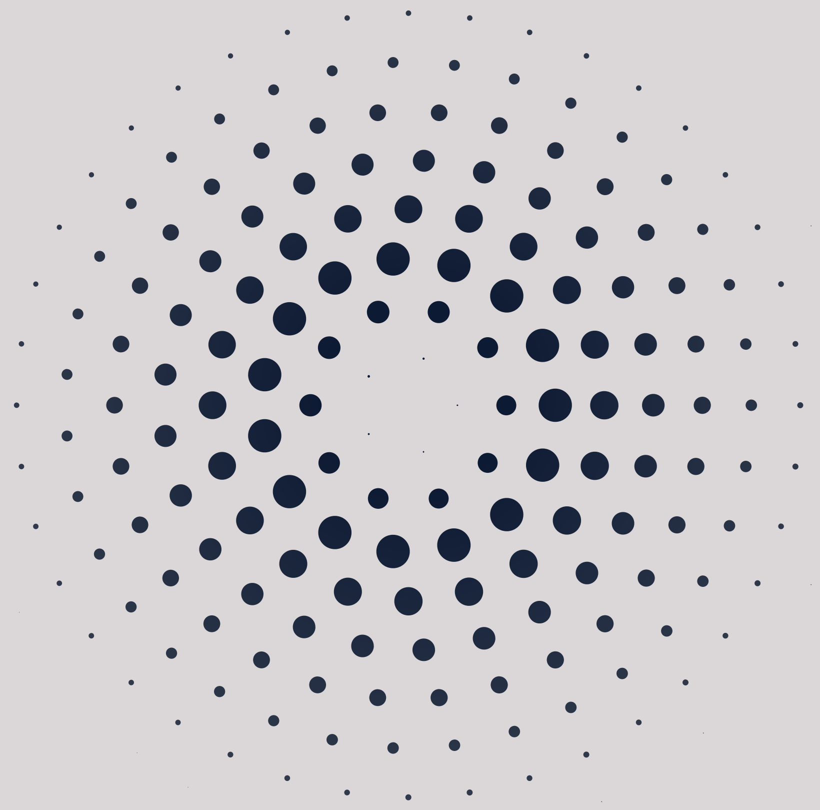
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

PROJETO 1 - HORÁRIOS DA L.EIC

ALGORITMOS E ESTRUTURAS DE DADOS

1º SEMESTRE - 2º ANO

Turma 12 - Grupo 8:
Luana Lima, 202206845
Lucas Greco, 202208296
Luís Cordeiro, 202205425



Índice

Visão geral

Este projeto tem como objetivo a criação de um programa em C++, capaz de gerir e lidar com os horários da L.EIC, possibilitando várias funcionalidades tais como: a visualização e consulta de informação relativa a alunos, turmas e UCs e a possibilidade de troca de UCs por parte de alunos.

Estruturas de Dados

VECTOR

Armazenamento de objetos

LIST

Armazenamento de objetos

STACK

Desfazer alterações

QUEUE

Armazenar por ordem de importância

BINARY SEARCH TREE

Armazenar objetos de forma ordenda

Classes

STUDENT

Armazena informações sobre os estudantes, incluindo nome e ID. Também mantém um vetor de pares contendo o ID da turma e o ID da UC associados a cada estudante.

CLASS

Armazena informações sobre as aulas, incluindo ID, duração, dia da semana, hora de início, tipo de aula e a UC associada a essa classe. UC é representada como outra classe.

UC

Mantém informações sobre unidades curriculares, como o nome da UC.

CHANGE

Nesta classe, os pedidos de trocas de turmas entre estudantes são armazenados e processados, o que implica a reorganização de alunos em turmas e UCs.

SYSTEM

É a classe central do programa, pois aqui, todas as outras classes interagem. É responsável por ler e criar arquivos, bem como armazenar listas de alunos e classes. Além disso, mantém uma pilha onde as ações realizadas pelo usuário são registradas.

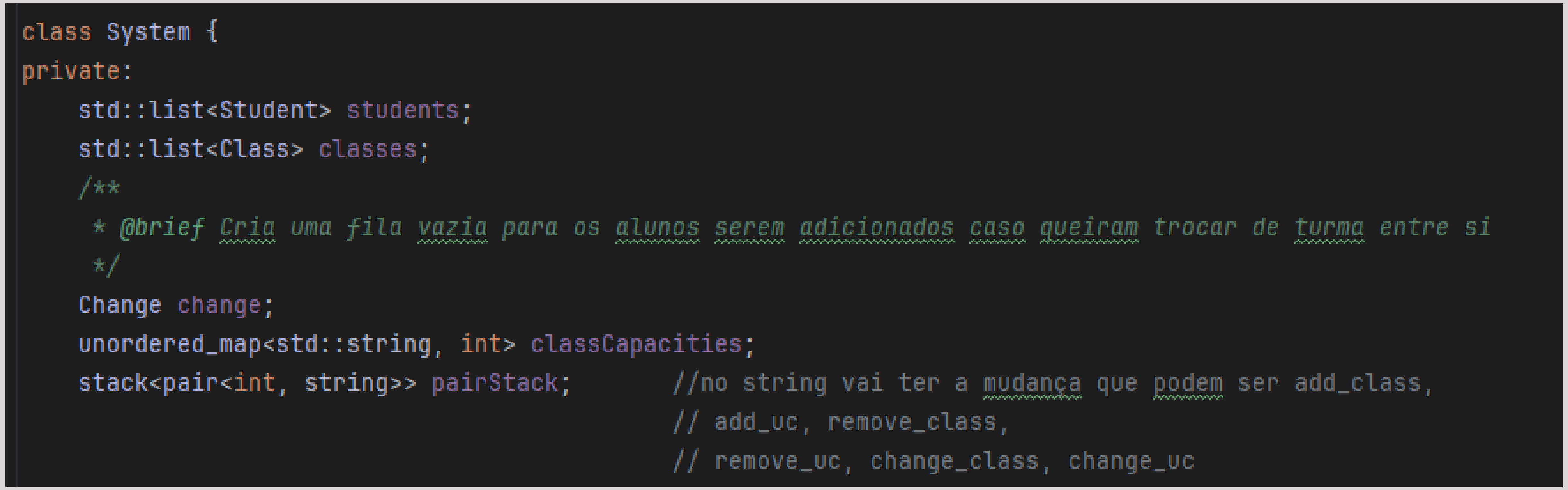
MENU

Esta classe permite que o usuário interaja com as funcionalidades do programa e exibe as saídas correspondentes.

Funcionalidades

Editar turmas e UCs dos estudantes
Obter a lotação das turmas e UCs
Obter os horários dos estudantes e das turmas
Desfazer alterações
Fazer trocas de turmas entre os estudantes
Diferentes maneiras de mostrar as informações

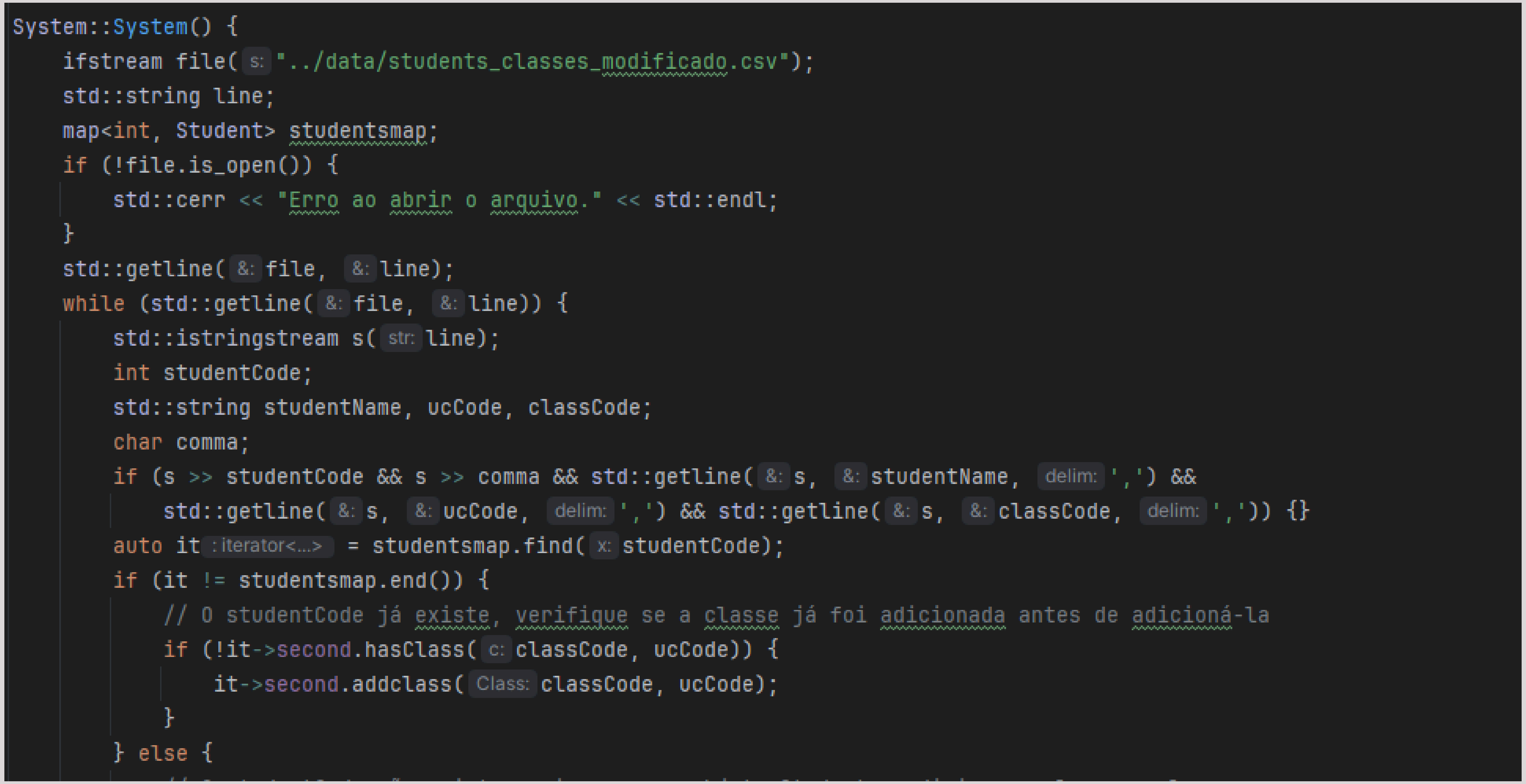
System



```
class System {
private:
    std::list<Student> students;
    std::list<Class> classes;
    /**
     * @brief Cria uma fila vazia para os alunos serem adicionados caso queiram trocar de turma entre si
     */
    Change change;
    unordered_map<std::string, int> classCapacities;
    stack<pair<int, string>> pairStack;       //no string vai ter a mudança que podem ser add_class,
                                              // add_uc, remove_class,
                                              // remove_uc, change_class, change_uc
```

Parsing

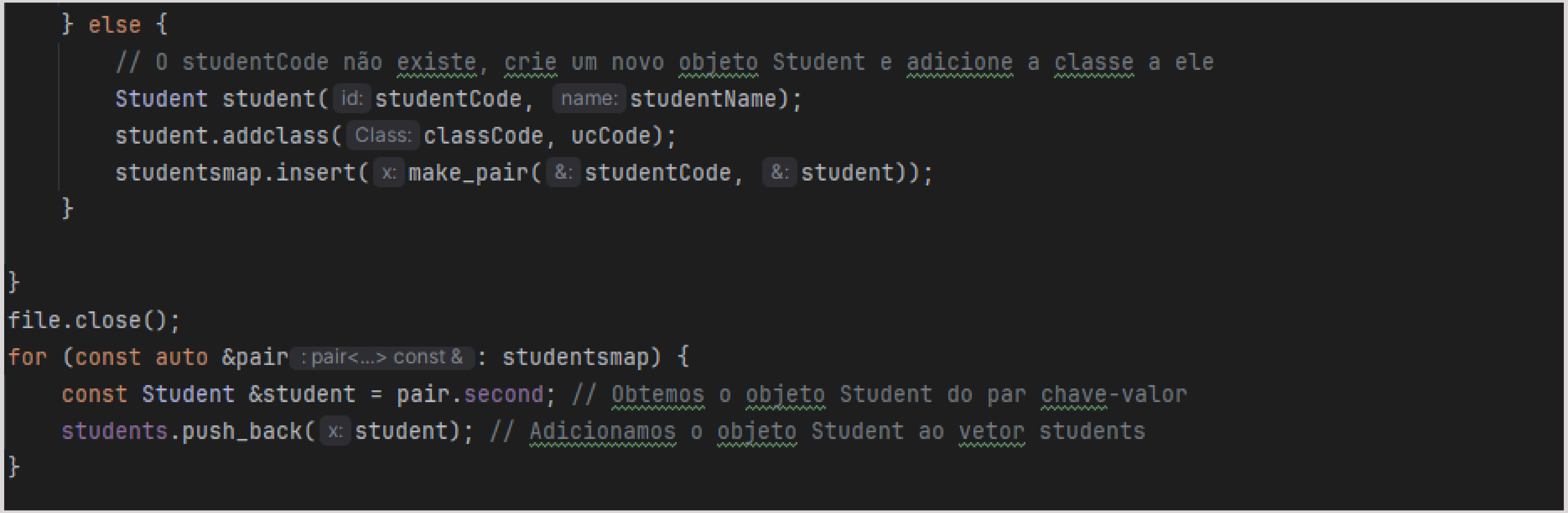
Estrutura usada: list



```
System::System() {
    ifstream file( s: "../data/students_classes_modificado.csv");
    std::string line;
    map<int, Student> studentsmap;
    if (!file.is_open()) {
        std::cerr << "Erro ao abrir o arquivo." << std::endl;
    }
    std::getline( &: file,  &: line);
    while (std::getline( &: file,  &: line)) {
        std::istringstream s( str: line);
        int studentCode;
        std::string studentName, ucCode, classCode;
        char comma;
        if (s >> studentCode && s >> comma && std::getline( &: s,  &: studentName,  delim: ',') &&
            std::getline( &: s,  &: ucCode,  delim: ',') && std::getline( &: s,  &: classCode,  delim: ',')) {}
        auto it : iterator<...> = studentsmap.find( x: studentCode);
        if (it != studentsmap.end()) {
            // O studentCode já existe, verifique se a classe já foi adicionada antes de adicioná-la
            if (!it->second.hasClass( c: classCode, ucCode)) {
                it->second.addclass( Class: classCode, ucCode);
            }
        } else {
```

Parsing

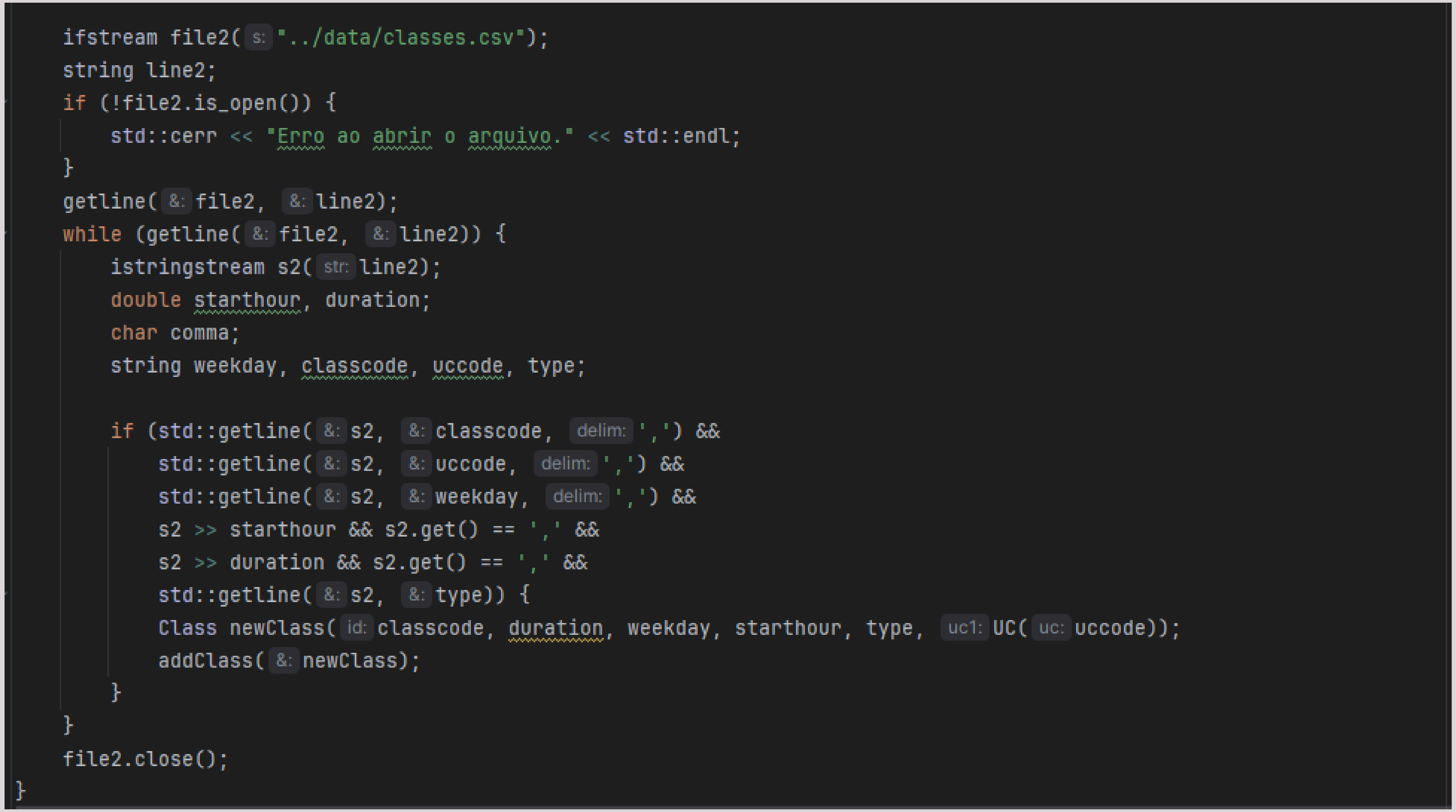
Estrutura usada: list



```
    } else {
        // O studentCode não existe, crie um novo objeto Student e adicione a classe a ele
        Student student( id: studentCode,  name: studentName);
        student.addclass( Class: classCode, ucCode);
        studentsmap.insert( x: make_pair( &: studentCode,  &: student));
    }

}
file.close();
for (const auto &pair : pair<...> const & : studentsmap) {
    const Student &student = pair.second; // Obtemos o objeto Student do par chave-valor
    students.push_back( x: student); // Adicionamos o objeto Student ao vetor students
}
```

Parsing

Estrutura usada: list

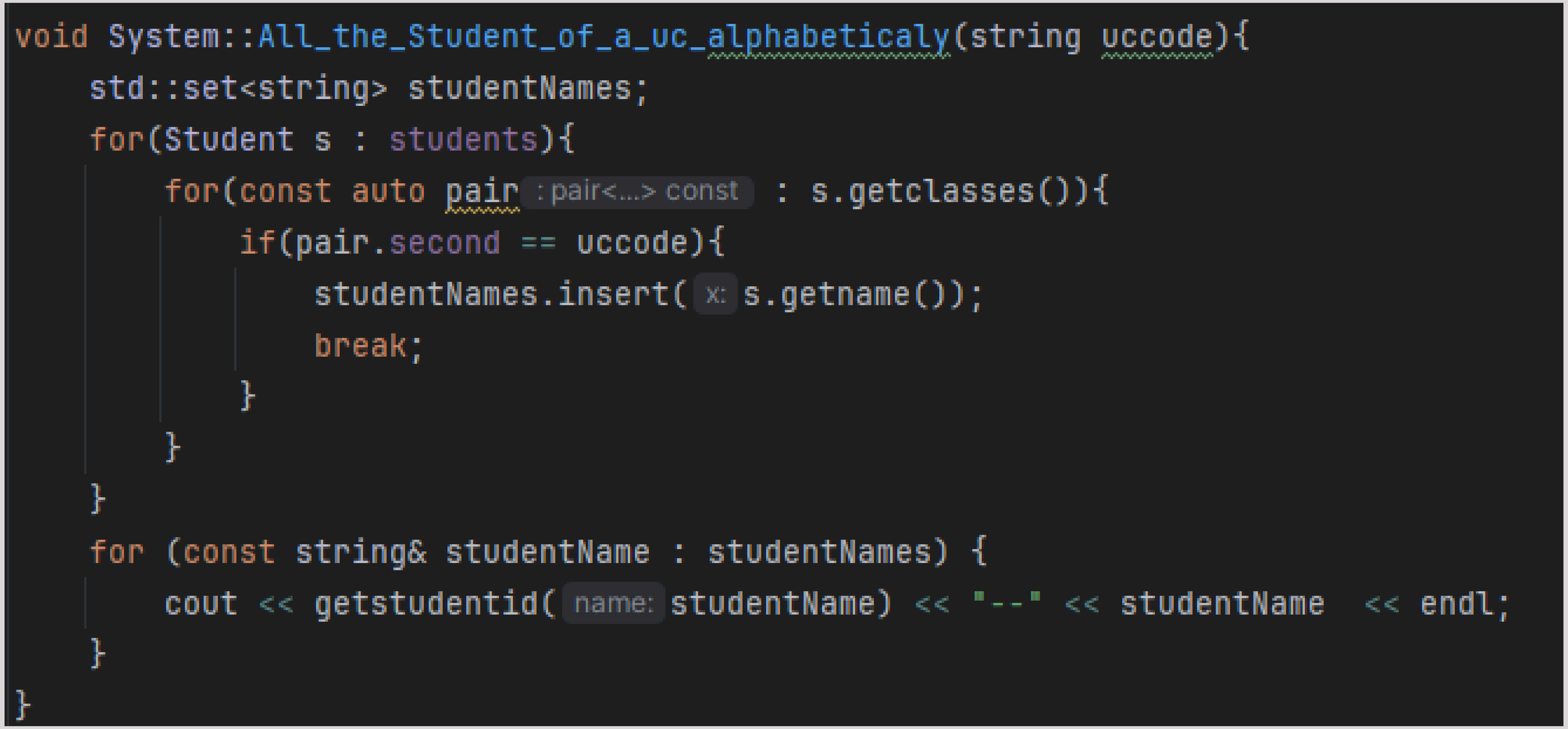


```
    ifstream file2(s: "../data/classes.csv");
    string line2;
    if (!file2.is_open()) {
        std::cerr << "Erro ao abrir o arquivo." << std::endl;
    }
    getline(&: file2, &: line2);
    while (getline(&: file2, &: line2)) {
        istringstream s2(str: line2);
        double starthour, duration;
        char comma;
        string weekday, classcode, uccode, type;

        if (std::getline(&: s2, &: classcode, delim: ',') &&
            std::getline(&: s2, &: uccode, delim: ',') &&
            std::getline(&: s2, &: weekday, delim: ',') &&
            s2 >> starthour && s2.get() == ',' &&
            s2 >> duration && s2.get() == ',' &&
            std::getline(&: s2, &: type)) {
            Class newClass(id: classcode, duration, weekday, starthour, type, uc1: UC(uc: uccode));
            addClass(&: newClass);
        }
    }
    file2.close();
}
```

Sorting

Estrutura usada: set

O(n²)

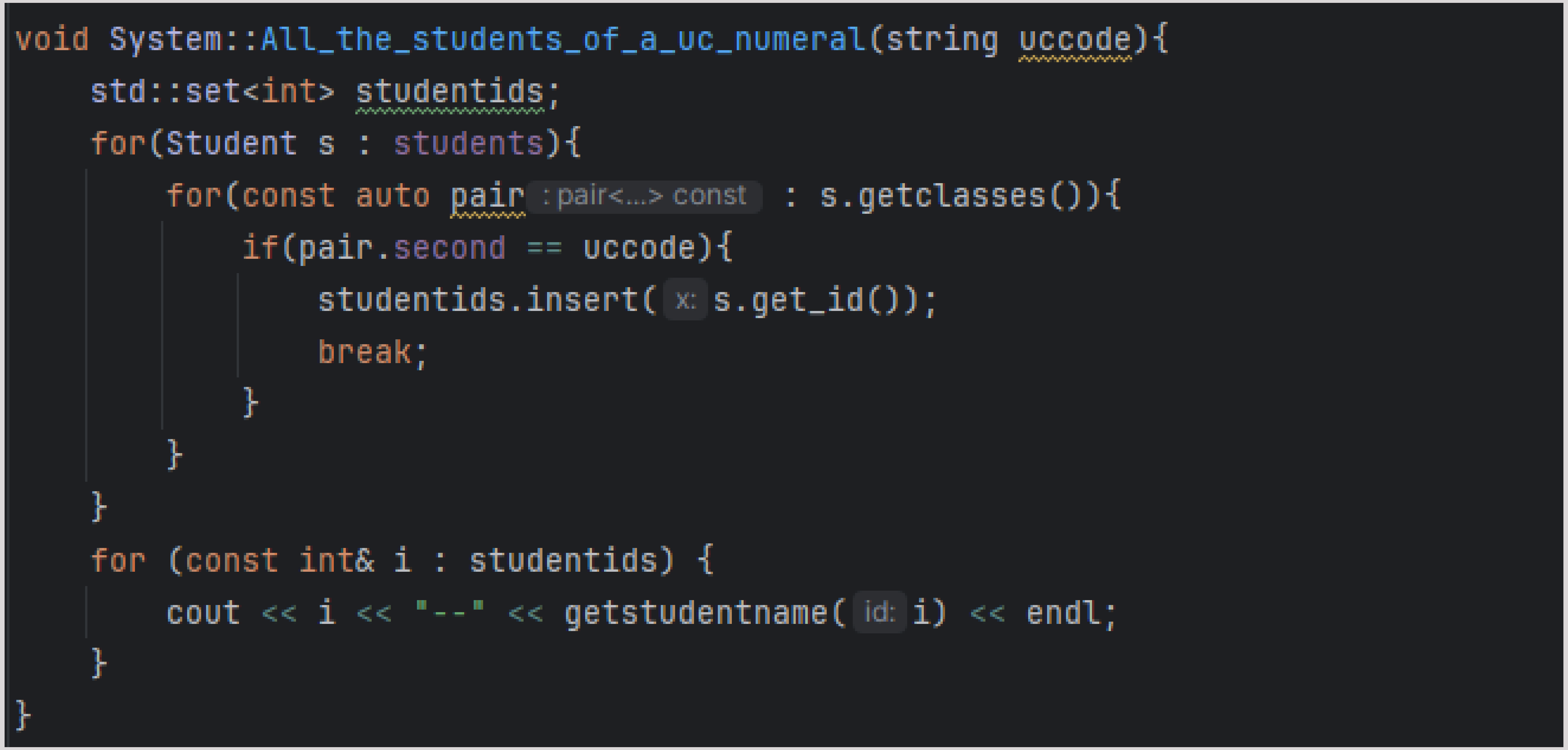


```
void System::All_the_Student_of_a_uc_alphabeticaly(string uccode){
    std::set<string> studentNames;
    for(Student s : students){
        for(const auto pair :pair<...> const : s.getclasses()){
            if(pair.second == uccode){
                studentNames.insert( x: s.getname());
                break;
            }
        }
    }
    for (const string& studentName : studentNames) {
        cout << getstudentid( name: studentName) << "--" << studentName  << endl;
    }
}
```

Sorting

Estrutura usada: set

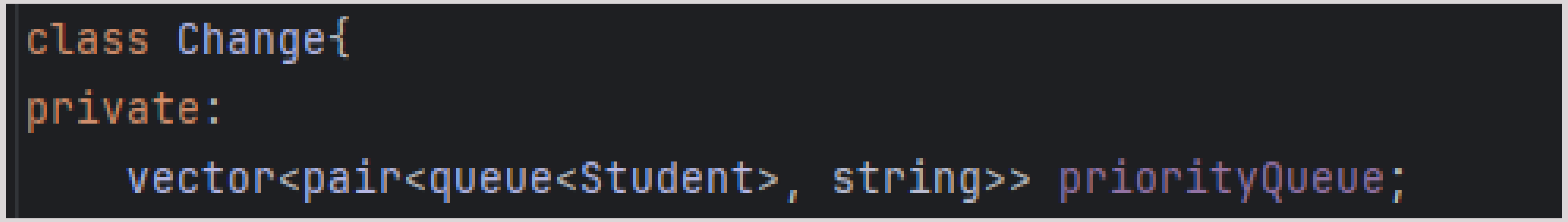
O(n²)



```
void System::All_the_students_of_a_uc_numeral(string uccode){
    std::set<int> studentids;
    for(Student s : students){
        for(const auto pair : pair<...> const  : s.getclasses()){
            if(pair.second == uccode){
                studentids.insert( x: s.get_id());
                break;
            }
        }
    }
    for (const int& i : studentids) {
        cout << i << "--" << getstudentname( id: i) << endl;
    }
}
```

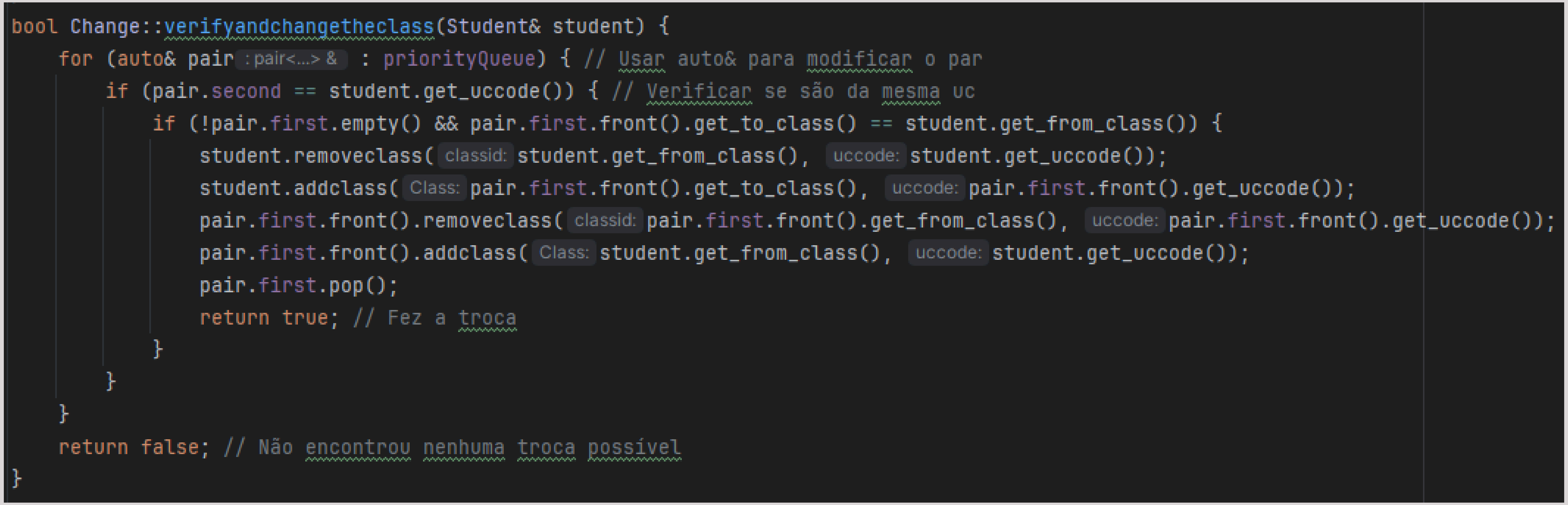
Change

Estrutura usada: queue



```
class Change{
private:
    vector<pair<queue<Student>, string>> priorityQueue;
```

O(n)

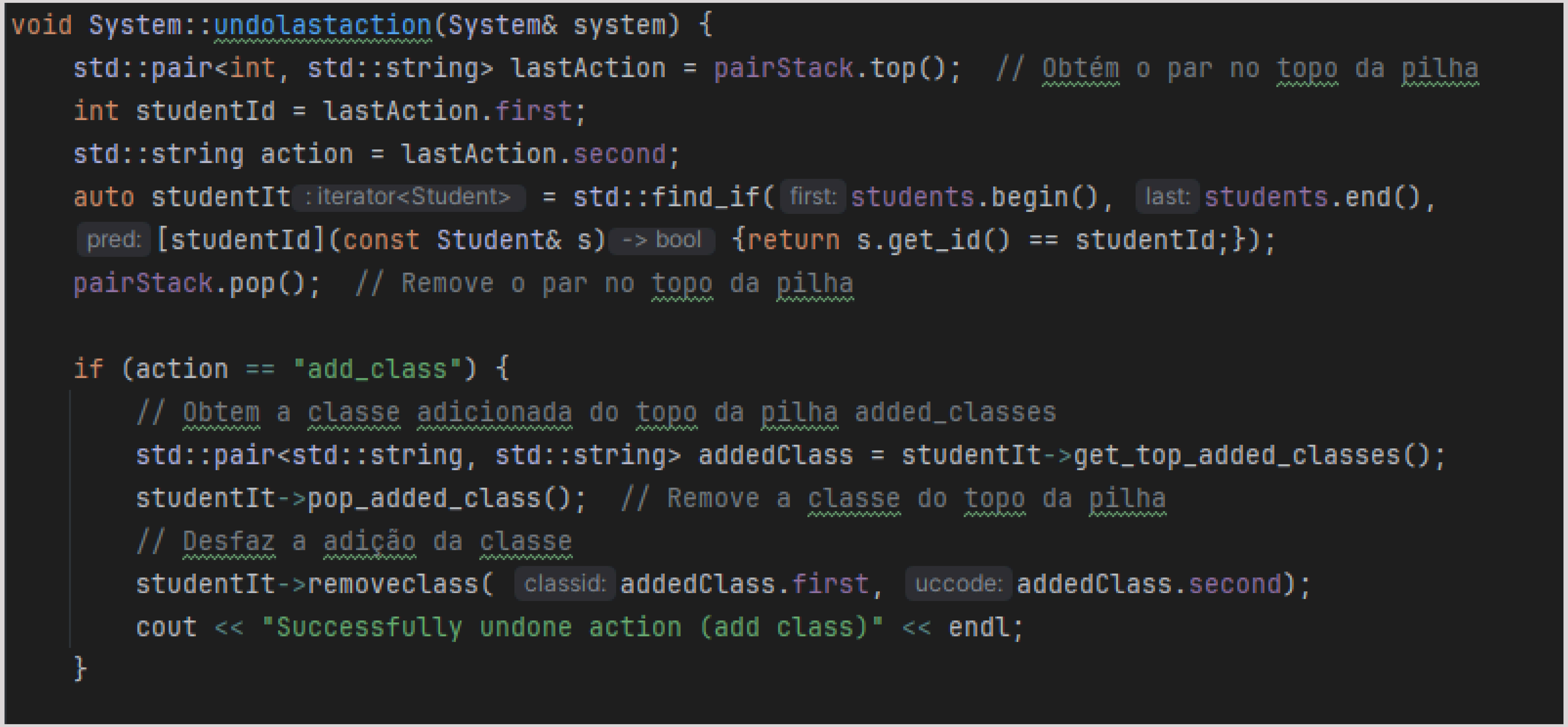


```
bool Change::verifyandchangetheclass(Student& student) {
    for (auto& pair :pair<...> & : priorityQueue) { // Usar auto& para modificar o par
        if (pair.second == student.get_uccode()) { // Verificar se são da mesma uc
            if (!pair.first.empty() && pair.first.front().get_to_class() == student.get_from_class()) {
                student.removeclass( classid: student.get_from_class(),  uccode: student.get_uccode());
                student.addclass( Class: pair.first.front().get_to_class(),  uccode: pair.first.front().get_uccode());
                pair.first.front().removeclass( classid: pair.first.front().get_from_class(),  uccode: pair.first.front().get_uccode());
                pair.first.front().addclass( Class: student.get_from_class(),  uccode: student.get_uccode());
                pair.first.pop();
                return true; // Fez a troca
            }
        }
    }
    return false; // Não encontrou nenhuma troca possível
}
```

Undo

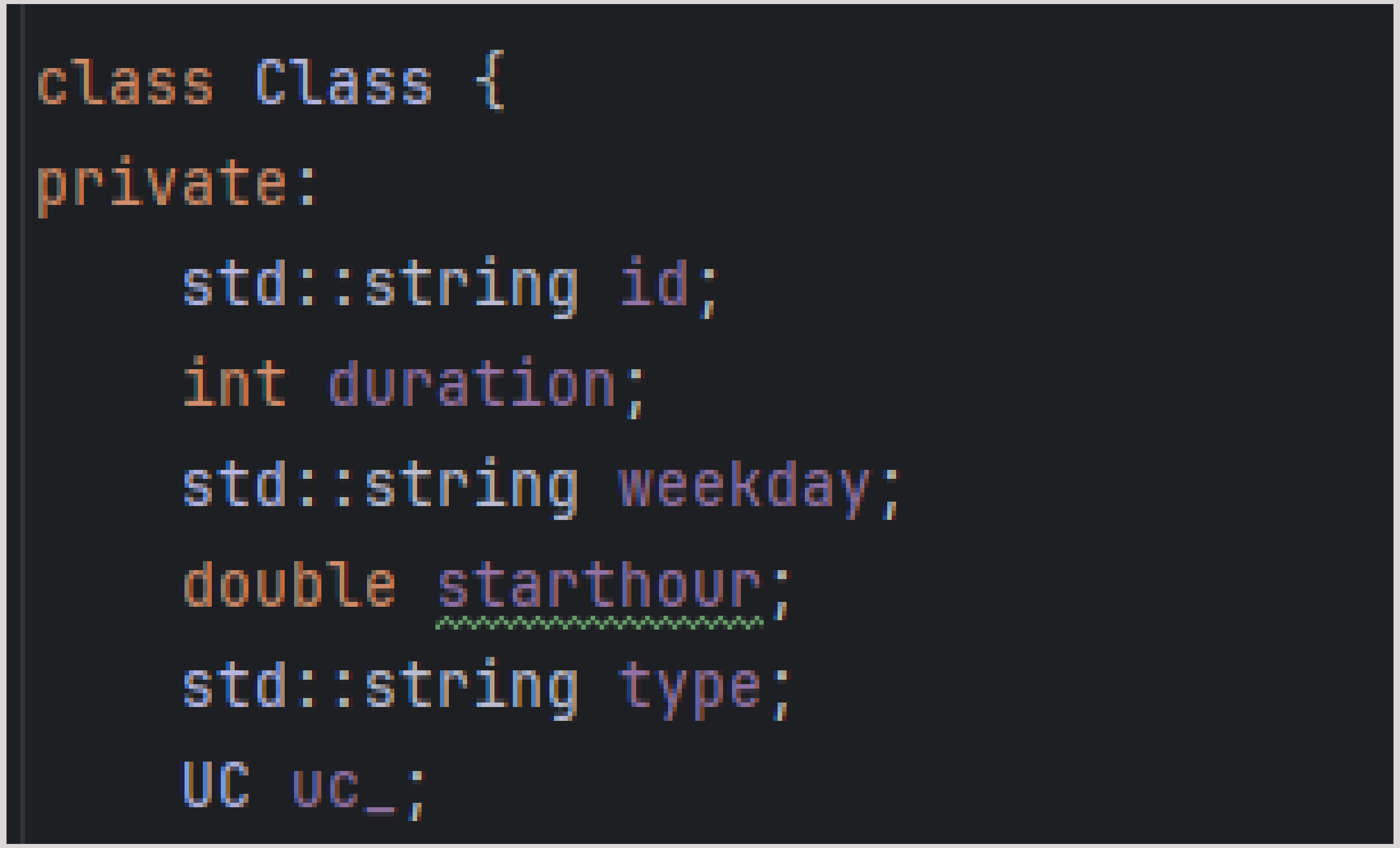
Estrutura usada: stack

O(1)



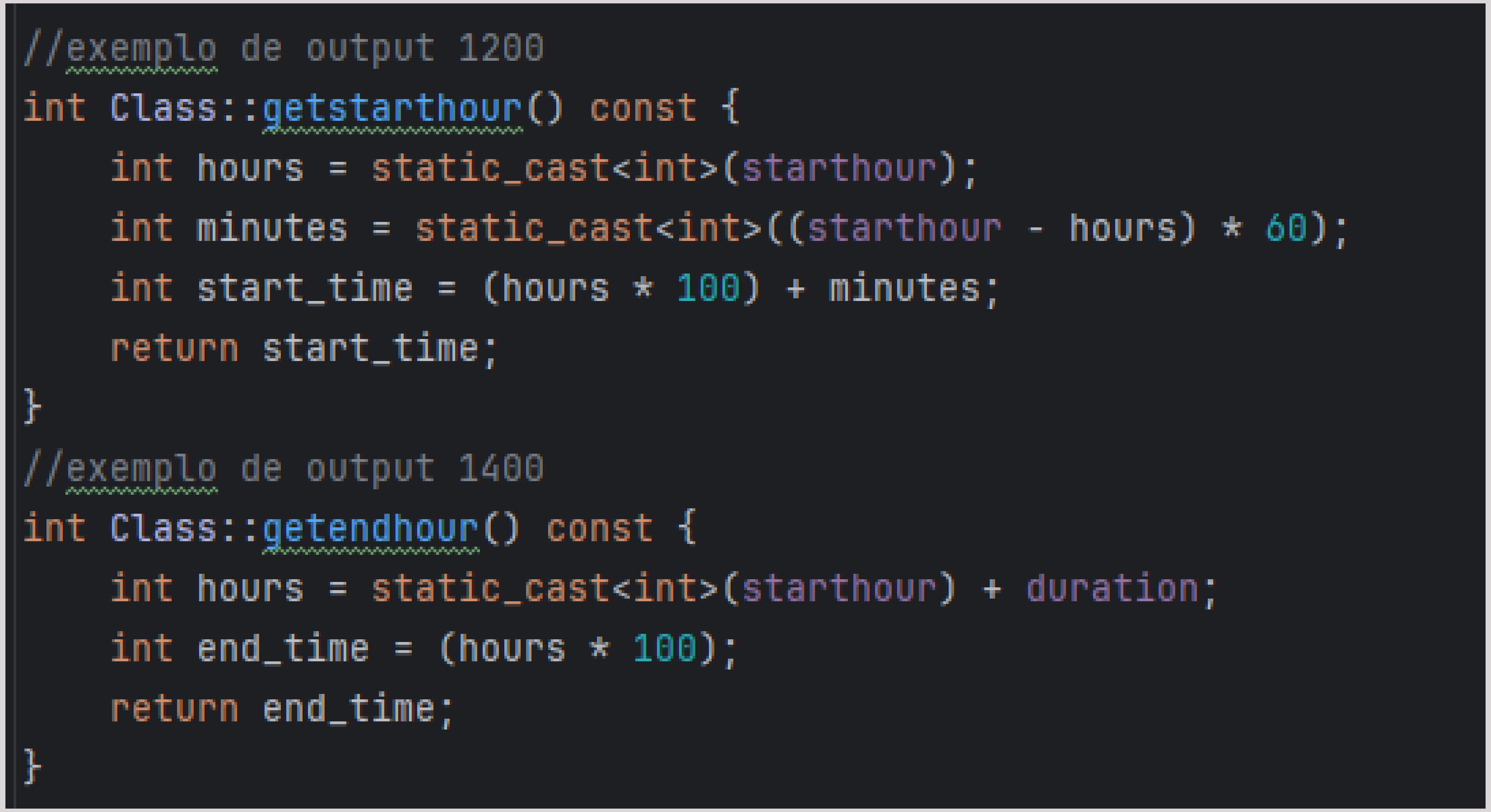
```
void System::undolastaction(System& system) {
    std::pair<int, std::string> lastAction = pairStack.top();  // Obtém o par no topo da pilha
    int studentId = lastAction.first;
    std::string action = lastAction.second;
    auto studentIt : iterator<Student> = std::find_if( first: students.begin(),  last: students.end(),
     pred: [studentId](const Student& s) -> bool {return s.get_id() == studentId;});
    pairStack.pop();  // Remove o par no topo da pilha

    if (action == "add_class") {
        // Obtem a classe adicionada do topo da pilha added_classes
        std::pair<std::string, std::string> addedClass = studentIt->get_top_added_classes();
        studentIt->pop_added_class();  // Remove a classe do topo da pilha
        // Desfaz a adição da classe
        studentIt->removeclass( classid: addedClass.first,  uccode: addedClass.second);
        cout << "Successfully undone action (add class)" << endl;
    }
```

Class



```
class Class {
private:
    std::string id;
    int duration;
    std::string weekday;
    double starthour;
    std::string type;
    UC uc_;
```

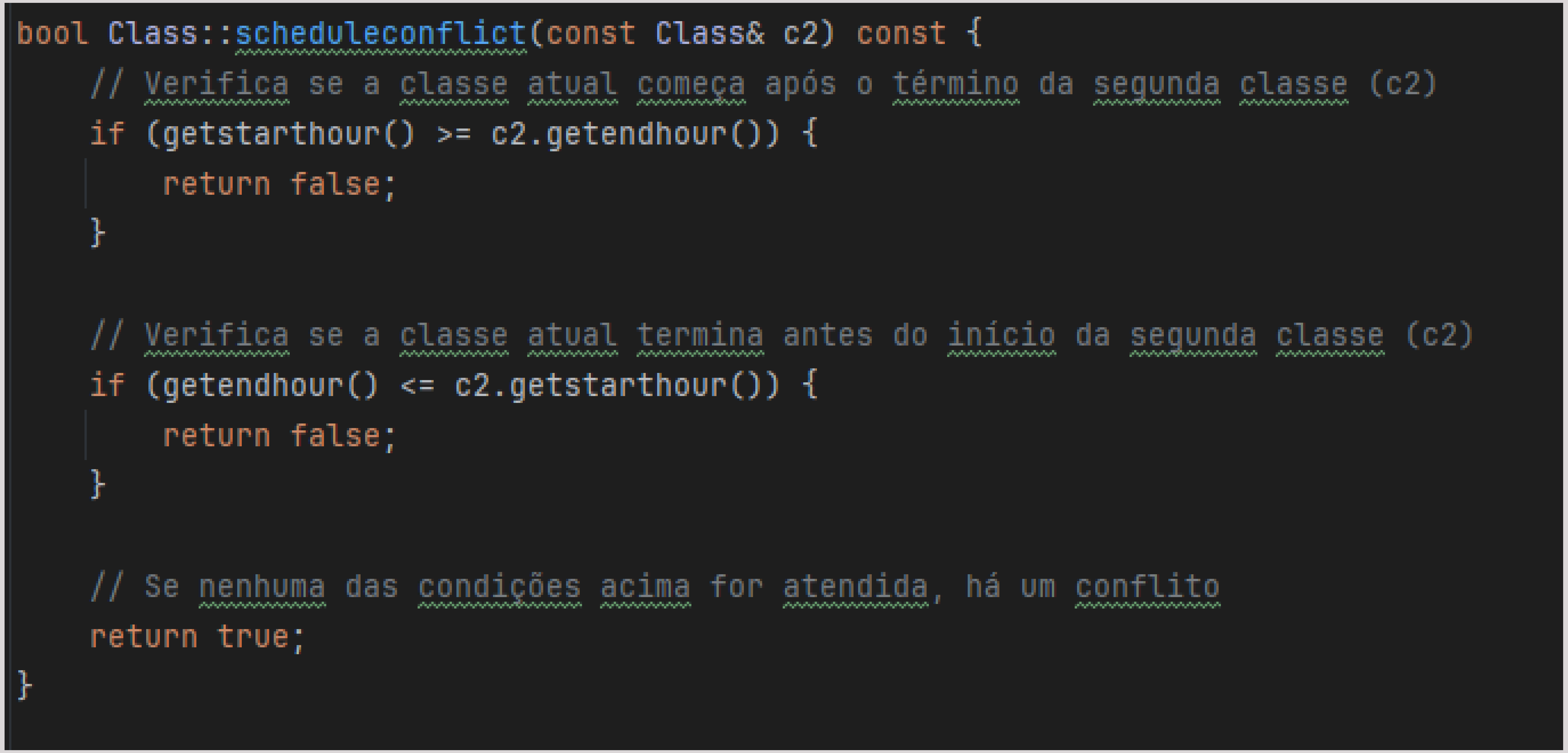
Getstarthour e Getendhour



```
//exemplo de output 1200
int Class::getstarthour() const {
    int hours = static_cast<int>(starthour);
    int minutes = static_cast<int>((starthour - hours) * 60);
    int start_time = (hours * 100) + minutes;
    return start_time;
}
//exemplo de output 1400
int Class::getendhour() const {
    int hours = static_cast<int>(starthour) + duration;
    int end_time = (hours * 100);
    return end_time;
}
```

ScheduleConflit

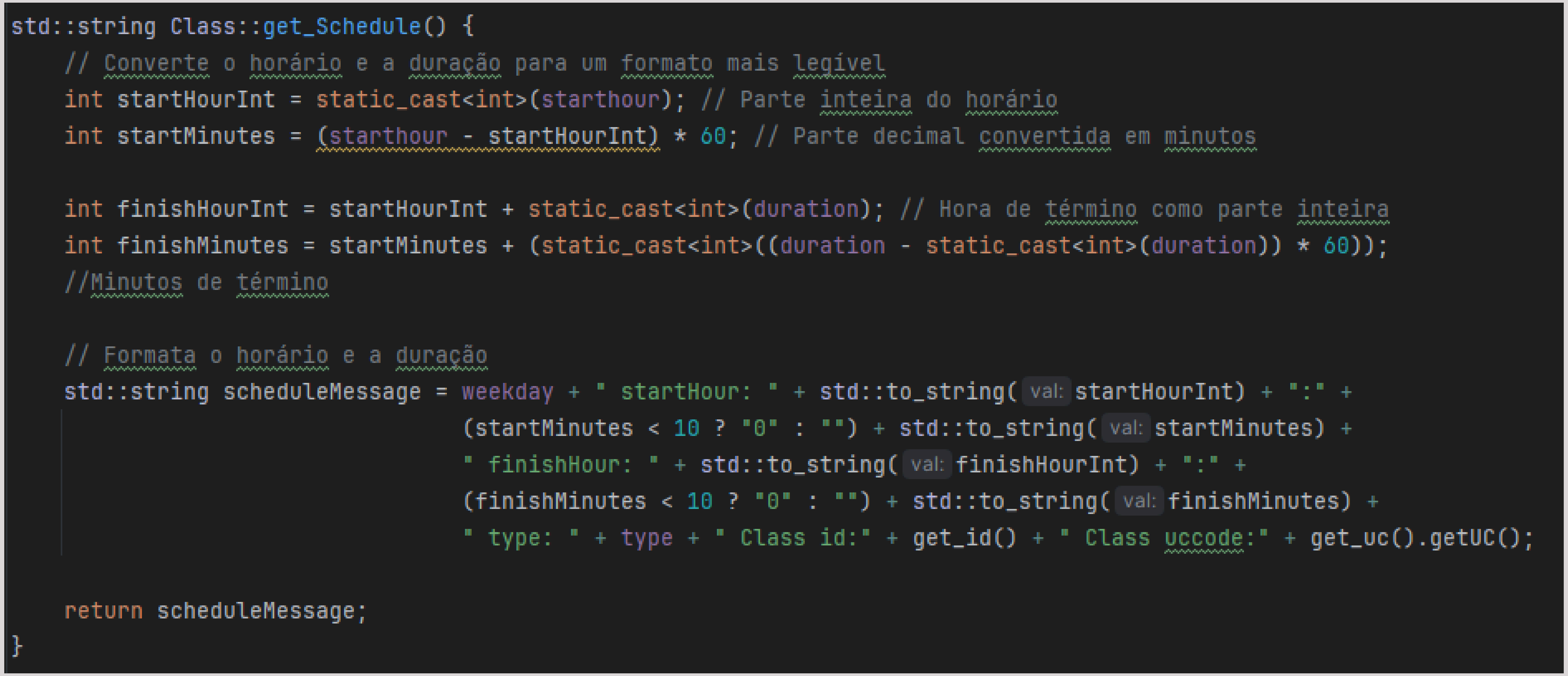
O(1)



```cpp
bool Class::scheduleconflict(const Class& c2) const {
    // Verifica se a classe atual começa após o término da segunda classe (c2)
    if (getstarthour() >= c2.getendhour()) {
        return false;
    }

    // Verifica se a classe atual termina antes do início da segunda classe (c2)
    if (getendhour() <= c2.getstarthour()) {
        return false;
    }

    // Se nenhuma das condições acima for atendida, há um conflito
    return true;
}
```

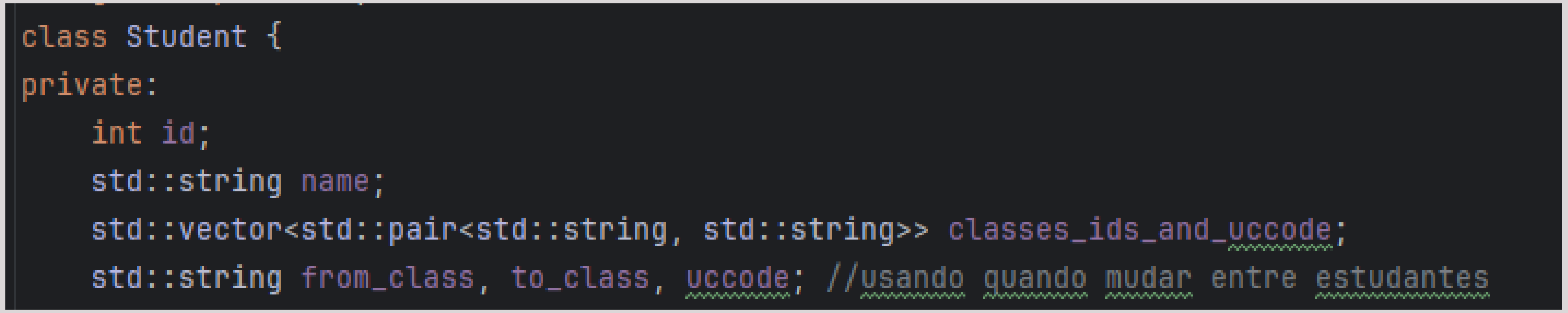
GetSchedule

O(1)



```
std::string Class::get_Schedule() {
    // Converte o horário e a duração para um formato mais legível
    int startHourInt = static_cast<int>(starthour); // Parte inteira do horário
    int startMinutes = (starthour - startHourInt) * 60; // Parte decimal convertida em minutos

    int finishHourInt = startHourInt + static_cast<int>(duration); // Hora de término como parte inteira
    int finishMinutes = startMinutes + (static_cast<int>((duration - static_cast<int>(duration)) * 60));
    //Minutos de término

    // Formata o horário e a duração
    std::string scheduleMessage = weekday + " startHour: " + std::to_string( val: startHourInt) + ":" +
                                  (startMinutes < 10 ? "0" : "") + std::to_string( val: startMinutes) +
                                  " finishHour: " + std::to_string( val: finishHourInt) + ":" +
                                  (finishMinutes < 10 ? "0" : "") + std::to_string( val: finishMinutes) +
                                  " type: " + type + " Class id:" + get_id() + " Class uccode:" + get_uc().getUC();

    return scheduleMessage;
}
```

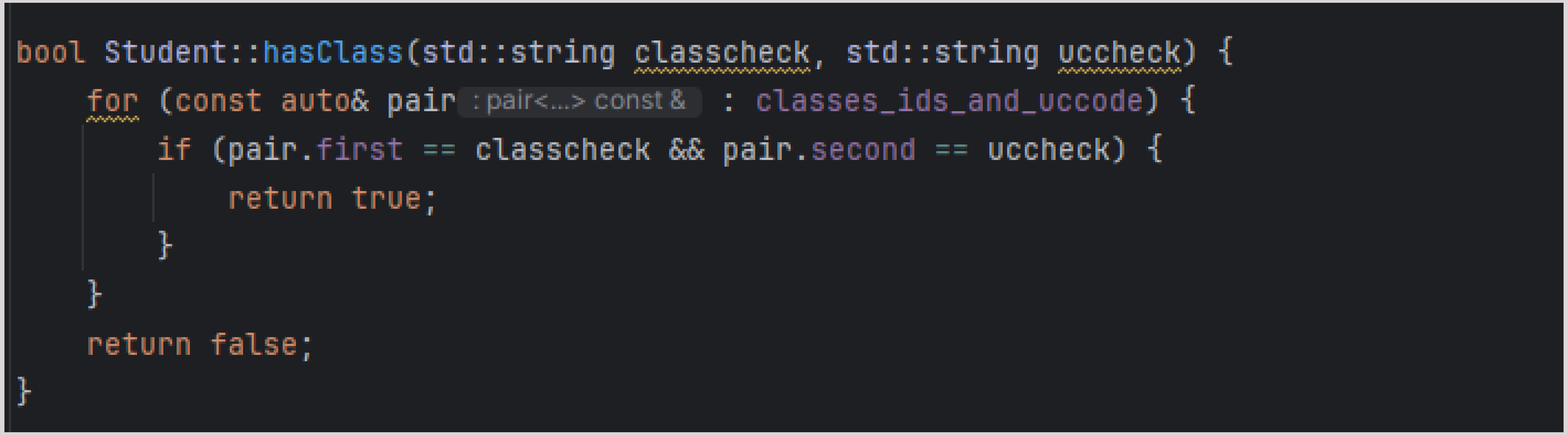
Student



```
class Student {
private:
    int id;
    std::string name;
    std::vector<std::pair<std::string, std::string>> classes_ids_and_uccode;
    std::string from_class, to_class, uccode; //usando quando mudar entre estudantes
```

HasClass

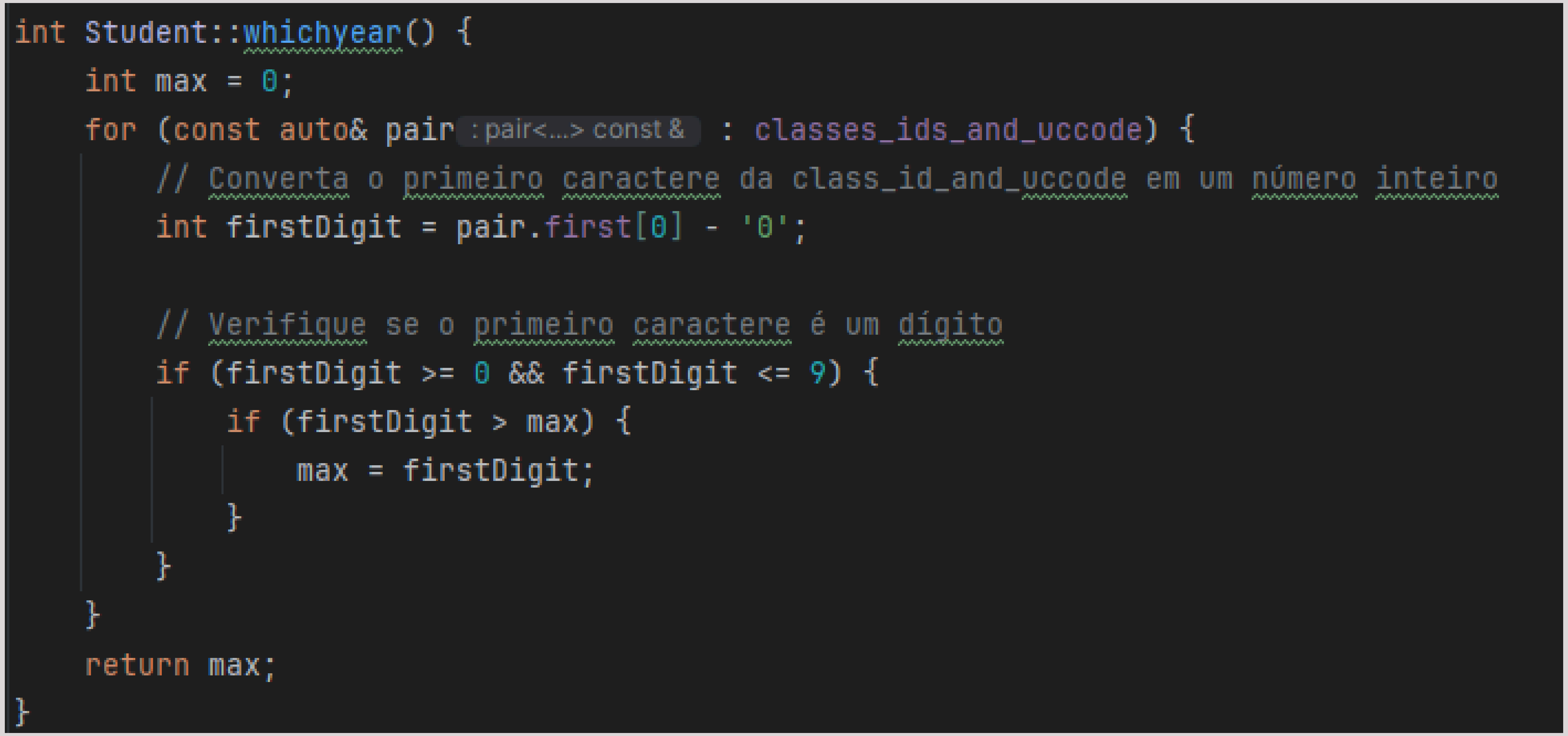
O(n)



```
bool Student::hasClass(std::string classcheck, std::string uccheck) {
    for (const auto& pair : pair<...> const &  : classes_ids_and_uccode) {
        if (pair.first == classcheck && pair.second == uccheck) {
            return true;
        }
    }
    return false;
}
```

WhichYear

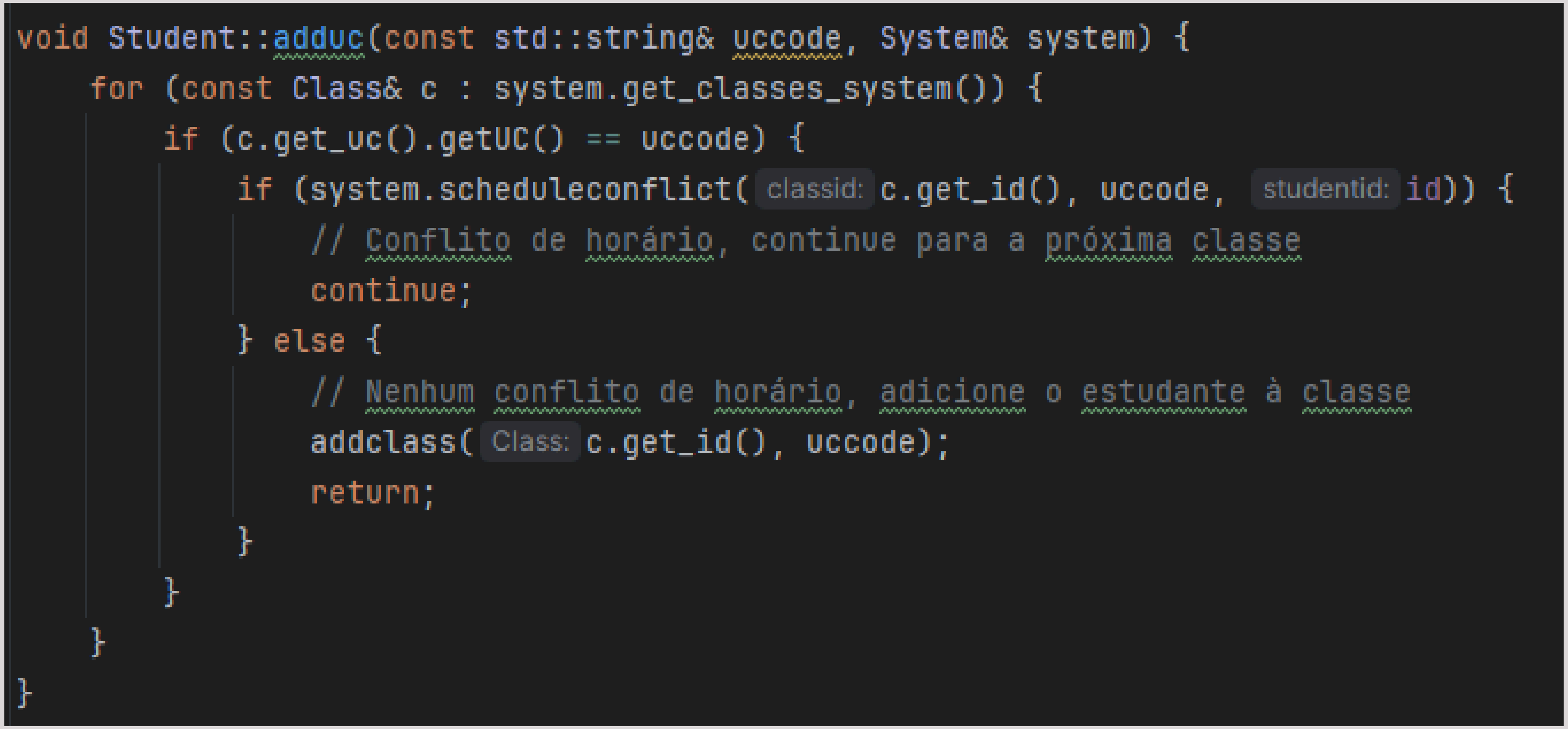
O(n)



```
int Student::whichyear() {
    int max = 0;
    for (const auto& pair : pair<...> const & : classes_ids_and_uccode) {
        // Converta o primeiro caractere da class_id_and_uccode em um número inteiro
        int firstDigit = pair.first[0] - '0';

        // Verifique se o primeiro caractere é um dígito
        if (firstDigit >= 0 && firstDigit <= 9) {
            if (firstDigit > max) {
                max = firstDigit;
            }
        }
    }
    return max;
}
```

AddUC

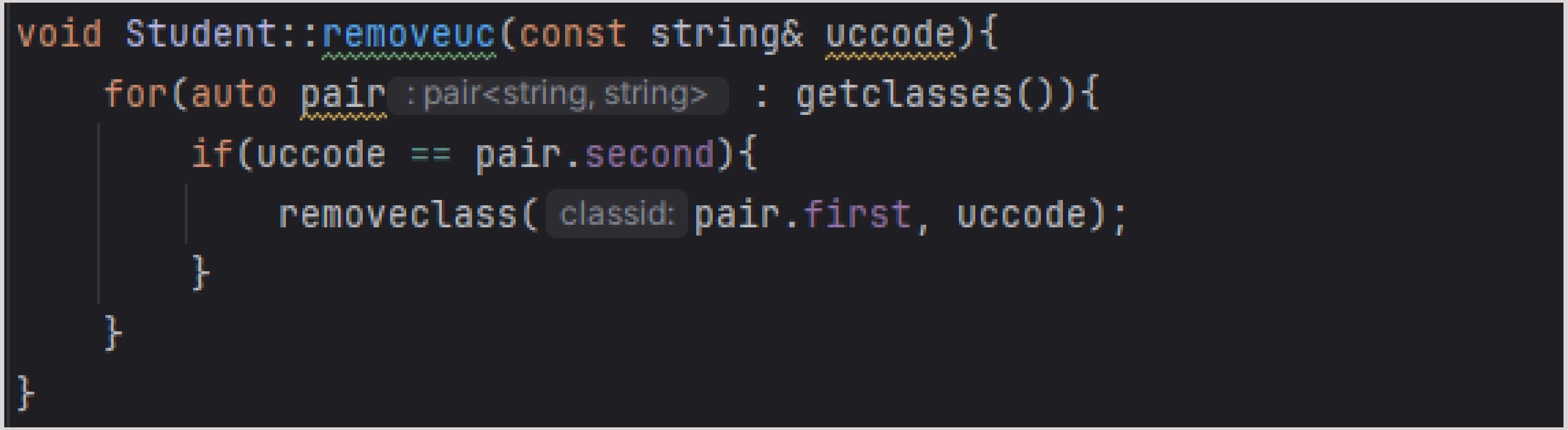
O(n)



```
void Student::adduc(const std::string& uccode, System& system) {
    for (const Class& c : system.get_classes_system()) {
        if (c.get_uc().getUC() == uccode) {
            if (system.scheduleconflict( classid: c.get_id(), uccode,  studentid: id)) {
                // Conflito de horário, continue para a próxima classe
                continue;
            } else {
                // Nenhum conflito de horário, adicione o estudante à classe
                addclass( Class: c.get_id(), uccode);
                return;
            }
        }
    }
}
```

RemoveUC

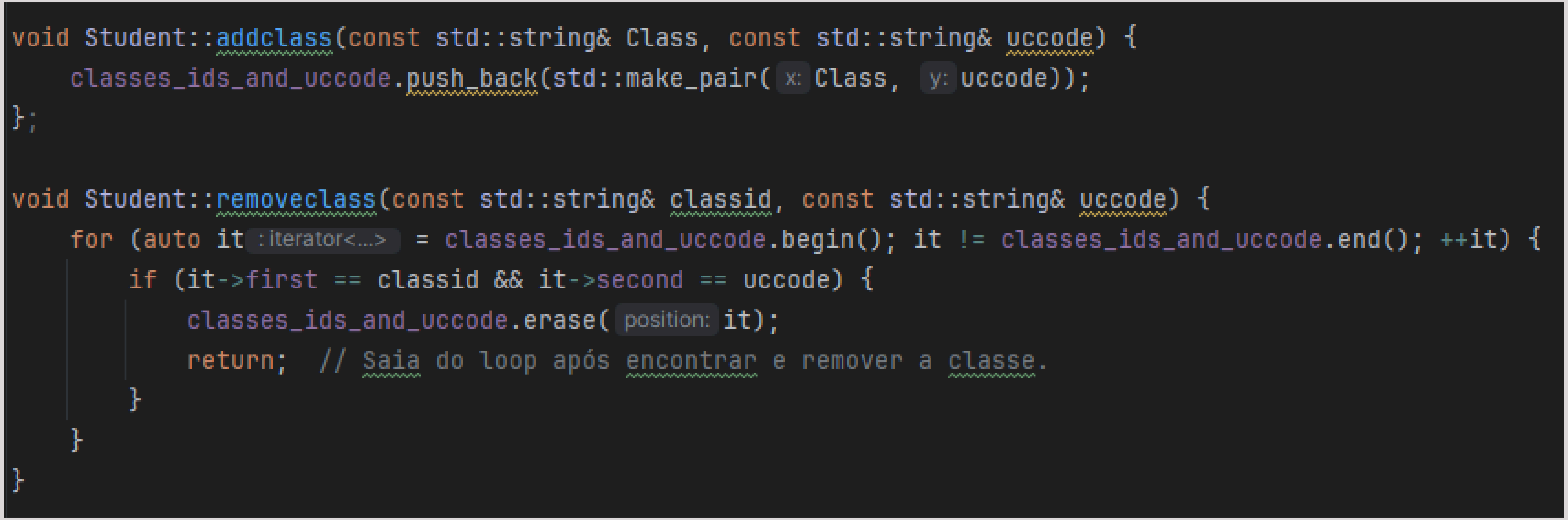
O(n)



```
void Student::removeuc(const string& uccode){
    for(auto pair : pair<string, string>  : getclasses()){
        if(uccode == pair.second){
            removeclass( classid: pair.first, uccode);
        }
    }
}
```

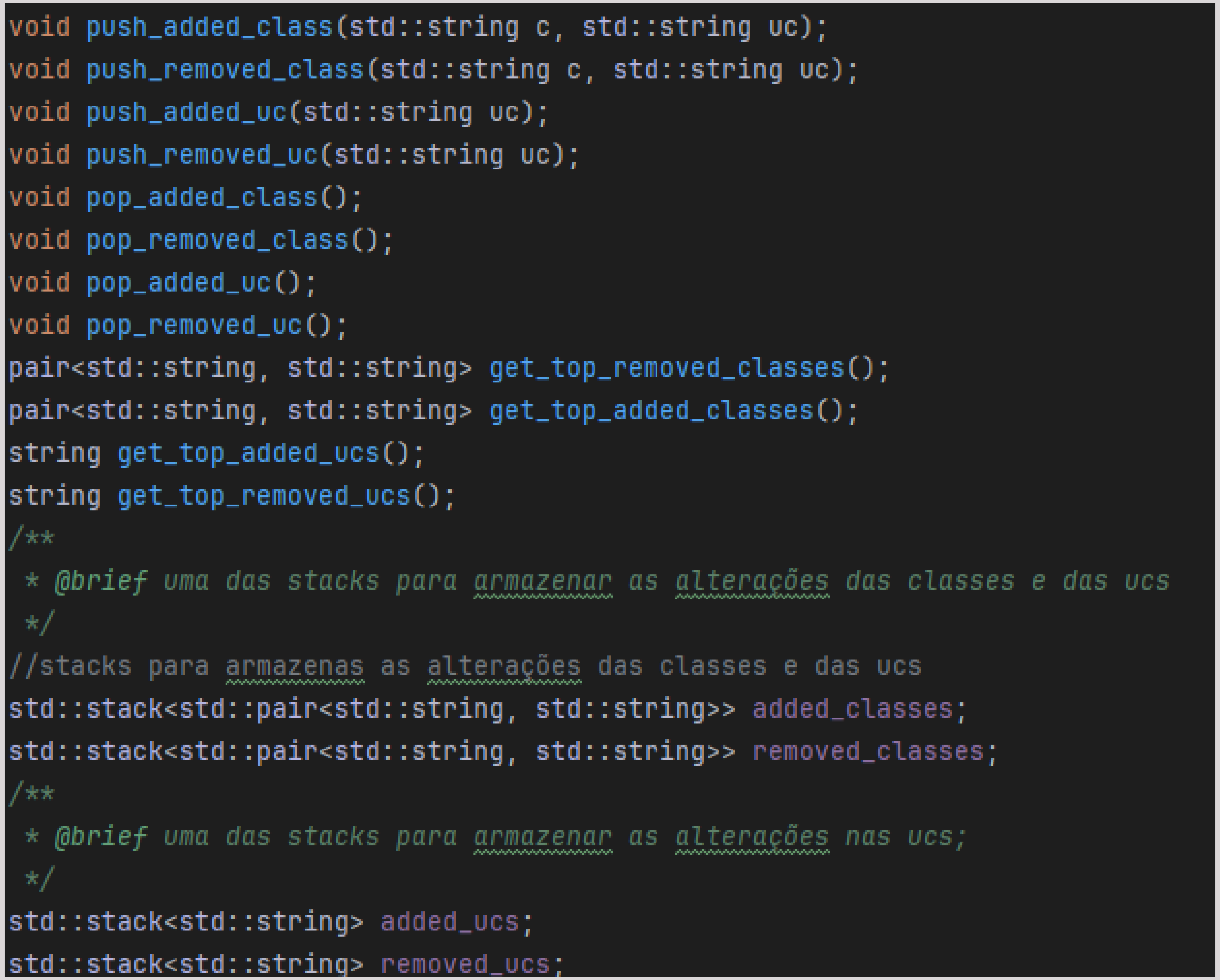
Add and Remove Class

O(1),O(n)



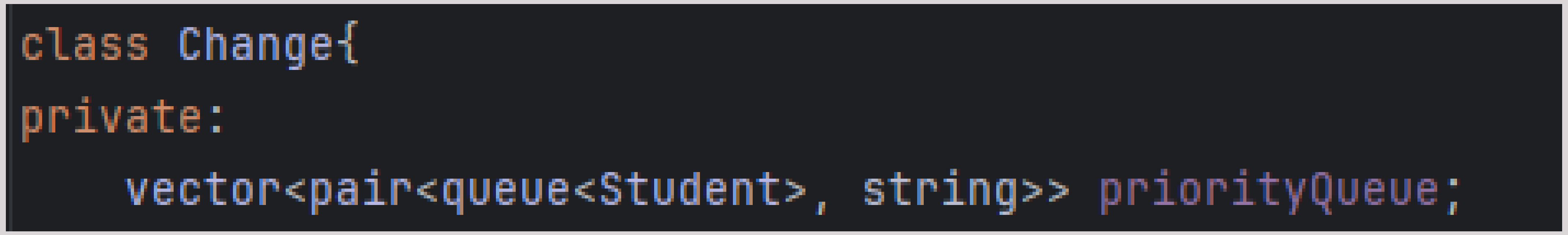
```
void Student::addclass(const std::string& Class, const std::string& uccode) {
    classes_ids_and_uccode.push_back(std::make_pair( x: Class,  y: uccode));
};

void Student::removeclass(const std::string& classid, const std::string& uccode) {
    for (auto it : iterator<...>  = classes_ids_and_uccode.begin(); it != classes_ids_and_uccode.end(); ++it) {
        if (it->first == classid && it->second == uccode) {
            classes_ids_and_uccode.erase( position: it);
            return;  // Saia do loop após encontrar e remover a classe.
        }
    }
}
```

Stack



```
void push_added_class(std::string c, std::string uc);
void push_removed_class(std::string c, std::string uc);
void push_added_uc(std::string uc);
void push_removed_uc(std::string uc);
void pop_added_class();
void pop_removed_class();
void pop_added_uc();
void pop_removed_uc();
pair<std::string, std::string> get_top_removed_classes();
pair<std::string, std::string> get_top_added_classes();
string get_top_added_ucs();
string get_top_removed_ucs();
/**
 * @brief uma das stacks para armazenar as alterações das classes e das ucs
 */
//stacks para armazenas as alterações das classes e das ucs
std::stack<std::pair<std::string, std::string>> added_classes;
std::stack<std::pair<std::string, std::string>> removed_classes;
/**
 * @brief uma das stacks para armazenar as alterações nas ucs;
 */
std::stack<std::string> added_ucs;
std::stack<std::string> removed_ucs;
```

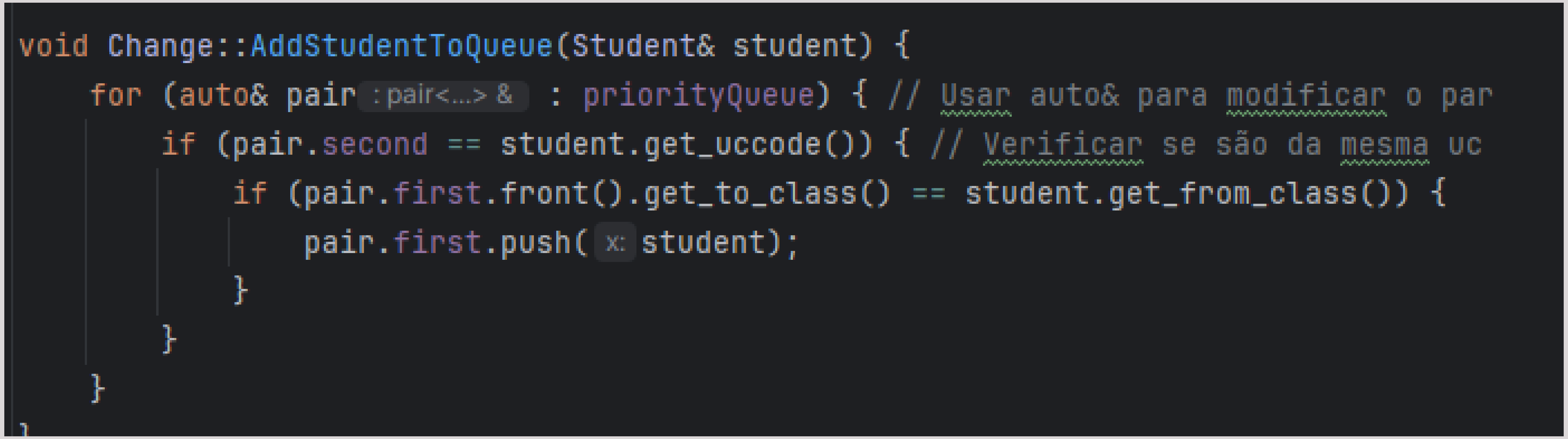
Change



```
class Change{
private:
    vector<pair<queue<Student>, string>> priorityQueue;
```

Addstudentoqueue

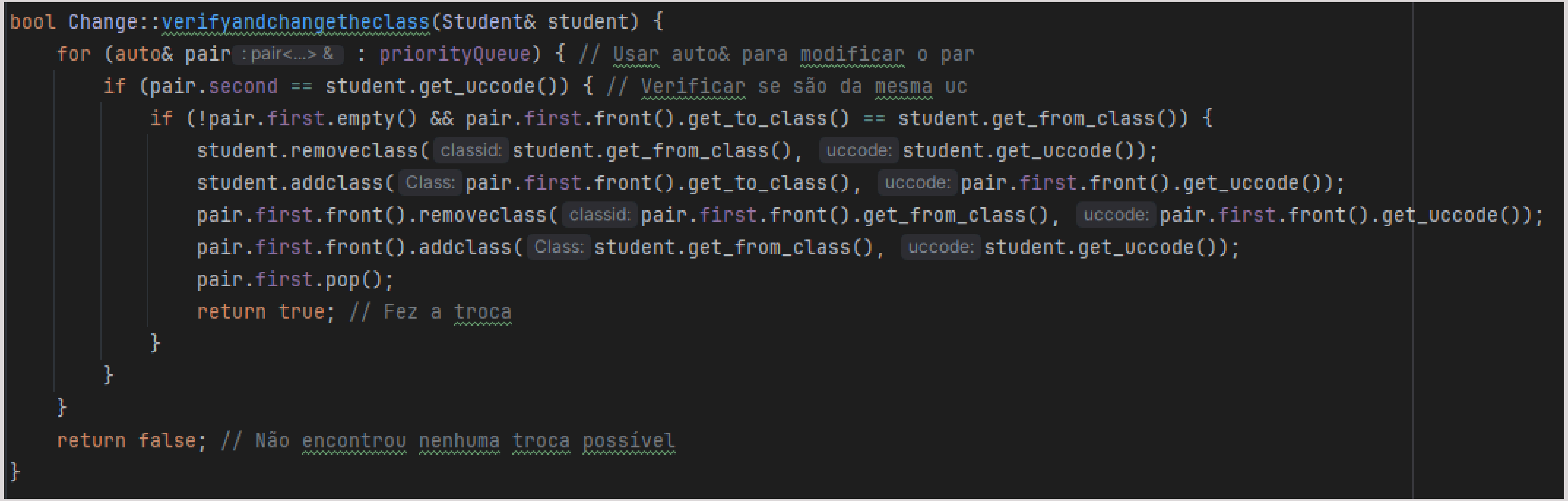
O(n)



```
void Change::AddStudentToQueue(Student& student) {
    for (auto& pair : pair<...> &  : priorityQueue) { // Usar auto& para modificar o par
        if (pair.second == student.get_uccode()) { // Verificar se são da mesma uc
            if (pair.first.front().get_to_class() == student.get_from_class()) {
                pair.first.push( x: student);
            }
        }
    }
```

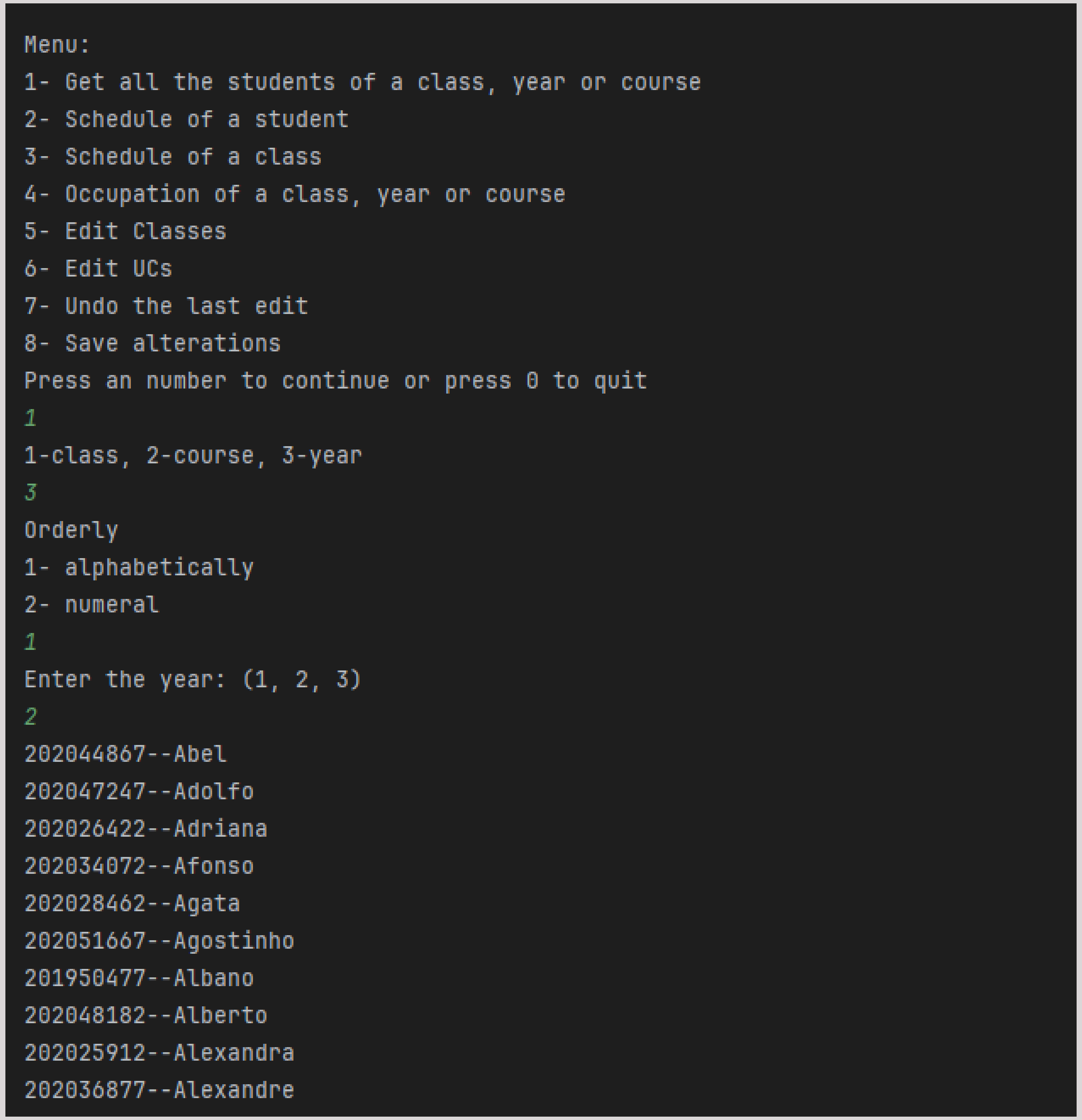
Verifyandchangeclass

O(n)

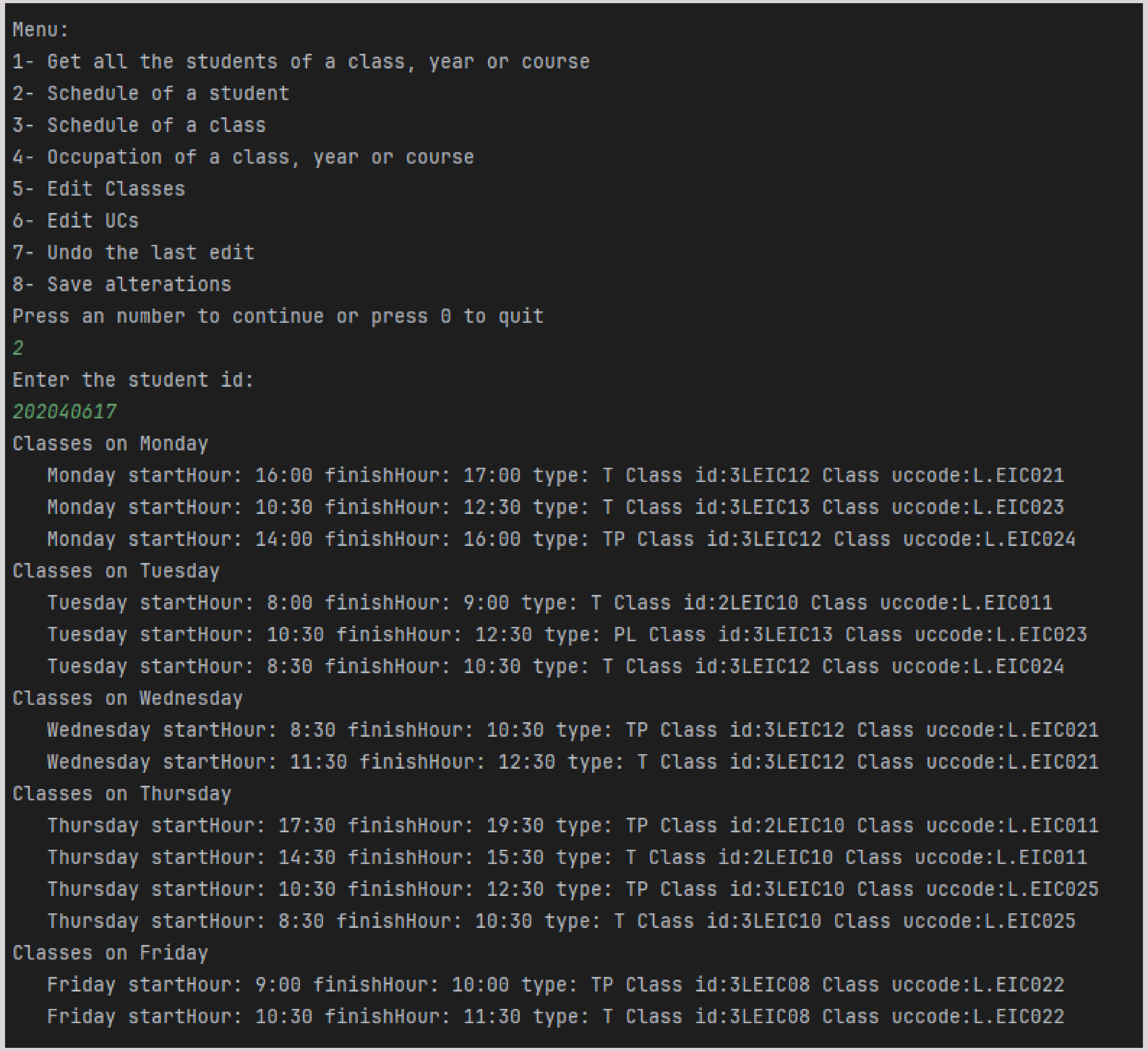


```cpp
bool Change::verifyandchangetheclass(Student& student) {
    for (auto& pair : pair<...> &  : priorityQueue) { // Usar auto& para modificar o par
        if (pair.second == student.get_uccode()) { // Verificar se são da mesma uc
            if (!pair.first.empty() && pair.first.front().get_to_class() == student.get_from_class()) {
                student.removeclass( classid: student.get_from_class(),  uccode: student.get_uccode());
                student.addclass( Class: pair.first.front().get_to_class(),  uccode: pair.first.front().get_uccode());
                pair.first.front().removeclass( classid: pair.first.front().get_from_class(),  uccode: pair.first.front().get_uccode());
                pair.first.front().addclass( Class: student.get_from_class(),  uccode: student.get_uccode());
                pair.first.pop();
                return true; // Fez a troca
            }
        }
    }
    return false; // Não encontrou nenhuma troca possível
}
```

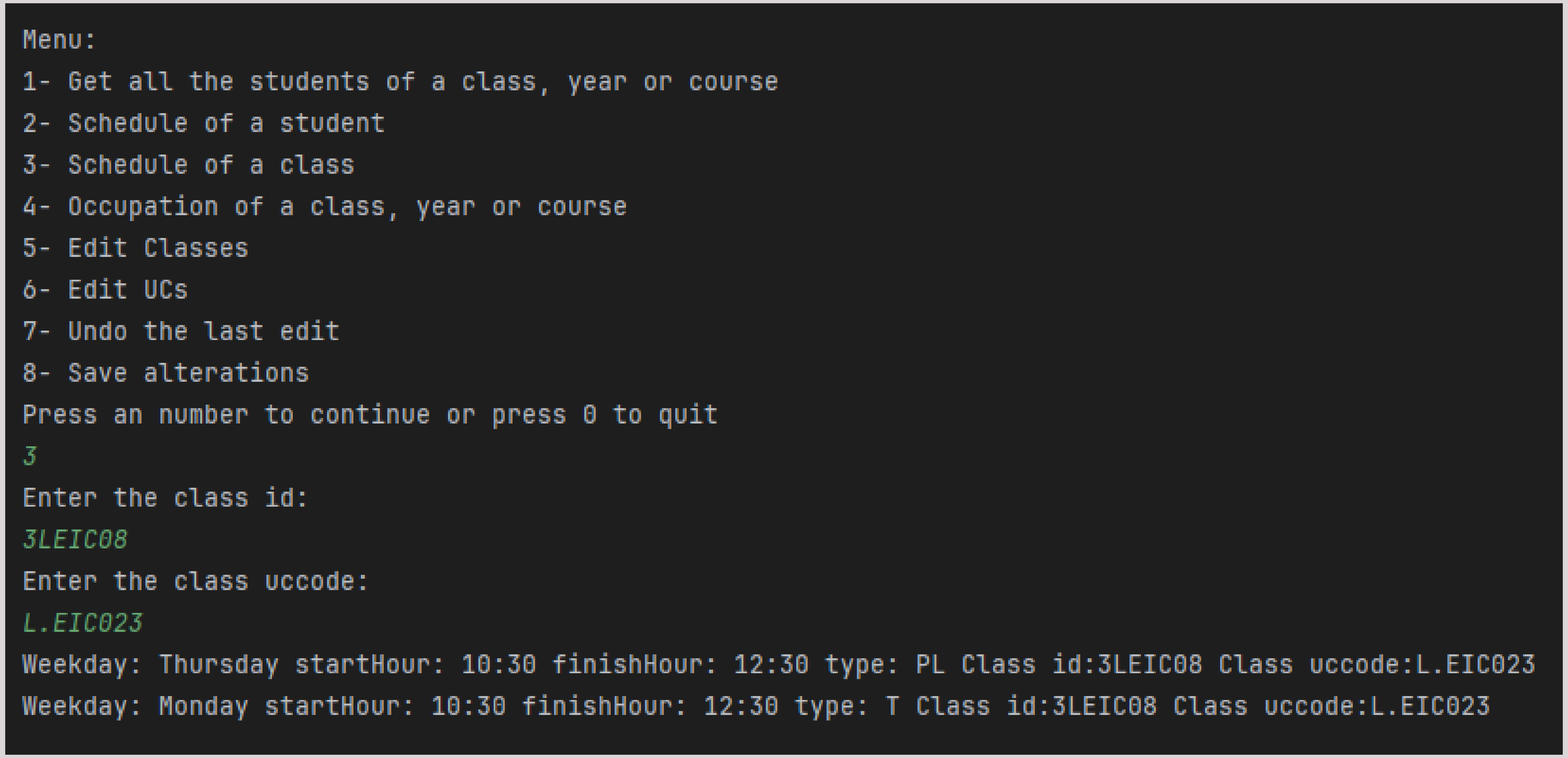
Exemplos de Execução



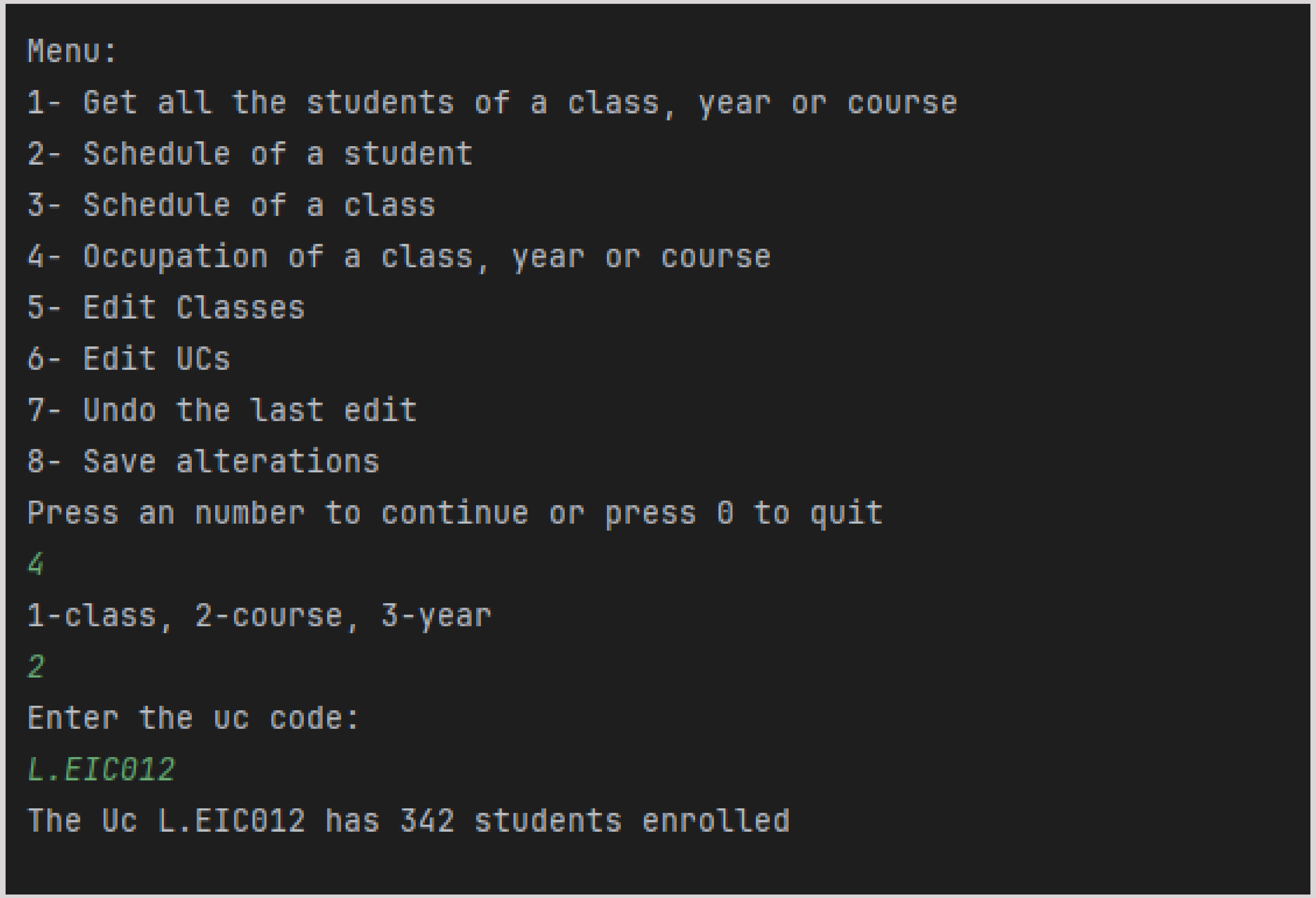
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
1
1-class, 2-course, 3-year
3
Orderly
1- alphabetically
2- numeral
1
Enter the year: (1, 2, 3)
2
202044867--Abel
202047247--Adolfo
202026422--Adriana
202034072--Afonso
202028462--Agata
202051667--Agostinho
201950477--Albano
202048182--Alberto
202025912--Alexandra
202036877--Alexandre
```



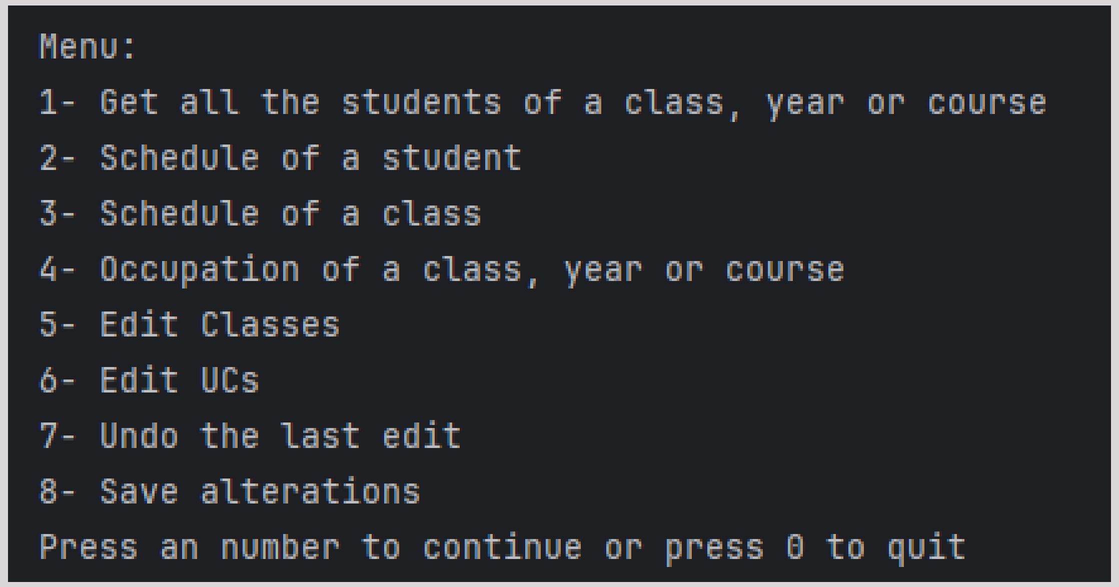
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
2
Enter the student id:
202040617
Classes on Monday
   Monday startHour: 16:00 finishHour: 17:00 type: T Class id:3LEIC12 Class uccode:L.EIC021
   Monday startHour: 10:30 finishHour: 12:30 type: T Class id:3LEIC13 Class uccode:L.EIC023
   Monday startHour: 14:00 finishHour: 16:00 type: TP Class id:3LEIC12 Class uccode:L.EIC024
Classes on Tuesday
   Tuesday startHour: 8:00 finishHour: 9:00 type: T Class id:2LEIC10 Class uccode:L.EIC011
   Tuesday startHour: 10:30 finishHour: 12:30 type: PL Class id:3LEIC13 Class uccode:L.EIC023
   Tuesday startHour: 8:30 finishHour: 10:30 type: T Class id:3LEIC12 Class uccode:L.EIC024
Classes on Wednesday
   Wednesday startHour: 8:30 finishHour: 10:30 type: TP Class id:3LEIC12 Class uccode:L.EIC021
   Wednesday startHour: 11:30 finishHour: 12:30 type: T Class id:3LEIC12 Class uccode:L.EIC021
Classes on Thursday
   Thursday startHour: 17:30 finishHour: 19:30 type: TP Class id:2LEIC10 Class uccode:L.EIC011
   Thursday startHour: 14:30 finishHour: 15:30 type: T Class id:2LEIC10 Class uccode:L.EIC011
   Thursday startHour: 10:30 finishHour: 12:30 type: TP Class id:3LEIC10 Class uccode:L.EIC025
   Thursday startHour: 8:30 finishHour: 10:30 type: T Class id:3LEIC10 Class uccode:L.EIC025
Classes on Friday
   Friday startHour: 9:00 finishHour: 10:00 type: TP Class id:3LEIC08 Class uccode:L.EIC022
   Friday startHour: 10:30 finishHour: 11:30 type: T Class id:3LEIC08 Class uccode:L.EIC022
```



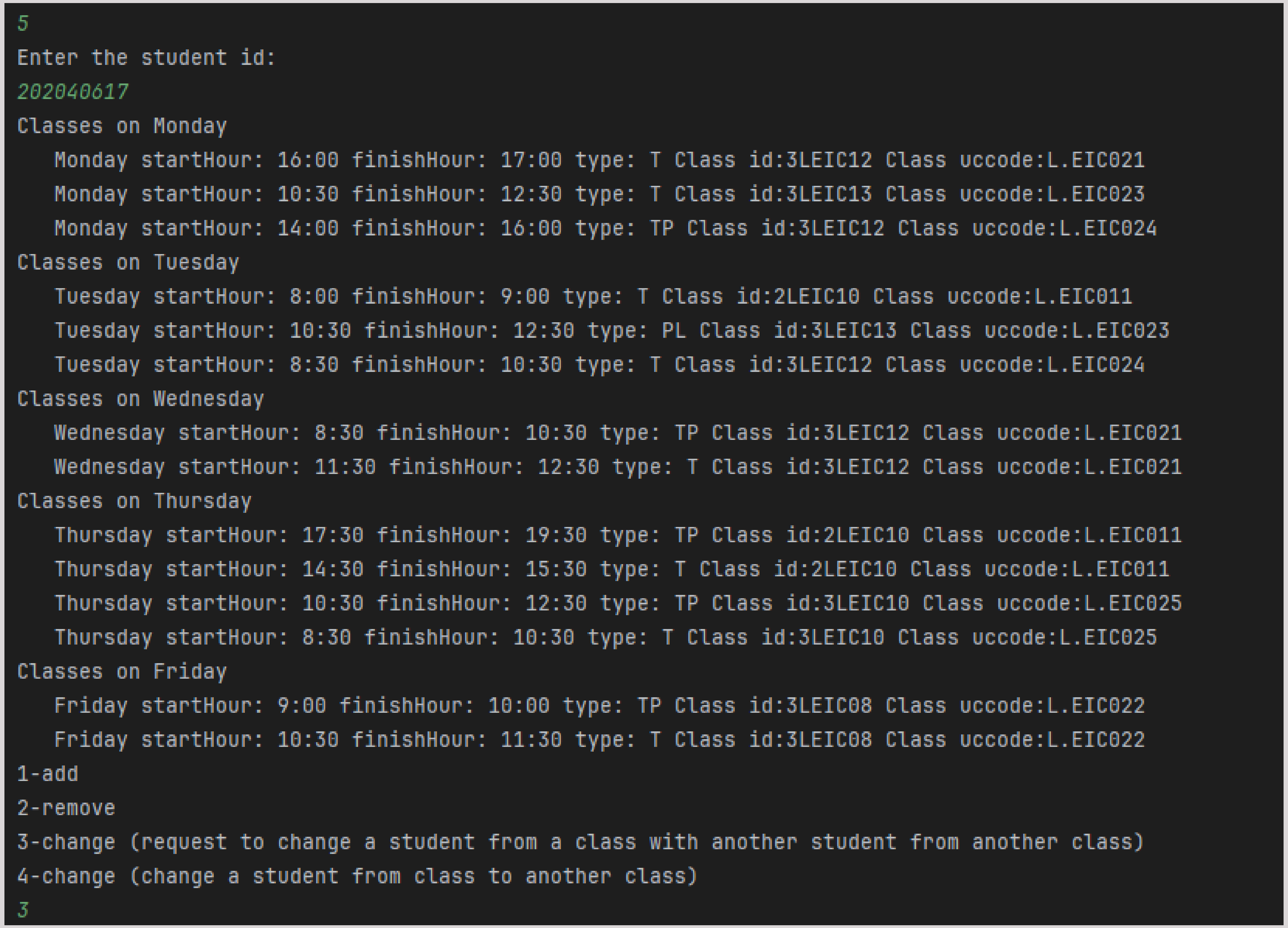
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
3
Enter the class id:
3LEIC08
Enter the class uccode:
L.EIC023
Weekday: Thursday startHour: 10:30 finishHour: 12:30 type: PL Class id:3LEIC08 Class uccode:L.EIC023
Weekday: Monday startHour: 10:30 finishHour: 12:30 type: T Class id:3LEIC08 Class uccode:L.EIC023
```



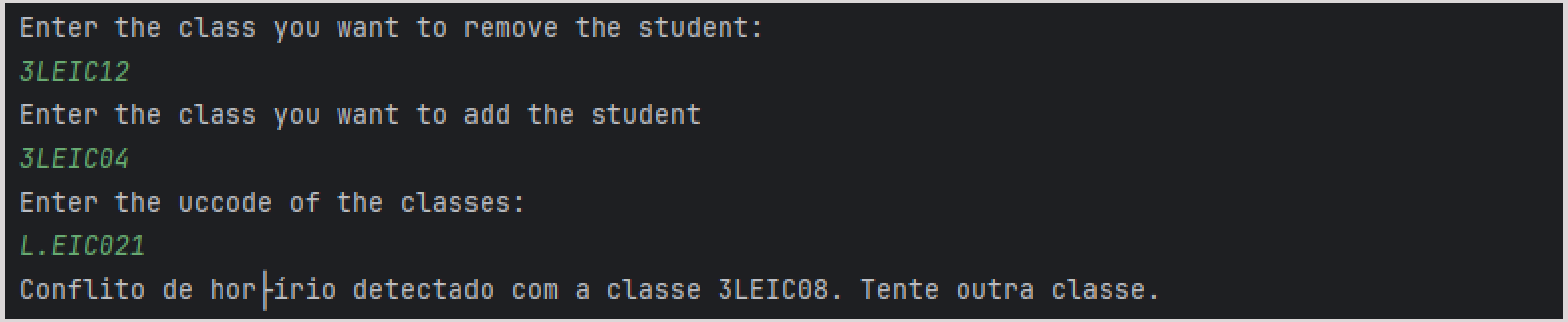
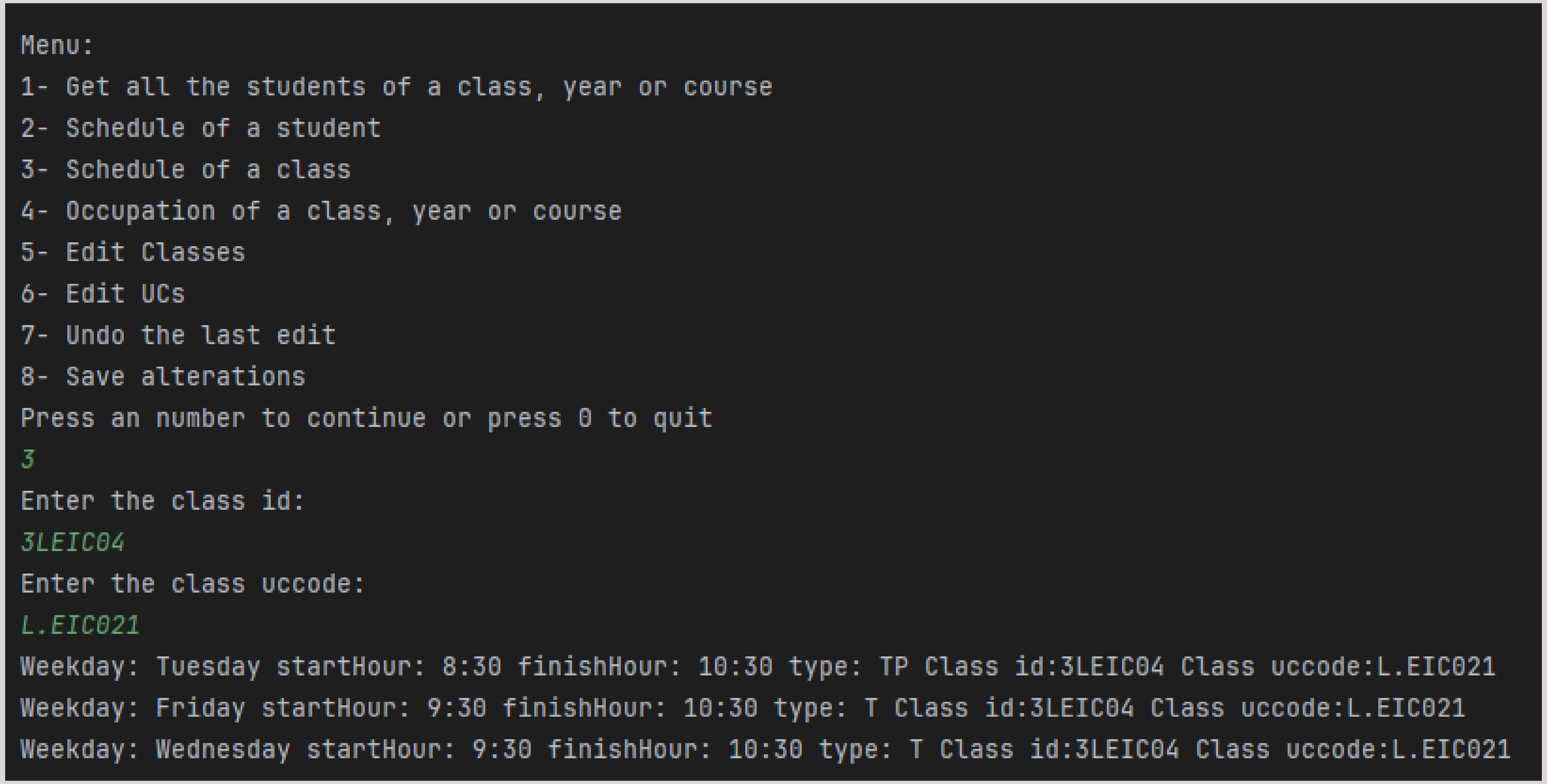
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
4
1-class, 2-course, 3-year
2
Enter the uc code:
L.EIC012
The Uc L.EIC012 has 342 students enrolled
```

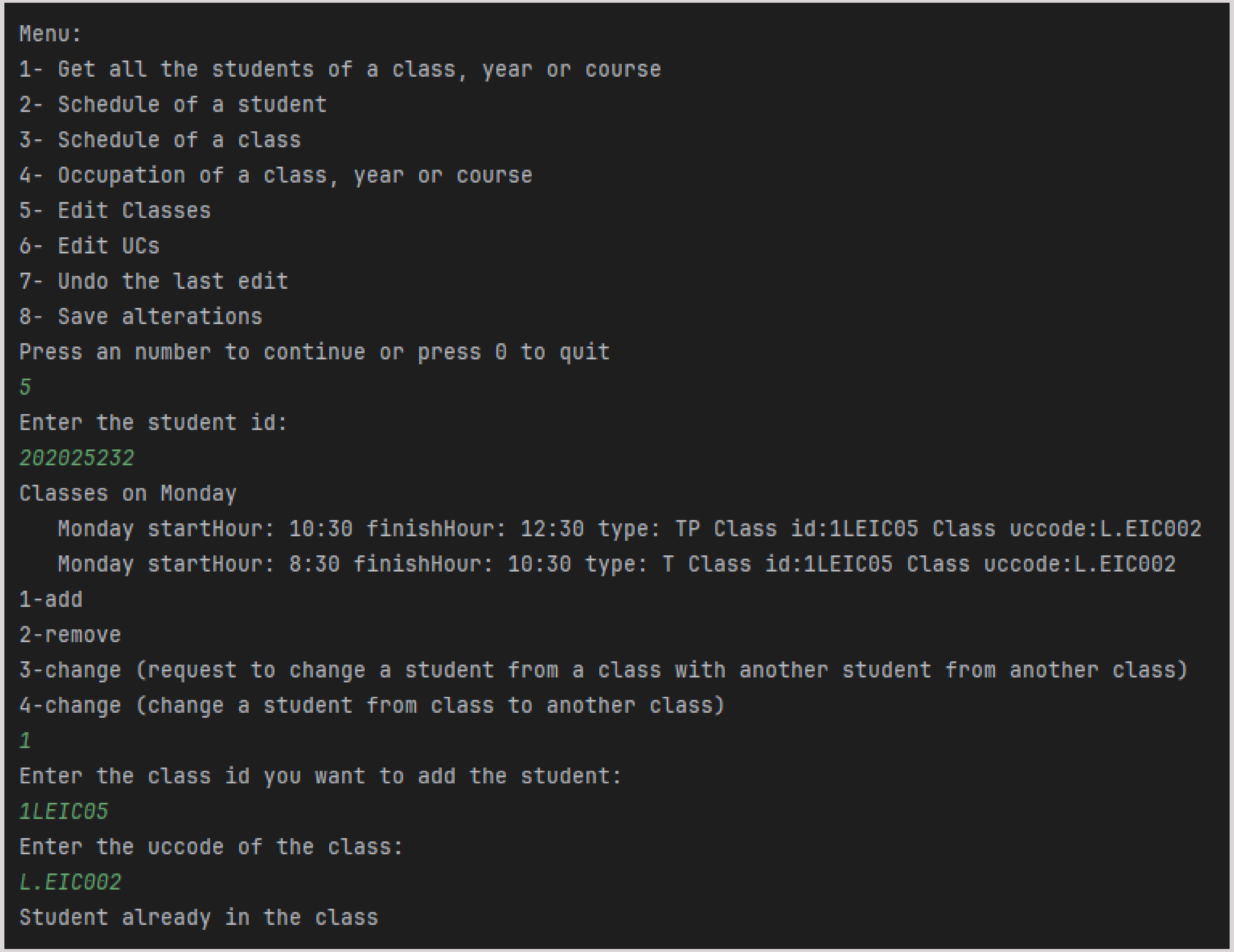


```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
```

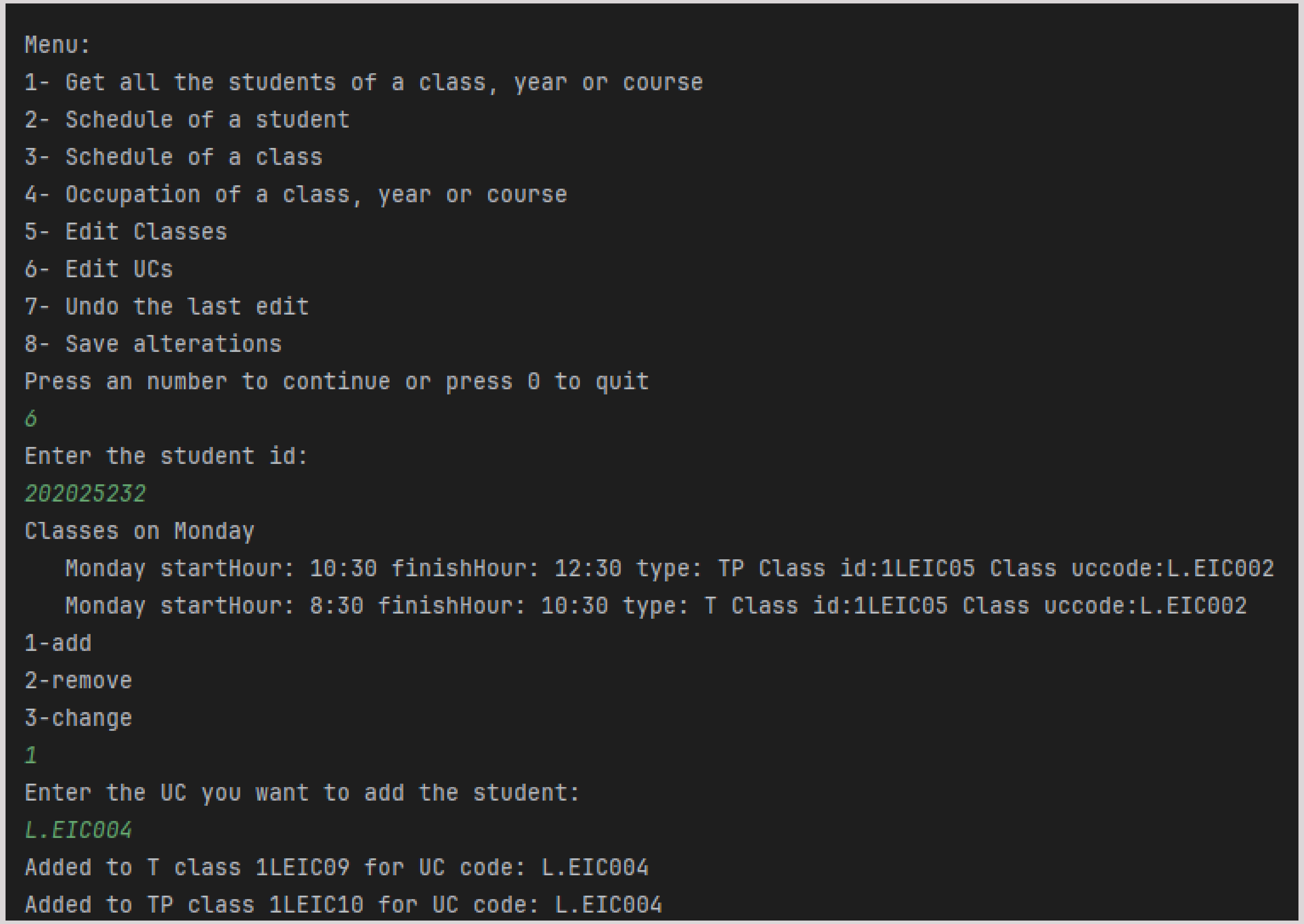


```
5
Enter the student id:
202040617
Classes on Monday
   Monday startHour: 16:00 finishHour: 17:00 type: T Class id:3LEIC12 Class uccode:L.EIC021
   Monday startHour: 10:30 finishHour: 12:30 type: T Class id:3LEIC13 Class uccode:L.EIC023
   Monday startHour: 14:00 finishHour: 16:00 type: TP Class id:3LEIC12 Class uccode:L.EIC024
Classes on Tuesday
   Tuesday startHour: 8:00 finishHour: 9:00 type: T Class id:2LEIC10 Class uccode:L.EIC011
   Tuesday startHour: 10:30 finishHour: 12:30 type: PL Class id:3LEIC13 Class uccode:L.EIC023
   Tuesday startHour: 8:30 finishHour: 10:30 type: T Class id:3LEIC12 Class uccode:L.EIC024
Classes on Wednesday
   Wednesday startHour: 8:30 finishHour: 10:30 type: TP Class id:3LEIC12 Class uccode:L.EIC021
   Wednesday startHour: 11:30 finishHour: 12:30 type: T Class id:3LEIC12 Class uccode:L.EIC021
Classes on Thursday
   Thursday startHour: 17:30 finishHour: 19:30 type: TP Class id:2LEIC10 Class uccode:L.EIC011
   Thursday startHour: 14:30 finishHour: 15:30 type: T Class id:2LEIC10 Class uccode:L.EIC011
   Thursday startHour: 10:30 finishHour: 12:30 type: TP Class id:3LEIC10 Class uccode:L.EIC025
   Thursday startHour: 8:30 finishHour: 10:30 type: T Class id:3LEIC10 Class uccode:L.EIC025
Classes on Friday
   Friday startHour: 9:00 finishHour: 10:00 type: TP Class id:3LEIC08 Class uccode:L.EIC022
   Friday startHour: 10:30 finishHour: 11:30 type: T Class id:3LEIC08 Class uccode:L.EIC022
1-add
2-remove
3-change (request to change a student from a class with another student from another class)
4-change (change a student from class to another class)
3
```

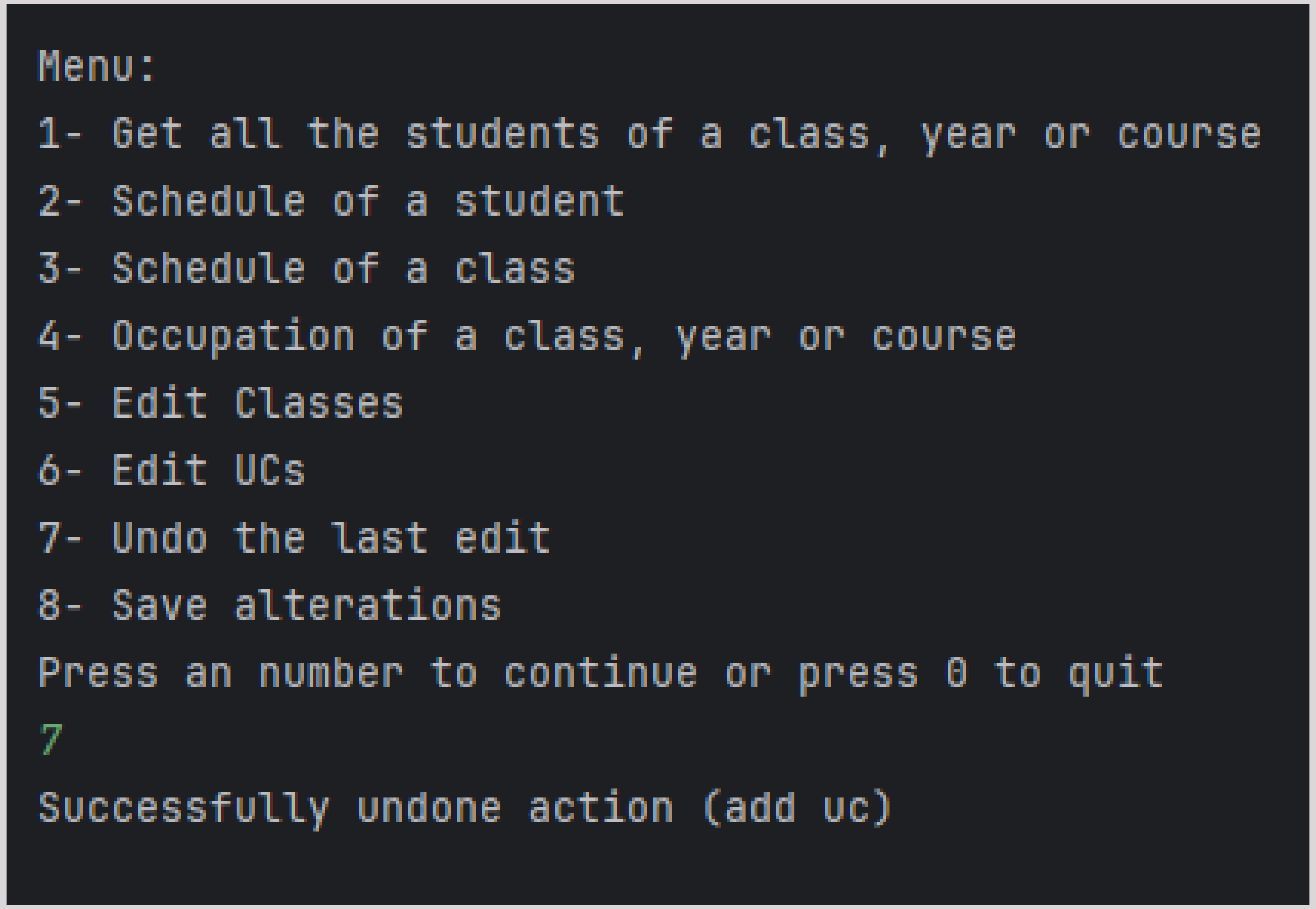

```
Enter the class you want to remove the student:
3LEIC12
Enter the class you want to add the student
3LEIC04
Enter the uccode of the classes:
L.EIC021
Conflito de hor├írio detectado com a classe 3LEIC08. Tente outra classe.
```

```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
3
Enter the class id:
3LEIC04
Enter the class uccode:
L.EIC021
Weekday: Tuesday startHour: 8:30 finishHour: 10:30 type: TP Class id:3LEIC04 Class uccode:L.EIC021
Weekday: Friday startHour: 9:30 finishHour: 10:30 type: T Class id:3LEIC04 Class uccode:L.EIC021
Weekday: Wednesday startHour: 9:30 finishHour: 10:30 type: T Class id:3LEIC04 Class uccode:L.EIC021
```



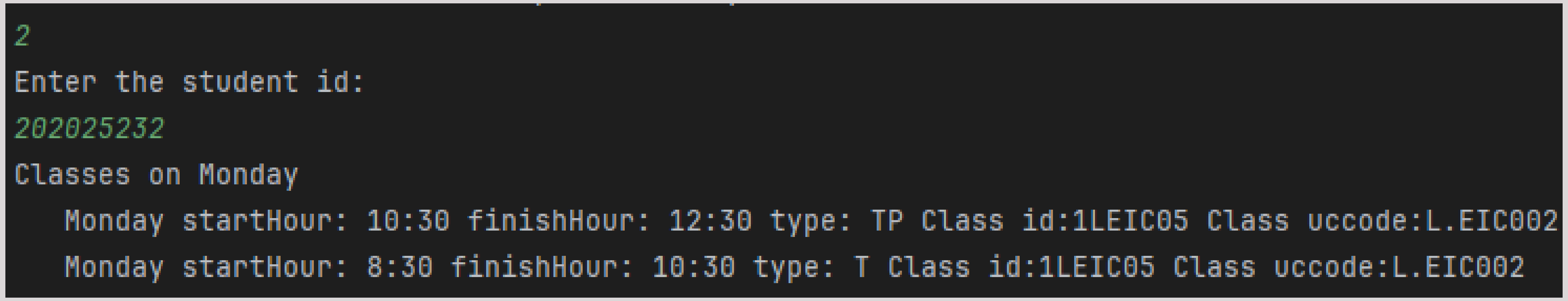
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
5
Enter the student id:
202025232
Classes on Monday
   Monday startHour: 10:30 finishHour: 12:30 type: TP Class id:1LEIC05 Class uccode:L.EIC002
   Monday startHour: 8:30 finishHour: 10:30 type: T Class id:1LEIC05 Class uccode:L.EIC002
1-add
2-remove
3-change (request to change a student from a class with another student from another class)
4-change (change a student from class to another class)
1
Enter the class id you want to add the student:
1LEIC05
Enter the uccode of the class:
L.EIC002
Student already in the class
```



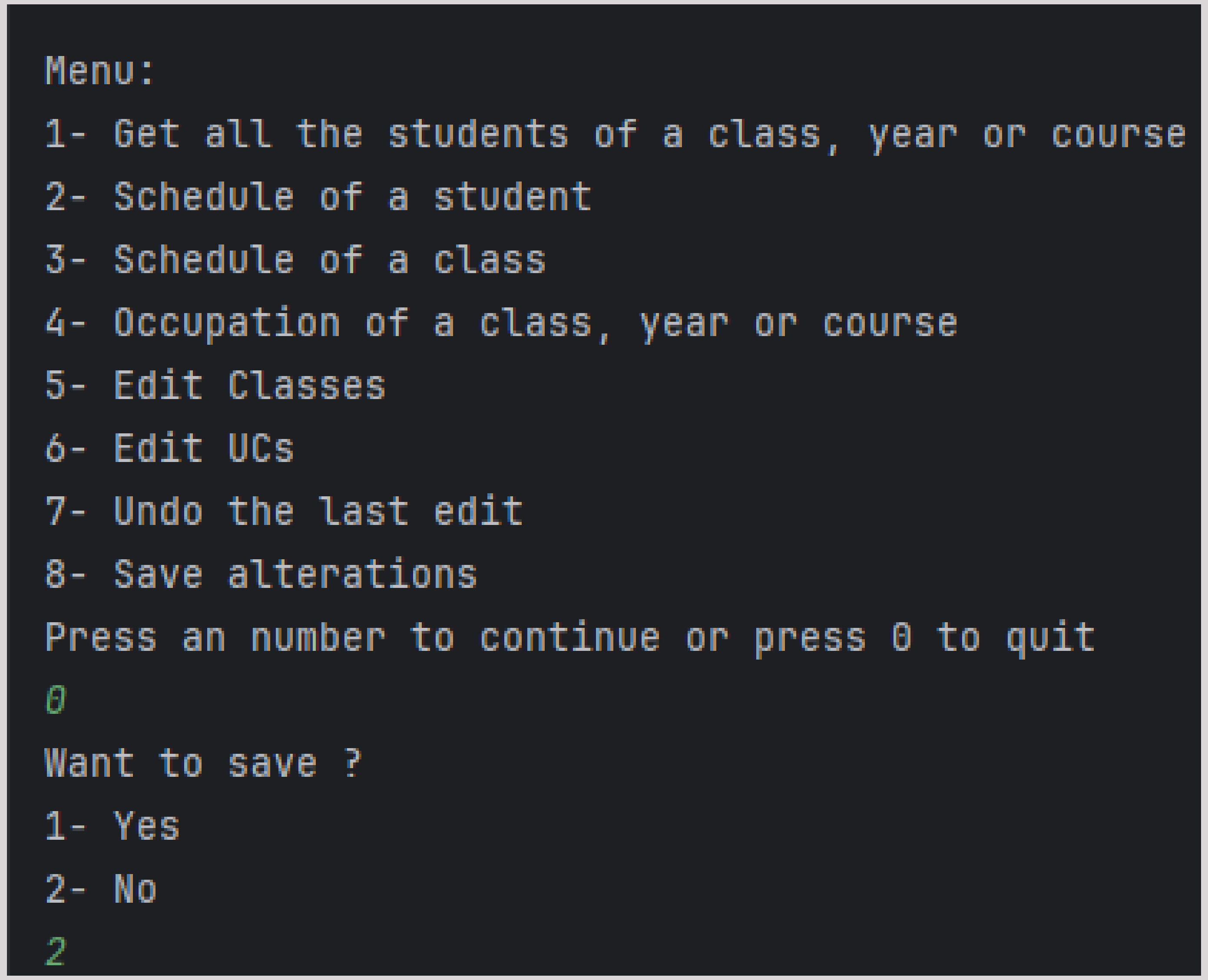
```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
6
Enter the student id:
202025232
Classes on Monday
   Monday startHour: 10:30 finishHour: 12:30 type: TP Class id:1LEIC05 Class uccode:L.EIC002
   Monday startHour: 8:30 finishHour: 10:30 type: T Class id:1LEIC05 Class uccode:L.EIC002
1-add
2-remove
3-change
1
Enter the UC you want to add the student:
L.EIC004
Added to T class 1LEIC09 for UC code: L.EIC004
Added to TP class 1LEIC10 for UC code: L.EIC004
```



```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
7
Successfully undone action (add uc)
```



```
2
Enter the student id:
202025232
Classes on Monday
   Monday startHour: 10:30 finishHour: 12:30 type: TP Class id:1LEIC05 Class uccode:L.EIC002
   Monday startHour: 8:30 finishHour: 10:30 type: T Class id:1LEIC05 Class uccode:L.EIC002
```



```
Menu:
1- Get all the students of a class, year or course
2- Schedule of a student
3- Schedule of a class
4- Occupation of a class, year or course
5- Edit Classes
6- Edit UCs
7- Undo the last edit
8- Save alterations
Press an number to continue or press 0 to quit
0
Want to save ?
1- Yes
2- No
2
```

Principais Dificuldades

CHANGE

A ordem de prioriadade na troca de turmas entre alunos

UNDO

Usar a stack para desfazer cada uma das operações

PARSING

Não repetir os estudantes e as turmas ao ler os arquivos