AED2324

2.0

Generated by Doxygen 1.10.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

alter								 					 									 			
classIr	nfo							 					 									 			
classC	Qtd							 					 									 			
myStu	den	t						 					 									 			
myUc								 					 									 			

2 Class Index

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

rc/errorMsgs.cpp	??
erc/main.cpp	??
erc/menu.cpp	??
erc/menu.h	??
erc/classes/student.cpp	??
erc/classes/student.h	??
rc/classes/uc.cpp	??
erc/classes/uc.h	??
erc/functions/dbStudents.cpp	??
erc/functions/dbStudents.h	??
erc/functions/dbUcs.cpp	??
erc/functions/dbUcs.h	??
erc/inputoutput/keepAllChanges.cpp	??
rc/inputoutput/keepAllChanges.h	??
erc/inputoutput/print.cpp	??
rc/inputoutput/print.h	??
erc/inputoutput/read.cpp	??
rc/inputoutput/read.h	??

File Index

Chapter 3

Class Documentation

3.1 alter Struct Reference

```
#include <student.h>
```

Public Attributes

- std::string studentCode
- std::string studentName
- std::string type
- std::string ucCode
- std::string classCode

3.1.1 Detailed Description

Definition at line 10 of file student.h.

3.1.2 Member Data Documentation

3.1.2.1 classCode

std::string alter::classCode

Definition at line 15 of file student.h.

3.1.2.2 studentCode

std::string alter::studentCode

Definition at line 11 of file student.h.

3.1.2.3 studentName

```
std::string alter::studentName
```

Definition at line 12 of file student.h.

3.1.2.4 type

```
std::string alter::type
```

Definition at line 13 of file student.h.

3.1.2.5 ucCode

```
std::string alter::ucCode
```

Definition at line 14 of file student.h.

The documentation for this struct was generated from the following file:

• src/classes/student.h

3.2 classInfo Struct Reference

```
#include <uc.h>
```

Public Member Functions

• bool operator< (const classInfo &other) const

Public Attributes

- std::string code
- std::string type
- std::string day
- int dayInt
- double startTime
- double duration

3.2.1 Detailed Description

Definition at line 8 of file uc.h.

3.2.2 Member Function Documentation

3.2.2.1 operator<()

3.2.3 Member Data Documentation

3.2.3.1 code

```
std::string classInfo::code
```

Definition at line 9 of file uc.h.

3.2.3.2 day

```
std::string classInfo::day
```

Definition at line 11 of file uc.h.

3.2.3.3 dayInt

```
int classInfo::dayInt
```

Definition at line 12 of file uc.h.

3.2.3.4 duration

double classInfo::duration

Definition at line 14 of file uc.h.

3.2.3.5 startTime

double classInfo::startTime

Definition at line 13 of file uc.h.

3.2.3.6 type

```
std::string classInfo::type
```

Definition at line 10 of file uc.h.

The documentation for this struct was generated from the following file:

src/classes/uc.h

3.3 classQtd Struct Reference

```
#include <uc.h>
```

Public Member Functions

bool operator< (const classQtd &other) const

Public Attributes

- std::string classCode
- int qtd

3.3.1 Detailed Description

Definition at line 21 of file uc.h.

3.3.2 Member Function Documentation

3.3.2.1 operator<()

3.3.3 Member Data Documentation

3.3.3.1 classCode

```
std::string classQtd::classCode
```

Definition at line 22 of file uc.h.

3.3.3.2 qtd

int classQtd::qtd

Definition at line 23 of file uc.h.

The documentation for this struct was generated from the following file:

· src/classes/uc.h

3.4 myStudent Class Reference

#include <student.h>

Public Member Functions

• myStudent (const std::string &sCode, const std::string &sName)

Constructor for the myStudent class.

• myStudent ()

Default constructor for the myStudent class.

- void setStudent (std::string &sCode, std::string &sName)
- void setStudentCode (std::string &n)

Set the student code for the myStudent object.

void setStudentName (std::string &n)

Set the student name for the myStudent object.

• std::string getStudentCode () const

Get the student code for the myStudent object.

std::string getStudentName () const

Get the student name for the myStudent object.

• std::vector< std::string > getUcCode () const

Returns a vector of UC codes associated with this student.

std::vector< std::string > getClassCode () const

Returns a vector of class codes associated with this student.

- std::vector< myUc > & getClasses () const

Gets a reference to the vector containing the student's classes.

void addClass (const myUc &myClass)

Adds a class to the student's classes vector.

- void changeClass (const myUc &myClass)
- void addUc (const myUc &myClass)
- void removeUc (const myUc &myClass)
- bool valideQtClasses ()

Validates if the quantity of classes exceeds the limit.

3.4.1 Detailed Description

Definition at line 18 of file student.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 myStudent() [1/2]

Constructor for the myStudent class.

Parameters

code	Student's code.
name	Student's name.

Definition at line 8 of file student.cpp.

3.4.2.2 myStudent() [2/2]

```
myStudent::myStudent ( )
```

Default constructor for the myStudent class.

This constructor initializes a myStudent object with default values for the student code and student name.

Definition at line 18 of file student.cpp.

```
00018 {
00019 studentCode = "";
00020 studentName = "";
00021 }
```

3.4.3 Member Function Documentation

3.4.3.1 addClass()

Adds a class to the student's classes vector.

Parameters

classe	ClassComp object to be added.	l
Classe	ClassComp object to be added.	l

Definition at line 99 of file student.cpp.

```
00099 { classes.push_back(myClass); }
```

3.4.3.2 addUc()

3.4.3.3 changeClass()

3.4.3.4 getClassCode()

```
std::vector< std::string > myStudent::getClassCode ( ) const
```

Returns a vector of class codes associated with this student.

This function iterates through the courses (UCs) associated with this student and collects the class codes of each course in the class code vector.

Returns

A vector of strings containing the class codes associated with the student.

Definition at line 78 of file student.cpp.

```
00078
00079
00080
std::vector<std::string> classCodes;
00081
ostd::vector<classInfo> classInfoVec = uc.getClassInfoVec();
00082
for (const auto &classInfo : classInfoVec) {
    classCodes.push_back(classInfo.code);
    }
00083
    return classCodes;
00087
}
```

3.4.3.5 getClasses()

```
std::vector< myUc > & myStudent::getClasses ( ) const
```

Gets a reference to the vector containing the student's classes.

Returns

A reference to a vector of ClassComp objects.

Definition at line 93 of file student.cpp.

```
00093 { return classes; }
```

3.4.3.6 getStudentCode()

```
std::string myStudent::getStudentCode ( ) const
```

Get the student code for the myStudent object.

Returns

The student code.

Definition at line 44 of file student.cpp.

```
00044 { return studentCode; }
```

3.4.3.7 getStudentName()

```
std::string myStudent::getStudentName ( ) const
```

Get the student name for the myStudent object.

Returns

The student name.

Definition at line 50 of file student.cpp.

```
00050 { return studentName; }
```

3.4.3.8 getUcCode()

```
std::vector< std::string > myStudent::getUcCode ( ) const
```

Returns a vector of UC codes associated with this student.

This function iterates through the courses (UCs) associated with this student and collects the UC codes of each course in the UC code vector.

Returns

A vector of strings containing the UC codes associated with the student.

Definition at line 61 of file student.cpp.

3.4.3.9 removeUc()

3.4.3.10 setStudent()

```
void myStudent::setStudent (
          std::string & sCode,
          std::string & sName )
```

Definition at line 23 of file student.cpp.

```
00023

00024 studentCode = sCode;

00025 studentName = sName;

00026 }
```

3.4.3.11 setStudentCode()

```
void myStudent::setStudentCode ( std::string \ \& \ n \ )
```

Set the student code for the myStudent object.

Parameters

n The new student code to be set.

Definition at line 32 of file student.cpp.

```
00032 { studentCode = n; }
```

3.4.3.12 setStudentName()

```
void myStudent::setStudentName ( std::string \ \& \ n \ )
```

Set the student name for the myStudent object.

Parameters

```
n \mid The new student name to be set.
```

Definition at line 38 of file student.cpp.

```
00038 { studentName = n; }
```

3.4.3.13 valideQtClasses()

```
bool myStudent::valideQtClasses ( )
```

Validates if the quantity of classes exceeds the limit.

Returns

True if the number of classes is greater than 7, false otherwise.

Definition at line 105 of file student.cpp.

```
00105

00106 if (classes.size() > 7) {

00107 return true;

00108 }

00109 return false;

00110 }
```

The documentation for this class was generated from the following files:

- · src/classes/student.h
- src/classes/student.cpp

3.5 myUc Class Reference

```
#include <uc.h>
```

Public Member Functions

• myUc (const std::string &ucC, std::string &classC)

Constructor for the myUc class.

• myUc ()

Default constructor for the myUc class.

void SetUc (std::string &ucC, std::string &classC)

Set the UC code for the myUc object.

void setUcCode (std::string &n)

Set the UC code for the myUc object.

void setClassCode (std::string &n)

Set the class code for the myUc object.

• std::string getUcCode () const

Get the UC code for the myUc object.

• std::string getClassCode () const

Get the class code for the myUc object.

• std::vector< classInfo > getClassInfoVec () const

Returns a vector of classInfo associated with this UC.

void addClass (const std::string &code)

Adds a class to the classInfo vector.

void addClassInfo (std::string type, std::string day, int dayInt, double startTime, double duration)

Adds a class to the classInfo vector.

bool operator< (const myUc &other) const

Static Public Member Functions

• static bool compareUcCode (const myUc &a, const myUc &b)

Adds a class to the classInfo vector.

3.5.1 Detailed Description

Definition at line 30 of file uc.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 myUc() [1/2]

Constructor for the myUc class.

Parameters

code	UC's code.
name	UC's name.

Definition at line 8 of file uc.cpp.

```
00008

00009 ucCode = ucC;

00010 classCode = classC;

00011 }
```

3.5.2.2 myUc() [2/2]

```
myUc::myUc ( )
```

Default constructor for the myUc class.

This constructor initializes a myUc object with default values for the UC code and class code.

Definition at line 19 of file uc.cpp.

```
00019 {
00020 ucCode = "";
00021 classCode = {};
00022 }
```

3.5.3 Member Function Documentation

3.5.3.1 addClass()

Adds a class to the classInfo vector.

This function adds a class to the classInfo vector.

Parameters

```
code The class code to be added.
```

Definition at line 75 of file uc.cpp.

```
00075 { classCode = code; }
```

3.5.3.2 addClassInfo()

Adds a class to the classInfo vector.

This function adds a class to the classInfo vector.

Parameters

Definition at line 84 of file uc.cpp.

```
00085
00086 classInfo newClassInfo;
00087 newClassInfo.type = type;
00088 newClassInfo.day = day;
00089 newClassInfo.dayInt = dayInt;
00090 newClassInfo.startTime = startTime;
00091 newClassInfo.duration = duration;
00092
00093 classInfoVec.push_back(newClassInfo);
00094
```

3.5.3.3 compareUcCode()

Adds a class to the classInfo vector.

This function adds a class to the classInfo vector.

Parameters

code The class code to be added.

Definition at line 103 of file uc.cpp.

3.5.3.4 getClassCode()

```
std::string myUc::getClassCode ( ) const
```

Get the class code for the myUc object.

Returns

The class code.

Definition at line 55 of file uc.cpp.

```
00055 { return classCode; }
```

3.5.3.5 getClassInfoVec()

```
std::vector< classInfo > myUc::getClassInfoVec ( ) const
```

Returns a vector of classInfo associated with this UC.

This function iterates through the classInfo (classes) associated with this UC and collects the classInfo of each class in the classInfo vector.

Returns

A vector of classInfo containing the classInfo associated with the UC.

Definition at line 66 of file uc.cpp.

```
00066 { return classInfoVec; }
```

3.5.3.6 getUcCode()

```
std::string myUc::getUcCode ( ) const
```

Get the UC code for the myUc object.

Returns

The UC code.

Definition at line 49 of file uc.cpp.

```
00049 { return ucCode; }
```

3.5.3.7 operator<()

3.5.3.8 setClassCode()

```
void myUc::setClassCode ( std::string & n )
```

Set the class code for the myUc object.

Parameters

```
n The new class code to be set.
```

Definition at line 43 of file uc.cpp.

```
00043 { classCode = n; }
```

3.5.3.9 SetUc()

Set the UC code for the myUc object.

Parameters

```
n The new UC code to be set.
```

Definition at line 28 of file uc.cpp.

```
00028

00029 ucCode = ucC;

00030 classCode = classC;

00031 }
```

3.5.3.10 setUcCode()

```
void myUc::setUcCode ( std::string \ \& \ n \ )
```

Set the UC code for the myUc object.

Parameters

n The new UC code to be set.

Definition at line 37 of file uc.cpp. 00037 { ucCode = n; }

The documentation for this class was generated from the following files:

- src/classes/uc.h
- src/classes/uc.cpp

Chapter 4

File Documentation

4.1 src/classes/student.cpp File Reference

```
#include "student.h"
```

4.2 student.cpp

Go to the documentation of this file.

```
00001 #include "student.h'
00002
00008 myStudent::myStudent(const std::string &sCode, const std::string &sName) {
00009 studentCode = sCode;
        studentName = sName;
00011 }
00018 myStudent::myStudent() {
00019 studentCode = "";
00020 studentName = "";
00021 }
00022
00023 void myStudent::setStudent(std::string &sCode, std::string &sName) {
00024 studentCode = sCode;
00025 studentName = sName;
00026 }
00027
00032 void myStudent::setStudentCode(std::string &n) { studentCode = n; }
00033
00038 void myStudent::setStudentName(std::string &n) { studentName = n; }
00039
00044 std::string myStudent::getStudentCode() const { return studentCode; }
00045
00050 std::string myStudent::getStudentName() const { return studentName; }
00061 std::vector<std::string> myStudent::getUcCode() const {
00062 std::vector<std::string> ucCodes;
00063 for (const auto &uc : classes) {
00064
          ucCodes.push_back(uc.getUcCode());
00065
00066
       return ucCodes;
00067 }
00068
00078 std::vector<std::string> myStudent::getClassCode() const {
00079 std::vector<std::string> classCodes;
00080 for (const auto &uc : classes) {
        std::vector<classInfo> classInfoVec = uc.getClassInfoVec();
for (const auto &classInfo : classInfoVec) {
00082
00083
             classCodes.push_back(classInfo.code);
00084
00085
00086
        return classCodes:
00087 }
00088
```

```
00093 std::vector<myUc> &myStudent::getClasses() const { return classes; }
00094
00099 void myStudent::addClass(const myUc &myClass) { classes.push_back(myClass); }
00100
00105 bool myStudent::valideQtClasses() {
00106    if (classes.size() > 7) {
00107        return true;
00108    }
00109    return false;
00110 }
```

4.3 src/classes/student.h File Reference

```
#include <iostream>
#include <string>
#include <vector>
#include "uc.h"
```

Classes

- · struct alter
- class myStudent

4.4 student.h

Go to the documentation of this file.

```
00001 #ifndef MYSTUDENT_H
00002 #define MYSTUDENT_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <vector>
00007
00008 #include "uc.h"
00009
00010 struct alter {
00011 std::string studentCode;
00012 std::string studentName;
00013
       std::string type;
00014 std::string ucCode;
00015
       std::string classCode;
00016 };
00017
00018 class myStudent {
00019 private:
00020 std::string studentCode;
00021
00021 std::string studentName;
00022 mutable std::vector<myUc> classes;
00023
00024 public:
00025
       // Constructor functions
       myStudent(const std::string &sCode, const std::string &sName);
00026
00027
       myStudent();
00028
00029
        // Setter functions
00030
        void setStudent(std::string &sCode, std::string &sName);
00031
        void setStudentCode(std::string &n);
00032
        void setStudentName(std::string &n);
00033
       // Getters functions
00034
00035
       std::string getStudentCode() const;
00036
       std::string getStudentName() const;
00037
        std::vector<std::string> getUcCode() const;
00038
       std::vector<std::string> getClassCode() const;
00039
       std::vector<myUc> &getClasses() const;
00040
00041
       // Others functions
00042
       void addClass(const myUc &myClass);
```

4.5 src/classes/uc.cpp File Reference

```
#include "uc.h"
```

4.6 uc.cpp

Go to the documentation of this file.

```
00001 #include "uc.h'
00002
00008 myUc::myUc(const std::string &ucC, std::string &classC) {
00009 ucCode = ucC;
00010
        classCode = classC;
00011 }
00012
00019 myUc::myUc()
00020 ucCode = "";
00021 classCode =
        classCode = {};
00022 }
00023
00028 void myUc::SetUc(std::string &ucC, std::string &classC) {
00029 ucCode = ucC;
00030 classCode = c
        classCode = classC;
00031 }
00032
00037 void myUc::setUcCode(std::string &n) { ucCode = n; }
00038
00043 void myUc::setClassCode(std::string &n) { classCode = n; }
00044
00049 std::string myUc::getUcCode() const { return ucCode; }
00050
00055 std::string myUc::getClassCode() const { return classCode; }
00056
00066 std::vector<classInfo> myUc::getClassInfoVec() const { return classInfoVec; }
00067
00075 void myUc::addClass(const std::string &code) { classCode = code; }
00076
{\tt 00084\ void\ myUc::addClassInfo(std::string\ type,\ std::string\ day,\ int\ dayInt,}
00085
                                  double startTime, double duration) {
00086
        classInfo newClassInfo;
00086 classInfo newClassInfo;
00087 newClassInfo.type = type;
00088 newClassInfo.day = day;
00089 newClassInfo.dayInt = dayInt;
00090 newClassInfo.startTime = startTime;
00091 newClassInfo.duration = duration;
00092
00093
        classInfoVec.push_back(newClassInfo);
00094 }
00095
00103 bool myUc::compareUcCode(const myUc &a, const myUc &b) {
00104
        return a.ucCode < b.ucCode;
00105 }
```

4.7 src/classes/uc.h File Reference

```
#include <iostream>
#include <string>
#include <vector>
```

Classes

- struct classInfo
- struct classQtd
- class myUc

4.8 uc.h

Go to the documentation of this file.

```
00001 #ifndef MYUC_H
00002 #define MYUC_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <vector>
00007
00008 struct classInfo {
00009 std::string code;
       std::string type;
00011
       std::string day;
00012
       int dayInt;
00013
       double startTime;
00014
       double duration;
00015
00017 return startTime < other.startTime;
00018 }
       bool operator<(const classInfo &other) const {</pre>
00019 };
00020
00021 struct classQtd {
00022
       std::string classCode;
00023
       int qtd;
00024
00025
       bool operator<(const classQtd &other) const {</pre>
       return classCode < other.classCode;
}</pre>
00026
00027
00028 };
00030 class myUc {
00032 std::string classCode; std::string classCode;
00031 private:
00034
       std::vector<classInfo> classInfoVec;
00035
00036 public:
00037
       // Constructor functions
00038
       myUc(const std::string &ucC, std::string &classC);
00039
       myUc();
00040
00041
       // Setter functions
00042
       void SetUc(std::string &ucC, std::string &classC);
00043
       void setUcCode(std::string &n);
00044
       void setClassCode(std::string &n);
00045
       // void setClassCode(std::string &n);
00046
00047
       // Getters functions
00048
       std::string getUcCode() const;
00049
       std::string getClassCode() const;
00050
       std::vector<classInfo> getClassInfoVec() const;
00051
00052
       // Others functions
00053
       void addClass(const std::string &code);
       00055
00056
00057
       bool operator<(const myUc &other) const;</pre>
00058
       static bool compareUcCode (const myUc &a, const myUc &b);
00059 };
00060
00061 #endif
```

4.9 src/errorMsgs.cpp File Reference

```
#include <iostream>
#include <string>
```

Functions

- void errorMessage ()
- void errorCheck (int n)
- void errorMessageFile ()
- void errorMessageLine (std::string line)
- void workingMessage ()

4.9.1 Function Documentation

4.9.1.1 errorCheck()

```
void errorCheck ( int n)
```

Definition at line 9 of file errorMsgs.cpp.

```
00009 {
00010 if (n == 0) {
00011 std::cout « "ERROR: Invalid number" « std::endl;
00012 exit(0);
00013 }
00014 }
```

4.9.1.2 errorMessage()

```
void errorMessage ( )
```

Definition at line 4 of file errorMsgs.cpp.

```
00004

00005 std::cout « "ERROR: Invalid choice." « std::endl;

00006 exit(0);

00007 }
```

4.9.1.3 errorMessageFile()

```
void errorMessageFile ( )
```

Definition at line 16 of file errorMsgs.cpp.

```
00016 {
00017 std::cerr « "Error: Could not open the file." « std::endl;
00018 exit(0);
00019 }
```

4.9.1.4 errorMessageLine()

```
void errorMessageLine (
          std::string line )
```

Definition at line 21 of file errorMsgs.cpp.

```
00021 {
00022 std::cerr « "Error: Invalid data format in line: " « line « std::endl;
00023 exit(0);
00024 }
```

4.9.1.5 workingMessage()

```
void workingMessage ( )

Definition at line 26 of file errorMsgs.cpp.
00026
00027    std::cout « "WARNING: Function not done yet." « std::endl;
00028 }
```

4.10 errorMsgs.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <string>
00003
00004 void errorMessage() {
00005 std::cout « "ERROR: Invalid choice." « std::endl;
00006
       exit(0);
00007 }
80000
00009 void errorCheck(int n) {
00010 if (n == 0) {
        std::cout « "ERROR: Invalid number" « std::endl;
00011
00012
         exit(0);
00013 }
00014 }
00015
00016 void errorMessageFile() {
00017 std::cerr « "Error: Could not open the file." « std::endl; 00018 exit(0);
00019 }
00020
00021 void errorMessageLine(std::string line) {
00022 std::cerr « "Error: Invalid data format in line: " « line « std::endl;
00023
       exit(0);
00024 }
00025
00026 void workingMessage() {
00027 std::cout « "WARNING: Function not done yet." « std::endl;
00028 }
```

4.11 src/functions/dbStudents.cpp File Reference

```
#include "dbStudents.h"
```

Functions

bool compareStudentsCodeAsc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their codes in ascending order.

• bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their codes in descending order.

• bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their names in ascending order.

bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their names in descending order.

std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)

Filter student information based on specified criteria.

• std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)

Order a vector of students based on the specified criterion.

std::map< std::string, myStudent > selectStudent (const std::string &str, const std::map< std::string, myStudent > &students)

Select students from a map based on a given student code.

void organizerUcStudent (std::map< std::string, myStudent >::iterator &it)

Organize the classes of a student based on Uc Code in ascending order.

bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter
 &stackAlter, std::map< std::string, std::vector< classQtd >> &count)

Remove a specific Uc Code from a student's classes.

void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator
 &it, std::stack< alter > &stackAlter)

Add a new class to a student's record.

void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector<
 classQtd >> &count, int type)

Update the class count tree by adding or removing a class.

bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator
 &it, std::map< std::string, myUc > &classes)

Validate the addition of a new class to a student's schedule.

std::map< int, std::set< classInfo > > orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

Organize a student's classes by day.

std::string weekDayString (int day)

Convert a numeric day value to a corresponding day of the week string.

bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)

Check whether a student is already enrolled in a UC class.

4.11.1 Function Documentation

4.11.1.1 addClassStudent()

```
void addClassStudent (
    std::string ucCode,
    std::string classCode,
    std::map< std::string, myStudent >::iterator & it,
    std::stack< alter > & stackAlter )
```

Add a new class to a student's record.

Parameters

ucCode	Uc Code of the new class.
classCode	Class Code of the new class.
it	Iterator pointing to a student in the map of students.
stackAlter	Stack for recording changes.

This function receives a reference to a student pointer and adds a new class to the student's record. It also organizes the classes, records the change in the stack, and updates the student's class list.

```
Definition at line 222 of file dbStudents.cpp.
```

```
00224
00225
00226 myUc classe(ucCode, classCode);
00227 it->second.getClasses().push_back(classe);
00228 organizerUcStudent(it);
```

4.11.1.2 compareStudentNameAsc()

Function to compare students based on their names in ascending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the name of the first student is less than that of the second.

```
Definition at line 35 of file dbStudents.cpp.
```

```
00036
00037 return student1.getStudentName() < student2.getStudentName();
00038 }</pre>
```

4.11.1.3 compareStudentNameDesc()

Function to compare students based on their names in descending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the name of the first student is greater than that of the second.

Definition at line 47 of file dbStudents.cpp.

```
00048
00049    return student1.getStudentName() > student2.getStudentName();
00050 }
```

4.11.1.4 compareStudentsCodeAsc()

Function to compare students based on their codes in ascending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the code of the first student is less than that of the second.

```
Definition at line 11 of file dbStudents.cpp.
```

```
00012
00013 return student1.getStudentCode() < student2.getStudentCode();
00014 }</pre>
```

4.11.1.5 compareStudentsCodeDesc()

Function to compare students based on their codes in descending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the code of the first student is greater than that of the second.

Definition at line 23 of file dbStudents.cpp.

```
00024
00025   return student1.getStudentCode() > student2.getStudentCode();
00026 }
```

4.11.1.6 filterInfoStudent()

```
\begin{tabular}{lll} {\tt std::vector}<& {\tt myStudent}>& {\tt filterInfoStudent}& (\\ & & {\tt int}~n,\\ & & {\tt std::string}~str,\\ & & {\tt const}~std::{\tt vector}<& {\tt myStudent}>& & students \end{tabular} \label{eq:std}
```

Filter student information based on specified criteria.

Parameters

n	Filter criterion: 1 for Uc Code, 2 for Class Code.
str	Search string.
students	Vector of students to filter.

Returns

Vector of students matching the criteria.

Definition at line 63 of file dbStudents.cpp.

```
00064
00065
        std::vector<myStudent> filterStudents;
00066
        switch (n) {
00067
        case 1:
00068
          // Filter by Uc Code
00069
           for (const auto &student : students) {
00070
             for (const auto &uc : student.getClasses()) {
  if (uc.getUcCode() == str) {
00071
                filterStudents.push_back(student);
00073
                 break;
00074
            }
00075
00076
00077
          break;
00078
        case 2:
00079
         // Filter by Class Code
08000
           for (const auto &student : students) {
00081
            for (const auto &uc : student.getClasses()) {
               for (const auto &classInfo : uc.getClassInfoVec()) {
  if (classInfo.code == str) {
00082
00083
00084
                  filterStudents.push_back(student);
00085
                   break; // No need to check other class codes for this student
00086
00087
            }
00088
00089
00090
          break;
00091
        default:
00092
          errorMessage();
00093
          break;
00094
00095
        return filterStudents;
00096 }
```

4.11.1.7 orderInfoStudent()

```
\begin{tabular}{ll} {\tt std::vector}<& {\tt myStudent}>& {\tt orderInfoStudent}&(\\ & {\tt int}&n,\\ & {\tt std::vector}<& {\tt myStudent}>& & {\tt students} &(\\ \end{tabular}
```

Order a vector of students based on the specified criterion.

Parameters

n	Ordering criterion: 1 for ascending Student Code, 2 for descending Student Code, 3 for ascending Student Name, 4 for descending Student Name.	
students	Vector of students to be ordered.	

Returns

Ordered vector of students based on the specified criterion.

Definition at line 108 of file dbStudents.cpp.

```
00109
        switch (n) {
00110
00111
        case 1:
         // Order by Student Code Asc
00112
00113
         std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00114
       case 2:
   // Order by Student Code Desc
00115
00116
00117
         std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00118
         break;
00119
        case 3:
```

```
// Order by Student Name Asc
00121
          std::sort(students.begin(), students.end(), compareStudentNameAsc);
00122
         break;
       case 4:
  // Order by Student Name Desc
00123
00124
         std::sort(students.begin(), students.end(), compareStudentNameDesc);
00125
00126
         break;
00127
       default:
       errorMessage();
00128
00129
         break;
       }
00130
00131
00132
       return students;
00133 }
```

4.11.1.8 orderStudentClass()

Organize a student's classes by day.

Parameters

it	Iterator pointing to a student in the map of students.
classes	Map of classes to organize.

Returns

Map of classes organized by day.

This function organizes a student's classes by day and returns a map where classes are grouped by their respective days.

Definition at line 345 of file dbStudents.cpp.

```
00347
00348
        // map to order the classes
        // by day
00349
00350
       std::map<int, std::set<classInfo> orderClasses;
00351
00352
        // for each class of the
00353
        // student, search in the
00354
       // class tree and add the
00355
        // classInfo in the
00356
        // orderClasses map
00357
        for (const auto &classe : it->second.getClasses()) {
00358
         std::string value = classe.getUcCode() + classe.getClassCode();
00359
00360
          // student one class
         // pointer, verify if the
// class exists in the
00361
00362
          // class tree
00363
00364
          auto it_class = classes.find(value);
00365
00366
          // if the class does not
         // exist, print error
if (it_class == classes.end()) {
00367
00368
00369
           std::cerr « "Error in
00370
                         "find class"
00371
                       « std::endl;
         } else {
  // if exists, add the
00372
00373
00374
            // classInfo in the
00375
            // orderClasses map
00376
            for (auto &classInfo : it_class->second.getClassInfoVec()) {
00377
             classInfo.code = classe.getUcCode();
00378
              orderClasses[classInfo.dayInt].insert(classInfo);
00379
00380
         }
00381
       }
00382
        return orderClasses;
00383 }
```

4.11.1.9 organizerUcStudent()

Organize the classes of a student based on Uc Code in ascending order.

Parameters

```
it Iterator pointing to a student in the map of students.
```

This function organizes the classes of a student in ascending order based on their Uc Codes.

Definition at line 170 of file dbStudents.cpp.

```
00170
00171
00172 std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
00173 myUc::compareUcCode);
00174 }
```

4.11.1.10 removeUcStudent()

Remove a specific Uc Code from a student's classes.

Parameters

ucCode	Uc Code to be removed.
it	Iterator pointing to a student in the map of students.
stackAlter	Stack for recording changes.
count	Map for tracking class counts.

Returns

Returns true if the Uc Code was successfully removed; otherwise, returns false.

This function removes a specific Uc Code from a student's classes. It records the change in the stack for later reference and updates the class count in the "count" map.

Definition at line 190 of file dbStudents.cpp.

```
00193
00194
00195
                                     bool remove = false;
00196
                                     for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
00197
                                        if (it->second.getClasses()[i].getUcCode() == ucCode) {
00198
                                                        stack \verb|Alter.push({it->}second.getStudentCode(), it->second.getStudentName(), it->second.getStudentN
                                                                                                                                         "remove", ucCode,
it->second.getClasses()[i].getUcCode()});
00199
00200
00201
                                                        it->second.getClasses().erase(it->second.getClasses().begin() + i);
                                                         remove = true;
00202
00203
                                                         updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
00204
                                                                                                                                                  count, 0);
00205
                                              }
00206
                                   }
00207
                                   return remove;
00208 }
```

4.11.1.11 selectStudent()

Select students from a map based on a given student code.

Parameters

str	Student code to search for.
students	Map of students to select from.

Returns

Map of selected students with matching student codes.

Definition at line 145 of file dbStudents.cpp.

```
00147
         std::map<std::string, myStudent> selectedStudents;
00148
00149
         for (auto &studentPair : students) {
         const myStudent & mystudent = studentPair.second;
if (str == mystudent.getStudentCode()) {
00150
00151
00152
             selectedStudents[studentPair.first] = mystudent;
00153
00154
00155
00156
        return selectedStudents;
00157 }
```

4.11.1.12 updateCountClasses()

Update the class count tree by adding or removing a class.

Parameters

ucCode	Uc Code associated with the class.	
classCode	Class Code to be added or removed.	
count	Map for tracking class counts.	
type	1 for adding a class, 0 for removing a class.	

This function updates the class count in the "count" map by either adding or removing a class.

Definition at line 248 of file dbStudents.cpp.

```
00257
00258
00259
00260
00261
00262
00263
00263
00264
}
classe.qtd++;
classe.qtd--;
```

4.11.1.13 valideNewClass()

Validate the addition of a new class to a student's schedule.

Parameters

ucCode	Uc Code of the class to be added.
classCode	Class Code of the class to be added.
it	Iterator pointing to a student in the map of students.
classes	Map of classes to validate against.

Returns

Returns true if the addition of the class is valid, or false if it conflicts with the student's existing schedule.

This function validates whether adding a new class to a student's schedule is compatible with their existing classes. It checks for schedule conflicts and ensures the class exists in the "classes" map.

Definition at line 283 of file dbStudents.cpp.

```
00285
                                                               {
00286
00287
        // call function to order the
       // classes of the student by // int day
00288
00289
00290
       std::map<int, std::set<classInfo» orderClasses =
00291
            orderStudentClass(it, classes);
00292
00293
        std::string value = ucCode + classCode;
00294
00295
        auto it_class = classes.find(value);
00296
00297
        if (it class == classes.end()) {
        std::cout « "Error in "
00298
00299
                       "find class"
00300
                    « std::endl;
00301
          return true;
       } else {
  // verify if has a class in
00302
00303
00304
          // the same day and time
          for (const auto &class_info : it_class->second.getClassInfoVec()) {
00305
           // get all classes of the // day of class
00306
00307
00308
            const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00309
00310
            // and verify if the
00311
            // student has a class in
00312
            // the same time aula -> student classes
00313
            // class_info -> class to add
00314
            for (const auto &aula : classesOfDay) {
00315
00316
              if (aula.type != "T" && class_info.type != "T" &&
00317
                  class_info.startTime >= aula.startTime &&
00318
                  class_info.startTime < aula.startTime + aula.duration) {</pre>
```

```
00319
                std::cout « "Error: "
                              "Incompatible"
" schedules"
00320
00321
                          « std::endl;
00322
00323
                 return true;
00324
00325
           }
00326
00327
          return false;
00328
       }
00329 }
```

4.11.1.14 verifyUcCode()

Check whether a student is already enrolled in a UC class.

Parameters

ucCode	Uc Code to check for enrollment.
it	Iterator pointing to a student in the map of students.

Returns

Returns true if the student is already enrolled in a class with the specified Uc Code; otherwise, returns false.

This function checks whether a student is already enrolled in a class with the specified Uc Code and returns true if a match is found, indicating a potential problem.

Definition at line 430 of file dbStudents.cpp.

4.11.1.15 weekDayString()

Convert a numeric day value to a corresponding day of the week string.

Parameters

```
day Numeric representation of a day (e.g., 2 for Monday).
```

Returns

Corresponding day of the week string.

This function converts a numeric day value to a string representation of the corresponding day of the week.

Definition at line 393 of file dbStudents.cpp.

```
00394
        switch (day) {
00395
        case 2:
        return "Monday";
break;
00396
00397
00398
        case 3:
        return "Tuesday";
break;
00399
00400
00401
        case 4:
        return "Wednesday";
break;
00402
00403
00404
        case 5:
        return "Thursday";
break;
00405
00407
        case 6:
        return "Friday";
break;
00408
00409
        case 7:
00410
        return "Saturday";
break;
00411
00412
00413
        default:
        return "Error Day";
00414
00415
          break;
00416
        }
00417 }
```

4.12 dbStudents.cpp

Go to the documentation of this file.

```
00001 #include "dbStudents.h"
00002
00003 // 0(1)
00011 bool compareStudentsCodeAsc(const myStudent &student1,
00012
                                    const myStudent &student2) {
00013
        return student1.getStudentCode() < student2.getStudentCode();</pre>
00014 }
00015 // 0(1)
00023 bool compareStudentsCodeDesc(const myStudent &student1,
00024
                                     const myStudent &student2) {
00025
        return student1.getStudentCode() > student2.getStudentCode();
00026 }
00027 // 0(1)
00035 bool compareStudentNameAsc(const myStudent &student1,
00036
                                  const myStudent &student2) {
00037
        return student1.getStudentName() < student2.getStudentName();</pre>
00039 // 0(1)
00047 bool compareStudentNameDesc(const myStudent &student1,
00048
        const myStudent &student2) {
return student1.getStudentName() > student2.getStudentName();
00049
00050 }
00052 // O(n)
00053 // n = number of lines in the file
00062 std::vector<myStudent>
00063 filterInfoStudent(int n, std::string str,
00064
                         const std::vector<myStudent> &students) {
        std::vector<myStudent> filterStudents;
00065
00066
        switch (n) {
00067
         // Filter by Uc Code
00068
00069
          for (const auto &student : students) {
00070
           for (const auto &uc : student.getClasses()) {
   if (uc.getUcCode() == str) {
00072
                filterStudents.push_back(student);
00073
00074
00075
            }
00076
          }
00077
          break;
        case 2:
```

4.12 dbStudents.cpp 37

```
// Filter by Class Code
08000
         for (const auto &student : students) {
00081
           for (const auto &uc : student.getClasses()) {
             for (const auto &classInfo : uc.getClassInfoVec()) {
00082
00083
               if (classInfo.code == str) {
00084
                 filterStudents.push_back(student);
                 break; // No need to check other class codes for this student
00086
00087
             }
00088
           }
         }
00089
00090
         break:
00091
       default:
00092
        errorMessage();
00093
         break;
00094
00095
       return filterStudents:
00096 }
00097
00098 // O(n*log(n))
00099 // n = number of lines in the file
00108 std::vector<myStudent> orderInfoStudent(int n,
00109
                                            std::vector<myStudent> &students) {
00110
       switch (n) {
00111
       case 1:
        // Order by Student Code Asc
00112
00113
         std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00114
         break;
       case 2:
  // Order by Student Code Desc
00115
00116
00117
         std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00118
         break;
00119
00120
         // Order by Student Name Asc
00121
         std::sort(students.begin(), students.end(), compareStudentNameAsc);
00122
         break;
00123
       case 4:
        // Order by Student Name Desc
00125
         std::sort(students.begin(), students.end(), compareStudentNameDesc);
00126
00127
       default:
       errorMessage();
00128
00129
         break;
00130
00131
00132
       return students;
00133 }
00134
00135 // O(m)
00136 // m = number of students
00144 std::map<std::string, myStudent>
00145 selectStudent(const std::string &str,
00146
                   const std::map<std::string, myStudent> &students) {
00147
       std::map<std::string, myStudent> selectedStudents;
00148
00149
       for (auto &studentPair : students) {
        const myStudent &mystudent = studentPair.second;
00151
         if (str == mystudent.getStudentCode()) {
00152
           selectedStudents[studentPair.first] = mystudent;
00153
00154
       }
00155
00156
       return selectedStudents;
00157 }
00158
00159 // ----- //
00160
00161 // O(m*log(m))
00162 // m = number of UCs of the student
00170 void organizerUcStudent(std::map<std::string, myStudent>::iterator &it) {
00171
00172
       std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
00173
                myUc::compareUcCode);
00174 }
00175
00176 // O(m)
00177 // m = number of UCs of the student
00190 bool removeUcStudent(std::string ucCode,
00191
                          std::map<std::string, myStudent>::iterator &it,
                          std::stack<alter> &stackAlter.
00192
00193
                          std::map<std::string, std::vector<classQtd> &count) {
00194
00195
       bool remove = false;
00196
       for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
00197
         if (it->second.getClasses()[i].getUcCode() == ucCode) {
           00198
00199
```

```
it->second.getClasses()[i].getUcCode()});
00201
            it->second.getClasses().erase(it->second.getClasses().begin() + i);
00202
            remove = true;
00203
            updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
                               count, 0);
00204
00205
         }
00207
       return remove;
00208 }
00209
00210 // O(log(m))
00211 // m = number of distinct student
00222 void addClassStudent(std::string ucCode, std::string classCode,
00223
                           std::map<std::string, myStudent>::iterator &it,
00224
                           std::stack<alter> &stackAlter) {
00225
       mvUc classe(ucCode, classCode);
00226
       it->second.getClasses().push_back(classe);
00227
       organizerUcStudent(it);
00229
       stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00230
                         "add", ucCode, classCode});
00231 }
00232
00233
00234 // O(m*log(n)
00235 // n = number of UCs
00236 // m = number of classes
00237
00248 void updateCountClasses(std::string ucCode, std::string classCode,
00249
                               std::map<std::string, std::vector<classQtd» &count,</pre>
00250
                               int type) {
00251
00252
       auto it_count = count.find(ucCode);
00253
       if (it_count != count.end()) {
00254
         for (auto &classe : it_count->second) {
           if (classe.classCode == classCode) {
00255
00256
             if (type == 1) {
               classe.qtd++;
00258
              } else {
00259
               classe.qtd--;
00260
              }
00261
            }
00262
         }
       }
00263
00264 }
00265
00266
00267 // O(m*log(n)*k)
00268 // n = number of UCs
00269 // m = number of classes of the student
00270 // k = number of type of classes (T,TP)
00271
00283 bool valideNewClass(std::string ucCode, std::string classCode,
00284
                          std::map<std::string, myStudent>::iterator &it,
00285
                          std::map<std::string, myUc> &classes) {
00286
00287
        // call function to order the
00288
       // classes of the student by
00289
        // int day
00290
        std::map<int, std::set<classInfo> orderClasses =
00291
            orderStudentClass(it, classes);
00292
00293
       std::string value = ucCode + classCode;
00294
00295
        auto it_class = classes.find(value);
00296
00297
        if (it class == classes.end()) {
        std::cout « "Error in
00298
                       "find class"
00299
00300
                    « std::endl;
00301
          return true;
        } else {
00302
00303
          // verify if has a class in
          // the same day and time
for (const auto &class_info : it_class->second.getClassInfoVec()) {
00304
00305
00306
           // get all classes of the
00307
            // day of class
00308
            const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00309
00310
            // and verify if the
            // student has a class in
00311
            // the same time aula -> student classes
00312
00313
            // class_info -> class to add
00314
            for (const auto &aula : classesOfDay) {
00315
              if (aula.type != "T" && class_info.type != "T" &&
00316
00317
                  class_info.startTime >= aula.startTime &&
```

```
00318
                   class_info.startTime < aula.startTime + aula.duration) {</pre>
00319
                 std::cout « "Error: "
00320
                               "Incompatible"
                               " schedules"
00321
                           « std::endl;
00322
00323
                 return true;
00325
            }
00326
00327
          return false;
       }
00328
00329 }
00330
00331 // O(m*log(n)*k)
00332 // n = number of UCs
00333 // m = number of classes
00334 // k = number of type of classes (T,TP)
00344 std::map<int, std::set<classInfo>
00345 orderStudentClass(std::map<std::string, myStudent>::iterator &it,
00346
                          std::map<std::string, myUc> &classes) {
00347
00348
        // map to order the classes
        // by day
00349
        std::map<int, std::set<classInfo> orderClasses;
00350
00351
00352
        // for each class of the
00353
        // student, search in the
00354
        // class tree and add the
00355
        // classInfo in the
00356
        // orderClasses map
00357
        for (const auto &classe : it->second.getClasses()) {
00358
          std::string value = classe.getUcCode() + classe.getClassCode();
00359
00360
          // student one class
          // pointer, verify if the
// class exists in the
00361
00362
00363
           // class tree
00364
          auto it_class = classes.find(value);
00365
00366
           // if the class does not
          // exist, print error
if (it_class == classes.end()) {
00367
00368
            std::cerr « "Error in "
"find class"
00369
00370
00371
                       « std::endl;
00372
          } else {
           // if exists, add the
// classInfo in the
00373
00374
00375
             // orderClasses map
00376
             for (auto &classInfo : it_class->second.getClassInfoVec()) {
00377
              classInfo.code = classe.getUcCode();
00378
               orderClasses[classInfo.dayInt].insert(classInfo);
00379
00380
          }
00381
00382
        return orderClasses;
00384
00393 std::string weekDayString(int day) {
00394
        switch (day) {
00395
        case 2:
        return "Monday";
00396
00397
          break;
00398
        case 3:
        return "Tuesday";
break;
00399
00400
00401
        case 4:
        return "Wednesday";
break;
00402
00403
00404
        case 5:
        return "Thursday";
break;
00405
00406
00407
        case 6:
        return "Friday";
break;
00408
00409
        case 7:
00410
        return "Saturday";
break;
00411
00412
00413
        default:
         return "Error Day";
00414
00415
          break;
00416
        }
00417 }
00418
00419 // O(m) 00420 // m = number of classes of the student
00430 bool verifyUcCode(std::string ucCode,
```

4.13 src/functions/dbStudents.h File Reference

```
#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
#include <liist>
#include <map>
#include <set>
#include <set>
#include <stack>
#include <string>
#include <vector>
#include "../classes/student.h"
```

Functions

- void errorMessage ()
- std::map< std::string, myStudent > selectStudent (const std::string &str, const std::map< std::string, myStudent > &students)

Select students from a map based on a given student code.

• std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)

Filter student information based on specified criteria.

• std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)

Order a vector of students based on the specified criterion.

bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter
 &stackAlter, std::map< std::string, std::vector< classQtd >> &count)

Remove a specific Uc Code from a student's classes.

void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator
 &it, std::stack< alter > &stackAlter)

Add a new class to a student's record.

std::map< int, std::set< classInfo >> orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

Organize a student's classes by day.

 bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

Validate the addition of a new class to a student's schedule.

void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector<
 classQtd >> &count, int type)

Update the class count tree by adding or removing a class.

std::string weekDayString (int day)

Convert a numeric day value to a corresponding day of the week string.

• bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)

Check whether a student is already enrolled in a UC class.

bool compareStudentsCodeAsc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their codes in ascending order.

• bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their codes in descending order.

• bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their names in ascending order.

bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)

Function to compare students based on their names in descending order.

4.13.1 Function Documentation

4.13.1.1 addClassStudent()

```
void addClassStudent (
    std::string ucCode,
    std::string classCode,
    std::map< std::string, myStudent >::iterator & it,
    std::stack< alter > & stackAlter )
```

Add a new class to a student's record.

Parameters

ucCode	Uc Code of the new class.
classCode	Class Code of the new class.
it	Iterator pointing to a student in the map of students.
stackAlter	Stack for recording changes.

This function receives a reference to a student pointer and adds a new class to the student's record. It also organizes the classes, records the change in the stack, and updates the student's class list.

Definition at line 222 of file dbStudents.cpp.

4.13.1.2 compareStudentNameAsc()

Function to compare students based on their names in ascending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the name of the first student is less than that of the second.

Definition at line 35 of file dbStudents.cpp.

4.13.1.3 compareStudentNameDesc()

Function to compare students based on their names in descending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the name of the first student is greater than that of the second.

Definition at line 47 of file dbStudents.cpp.

4.13.1.4 compareStudentsCodeAsc()

Function to compare students based on their codes in ascending order.

Parameters

student1	The first student to compare.	
student2	The second student to compare.	Ì

Returns

Returns true if the code of the first student is less than that of the second.

Definition at line 11 of file dbStudents.cpp.

```
00012
00013 return student1.getStudentCode() < student2.getStudentCode();
00014 }</pre>
```

4.13.1.5 compareStudentsCodeDesc()

Function to compare students based on their codes in descending order.

Parameters

student1	The first student to compare.
student2	The second student to compare.

Returns

Returns true if the code of the first student is greater than that of the second.

Definition at line 23 of file dbStudents.cpp.

```
00024 {
00025 return student1.getStudentCode() > student2.getStudentCode();
00026 }
```

4.13.1.6 errorMessage()

```
void errorMessage ( )
```

Definition at line 4 of file errorMsgs.cpp.

```
00004

00005 std::cout « "ERROR: Invalid choice." « std::endl;

00006 exit(0);

00007 }
```

4.13.1.7 filterInfoStudent()

```
\begin{tabular}{lll} {\tt std::vector}<& {\tt myStudent}>& {\tt filterInfoStudent}& (\\ & & {\tt int}~n,\\ & & {\tt std::string}~str,\\ & & {\tt const}~std::vector}<& {\tt myStudent}>& & students \end{tabular}
```

Filter student information based on specified criteria.

Parameters

n	Filter criterion: 1 for Uc Code, 2 for Class Code.
str	Search string.
students	Vector of students to filter.

Returns

Vector of students matching the criteria.

Definition at line 63 of file dbStudents.cpp.

```
00065
        std::vector<myStudent> filterStudents;
00066
        switch (n) {
        case 1:
   // Filter by Uc Code
00067
00068
00069
          for (const auto &student : students) {
00070
            for (const auto &uc : student.getClasses()) {
00071
              if (uc.getUcCode() == str) {
00072
                 filterStudents.push_back(student);
00073
                 break;
00074
              }
00075
            }
00076
00077
          break;
00078
        case 2:
00079
          // Filter by Class Code
           for (const auto &student : students) {
08000
            for (const auto &uc: student.getClasses()) {
   for (const auto &classInfo : uc.getClassInfoVec()) {
00081
00082
00083
                 if (classInfo.code == str) {
00084
                  filterStudents.push_back(student);
00085
                   break; // No need to check other class codes for this student
00086
                 }
00087
              }
00088
            }
00089
           break;
00090
00091
        default:
00092
          errorMessage();
00093
          break;
00094
00095
        return filterStudents;
00096 }
```

4.13.1.8 orderInfoStudent()

Order a vector of students based on the specified criterion.

Parameters

n	Ordering criterion: 1 for ascending Student Code, 2 for descending Student Code, 3 for ascending Student Name, 4 for descending Student Name.	
students	Vector of students to be ordered.	

Returns

Ordered vector of students based on the specified criterion.

Definition at line 108 of file dbStudents.cpp.

```
00109
00110
        switch (n) {
00111
       case 1:
         // Order by Student Code Asc
00112
00113
         std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00114
         break;
00115
         // Order by Student Code Desc
00116
00117
         std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00118
         break;
00119
       case 3:
00120
        // Order by Student Name Asc
00121
          std::sort(students.begin(), students.end(), compareStudentNameAsc);
00122
         break;
00123
       case 4:
         // Order by Student Name Desc
00124
00125
         std::sort(students.begin(), students.end(), compareStudentNameDesc);
00126
         break;
```

```
00127    default:
00128         errorMessage();
00129         break;
00130    }
00131    
00132    return students;
00133 }
```

4.13.1.9 orderStudentClass()

Organize a student's classes by day.

Parameters

it	Iterator pointing to a student in the map of students.
classes	Map of classes to organize.

Returns

Map of classes organized by day.

This function organizes a student's classes by day and returns a map where classes are grouped by their respective days.

Definition at line 345 of file dbStudents.cpp.

```
00347
00348
        // map to order the classes
00349
        // by day
00350
        std::map<int, std::set<classInfo» orderClasses;
00351
00352
        // for each class of the
        // student, search in the
// class tree and add the
00353
00354
00355
        // classInfo in the
00356
        // orderClasses map
00357
        for (const auto &classe : it->second.getClasses()) {
00358
          std::string value = classe.getUcCode() + classe.getClassCode();
00359
          // student one class
00360
          // pointer, verify if the
// class exists in the
00361
00362
00363
          // class tree
00364
          auto it_class = classes.find(value);
00365
00366
          // if the class does not
00367
          // exist, print error
          if (it_class == classes.end()) {
00368
           std::cerr « "Error in "
"find class"
00369
00370
00371
                       « std::endl;
00372
          } else {
          // if exists, add the
// classInfo in the
00373
00374
00375
             // orderClasses map
00376
            for (auto &classInfo : it_class->second.getClassInfoVec()) {
00377
              classInfo.code = classe.getUcCode();
00378
               orderClasses[classInfo.dayInt].insert(classInfo);
00379
00380
         }
00381
00382
        return orderClasses;
00383 }
```

4.13.1.10 removeUcStudent()

Remove a specific Uc Code from a student's classes.

Parameters

ucCode	Uc Code to be removed.
it	Iterator pointing to a student in the map of students.
stackAlter	Stack for recording changes.
count	Map for tracking class counts.

Returns

Returns true if the Uc Code was successfully removed; otherwise, returns false.

This function removes a specific Uc Code from a student's classes. It records the change in the stack for later reference and updates the class count in the "count" map.

Definition at line 190 of file dbStudents.cpp.

```
00194
00195
          bool remove = false;
          for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
   if (it->second.getClasses()[i].getUcCode() == ucCode) {
     stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00196
00197
00198
00199
                                      "remove", ucCode,
00200
                                      it->second.getClasses()[i].getUcCode()});
00201
               it->second.getClasses().erase(it->second.getClasses().begin() + i);
00202
                remove = true;
00203
               updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
00204
                                        count, 0);
00205
            }
00206
00207
          return remove;
00208 }
```

4.13.1.11 selectStudent()

```
std::map< std::string, myStudent > selectStudent ( const std::string & str, const std::map< std::string, myStudent > & students)
```

Select students from a map based on a given student code.

Parameters

str	Student code to search for.
students	Map of students to select from.

Returns

Map of selected students with matching student codes.

Definition at line 145 of file dbStudents.cpp.

```
00147
       std::map<std::string, myStudent> selectedStudents;
00148
00149
       for (auto &studentPair : students) {
00150
         const myStudent &mystudent = studentPair.second;
         if (str == mystudent.getStudentCode()) {
00151
            selectedStudents[studentPair.first] = mystudent;
00152
00153
00154
00156
       return selectedStudents;
00157 }
```

4.13.1.12 updateCountClasses()

Update the class count tree by adding or removing a class.

Parameters

ucCode	Uc Code associated with the class.
classCode	Class Code to be added or removed.
count	Map for tracking class counts.
type	1 for adding a class, 0 for removing a class.

This function updates the class count in the "count" map by either adding or removing a class.

Definition at line 248 of file dbStudents.cpp.

```
00251
        auto it_count = count.find(ucCode);
if (it_count != count.end()) {
00252
00253
00254
         for (auto &classe : it_count->second) {
00255
            if (classe.classCode == classCode) {
              if (type == 1) {
00257
                classe.qtd++;
00258
              } else {
00259
                classe.qtd--;
00260
00261
             }
00262
          }
00264 }
```

4.13.1.13 valideNewClass()

Validate the addition of a new class to a student's schedule.

Parameters

ucCode	Uc Code of the class to be added.
classCode	Class Code of the class to be added.
it	Iterator pointing to a student in the map of students.
classes	Map of classes to validate against.

Returns

Returns true if the addition of the class is valid, or false if it conflicts with the student's existing schedule.

This function validates whether adding a new class to a student's schedule is compatible with their existing classes. It checks for schedule conflicts and ensures the class exists in the "classes" map.

Definition at line 283 of file dbStudents.cpp.

```
00286
00287
         // call function to order the
        // classes of the student by
// int day
00288
00289
00290
        std::map<int, std::set<classInfo> orderClasses =
00291
            orderStudentClass(it, classes);
00292
00293
        std::string value = ucCode + classCode;
00294
00295
        auto it class = classes.find(value);
00296
00297
        if (it_class == classes.end()) {
00298
          std::cout « "Error in "
                        "find class"
00299
00300
                     « std::endl;
00301
          return true;
00302
        } else {
00303
          // verify if has a class in
00304
          // the same day and time
00305
          for (const auto &class_info : it_class->second.getClassInfoVec()) {
            // get all classes of the
// day of class
00306
00307
00308
            const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00309
00310
             // and verify if the
            // student has a class in
// the same time aula -> student classes
// class_info -> class to add
00311
00312
00313
00314
            for (const auto &aula : classesOfDay) {
00315
00316
              if (aula.type != "T" && class_info.type != "T" &&
00317
                   class_info.startTime >= aula.startTime &&
00318
                   class_info.startTime < aula.startTime + aula.duration) {</pre>
                 std::cout « "Error: '
00319
                               "Incompatible"
00320
                               " schedules"
00321
00322
                            « std::endl;
00323
                 return true;
00324
              }
00325
            }
00326
00327
          return false;
00328
        }
00329 }
```

4.13.1.14 verifyUcCode()

Check whether a student is already enrolled in a UC class.

Parameters

ucCode	Uc Code to check for enrollment.
it	Iterator pointing to a student in the map of students.

Returns

Returns true if the student is already enrolled in a class with the specified Uc Code; otherwise, returns false.

This function checks whether a student is already enrolled in a class with the specified Uc Code and returns true if a match is found, indicating a potential problem.

Definition at line 430 of file dbStudents.cpp.

4.13.1.15 weekDayString()

Convert a numeric day value to a corresponding day of the week string.

Parameters

```
day Numeric representation of a day (e.g., 2 for Monday).
```

Returns

Corresponding day of the week string.

This function converts a numeric day value to a string representation of the corresponding day of the week.

Definition at line 393 of file dbStudents.cpp.

```
00393
00394
        switch (day) {
00395
       case 2:
        return "Monday";
break;
00396
00397
00398
       case 3:
       return "Tuesday";
break;
00399
00400
00401
       case 4:
        return "Wednesday";
break;
00402
00403
00404
       case 5:
         return "Thursday";
break;
00405
00406
00407
       case 6:
        return "Friday";
break;
00408
00409
00410
       case 7:
00411
         return "Saturday";
00412
          break;
```

```
00413 default:

00414 return "Error Day";

00415 break;

00416 }
```

4.14 dbStudents.h

Go to the documentation of this file.

```
00001 #ifndef DBSTUDENTS_H
00002 #define DBSTUDENTS_H
00003
00004 #include <algorithm>
00005 #include <climits>
00006 #include <fstream>
00007 #include <iostream>
00008 #include <list>
00009 #include <map>
00010 #include <set>
00011 #include <stack>
00012 #include <string>
00013 #include <vector>
00014
00015 #include "../classes/student.h"
00016
00017 void errorMessage();
00018
00019 std::map<std::string, myStudent>
00020 selectStudent(const std::string &str,
00021
                    const std::map<std::string, myStudent> &students);
00022 std::vector<myStudent>
00023 filterInfoStudent(int n, std::string str,
                        const std::vector<myStudent> &students);
00025 std::vector<myStudent> orderInfoStudent(int n,
00026
                                               std::vector<myStudent> &students);
00027
00028 std::vector<myStudent>
00029 filterInfoStudent(int n, std::string str,
                        const std::vector<myStudent> &students);
00031 std::vector<myStudent> orderInfoStudent(int n,
00032
                                               std::vector<myStudent> &students);
00033
00034 bool removeUcStudent(std::string ucCode,
                           std::map<std::string, myStudent>::iterator &it,
00035
00036
                           std::stack<alter> &stackAlter,
00037
                           std::map<std::string, std::vector<classQtd» &count);</pre>
00038
00039 void addClassStudent(std::string ucCode, std::string classCode,
00040
                          std::map<std::string, myStudent>::iterator &it,
00041
                           std::stack<alter> &stackAlter);
00042
00043 std::map<int, std::set<classInfo>
00044 orderStudentClass(std::map<std::string, myStudent>::iterator &it,
00045
                        std::map<std::string, myUc> &classes);
00046 bool valideNewClass(std::string ucCode, std::string classCode,
                          std::map<std::string, myStudent>::iterator &it,
00047
00048
                          std::map<std::string, myUc> &classes);
00050 void updateCountClasses(std::string ucCode, std::string classCode,
00051
                              std::map<std::string, std::vector<classQtd» &count,
00052
                              int type);
00053
00054 std::string weekDayString(int day);
00055 bool verifyUcCode(std::string ucCode,
                        std::map<std::string, myStudent>::iterator &it);
00056
00057
00058 bool compareStudentsCodeAsc(const myStudent &student1,
00059
                                   const myStudent &student2);
00060 bool compareStudentsCodeDesc(const myStudent &student1,
00061
                                   const myStudent &student2);
00062 bool compareStudentNameAsc(const myStudent &student1,
                                  const myStudent &student2);
00064 bool compareStudentNameDesc(const myStudent &student1,
00065
                                   const myStudent &student2);
00066
00067 #endif
```

4.15 src/functions/dbUcs.cpp File Reference

```
#include "dbUcs.h"
```

Functions

• bool compareClassesCodeAsc (const myUc &uc1, const myUc &uc2)

Compare two myUc objects by their class codes in ascending order.

bool compareUcsCodeASC (const myUc &uc1, const myUc &uc2)

Compare two myUc objects by their UC codes in ascending order.

bool compareClassesCodeDesc (const myUc &uc1, const myUc &uc2)

Compare two myUc objects by their UC codes in descending order.

bool compareUcsCodeDesc (const myUc &uc1, const myUc &uc2)

Compare two myUc objects by their UC codes in descending order.

- std::vector < myUc > filterInfoUc (int n, std::string str, std::vector < myUc > &ucs)

Filters UC information.

std::vector< myUc > orderInfoUc (int n, std::vector< myUc > &ucs)

Sorts UC information.

• std::vector< myUc > selectUc (const std::string &str, const std::map< std::string, myUc > &classes)

Selects UCs on the provided code.

4.15.1 Function Documentation

4.15.1.1 compareClassesCodeAsc()

Compare two myUc objects by their class codes in ascending order.

Parameters

uc1	The first myUc object to compare.
uc2	The second myUc object to compare.

Returns

True if 'uc1' class code is less than 'uc2' class code, otherwise false.

```
Definition at line 11 of file dbUcs.cpp.
00011
00012    return ucl.getClassCode() < uc2.getClassCode();
00013 }</pre>
```

4.15.1.2 compareClassesCodeDesc()

Compare two myUc objects by their UC codes in descending order.

Parameters

uc1	The first myUc object to compare.
uc2	The second myUc object to compare.

Returns

True if 'uc1' UC code is greater than 'uc2' UC code, otherwise false.

```
Definition at line 33 of file dbUcs.cpp.
```

4.15.1.3 compareUcsCodeASC()

Compare two myUc objects by their UC codes in ascending order.

Parameters

uc1	The first myUc object to compare.
uc2	The second myUc object to compare.

Returns

True if 'uc1' UC code is less than 'uc2' UC code, otherwise false.

Definition at line 22 of file dbUcs.cpp.

4.15.1.4 compareUcsCodeDesc()

Compare two myUc objects by their UC codes in descending order.

Parameters

uc1	The first myUc object to compare.
uc2	The second myUc object to compare.

Returns

True if 'uc1' UC code is greater than 'uc2' UC code, otherwise false.

Definition at line 44 of file dbUcs.cpp.

```
00044 (00045 return ucl.getUcCode() > uc2.getUcCode(); (00046 }
```

4.15.1.5 filterInfoUc()

```
std::vector< myUc > filterInfoUc (
    int n,
    std::string str,
    std::vector< myUc > & ucs )
```

Filters UC information.

Parameters

n	Number representing the filter.
str	Search string.
ucs	Vector of UCs to be filtered.

Returns

std::vector<myUc> Vector of filtered UCs.

Definition at line 58 of file dbUcs.cpp.

```
00059
         std::vector<myUc> filterUc;
00060
         switch (n) {
         case 1:
  // Filter by Uc Code
  for (const auto &uc : ucs) {
    if (uc.getUcCode() == str) {
00061
00062
00063
00064
00065
                filterUc.push_back(uc);
              }
00066
00067
00068
           break;
00069
         case 2:
         // Filter by Class Code
for (const auto &uc: ucs) {
00070
00071
           for (const auto &classInfo : uc.getClassInfoVec()) {
   if (classInfo.code == str) {
00072
00073
00074
                  filterUc.push_back(uc);
00075
                   break;
00076
00077
             }
00078
00079
           break;
         default:
08000
         errorMessage();
00081
00082
           break;
00083
00084
         return filterUc;
00085 }
```

4.15.1.6 orderInfoUc()

Sorts UC information.

Parameters

n	Number representing the sorting criterion.
ucs	Vector of UCs to be sorted.

Returns

std::vector<myUc> Vector of sorted UCs.

Definition at line 95 of file dbUcs.cpp.

```
00095
00096
00097
        switch (n) {
00098
        case 1:
00099
        // Order by Uc Code Asc
00100
          std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00101
          break;
00102
        case 2:
        // Order by Uc Code Desc
std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00103
00104
00105
          break;
00106
        case 3:
00107
         // Order by Class Code Asc
00108
          std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00109
          break;
00110
        case 4:
         // Order by Class Code Desc
00112
          std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00113
          break;
00114
        default:
        errorMessage();
00115
00116
          break;
00117
00118
        return ucs;
00119 }
```

4.15.1.7 selectUc()

Selects UCs on the provided code.

Parameters

str	Code of the UC to be selected.
ucs	Vector of UCs to be filtered.

Returns

std::vector<myUc> Vector of selected UCs.

Definition at line 130 of file dbUcs.cpp.

```
00131
00132
        std::vector<myUc> selectedUcs;
00133
       for (const auto &pair : classes) {
00134
00135
         auto ucObj = pair.second;
00136
00137
         if (ucObj.getUcCode() == str) {
00138
            selectedUcs.push_back(ucObj);
00139
00140
       }
00141
       return selectedUcs;
00142 }
```

4.16 dbUcs.cpp 55

4.16 dbUcs.cpp

```
Go to the documentation of this file.
```

```
00001 #include "dbUcs.h'
00002
00003 // 0(1)
00011 bool compareClassesCodeAsc(const myUc &uc1, const myUc &uc2) {
00012
       return ucl.getClassCode() < uc2.getClassCode();</pre>
00013 }
00014
00015 // 0(1)
00022 bool compareUcsCodeASC(const myUc &ucl, const myUc &uc2) {
       return ucl.getUcCode() < uc2.getUcCode();</pre>
00024 }
00025
00026 // 0(1)
00033 bool compareClassesCodeDesc(const myUc &uc1, const myUc &uc2) {
00034 return ucl.getClassCode() > uc2.getClassCode();
00035 }
00036
00037 // 0(1)
00044 bool compareUcsCodeDesc(const myUc &uc1, const myUc &uc2) {
00045
        return uc1.getUcCode() > uc2.getUcCode();
00046 }
00048 // Average: O(n)
00049 // Best: O(1)
00050 // n = number of lines in the file
00058 std::vector<myUc> filterInfoUc(int n, std::string str, std::vector<myUc> &ucs) {
00059
       std::vector<myUc> filterUc;
00060
        switch (n) {
00061
        case 1:
00062
         // Filter by Uc Code
00063
          for (const auto &uc : ucs) {
           if (uc.getUcCode() == str)
00064
             filterUc.push_back(uc);
00065
00066
            }
00067
00068
          break;
00069
        case 2:
00070
         // Filter by Class Code
00071
          for (const auto &uc : ucs) {
00072
           for (const auto &classInfo : uc.getClassInfoVec()) {
             if (classInfo.code == str) {
00074
               filterUc.push_back(uc);
00075
00076
00077
           }
00078
00079
         break;
        default:
00081
         errorMessage();
00082
         break;
00083
00084
       return filterUc;
00085 }
00086
00087 // O(n*log(n))
00088 // n = number of lines in the file
00095 std::vector<myUc> orderInfoUc(int n, std::vector<myUc> &ucs) {
00096
00097
        switch (n) {
       case 1:
00099
        // Order by Uc Code Asc
00100
         std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00101
         break;
00102
        case 2:
        // Order by Uc Code Desc
00103
00104
         std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00105
         break;
00106
        case 3:
        // Order by Class Code Asc
00107
00108
         std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00109
         break:
00110
        case 4:
00111
         // Order by Class Code Desc
00112
         std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00113
         break;
00114
       default:
         errorMessage();
00115
00116
         break;
00118
       return ucs;
00119 }
00120
```

```
00121 // O(n) 00122 // n = number of lines in the file
00130 std::vector<myUc> selectUc(const std::string &str,
                                 const std::map<std::string, myUc> &classes) {
00131
        std::vector<myUc> selectedUcs;
00132
00133
00134 for (const auto &pair : classes) {
00135
          auto ucObj = pair.second;
00136
         if (ucObj.getUcCode() == str) {
00137
00138
            selectedUcs.push_back(ucObj);
00139
00140 }
00141 return selectedUcs;
00142 }
```

4.17 src/functions/dbUcs.h File Reference

```
#include <algorithm>
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include "../classes/uc.h"
```

Functions

- void errorMessage ()
- bool compareClassesCode (const myUc &uc1, const myUc &uc2)
- bool compareUcsCode (const myUc &uc1, const myUc &uc2)
- std::vector< myUc > selectUc (const std::string &str, const std::map< std::string, myUc > &classes)

 Selects UCs on the provided code.
- std::vector< myUc > filterInfoUc (int n, std::string str, std::vector< myUc > &ucs)
 Filters UC information.
- std::vector< $\mbox{myUc} > \mbox{orderInfoUc}$ (int n, std::vector< $\mbox{myUc} > \mbox{\&ucs})$

Sorts UC information.

4.17.1 Function Documentation

4.17.1.1 compareClassesCode()

4.17.1.2 compareUcsCode()

4.17.1.3 errorMessage()

```
void errorMessage ( )

Definition at line 4 of file errorMsgs.cpp.

00004
00005   std::cout « "ERROR: Invalid choice." « std::endl;
00006   exit(0);
00007 }
```

4.17.1.4 filterInfoUc()

Filters UC information.

Parameters

n	Number representing the filter.
str	Search string.
ucs	Vector of UCs to be filtered.

Returns

std::vector<myUc> Vector of filtered UCs.

Definition at line 58 of file dbUcs.cpp.

```
00058
00059
         std::vector<myUc> filterUc;
00060
         switch (n) {
00061
        case 1:
00062
          // Filter by Uc Code
00063
           for (const auto &uc : ucs) {
00064
             if (uc.getUcCode() == str) {
               filterUc.push_back(uc);
00065
00066
             }
00067
00068
          break;
00069
        case 2:
         // Filter by Class Code
for (const auto &uc : ucs) {
00070
00071
            for (const auto &classInfo : uc.getClassInfoVec()) {
   if (classInfo.code == str) {
00072
00073
00074
                 filterUc.push_back(uc);
00075
                 break;
00076
               }
00077
00078
            }
00079
           break;
08000
        default:
         errorMessage();
break;
00081
00082
        }
00083
00084
        return filterUc;
00085 }
```

4.17.1.5 orderInfoUc()

Sorts UC information.

Parameters

n	Number representing the sorting criterion.
ucs	Vector of UCs to be sorted.

Returns

std::vector<myUc> Vector of sorted UCs.

Definition at line 95 of file dbUcs.cpp.

```
00095
00096
00097
        switch (n) {
00098
        case 1:
00099
        // Order by Uc Code Asc
00100
          std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00101
          break;
00102
        case 2:
        // Order by Uc Code Desc
std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00103
00104
00105
          break;
00106
        case 3:
00107
         // Order by Class Code Asc
00108
          std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00109
          break;
00110
        case 4:
         // Order by Class Code Desc
00112
          std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00113
          break;
00114
        default:
        errorMessage();
00115
00116
          break;
00117
00118
        return ucs;
00119 }
```

4.17.1.6 selectUc()

Selects UCs on the provided code.

Parameters

str	Code of the UC to be selected.
ucs	Vector of UCs to be filtered.

Returns

std::vector<myUc> Vector of selected UCs.

Definition at line 130 of file dbUcs.cpp.

```
00131
00132
        std::vector<myUc> selectedUcs;
00133
       for (const auto &pair : classes) {
00134
00135
         auto ucObj = pair.second;
00136
00137
         if (ucObj.getUcCode() == str) {
00138
            selectedUcs.push_back(ucObj);
00139
00140
       }
00141
       return selectedUcs;
00142 }
```

4.18 dbUcs.h 59

4.18 dbUcs.h

Go to the documentation of this file.

```
00001 #ifndef DBUCS_H
00002 #define DBUCS H
00003
00004 #include <algorithm>
00005 #include <iostream>
00006 #include <map>
00007 #include <string>
00008 #include <vector>
00009
00010 #include "../classes/uc.h"
00012 void errorMessage();
00013
00014 bool compareClassesCode(const myUc &uc1, const myUc &uc2);
00015 bool compareUcsCode (const myUc &ucl, const myUc &uc2);
00016
00017 std::vector<myUc> selectUc(const std::string &str,
00018
                                       const std::map<std::string, myUc> &classes);
00019 std::vector<myUc> filterInfoUc(int n, std::string str, std::vector<myUc> &ucs);
00020 std::vector<myUc> orderInfoUc(int n, std::vector<myUc> &ucs);
00021
00022 #endif
```

4.19 src/inputoutput/keepAllChanges.cpp File Reference

```
#include "keepAllChanges.h"
#include <ctime>
```

Functions

bool orderVector (const std::string &str1, const std::string &str2)

Compare two strings in descending order.

• std::string getSysdate ()

Get the system date.

void makeBackup ()

Creates a backup of the "students_classes.csv" file with the lastest archive modified. The backup file is named with the current system date.

void keepAllChanges (std::map< std::string, myStudent > &students, std::stack< alter > &stackAlter)

Saves all changes made to the student tree in the "students_classes.csv" file.

· void listAllBackups ()

List all backup files.

· bool printAllBackups ()

Prints all backup file names stored in the public vector backups.

void printChanges (int cdBkp)

Print the changes from backup files.

void backupFile (int cdBkp)

Backup a specific file and remove related changes.

Variables

std::vector< std::string > backups

4.19.1 Function Documentation

4.19.1.1 backupFile()

Backup a specific file and remove related changes.

This function backs up a specified file from "schedule/backup" to "schedule/students_classes.csv" and removes related change files in the "schedule/alter" and "schedule/backup" directories.

Parameters

cdBkp The index of the backup file to restore.

Definition at line 204 of file keepAllChanges.cpp.

```
00205
00206
        std::string path = "schedule/backup/" + backups[cdBkp];
00207
00208
        std::ifstream backup(path, std::ios::binary);
00209
00210
        if (!backup) {
00211
          std::cerr « "Error opening file" « std::endl;
00212
00213
00214
        std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00215
00216
        if (!file) {
00217
          std::cerr « "Error opening file" « std::endl;
00218
00219
00220
        file « backup.rdbuf();
00221
        file.close();
00222
        backup.close();
00223
00224
        unsigned size = cdBkp;
        for (unsigned i = 0; i <= size; i++) {
   if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {</pre>
00225
00226
00227
             try {
              std::filesystem::remove("schedule/alter/" + backups[i]);
std::filesystem::remove("schedule/backup/" + backups[i]);
00228
00229
00230
             } catch (const std::filesystem::filesystem_error &e) {
00231
               std::cerr « "Error to remove the file" « e.what() « std::endl;
00232
00233
          } else {
             std::cout « "The file of changes not exist" « std::endl;
00234
00235
00236
        }
00237 }
```

4.19.1.2 getSysdate()

```
std::string getSysdate ( )
```

Get the system date.

Returns

A string with the system date.

Definition at line 22 of file keepAllChanges.cpp.

00022 00023

4.19.1.3 keepAllChanges()

Saves all changes made to the student tree in the "students_classes.csv" file.

Parameters

students	Reference to the map containing student data.
stackAlter	Reference to a stack containing alteration records.

Definition at line 69 of file keepAllChanges.cpp.

```
00070
00071
        makeBackup();
00072
        std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00073
                                   ".csv",
00074
                               std::ios::app);
        if (!alter.is_open()) {
   std::cerr « "Error opening file" « std::endl;
00075
00076
00077
00078
        while (!stackAlter.empty()) {
   alter « "The student: " « stackAlter.top().studentCode « " - "
08000
                 « stackAlter.top().studentName « " " « stackAlter.top().type
« " UC: " « stackAlter.top().ucCode
« " Class: " « stackAlter.top().classCode « std::endl;
00081
00082
00083
00084
          stackAlter.pop();
00085
00086
00087
        std::ofstream file("schedule/students_classes.csv");
00088
        if (!file.is_open()) {
   std::cerr « "Error opening file" « std::endl;
00089
00090
00091
00092
00093
00094
        file « "StudentCode, StudentName, UcCode, ClassCode" « std::endl;
00095
00096
        // Write the tree in the file
00097
        for (auto it = students.begin(); it != students.end(); it++) {
00098
               (auto classe : it->second.getClasses()) {
            00099
00100
00101
                  « std::endl;
00102
00103
        }
00104 }
```

4.19.1.4 listAllBackups()

```
void listAllBackups ( )
```

List all backup files.

If no backup files exist, this function searches for and populates the 'backups' vector with filenames from the "schedule/backup" directory.

Definition at line 115 of file keepAllChanges.cpp.

4.19.1.5 makeBackup()

```
void makeBackup ( )
```

Creates a backup of the "students_classes.csv" file with the lastest archive modified. The backup file is named with the current system date.

Definition at line 38 of file keepAllChanges.cpp.

```
00038
        std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00040
00041
        if (!file) {
00042
         std::cerr « "Error opening file" « std::endl;
00043
00044
00045
        std::string dateString = getSysdate();
00046
00047
        std::string backupName =
00048
            "schedule/backup/students_classes-" + dateString + ".csv";
00049
       std::ofstream backup(backupName, std::ios::binary);
00050
       if (!backup) {
00051
00052
        std::cerr « "Error to create a backup file" « std::endl;
00053
         return:
00054
00055
00056
       backup « file.rdbuf();
00057
        file.close();
00058 backup.close();
00059 }
```

4.19.1.6 orderVector()

```
bool orderVector (  {\it const std::string \& str1,} \\ {\it const std::string \& str2} )
```

Compare two strings in descending order.

Parameters

str1	The first string to compare.
str2	The second string to compare.

Returns

True if 'str1' is greater than 'str2', otherwise false.

```
Definition at line 13 of file keepAllChanges.cpp.
```

```
00013
00014 return str1 > str2;
00015 }
```

{

4.19.1.7 printAllBackups()

```
bool printAllBackups ( )
```

Prints all backup file names stored in the public vector backups.

Definition at line 132 of file keepAllChanges.cpp.

```
00132
            if (backups.size() != 0) {
   std::cout « "Backups: " « std::endl;
   for (unsigned i = 0; i < backups.size(); i++) {
     std::cout « i « " - " « backups.at(i) « std::endl;</pre>
00133
00134
00135
00136
00137
               return true;
00138
00139
            } else {
            std::cout « "No backups" « std::endl;
00140
00141
               return false;
            }
00142
00143 }
```

4.19.1.8 printChanges()

```
void printChanges (
          int cdBkp )
```

Print the changes from backup files.

This function prints the content of backup files located in the "schedule/alter" directory, up to the specified 'cdBkp' index, to the standard output.

Parameters

cdBkp The index of the backup files to print.

Definition at line 158 of file keepAllChanges.cpp.

```
00158
00159
        unsigned size = cdBkp;
00160
        for (unsigned i = 0; i <= size; i++) {</pre>
          std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);
00161
00162
00163
         std::cerr « "Error opening file" « std::endl;
}
          if (!file) {
00164
00165
00166
00167
          std::string line;
00168
00169
          while (std::getline(file, line)) {
00170
            std::cout « " " « line « std::endl;
00171
00172
          file.close():
00173
        }
00174
00175
        // // Write the tree in the file
00176
        // for (auto it = students.begin(); it != students.end(); it++) {
            // for (auto classe : it->second.getClasses()) {
// // std::cout« it->second.getCode() « "," « it->second.getName()
00177
00178
00179
             00180
00181
00182
             std::endl;
             // file « it->second.getStudentCode() « "," «
// it->second.getStudentName() « ","
00183
00184
                       « classe.getUcCode() « "," « classe.getClassCode() «
00185
             11
00186
                       std::endl;
00187
00188
00189 }
```

4.19.2 Variable Documentation

4.19.2.1 backups

std::vector<std::string> backups

Definition at line 4 of file keepAllChanges.cpp.

4.20 keepAllChanges.cpp

Go to the documentation of this file.

```
00001 #include "keepAllChanges.h'
00002 #include <ctime>
00004 std::vector<std::string> backups;
00005
00006 //0(1)
00013 bool orderVector(const std::string &str1, const std::string &str2) {
00014
        return str1 > str2:
00015 }
00016
00017 //0(1)
00022 std::string getSysdate() {
00023
00024
         std::time t date = std::time(0);
00025
        std::tm *now = std::localtime(&date);
        return std::to_string(now->tm_year + 1900) + "-" +
    std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
    "-" + std::to_string(now->tm_hour) + ":" +
    std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
00027
00028
00029
00030
00031 }
00032
00033 //0(1)
00038 void makeBackup() {
        std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00039
00040
00041
00042
           std::cerr « "Error opening file" « std::endl;
00043
00044
00045
         std::string dateString = getSysdate();
00046
00047
         std::string backupName =
00048
               "schedule/backup/students_classes-" + dateString + ".csv";
00049
         std::ofstream backup(backupName, std::ios::binary);
00050
00051
         if (!backup) {
         std::cerr « "Error to create a backup file" « std::endl;
00052
00053
           return;
00054
00055
00056
         backup « file.rdbuf();
00057
         file.close();
00058
         backup.close();
00059 }
00060
00061 // O(m)
00062 // m = number of changes
00069 void keepAllChanges(std::map<std::string, myStudent> &students,
00070
                              std::stack<alter> &stackAlter) {
00071
         makeBackup();
00072
         std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
                                       ".csv",
00074
                                 std::ios::app);
         if (!alter.is_open()) {
   std::cerr « "Error opening file" « std::endl;
00075
00076
00077
00078
         while (!stackAlter.empty()) {
   alter « "The student: " « stackAlter.top().studentCode « " - "
00079
08000
                  « stackAlter.top().studentName « " " « stackAlter.top().type
« " UC: " « stackAlter.top().ucCode
« " Class: " « stackAlter.top().classCode « std::endl;
00081
00082
00083
00084
           stackAlter.pop();
00085
00086
```

```
std::ofstream file("schedule/students_classes.csv");
00088
00089
         if (!file.is_open()) {
00090
          std::cerr « "Error opening file" « std::endl;
00091
00092
00093
         // Header
00094
        file « "StudentCode, StudentName, UcCode, ClassCode" « std::endl;
00095
00096
         \ensuremath{//} Write the tree in the file
        for (auto it = students.begin(); it != students.end(); it++) {
00097
          00098
00099
00100
00101
                   « std::endl;
00102
        }
00103
00104 }
00106 // O(m)
00107 // m = number of backups files
00108 \!\!\!// Ideally, use a script to maintain a maximum of 10
00115 void listAllBackups() {
        if (backups.size() == 0) {
  std::string way = "schedule/backup";
  for (const auto &in : std::filesystem::directory_iterator(way)) {
00116
00117
00118
00119
            if (std::filesystem::is_regular_file(in)) {
00120
               backups.push_back(in.path().filename().string());
00121
            }
00122
00123
          std::sort(backups.begin(), backups.end(), orderVector);
00124
        }
00125 }
00126
00127 // O(m)
00128 // m = number of backups files
00132 bool printAllBackups() {
00133    if (backups.size() != 0)
          std::cout « "Backups: " « std::endl;
for (unsigned i = 0; i < backups.size(); i++) {
   std::cout « i « " - " « backups.at(i) « std::endl;</pre>
00134
00135
00136
          }
00137
00138
          return true:
00139
        } else {
         std::cout « "No backups" « std::endl;
00140
00141
           return false;
00142
00143 }
00144
00145 // O(m)
00146 // m = number of backups files
00147 // Best case: O(1) when the user select the newer backup
00148 // Worst case: O(m) when the user select the older backup
00158 void printChanges(int cdBkp) {
00159
        unsigned size = cdBkp;
for (unsigned i = 0; i <= size; i++) {
  std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);</pre>
00160
00162
           if (!file) {
00163
            std::cerr « "Error opening file" « std::endl;
00164
          }
00165
00166
00167
          std::string line;
00168
00169
           while (std::getline(file, line)) {
00170
            std::cout « " " « line « std::endl;
00171
00172
          file.close();
00173
00174
00175
         // // Write the tree in the file
00176
        // for (auto it = students.begin(); it != students.end(); it++) {
             // for (auto classe : it->second.getClasses()) {
// // std::cout« it->second.getCode() « "," « it->second.getName()
00177
00178
00179
              // ","
// // « classe.getUcCode() « "," « classe.getClassCode() «
00180
00181
00182
               std::endl;
00183
              11
                   file « it->second.getStudentCode() « "," «
              11
00184
                   it->second.getStudentName() « ", '
                         « classe.getUcCode() « "," « classe.getClassCode() «
00185
              //
00186
              //
                         std::endl;
00187
00188
00189 }
00190
00191 // O(m)
```

```
00192 // m = number of backups files
00193 // Best case: O(1) when the user select the newer backup
00194 // Worst case: O(m) when the user select the older backup
00204 void backupFile(int cdBkp) {
00205
00206
        std::string path = "schedule/backup/" + backups[cdBkp];
00208
        std::ifstream backup(path, std::ios::binary);
00209
00210
        if (!backup) {
          std::cerr « "Error opening file" « std::endl;
00211
00212
00213
00214
        std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00215
        if (!file) {
00216
          std::cerr « "Error opening file" « std::endl;
00217
00218
00219
00220
        file « backup.rdbuf();
00221
        file.close();
00222
        backup.close();
00223
        unsigned size = cdBkp;
for (unsigned i = 0; i <= size; i++) {
  if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {</pre>
00224
00225
00227
              std::filesystem::remove("schedule/alter/" + backups[i]);
std::filesystem::remove("schedule/backup/" + backups[i]);
00228
00229
00230
             } catch (const std::filesystem::filesystem_error &e) {
               std::cerr « "Error to remove the file" « e.what() « std::endl;
00231
00232
00233
         } else {
00234
             std::cout « "The file of changes not exist" « std::endl;
00235
00236
00237 }
```

4.21 src/inputoutput/keepAllChanges.h File Reference

```
#include <algorithm>
#include <ctime>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <map>
#include <stack>
#include <string>
#include <vector>
#include "../classes/student.h"
```

Functions

void makeBackup ()

Creates a backup of the "students_classes.csv" file with the lastest archive modified. The backup file is named with the current system date.

bool orderVector (const std::string &str1, const std::string &str2)

Compare two strings in descending order.

• void keepAllChanges (std::map< std::string, myStudent > &students, std::stack< alter > &stackAlter)

Saves all changes made to the student tree in the "students_classes.csv" file.

std::string getSysdate ()

Get the system date.

void listAllBackups ()

List all backup files.

void printChanges (int cdBkp)

Print the changes from backup files.

• bool printAllBackups ()

Prints all backup file names stored in the public vector backups.

void backupFile (int cdBkp)

Backup a specific file and remove related changes.

void keepAllChanges (std::map< std::string, myStudent > &students)

4.21.1 Function Documentation

4.21.1.1 backupFile()

```
void backupFile ( int \ \textit{cdBkp} \ )
```

Backup a specific file and remove related changes.

This function backs up a specified file from "schedule/backup" to "schedule/students_classes.csv" and removes related change files in the "schedule/alter" and "schedule/backup" directories.

Parameters

cdBkp The index of the backup file to restore.

Definition at line 204 of file keepAllChanges.cpp.

```
00204
00205
         std::string path = "schedule/backup/" + backups[cdBkp];
00206
00207
00208
         std::ifstream backup(path, std::ios::binary);
00209
00210
00211
           std::cerr « "Error opening file" « std::endl;
00212
00213
00214
         std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00215
00216
        std::cerr « "Error opening file" « std::endl;
}
         if (!file) {
00217
00218
00219
00220
         file « backup.rdbuf();
00221
         file.close();
00222
         backup.close();
00223
         unsigned size = cdBkp;
for (unsigned i = 0; i <= size; i++) {
   if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {</pre>
00224
00225
00226
              try {
               std::filesystem::remove("schedule/alter/" + backups[i]);
std::filesystem::remove("schedule/backup/" + backups[i]);
00228
00229
00230
             } catch (const std::filesystem::filesystem_error &e) {
  std::cerr « "Error to remove the file" « e.what() « std::endl;
00231
00232
00233
          } else {
00234
              std::cout « "The file of changes not exist" « std::endl;
00235
00236
00237 }
```

4.21.1.2 getSysdate()

```
std::string getSysdate ( )
```

Get the system date.

Returns

A string with the system date.

Definition at line 22 of file keepAllChanges.cpp.

4.21.1.3 keepAllChanges() [1/2]

4.21.1.4 keepAllChanges() [2/2]

Saves all changes made to the student tree in the "students classes.csv" file.

Parameters

students	Reference to the map containing student data.
stackAlter	Reference to a stack containing alteration records.

Definition at line 69 of file keepAllChanges.cpp.

```
00071
        makeBackup();
00072
        std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00073
                                " CSV".
00074
                            std::ios::app);
00075
        if (!alter.is_open()) {
00076
         std::cerr « "Error opening file" « std::endl;
00077
00078
       while (!stackAlter.empty()) {
   alter « "The student: " « stackAlter.top().studentCode « " - "
00079
08000
                « stackAlter.top().studentName « " " « stackAlter.top().type
00081
                " "UC: " « stackAlter.top().ucCode
« " Class: " « stackAlter.top().classCode « std::endl;
00082
00083
00084
         stackAlter.pop();
       }
00085
00086
00087
        std::ofstream file("schedule/students classes.csv");
00088
00089
        if (!file.is_open()) {
00090
         std::cerr « "Error opening file" « std::endl;
00091
00092
00093
        // Header
00094
        file « "StudentCode, StudentName, UcCode, ClassCode" « std::endl;
00095
00096
        // Write the tree in the file
00097
        for (auto it = students.begin(); it != students.end(); it++) {
         00098
00099
00100
00101
                 « std::endl;
```

```
00102 }
00103 }
00104 }
```

4.21.1.5 listAllBackups()

```
void listAllBackups ( )
```

List all backup files.

If no backup files exist, this function searches for and populates the 'backups' vector with filenames from the "schedule/backup" directory.

Definition at line 115 of file keepAllChanges.cpp.

```
00116
         if (backups.size() == 0) {
          std::string way = "schedule/backup";
for (const auto &in : std::filesystem::directory_iterator(way)) {
00117
00118
            if (std::filesystem::is_regular_file(in)) {
00119
00120
               backups.push_back(in.path().filename().string());
00121
            }
00122
00123
          std::sort(backups.begin(), backups.end(), orderVector);
        }
00124
00125 }
```

4.21.1.6 makeBackup()

```
void makeBackup ( )
```

Creates a backup of the "students_classes.csv" file with the lastest archive modified. The backup file is named with the current system date.

Definition at line 38 of file keepAllChanges.cpp.

```
00038
00039
        std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00040
00041
       if (!file) {
00042
         std::cerr « "Error opening file" « std::endl;
00043
00044
00045
       std::string dateString = getSysdate();
00046
00047
       std::string backupName =
00048
            "schedule/backup/students_classes-" + dateString + ".csv";
00049
       std::ofstream backup(backupName, std::ios::binary);
00050
       if (!backup) {
00051
       std::cerr « "Error to create a backup file" « std::endl;
00052
00053
         return;
00054
00055
00056
       backup « file.rdbuf();
00057
       file.close();
00058
       backup.close();
00059 }
```

4.21.1.7 orderVector()

Compare two strings in descending order.

Parameters

str1	The first string to compare.
str2	The second string to compare.

Returns

True if 'str1' is greater than 'str2', otherwise false.

Definition at line 13 of file keepAllChanges.cpp.

```
00013
00014 return str1 > str2;
00015 }
```

4.21.1.8 printAllBackups()

```
bool printAllBackups ( )
```

Prints all backup file names stored in the public vector backups.

Definition at line 132 of file keepAllChanges.cpp.

```
00132
            if (backups.size() != 0) {
   std::cout « "Backups: " « std::endl;
   for (unsigned i = 0; i < backups.size(); i++) {
     std::cout « i « " - " « backups.at(i) « std::endl;</pre>
00133
00134
00135
00136
00137
               }
00138
               return true;
00139
            } else {
            std::cout « "No backups" « std::endl;
00140
00141
               return false;
00142
00143 }
```

4.21.1.9 printChanges()

Print the changes from backup files.

This function prints the content of backup files located in the "schedule/alter" directory, up to the specified 'cdBkp' index, to the standard output.

Parameters

r		_
ı	cdBkp	The index of the backup files to print.
П	Cabhp	The mack of the backup mes to print.

Definition at line 158 of file keepAllChanges.cpp.

```
00158
00159
         unsigned size = cdBkp;
         for (unsigned i = 0; i <= size; i++) {
   std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);</pre>
00160
00161
00162
           if (!file) {
00163
             std::cerr « "Error opening file" « std::endl;
00164
           }
00165
00166
00167
           std::string line;
```

```
00169
         while (std::getline(file, line)) {
00170
           std::cout « "
                           " « line « std::endl;
00171
00172
         file.close();
00173
00174
00175
       // // Write the tree in the file
00176
       // for (auto it = students.begin(); it != students.end(); it++) {
            // for (auto classe : it->second.getClasses()) {
// // std::cout« it->second.getCode() « "," « it->second.getName()
00177
00178
00179
            // ","
// // « classe.getUcCode() « "," « classe.getClassCode() «
00180
00181
00182
            std::endl;
00183
             // file « it->second.getStudentCode() « "," «
00184
                 it->second.getStudentName() «
                     00185
00186
00187 //
00188 // }
             // }
00189 }
```

4.22 keepAllChanges.h

Go to the documentation of this file.

```
00001 #ifndef KEEPALLCHANGES H
00002 #define KEEPALLCHANGES_H
00004 #include <algorithm>
00005 #include <ctime>
00006 #include <filesystem>
00007 #include <fstream>
00008 #include <iostream>
00009 #include <map>
00010 #include <stack>
00011 #include <string>
00012 #include <vector>
00013
00014 #include "../classes/student.h"
00016 void makeBackup();
00017 bool orderVector(const std::string &str1, const std::string &str2);
00018 void keepAllChanges(std::map<std::string, myStudent> &students,
00019
                          std::stack<alter> &stackAlter);
00020 std::string getSysdate();
00021 void listAllBackups();
00022 void printChanges (int cdBkp);
00023 bool printAllBackups();
00024 void backupFile(int cdBkp);
00025
00026 void makeBackup();
00027 void keepAllChanges(std::map<std::string, myStudent> &students);
00028
00029 #endif
```

4.23 src/inputoutput/print.cpp File Reference

```
#include "print.h"
```

Functions

- $\bullet \ \ \mathsf{void} \ \mathsf{printStudent} \ (\mathsf{const} \ \mathsf{std} :: \mathsf{map} < \mathsf{std} :: \mathsf{string}, \ \mathsf{myStudent} > \& \mathsf{students}) \\$
- void printStudents (const std::vector< myStudent > &students)

Print students information from a vector.

Print student information.

• void printStudentClasses (std::map< std::string, myStudent >::iterator &it)

Print student's classes.

void printUcClasses (const std::vector< myUc > &ucVector)

Print UC classes information.

void printUcs (const std::vector< myUc > &ucs)

Print UC information.

- std::list< std::string > valideFreeClass (std::map< std::string, std::vector< classQtd > >::iterator it_count)
 Find and return valid free classes.
- bool verifyClassCode (std::string classCode, std::string ucCode, std::map< std::string, std::vector< classQtd >> &count)

Verify class code for availability.

- void printFreeClasses (std::string ucCode, std::map< std::string, std::vector< classQtd >> &count)
- void printStudentSchedules (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

Print available free classes for a specific UC.

Variables

- int equilibre = 3
- int max_students = 6

4.23.1 Function Documentation

4.23.1.1 printFreeClasses()

```
void printFreeClasses (
                std::string ucCode,
                std::map< std::string, std::vector< classQtd > > & count )
Definition at line 202 of file print.cpp.
00203
00204
00205
        auto it count = count.find(ucCode);
00206
        std::list<std::string> free_classes;
00207
        if (it_count != count.end()) {
  free_classes = valideFreeClass(it_count);
  std::cout « " Classes: " « std::endl;
00208
00209
00210
00211
00212
         if (!free_classes.empty()) {
00213
            for (auto it_list = free_classes.begin(); it_list != free_classes.end();
               it_list++) {
std::cout « "
00214
                                   " « *it_list « std::endl;
00215
00216
00217
         } else {
00218
            std::cout « "
                                No classes available" « std::endl;
00219
00220 } else {
00221
          std::cout « " Uc not found" « std::endl;
        }
00222
00223 }
```

4.23.1.2 printStudent()

Print student information.

This function prints a tabular representation of student information, including student code, student name, associated UC codes, and class codes.

Parameters

students

A map containing student information.

Definition at line 14 of file print.cpp.

```
00015
         std::cout « "Student Code | Student Name"
00016
                     « std::endl;
00017
         for (const auto &studentPair : students) {
00018
         const myStudent &student = studentPair.second;
00019
           std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
std::cout « " " « "Classes: " « std::endl;
00020
          for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00022
00023
           }
00024
00025
00026
         }
00027 }
```

4.23.1.3 printStudentClasses()

Print student's classes.

This function clears the screen and displays information about a student's classes, including the student's code, name, and associated class codes.

Parameters

it An iterator pointing to a student in a map.

Definition at line 69 of file print.cpp.

4.23.1.4 printStudents()

Print students information from a vector.

This function prints a tabular representation of student information, including student code, student name, associated UC codes, and class codes, from a vector of myStudent objects.

Parameters

students

A vector containing myStudent objects.

Definition at line 40 of file print.cpp.

```
std::cout « "Student Code | Student Name"
00041
00042
                     « std::endl;
00043
00044
         if (students.emptv()) {
           std::cout « "Empty vector ucs" « std::endl;
00046
00047
00048
         for (const auto &student : students) {
         std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
std::cout « " " « "Classes: " « std::endl;
00049
00050
           for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051
00052
00053
00054
00055 }
```

4.23.1.5 printStudentSchedules()

```
void printStudentSchedules (  std::map < std::string, \ myStudent >::iterator \& it, \\ std::map < std::string, \ myUc > \& \ classes \ )
```

Print available free classes for a specific UC.

This function identifies and prints the class codes that are available for enrollment within a given UC, based on class quantity information.

Parameters

ucCode	The UC code for which to find available classes.
count	A map of class quantity information.

```
Definition at line 244 of file print.cpp.
```

```
00246
           auto orderClasses = orderStudentClass(it, classes);
          std::cout « "\nSchedules: " « std::endl;
for (const auto &pair : orderClasses) {
00247
00248
00249
             std::string day = weekDayString(pair.first);
std::cout « "Day: " « day « std::endl;
for (const auto &info : pair.second) {
00250
00251
               std::cout « info.code « " - ";
std::cout « info.startTime « " to ";
00252
00253
                 std::cout « info.startTime + info.duration « " - ";
00254
00255
                std::cout « info.type « std::endl;
00256
00257
              std::cout « std::endl;
00258
00259 }
```

4.23.1.6 printUcClasses()

```
void printUcClasses ( {\tt const\ std::vector<\ myUc>\&\ \it ucVector\ )}
```

Print UC classes information.

This function displays information about UC classes, including UC code, class code, type, day, dayInt, start time, and duration, from a vector of myUc objects.

Parameters

ucVector	A vector of myUc objects.
classes	A map of class information.

Definition at line 92 of file print.cpp.

```
std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00093
00094
                « std::endl;
00095
00096
       for (const auto &classes : ucVector) {
00098
         auto infoVec = classes.getClassInfoVec();
00099
        for (const auto &classInfo : infoVec) {
00100
          std::string type = classInfo.type;
          std::string day = classInfo.day;
int dayInt = classInfo.dayInt;
00101
00102
00103
          double startTime = classInfo.startTime;
          00104
00105
00106
00107
00108
00109 }
00110 }
```

4.23.1.7 printUcs()

```
void printUcs ( \label{eq:const_std::vector} \mbox{const std::vector} < \mbox{ myUc } > \mbox{ \& } \mbox{ } ucs \mbox{ )}
```

Print UC information.

This function displays information about UCs, including UC code and class code, from a vector of myUc objects.

Parameters

ucs A vector of myUc objects.

Definition at line 121 of file print.cpp.

4.23.1.8 valideFreeClass()

Find and return valid free classes.

This function calculates and returns a list of valid free classes based on the input class information. Valid free classes have a minimum number of students and can accept new students within certain limits.

Parameters

it_count An iterator pointing to class quantity information.

Returns

A list of valid free class codes.

Definition at line 142 of file print.cpp.

```
00144
        int min = INT_MAX;
00145
        std::list<std::string> free_classes;
00146
00147
        // first verify the class with the minimum number of students
00148
        for (auto &classe : it_count->second) {
00149
          if (classe.qtd < min) {</pre>
00150
            min = classe.qtd;
00151
00152
        ^{\prime\prime} then verify if the class is able to accept new students and add to the
00153
        // list
00154
00155
        for (auto &classe : it_count->second) {
00156
         if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {</pre>
00157
            free_classes.push_back(classe.classCode);
00158
00159
       }
00160
00161
       // return list
00162
       return free_classes;
00163 }
```

4.23.1.9 verifyClassCode()

Verify class code for availability.

This function checks whether a given class code in the context of a specific UC code is available and can accept new students. It uses the class quantity information to determine availability.

Parameters

classCode	The class code to verify.
ucCode	The UC code associated with the class.
count	A map of class quantity information.

Returns

True if the class code is available, else false.

Definition at line 181 of file print.cpp.

```
00182
        auto it_count = count.find(ucCode);
00183
00184
00185
        if (it_count != count.end()) {
00186
         std::list<std::string> free_classes = valideFreeClass(it_count);
00187
         for (auto it_list = free_classes.begin(); it_list != free_classes.end();
               it_list++) {
00188
            if (*it_list == classCode) {
00189
00190
             return true;
00191
           }
00192
       } else {
00193
00194
         std::cout « "Error in find uc" « std::endl;
00195
00196
        return false;
00197 }
```

4.23.2 Variable Documentation

4.23.2.1 equilibre

```
int equilibre = 3
```

4.24 print.cpp 77

Definition at line 3 of file print.cpp.

4.23.2.2 max_students

```
int max students = 6
```

Definition at line 4 of file print.cpp.

4.24 print.cpp

Go to the documentation of this file.

```
00001 #include "print.h"
00003 int equilibre = 3;
00004 int max_students = 6;
00005
00014 void printStudent(const std::map<std::string, myStudent> &students) {
00015 std::cout « "Student Code | Student Name"
                  « std::endl;
00017
00018
       for (const auto &studentPair : students) {
        const myStudent &student = studentPair.second;
std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
std::cout « " " « "Classes: " « std::endl;
00019
00020
00021
         for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00022
00023
00024
00025
00026
       }
00027 }
00028
00030 // n = number of lines in the file
00042
                  « std::endl;
00043
00044
       if (students.empty()) {
00045
         std::cout « "Empty vector ucs" « std::endl;
00046
00047
00048
        for (const auto &student : students) {
        std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00049
          std::cout « " " « "Classes: " « std::endl;
00050
         for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051
00052
00053
00054 }
00055 }
00056
00057 // O(m)
00058 // Find the student in the map
00059 // m = number of students
00060 // m >= 7
00069 void printStudentClasses(std::map<std::string, myStudent>::iterator &it) {
00070 system("clear");
        00072
00073
       for (const auto &classe : it->second.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode()
00074
00075
00076
                    « std::endl;
00077
00078 }
00079
00080 //O(n)
00081 // n = number of lines in the file
00092 void printUcClasses(const std::vector<myUc> &ucVector) {
       std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00094
                  « std::endl;
00095
00096
       for (const auto &classes : ucVector) {
00097
00098
         auto infoVec = classes.getClassInfoVec();
00099
         for (const auto &classInfo : infoVec) {
            std::string type = classInfo.type;
```

```
std::string day = classInfo.day;
            int dayInt = classInfo.dayInt;
00102
00103
            double startTime = classInfo.startTime;
           00104
00105
00106
00107
00108
00109 }
00110 }
00111
00112 // O(n)
00121 void printUcs(const std::vector<myUc> &ucs) {
00122 std::cout « "UcCode | ClassCode" « std::endl;
00123
       std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
}
00124
00125
00126
00127 }
00128
00129 // O(m)
00130 // Iterator to UC \,
00131 // m = number of classes in the UC
00142 std::list<std::string> valideFreeClass(
00143
         std::map<std::string, std::vector<classQtd»::iterator it_count) {</pre>
        int min = INT_MAX;
00144
00145
        std::list<std::string> free_classes;
00146
00147
        \ensuremath{//} first verify the class with the minimum number of students
00148
       for (auto &classe : it_count->second) {
   if (classe.qtd < min) {</pre>
00149
00150
           min = classe.qtd;
00151
00152
00153
        // then verify if the class is able to accept new students and add to the
        // list
00154
00155
        for (auto &classe : it_count->second) {
         if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {</pre>
00157
           free_classes.push_back(classe.classCode);
00158
00159
00160
       // return list
00161
00162
       return free_classes;
00163 }
00164
00165 // O(log(n) + m)
00166 // n = number of UCs
00167 // m = number of free classes
00168
00181 bool verifyClassCode(std::string classCode, std::string ucCode,
00182
                           std::map<std::string, std::vector<classQtd> &count) {
00183
        auto it_count = count.find(ucCode);
00184
       if (it count != count.end()) {
00185
        std::list<std::string> free_classes = valideFreeClass(it_count);
00186
         for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00188
               it_list++) {
00189
            if (*it_list == classCode) {
00190
              return true;
            }
00191
00192
          }
00193
       } else {
00194
         std::cout « "Error in find uc" « std::endl;
00195
       return false;
00196
00197 }
00198
00199 // O(log(n) + m)
00200 // n = number of UCs
00201 // m = number of free classes
00202 void printFreeClasses(std::string ucCode,
00203
                            std::map<std::string, std::vector<classQtd» &count) {</pre>
00204
00205
       auto it count = count.find(ucCode);
       std::list<std::string> free_classes;
00206
00207
00208
       if (it_count != count.end()) {
         free_classes = valideFreeClass(it_count);
std::cout « " Classes: " « std::endl;
00209
00210
00211
00212
          if (!free_classes.empty()) {
00213
           for (auto it_list = free_classes.begin(); it_list != free_classes.end();
              it_list++) {
std::cout « "
00214
                                  " « *it_list « std::endl;
00215
00216
00217
          } else {
```

```
00218
             std::cout « "
                                   No classes available" « std::endl;
00219
       std::cout « " Uc not found" « std::endl;
}
00220
00221
00222
00223 }
00225 // orderStudentClass:
00226 // O(m*log(n)*k)
00227 // n = number of UCs
00228 // m = number of classes
00229 // k = number of type of classes (T,TP)
00230
00231 // O(m*k)
00232 // m = number of classes
00233 // k = number of type of classes (T,TP)
00234
00244 void printStudentSchedules(std::map<std::string, myStudent>::iterator &it,
                                     std::map<std::string, myUc> &classes) {
        auto orderClasses = orderStudentClass(it, classes);
00247
        std::cout « "\nSchedules: " « std::endl;
00248 for (const auto &pair : orderClasses) {
         std::string day = weekDayString(pair.first);
std::cout « "Day: " « day « std::endl;
for (const auto &info : pair.second) {
   std::cout « info.code « " - ";
   std::cout « info.startTime « " to ";
00249
00250
00251
00253
00254
            std::cout « info.startTime + info.duration « " - ";
00255
             std::cout « info.type « std::endl;
00256
00257
           std::cout « std::endl;
00258
        }
00259 }
```

4.25 src/inputoutput/print.h File Reference

```
#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
#include <liist>
#include <map>
#include <string>
#include <vector>
#include "../classes/student.h"
#include "../functions/dbStudents.h"
```

Functions

- void workingMessage ()
- void errorMessage ()
- void printStudents (const std::vector< myStudent > &students)

Print students information from a vector.

void printStudent (const std::map< std::string, myStudent > &students)

Print student information.

void printUcClasses (const std::vector< myUc > &ucVector)

Print UC classes information.

void printUcs (const std::vector< myUc > &ucs)

Print UC information.

void printStudentSchedules (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

Print available free classes for a specific UC.

- void printStudentClasses (std::map< std::string, myStudent >::iterator &it)

 Print student's classes.
- void printFreeClasses (std::string ucCode, std::map< std::string, std::vector< classQtd >> &count)
- std::list< std::string > valideFreeClass (std::map< std::string, std::vector< classQtd > >::iterator it_count)
 Find and return valid free classes.
- bool verifyClassCode (std::string classCode, std::string ucCode, std::map< std::string, std::vector< classQtd >> &count)

Verify class code for availability.

4.25.1 Function Documentation

4.25.1.1 errorMessage()

void errorMessage ()

```
Definition at line 4 of file errorMsgs.cpp.
```

```
00005 std::cout « "ERROR: Invalid choice." « std::endl; 00006 exit(0); 00007 }
```

4.25.1.2 printFreeClasses()

```
Definition at line 202 of file print.cpp.
```

```
00203
00204
00205
         auto it count = count.find(ucCode);
00206
         std::list<std::string> free_classes;
00207
         if (it_count != count.end()) {
  free_classes = valideFreeClass(it_count);
  std::cout « " Classes: " « std::endl;
00208
00209
00210
00211
00212
           if (!free_classes.empty()) {
00213
            for (auto it_list = free_classes.begin(); it_list != free_classes.end();
                it_list++) {
std::cout « "
00214
                                       " « *it_list « std::endl;
00215
00216
00217
          } else {
00218
             std::cout « "
                                  No classes available" « std::endl;
00219
00220
        } else {
           std::cout « " Uc not found" « std::endl;
00221
00222
00223 }
```

4.25.1.3 printStudent()

Print student information.

This function prints a tabular representation of student information, including student code, student name, associated UC codes, and class codes.

Parameters

students

A map containing student information.

Definition at line 14 of file print.cpp.

```
00015
         std::cout « "Student Code | Student Name"
00016
                     « std::endl;
00017
         for (const auto &studentPair : students) {
00018
         const myStudent &student = studentPair.second;
00019
           std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
std::cout « " " « "Classes: " « std::endl;
00020
          for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00022
00023
           }
00024
00025
00026
         }
00027 }
```

4.25.1.4 printStudentClasses()

Print student's classes.

This function clears the screen and displays information about a student's classes, including the student's code, name, and associated class codes.

Parameters

it An iterator pointing to a student in a map.

Definition at line 69 of file print.cpp.

4.25.1.5 printStudents()

Print students information from a vector.

This function prints a tabular representation of student information, including student code, student name, associated UC codes, and class codes, from a vector of myStudent objects.

Parameters

students

A vector containing myStudent objects.

Definition at line 40 of file print.cpp.

```
std::cout « "Student Code | Student Name"
00041
00042
                     « std::endl;
00043
00044
         if (students.emptv()) {
          std::cout « "Empty vector ucs" « std::endl;
00046
00047
00048
         for (const auto &student : students) {
         std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl; std::cout « " " « "Classes: " « std::endl;
00049
00050
           for(const auto &classe : student.getClasses()) {
   std::cout « " " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051
00052
00053
00054
00055 }
```

4.25.1.6 printStudentSchedules()

```
void printStudentSchedules (  std::map < std::string, \ myStudent >::iterator \& it, \\ std::map < std::string, \ myUc > \& \ classes \ )
```

Print available free classes for a specific UC.

This function identifies and prints the class codes that are available for enrollment within a given UC, based on class quantity information.

Parameters

ucCode	The UC code for which to find available classes.
count	A map of class quantity information.

```
Definition at line 244 of file print.cpp.
```

```
00246
           auto orderClasses = orderStudentClass(it, classes);
          std::cout « "\nSchedules: " « std::endl;
for (const auto &pair : orderClasses) {
00247
00248
00249
             std::string day = weekDayString(pair.first);
std::cout « "Day: " « day « std::endl;
for (const auto &info : pair.second) {
00250
00251
               std::cout « info.code « " - ";
std::cout « info.startTime « " to ";
00252
00253
                 std::cout « info.startTime + info.duration « " - ";
00254
00255
                std::cout « info.type « std::endl;
00256
00257
              std::cout « std::endl;
00258
00259 }
```

4.25.1.7 printUcClasses()

```
void printUcClasses ( {\tt const\ std::vector} < \ {\tt myUc} \ > \ {\tt \&} \ \ {\tt ucVector} \ )
```

Print UC classes information.

This function displays information about UC classes, including UC code, class code, type, day, dayInt, start time, and duration, from a vector of myUc objects.

Parameters

ucVector	A vector of myUc objects.
classes	A map of class information.

Definition at line 92 of file print.cpp.

```
std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00093
00094
                « std::endl;
00095
00096
       for (const auto &classes : ucVector) {
00098
         auto infoVec = classes.getClassInfoVec();
00099
        for (const auto &classInfo : infoVec) {
00100
          std::string type = classInfo.type;
          std::string day = classInfo.day;
int dayInt = classInfo.dayInt;
00101
00102
00103
          double startTime = classInfo.startTime;
          00104
00105
00106
00107
00108
00109 }
00110 }
```

4.25.1.8 printUcs()

```
void printUcs ( \label{eq:const_std::vector} \mbox{const std::vector} < \mbox{ myUc } > \mbox{ \& } \mbox{ $ucs$ } \mbox{)}
```

Print UC information.

This function displays information about UCs, including UC code and class code, from a vector of myUc objects.

Parameters

ucs A vector of myUc objects.

Definition at line 121 of file print.cpp.

4.25.1.9 valideFreeClass()

Find and return valid free classes.

This function calculates and returns a list of valid free classes based on the input class information. Valid free classes have a minimum number of students and can accept new students within certain limits.

Parameters

it_count An iterator pointing to class quantity information.

Returns

A list of valid free class codes.

Definition at line 142 of file print.cpp.

```
00144
        int min = INT_MAX;
00145
        std::list<std::string> free_classes;
00146
00147
        // first verify the class with the minimum number of students
00148
        for (auto &classe : it_count->second) {
00149
          if (classe.qtd < min) {</pre>
00150
            min = classe.qtd;
00151
00152
        ^{\prime\prime} then verify if the class is able to accept new students and add to the
00153
        // list
00154
00155
        for (auto &classe : it_count->second) {
00156
         if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {</pre>
00157
            free_classes.push_back(classe.classCode);
00158
00159
       }
00160
00161
        // return list
00162
       return free_classes;
00163 }
```

4.25.1.10 verifyClassCode()

Verify class code for availability.

This function checks whether a given class code in the context of a specific UC code is available and can accept new students. It uses the class quantity information to determine availability.

Parameters

classCode	The class code to verify.
ucCode	The UC code associated with the class.
count	A map of class quantity information.

Returns

True if the class code is available, else false.

Definition at line 181 of file print.cpp.

```
00182
00183
                                                                             {
        auto it_count = count.find(ucCode);
00184
        if (it_count != count.end()) {
00185
00186
          std::list<std::string> free_classes = valideFreeClass(it_count);
00187
          for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00188
               it list++) {
            if (*it_list == classCode) {
00189
00190
             return true;
            }
00191
00192
00193
00194
         std::cout « "Error in find uc" « std::endl;
00195
00196
        return false;
00197 }
```

4.25.1.11 workingMessage()

```
void workingMessage ( )
```

4.26 print.h 85

Definition at line 26 of file errorMsgs.cpp.

```
00026 {
00027 std::cout « "WARNING: Function not done yet." « std::endl;
00028 }
```

4.26 print.h

Go to the documentation of this file.

```
00001 #ifndef PRINT_H
00002 #define PRINT_H
00003
00004 #include <algorithm>
00005 #include <climits>
00006 #include <fstream>
00007 #include <iostream>
00008 #include <list>
00009 #include <map>
00010 #include <string>
00011 #include <vector>
00012
00013 #include "../classes/student.h"
00014 #include "../functions/dbStudents.h"
00015
00016 void workingMessage();
00017 void errorMessage();
00018
00019 void printStudents(const std::vector<myStudent> &students);
00020 void printStudent(const std::map<std::string, myStudent> &students);
00022 void printUcClasses(const std::vector<myUc> &ucVector);
00023 void printUcs(const std::vector<myUc> &ucs);
00024
00025 void printStudentSchedules(std::map<std::string, myStudent>::iterator &it,
00026
                               std::map<std::string, myUc> &classes);
00027 void printStudentClasses(std::map<std::string, myStudent>::iterator &it);
00028 void printFreeClasses(std::string ucCode,
std::map<std::string, std::vector<classQtd>::iterator it_count);
00032 bool verifyClassCode(std::string classCode, std::string ucCode,
00033
                         std::map<std::string, std::vector<classQtd» &count);</pre>
00034
00035 #endif
```

4.27 src/inputoutput/read.cpp File Reference

```
#include "read.h"
```

Functions

- std::map< std::string, myStudent > readStudents (std::map< std::string, std::vector< classQtd > > &count)

 Read and process student and class information from a CSV file.
- std::map< std::string, std::vector< myUc >> readUcs (std::map< std::string, std::vector< classQtd >> &count)

Read and process UC and class information from a CSV file.

std::map< std::string, myUc > readSchedules ()

Read and process class schedule information from a CSV file.

4.27.1 Function Documentation

4.27.1.1 readSchedules()

```
std::map< std::string, myUc > readSchedules ( )
```

Read and process class schedule information from a CSV file.

This function reads and processes class schedule information from a CSV file, populating a map of classes and their associated details, including UC code, day, type, start time, and duration.

Returns

A map of classes with their associated information.

Definition at line 185 of file read.cpp.

```
00185
                                                   { // O(n*log(n))
00186
         std::string line;
        std::map<std::string, myUc> classes;
00187
        std::map<std::string, int> dayToInt = {
    {"Sunday", 1},    {"Monday", 2},    {"Tuesday", 3},    {"Wednesday", 4},
    {"Thursday", 5},    {"Friday", 6},    {"Saturday", 7}};
00188
00189
00190
00191
00192
         std::ifstream file("schedule/classes.csv");
00193
        if (!file.is_open()) {
00194
          errorMessageFile();
00195
00196
00197
        bool header = true;
00198
        while (std::getline(file, line)) {
00199
          if (header) {
           header = false;
00200
00201
             continue:
00202
00203
          std::istringstream ss(line);
00204
           std::string classCode, ucCode, day, type;
00205
           double startTime, duration;
00206
          int dayInt = 0;
00207
00208
           std::getline(ss, classCode, ',');
00209
           std::getline(ss, ucCode, ',');
           std::getline(ss, day, ',');
00210
00211
           ss » startTime;
00212
           ss.ignore();
00213
           ss » duration;
00214
           ss.ignore();
00215
           std::getline(ss, type);
00216
00217
           type.erase(std::find_if(type.rbegin(), type.rend(),
00218
                                      [](unsigned char ch) { return !std::isspace(ch); })
                           .base().
00219
00220
                       type.end());
00221
00222
           auto it1 = dayToInt.find(day);
00223
           if (it1 != dayToInt.end()) {
            dayInt = it1->second;
00224
00225
           } else {
00226
            std::cout « "Invalid day: " « day « std::endl;
00227
00228
00229
           // Check if the class code already exists in the map
00230
           auto it2 = classes.find(ucCode + classCode);
           if (it2 != classes.end()) {
00231
            it2->second.addClassInfo(type, day, dayInt, startTime, duration);
00232
00233
           } else {
00234
            myUc newUcClass;
00235
             newUcClass.setUcCode(ucCode);
00236
             newUcClass.addClass(classCode);
             newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
classes[ucCode + classCode] = newUcClass;
00237
00238
00239
00240
00241
        return classes;
00242 }
```

4.27.1.2 readStudents()

Read and process student and class information from a CSV file.

This function reads and processes student and class information from a CSV file, populating a map of students and updating class quantity information based on the data.

Parameters

count A map of class quantity information.

Returns

A map of students with associated classes.

Definition at line 17 of file read.cpp.

```
00017
00018
        std::string line;
00019
        std::map<std::string, myStudent> students;
00020
00021
        std::ifstream file("schedule/students_classes.csv");
00022
        if (!file.is_open()) {
00023
          errorMessageFile();
        1
00024
00025
00026
        bool header = true;
00027
        while (std::getline(file, line)) {
         if (header) {
  header = false;
00028
00029
00030
             continue;
00031
00032
          std::istringstream ss(line);
00033
00034
          std::string studentCode, studentName, ucCode, classCode;
00035
00036
          std::getline(ss, studentCode, ',');
          std::getline(ss, studentName, ',');
00037
          std::getline(ss, ucCode, ',');
std::getline(ss, classCode);
00038
00039
00040
00041
          classCode.erase(
00042
             std::find_if(classCode.rbegin(), classCode.rend(),
00043
                             [](unsigned char ch) { return !std::isspace(ch); })
00044
                   .base(),
00045
              classCode.end());
00046
          auto it = students.find(studentCode);
if (it != students.end()) {
00047
00048
00049
            it->second.addClass(myUc(ucCode, classCode));
00050
          } else {
            myStudent newStudent(studentCode, studentName);
newStudent.addClass(myUc(ucCode, classCode));
00051
00052
00053
             students[studentCode] = newStudent;
00054
00055
          \ensuremath{//} verify if the uc exists in the count tree
00056
00057
          auto it count = count.find(ucCode);
00058
00059
           // if not exists, add the class with one student in the count tree
00060
           if (it_count == count.end()) {
             std::vector<classQtd> classVec;
00061
00062
             classVec.push_back({classCode, 1});
00063
             count.emplace(ucCode, classVec);
00064
00065
             // if uc exist, then verify if the class exists in the vector
00066
             bool exist = false;
00067
             for (auto &class_it : it_count->second) {
00068
              // if exists, add +1 in the qtd
00069
               if (class_it.classCode == classCode) {
00070
                 class_it.gtd++;
00071
                 exist = true;
```

```
break;
00073
00074
            // if not exists, add the class with one student in the vector
00075
00076
            if (!exist) {
00077
              it_count->second.push_back({classCode, 1});
00078
00079
08000
00081
        file.close();
00082
00083
       return students;
00084 }
```

4.27.1.3 readUcs()

```
std::map< std::string, std::vector< myUc >> readUcs ( std::map< std::string, std::vector< classQtd >> \& count )
```

Read and process UC and class information from a CSV file.

This function reads and processes UC and class information from a CSV file, populating a map of UCs and their associated classes, as well as updating class quantity information based on the data.

Parameters

count A map of class quantity information.

Returns

00136

00137

A map of UCs and their associated classes.

newUc.setClassCode(classCode);

it->second.push_back(newUc);

Definition at line 100 of file read.cpp.

```
00101
        std::string line;
00102
        std::map<std::string, std::vector<myUc» ucClasses;</pre>
00103
00104
        std::ifstream file("schedule/classes per uc.csv");
00105
        if (!file.is open()) {
00106
          errorMessageFile();
00107
00108
00109
        bool header = true;
        while (std::getline(file, line)) {
00110
00111
          // testing
00112
          // std::cout « "line" « std::endl;
00113
00114
          if (header) {
00115
            header = false;
00116
            continue:
00117
00118
          std::istringstream ss(line);
00119
          std::string ucCode, classCode;
00120
          std::getline(ss, ucCode, ',');
std::getline(ss, classCode, ',');
00121
00122
00123
00124
          auto it = ucClasses.find(ucCode);
00125
00126
          classCode.erase(
              std::find_if(classCode.rbegin(), classCode.rend(),
00127
00128
                             [](unsigned char ch) { return !std::isspace(ch); })
                   .base(),
00129
00130
               classCode.end());
00131
00132
          if (it != ucClasses.end()) {
00133
            // exist
            myUc newUc;
00134
00135
            newUc.setUcCode(ucCode);
```

4.28 read.cpp 89

```
00138
         } else {
00139
           // doesnt exist
00140
            std::vector<myUc> ucVector;
00141
            myUc newUc;
            newUc.setUcCode(ucCode);
00142
00143
            newUc.setClassCode(classCode);
00144
            ucVector.push_back(newUc);
00145
            ucClasses[ucCode] = ucVector;
00146
00147
00148
         bool exist = false;
          // try to find the uc in the count tree
00149
00150
          auto it_count = count.find(ucCode);
00151
          // if found, verify if the class exists in the vector
00152
          if (it_count != count.end()) {
00153
            for (auto &class_it : it_count->second) {
              if (class_it.classCode == classCode) {
00154
00155
               exist = true;
00156
00157
00158
            // if exist uc in the count tree, but not exist the class, add the class
            // with 0 students
00159
            if (!exist) {
00160
00161
              it_count->second.push_back({classCode, 0});
00162
            ^{\prime} // if not found, add the uc and class with 0 students
00163
00164
00165
            std::vector<classQtd> classVec;
00166
            classVec.push_back({classCode, 0});
00167
            count.emplace(ucCode, classVec);
00168
         }
00169
00170
        file.close();
00171
        return ucClasses;
00172 }
```

4.28 read.cpp

Go to the documentation of this file.

```
00001 #include "read.h'
00002
00003
00004 // O(n*log(m))
00005 // n = number of lines in the file
00006 // m = number of differents students
00016 std::map<std::string, myStudent>
00017 readStudents(std::map<std::string, std::vector<classQtd» &count) {
00018
       std::string line;
00019
        std::map<std::string, myStudent> students;
00020
00021
        std::ifstream file("schedule/students classes.csv");
00022
        if (!file.is_open()) {
00023
          errorMessageFile();
        }
00024
00025
00026
        bool header = true;
00027
        while (std::getline(file, line)) {
00028
         if (header) {
00029
           header = false;
00030
             continue;
00031
00032
          std::istringstream ss(line);
00033
00034
          std::string studentCode, studentName, ucCode, classCode;
00035
          std::getline(ss, studentCode, ',');
std::getline(ss, studentName, ',');
00036
00037
00038
          std::getline(ss, ucCode, ',');
std::getline(ss, classCode);
00039
00040
00041
          classCode.erase(
00042
              std::find_if(classCode.rbegin(), classCode.rend(),
                            [](unsigned char ch) { return !std::isspace(ch); })
00043
00044
                   .base(),
00045
              classCode.end());
00046
00047
          auto it = students.find(studentCode);
          if (it != students.end()) {
00048
00049
            it->second.addClass(myUc(ucCode, classCode));
00050
00051
            myStudent newStudent(studentCode, studentName);
```

```
newStudent.addClass(myUc(ucCode, classCode));
00053
            students[studentCode] = newStudent;
00054
00055
00056
          // verify if the uc exists in the count tree
00057
          auto it_count = count.find(ucCode);
00059
          // if not exists, add the class with one student in the count tree
00060
          if (it_count == count.end()) {
00061
             std::vector<classQtd> classVec;
             classVec.push_back({classCode, 1});
00062
00063
             count.emplace(ucCode, classVec);
00064
00065
             // if uc exist, then verify if the class exists in the vector
00066
             bool exist = false;
            for (auto &class_it : it_count->second) {
   // if exists, add +1 in the qtd
   if (class_it.classCode == classCode) {
00067
00068
00069
00070
                class_it.qtd++;
00071
                 exist = true;
00072
                 break;
00073
              }
00074
00075
             ^{\prime} // if not exists, add the class with one student in the vector
00076
            if (!exist) {
00077
              it_count->second.push_back({classCode, 1});
00078
00079
         }
08000
00081
        file.close();
00082
00083
        return students;
00084 }
00085
00086 //O(n*log(m))
00087 // n = number of lines in the file
00088 // m = number of differents UCs
00099 std::map<std::string, std::vector<myUc» // O(n*log(n))
00100 readUcs(std::map<std::string, std::vector<classQtd» &count) {
00101
        std::string line;
00102
        std::map<std::string, std::vector<myUc» ucClasses;</pre>
00103
        std::ifstream file("schedule/classes_per_uc.csv");
00104
00105
        if (!file.is_open()) {
00106
          errorMessageFile();
00107
00108
        bool header = true;
00109
        while (std::getline(file, line)) {
00110
00111
         // testing
          // std::cout « "line" « std::endl;
00112
00113
00114
          if (header) {
00115
           header = false;
00116
            continue;
00117
00118
          std::istringstream ss(line);
00119
          std::string ucCode, classCode;
00120
          std::getline(ss, ucCode, ',');
00121
          std::getline(ss, classCode, ',');
00122
00123
00124
          auto it = ucClasses.find(ucCode);
00125
          classCode.erase(
00126
00127
              std::find_if(classCode.rbegin(), classCode.rend(),
00128
                             [](unsigned char ch) { return !std::isspace(ch); })
                   .base().
00129
00130
              classCode.end());
00131
00132
          if (it != ucClasses.end()) {
00133
            // exist
            myUc newUc;
00134
            newUc.setUcCode(ucCode);
newUc.setClassCode(classCode);
00135
00136
00137
             it->second.push_back(newUc);
00138
          } else {
00139
            // doesnt exist
00140
            std::vector<myUc> ucVector;
00141
            myUc newUc;
            newUc.setUcCode(ucCode);
00142
            newUc.setClassCode(classCode);
00143
00144
             ucVector.push_back(newUc);
00145
            ucClasses[ucCode] = ucVector;
00146
00147
00148
          bool exist = false;
```

4.28 read.cpp 91

```
// try to find the uc in the count tree
00150
          auto it_count = count.find(ucCode);
00151
          // if found, verify if the class exists in the vector
          if (it_count != count.end()) {
00152
00153
            for (auto &class_it : it_count->second) {
              if (class_it.classCode == classCode) {
00154
00155
               exist = true;
00156
              }
00157
            ^{\prime} // if exist uc in the count tree, but not exist the class, add the class
00158
            // with 0 students
00159
00160
            if (!exist) {
00161
              it_count->second.push_back({classCode, 0});
00162
00163
            // if not found, add the uc and class with 0 students
          } else {
00164
           std::vector<classQtd> classVec;
00165
00166
            classVec.push_back({classCode, 0});
00167
            count.emplace(ucCode, classVec);
00168
          }
00169
00170
       file.close();
00171
       return ucClasses;
00172 }
00173
00174 //O(n*log(m))
00175 // n = number of lines in the file
00176 // m = number of differents classes
00185 std::map<std::string, myUc> readSchedules() { // O(n*log(n))
00186
        std::string line;
00187
        std::map<std::string, myUc> classes;
        00188
00189
00190
00191
        std::ifstream file("schedule/classes.csv");
00192
00193
        if (!file.is_open()) {
00194
         errorMessageFile();
00195
00196
00197
        bool header = true;
        while (std::getline(file, line)) {
00198
         if (header) {
00199
           header = false;
00200
00201
            continue;
00202
00203
          std::istringstream ss(line);
00204
          std::string classCode, ucCode, day, type;
00205
          double startTime, duration;
00206
          int davInt = 0;
00207
          std::getline(ss, classCode, ',');
std::getline(ss, ucCode, ',');
std::getline(ss, day, ',');
00208
00209
00210
00211
          ss » startTime:
00212
          ss.ignore();
00213
          ss » duration;
00214
          ss.ignore();
00215
          std::getline(ss, type);
00216
          type.erase(std::find_if(type.rbegin(), type.rend(),
00217
                                   [](unsigned char ch) { return !std::isspace(ch); })
00218
00219
                         .base(),
00220
                     type.end());
00221
00222
          auto it1 = dayToInt.find(day);
          if (it1 != dayToInt.end()) {
00223
00224
           dayInt = it1->second;
00225
          } else {
00226
           std::cout « "Invalid day: " « day « std::endl;
00227
00228
00229
          // Check if the class code already exists in the map
00230
          auto it2 = classes.find(ucCode + classCode);
          if (it2 != classes.end())
00231
00232
            it2->second.addClassInfo(type, day, dayInt, startTime, duration);
00233
          } else {
00234
            myUc newUcClass;
00235
            newUcClass.setUcCode(ucCode);
            newUcClass.addClass(classCode);
00236
            newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
classes[ucCode + classCode] = newUcClass;
00237
00238
00239
00240
       }
00241
        return classes;
00242 }
```

4.29 src/inputoutput/read.h File Reference

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>
#include "../classes/student.h"
```

Functions

- · void errorMessageFile ()
- void errorMessageLine (std::string)
- std::map< std::string, myStudent > readStudents (std::map< std::string, std::vector< classQtd > > &count)
 Read and process student and class information from a CSV file.
- std::map< std::string, std::vector< myUc >> readUcs (std::map< std::string, std::vector< classQtd >> &count)

Read and process UC and class information from a CSV file.

std::map< std::string, myUc > readSchedules ()

Read and process class schedule information from a CSV file.

4.29.1 Function Documentation

4.29.1.1 errorMessageFile()

4.29.1.2 errorMessageLine()

4.29.1.3 readSchedules()

```
std::map< std::string, myUc > readSchedules ( )
```

Read and process class schedule information from a CSV file.

This function reads and processes class schedule information from a CSV file, populating a map of classes and their associated details, including UC code, day, type, start time, and duration.

Returns

A map of classes with their associated information.

Definition at line 185 of file read.cpp.

```
{ // O(n*log(n))
00185
00186
        std::string line;
00187
        std::map<std::string, myUc> classes;
        00188
00189
00190
00191
00192
        std::ifstream file("schedule/classes.csv");
00193
        if (!file.is open()) {
00194
          errorMessageFile();
00195
00196
00197
        bool header = true;
00198
        while (std::getline(file, line)) {
         if (header) {
00199
00200
            header = false;
00201
            continue;
00202
00203
          std::istringstream ss(line);
00204
          std::string classCode, ucCode, day, type;
double startTime, duration;
00205
00206
          int dayInt = 0;
00207
          std::getline(ss, classCode, ',');
std::getline(ss, ucCode, ',');
std::getline(ss, day, ',');
00208
00209
00210
00211
          ss » startTime;
          ss.ignore();
00212
00213
          ss » duration;
00214
          ss.ignore();
00215
          std::getline(ss, type);
00216
00217
          type.erase(std::find_if(type.rbegin(), type.rend(),
00218
                                     [](unsigned char ch) { return !std::isspace(ch); })
00219
                           .base(),
00220
                      type.end());
00221
00222
           auto it1 = dayToInt.find(day);
          if (it1 != dayToInt.end()) {
  dayInt = it1->second;
00223
00224
00225
          } else {
00226
            std::cout « "Invalid day: " « day « std::endl;
00227
00228
00229
          // Check if the class code already exists in the map
          auto it2 = classes.find(ucCode + classCode);
if (it2 != classes.end()) {
00230
00231
00232
            it2->second.addClassInfo(type, day, dayInt, startTime, duration);
00233
00234
            myUc newUcClass;
            newUcClass.setUcCode(ucCode);
00235
00236
            newUcClass.addClass(classCode);
            newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
classes[ucCode + classCode] = newUcClass;
00237
00238
00239
00240
        }
00241
        return classes;
00242 }
```

4.29.1.4 readStudents()

Read and process student and class information from a CSV file.

This function reads and processes student and class information from a CSV file, populating a map of students and updating class quantity information based on the data.

Parameters

count A map of class quantity information.

Returns

A map of students with associated classes.

Definition at line 17 of file read.cpp.

```
00017
00018
        std::string line;
00019
        std::map<std::string, myStudent> students;
00020
00021
        std::ifstream file("schedule/students_classes.csv");
00022
        if (!file.is_open()) {
00023
          errorMessageFile();
00024
00025
00026
        bool header = true;
00027
        while (std::getline(file, line)) {
          if (header) {
  header = false;
00028
00029
00030
             continue;
00031
00032
          std::istringstream ss(line);
00033
00034
          std::string studentCode, studentName, ucCode, classCode;
00035
00036
           std::getline(ss, studentCode, ',');
          std::getline(ss, studentName, ',');
00037
          std::getline(ss, ucCode, ',');
std::getline(ss, classCode);
00038
00039
00040
00041
           classCode.erase(
00042
              std::find_if(classCode.rbegin(), classCode.rend(),
00043
                              [](unsigned char ch) { return !std::isspace(ch); })
00044
                    .base(),
00045
               classCode.end());
00046
          auto it = students.find(studentCode);
if (it != students.end()) {
00047
00048
00049
            it->second.addClass(myUc(ucCode, classCode));
00050
           } else {
            myStudent newStudent(studentCode, studentName);
newStudent.addClass(myUc(ucCode, classCode));
00051
00052
00053
             students[studentCode] = newStudent;
00054
00055
00056
           // verify if the uc exists in the count tree
00057
          auto it count = count.find(ucCode);
00058
00059
           // if not exists, add the class with one student in the count tree
           if (it_count == count.end()) {
   std::vector<classQtd> classVec;
00060
00061
             classVec.push_back({classCode, 1});
00062
00063
             count.emplace(ucCode, classVec);
00064
00065
             // if uc exist, then verify if the class exists in the vector
00066
             bool exist = false;
00067
             for (auto &class_it : it_count->second) {
00068
               \ensuremath{//} if exists, add +1 in the qtd
00069
               if (class_it.classCode == classCode) {
00070
                 class_it.qtd++;
00071
                 exist = true;
```

```
break;
00073
00074
            // if not exists, add the class with one student in the vector
00075
00076
            if (!exist) {
00077
              it_count->second.push_back({classCode, 1});
00078
00079
08000
00081
       file.close();
00082
00083
       return students;
00084 }
```

4.29.1.5 readUcs()

```
std::map< std::string, std::vector< myUc >> readUcs ( std::map< std::string, std::vector< classQtd >> \& count )
```

Read and process UC and class information from a CSV file.

This function reads and processes UC and class information from a CSV file, populating a map of UCs and their associated classes, as well as updating class quantity information based on the data.

Parameters

count A map of class quantity information.

Returns

00111

00112

00113 00114

00115

00116

00117 00118

00119

00120

00125 00126

00127

00128

00136

00137

A map of UCs and their associated classes.

Definition at line 100 of file read.cpp.

// testing

if (header) {

continue:

header = false;

classCode.erase(

```
00101
        std::string line;
00102
        std::map<std::string, std::vector<myUc» ucClasses;</pre>
00103
00104
        std::ifstream file("schedule/classes per uc.csv");
00105
        if (!file.is open()) {
00106
         errorMessageFile();
00107
00108
00109
       bool header = true;
00110
       while (std::getline(file, line)) {
```

// std::cout « "line" « std::endl;

std::istringstream ss(line);

std::string ucCode, classCode;

std::getline(ss, ucCode, ',');
std::getline(ss, classCode, ',');

auto it = ucClasses.find(ucCode);

newUc.setClassCode(classCode);

it->second.push_back(newUc);

std::find_if(classCode.rbegin(), classCode.rend(),

[](unsigned char ch) { return !std::isspace(ch); })

```
00138
          } else {
00139
           // doesnt exist
00140
            std::vector<myUc> ucVector;
00141
            myUc newUc;
            newUc.setUcCode(ucCode);
00142
            newUc.setClassCode(classCode);
00143
            ucVector.push_back(newUc);
00145
            ucClasses[ucCode] = ucVector;
00146
00147
00148
          bool exist = false;
          // try to find the uc in the count tree
00149
00150
          auto it_count = count.find(ucCode);
00151
          // if found, verify if the class exists in the vector
00152
          if (it_count != count.end()) {
            for (auto &class_it : it_count->second) {
   if (class_it.classCode == classCode) {
00153
00154
00155
                exist = true;
00156
00157
00158
            // if exist uc in the count tree, but not exist the class, add the class
            // with 0 students
if (!exist) {
00159
00160
              it_count->second.push_back({classCode, 0});
00161
00162
            ^{\prime} // if not found, add the uc and class with 0 students
00163
00164
00165
            std::vector<classQtd> classVec;
00166
            classVec.push_back({classCode, 0});
00167
            count.emplace(ucCode, classVec);
00168
          }
00169
00170
        file.close();
00171
        return ucClasses;
00172 }
```

4.30 read.h

Go to the documentation of this file.

```
00001 #ifndef READ_H
00002 #define READ_H
00003
00004 #include <algorithm>
00005 #include <fstream>
00006 #include <iostream>
00007 #include <map>
00008 #include <sstream>
00009 #include <string>
00010 #include <vector>
00011
00012 #include "../classes/student.h"
00014 void errorMessageFile();
00015 void errorMessageLine(std::string);
00016
00017 std::map<std::string, myStudent>
00018 readStudents(std::map<std::string, std::vector<classQtd» &count);
00019 std::map<std::string, std::vector<myUc>
00020 readUcs(std::map<std::string, std::vector<classQtd» &count);
00021
00022 std::map<std::string, myUc> readSchedules();
00023
00024 #endif
```

4.31 src/main.cpp File Reference

```
#include "inputoutput/print.h"
#include "inputoutput/read.h"
#include <iostream>
```

Functions

• void menu ()

Display the main menu and handle user options.

• int main ()

4.31.1 Function Documentation

4.31.1.1 main()

```
Definition at line 7 of file main.cpp.

Definition at line 7 of file main.cpp.

00007
00008
00009 menu();
00010
```

return 0;

4.31.1.2 menu()

00011

00012 }

```
void menu ( )
```

Display the main menu and handle user options.

This function displays the main menu of the application and handles user input to perform various actions. Users can choose to view the database, change the database, perform a backup, or exit the application.

Definition at line 26 of file menu.cpp.

```
00026
00027
00028
       menuUpdate();
       system("clear");
00029
00030
00031
       int flag = 0;
00032
       std::cout « "-----" welcome to our app :) -----" « std::endl;
00033
00034
       std::cout « "| 1) See database
                                                                  |" « std::endl;
       std::cout « "| 2) Change database
                                                                   " « std::endl;
00035
00036
       std::cout « "| 3) Backup
                                                                   |" « std::endl;
       std::cout « "| 4) Exit
                                                                   " « std::endl;
00037
       std::cout « "--
                                                                  --" « std::endl;
00038
       std::cout « "Choose an option: ";
00039
00040
       std::cin » flag;
00041
00042
       errorCheck(flag);
00043
00044
       switch (flag) {
00045
       case 1:
       menuSeeDatabase();
break;
00046
00047
00048
       case 2:
00049
       menuRequests();
break;
00050
00051
       case 3:
       menuBackup();
break;
00052
00053
       case 4:
00054
00055
         exit(0);
00056
       errorMessage();
break;
00057
00058
00059
00060 }
```

4.32 main.cpp

Go to the documentation of this file.

```
00001 #include "inputoutput/print.h"
00002 #include "inputoutput/read.h"
00003 #include <iostream>
00004
00005 void menu();
00006
00007 int main() {
00008
00009 menu();
00010
00011 return 0;
00012 }
```

4.33 src/menu.cpp File Reference

```
#include "menu.h"
#include "inputoutput/read.h"
```

Functions

void menuUpdate ()

Update student information.

· void menu ()

Display the main menu and handle user options.

void menuSeeDatabase ()

Display options to view database information.

void menuRequests ()

Display options to change the database.

void menuStudentCode (int flag)

Enter a registration number and access student-related actions.

• void menuTryAgain (int menuType, std::map< std::string, myStudent >::iterator &it)

Display options to try the current operation again or exit.

void menuRemove (std::map< std::string, myStudent >::iterator &it)

Remove a UC from a student's classes.

void menuAdd (std::map< std::string, myStudent >::iterator &it)

Add a new class to a student's schedule.

void menuSwitch (std::map< std::string, myStudent >::iterator &it)

Perform a switch operation for a student's schedule.

· void saveOrReturn ()

Prompt the user to save changes or return to the previous menu.

• void save ()

Save all changes to the student data and exit the program.

int selectBackupCode (int type)

Select a backup for viewing or restoration.

void menuBackup ()

Display the backup menu.

void restoreBackup ()

Restore data from a selected backup.

void menuChanges ()

Display menu options for handling backup changes.

• int selectOrderStudents ()

Prompt the user to select the sorting order for students.

• int selectOrderUcs ()

Prompt the user to select the sorting order for UCs.

int selectType ()

Prompt the user to select the viewing type.

std::string selectCode ()

Prompt the user to enter a code for searching.

• int selectFilter ()

Prompt the user to select a filter for data search.

std::string selectValue ()

Prompt the user to enter a value for filtering data.

• void menuStudents (std::string str, int type, int filter, int order)

Display student data based on specified criteria.

void menuUcs (std::string str, int type, int filter, int order)

Display UC and class data based on specified criteria.

Variables

```
    std::map< std::string, std::vector< classQtd >> count
```

- std::map< std::string, myStudent > students
- std::map< std::string, std::vector< myUc >> ucs = readUcs(count)
- std::map< std::string, myUc > classes = readSchedules()
- std::stack< alter > stackAlter

4.33.1 Function Documentation

4.33.1.1 menu()

```
void menu ( )
```

Display the main menu and handle user options.

This function displays the main menu of the application and handles user input to perform various actions. Users can choose to view the database, change the database, perform a backup, or exit the application.

Definition at line 26 of file menu.cpp.

```
00026
00028
        menuUpdate();
00029
        system("clear");
00030
00031
        int flag = 0;
00032
00033
        std::cout « "-----
                                  --- Welcome to our app :) -----
                                                                          -" « std::endl;
        std::cout « "| 1) See database
std::cout « "| 2) Change database
std::cout « "| 3) Backup
                                                                          " « std::endl;
00034
00035
                                                                          " « std::endl;
00036
                                                                          " « std::endl;
        std::cout « "| 4) Exit
00037
        std::cout « "-
                                                                          -" « std::endl;
00038
00039
        std::cout « "Choose an option: ";
00040
        std::cin » flag;
00041
00042
        errorCheck(flag);
00043
00044
        switch (flag) {
00045
        case 1:
00046
          menuSeeDatabase();
```

```
00047
         break;
00048
       case 2:
       menuRequests();
00049
00050
        break;
00051
       case 3:
       menuBackup();
break;
00052
00054
00055
         exit(0);
00056
       default:
       errorMessage();
00057
00058
         break:
00059
       }
00060 }
```

4.33.1.2 menuAdd()

Add a new class to a student's schedule.

This function allows the user to add a new class to a student's schedule by providing the UC code and the class code. It validates the student's schedule and class availability.

Parameters

it An iterator referring to a specific student.

Definition at line 292 of file menu.cpp.

```
{
00293
       printStudentClasses(it);
       std::string ucCode;
00295
       std::string classCode;
00296
      bool check_class = false;
00297
00298
       // validates if the student is enrolled in more than 7 classes
       if (it->second.valideQtClasses()) {
00299
       std::cout « "-----std::cout « " You have already 7 classes" « std::endl;
                                          ----- « std::endl;
00300
00301
00302
       std::cout « "-----
00303
                                                       ----- « std::endl;
        std::cout « "Enter UC code to see all classes: " « std::endl;
00304
00305
         std::cin » ucCode;
00306
00307
        if (!verifyUcCode(ucCode, it)) {
          // checks if ucCode exists
00308
00309
           auto it_uc = ucs.find(ucCode);
00310
          if (it_uc == ucs.end()) {
00311
           std::cout « "---
00312
00313
                     « std::endl;
00314
            std::cout « "UC code not found" « std::endl;
00315
            menuTryAgain(1, it);
00316
         } else {
00317
00318
            std::cout « "---
00319
                      « std::endl;
00320
            std::cout « "Uc. Code: " « it_uc->first « std::endl;
00321
            00322
00323
                      « std::endl;
00324
             std::cout « "Enter class code to add: " « std::endl;
00325
00326
             std::cin » classCode;
00327
00328
             check_class = verifyClassCode(classCode, ucCode, count);
00329
00330
             if (check class) {
00331
              // validates that the class chosen by the student does not conflict
00332
               // with the schedule of other classes
00333
              bool validate = valideNewClass(ucCode, classCode, it, classes);
00334
              if (!validate) {
00335
```

```
addClassStudent(ucCode, classCode, it, stackAlter);
             printStudentClasses(it);
std::cout « "\nSucessfully added" « std::endl;
00337
00338
00339
             saveOrReturn();
00340
00341
00342
00343
            std::cout « "---
            00344
00345
00346
            menuTryAgain(1, it);
00347
00348
00349
     } else {
        std::cout « "-----
00350
        00351
00352
00353
         menuTryAgain(1, it);
00354
00355
     }
00356 }
```

4.33.1.3 menuBackup()

```
void menuBackup ( )
```

Display the backup menu.

This function lists all available backups, allows the user to select a backup to view changes, and provides options to navigate between viewing changes and returning to the main menu.

Definition at line 550 of file menu.cpp.

```
00550
00551
      int flag;
system("clear");
00552
00553
      listAllBackups();
00554
00555
      bool valide = printAllBackups();
      if (valide == true) {
00556
       printChanges(selectBackupCode(0));
00557
00558
        menuChanges();
      } else {
      std::cout « "------std::cout « "| 1) - Main menu
00560
                                                        ----" « std::endl;
                                                              |" « std::endl;
00561
        std::cout « "---
                         ----- « std::endl;
00562
00563
        std::cin » flag;
00564
00565
       if (flag == 1) {
00566
          menu();
00567
        } else {
00568
          errorMessage();
        }
00569
00570
      }
00571 }
```

4.33.1.4 menuChanges()

```
void menuChanges ( )
```

Display menu options for handling backup changes.

This function presents menu options for the user to manage backup changes, including returning to the previous menu, going back to the main menu, or restoring data from a selected backup.

Definition at line 590 of file menu.cpp.

```
|" « std::endl;
                                                               " « std::endl;
00597
00598
                                                                -" « std::endl;
00599
00600
       std::cin » flag;
00601
00602
       switch (flag) {
00603
       case (1):
       menuBackup();
00604
00605
        break;
00606
      case (2):
        menu();
00607
00608
        break;
00609
      case (3):
       restoreBackup();
break;
00610
00611
00612
       default:
00613
       errorMessage();
break;
00614
00615
      }
00616 }
```

4.33.1.5 menuRemove()

```
void menuRemove (
          std::map< std::string, myStudent >::iterator & it )
```

Remove a UC from a student's classes.

This function allows the user to remove a specific UC from a student's class list. It prompts the user to enter the UC code, removes it from the student's classes, and provides success or error feedback.

Parameters

it An iterator referring to a specific student.

```
Definition at line 262 of file menu.cpp.
```

```
00263
       printStudentClasses(it);
00264
       std::string ucCode;
00265
                                                           ----" « std::endl;
00266
       std::cout « "----
       std::cout « "Enter UC code to remove " « std::endl;
00267
00268
       std::cin » ucCode;
00269
                          -----" « std::endl;
00270
00271
       bool remove = removeUcStudent(ucCode, it, stackAlter, count);
00272
00273
       if (remove) {
       printStudentClasses(it);
std::cout « "\nRemovido com sucesso" « std::endl;
00274
00275
00276
         saveOrReturn();
00277
       } else {
       std::cout « "-----std::cout « "UC code not found" « std::endl;
00278
                                                        ----- « std::endl;
00280
         menuTryAgain(2, it);
00281 }
00282 }
```

4.33.1.6 menuRequests()

```
void menuRequests ( )
```

Display options to change the database.

This function presents a menu allowing the user to choose between adding, removing, or switching database entries. It further provides options for selecting specific actions and database entries.

Definition at line 133 of file menu.cpp.

```
00134
       int flag = 0;
00135
00136
       system("clear");
       std::cout « "Change database" « std::endl;
00137
                                                           ----" « std::endl;
       std::cout « "-
00138
00139
       std::cout « "| 1) Add
                                                               |" « std::endl;
       std::cout « "| 2) Remove
std::cout « "| 3) Switch
                                                                " « std::endl;
00140
                                                                " « std::endl;
00141
       std::cout « "-----" « std::endl;
00142
       std::cout « "Choose an option: ";
00143
       std::cin » flag;
00144
00145
00146
       if (flag > 4 || flag == 0) {
00147
        errorMessage();
       } else {
00148
00149
        menuStudentCode(flag);
00150 }
00151 }
```

4.33.1.7 menuSeeDatabase()

```
void menuSeeDatabase ( )
```

Display options to view database information.

This function presents a menu allowing the user to choose between viewing students, classes and UCs, or their own schedules. It further provides options for selecting display filters, orders, and specific details.

Definition at line 68 of file menu.cpp.

00068

```
00069
        int flag = 0:
00070
        int type;
00071
00072
        std::cout « "| 1) See Students
std::cout « "| 2) See Classes and UC's
std::cout « "| 3) See My Schedules
00073
                                                                      |" « std::endl;
                                                                         " « std::endl;
00074
00075
                                                                         " « std::endl;
        std::cout « "-----
                                                                         --" « std::endl;
00076
00077
        std::cout « "Choose an option: ";
00078
        std::cin » flag;
00079
08000
        errorCheck(flag);
00081
00082
        if (flag != 3) {
          type = selectType();
00083
00084
00085
        // std::cout « type;
00086
00087
        if (type == 1) {
        std::string code = selectCode();
00088
00089
          switch (flag) {
00090
          menuStudents(code, type);
break;
00091
00092
00093
          case 2:
00094
          menuUcs(code, type);
break;
00095
00096
          default:
          errorMessage();
00097
00098
            break;
00099
        } else {
00100
        int filter;
int order;
00101
00102
00103
          std::string value;
          if (type == 2) {
  filter = selectFilter();
00104
00105
            value = selectValue();
00106
00107
00108
          switch (flag) {
          case 1:
00109
           order = selectOrderStudents();
00110
00111
            menuStudents(value, type, filter, order);
00112
            break;
00113
          case 2:
00114
           order = selectOrderUcs();
00115
            menuUcs(value, type, filter, order);
```

4.33.1.8 menuStudentCode()

```
void menuStudentCode ( int\ flag\ )
```

Enter a registration number and access student-related actions.

This function prompts the user to enter their registration number and provides access to various student-related actions, such as adding, removing, switching, or viewing schedules.

Parameters

flag An integer representing the selected action.

```
Definition at line 161 of file menu.cpp.
00162
        std::string registrationNumber;
00163
       std::cout « "-
       std::cout « "Enter your registration number: ";
00164
00165
       std::cin » registrationNumber;
00166
00167
       auto it = students.find(registrationNumber);
00168
00169
       if (it == students.end()) {
                                                   ----- « std::endl;
00170
         std::cout « "---
         std::cout « "| Registration number not found | " « std::endl;
00171
00172
         std::cout « "-
                                                                     --" « std::endl;
00173
         std::cout « "| 1) Try again
std::cout « "| 2) Exit
00174
                                                                      |" « std::endl;
                                                                     " « std::endl;
00175
00176
00177
         int flag2;
00178
00179
         std::cin » flag2;
00180
00181
         switch (flag2) {
00182
         system("clear");
00183
00184
           menuStudentCode(flag);
00185
           break;
00186
         case 2:
00187
           exit(0);
         default:
00188
00189
          errorMessage();
00190
           break;
00191
00192
00193
         menuRequests();
00194
00195
         // printStudentClasses(it);
00196
00197
00198
       switch (flag) {
00199
       case (1):
        menuAdd(it);
00200
00201
         break;
00202
       case (2):
00203
        menuRemove(it);
00204
         break;
00205
       case (3):
       menuSwitch(it);
break;
00206
00207
00208
       case (4):
```

4.33.1.9 menuStudents()

```
void menuStudents (
    std::string str,
    int type,
    int filter,
    int order )
```

Display student data based on specified criteria.

This function displays student data based on specified search criteria, filtering, and ordering.

Parameters

str	A string containing the search term or code.
type	An integer indicating the search type: 1 for one student, 2 for a group, 3 for all students.
filter	An integer specifying the filter type: 1 for UC code, 2 for class code (optional).
order	An integer indicating the order type (optional).

Definition at line 764 of file menu.cpp.

```
00765
        std::map<std::string, myStudent> oneStudent = students;
00766
       std::vector<myStudent> data;
00767
00768
       for (const auto &studentPair : students) {
00769
         data.push_back(studentPair.second);
00770
00771
00772
       if (type == 1) {
       oneStudent = selectStudent(str, oneStudent);
00773
         printStudent(oneStudent);
00775
       } else {
00776
       if (type == 2) {
00777
           data = filterInfoStudent(filter, str, data);
00778
00779
         data = orderInfoStudent(order, data);
         printStudents(data);
00780
00781 }
00782 }
```

4.33.1.10 menuSwitch()

```
void menuSwitch ( {\tt std::map<\ std::string,\ myStudent\ >::iterator\ \&\ it\ )}
```

Perform a switch operation for a student's schedule.

This function allows the user to perform switching operations for a student's schedule, such as switching UCs or classes within a specific UC. It validates the student's current schedule and class availability for the switch.

Parameters

it An iterator referring to a specific student.

```
Definition at line 367 of file menu.cpp.
00368
      printStudentClasses(it);
00369
      std::string ucCode, classCode;
00370
      int flag;
00371
      auto it_uc = ucs.begin();
00372
      std::list<std::string> free_classes;
00373
      bool validate = false;
00374
      bool check_class = false;
00375
      std::cout « "-----" « std::endl;
00376
                                                 |" « std::endl;
      std::cout « "| 1) Switch UC
00377
      std::cout « "| 2) Switch Class
                                                          " « std::endl;
00378
      std::cout « "---
00379
00380
      std::cin » flag;
00381
00382
      switch (flag) {
00383
      case (1):
       std::cout « "-----
                                                       ----- « std::endl;
00384
00385
        std::cout « "Enter UC code to remove: " « std::endl;
00386
        std::cin » ucCode;
00387
00388
        if (verifyUcCode(ucCode, it)) {
00389
00390
          std::cout « "-----
00391
                  « std::endl;
00392
          std::cout « "Enter UC code to add: " « std::endl;
00393
          std::cin » ucCode;
00394
00395
          it uc = ucs.find(ucCode);
00396
00397
          if (it_uc != ucs.end()) {
00398
00399
           printFreeClasses(ucCode, count);
00400
           std::cout « "-----"
00401
00402
                    « std::endl;
            std::cout « "Enter class code to add: " « std::endl;
00404
            std::cin » classCode;
00405
00406
            check_class = verifyClassCode(classCode, ucCode, count);
00407
00408
           if (check class) {
            validate = valideNewClass(ucCode, classCode, it, classes);
if (!validate) {
00409
00410
             removeUcStudent(ucCode, it, stackAlter, count);
00411
00412
              addClassStudent(ucCode, classCode, it, stackAlter);
              printStudentClasses(it);
00413
               std::cout « "\nSuccessfully switched" « std::endl;
00414
00415
               saveOrReturn();
00416
             }
00417
           } else {
             std::cout « "-----
00418
             00419
00420
00421
             menuTryAgain(3, it);
00422
00423
          } else {
          std::cout « "-----
00424
           00425
         ...out « "UC code
menuTryAgain(3, it);
}
00426
00427
00428
00429
        } else {
        std::cout « "-----
00430
         00431
00432
00433
         menuTryAgain(3, it);
00434
00435
00436
        break;
00437
      case (2):
       std::cout « "-----
00438
                                            -----" « std::endl;
        std::cout « "Enter UC to change class: " « std::endl;
00439
00440
        std::cin » ucCode;
00441
00442
        if (verifyUcCode(ucCode, it)) {
00443
00444
          printFreeClasses(ucCode, count);
00445
          std::cout « "---
                  « std::endl;
00446
          std::cout « "Enter class code to add: " « std::endl;
00447
00448
         std::cin » classCode;
00449
00450
          check_class = verifyClassCode(classCode, ucCode, count);
00451
00452
          if (check class) {
```

```
removeUcStudent(ucCode, it, stackAlter, count);
           validate = valideNewClass(ucCode, classCode, it, classes);
00455
          if (!validate) {
00456
           addClassStudent(ucCode, classCode, it, stackAlter);
            printStudentClasses(it);
00457
            std::cout « "\nSuccessfully switched" « std::endl;
00458
00459
            saveOrReturn();
00460
00461
       } else {
         std::cout « "-----
00462
          00463
00464
          menuTryAgain(3, it);
00465
00466
00467
      } else {
       std::cout « "-----
00468
       00469
00470
         menuTryAgain(3, it);
00472
00473
       break;
00474
      default:
00475
      errorMessage();
break;
00476
00477
      }
00478 }
```

4.33.1.11 menuTryAgain()

Display options to try the current operation again or exit.

This function presents a menu allowing the user to choose between trying the current operation again or exiting the menu for adding, removing, or switching database entries.

Parameters

тепиТуре	An integer representing the type of operation (1 for add, 2 for remove, 3 for switch).
it	An iterator referring to a specific database entry.

Definition at line 225 of file menu.cpp.

```
00227
       std::cout « "-----" « std::endl;
00228
       std::cout « "| 1) Try again
                                                             |" « std::endl;
00229
      std::cout « "| 2) Exit
std::cout « "-----
                                                               " « std::endl;
00230
                                                               --" « std::endl;
00231
00232
       std::cin » flag;
00233
00234
       switch (flag) {
00235
       case 1:
       system("clear");
if (menuType == 1) {
00236
00237
          menuAdd(it);
00238
00239
        } else if (menuType == 2) {
00240
          menuRemove(it);
        } else if (menuType == 3) {
00241
       menuSwitch(it);
}
00242
00243
00244
        break;
00245 case 2:
00246
        exit(0);
       default:
00247
00248
       errorMessage();
00249
        break;
00250 }
00251 }
```

4.33.1.12 menuUcs()

```
void menuUcs (
    std::string str,
    int type,
    int filter,
    int order)
```

Display UC and class data based on specified criteria.

This function displays UC and class data based on specified search criteria, filtering, and ordering.

Parameters

str	A string containing the search term or code.
type	An integer indicating the search type: 1 for one UC and its classes, 2 for a group, 3 for all UCs.
filter	An integer specifying the filter type: 1 for UC code, 2 for class code (optional).
order	An integer indicating the order type (optional).

Definition at line 796 of file menu.cpp.

```
00796
00797
        std::vector<myUc> data;
00798
        std::vector<myUc> oneUc;
00799
00800
        for (const auto &ucVectorPair : ucs) {
        for (const myUc &ucObj : ucVectorPair.second) {
00801
            data.push_back(ucObj);
00803
00804
00805
       if (type == 1) {
  oneUc = selectUc(str, classes);
00806
00807
80800
         printUcClasses(oneUc);
00809
       } else {
00810
        if (type == 2) {
00811
           data = filterInfoUc(filter, str, data);
00812
         data = orderInfoUc(order, data);
00813
         printUcs(data);
00814
00815
00816 }
```

4.33.1.13 menuUpdate()

```
void menuUpdate ( )
```

Update student information.

This function updates the student information by reading data from a CSV file and populating the 'students' map. It relies on the 'readStudents' function to perform the data retrieval and update.

```
Definition at line 18 of file menu.cpp.
00018 { students = readStudents(count); }
```

4.33.1.14 restoreBackup()

```
void restoreBackup ( )
```

Restore data from a selected backup.

This function allows the user to choose a backup to restore data from and initiates the restoration process. After restoring the data, the user is returned to the main menu.

Definition at line 579 of file menu.cpp.

4.33.1.15 save()

```
void save ( )
```

Save all changes to the student data and exit the program.

This function saves all the changes made to the student data and exits the program. It uses the "keepAllChanges" function to preserve any modifications, such as adding or switching classes, before exiting.

Definition at line 518 of file menu.cpp.

```
00518
00519 keepAllChanges(students, stackAlter);
00520 exit(0);
00521 }
```

4.33.1.16 saveOrReturn()

```
void saveOrReturn ( )
```

Prompt the user to save changes or return to the previous menu.

This function displays options for the user to either save their changes or return to the previous menu. Users can select to save their actions, which may include adding or switching classes, or choose to return without saving.

Definition at line 487 of file menu.cpp.

```
00488
       int flag = 0;
00489
                                                                  ----" « std::endl;
00490
       std::cout « "-----
                                                                |" « std::endl;
|" « std::endl;
00491
        std::cout « "| 1) Save
00492
       std::cout « "| 2) Return
       std::cout « "---
00493
                                                                  ----" « std::endl;
       std::cout « "Choose an option: ";
00494
       std::cin » flag;
00495
00496
00497
       errorCheck(flag);
00498
00499
       switch (flag) {
00500
       case 1:
        save();
00501
00502
         break;
00503
       case 2:
        menuRequests();
00505
          break;
00506
       default:
        errorMessage();
00507
00508
         break;
00509
00510 }
```

4.33.1.17 selectBackupCode()

Select a backup for viewing or restoration.

Parameters

```
type The type of operation (0 for viewing, 1 for restoration).
```

Returns

The selected backup code to view or restore changes.

Definition at line 530 of file menu.cpp.

```
00530
00531
       int cdBkp;
00532
00533
       if (type == 0) {
00534
         std::cout « "Choose a backup to view changes: ";
       } else if (type == 1) {
00535
00536
         std::cout « "Choose a backup to restore: ";
       }
00537
00538
00539
       std::cin » cdBkp;
00540
00541
       return cdBkp;
00542 }
```

4.33.1.18 selectCode()

```
std::string selectCode ( )
```

Prompt the user to enter a code for searching.

This function displays a prompt to the user and collects a code to use for searching data.

Returns

A string containing the entered code for searching.

Definition at line 700 of file menu.cpp.

```
00700
       std::string str;
std::cout « "---
00701
                                                               ----" « std::endl;
00702
       std::cout « "| 1) Search by code
00703
                                                          |" « std::endl;
                                                           ----" « std::endl;
       std::cout « "
00704
00705
       std::cout « "Enter the code: ";
00706 std::cin » str;
00707
       // errorcheck (str)
00708
00709
       return str;
00710 }
```

4.33.1.19 selectFilter()

```
int selectFilter ( )
```

Prompt the user to select a filter for data search.

This function displays a menu to the user for selecting a filter to apply during data search.

Returns

An integer representing the selected filter:

- 1: Filter by UC Code
- · 2: Filter by Class Code

Definition at line 721 of file menu.cpp.

```
00721
00722
        int flag = 0;
00723
        std::cout « "-----
                                                                     ---" « std::endl;
00724
       std::cout « "| 1) Uc Code
std::cout « "| 2) Class Code
                                                                  |" « std::endl;
00725
00726
                                                                      " « std::endl;
       std::cout « "-
                                                                   ----" « std::endl;
00727
00728
00729
       std::cout « "Choose an option: ";
00730 std::cin » flag;
00731
       errorCheck(flag);
00732
00733
       return flag;
00734 }
```

4.33.1.20 selectOrderStudents()

```
int selectOrderStudents ( )
```

Prompt the user to select the sorting order for students.

This function displays a menu to allow the user to choose the sorting order for the list of students.

Returns

An integer representing the selected sorting order (1: ascending by student code, 2: descending by student code, 3: ascending by student name, 4: descending by student name).

Definition at line 626 of file menu.cpp.

```
00626
00627
         int flag = 0;
00628
00629
         std::cout « "----
                                                                                  -" « std::endl;
         std::cout « "| 1) Sort by student code asc
std::cout « "| 2) Sort by student code desc
00630
                                                                                  " « std::endl;
00631
         std::cout « "| 3) Sort by student name asc std::cout « "| 4) Sort by student name desc
                                                                                  " « std::endl;
00632
                                                                                  " « std::endl;
00633
         std::cout « "
                                                                                   -" « std::endl;
00634
         // add more order like - n° ucs,
00636
        std::cout « "Choose an option: ";
00637
         std::cin » flag;
00638
00639
         errorCheck(flag);
00640
00641
         return flag;
00642 }
```

4.33.1.21 selectOrderUcs()

```
int selectOrderUcs ( )
```

Prompt the user to select the sorting order for UCs.

This function displays a menu to allow the user to choose the sorting order for the list of UCs.

Returns

An integer representing the selected sorting order (1: ascending by UC code, 2: descending by UC code, 3: ascending by class code, 4: descending by class code).

Definition at line 652 of file menu.cpp.

```
00652
00653
          int flag = 0;
00654
00655
         std::cout « "--
                                                                                        -" « std::endl;
          std::cout « "| 1) Sort by uc code asc
          std::cout « "| 2) Sort by uc code desc
std::cout « "| 3) Sort by class code asc
std::cout « "| 4) Sort by class code desc
00657
                                                                                        |" « std::endl;
                                                                                        " « std::endl;
00658
                                                                                        " « std::endl;
00659
          std::cout « "-
                                                                                        -" « std::endl;
00660
         // add more order like - n° ucs,
std::cout « "Choose an option: ";
00661
00662
00663
         std::cin » flag;
00664
00665
         errorCheck(flag);
00666
00667
         return flag;
00668 }
```

4.33.1.22 selectType()

```
int selectType ( )
```

Prompt the user to select the viewing type.

This function displays a menu to allow the user to choose the type of data viewing.

Returns

An integer representing the selected viewing type (1: See one, 2: See a particular group, 3: See all).

Definition at line 677 of file menu.cpp.

```
00677
00678
           int flag = 0;
00679
                                                                                      ----" « std::endl;
00680
          std::cout « "--
          std::cout « "| 1) See one
std::cout « "| 2) See a particular group
std::cout « "| 3) See all
std::cout « "-----
00681
                                                                                              |" « std::endl;
00682
00683
00684
                                                                                                 |" « std::endl;
                                                                                                " « std::endl;
                                                                                           ----" « std::endl;
00685
00686 std::cout « "Choose an option: ";
00687 std::cin » flag;
00688 errorCheck(flag);
00689
00690 return flag;
00691 }
```

4.33.1.23 selectValue()

```
std::string selectValue ( )
```

Prompt the user to enter a value for filtering data.

This function prompts the user to enter a value to be used as a filter during data search.

Returns

A string representing the user-entered value.

Definition at line 743 of file menu.cpp.

4.33.2 Variable Documentation

4.33.2.1 classes

```
std::map<std::string, myUc> classes = readSchedules()
```

Definition at line 7 of file menu.cpp.

4.34 menu.cpp 113

4.33.2.2 count

```
std::map<std::string, std::vector<classQtd> > count
```

Definition at line 4 of file menu.cpp.

4.33.2.3 stackAlter

```
std::stack<alter> stackAlter
```

Definition at line 9 of file menu.cpp.

4.33.2.4 students

```
std::map<std::string, myStudent> students
```

Definition at line 5 of file menu.cpp.

4.33.2.5 ucs

```
std::map<std::string, std::vector<myUc> > ucs = readUcs(count)
```

Definition at line 6 of file menu.cpp.

4.34 menu.cpp

Go to the documentation of this file.

```
00001 #include "menu.h"
00002 #include "inputoutput/read.h"
00003
00004 std::map<std::string, std::vector<classQtd» count;
00005 std::map<std::string, myStudent> students;
00006 std::map<std::string, std::vector<myUc» ucs = readUcs(count);</pre>
00007 std::map<std::string, myUc> classes = readSchedules();
80000
00009 std::stack<alter> stackAlter;
00010
00018 void menuUpdate() { students = readStudents(count); }
00019
00026 void menu() {
00027
00028
       menuUpdate();
00029
       system("clear");
00030
00031
       int flag = 0;
00032
       std::cout « "----- Welcome to our app :) ----- « std::endl;
00033
        std::cout « "| 1) See database
00034
                                                                     |" « std::endl;
        std::cout « "| 2) Change database
std::cout « "| 3) Backup
                                                                      |" « std::endl;
00035
00036
                                                                      " « std::endl;
       std::cout « "| 4) Exit
std::cout « "-----
                                                                      " « std::endl;
00037
                                                                  ----" « std::endl;
00038
       std::cout « "Choose an option: ";
00039
00040
       std::cin » flag;
00041
00042
       errorCheck(flag);
00043
00044
       switch (flag) {
00045
       case 1:
        menuSeeDatabase();
break;
00046
00047
00048
       case 2:
```

```
menuRequests();
00050
          break;
00051
        case 3:
        menuBackup();
00052
00053
        break; case 4:
00054
00055
         exit(0);
00056
        default:
        errorMessage();
break;
00057
00058
        }
00059
00060 }
00061
00068 void menuSeeDatabase() {
00069
        int flag = 0;
00070
        int type;
00071
00072
                                                            ----- « std::endl;
        std::cout « "---
        std::cout « "| 1) See Students
                                                                     |" « std::endl;
        std::cout « "| 2) See Classes and UC's std::cout « "| 3) See My Schedules std::cout « "-----
                                                                         " « std::endl;
00074
00075
                                                                 ----" « std::endl;
00076
        std::cout « "Choose an option: ";
00077
00078
        std::cin » flag;
00079
08000
        errorCheck(flag);
00081
00082
        type = selectType();

        if (flag != 3) {
00083
00084
00085
        // std::cout « type;
00086
00087
        if (type == 1) {
00088
        std::string code = selectCode();
00089
          switch (flag) {
00090
          case 1:
          menuStudents(code, type);
break;
00091
00093
          case 2:
          menuUcs(code, type);
break;
00094
00095
00096
          default:
00097
          errorMessage();
00098
            break;
00099
00100
        } else {
        int filter;
int order;
00101
00102
          std::string value;
00103
00104
          if (type == 2) {
          filter = selectFilter();
value = selectValue();
00105
00106
00107
00108
          switch (flag) {
00109
          case 1:
          order = selectOrderStudents();
menuStudents(value, type, filter, order);
00110
00111
00112
            break;
00113
          case 2:
          order = selectOrderUcs();
menuUcs(value, type, filter, order);
00114
00115
00116
            break;
00117
          case 3:
          menuStudentCode(4);
00118
00119
            break;
00120
          default:
00121
           errorMessage();
00122
            break:
00123
          }
00124
        }
00125 }
00126
00133 void menuRequests() {
00134
        int flag = 0;
00135
00136
        system("clear");
00137
        std::cout « "Change database" « std::endl;
        std::cout « "----
00138
                                                                        --- « std::endl;
                                                                        |" « std::endl;
        std::cout « "| 1) Add
00139
        std::cout « "| 2) Remove
                                                                         " « std::endl;
00140
        std::cout « "| 3) Switch
                                                                         " « std::endl;
00141
                                                                        --" « std::endl;
00142
        std::cout « "----
00143
        std::cout « "Choose an option: ";
00144
        std::cin » flag;
00145
        if (flag > 4 || flag == 0) {
  errorMessage();
00146
00147
```

4.34 menu.cpp 115

```
00148 } else {
00149
         menuStudentCode(flag);
00150 }
00151 }
00152
00161 void menuStudentCode(int flag) {
00162 std::string registrationNumber;
00163
       std::cout « "---
                                                       ----- « std::endl;
       std::cout « "Enter your registration number: ";
00164
00165
       std::cin » registrationNumber;
00166
00167
       auto it = students.find(registrationNumber);
00168
00169
       if (it == students.end()) {
        std::cout « "---
00170
                                                 found | " « std::endl;
         std::cout « "| Registration number not found
00171
         std::cout « "--
00172
00173
         std::cout « "| 1) Try again
                                                                     |" « std::endl;
00175
         std::cout « "| 2) Exit
                                                                     " « std::endl;
00176
00177
         int flag2;
00178
00179
         std::cin » flag2;
00180
         switch (flag2) {
00181
00182
         system("clear");
menuStudentCode(flag);
break;
00183
00184
00185
00186
         case 2:
00187
           exit(0);
00188
         default:
         errorMessage();
break;
00189
00190
00191
00192
00193
         menuRequests();
00194
       } else {
00195
         // printStudentClasses(it);
00196
00197
       switch (flag) {
00198
00199
       case (1):
       menuAdd(it);
00200
00201
         break;
00202
       case (2):
       menuRemove(it);
break;
00203
00204
00205
       case (3):
       menuSwitch(it);
break;
00206
00207
00208
       case (4):
       printStudentSchedules(it, classes);
00209
00210
         break:
00211
       default:
00212
         errorMessage();
00213
00214 }
00215
00225 void menuTryAgain(int menuType,
00226
                      std::map<std::string, myStudent>::iterator &it) {
00227
       int flag;
00228
       std::cout « "-----
                                                                ----" « std::endl;
       std::cout « "| 1) Try again
std::cout « "| 2) Exit
std::cout « "-----
00229
                                                                  |" « std::endl;
                                                                  " « std::endl;
00230
                                  ----- « std::endl;
00231
00232
       std::cin » flag;
00233
00234
       switch (flag) {
00235
       case 1:
       system("clear");
if (menuType == 1) {
  menuAdd(it);
00236
00237
00238
       } else if (menuType == 2) {
  menuRemove(it);
00239
00240
       } else if (menuType == 3) {
00241
00242
          menuSwitch(it);
00243
00244
         break:
00245
       case 2:
00246
         exit(0);
       default:
00247
       errorMessage();
break;
00248
00249
00250
       }
00251 }
```

```
00252
00262 void menuRemove(std::map<std::string, myStudent>::iterator &it) {
00263
      printStudentClasses(it);
00264
       std::string ucCode;
00265
00266
                                                       ----- « std::endl;
       std::cout «
       std::cout « "Enter UC code to remove " « std::endl;
00267
00268
       std::cin » ucCode;
                             -----" « std::endl;
00269
       std::cout « "----
00270
00271
       bool remove = removeUcStudent(ucCode, it, stackAlter, count);
00272
00273
       if (remove) {
       printStudentClasses(it);
std::cout « "\nRemovido com sucesso" « std::endl;
00274
00275
00276
        saveOrReturn();
00277
       } else {
00278
       std::cout « "-----std::cout « "UC code not found" « std::endl;
                                                    ----" « std::endl;
00279
00280
        menuTryAgain(2, it);
00281 }
00282 }
00283
00292 void menuAdd(std::map<std::string, myStudent>::iterator &it) {
00293
      printStudentClasses(it);
       std::string ucCode;
00294
       std::string classCode;
00295
00296
      bool check_class = false;
00297
00298
       // validates if the student is enrolled in more than 7 classes
00299
       if (it->second.valideOtClasses()) {
       std::cout « "-
                                                -----" « std::endl;
00300
         std::cout « " You have already 7 classes" « std::endl;
00301
00302
       } else {
       std::cout « "-----
                                                         ----- « std::endl;
00303
        std::cout « "Enter UC code to see all classes: " « std::endl;
00304
00305
        std::cin » ucCode;
00307
         if (!verifyUcCode(ucCode, it)) {
00308
          // checks if ucCode exists
00309
          auto it_uc = ucs.find(ucCode);
00310
          if (it uc == ucs.end()) {
00311
00312
            std::cout « "---
            00313
00314
00315
            menuTryAgain(1, it);
00316
         } else {
00317
            std::cout « "----
00318
00319
                     « std::endl;
            std::cout « "Uc. Code: " « it_uc->first « std::endl;
00320
00321
00322
            printFreeClasses(ucCode, count);
            std::cout « "----
00323
00324
                     « std::endl;
             std::cout « "Enter class code to add: " « std::endl;
00325
00326
            std::cin » classCode;
00327
00328
             check_class = verifyClassCode(classCode, ucCode, count);
00329
00330
            if (check class) {
00331
              // validates that the class chosen by the student does not conflict
00332
               // with the schedule of other classes
00333
              bool validate = valideNewClass(ucCode, classCode, it, classes);
00334
00335
              if (!validate) {
               addClassStudent(ucCode, classCode, it, stackAlter);
00336
00337
                printStudentClasses(it);
                std::cout « "\nSucessfully added" « std::endl;
00338
00339
00340
                saveOrReturn();
00341
            } else {
00342
00343
              std::cout « "----
00344
                      « std::endl;
00345
               std::cout « "Class code not found" « std::endl;
00346
              menuTryAgain(1, it);
00347
            }
          }
00348
        } else {
00349
          std::cout « "----
00350
           00351
00352
00353
          menuTryAgain(1, it);
00354
00355 }
```

4.34 menu.cpp 117

```
00356 }
00357
00367 void menuSwitch(std::map<std::string, myStudent>::iterator &it) {
00368 printStudentClasses(it);
00369
       std::string ucCode, classCode;
00370
      int flag;
00371
      auto it_uc = ucs.begin();
00372
      std::list<std::string> free_classes;
00373
      bool validate = false;
00374
      bool check_class = false;
00375
      std::cout « "-----
                                          ----- « std::endl;
00376
      std::cout « "| 1) Switch UC
std::cout « "| 2) Switch Class
                                                           |" « std::endl;
|" « std::endl;
00377
00378
      std::cout « "-----
                                                       ----" « std::endl;
00379
00380
      std::cin » flag;
00381
00382
      switch (flag) {
      case (1):
00383
       std::cout « "---
00384
                                                       ----- « std::endl;
        std::cout « "Enter UC code to remove: " « std::endl;
00385
00386
        std::cin » ucCode;
00387
00388
        if (verifyUcCode(ucCode, it)) {
00389
00390
         std::cout « "-----
                   « std::endl;
00391
00392
          std::cout « "Enter UC code to add: " « std::endl;
00393
          std::cin » ucCode;
00394
00395
          it uc = ucs.find(ucCode);
00396
00397
          if (it_uc != ucs.end()) {
00398
00399
           printFreeClasses(ucCode, count);
00400
           std::cout « "---
00401
00402
                     « std::endl;
00403
            std::cout « "Enter class code to add: " « std::endl;
00404
            std::cin » classCode;
00405
00406
            check class = verifyClassCode(classCode, ucCode, count);
00407
00408
            if (check_class) {
             validate = valideNewClass(ucCode, classCode, it, classes);
00409
00410
              if (!validate) {
00411
              removeUcStudent(ucCode, it, stackAlter, count);
00412
               addClassStudent(ucCode, classCode, it, stackAlter);
               printStudentClasses(it);
00413
               std::cout « "\nSuccessfully switched" « std::endl;
00414
00415
               saveOrReturn();
00416
            } else {
00417
            std::cout « "-----"
00418
                      « std::endl;
00419
00420
              std::cout « "Class code not found" « std::endl;
00421
             menuTryAgain(3, it);
00422
00423
        } else {
           std::cout « "-----
00424
            00425
00426
00427
            menuTryAgain(3, it);
00428
00429
        } else {
00430
        std::cout « "-----"
00431
          00432
00433
          menuTrvAgain(3, it);
        }
00434
00435
00436
00437
       case (2):
        std::cout « "-----
                                             ----- « std::endl;
00438
        std::cout « "Enter UC to change class: " « std::endl;
00439
00440
        std::cin » ucCode;
00441
00442
        if (verifyUcCode(ucCode, it)) {
00443
00444
          printFreeClasses(ucCode, count):
          std::cout « "----
00445
00446
                   « std::endl;
00447
          std::cout « "Enter class code to add: " « std::endl;
00448
          std::cin » classCode;
00449
00450
          check class = verifyClassCode(classCode, ucCode, count);
00451
```

```
if (check_class) {
           removeUcStudent(ucCode, it, stackAlter, count);
00453
00454
             validate = valideNewClass(ucCode, classCode, it, classes);
00455
            if (!validate) {
00456
              addClassStudent(ucCode, classCode, it, stackAlter);
00457
               printStudentClasses(it);
               std::cout « "\nSuccessfully switched" « std::endl;
00459
               saveOrReturn();
00460
           } else {
00461
            std::cout « "-----
00462
                      « std::endl;
00463
             std::cout « "Class code not found" « std::endl;
00464
00465
            menuTryAgain(3, it);
00466
           }
00467
       } else {
        std::cout « "-----
00468
00469
                    « std::endl;
           std::cout « "You are not enrolled in this UC" « std::endl;
00471
           menuTryAgain(3, it);
00472
00473
         break;
       default:
00474
00475
        errorMessage();
00476
         break;
00477
00478 }
00479
00487 void saveOrReturn() {
00488
       int flag = 0;
00489
00490
       std::cout « "---
                                                       ----- « std::endl;
00491
       std::cout « "| 1) Save
                                                              |" « std::endl;
       std::cout « "| 2) Return
std::cout « "-----
                                                                  " « std::endl;
00492
                                                            ----" « std::endl;
00493
       std::cout « "Choose an option: ";
00494
00495
       std::cin » flag;
00496
00497
       errorCheck(flag);
00498
00499
       switch (flag) {
00500
       case 1:
00501
       save();
break;
00502
00503
       case 2:
       menuRequests();
break;
00504
00505
00506
       default:
00507
       errorMessage();
break;
00508
00509
       }
00510 }
00511
00518 void save() {
00519 keepAllChanges(students, stackAlter);
00520
       exit(0);
00521 }
00522
00530 int selectBackupCode(int type) {
00531
       int cdBkp;
00532
       if (type == 0) {
  std::cout « "Choose a backup to view changes: ";
00533
00534
00535
       } else if (type == 1) {
00536
         std::cout « "Choose a backup to restore: ";
       }
00537
00538
00539
       std::cin » cdBkp;
00540
00541
       return cdBkp;
00542 }
00543
00550 void menuBackup() {
00551 int flag;
00552 system("clear");
00553
       listAllBackups();
00554
00555
       bool valide = printAllBackups();
       if (valide == true) {
00556
       printChanges(selectBackupCode(0));
00557
00558
         menuChanges();
       } else {
       std::cout « "---
                                                               -----" « std::endl;
|" « std::endl;
00560
         std::cout « "| 1) - Main menu
std::cout « "-----
00561
                                                                ----" « std::endl;
00562
00563
         std::cin » flag;
00564
```

4.34 menu.cpp 119

```
if (flag == 1) {
00566
            menu();
00567
          } else {
           errorMessage();
00568
00569
          }
00570
        }
00571 }
00572
00579 void restoreBackup() {
00580 backupFile(selectBackupCode(1));
00581
        menu();
00582 }
00583
00590 void menuChanges() {
00591
00592
        int flag;
00593
                                                                   ----" « std::endl;
00594
        std::cout « "---
        std::cout « "| 1) Return
                                                                       |" « std::endl;
        std::cout « "| 2) Main menu
std::cout « "| 3) Restore
std::cout « "----
00596
                                                                          |" « std::endl;
                                                                           " « std::endl;
00597
                                                                  ----" « std::endl;
00598
00599
00600
        std::cin » flag;
00601
        switch (flag) {
00602
        case (1):
00603
        menuBackup();
00604
00605
          break;
00606
        case (2):
00607
        menu();
00608
          break;
00609
        case (3):
        restoreBackup();
break;
00610
00611
00612
        default:
        errorMessage();
break;
00613
00614
00615
00616 }
00617
00626 int selectOrderStudents() {
00627
        int flag = 0;
00628
00629
        std::cout « "| 1) Sort by student code asc
                                                             |" « std::endl;
|" « std::endl;
00630
        std::cout « "| 2) Sort by student code desc std::cout « "| 3) Sort by student name asc std::cout « "| 4) Sort by student name desc
00631
                                                                          " « std::endl;
00632
                                                                         " « std::endl;
00633
                                                                    ----" « std::endl;
        std::cout « "-----
00634
        // add more order like - n° ucs,
std::cout « "Choose an option: ";
00635
00636
00637
        std::cin » flag;
00638
00639
        errorCheck(flag);
00640
00641
        return flag;
00642 }
00643
00652 int selectOrderUcs() {
00653
        int flag = 0;
00654
00655
        std::cout « "--
                                                                    ----" « std::endl;
        std::cout « "| 1) Sort by uc code asc
                                                                     |" « std::endl;
|" « std::endl;
00656
        std::cout « "| 2) Sort by uc code desc
00657
        std::cout « "| 3) Sort by class code asc
                                                                          " « std::endl;
00658
        std::cout « "| 3) Sort by class code asc std::cout « "| 4) Sort by class code desc std::cout « "-----
                                                                      | " « std::endl;
00659
00660
00661
        // add more order like - n° ucs,
        std::cout « "Choose an option: ";
00662
00663
        std::cin » flag;
00664
00665
        errorCheck(flag);
00666
00667
        return flag;
00668 }
00669
00677 int selectType() {
00678
        int flag = 0;
00679
        std::cout « "-
                                                                           " « std::endl;
00680
                                                                       " « std::endl;
|" « std::endl;
        std::cout « "| 1) See one
std::cout « "| 2) See a particular group
std::cout « "| 3) See all
00681
00682
                                                                          " « std::endl;
00683
        std::cout « "----- « std::endl;
00684
00685
00686
        std::cout « "Choose an option: ";
```

```
std::cin » flag;
00688
       errorCheck(flag);
00689
00690
       return flag;
00691 }
00692
00700 std::string selectCode() {
00701
       std::string str;
00702
       std::cout « "----
                                                  ----- « std::endl;
       std::cout « "| 1) Search by code
                                                             |" « std::endl;
00703
       std::cout « "-----" « std::endl;
00704
       std::cout « "Enter the code: ";
00705
00706
       std::cin » str;
       // errorcheck (str)
00707
00708
00709
      return str;
00710 }
00711
00721 int selectFilter() {
00722
       int flag = 0;
00723
00724
       std::cout « "-----" « std::endl;
       std::cout « "| 1) Uc Code
std::cout « "| 2) Class Code
                                               " « std::endl;
00725
                                                              |" « std::endl;
00726
00727
       std::cout « "-----" « std::endl;
00728
00729
       std::cout « "Choose an option: ";
00730 std::cin » flag;
00731
       errorCheck(flag);
00732
00733
       return flag:
00734 }
00735
00743 std::string selectValue() {
00744 std::string str;
00745
00746
      std::cout « "Enter the value: ";
00747
       std::cin » str;
00748
      // errorcheck (str)
00749
00750
       return str;
00751 }
00752
00764 void menuStudents(std::string str, int type, int filter, int order) {
00765 std::map<std::string, myStudent> oneStudent = students;
00766
       std::vector<myStudent> data;
00767
00768
       for (const auto &studentPair : students) {
00769
        data.push_back(studentPair.second);
00770
00771
00772
       if (type == 1) {
       oneStudent = selectStudent(str, oneStudent);
00773
00774
         printStudent(oneStudent);
00775
       } else {
       if (type == 2) {
00776
00777
          data = filterInfoStudent(filter, str, data);
00778
00779
        data = orderInfoStudent(order, data);
00780
         printStudents(data);
00781
       1
00782 }
00783
00784
00796 void menuUcs(std::string str, int type, int filter, int order) {
00797 std::vector<myUc> data;
00798
       std::vector<myUc> oneUc;
00799
00800
       for (const auto &ucVectorPair : ucs) {
       for (const myUc &ucObj : ucVectorPair.second) {
00801
00802
           data.push_back(ucObj);
00803
00804
       }
00805
00806
       if (type == 1) {
00807
       oneUc = selectUc(str, classes);
80800
         printUcClasses(oneUc);
00809
         else {
00810
       if (type == 2) {
00811
          data = filterInfoUc(filter, str, data);
00812
         data = orderInfoUc(order, data);
00814
        printUcs(data);
00815
00816 }
```

4.35 src/menu.h File Reference

```
#include <iostream>
#include <list>
#include <map>
#include <stack>
#include "classes/student.h"
#include "classes/uc.h"
#include "functions/dbStudents.h"
#include "functions/dbUcs.h"
#include "inputoutput/keepAllChanges.h"
#include "inputoutput/print.h"
#include "inputoutput/read.h"
```

Functions

- void errorMessage ()
- void errorCheck (int n)
- void menuStudents (std::string str="", int type=0, int filter=0, int order=0)

Display student data based on specified criteria.

• void menuUcs (std::string str="", int type=0, int filter=0, int order=0)

Display UC and class data based on specified criteria.

void menuStudentCode (int flag)

Enter a registration number and access student-related actions.

void menuTryAgain (int menuType, std::map< std::string, myStudent >::iterator &it)

Display options to try the current operation again or exit.

· void menu ()

Display the main menu and handle user options.

• void menuSeeDatabase ()

Display options to view database information.

• void menuRequests ()

Display options to change the database.

void menuRemove (std::map< std::string, myStudent >::iterator &it)

Remove a UC from a student's classes.

void menuAdd (std::map< std::string, myStudent >::iterator &it)

Add a new class to a student's schedule.

void menuSwitch (std::map< std::string, myStudent >::iterator &it)

Perform a switch operation for a student's schedule.

• void menuBackup ()

Display the backup menu.

• void menuChanges ()

Display menu options for handling backup changes.

void restoreBackup ()

Restore data from a selected backup.

- int selectBackupCode ()
- int selectOrderStudents ()

Prompt the user to select the sorting order for students.

• int selectOrderUcs ()

Prompt the user to select the sorting order for UCs.

int selectType ()

Prompt the user to select the viewing type.

• int selectFilter ()

Prompt the user to select a filter for data search.

std::string selectCode ()

Prompt the user to enter a code for searching.

std::string selectValue ()

Prompt the user to enter a value for filtering data.

void saveOrReturn ()

Prompt the user to save changes or return to the previous menu.

• void save ()

Save all changes to the student data and exit the program.

4.35.1 Function Documentation

4.35.1.1 errorCheck()

```
void errorCheck ( \inf \ n \ )
```

Definition at line 9 of file errorMsgs.cpp.

4.35.1.2 errorMessage()

```
void errorMessage ( )
```

Definition at line 4 of file errorMsgs.cpp.

```
00004 {
00005 std::cout « "ERROR: Invalid choice." « std::endl;
00006 exit(0);
00007 }
```

4.35.1.3 menu()

```
void menu ( )
```

Display the main menu and handle user options.

This function displays the main menu of the application and handles user input to perform various actions. Users can choose to view the database, change the database, perform a backup, or exit the application.

Definition at line 26 of file menu.cpp.

```
00026
00027
00028
       menuUpdate();
00029
       system("clear");
00030
00031
       int flag = 0;
00032
       std::cout « "-----" « std::endl;
00033
      std::cout « "| 1) See database
std::cout « "| 2) Change database
                                                               " « std::endl;
00034
00035
                                                                " « std::endl;
00036
       std::cout « "| 3) Backup
```

```
|" « std::endl;
00037
        std::cout « "| 4) Exit
        std::cout « "-
00038
        std::cout « "Choose an option: ";
00039
       std::cin » flag;
00040
00041
00042
        errorCheck(flag);
00043
00044
        switch (flag) {
00045
       case 1:
        menuSeeDatabase();
00046
00047
         break;
00048
       case 2:
       menuRequests();
break;
00049
00050
00051
       case 3:
        menuBackup();
break;
00052
00053
00054
       case 4:
00055
         exit(0);
00056
       default:
       errorMessage();
break;
00057
00058
00059 }
00060 }
```

4.35.1.4 menuAdd()

Add a new class to a student's schedule.

This function allows the user to add a new class to a student's schedule by providing the UC code and the class code. It validates the student's schedule and class availability.

Parameters

it An iterator referring to a specific student.

Definition at line 292 of file menu.cpp.

```
00292
00293
       printStudentClasses(it);
00294
       std::string ucCode;
00295
       std::string classCode;
00296
       bool check class = false;
00297
       00298
       // validates if the student is enrolled in more than 7 classes
00299
                                                ----- « std::endl;
00300
         std::cout « " You have already 7 classes" « std::endl;
00301
00302
       } else {
       std::cout « "-----
00303
                                                               ---" « std::endl;
00304
        std::cout « "Enter UC code to see all classes: " « std::endl;
00305
        std::cin » ucCode;
00306
00307
        if (!verifyUcCode(ucCode, it)) {
          // checks if ucCode exists
00308
00309
          auto it_uc = ucs.find(ucCode);
00310
00311
          if (it_uc == ucs.end()) {
00312
           std::cout « "-
00313
                     « std::endl;
            std::cout « "UC code not found" « std::endl;
00314
00315
            menuTryAgain(1, it);
00316
00317
          } else {
            std::cout « "----
00318
                      « std::endl;
00319
             std::cout « "Uc. Code: " « it_uc->first « std::endl;
00320
00321
00322
             printFreeClasses(ucCode, count);
00323
             std::cout « "-
00324
                      « std::endl;
             std::cout « "Enter class code to add: " « std::endl;
00325
```

```
std::cin » classCode;
00328
            check_class = verifyClassCode(classCode, ucCode, count);
00329
00330
            if (check class) {
            // validates that the class chosen by the student does not conflict
00331
              // with the schedule of other classes
00332
00333
              bool validate = valideNewClass(ucCode, classCode, it, classes);
00334
             if (!validate) {
00335
              addClassStudent(ucCode, classCode, it, stackAlter);
00336
                printStudentClasses(it);
00337
00338
                std::cout « "\nSucessfully added" « std::endl;
00339
00340
                saveOrReturn();
00341
            } else {
00342
              std::cout « "-
00343
00344
                       « std::endl;
00345
              std::cout « "Class code not found" « std::endl;
00346
              menuTryAgain(1, it);
00347
            }
          }
00348
      } else {
   std::c
00349
00350
          std::cout « "-
00351
          00352
00353
          menuTryAgain(1, it);
00354
00355 }
00356 }
```

4.35.1.5 menuBackup()

```
void menuBackup ( )
```

Display the backup menu.

This function lists all available backups, allows the user to select a backup to view changes, and provides options to navigate between viewing changes and returning to the main menu.

Definition at line 550 of file menu.cpp.

```
00550
00551
       int flag;
system("clear");
00552
00553
       listAllBackups();
00554
00555
      bool valide = printAllBackups();
      if (valide == true) {
00556
       printChanges(selectBackupCode(0));
00557
00558
         menuChanges();
00559
      } else {
        std::cout « "-----
00560
                                                         ----- « std::endl;
       std::cout « "| 1) - Main menu
std::cout « "-----
                                                                  |" « std::endl;
00561
                                                               ----" « std::endl;
00562
         std::cin » flag;
00563
00564
00565
        if (flag == 1) {
00566
          menu();
00567
         } else {
00568
           errorMessage();
         }
00569
00570
       }
00571 }
```

4.35.1.6 menuChanges()

```
void menuChanges ( )
```

Display menu options for handling backup changes.

This function presents menu options for the user to manage backup changes, including returning to the previous menu, going back to the main menu, or restoring data from a selected backup.

```
Definition at line 590 of file menu.cpp.
```

```
00591
00592
       int flag;
00593
                                                      ----" « std::endl;
00594
       std::cout « "---
00595
       std::cout « "| 1) Return
                                                                 |" « std::endl;
       std::cout « "| 2) Main menu
00596
                                                                   |" « std::endl;
       std::cout « "| 3) Restore
                                                                  |" « std::endl;
00597
       std::cout « "--
                                                              ----" « std::endl;
00598
00599
00600
       std::cin » flag;
00601
00602
       switch (flag) {
00603
       case (1):
        menuBackup();
00604
00605
         break;
       case (2):
00606
       menu();
00607
00608
         break;
00609
       case (3):
       restoreBackup();
break;
00610
00611
00612
       default:
00613
       errorMessage();
break;
00614
00615
00616 }
```

4.35.1.7 menuRemove()

```
void menuRemove (
          std::map< std::string, myStudent >::iterator & it )
```

Remove a UC from a student's classes.

This function allows the user to remove a specific UC from a student's class list. It prompts the user to enter the UC code, removes it from the student's classes, and provides success or error feedback.

Parameters

it An iterator referring to a specific student.

```
Definition at line 262 of file menu.cpp.
```

```
00262
                                                                  {
        printStudentClasses(it);
00263
00264
       std::string ucCode;
00265
00266
       std::cout « "----
       std::cout « "Enter UC code to remove " « std::endl;
00267
00268
       std::cin » ucCode;
                                   -----" « std::endl;
00269
       std::cout « "--
00270
00271
       bool remove = removeUcStudent(ucCode, it, stackAlter, count);
00272
00273
       if (remove) {
       printStudentClasses(it);
std::cout « "\nRemovido com sucesso" « std::endl;
saveOrPotyme().
00274
00275
00276
         saveOrReturn();
00277
       } else {
       std::cout « "------std::cout « "UC code not found" « std::endl;
                                                      -----" « std::endl;
00278
00279
00280
         menuTryAgain(2, it);
00281
00282 }
```

4.35.1.8 menuRequests()

```
void menuRequests ( )
```

Display options to change the database.

This function presents a menu allowing the user to choose between adding, removing, or switching database entries. It further provides options for selecting specific actions and database entries.

Definition at line 133 of file menu.cpp.

```
00133
00134
        int flag = 0;
00135
00136
        system("clear");
        std::cout « "Change database" « std::endl;
00137
        std::cout « "---
                                                                        ---" « std::endl;
00138
        std::cout « "| 1) Add
                                                                         |" « std::endl;
00139
        std::cout « "| 2) Remove
std::cout « "| 3) Switch
std::cout « "-----
00140
                                                                          " « std::endl;
                                                                          " « std::endl;
00141
                                                                         --" « std::endl;
00142
        std::cout « "Choose an option: ";
00143
00144
        std::cin » flag;
00146
        if (flag > 4 || flag == 0) {
00147
          errorMessage();
        } else {
00148
00149
          menuStudentCode(flag);
        }
00150
00151 }
```

4.35.1.9 menuSeeDatabase()

```
void menuSeeDatabase ( )
```

Display options to view database information.

This function presents a menu allowing the user to choose between viewing students, classes and UCs, or their own schedules. It further provides options for selecting display filters, orders, and specific details.

Definition at line 68 of file menu.cpp.

```
00068
00069
         int flag = 0;
00070
        int type;
00071
00072
        std::cout « "----
        std::cout « "| 1) See Students
std::cout « "| 2) See Classes and UC's
std::cout « "| 3) See My Schedules
                                                                      |" « std::endl;
00073
                                                                          " « std::endl;
00074
00075
                                                                          " « std::endl;
        std::cout « "-
                                                                           -" « std::endl;
00076
00077
        std::cout « "Choose an option: ";
00078
        std::cin » flag;
00079
00080
        errorCheck(flag);
00081
00082
        if (flag != 3) {
00083
          type = selectType();
00084
00085
        // std::cout « type;
00086
00087
        if (type == 1) {
00088
         std::string code = selectCode();
00089
          switch (flag) {
00090
00091
           menuStudents(code, type);
00092
            break;
00093
          case 2:
           menuUcs(code, type);
00094
00095
             break;
00096
00097
           errorMessage();
00098
             break;
00099
        } else {
00100
00101
          int filter;
00102
          int order;
00103
           std::string value;
          if (type == 2) {
  filter = selectFilter();
00104
00105
00106
            value = selectValue();
00107
00108
           switch (flag) {
```

```
case 1:
         order = selectOrderStudents();
menuStudents('...'
00110
00111
            menuStudents(value, type, filter, order);
00112
            break;
00113
         case 2:
         order = selectOrderUcs();
menuUcs(value, type, filter, order);
00114
00115
00116
            break;
00117
        case 3:
         menuStudentCode(4);
00118
00119
            break:
00120
         default:
         errorMessage();
break;
00121
00122
00123
00124 }
00125 }
```

4.35.1.10 menuStudentCode()

Enter a registration number and access student-related actions.

This function prompts the user to enter their registration number and provides access to various student-related actions, such as adding, removing, switching, or viewing schedules.

Parameters

flag An integer representing the selected action.

Definition at line 161 of file menu.cpp.

```
00161
00162
        std::string registrationNumber;
                                                         ----- « std::endl;
00163
        std::cout « "-----
        std::cout « "Enter your registration number: ";
00164
        std::cin » registrationNumber;
00165
00166
00167
        auto it = students.find(registrationNumber);
00168
00169
        if (it == students.end()) {
         std::cout « "| Registration number not found | " « std::endl; std::cout « "------
        std::cout « "-----
00170
00171
00172
00173
         std::cout « "| 1) Try again
std::cout « "| 2) Exit
00174
                                                                       |" « std::endl;
                                                                       " « std::endl;
00175
00176
00177
          int flag2;
00178
          std::cin » flag2;
00180
00181
          switch (flag2) {
00182
          case 1:
          system("clear");
menuStudentCode(flag);
00183
00184
00185
            break;
00186
          case 2:
00187
            exit(0);
          default:
00188
00189
          errorMessage();
00190
            break;
00191
00192
00193
          menuRequests();
00194
          // printStudentClasses(it);
00195
00196
00197
00198
        switch (flag) {
00199
        case (1):
        menuAdd(it);
00200
00201
          break;
```

```
case (2):
        menuRemove(it);
break;
00203
00204
00205
        case (3):
        menuSwitch(it);
00206
00207
          break:
        case (4):
        printStudentSchedules(it, classes);
break;
00209
00210
uu212 errorMessage();
00213 }
00213 }
00214 }
```

4.35.1.11 menuStudents()

```
void menuStudents (
    std::string str,
    int type,
    int filter,
    int order )
```

Display student data based on specified criteria.

This function displays student data based on specified search criteria, filtering, and ordering.

Parameters

str	A string containing the search term or code.
type	An integer indicating the search type: 1 for one student, 2 for a group, 3 for all students.
filter	An integer specifying the filter type: 1 for UC code, 2 for class code (optional).
order	An integer indicating the order type (optional).

Definition at line 764 of file menu.cpp.

```
00764
00765
       std::map<std::string, myStudent> oneStudent = students;
00766
      std::vector<myStudent> data;
00767
00768
      for (const auto &studentPair : students) {
00769
       data.push_back(studentPair.second);
00771
      00772
00773
00774
        printStudent(oneStudent);
00775
      } else {
       if (type == 2) {
00777
          data = filterInfoStudent(filter, str, data);
00778
00779
        data = orderInfoStudent(order, data);
00780
        printStudents(data);
00781 }
00782 }
```

4.35.1.12 menuSwitch()

Perform a switch operation for a student's schedule.

This function allows the user to perform switching operations for a student's schedule, such as switching UCs or classes within a specific UC. It validates the student's current schedule and class availability for the switch.

Parameters

it An iterator referring to a specific student.

```
Definition at line 367 of file menu.cpp.
00368
       printStudentClasses(it);
00369
       std::string ucCode, classCode;
00370
      int flag;
      auto it_uc = ucs.begin();
00371
      std::list<std::string> free_classes;
00372
00373
      bool validate = false;
00374
      bool check_class = false;
00375
      std::cout « "-----" « std::endl;
00376
      std::cout « "| 1) Switch UC
std::cout « "| 2) Switch Class
std::cout « "-----
                                                  |" « std::endl;
|" « std::endl;
00377
00378
                                                    ----" « std::endl;
00379
00380
      std::cin » flag;
00381
00382
       switch (flag) {
00383
      case (1):
       std::cout « "-----
                                             ----- « std::endl;
00384
        std::cout « "Enter UC code to remove: " « std::endl;
00385
00386
        std::cin » ucCode;
00387
00388
        if (verifyUcCode(ucCode, it)) {
00389
          std::cout « "-----
00390
00391
                   « std::endl;
          std::cout « "Enter UC code to add: " « std::endl;
00392
00393
          std::cin » ucCode;
00394
00395
          it_uc = ucs.find(ucCode);
00396
          if (it uc != ucs.end()) {
00397
00398
00399
           printFreeClasses(ucCode, count);
00400
00401
            std::cout « "-----"
00402
                     « std::endl;
            std::cout « "Enter class code to add: " « std::endl;
00403
00404
            std::cin » classCode;
00405
00406
            check_class = verifyClassCode(classCode, ucCode, count);
00407
00408
            if (check class) {
00409
              validate = valideNewClass(ucCode, classCode, it, classes);
            if (!validate) {
00410
              removeUcStudent(ucCode, it, stackAlter, count);
00411
00412
                addClassStudent(ucCode, classCode, it, stackAlter);
               printStudentClasses(it);
std::cout « "\nSuccessfully switched" « std::endl;
00413
00414
00415
                saveOrReturn();
00416
         } else {
00417
            std::cout « "----
00418
00419
                       « std::endl;
              std::cout « "Class code not found" « std::endl;
00420
00421
              menuTryAgain(3, it);
00422
00423
        } else {
          std::cout « "-----
00424
            00425
00426
00427
            menuTryAgain(3, it);
         }
00428
       } else {
00429
       std::cout « "---
00430
00431
                   « std::endl;
00432
         std::cout « "You are not enrolled in this UC" « std::endl;
00433
          menuTryAgain(3, it);
        }
00434
00435
00436
        break;
00437
      case (2):
00438
        std::cout « "Enter UC to change class: " « std::endl;
00439
00440
        std::cin » ucCode;
00441
00442
        if (verifyUcCode(ucCode, it)) {
00444
         printFreeClasses(ucCode, count);
```

```
std::cout « "-
          00446
00447
          std::cin » classCode;
00448
00449
           check_class = verifyClassCode(classCode, ucCode, count);
00450
00451
00452
           removeUcStudent(ucCode, it, stackAlter, count);
00453
00454
             validate = valideNewClass(ucCode, classCode, it, classes);
           if (!validate) {
00455
              addClassStudent(ucCode, classCode, it, stackAlter);
00456
              printStudentClasses(it);
std::cout « "\nSuccessfully switched" « std::endl;
00457
00458
00459
              saveOrReturn();
00460
        } else {
00461
            std::cout « "--
00462
00463
                     « std::endl;
00464
            std::cout « "Class code not found" « std::endl;
00465
            menuTryAgain(3, it);
00466
          }
       } else {
00467
          std::cout « "-----
00468
00469
                    « std::endl;
00470
           std::cout « "You are not enrolled in this UC" « std::endl;
00471
          menuTryAgain(3, it);
00472
00473
         break;
00474
      default:
00475
       errorMessage();
00476
        break;
00477 }
00478 }
```

4.35.1.13 menuTryAgain()

Display options to try the current operation again or exit.

This function presents a menu allowing the user to choose between trying the current operation again or exiting the menu for adding, removing, or switching database entries.

Parameters

тепиТуре	An integer representing the type of operation (1 for add, 2 for remove, 3 for switch).	
it	An iterator referring to a specific database entry.	

Definition at line 225 of file menu.cpp.

```
00226
        int flag;
00228
       std::cout « "-----
                                                                 ---" « std::endl;
       std::cout « "| 1) Try again std::cout « "| 2) Exit
00229
                                                                  |" « std::endl;
                                                                  " « std::endl;
00230
       std::cout « "-
                                                       ----" « std::endl;
00231
00232
       std::cin » flag;
00233
00234
       switch (flag) {
00235
       case 1:
         system("clear");
00236
00237
        if (menuType == 1) {
00238
           menuAdd(it);
00239
        } else if (menuType == 2) {
00240
          menuRemove(it);
00241
        } else if (menuType == 3) {
00242
          menuSwitch(it);
00243
00244
         break:
00245
       case 2:
00246
         exit(0);
```

```
00247 default:

00248 errorMessage();

00249 break;

00250 }
```

4.35.1.14 menuUcs()

```
void menuUcs (
    std::string str,
    int type,
    int filter,
    int order )
```

Display UC and class data based on specified criteria.

This function displays UC and class data based on specified search criteria, filtering, and ordering.

Parameters

str	A string containing the search term or code.
type	An integer indicating the search type: 1 for one UC and its classes, 2 for a group, 3 for all UCs.
filter	An integer specifying the filter type: 1 for UC code, 2 for class code (optional).
order	An integer indicating the order type (optional).

Definition at line 796 of file menu.cpp.

```
00796
00797
        std::vector<myUc> data;
00798
        std::vector<myUc> oneUc;
00799
00800
        for (const auto &ucVectorPair : ucs) {
        for (const myUc &ucObj : ucVectorPair.second) {
00802
            data.push_back(ucObj);
00803
00804
       }
00805
00806
       if (type == 1) {
        oneUc = selectUc(str, classes);
00807
80800
         printUcClasses(oneUc);
00809
        if (type == 2) {
00810
         data = filterInfoUc(filter, str, data);
}
00811
00812
00813
         data = orderInfoUc(order, data);
00814
00815 }
         printUcs(data);
00816 }
```

4.35.1.15 restoreBackup()

```
void restoreBackup ( )
```

Restore data from a selected backup.

This function allows the user to choose a backup to restore data from and initiates the restoration process. After restoring the data, the user is returned to the main menu.

Definition at line 579 of file menu.cpp.

4.35.1.16 save()

```
void save ( )
```

Save all changes to the student data and exit the program.

This function saves all the changes made to the student data and exits the program. It uses the "keepAllChanges" function to preserve any modifications, such as adding or switching classes, before exiting.

Definition at line 518 of file menu.cpp.

```
00518 {
00519 keepAllChanges(students, stackAlter);
00520 exit(0);
00521 }
```

4.35.1.17 saveOrReturn()

```
void saveOrReturn ( )
```

Prompt the user to save changes or return to the previous menu.

This function displays options for the user to either save their changes or return to the previous menu. Users can select to save their actions, which may include adding or switching classes, or choose to return without saving.

Definition at line 487 of file menu.cpp.

```
int flag = 0;
00488
00489
                                                                   ----" « std::endl;
00490
        std::cout « "-----
                                                                    " « std::endl;
00491
        std::cout « "| 1) Save
00492
        std::cout « "| 2) Return
        std::cout « "---
00493
                                                                    --- « std::endl;
        std::cout « "Choose an option: ";
std::cin » flag;
00494
00495
00496
00497
       errorCheck(flag);
00498
00499
       switch (flag) {
00500
        case 1:
        save();
00501
00502
         break;
00503
       case 2:
        menuRequests();
00505
          break;
00506
       default:
00507
        errorMessage();
00508
         break;
00509
00510 }
```

4.35.1.18 selectBackupCode()

```
int selectBackupCode ( )
```

4.35.1.19 selectCode()

```
std::string selectCode ( )
```

Prompt the user to enter a code for searching.

This function displays a prompt to the user and collects a code to use for searching data.

Returns

A string containing the entered code for searching.

Definition at line 700 of file menu.cpp.

```
00700
00701
        std::string str;
00702
        std::cout « "-----
       std::cout « "| 1) Search by code std::cout « "-----
                                                             | " « std::endl;
00703
00704
00705 std::cout « "End
00706 std::cin » str;
       std::cout « "Enter the code: ";
        // errorcheck (str)
00708
00709
       return str;
00710 }
```

4.35.1.20 selectFilter()

```
int selectFilter ( )
```

Prompt the user to select a filter for data search.

This function displays a menu to the user for selecting a filter to apply during data search.

Returns

An integer representing the selected filter:

- 1: Filter by UC Code
- · 2: Filter by Class Code

Definition at line 721 of file menu.cpp.

```
00721
00722
00723
           int flag = 0;
00724
          std::cout « "-----
                                                                                                --" « std::endl;
          std::cout « "| 1) Uc Code
std::cout « "| 2) Class Code
                                                                                           | " « std::endl;
| " « std::endl;
00725
00726
00727
00728
00729     std::cout « "Choose an option: ";
00730     std::cin » flag;
00731     errorCheck(flag);
00732
00733
           return flag;
00734 }
```

4.35.1.21 selectOrderStudents()

```
int selectOrderStudents ( )
```

Prompt the user to select the sorting order for students.

This function displays a menu to allow the user to choose the sorting order for the list of students.

Returns

An integer representing the selected sorting order (1: ascending by student code, 2: descending by student code, 3: ascending by student name, 4: descending by student name).

Definition at line 626 of file menu.cpp.

```
00626
00627
         int flag = 0;
00628
00629
         std::cout « "----
                                                                                  -" « std::endl;
         std::cout « "| 1) Sort by student code asc std::cout « "| 2) Sort by student code desc
00630
                                                                                  |" « std::endl;
                                                                                  " « std::endl;
00631
         std::cout « "| 3) Sort by student name asc std::cout « "| 4) Sort by student name desc
                                                                                  " « std::endl;
00632
                                                                                  " « std::endl;
00633
         std::cout « "
                                                                                   -" « std::endl;
00634
         // add more order like - n° ucs,
00636
         std::cout « "Choose an option: ";
00637
         std::cin » flag;
00638
00639
         errorCheck(flag);
00640
00641
         return flag;
00642 }
```

4.35.1.22 selectOrderUcs()

```
int selectOrderUcs ( )
```

Prompt the user to select the sorting order for UCs.

This function displays a menu to allow the user to choose the sorting order for the list of UCs.

Returns

An integer representing the selected sorting order (1: ascending by UC code, 2: descending by UC code, 3: ascending by class code, 4: descending by class code).

Definition at line 652 of file menu.cpp.

```
00652
00653
          int flag = 0;
00654
00655
          std::cout « "--
                                                                                         -" « std::endl;
          std::cout « "| 1) Sort by uc code asc
          std::cout « "| 2) Sort by uc code desc
std::cout « "| 3) Sort by class code asc
std::cout « "| 4) Sort by class code desc
00657
                                                                                         |" « std::endl;
                                                                                         " « std::endl;
00658
                                                                                        " « std::endl;
00659
          std::cout « "-
                                                                                         -" « std::endl;
00660
         // add more order like - n° ucs,
std::cout « "Choose an option: ";
00661
00662
00663
         std::cin » flag;
00664
00665
         errorCheck(flag);
00666
00667
          return flag;
00668 }
```

4.35.1.23 selectType()

```
int selectType ( )
```

Prompt the user to select the viewing type.

This function displays a menu to allow the user to choose the type of data viewing.

Returns

An integer representing the selected viewing type (1: See one, 2: See a particular group, 3: See all).

Definition at line 677 of file menu.cpp.

```
00677
00678
         int flag = 0;
00679
00680
        std::cout « "----
                                       -----" « std::endl;
        std::cout « "| 1) See one
std::cout « "| 2) See a particular group
std::cout « "| 3) See all
                                                                              |" « std::endl;
00681
                                                                                " « std::endl;
00682
00683
                                                                                |" « std::endl;
                                                                            ----" « std::endl;
00684
        std::cout « "-
00685
00686 std::cout « "Choose an option: ";
00687 std::cin » flag;
00688 errorCheck(flag);
00689
00690 return flag;
00691 }
```

4.35.1.24 selectValue()

```
std::string selectValue ( )
```

Prompt the user to enter a value for filtering data.

This function prompts the user to enter a value to be used as a filter during data search.

Returns

A string representing the user-entered value.

Definition at line 743 of file menu.cpp.

```
00743 {
00744 std::string str;
00745
00745 std::cout « "Enter the value: ";
00747 std::cin » str;
00748 // errorcheck (str)
00749
00750 return str;
00751 }
```

4.36 menu.h

Go to the documentation of this file.

```
00001 #ifndef MENU_H
00002 #define MENU_H
00003
00004 #include <iostream>
00005 #include <list>
00006 #include <map>
00007 #include <stack>
80000
00009 #include "classes/student.h"
00010 #include "classes/uc.h"
00011 #include "functions/dbStudents.h"
00012 #include "functions/dbUcs.h"
00013 #include "inputoutput/keepAllChanges.h"
00014 #include "inputoutput/print.h"
00015 #include "inputoutput/read.h"
00016
00017 void errorMessage();
00018 void errorCheck(int n);
00019
00020 void menuStudents(std::string str = "", int type = 0, int filter = 0,
00021 int order = 0);
00022 void menuUcs(std::string str = "", int type = 0, int filter = 0, int order = 0);
00024 void menuStudentCode(int flag);
00025 void menuTryAgain(int menuType, std::map<std::string, myStudent>::iterator &it);
00026
00027 void menu();
00028 void menuSeeDatabase();
00029 void menuRequests();
00030 void menuRemove(std::map<std::string, myStudent>::iterator &it);
00031 void menuAdd(std::map<std::string, myStudent>::iterator &it);
00032 void menuSwitch(std::map<std::string, myStudent>::iterator &it);
00033
00034 void menuBackup();
00035 void menuChanges();
00036 void restoreBackup();
00037 int selectBackupCode();
00038
00039 int selectOrderStudents();
00040 int selectOrderUcs();
00041 int selectType();
00042 int selectFilter();
00043 std::string selectCode();
00044 std::string selectValue();
00045
00046 void saveOrReturn();
00047 void save();
00048
00049 #endif
```