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:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

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<()

```
bool classInfo::operator< (
            const classInfo & other ) const  [inline]
```

Definition at line 16 of file uc.h.

```
00016                                                {
00017     return startTime < other.startTime;
00018   }
```

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<()

```
bool classQtd::operator< (
            const classQtd & other ) const  [inline]
```

Definition at line 25 of file uc.h.

```
00025                                       {
00026     return classCode < other.classCode;
00027   }
```

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]

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

Constructor for the myStudent class.

Parameters

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

Definition at line 8 of file student.cpp.

```
00008                                                                   {
00009   studentCode = sCode;
00010   studentName = sName;
00011 }
```

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()

```
void myStudent::addClass (
            const myUc & myClass )
```

Adds a class to the student's classes vector.

Parameters

classe	ClassComp object to be added.

Definition at line 99 of file student.cpp.

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

3.4.3.2 addUc()

```
void myStudent::addUc (
            const myUc & myClass )
```

3.4.3.3 changeClass()

```
void myStudent::changeClass (
            const myUc & myClass )
```

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   std::vector<std::string> classCodes;
00080   for (const auto &uc : classes) {
00081     std::vector<classInfo> classInfoVec = uc.getClassInfoVec();
00082     for (const auto &classInfo : classInfoVec) {
00083       classCodes.push_back(classInfo.code);
00084     }
00085   }
00086   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.

```
00061                                                {
00062   std::vector<std::string> ucCodes;
00063   for (const auto &uc : classes) {
00064     ucCodes.push_back(uc.getUcCode());
00065   }
00066   return ucCodes;
00067 }
```

3.4.3.9 removeUc()

```
void myStudent::removeUc (
            const myUc & myClass )
```

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	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]

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

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()

```
void myUc::addClass (
             const std::string & code )
```

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()

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

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 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()

```
bool myUc::compareUcCode (
            const myUc & a,
            const myUc & b )  [static]
```

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.

```
00103                                                   {
00104   return a.ucCode < b.ucCode;
00105 }
```

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<()

```
bool myUc::operator< (
            const myUc & other ) const
```

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()

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

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;
00010   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; }
00051
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) {
00081     std::vector<classInfo> classInfoVec = uc.getClassInfoVec();
00082     for (const auto &classInfo : classInfoVec) {
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   std::string studentName;
00022   mutable std::vector<myUc> classes;
00023
00024 public:
00025   // Constructor functions
00026   myStudent(const std::string &sCode, const std::string &sName);
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
00034   // Getters functions
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);
```

```
00043   void changeClass(const myUc &myClass);
00044   void addUc(const myUc &myClass);
00045   void removeUc(const myUc &myClass);
00046   bool valideQtClasses();
00047 };
00048
00049 #endif
```

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 = {};
00022 }
00023
00028 void myUc::SetUc(std::string &ucC, std::string &classC) {
00029   ucCode = ucC;
00030   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
00084 void myUc::addClassInfo(std::string type, std::string day, int dayInt,
00085                         double startTime, double duration) {
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;
00010   std::string type;
00011   std::string day;
00012   int dayInt;
00013   double startTime;
00014   double duration;
00015
00016   bool operator<(const classInfo &other) const {
00017     return startTime < other.startTime;
00018   }
00019 };
00020
00021 struct classQtd {
00022   std::string classCode;
00023   int qtd;
00024
00025   bool operator<(const classQtd &other) const {
00026     return classCode < other.classCode;
00027   }
00028 };
00029
00030 class myUc {
00031 private:
00032   std::string classCode;
00033   std::string ucCode;
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);
00054   void addClassInfo(std::string type, std::string day, int dayInt,
00055                     double startTime, double duration);
00056
00057   bool operator<(const myUc &other) const;
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 }
00008
00009 void errorCheck(int n) {
00010   if (n == 0) {
00011     std::cout « "ERROR: Invalid number" « std::endl;
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)
- bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)
- bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)
- bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)
- std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)
- std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)
- std::map< std::string, myStudent > selectStudent (const std::string &str, const std::map< std::string, myStudent > &students)
- void organizerUcStudent (std::map< std::string, myStudent >::iterator &it)
- bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter, std::map< std::string, std::vector< classQtd > > &count)
- void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter)

- void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector< classQtd > > &count, int type)
- bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)
- std::map< int, std::set< classInfo > > orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)
- std::string weekDayString (int day)
- bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)

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 )
```

Definition at line 128 of file dbStudents.cpp.

```
00130                                                   {
00131
00132   myUc classe(ucCode, classCode);
00133   it->second.getClasses().push_back(classe);
00134   organizerUcStudent(it);
00135   stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00136                    "add", ucCode, classCode});
00137 }
```

4.11.1.2 compareStudentNameAsc()

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

Definition at line 11 of file dbStudents.cpp.

```
00012                                                       {
00013   return student1.getStudentName() < student2.getStudentName();
00014 }
```

4.11.1.3 compareStudentNameDesc()

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

Definition at line 15 of file dbStudents.cpp.

```
00016                                                       {
00017   return student1.getStudentName() > student2.getStudentName();
00018 }
```

4.11.1.4 compareStudentsCodeAsc()

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

Definition at line 3 of file dbStudents.cpp.

```
00004                                                       {
00005   return student1.getStudentCode() < student2.getStudentCode();
00006 }
```

4.11.1.5 compareStudentsCodeDesc()

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

Definition at line 7 of file dbStudents.cpp.

```
00008                                                    {
00009   return student1.getStudentCode() > student2.getStudentCode();
00010 }
```

4.11.1.6 filterInfoStudent()

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

Definition at line 21 of file dbStudents.cpp.

```
00022                                                                {
00023   std::vector<myStudent> filterStudents;
00024   switch (n) {
00025   case 1:
00026     // Filter by Uc Code
00027     for (const auto &student : students) {
00028       for (const auto &uc : student.getClasses()) {
00029         if (uc.getUcCode() == str) {
00030           filterStudents.push_back(student);
00031           break;
00032         }
00033       }
00034     }
00035     break;
00036   case 2:
00037     // Filter by Class Code
00038     for (const auto &student : students) {
00039       for (const auto &uc : student.getClasses()) {
00040         for (const auto &classInfo : uc.getClassInfoVec()) {
00041           if (classInfo.code == str) {
00042             filterStudents.push_back(student);
00043             break; // No need to check other class codes for this student
00044           }
00045         }
00046       }
00047     }
00048     break;
00049   default:
00050     errorMessage();
00051     break;
00052   }
00053   return filterStudents;
00054 }
```

4.11.1.7 orderInfoStudent()

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

Definition at line 56 of file dbStudents.cpp.

```
00057                                                                    {
00058   switch (n) {
00059   case 1:
00060     // Order by Student Code Asc
00061     std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00062     break;
00063   case 2:
00064     // Order by Student Code Desc
00065     std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00066     break;
```

```
00067   case 3:
00068     // Order by Student Name Asc
00069     std::sort(students.begin(), students.end(), compareStudentNameAsc);
00070     break;
00071   case 4:
00072     // Order by Student Name Desc
00073     std::sort(students.begin(), students.end(), compareStudentNameDesc);
00074     break;
00075   default:
00076     errorMessage();
00077     break;
00078   }
00079
00080   return students;
00081 }
```

4.11.1.8 orderStudentClass()

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

Definition at line 211 of file dbStudents.cpp.

```
00212                                                        {
00213
00214   // map to order the classes
00215   // by day
00216   std::map<int, std::set<classInfo» orderClasses;
00217
00218   // for each class of the
00219   // student, search in the
00220   // class tree and add the
00221   // classInfo in the
00222   // orderClasses map
00223   for (const auto &classe : it->second.getClasses()) {
00224     std::string value = classe.getUcCode() + classe.getClassCode();
00225
00226     // student one class
00227     // pointer, verify if the
00228     // class exists in the
00229     // class tree
00230     auto it_class = classes.find(value);
00231
00232     // if the class does not
00233     // exist, print error
00234     if (it_class == classes.end()) {
00235       std::cerr « "Error in "
00236                   "find class"
00237                « std::endl;
00238     } else {
00239       // if exists, add the
00240       // classInfo in the
00241       // orderClasses map
00242       for (auto &classInfo : it_class->second.getClassInfoVec()) {
00243         classInfo.code = classe.getUcCode();
00244         orderClasses[classInfo.dayInt].insert(classInfo);
00245       }
00246     }
00247   }
00248   return orderClasses;
00249 }
```

4.11.1.9 organizerUcStudent()

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

Definition at line 100 of file dbStudents.cpp.

```
00100                                                                {
00101
00102   std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
00103             myUc::compareUcCode);
00104 }
```

4.11.1.10 removeUcStudent()

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

Definition at line 107 of file dbStudents.cpp.

```
00110                                                                    {
00111
00112   bool remove = false;
00113   for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
00114     if (it->second.getClasses()[i].getUcCode() == ucCode) {
00115       stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00116                        "remove", ucCode,
00117                        it->second.getClasses()[i].getUcCode()});
00118       it->second.getClasses().erase(it->second.getClasses().begin() + i);
00119       remove = true;
00120       updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
00121                          count, 0);
00122     }
00123   }
00124   return remove;
00125 }
```

4.11.1.11 selectStudent()

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

Definition at line 84 of file dbStudents.cpp.

```
00085                                                                    {
00086   std::map<std::string, myStudent> selectedStudents;
00087
00088   for (auto &studentPair : students) {
00089     const myStudent &mystudent = studentPair.second;
00090     if (str == mystudent.getStudentCode()) {
00091       selectedStudents[studentPair.first] = mystudent;
00092     }
00093   }
00094
00095   return selectedStudents;
00096 }
```

4.11.1.12 updateCountClasses()

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

Definition at line 141 of file dbStudents.cpp.

```
00143                                    {
00144
00145   auto it_count = count.find(ucCode);
00146   if (it_count != count.end()) {
00147     for (auto &classe : it_count->second) {
00148       if (classe.classCode == classCode) {
00149         if (type == 1) {
00150           classe.qtd++;
00151         } else {
00152           classe.qtd--;
00153         }
00154       }
00155     }
00156   }
00157 }
```

4.11.1.13 valideNewClass()

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

Definition at line 161 of file dbStudents.cpp.

```
00163                                                                {
00164
00165   // call function to order the
00166   // classes of the student by
00167   // int day
00168   std::map<int, std::set<classInfo» orderClasses =
00169       orderStudentClass(it, classes);
00170
00171   std::string value = ucCode + classCode;
00172
00173   auto it_class = classes.find(value);
00174
00175   if (it_class == classes.end()) {
00176     std::cout « "Error in "
00177                 "find class"
00178              « std::endl;
00179     return true;
00180   } else {
00181     // verify if has a class in
00182     // the same day and time
00183     for (const auto &class_info : it_class->second.getClassInfoVec()) {
00184       // get all classes of the
00185       // day of class
00186       const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00187
00188       // and verify if the
00189       // student has a class in
00190       // the same time aula -> student classes
00191       // class_info -> class to add
00192       for (const auto &aula : classesOfDay) {
00193
00194         if (aula.type != "T" && class_info.type != "T" &&
00195             class_info.startTime >= aula.startTime &&
00196             class_info.startTime < aula.startTime + aula.duration) {
00197           std::cout « "Error: "
00198                       "Incompatible"
00199                       " schedules"
00200                    « std::endl;
00201           return true;
00202         }
00203       }
00204     }
00205     return false;
00206   }
00207 }
```

4.11.1.14 verifyUcCode()

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

Definition at line 279 of file dbStudents.cpp.

```
00280                                                                {
00281
00282   for (const auto &classe : it->second.getClasses()) {
00283     if (classe.getUcCode() == ucCode) {
00284       return true;
00285     }
00286   }
00287
00288   return false;
00289 }
```

4.11.1.15 weekDayString()

```
std::string weekDayString (
            int day )
```

Definition at line 251 of file dbStudents.cpp.

```
00251                                {
00252   switch (day) {
00253   case 2:
00254     return "Monday";
00255     break;
00256   case 3:
00257     return "Tuesday";
00258     break;
00259   case 4:
00260     return "Wednesday";
00261     break;
00262   case 5:
00263     return "Thursday";
00264     break;
00265   case 6:
00266     return "Friday";
00267     break;
00268   case 7:
00269     return "Saturday";
00270     break;
00271   default:
00272     return "Error Day";
00273     break;
00274   }
00275 }
```

4.12 dbStudents.cpp

Go to the documentation of this file.

```
00001 #include "dbStudents.h"
00002
00003 bool compareStudentsCodeAsc(const myStudent &student1,
00004                             const myStudent &student2) {
00005   return student1.getStudentCode() < student2.getStudentCode();
00006 }
00007 bool compareStudentsCodeDesc(const myStudent &student1,
00008                              const myStudent &student2) {
00009   return student1.getStudentCode() > student2.getStudentCode();
00010 }
00011 bool compareStudentNameAsc(const myStudent &student1,
00012                            const myStudent &student2) {
00013   return student1.getStudentName() < student2.getStudentName();
00014 }
00015 bool compareStudentNameDesc(const myStudent &student1,
00016                             const myStudent &student2) {
00017   return student1.getStudentName() > student2.getStudentName();
00018 }
00019
00020 std::vector<myStudent>
00021 filterInfoStudent(int n, std::string str,
00022                   const std::vector<myStudent> &students) {
00023   std::vector<myStudent> filterStudents;
00024   switch (n) {
00025   case 1:
00026     // Filter by Uc Code
00027     for (const auto &student : students) {
00028       for (const auto &uc : student.getClasses()) {
00029         if (uc.getUcCode() == str) {
00030           filterStudents.push_back(student);
00031           break;
00032         }
00033       }
00034     }
00035     break;
00036   case 2:
00037     // Filter by Class Code
00038     for (const auto &student : students) {
00039       for (const auto &uc : student.getClasses()) {
00040         for (const auto &classInfo : uc.getClassInfoVec()) {
00041           if (classInfo.code == str) {
00042             filterStudents.push_back(student);
00043             break; // No need to check other class codes for this student
```

```
00044           }
00045         }
00046       }
00047     }
00048     break;
00049   default:
00050     errorMessage();
00051     break;
00052   }
00053   return filterStudents;
00054 }
00055
00056 std::vector<myStudent> orderInfoStudent(int n,
00057                                         std::vector<myStudent> &students) {
00058   switch (n) {
00059   case 1:
00060     // Order by Student Code Asc
00061     std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00062     break;
00063   case 2:
00064     // Order by Student Code Desc
00065     std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00066     break;
00067   case 3:
00068     // Order by Student Name Asc
00069     std::sort(students.begin(), students.end(), compareStudentNameAsc);
00070     break;
00071   case 4:
00072     // Order by Student Name Desc
00073     std::sort(students.begin(), students.end(), compareStudentNameDesc);
00074     break;
00075   default:
00076     errorMessage();
00077     break;
00078   }
00079
00080   return students;
00081 }
00082
00083 std::map<std::string, myStudent>
00084 selectStudent(const std::string &str,
00085               const std::map<std::string, myStudent> &students) {
00086   std::map<std::string, myStudent> selectedStudents;
00087
00088   for (auto &studentPair : students) {
00089     const myStudent &mystudent = studentPair.second;
00090     if (str == mystudent.getStudentCode()) {
00091       selectedStudents[studentPair.first] = mystudent;
00092     }
00093   }
00094
00095   return selectedStudents;
00096 }
00097
00098 // ------------------------------------------------ //
00099
00100 void organizerUcStudent(std::map<std::string, myStudent>::iterator &it) {
00101
00102   std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
00103             myUc::compareUcCode);
00104 }
00105
00106 // receives the student pointer by reference and removes the UC
00107 bool removeUcStudent(std::string ucCode,
00108                      std::map<std::string, myStudent>::iterator &it,
00109                      std::stack<alter> &stackAlter,
00110                      std::map<std::string, std::vector<classQtd» &count) {
00111
00112   bool remove = false;
00113   for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
00114     if (it->second.getClasses()[i].getUcCode() == ucCode) {
00115       stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00116                        "remove", ucCode,
00117                        it->second.getClasses()[i].getUcCode()});
00118       it->second.getClasses().erase(it->second.getClasses().begin() + i);
00119       remove = true;
00120       updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
00121                          count, 0);
00122     }
00123   }
00124   return remove;
00125 }
00126
00127 // receives the stuede pointer by reference and add the new Class
00128 void addClassStudent(std::string ucCode, std::string classCode,
00129                      std::map<std::string, myStudent>::iterator &it,
00130                      std::stack<alter> &stackAlter) {
```

```
00131
00132   myUc classe(ucCode, classCode);
00133   it->second.getClasses().push_back(classe);
00134   organizerUcStudent(it);
00135   stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00136                    "add", ucCode, classCode});
00137 }
00138
00139 // update the class count tree
00140 // 1 for add and 0 for remove
00141 void updateCountClasses(std::string ucCode, std::string classCode,
00142                         std::map<std::string, std::vector<classQtd» &count,
00143                         int type) {
00144
00145   auto it_count = count.find(ucCode);
00146   if (it_count != count.end()) {
00147     for (auto &classe : it_count->second) {
00148       if (classe.classCode == classCode) {
00149         if (type == 1) {
00150           classe.qtd++;
00151         } else {
00152           classe.qtd--;
00153         }
00154       }
00155     }
00156   }
00157 }
00158
00159 // receives the student pointer by reference and class Tree (classes) and th
00160 // UC code and class code
00161 bool valideNewClass(std::string ucCode, std::string classCode,
00162                     std::map<std::string, myStudent>::iterator &it,
00163                     std::map<std::string, myUc> &classes) {
00164
00165   // call function to order the
00166   // classes of the student by
00167   // int day
00168   std::map<int, std::set<classInfo» orderClasses =
00169       orderStudentClass(it, classes);
00170
00171   std::string value = ucCode + classCode;
00172
00173   auto it_class = classes.find(value);
00174
00175   if (it_class == classes.end()) {
00176     std::cout « "Error in "
00177                 "find class"
00178              « std::endl;
00179     return true;
00180   } else {
00181     // verify if has a class in
00182     // the same day and time
00183     for (const auto &class_info : it_class->second.getClassInfoVec()) {
00184       // get all classes of the
00185       // day of class
00186       const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00187
00188       // and verify if the
00189       // student has a class in
00190       // the same time aula -> student classes
00191       // class_info -> class to add
00192       for (const auto &aula : classesOfDay) {
00193
00194         if (aula.type != "T" && class_info.type != "T" &&
00195             class_info.startTime >= aula.startTime &&
00196             class_info.startTime < aula.startTime + aula.duration) {
00197           std::cout « "Error: "
00198                       "Incompatible"
00199                       " schedules"
00200                    « std::endl;
00201           return true;
00202         }
00203       }
00204     }
00205     return false;
00206   }
00207 }
00208
00209 // This function receives the student pointer and the class tree (classes)
00210 std::map<int, std::set<classInfo»
00211 orderStudentClass(std::map<std::string, myStudent>::iterator &it,
00212                   std::map<std::string, myUc> &classes) {
00213
00214   // map to order the classes
00215   // by day
00216   std::map<int, std::set<classInfo» orderClasses;
00217
```

```
00218   // for each class of the
00219   // student, search in the
00220   // class tree and add the
00221   // classInfo in the
00222   // orderClasses map
00223   for (const auto &classe : it->second.getClasses()) {
00224     std::string value = classe.getUcCode() + classe.getClassCode();
00225
00226     // student one class
00227     // pointer, verify if the
00228     // class exists in the
00229     // class tree
00230     auto it_class = classes.find(value);
00231
00232     // if the class does not
00233     // exist, print error
00234     if (it_class == classes.end()) {
00235       std::cerr « "Error in "
00236                   "find class"
00237                « std::endl;
00238     } else {
00239       // if exists, add the
00240       // classInfo in the
00241       // orderClasses map
00242       for (auto &classInfo : it_class->second.getClassInfoVec()) {
00243         classInfo.code = classe.getUcCode();
00244         orderClasses[classInfo.dayInt].insert(classInfo);
00245       }
00246     }
00247   }
00248   return orderClasses;
00249 }
00250
00251 std::string weekDayString(int day) {
00252   switch (day) {
00253   case 2:
00254     return "Monday";
00255     break;
00256   case 3:
00257     return "Tuesday";
00258     break;
00259   case 4:
00260     return "Wednesday";
00261     break;
00262   case 5:
00263     return "Thursday";
00264     break;
00265   case 6:
00266     return "Friday";
00267     break;
00268   case 7:
00269     return "Saturday";
00270     break;
00271   default:
00272     return "Error Day";
00273     break;
00274   }
00275 }
00276
00277 // Checks whether the student is already enrolled in a UC class. If it
00278 // returns true it means a problem was found
00279 bool verifyUcCode(std::string ucCode,
00280                   std::map<std::string, myStudent>::iterator &it) {
00281
00282   for (const auto &classe : it->second.getClasses()) {
00283     if (classe.getUcCode() == ucCode) {
00284       return true;
00285     }
00286   }
00287
00288   return false;
00289 }
```

4.13 src/functions/dbStudents.h File Reference

```
#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
```

```
#include <list>
#include <map>
#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)
- std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)
- std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)
- bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter, std::map< std::string, std::vector< classQtd > > &count)
- void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter)
- std::map< int, std::set< classInfo > > orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)
- bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)
- void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector< classQtd > > &count, int type)
- std::string weekDayString (int day)
- bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)
- bool compareStudentsCodeAsc (const myStudent &student1, const myStudent &student2)
- bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)
- bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)
- bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)

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 )
```

Definition at line 128 of file dbStudents.cpp.

```
00130                                                   {
00131
00132   myUc classe(ucCode, classCode);
00133   it->second.getClasses().push_back(classe);
00134   organizerUcStudent(it);
00135   stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00136                    "add", ucCode, classCode});
00137 }
```

4.13.1.2 compareStudentNameAsc()

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

Definition at line 11 of file dbStudents.cpp.

```
00012                                                   {
00013   return student1.getStudentName() < student2.getStudentName();
00014 }
```

4.13.1.3 compareStudentNameDesc()

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

Definition at line 15 of file dbStudents.cpp.

```
00016                                                   {
00017   return student1.getStudentName() > student2.getStudentName();
00018 }
```

4.13.1.4 compareStudentsCodeAsc()

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

Definition at line 3 of file dbStudents.cpp.

```
00004                                                   {
00005   return student1.getStudentCode() < student2.getStudentCode();
00006 }
```

4.13.1.5 compareStudentsCodeDesc()

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

Definition at line 7 of file dbStudents.cpp.

```
00008                                                   {
00009   return student1.getStudentCode() > student2.getStudentCode();
00010 }
```

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()

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

Definition at line 21 of file dbStudents.cpp.

```
00022                                                                   {
00023   std::vector<myStudent> filterStudents;
00024   switch (n) {
00025   case 1:
00026     // Filter by Uc Code
00027     for (const auto &student : students) {
00028       for (const auto &uc : student.getClasses()) {
00029         if (uc.getUcCode() == str) {
00030           filterStudents.push_back(student);
00031           break;
00032         }
00033       }
00034     }
00035     break;
00036   case 2:
00037     // Filter by Class Code
00038     for (const auto &student : students) {
00039       for (const auto &uc : student.getClasses()) {
00040         for (const auto &classInfo : uc.getClassInfoVec()) {
00041           if (classInfo.code == str) {
00042             filterStudents.push_back(student);
00043             break; // No need to check other class codes for this student
00044           }
00045         }
00046       }
00047     }
00048     break;
00049   default:
00050     errorMessage();
00051     break;
00052   }
00053   return filterStudents;
00054 }
```

4.13.1.8 orderInfoStudent()

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

Definition at line 56 of file dbStudents.cpp.

```
00057                                                                   {
00058   switch (n) {
00059   case 1:
00060     // Order by Student Code Asc
00061     std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
00062     break;
00063   case 2:
00064     // Order by Student Code Desc
00065     std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
00066     break;
00067   case 3:
00068     // Order by Student Name Asc
00069     std::sort(students.begin(), students.end(), compareStudentNameAsc);
00070     break;
00071   case 4:
00072     // Order by Student Name Desc
00073     std::sort(students.begin(), students.end(), compareStudentNameDesc);
00074     break;
00075   default:
00076     errorMessage();
00077     break;
00078   }
00079
00080   return students;
00081 }
```

4.13.1.9 orderStudentClass()

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

Definition at line 211 of file dbStudents.cpp.

```
00212                                                    {
00213
00214   // map to order the classes
00215   // by day
00216   std::map<int, std::set<classInfo» orderClasses;
00217
00218   // for each class of the
00219   // student, search in the
00220   // class tree and add the
00221   // classInfo in the
00222   // orderClasses map
00223   for (const auto &classe : it->second.getClasses()) {
00224     std::string value = classe.getUcCode() + classe.getClassCode();
00225
00226     // student one class
00227     // pointer, verify if the
00228     // class exists in the
00229     // class tree
00230     auto it_class = classes.find(value);
00231
00232     // if the class does not
00233     // exist, print error
00234     if (it_class == classes.end()) {
00235       std::cerr « "Error in "
00236                   "find class"
00237                « std::endl;
00238     } else {
00239       // if exists, add the
00240       // classInfo in the
00241       // orderClasses map
00242       for (auto &classInfo : it_class->second.getClassInfoVec()) {
00243         classInfo.code = classe.getUcCode();
00244         orderClasses[classInfo.dayInt].insert(classInfo);
00245       }
00246     }
00247   }
00248   return orderClasses;
00249 }
```

4.13.1.10 removeUcStudent()

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

Definition at line 107 of file dbStudents.cpp.

```
00110                                                                {
00111
00112   bool remove = false;
00113   for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
00114     if (it->second.getClasses()[i].getUcCode() == ucCode) {
00115       stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
00116                        "remove", ucCode,
00117                        it->second.getClasses()[i].getUcCode()});
00118       it->second.getClasses().erase(it->second.getClasses().begin() + i);
00119       remove = true;
00120       updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
00121                          count, 0);
00122     }
00123   }
00124   return remove;
00125 }
```

4.13.1.11 selectStudent()

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

Definition at line 84 of file dbStudents.cpp.

```
00085                                                                    {
00086   std::map<std::string, myStudent> selectedStudents;
00087
00088   for (auto &studentPair : students) {
00089     const myStudent &mystudent = studentPair.second;
00090     if (str == mystudent.getStudentCode()) {
00091       selectedStudents[studentPair.first] = mystudent;
00092     }
00093   }
00094
00095   return selectedStudents;
00096 }
```

4.13.1.12 updateCountClasses()

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

Definition at line 141 of file dbStudents.cpp.

```
00143                                    {
00144
00145   auto it_count = count.find(ucCode);
00146   if (it_count != count.end()) {
00147     for (auto &classe : it_count->second) {
00148       if (classe.classCode == classCode) {
00149         if (type == 1) {
00150           classe.qtd++;
00151         } else {
00152           classe.qtd--;
00153         }
00154       }
00155     }
00156   }
00157 }
```

4.13.1.13 valideNewClass()

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

Definition at line 161 of file dbStudents.cpp.

```
00163                                                        {
00164
00165   // call function to order the
00166   // classes of the student by
00167   // int day
00168   std::map<int, std::set<classInfo» orderClasses =
00169       orderStudentClass(it, classes);
00170
00171   std::string value = ucCode + classCode;
00172
00173   auto it_class = classes.find(value);
00174
00175   if (it_class == classes.end()) {
00176     std::cout « "Error in "
```

```
00177                 "find class"
00178               « std::endl;
00179     return true;
00180   } else {
00181     // verify if has a class in
00182     // the same day and time
00183     for (const auto &class_info : it_class->second.getClassInfoVec()) {
00184       // get all classes of the
00185       // day of class
00186       const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];
00187
00188       // and verify if the
00189       // student has a class in
00190       // the same time aula -> student classes
00191       // class_info -> class to add
00192       for (const auto &aula : classesOfDay) {
00193
00194         if (aula.type != "T" && class_info.type != "T" &&
00195             class_info.startTime >= aula.startTime &&
00196             class_info.startTime < aula.startTime + aula.duration) {
00197           std::cout « "Error: "
00198                       "Incompatible"
00199                       " schedules"
00200                    « std::endl;
00201           return true;
00202         }
00203       }
00204     }
00205     return false;
00206   }
00207 }
```

4.13.1.14 verifyUcCode()

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

Definition at line 279 of file dbStudents.cpp.

```
00280                                                          {
00281
00282   for (const auto &classe : it->second.getClasses()) {
00283     if (classe.getUcCode() == ucCode) {
00284       return true;
00285     }
00286   }
00287
00288   return false;
00289 }
```

4.13.1.15 weekDayString()

```
std::string weekDayString (
              int day )
```

Definition at line 251 of file dbStudents.cpp.

```
00251                                  {
00252   switch (day) {
00253   case 2:
00254     return "Monday";
00255     break;
00256   case 3:
00257     return "Tuesday";
00258     break;
00259   case 4:
00260     return "Wednesday";
00261     break;
00262   case 5:
00263     return "Thursday";
00264     break;
00265   case 6:
00266     return "Friday";
00267     break;
00268   case 7:
```

```
00269     return "Saturday";
00270     break;
00271   default:
00272     return "Error Day";
00273     break;
00274   }
00275 }
```

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,
00024                   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,
00030                   const std::vector<myStudent> &students);
00031 std::vector<myStudent> orderInfoStudent(int n,
00032                                         std::vector<myStudent> &students);
00033
00034 bool removeUcStudent(std::string ucCode,
00035                      std::map<std::string, myStudent>::iterator &it,
00036                      std::stack<alter> &stackAlter,
00037                      std::map<std::string, std::vector<classQtd» &count);
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,
00047                     std::map<std::string, myStudent>::iterator &it,
00048                     std::map<std::string, myUc> &classes);
00049
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,
00056                   std::map<std::string, myStudent>::iterator &it);
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,
00063                            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()

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

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 10 of file dbUcs.cpp.

```
00010                                                    {
00011   return uc1.getClassCode() < uc2.getClassCode();
00012 }
```

4.15.1.2 compareClassesCodeDesc()

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

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 30 of file dbUcs.cpp.

```
00030                                                                    {
00031   return uc1.getClassCode() > uc2.getClassCode();
00032 }
```

4.15.1.3 compareUcsCodeASC()

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

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 20 of file dbUcs.cpp.

```
00020                                                                    {
00021   return uc1.getUcCode() < uc2.getUcCode();
00022 }
```

4.15.1.4 compareUcsCodeDesc()

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

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 40 of file dbUcs.cpp.

```
00040                                                                {
00041   return uc1.getUcCode() > uc2.getUcCode();
00042 }
```

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 51 of file dbUcs.cpp.

```
00051                                                                {
00052   std::vector<myUc> filterUc;
00053   switch (n) {
00054   case 1:
00055     // Filter by Uc Code
00056     for (const auto &uc : ucs) {
00057       if (uc.getUcCode() == str) {
00058         filterUc.push_back(uc);
00059       }
00060     }
00061     break;
00062   case 2:
00063     // Filter by Class Code
00064     for (const auto &uc : ucs) {
00065       for (const auto &classInfo : uc.getClassInfoVec()) {
00066         if (classInfo.code == str) {
00067           filterUc.push_back(uc);
00068           break;
00069         }
00070       }
00071     }
00072     break;
00073   default:
00074     errorMessage();
00075     break;
00076   }
00077   return filterUc;
00078 }
```

4.15.1.6 orderInfoUc()

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

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 86 of file dbUcs.cpp.

```
00086                                                          {
00087
00088   switch (n) {
00089   case 1:
00090     // Order by Uc Code Asc
00091     std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00092     break;
00093   case 2:
00094     // Order by Uc Code Desc
00095     std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00096     break;
00097   case 3:
00098     // Order by Class Code Asc
00099     std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00100     break;
00101   case 4:
00102     // Order by Class Code Desc
00103     std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00104     break;
00105   default:
00106     errorMessage();
00107     break;
00108   }
00109   return ucs;
00110 }
```

4.15.1.7 selectUc()

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

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 119 of file dbUcs.cpp.

```
00120                                                                  {
00121   std::vector<myUc> selectedUcs;
00122
00123   for (const auto &pair : classes) {
00124     auto ucObj = pair.second;
00125
00126     if (ucObj.getUcCode() == str) {
00127       selectedUcs.push_back(ucObj);
00128     }
00129   }
00130   return selectedUcs;
00131 }
```

4.16 dbUcs.cpp

Go to the documentation of this file.

```
00001 #include "dbUcs.h"
00002
00010 bool compareClassesCodeAsc(const myUc &uc1, const myUc &uc2) {
00011   return uc1.getClassCode() < uc2.getClassCode();
00012 }
00013
00020 bool compareUcsCodeASC(const myUc &uc1, const myUc &uc2) {
00021   return uc1.getUcCode() < uc2.getUcCode();
00022 }
00023
00030 bool compareClassesCodeDesc(const myUc &uc1, const myUc &uc2) {
00031   return uc1.getClassCode() > uc2.getClassCode();
00032 }
00033
00040 bool compareUcsCodeDesc(const myUc &uc1, const myUc &uc2) {
00041   return uc1.getUcCode() > uc2.getUcCode();
00042 }
00043
00051 std::vector<myUc> filterInfoUc(int n, std::string str, std::vector<myUc> &ucs) {
00052   std::vector<myUc> filterUc;
00053   switch (n) {
00054   case 1:
00055     // Filter by Uc Code
00056     for (const auto &uc : ucs) {
00057       if (uc.getUcCode() == str) {
00058         filterUc.push_back(uc);
00059       }
00060     }
00061     break;
00062   case 2:
00063     // Filter by Class Code
00064     for (const auto &uc : ucs) {
00065       for (const auto &classInfo : uc.getClassInfoVec()) {
00066         if (classInfo.code == str) {
00067           filterUc.push_back(uc);
00068           break;
00069         }
00070       }
00071     }
00072     break;
00073   default:
00074     errorMessage();
00075     break;
00076   }
00077   return filterUc;
00078 }
00079
00086 std::vector<myUc> orderInfoUc(int n, std::vector<myUc> &ucs) {
00087
00088   switch (n) {
00089   case 1:
00090     // Order by Uc Code Asc
00091     std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00092     break;
00093   case 2:
00094     // Order by Uc Code Desc
00095     std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00096     break;
00097   case 3:
00098     // Order by Class Code Asc
00099     std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00100     break;
00101   case 4:
00102     // Order by Class Code Desc
00103     std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00104     break;
00105   default:
00106     errorMessage();
00107     break;
00108   }
00109   return ucs;
00110 }
00111
00119 std::vector<myUc> selectUc(const std::string &str,
00120                            const std::map<std::string, myUc> &classes) {
00121   std::vector<myUc> selectedUcs;
00122
00123   for (const auto &pair : classes) {
00124     auto ucObj = pair.second;
00125
00126     if (ucObj.getUcCode() == str) {
00127       selectedUcs.push_back(ucObj);
```

```
00128     }
00129   }
00130   return selectedUcs;
00131 }
```

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

  Sorts UC information.

4.17.1 Function Documentation

4.17.1.1 compareClassesCode()

```
bool compareClassesCode (
            const myUc & uc1,
            const myUc & uc2 )
```

4.17.1.2 compareUcsCode()

```
bool compareUcsCode (
            const myUc & uc1,
            const myUc & uc2 )
```

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()

```
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 51 of file dbUcs.cpp.

```
00051                                                                    {
00052   std::vector<myUc> filterUc;
00053   switch (n) {
00054   case 1:
00055     // Filter by Uc Code
00056     for (const auto &uc : ucs) {
00057       if (uc.getUcCode() == str) {
00058         filterUc.push_back(uc);
00059       }
00060     }
00061     break;
00062   case 2:
00063     // Filter by Class Code
00064     for (const auto &uc : ucs) {
00065       for (const auto &classInfo : uc.getClassInfoVec()) {
00066         if (classInfo.code == str) {
00067           filterUc.push_back(uc);
00068           break;
00069         }
00070       }
00071     }
00072     break;
00073   default:
00074     errorMessage();
00075     break;
00076   }
00077   return filterUc;
00078 }
```

4.17.1.5 orderInfoUc()

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

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 86 of file dbUcs.cpp.

```
00086                                                    {
00087
00088   switch (n) {
00089   case 1:
00090     // Order by Uc Code Asc
00091     std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
00092     break;
00093   case 2:
00094     // Order by Uc Code Desc
00095     std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
00096     break;
00097   case 3:
00098     // Order by Class Code Asc
00099     std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
00100     break;
00101   case 4:
00102     // Order by Class Code Desc
00103     std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
00104     break;
00105   default:
00106     errorMessage();
00107     break;
00108   }
00109   return ucs;
00110 }
```

4.17.1.6 selectUc()

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

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 119 of file dbUcs.cpp.

```
00120                                                    {
00121   std::vector<myUc> selectedUcs;
00122
00123   for (const auto &pair : classes) {
00124     auto ucObj = pair.second;
00125
00126     if (ucObj.getUcCode() == str) {
00127       selectedUcs.push_back(ucObj);
00128     }
00129   }
00130   return selectedUcs;
00131 }
```

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"
00011
00012 void errorMessage();
00013
00014 bool compareClassesCode(const myUc &uc1, const myUc &uc2);
00015 bool compareUcsCode(const myUc &uc1, 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()

```
void backupFile (
            int 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 186 of file keepAllChanges.cpp.

```
00186                                {
00187
00188   std::string path = "schedule/backup/" + backups[cdBkp];
00189
00190   std::ifstream backup(path, std::ios::binary);
00191
00192   if (!backup) {
00193     std::cerr « "Error opening file" « std::endl;
00194   }
00195
00196   std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00197
00198   if (!file) {
00199     std::cerr « "Error opening file" « std::endl;
00200   }
00201
00202   file « backup.rdbuf();
00203   file.close();
00204   backup.close();
00205
00206   unsigned size = cdBkp;
00207   for (unsigned i = 0; i <= size; i++) {
00208     if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
00209       try {
00210         std::filesystem::remove("schedule/alter/" + backups[i]);
00211         std::filesystem::remove("schedule/backup/" + backups[i]);
00212       } catch (const std::filesystem::filesystem_error &e) {
00213         std::cerr « "Error to remove the file" « e.what() « std::endl;
00214       }
00215     } else {
00216       std::cout « "The file of changes not exist" « std::endl;
00217     }
00218   }
00219 }
```

4.19.1.2 getSysdate()

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

Get the system date.

Returns

A string with the system date.

Definition at line 20 of file keepAllChanges.cpp.

```
00020                          {
00021
```

```
00022   std::time_t date = std::time(0);
00023   std::tm *now = std::localtime(&date);
00024
00025   return std::to_string(now->tm_year + 1900) + "-" +
00026          std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
00027          "-" + std::to_string(now->tm_hour) + ":" +
00028          std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
00029 }
```

4.19.1.3 keepAllChanges()

```
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.

Parameters

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

Definition at line 64 of file keepAllChanges.cpp.

```
00065                                                   {
00066   makeBackup();
00067   std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00068                       ".csv",
00069                       std::ios::app);
00070   if (!alter.is_open()) {
00071     std::cerr « "Error opening file" « std::endl;
00072   }
00073
00074   while (!stackAlter.empty()) {
00075     alter « "The student: " « stackAlter.top().studentCode « " - "
00076           « stackAlter.top().studentName « " " « stackAlter.top().type
00077           « " UC: " « stackAlter.top().ucCode
00078           « " Class: " « stackAlter.top().classCode « std::endl;
00079     stackAlter.pop();
00080   }
00081
00082   std::ofstream file("schedule/students_classes.csv");
00083
00084   if (!file.is_open()) {
00085     std::cerr « "Error opening file" « std::endl;
00086   }
00087
00088   // Header
00089   file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;
00090
00091   // Write the tree in the file
00092   for (auto it = students.begin(); it != students.end(); it++) {
00093     for (auto classe : it->second.getClasses()) {
00094       file « it->second.getStudentCode() « "," « it->second.getStudentName()
00095            « "," « classe.getUcCode() « "," « classe.getClassCode()
00096            « std::endl;
00097     }
00098   }
00099 }
```

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 107 of file keepAllChanges.cpp.

```
00107                       {
00108   if (backups.size() == 0) {
00109     std::string way = "schedule/backup";
00110     for (const auto &in : std::filesystem::directory_iterator(way)) {
00111       if (std::filesystem::is_regular_file(in)) {
00112         backups.push_back(in.path().filename().string());
00113       }
00114     }
00115     std::sort(backups.begin(), backups.end(), orderVector);
00116   }
00117 }
```

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 35 of file keepAllChanges.cpp.

```
00035                     {
00036   std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00037
00038   if (!file) {
00039     std::cerr « "Error opening file" « std::endl;
00040   }
00041
00042   std::string dateString = getSysdate();
00043
00044   std::string backupName =
00045       "schedule/backup/students_classes-" + dateString + ".csv";
00046   std::ofstream backup(backupName, std::ios::binary);
00047
00048   if (!backup) {
00049     std::cerr « "Error to create a backup file" « std::endl;
00050     return;
00051   }
00052
00053   backup « file.rdbuf();
00054   file.close();
00055   backup.close();
00056 }
```

4.19.1.6 orderVector()

```
bool orderVector (
            const std::string & str1,
            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 12 of file keepAllChanges.cpp.

```
00012                                                       {
00013   return str1 > str2;
00014 }
```

4.19.1.7 printAllBackups()

```
bool printAllBackups ( )
```

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

Definition at line 122 of file keepAllChanges.cpp.

```
00122                                {
00123   if (backups.size() != 0) {
00124     std::cout « "Backups: " « std::endl;
00125     for (unsigned i = 0; i < backups.size(); i++) {
00126       std::cout « i « " - " « backups.at(i) « std::endl;
00127     }
00128     return true;
00129   } else {
00130     std::cout « "No backups" « std::endl;
00131     return false;
00132   }
00133 }
```

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 144 of file keepAllChanges.cpp.

```
00144                                {
00145   unsigned size = cdBkp;
00146   for (unsigned i = 0; i <= size; i++) {
00147     std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);
00148
00149     if (!file) {
00150       std::cerr « "Error opening file" « std::endl;
00151     }
00152
00153     std::string line;
00154
00155     while (std::getline(file, line)) {
00156       std::cout « "    " « line « std::endl;
00157     }
00158     file.close();
00159   }
00160
00161   // // Write the tree in the file
00162   // for (auto it = students.begin(); it != students.end(); it++) {
00163   //   // for (auto classe : it->second.getClasses()) {
00164   //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
00165   //   «
00166   //   //   ","
00167   //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
00168   //   std::endl;
00169   //   //   file « it->second.getStudentCode() « "," «
00170   //   //   it->second.getStudentName() « ","
00171   //   //         « classe.getUcCode() « "," « classe.getClassCode() «
00172   //   //         std::endl;
00173   //   // }
00174   // }
00175 }
```

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>
00003
00004 std::vector<std::string> backups;
00005
00012 bool orderVector(const std::string &str1, const std::string &str2) {
00013   return str1 > str2;
00014 }
00015
00020 std::string getSysdate() {
00021
00022   std::time_t date = std::time(0);
00023   std::tm *now = std::localtime(&date);
00024
00025   return std::to_string(now->tm_year + 1900) + "-" +
00026          std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
00027          "-" + std::to_string(now->tm_hour) + ":" +
00028          std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
00029 }
00030
00035 void makeBackup() {
00036   std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00037
00038   if (!file) {
00039     std::cerr « "Error opening file" « std::endl;
00040   }
00041
00042   std::string dateString = getSysdate();
00043
00044   std::string backupName =
00045       "schedule/backup/students_classes-" + dateString + ".csv";
00046   std::ofstream backup(backupName, std::ios::binary);
00047
00048   if (!backup) {
00049     std::cerr « "Error to create a backup file" « std::endl;
00050     return;
00051   }
00052
00053   backup « file.rdbuf();
00054   file.close();
00055   backup.close();
00056 }
00057
00064 void keepAllChanges(std::map<std::string, myStudent> &students,
00065                     std::stack<alter> &stackAlter) {
00066   makeBackup();
00067   std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00068                           ".csv",
00069                       std::ios::app);
00070   if (!alter.is_open()) {
00071     std::cerr « "Error opening file" « std::endl;
00072   }
00073
00074   while (!stackAlter.empty()) {
00075     alter « "The student: " « stackAlter.top().studentCode « " - "
00076           « stackAlter.top().studentName « " " « stackAlter.top().type
00077           « " UC: " « stackAlter.top().ucCode
00078           « " Class: " « stackAlter.top().classCode « std::endl;
00079     stackAlter.pop();
00080   }
00081
00082   std::ofstream file("schedule/students_classes.csv");
00083
00084   if (!file.is_open()) {
00085     std::cerr « "Error opening file" « std::endl;
00086   }
```

```
00087
00088   // Header
00089   file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;
00090
00091   // Write the tree in the file
00092   for (auto it = students.begin(); it != students.end(); it++) {
00093     for (auto classe : it->second.getClasses()) {
00094       file « it->second.getStudentCode() « "," « it->second.getStudentName()
00095            « "," « classe.getUcCode() « "," « classe.getClassCode()
00096            « std::endl;
00097     }
00098   }
00099 }
00100
00107 void listAllBackups() {
00108   if (backups.size() == 0) {
00109     std::string way = "schedule/backup";
00110     for (const auto &in : std::filesystem::directory_iterator(way)) {
00111       if (std::filesystem::is_regular_file(in)) {
00112         backups.push_back(in.path().filename().string());
00113       }
00114     }
00115     std::sort(backups.begin(), backups.end(), orderVector);
00116   }
00117 }
00118
00122 bool printAllBackups() {
00123   if (backups.size() != 0) {
00124     std::cout « "Backups: " « std::endl;
00125     for (unsigned i = 0; i < backups.size(); i++) {
00126       std::cout « i « " - " « backups.at(i) « std::endl;
00127     }
00128     return true;
00129   } else {
00130     std::cout « "No backups" « std::endl;
00131     return false;
00132   }
00133 }
00134
00144 void printChanges(int cdBkp) {
00145   unsigned size = cdBkp;
00146   for (unsigned i = 0; i <= size; i++) {
00147     std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);
00148
00149     if (!file) {
00150       std::cerr « "Error opening file" « std::endl;
00151     }
00152
00153     std::string line;
00154
00155     while (std::getline(file, line)) {
00156       std::cout « "    " « line « std::endl;
00157     }
00158     file.close();
00159   }
00160
00161   // // Write the tree in the file
00162   // for (auto it = students.begin(); it != students.end(); it++) {
00163   //   // for (auto classe : it->second.getClasses()) {
00164   //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
00165   //   «
00166   //   //   ","
00167   //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
00168   //   std::endl;
00169   //   //   file « it->second.getStudentCode() « "," «
00170   //   //   it->second.getStudentName() « ","
00171   //   //        « classe.getUcCode() « "," « classe.getClassCode() «
00172   //   //        std::endl;
00173   //   // }
00174   // }
00175 }
00176
00186 void backupFile(int cdBkp) {
00187
00188   std::string path = "schedule/backup/" + backups[cdBkp];
00189
00190   std::ifstream backup(path, std::ios::binary);
00191
00192   if (!backup) {
00193     std::cerr « "Error opening file" « std::endl;
00194   }
00195
00196   std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00197
00198   if (!file) {
00199     std::cerr « "Error opening file" « std::endl;
00200   }
```

```
00201
00202   file « backup.rdbuf();
00203   file.close();
00204   backup.close();
00205
00206   unsigned size = cdBkp;
00207   for (unsigned i = 0; i <= size; i++) {
00208     if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
00209       try {
00210         std::filesystem::remove("schedule/alter/" + backups[i]);
00211         std::filesystem::remove("schedule/backup/" + backups[i]);
00212       } catch (const std::filesystem::filesystem_error &e) {
00213         std::cerr « "Error to remove the file" « e.what() « std::endl;
00214       }
00215     } else {
00216       std::cout « "The file of changes not exist" « std::endl;
00217     }
00218   }
00219 }
```

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 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 186 of file keepAllChanges.cpp.

```
00186                                {
00187
00188   std::string path = "schedule/backup/" + backups[cdBkp];
00189
00190   std::ifstream backup(path, std::ios::binary);
00191
00192   if (!backup) {
00193     std::cerr « "Error opening file" « std::endl;
00194   }
00195
00196   std::ofstream file("schedule/students_classes.csv", std::ios::binary);
00197
00198   if (!file) {
00199     std::cerr « "Error opening file" « std::endl;
00200   }
00201
00202   file « backup.rdbuf();
00203   file.close();
00204   backup.close();
00205
00206   unsigned size = cdBkp;
00207   for (unsigned i = 0; i <= size; i++) {
00208     if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
00209       try {
00210         std::filesystem::remove("schedule/alter/" + backups[i]);
00211         std::filesystem::remove("schedule/backup/" + backups[i]);
00212       } catch (const std::filesystem::filesystem_error &e) {
00213         std::cerr « "Error to remove the file" « e.what() « std::endl;
00214       }
00215     } else {
00216       std::cout « "The file of changes not exist" « std::endl;
00217     }
00218   }
00219 }
```

4.21.1.2 getSysdate()

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

Get the system date.

Returns

A string with the system date.

Definition at line 20 of file keepAllChanges.cpp.

```
00020                          {
00021
```

```
00022   std::time_t date = std::time(0);
00023   std::tm *now = std::localtime(&date);
00024
00025   return std::to_string(now->tm_year + 1900) + "-" +
00026          std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
00027          "-" + std::to_string(now->tm_hour) + ":" +
00028          std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
00029 }
```

4.21.1.3 keepAllChanges() [1/2]

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

4.21.1.4 keepAllChanges() [2/2]

```
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.

Parameters

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

Definition at line 64 of file keepAllChanges.cpp.

```
00065                                                    {
00066   makeBackup();
00067   std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00068                       ".csv",
00069                       std::ios::app);
00070   if (!alter.is_open()) {
00071     std::cerr « "Error opening file" « std::endl;
00072   }
00073
00074   while (!stackAlter.empty()) {
00075     alter « "The student: " « stackAlter.top().studentCode « " - "
00076           « stackAlter.top().studentName « " " « stackAlter.top().type
00077           « " UC: " « stackAlter.top().ucCode
00078           « " Class: " « stackAlter.top().classCode « std::endl;
00079     stackAlter.pop();
00080   }
00081
00082   std::ofstream file("schedule/students_classes.csv");
00083
00084   if (!file.is_open()) {
00085     std::cerr « "Error opening file" « std::endl;
00086   }
00087
00088   // Header
00089   file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;
00090
00091   // Write the tree in the file
00092   for (auto it = students.begin(); it != students.end(); it++) {
00093     for (auto classe : it->second.getClasses()) {
00094       file « it->second.getStudentCode() « "," « it->second.getStudentName()
00095            « "," « classe.getUcCode() « "," « classe.getClassCode()
00096            « std::endl;
00097     }
00098   }
00099 }
```

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 107 of file keepAllChanges.cpp.

```
00107                                {
00108   if (backups.size() == 0) {
00109     std::string way = "schedule/backup";
00110     for (const auto &in : std::filesystem::directory_iterator(way)) {
00111       if (std::filesystem::is_regular_file(in)) {
00112         backups.push_back(in.path().filename().string());
00113       }
00114     }
00115     std::sort(backups.begin(), backups.end(), orderVector);
00116   }
00117 }
```

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 35 of file keepAllChanges.cpp.

```
00035                    {
00036   std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00037
00038   if (!file) {
00039     std::cerr « "Error opening file" « std::endl;
00040   }
00041
00042   std::string dateString = getSysdate();
00043
00044   std::string backupName =
00045       "schedule/backup/students_classes-" + dateString + ".csv";
00046   std::ofstream backup(backupName, std::ios::binary);
00047
00048   if (!backup) {
00049     std::cerr « "Error to create a backup file" « std::endl;
00050     return;
00051   }
00052
00053   backup « file.rdbuf();
00054   file.close();
00055   backup.close();
00056 }
```

4.21.1.7 orderVector()

```
bool orderVector (
            const std::string & str1,
            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 12 of file keepAllChanges.cpp.

```
00012                                                       {
00013   return str1 > str2;
00014 }
```

4.21.1.8 printAllBackups()

```
bool printAllBackups ( )
```

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

Definition at line 122 of file keepAllChanges.cpp.

```
00122                          {
00123   if (backups.size() != 0) {
00124     std::cout « "Backups: " « std::endl;
00125     for (unsigned i = 0; i < backups.size(); i++) {
00126       std::cout « i « " - " « backups.at(i) « std::endl;
00127     }
00128     return true;
00129   } else {
00130     std::cout « "No backups" « std::endl;
00131     return false;
00132   }
00133 }
```

4.21.1.9 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 144 of file keepAllChanges.cpp.

```
00144                                {
00145   unsigned size = cdBkp;
00146   for (unsigned i = 0; i <= size; i++) {
00147     std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);
00148
00149     if (!file) {
00150       std::cerr « "Error opening file" « std::endl;
00151     }
00152
00153     std::string line;
00154
00155     while (std::getline(file, line)) {
00156       std::cout « "    " « line « std::endl;
00157     }
00158     file.close();
00159   }
00160
00161   // // Write the tree in the file
00162   // for (auto it = students.begin(); it != students.end(); it++) {
00163   //   // for (auto classe : it->second.getClasses()) {
00164   //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
```

```
00165   //   «
00166   //   //   ","
00167   //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
00168   //   std::endl;
00169   //   //   file « it->second.getStudentCode() « "," «
00170   //   //   it->second.getStudentName() « ","
00171   //   //        « classe.getUcCode() « "," « classe.getClassCode() «
00172   //   //        std::endl;
00173   //   // }
00174   // }
00175 }
```

4.22 keepAllChanges.h

Go to the documentation of this file.

```
00001 #ifndef KEEPALLCHANGES_H
00002 #define KEEPALLCHANGES_H
00003
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"
00015
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

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

  Print student information.
- void printStudents (const std::vector< myStudent > &students)

  Print students information from a vector.
- 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 183 of file print.cpp.

```
00184                                                                    {
00185
00186   auto it_count = count.find(ucCode);
00187   std::list<std::string> free_classes;
00188
00189   if (it_count != count.end()) {
00190     free_classes = valideFreeClass(it_count);
00191     std::cout « "   Classes: " « std::endl;
00192
00193     if (!free_classes.empty()) {
00194       for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00195            it_list++) {
00196         std::cout « "      " « *it_list « std::endl;
00197       }
00198     } else {
00199       std::cout « "      No classes available" « std::endl;
00200     }
00201   } else {
00202     std::cout « " Uc not found" « std::endl;
00203   }
00204 }
```

4.23.1.2 printStudent()

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

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.

```
00014                                                                    {
00015   std::cout « "Student Code | Student Name"
00016             « std::endl;
00017
00018   for (const auto &studentPair : students) {
00019     const myStudent &student = studentPair.second;
00020     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00021     std::cout « "   " « "Classes: " « std::endl;
00022     for(const auto &classe : student.getClasses()) {
00023       std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00024     }
00025
00026   }
00027 }
```

4.23.1.3 printStudentClasses()

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

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 63 of file print.cpp.

```
00063                                                                        {
00064   system("clear");
00065   std::cout « "\nCode: " « it->first « " - ";
00066   std::cout « "Name: " « it->second.getStudentName() « std::endl;
00067   std::cout « "Classes: " « std::endl;
00068   for (const auto &classe : it->second.getClasses()) {
00069     std::cout « "  " « classe.getUcCode() « " - " « classe.getClassCode()
00070               « std::endl;
00071   }
00072 }
```

4.23.1.4 printStudents()

```
void printStudents (
            const std::vector< myStudent > & students )
```

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 38 of file print.cpp.

```
00038                                                                    {
00039   std::cout « "Student Code | Student Name"
00040             « std::endl;
00041
```

```
00042   if (students.empty()) {
00043     std::cout « "Empty vector ucs" « std::endl;
00044   }
00045
00046   for (const auto &student : students) {
00047     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00048     std::cout « "   " « "Classes: " « std::endl;
00049     for(const auto &classe : student.getClasses()) {
00050       std::cout « "   " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051     }
00052   }
00053 }
```

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 215 of file print.cpp.

```
00216                                                                                   {
00217   auto orderClasses = orderStudentClass(it, classes);
00218   std::cout « "\nSchedules: " « std::endl;
00219   for (const auto &pair : orderClasses) {
00220     std::string day = weekDayString(pair.first);
00221     std::cout « "Day: " « day « std::endl;
00222     for (const auto &info : pair.second) {
00223       std::cout « info.code « " - ";
00224       std::cout « info.startTime « " to ";
00225       std::cout « info.startTime + info.duration « " - ";
00226       std::cout « info.type « std::endl;
00227     }
00228     std::cout « std::endl;
00229   }
00230 }
```

4.23.1.6 printUcClasses()

```
void printUcClasses (
            const std::vector< myUc > & 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 84 of file print.cpp.

```
00084                                                       {
00085   std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00086             « std::endl;
00087
00088   for (const auto &classes : ucVector) {
00089
00090     auto infoVec = classes.getClassInfoVec();
00091     for (const auto &classInfo : infoVec) {
00092       std::string type = classInfo.type;
00093       std::string day = classInfo.day;
00094       int dayInt = classInfo.dayInt;
00095       double startTime = classInfo.startTime;
00096       double duration = classInfo.duration;
00097       std::cout « classes.getUcCode() « " | " « classes.getClassCode()
00098                 « " | " « type « " | " « day « " | " « dayInt « " | "
00099                 « startTime « " | " « duration « std::endl;
00100     }
00101   }
00102 }
```

4.23.1.7 printUcs()

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

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 112 of file print.cpp.

```
00112                                             {
00113   std::cout « "UcCode | ClassCode" « std::endl;
00114
00115   for (const auto &uc : ucs) {
00116     std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
00117   }
00118 }
```

4.23.1.8 valideFreeClass()

```
std::list< std::string > valideFreeClass (
            std::map< std::string, std::vector< classQtd > >::iterator it_count )
```

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 130 of file print.cpp.

```
00131                                                                                                {
00132   int min = INT_MAX;
00133   std::list<std::string> free_classes;
00134
00135   // first verify the class with the minimum number of students
00136   for (auto &classe : it_count->second) {
00137     if (classe.qtd < min) {
00138       min = classe.qtd;
00139     }
00140   }
00141   // then verify if the class is able to accept new students and add to the
00142   // list
00143   for (auto &classe : it_count->second) {
00144     if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
00145       free_classes.push_back(classe.classCode);
00146     }
00147   }
00148
00149   // return list
00150   return free_classes;
00151 }
```

4.23.1.9 verifyClassCode()

```
bool verifyClassCode (
              std::string classCode,
              std::string ucCode,
              std::map< std::string, std::vector< classQtd > > & count )
```

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 165 of file print.cpp.

```
00166                                                                                                {
00167   auto it_count = count.find(ucCode);
00168
00169   if (it_count != count.end()) {
00170     std::list<std::string> free_classes = valideFreeClass(it_count);
00171     for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00172          it_list++) {
00173       if (*it_list == classCode) {
00174         return true;
00175       }
00176     }
00177   } else {
00178     std::cout << "Error in find uc" << std::endl;
00179   }
00180   return false;
00181 }
```

4.23.2 Variable Documentation

4.23.2.1 equilibre

```
int equilibre = 3
```

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"
00002
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"
00016             « std::endl;
00017
00018   for (const auto &studentPair : students) {
00019     const myStudent &student = studentPair.second;
00020     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00021     std::cout « "   " « "Classes: " « std::endl;
00022     for(const auto &classe : student.getClasses()) {
00023       std::cout « "   " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00024     }
00025
00026   }
00027 }
00028
00038 void printStudents(const std::vector<myStudent> &students) {
00039   std::cout « "Student Code | Student Name"
00040             « std::endl;
00041
00042   if (students.empty()) {
00043     std::cout « "Empty vector ucs" « std::endl;
00044   }
00045
00046   for (const auto &student : students) {
00047     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00048     std::cout « "   " « "Classes: " « std::endl;
00049     for(const auto &classe : student.getClasses()) {
00050       std::cout « "   " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051     }
00052   }
00053 }
00054
00063 void printStudentClasses(std::map<std::string, myStudent>::iterator &it) {
00064   system("clear");
00065   std::cout « "\nCode: " « it->first « " - ";
00066   std::cout « "Name: " « it->second.getStudentName() « std::endl;
00067   std::cout « "Classes: " « std::endl;
00068   for (const auto &classe : it->second.getClasses()) {
00069     std::cout « "  " « classe.getUcCode() « " - " « classe.getClassCode()
00070               « std::endl;
00071   }
00072 }
00073
00084 void printUcClasses(const std::vector<myUc> &ucVector) {
00085   std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00086             « std::endl;
00087
00088   for (const auto &classes : ucVector) {
00089
00090     auto infoVec = classes.getClassInfoVec();
00091     for (const auto &classInfo : infoVec) {
00092       std::string type = classInfo.type;
00093       std::string day = classInfo.day;
00094       int dayInt = classInfo.dayInt;
00095       double startTime = classInfo.startTime;
00096       double duration = classInfo.duration;
00097       std::cout « classes.getUcCode() « " | " « classes.getClassCode()
00098                 « " | " « type « " | " « day « " | " « dayInt « " | "
00099                 « startTime « " | " « duration « std::endl;
00100     }
```

```
00101   }
00102 }
00103
00112 void printUcs(const std::vector<myUc> &ucs) {
00113   std::cout « "UcCode | ClassCode" « std::endl;
00114
00115   for (const auto &uc : ucs) {
00116     std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
00117   }
00118 }
00119
00130 std::list<std::string> valideFreeClass(
00131     std::map<std::string, std::vector<classQtd»::iterator it_count) {
00132   int min = INT_MAX;
00133   std::list<std::string> free_classes;
00134
00135   // first verify the class with the minimum number of students
00136   for (auto &classe : it_count->second) {
00137     if (classe.qtd < min) {
00138       min = classe.qtd;
00139     }
00140   }
00141   // then verify if the class is able to accept new students and add to the
00142   // list
00143   for (auto &classe : it_count->second) {
00144     if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
00145       free_classes.push_back(classe.classCode);
00146     }
00147   }
00148
00149   // return list
00150   return free_classes;
00151 }
00152
00165 bool verifyClassCode(std::string classCode, std::string ucCode,
00166                      std::map<std::string, std::vector<classQtd» &count) {
00167   auto it_count = count.find(ucCode);
00168
00169   if (it_count != count.end()) {
00170     std::list<std::string> free_classes = valideFreeClass(it_count);
00171     for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00172          it_list++) {
00173       if (*it_list == classCode) {
00174         return true;
00175       }
00176     }
00177   } else {
00178     std::cout « "Error in find uc" « std::endl;
00179   }
00180   return false;
00181 }
00182
00183 void printFreeClasses(std::string ucCode,
00184                       std::map<std::string, std::vector<classQtd» &count) {
00185
00186   auto it_count = count.find(ucCode);
00187   std::list<std::string> free_classes;
00188
00189   if (it_count != count.end()) {
00190     free_classes = valideFreeClass(it_count);
00191     std::cout « "   Classes: " « std::endl;
00192
00193     if (!free_classes.empty()) {
00194       for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00195            it_list++) {
00196         std::cout « "      " « *it_list « std::endl;
00197       }
00198     } else {
00199       std::cout « "      No classes available" « std::endl;
00200     }
00201   } else {
00202     std::cout « " Uc not found" « std::endl;
00203   }
00204 }
00205
00215 void printStudentSchedules(std::map<std::string, myStudent>::iterator &it,
00216                            std::map<std::string, myUc> &classes) {
00217   auto orderClasses = orderStudentClass(it, classes);
00218   std::cout « "\nSchedules: " « std::endl;
00219   for (const auto &pair : orderClasses) {
00220     std::string day = weekDayString(pair.first);
00221     std::cout « "Day: " « day « std::endl;
00222     for (const auto &info : pair.second) {
00223       std::cout « info.code « " - ";
00224       std::cout « info.startTime « " to ";
00225       std::cout « info.startTime + info.duration « " - ";
00226       std::cout « info.type « std::endl;
```

```
00227     }
00228     std::cout « std::endl;
00229   }
00230 }
```

4.25 src/inputoutput/print.h File Reference

```
#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
#include <list>
#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.

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

4.25.1.2 printFreeClasses()

```
void printFreeClasses (
            std::string ucCode,
            std::map< std::string, std::vector< classQtd > > & count )
```

Definition at line 183 of file print.cpp.

```
00184                                                                    {
00185
00186   auto it_count = count.find(ucCode);
00187   std::list<std::string> free_classes;
00188
00189   if (it_count != count.end()) {
00190     free_classes = valideFreeClass(it_count);
00191     std::cout « "   Classes: " « std::endl;
00192
00193     if (!free_classes.empty()) {
00194       for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00195            it_list++) {
00196         std::cout « "      " « *it_list « std::endl;
00197       }
00198     } else {
00199       std::cout « "      No classes available" « std::endl;
00200     }
00201   } else {
00202     std::cout « " Uc not found" « std::endl;
00203   }
00204 }
```

4.25.1.3 printStudent()

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

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.

```
00014                                                                {
00015   std::cout « "Student Code | Student Name"
00016             « std::endl;
00017
00018   for (const auto &studentPair : students) {
00019     const myStudent &student = studentPair.second;
00020     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00021     std::cout « "   " « "Classes: " « std::endl;
00022     for(const auto &classe : student.getClasses()) {
00023       std::cout « "   " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00024     }
00025
00026   }
00027 }
```

4.25.1.4 printStudentClasses()

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

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 63 of file print.cpp.

```
00063                                                                    {
00064   system("clear");
00065   std::cout « "\nCode: " « it->first « " - ";
00066   std::cout « "Name: " « it->second.getStudentName() « std::endl;
00067   std::cout « "Classes: " « std::endl;
00068   for (const auto &classe : it->second.getClasses()) {
00069     std::cout « "   " « classe.getUcCode() « " - " « classe.getClassCode()
00070               « std::endl;
00071   }
00072 }
```

4.25.1.5 printStudents()

```
void printStudents (
            const std::vector< myStudent > & students )
```

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 38 of file print.cpp.

```
00038                                                          {
00039   std::cout « "Student Code | Student Name"
00040             « std::endl;
00041
00042   if (students.empty()) {
00043     std::cout « "Empty vector ucs" « std::endl;
00044   }
00045
00046   for (const auto &student : students) {
00047     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00048     std::cout « "    " « "Classes: " « std::endl;
00049     for(const auto &classe : student.getClasses()) {
00050       std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00051     }
00052   }
00053 }
```

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 215 of file print.cpp.

```
00216                                                                    {
00217   auto orderClasses = orderStudentClass(it, classes);
00218   std::cout « "\nSchedules: " « std::endl;
00219   for (const auto &pair : orderClasses) {
00220     std::string day = weekDayString(pair.first);
00221     std::cout « "Day: " « day « std::endl;
00222     for (const auto &info : pair.second) {
00223       std::cout « info.code « " - ";
00224       std::cout « info.startTime « " to ";
00225       std::cout « info.startTime + info.duration « " - ";
00226       std::cout « info.type « std::endl;
00227     }
00228     std::cout « std::endl;
00229   }
00230 }
```

4.25.1.7 printUcClasses()

```
void printUcClasses (
            const std::vector< myUc > & 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 84 of file print.cpp.

```
00084                                                        {
00085   std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00086             « std::endl;
00087
00088   for (const auto &classes : ucVector) {
00089
00090     auto infoVec = classes.getClassInfoVec();
00091     for (const auto &classInfo : infoVec) {
00092       std::string type = classInfo.type;
00093       std::string day = classInfo.day;
00094       int dayInt = classInfo.dayInt;
00095       double startTime = classInfo.startTime;
00096       double duration = classInfo.duration;
00097       std::cout « classes.getUcCode() « " | " « classes.getClassCode()
00098                 « " | " « type « " | " « day « " | " « dayInt « " | "
00099                 « startTime « " | " « duration « std::endl;
00100     }
00101   }
00102 }
```

4.25.1.8 printUcs()

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

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 112 of file print.cpp.

```
00112                                                {
00113   std::cout « "UcCode | ClassCode" « std::endl;
00114
00115   for (const auto &uc : ucs) {
00116     std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
00117   }
00118 }
```

4.25.1.9 valideFreeClass()

```
std::list< std::string > valideFreeClass (
            std::map< std::string, std::vector< classQtd > >::iterator it_count )
```

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 130 of file print.cpp.

```
00131                                                                    {
00132   int min = INT_MAX;
00133   std::list<std::string> free_classes;
00134
00135   // first verify the class with the minimum number of students
00136   for (auto &classe : it_count->second) {
00137     if (classe.qtd < min) {
00138       min = classe.qtd;
00139     }
00140   }
00141   // then verify if the class is able to accept new students and add to the
00142   // list
00143   for (auto &classe : it_count->second) {
00144     if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
00145       free_classes.push_back(classe.classCode);
00146     }
00147   }
00148
00149   // return list
00150   return free_classes;
00151 }
```

4.25.1.10 verifyClassCode()

```
bool verifyClassCode (
            std::string classCode,
            std::string ucCode,
            std::map< std::string, std::vector< classQtd > > & count )
```

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 165 of file print.cpp.

```
00166                                                                                {
00167   auto it_count = count.find(ucCode);
00168
00169   if (it_count != count.end()) {
00170     std::list<std::string> free_classes = valideFreeClass(it_count);
00171     for (auto it_list = free_classes.begin(); it_list != free_classes.end();
00172          it_list++) {
00173       if (*it_list == classCode) {
00174         return true;
00175       }
00176     }
00177   } else {
00178     std::cout « "Error in find uc" « std::endl;
00179   }
00180   return false;
00181 }
```

4.25.1.11 workingMessage()

```
void workingMessage ( )
```

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);
00021
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,
00029                       std::map<std::string, std::vector<classQtd» &count);
00030 std::list<std::string> valideFreeClass(
00031     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);
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 179 of file read.cpp.

```
00179                                          {
00180   std::string line;
00181   std::map<std::string, myUc> classes;
00182   std::map<std::string, int> dayToInt = {
00183       {"Sunday", 1},   {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
00184       {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};
00185
00186   std::ifstream file("schedule/classes.csv");
00187   if (!file.is_open()) {
00188     errorMessageFile();
00189   }
00190
00191   bool header = true;
00192   while (std::getline(file, line)) {
00193     if (header) {
00194       header = false;
00195       continue;
00196     }
00197     std::istringstream ss(line);
00198     std::string classCode, ucCode, day, type;
00199     double startTime, duration;
00200     int dayInt = 0;
00201
00202     std::getline(ss, classCode, ',');
00203     std::getline(ss, ucCode, ',');
00204     std::getline(ss, day, ',');
00205     ss » startTime;
00206     ss.ignore();
00207     ss » duration;
00208     ss.ignore();
00209     std::getline(ss, type);
00210
```

```
00211     type.erase(std::find_if(type.rbegin(), type.rend(),
00212                             [](unsigned char ch) { return !std::isspace(ch); })
00213                     .base(),
00214                 type.end());
00215
00216     auto it1 = dayToInt.find(day);
00217     if (it1 != dayToInt.end()) {
00218       dayInt = it1->second;
00219     } else {
00220       std::cout « "Invalid day: " « day « std::endl;
00221     }
00222
00223     // Check if the class code already exists in the map
00224     auto it2 = classes.find(ucCode + classCode);
00225     if (it2 != classes.end()) {
00226       it2->second.addClassInfo(type, day, dayInt, startTime, duration);
00227     } else {
00228       myUc newUcClass;
00229       newUcClass.setUcCode(ucCode);
00230       newUcClass.addClass(classCode);
00231       newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
00232       classes[ucCode + classCode] = newUcClass;
00233     }
00234   }
00235   return classes;
00236 }
```

4.27.1.2 readStudents()

```
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.

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 14 of file read.cpp.

```
00014                                                                    {
00015   std::string line;
00016   std::map<std::string, myStudent> students;
00017
00018   std::ifstream file("schedule/students_classes.csv");
00019   if (!file.is_open()) {
00020     errorMessageFile();
00021   }
00022
00023   bool header = true;
00024   while (std::getline(file, line)) {
00025     if (header) {
00026       header = false;
00027       continue;
00028     }
00029     std::istringstream ss(line);
00030
00031     std::string studentCode, studentName, ucCode, classCode;
00032
00033     std::getline(ss, studentCode, ',');
00034     std::getline(ss, studentName, ',');
00035     std::getline(ss, ucCode, ',');
00036     std::getline(ss, classCode);
00037
00038     classCode.erase(
```

```
00039         std::find_if(classCode.rbegin(), classCode.rend(),
00040                      [](unsigned char ch) { return !std::isspace(ch); })
00041             .base(),
00042         classCode.end());
00043
00044     auto it = students.find(studentCode);
00045     if (it != students.end()) {
00046       it->second.addClass(myUc(ucCode, classCode));
00047     } else {
00048       myStudent newStudent(studentCode, studentName);
00049       newStudent.addClass(myUc(ucCode, classCode));
00050       students[studentCode] = newStudent;
00051     }
00052
00053     // verify if the uc exists in the count tree
00054     auto it_count = count.find(ucCode);
00055
00056     // if not exists, add the class with one student in the count tree
00057     if (it_count == count.end()) {
00058       std::vector<classQtd> classVec;
00059       classVec.push_back({classCode, 1});
00060       count.emplace(ucCode, classVec);
00061     } else {
00062       // if uc exist, then verify if the class exists in the vector
00063       bool exist = false;
00064       for (auto &class_it : it_count->second) {
00065         // if exists, add +1 in the qtd
00066         if (class_it.classCode == classCode) {
00067           class_it.qtd++;
00068           exist = true;
00069           break;
00070         }
00071       }
00072       // if not exists, add the class with one student in the vector
00073       if (!exist) {
00074         it_count->second.push_back({classCode, 1});
00075       }
00076     }
00077   }
00078   file.close();
00079
00080   return students;
00081 }
```

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

A map of UCs and their associated classes.

Definition at line 94 of file read.cpp.

```
00094                                                                    {
00095   std::string line;
00096   std::map<std::string, std::vector<myUc» ucClasses;
00097
00098   std::ifstream file("schedule/classes_per_uc.csv");
00099   if (!file.is_open()) {
00100     errorMessageFile();
00101   }
```

```
00102
00103   bool header = true;
00104   while (std::getline(file, line)) {
00105     // testing
00106     // std::cout « "line" « std::endl;
00107
00108     if (header) {
00109       header = false;
00110       continue;
00111     }
00112     std::istringstream ss(line);
00113     std::string ucCode, classCode;
00114
00115     std::getline(ss, ucCode, ',');
00116     std::getline(ss, classCode, ',');
00117
00118     auto it = ucClasses.find(ucCode);
00119
00120     classCode.erase(
00121         std::find_if(classCode.rbegin(), classCode.rend(),
00122                      [](unsigned char ch) { return !std::isspace(ch); })
00123             .base(),
00124         classCode.end());
00125
00126     if (it != ucClasses.end()) {
00127       // exist
00128       myUc newUc;
00129       newUc.setUcCode(ucCode);
00130       newUc.setClassCode(classCode);
00131       it->second.push_back(newUc);
00132     } else {
00133       // doesnt exist
00134       std::vector<myUc> ucVector;
00135       myUc newUc;
00136       newUc.setUcCode(ucCode);
00137       newUc.setClassCode(classCode);
00138       ucVector.push_back(newUc);
00139       ucClasses[ucCode] = ucVector;
00140     }
00141
00142     bool exist = false;
00143
00144     // try to find the uc in the count tree
00145     auto it_count = count.find(ucCode);
00146
00147     // if found, verify if the class exists in the vector
00148     if (it_count != count.end()) {
00149       for (auto &class_it : it_count->second) {
00150         if (class_it.classCode == classCode) {
00151           exist = true;
00152         }
00153       }
00154       // if exist uc in the count tree, but not exist the class, add the class
00155       // with 0 students
00156       if (!exist) {
00157         it_count->second.push_back({classCode, 0});
00158       }
00159       // if not found, add the uc and class with 0 students
00160     } else {
00161       std::vector<classQtd> classVec;
00162       classVec.push_back({classCode, 0});
00163       count.emplace(ucCode, classVec);
00164     }
00165   }
00166   file.close();
00167
00168   return ucClasses;
00169 }
```

4.28 read.cpp

Go to the documentation of this file.

```
00001 #include "read.h"
00002
00003
00013 std::map<std::string, myStudent>
00014 readStudents(std::map<std::string, std::vector<classQtd» &count) {
00015   std::string line;
00016   std::map<std::string, myStudent> students;
00017
00018   std::ifstream file("schedule/students_classes.csv");
```

```
00019   if (!file.is_open()) {
00020     errorMessageFile();
00021   }
00022
00023   bool header = true;
00024   while (std::getline(file, line)) {
00025     if (header) {
00026       header = false;
00027       continue;
00028     }
00029     std::istringstream ss(line);
00030
00031     std::string studentCode, studentName, ucCode, classCode;
00032
00033     std::getline(ss, studentCode, ',');
00034     std::getline(ss, studentName, ',');
00035     std::getline(ss, ucCode, ',');
00036     std::getline(ss, classCode);
00037
00038     classCode.erase(
00039         std::find_if(classCode.rbegin(), classCode.rend(),
00040                      [](unsigned char ch) { return !std::isspace(ch); })
00041             .base(),
00042         classCode.end());
00043
00044     auto it = students.find(studentCode);
00045     if (it != students.end()) {
00046       it->second.addClass(myUc(ucCode, classCode));
00047     } else {
00048       myStudent newStudent(studentCode, studentName);
00049       newStudent.addClass(myUc(ucCode, classCode));
00050       students[studentCode] = newStudent;
00051     }
00052
00053     // verify if the uc exists in the count tree
00054     auto it_count = count.find(ucCode);
00055
00056     // if not exists, add the class with one student in the count tree
00057     if (it_count == count.end()) {
00058       std::vector<classQtd> classVec;
00059       classVec.push_back({classCode, 1});
00060       count.emplace(ucCode, classVec);
00061     } else {
00062       // if uc exist, then verify if the class exists in the vector
00063       bool exist = false;
00064       for (auto &class_it : it_count->second) {
00065         // if exists, add +1 in the qtd
00066         if (class_it.classCode == classCode) {
00067           class_it.qtd++;
00068           exist = true;
00069           break;
00070         }
00071       }
00072       // if not exists, add the class with one student in the vector
00073       if (!exist) {
00074         it_count->second.push_back({classCode, 1});
00075       }
00076     }
00077   }
00078   file.close();
00079
00080   return students;
00081 }
00082
00093 std::map<std::string, std::vector<myUc»
00094 readUcs(std::map<std::string, std::vector<classQtd» &count) {
00095   std::string line;
00096   std::map<std::string, std::vector<myUc» ucClasses;
00097
00098   std::ifstream file("schedule/classes_per_uc.csv");
00099   if (!file.is_open()) {
00100     errorMessageFile();
00101   }
00102
00103   bool header = true;
00104   while (std::getline(file, line)) {
00105     // testing
00106     // std::cout « "line" « std::endl;
00107
00108     if (header) {
00109       header = false;
00110       continue;
00111     }
00112     std::istringstream ss(line);
00113     std::string ucCode, classCode;
00114
00115     std::getline(ss, ucCode, ',');
```

```
00116     std::getline(ss, classCode, ',');
00117
00118     auto it = ucClasses.find(ucCode);
00119
00120     classCode.erase(
00121         std::find_if(classCode.rbegin(), classCode.rend(),
00122                      [](unsigned char ch) { return !std::isspace(ch); })
00123             .base(),
00124         classCode.end());
00125
00126     if (it != ucClasses.end()) {
00127       // exist
00128       myUc newUc;
00129       newUc.setUcCode(ucCode);
00130       newUc.setClassCode(classCode);
00131       it->second.push_back(newUc);
00132     } else {
00133       // doesnt exist
00134       std::vector<myUc> ucVector;
00135       myUc newUc;
00136       newUc.setUcCode(ucCode);
00137       newUc.setClassCode(classCode);
00138       ucVector.push_back(newUc);
00139       ucClasses[ucCode] = ucVector;
00140     }
00141
00142     bool exist = false;
00143
00144     // try to find the uc in the count tree
00145     auto it_count = count.find(ucCode);
00146
00147     // if found, verify if the class exists in the vector
00148     if (it_count != count.end()) {
00149       for (auto &class_it : it_count->second) {
00150         if (class_it.classCode == classCode) {
00151           exist = true;
00152         }
00153       }
00154       // if exist uc in the count tree, but not exist the class, add the class
00155       // with 0 students
00156       if (!exist) {
00157         it_count->second.push_back({classCode, 0});
00158       }
00159       // if not found, add the uc and class with 0 students
00160     } else {
00161       std::vector<classQtd> classVec;
00162       classVec.push_back({classCode, 0});
00163       count.emplace(ucCode, classVec);
00164     }
00165   }
00166   file.close();
00167
00168   return ucClasses;
00169 }
00170
00179 std::map<std::string, myUc> readSchedules() {
00180   std::string line;
00181   std::map<std::string, myUc> classes;
00182   std::map<std::string, int> dayToInt = {
00183       {"Sunday", 1},   {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
00184       {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};
00185
00186   std::ifstream file("schedule/classes.csv");
00187   if (!file.is_open()) {
00188     errorMessageFile();
00189   }
00190
00191   bool header = true;
00192   while (std::getline(file, line)) {
00193     if (header) {
00194       header = false;
00195       continue;
00196     }
00197     std::istringstream ss(line);
00198     std::string classCode, ucCode, day, type;
00199     double startTime, duration;
00200     int dayInt = 0;
00201
00202     std::getline(ss, classCode, ',');
00203     std::getline(ss, ucCode, ',');
00204     std::getline(ss, day, ',');
00205     ss » startTime;
00206     ss.ignore();
00207     ss » duration;
00208     ss.ignore();
00209     std::getline(ss, type);
00210
```

```
00211     type.erase(std::find_if(type.rbegin(), type.rend(),
00212                             [](unsigned char ch) { return !std::isspace(ch); })
00213                    .base(),
00214                type.end());
00215
00216     auto it1 = dayToInt.find(day);
00217     if (it1 != dayToInt.end()) {
00218       dayInt = it1->second;
00219     } else {
00220       std::cout « "Invalid day: " « day « std::endl;
00221     }
00222
00223     // Check if the class code already exists in the map
00224     auto it2 = classes.find(ucCode + classCode);
00225     if (it2 != classes.end()) {
00226       it2->second.addClassInfo(type, day, dayInt, startTime, duration);
00227     } else {
00228       myUc newUcClass;
00229       newUcClass.setUcCode(ucCode);
00230       newUcClass.addClass(classCode);
00231       newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
00232       classes[ucCode + classCode] = newUcClass;
00233     }
00234   }
00235   return classes;
00236 }
```

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()

```
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 }
```

4.29.1.2 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 }
```

4.29.1.3 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 179 of file read.cpp.

```
00179                                         {
00180   std::string line;
00181   std::map<std::string, myUc> classes;
00182   std::map<std::string, int> dayToInt = {
00183       {"Sunday", 1},   {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
00184       {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};
00185
00186   std::ifstream file("schedule/classes.csv");
00187   if (!file.is_open()) {
00188     errorMessageFile();
00189   }
00190
00191   bool header = true;
00192   while (std::getline(file, line)) {
00193     if (header) {
00194       header = false;
00195       continue;
00196     }
00197     std::istringstream ss(line);
00198     std::string classCode, ucCode, day, type;
00199     double startTime, duration;
00200     int dayInt = 0;
00201
00202     std::getline(ss, classCode, ',');
00203     std::getline(ss, ucCode, ',');
00204     std::getline(ss, day, ',');
00205     ss » startTime;
00206     ss.ignore();
00207     ss » duration;
00208     ss.ignore();
00209     std::getline(ss, type);
00210
00211     type.erase(std::find_if(type.rbegin(), type.rend(),
00212                             [](unsigned char ch) { return !std::isspace(ch); })
00213                 .base(),
00214                type.end());
00215
00216     auto it1 = dayToInt.find(day);
00217     if (it1 != dayToInt.end()) {
00218       dayInt = it1->second;
00219     } else {
00220       std::cout « "Invalid day: " « day « std::endl;
00221     }
00222
00223     // Check if the class code already exists in the map
00224     auto it2 = classes.find(ucCode + classCode);
00225     if (it2 != classes.end()) {
00226       it2->second.addClassInfo(type, day, dayInt, startTime, duration);
```

```
00227     } else {
00228       myUc newUcClass;
00229       newUcClass.setUcCode(ucCode);
00230       newUcClass.addClass(classCode);
00231       newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
00232       classes[ucCode + classCode] = newUcClass;
00233     }
00234   }
00235   return classes;
00236 }
```

4.29.1.4 readStudents()

```
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.

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 14 of file read.cpp.

```
00014                                                                    {
00015   std::string line;
00016   std::map<std::string, myStudent> students;
00017
00018   std::ifstream file("schedule/students_classes.csv");
00019   if (!file.is_open()) {
00020     errorMessageFile();
00021   }
00022
00023   bool header = true;
00024   while (std::getline(file, line)) {
00025     if (header) {
00026       header = false;
00027       continue;
00028     }
00029     std::istringstream ss(line);
00030
00031     std::string studentCode, studentName, ucCode, classCode;
00032
00033     std::getline(ss, studentCode, ',');
00034     std::getline(ss, studentName, ',');
00035     std::getline(ss, ucCode, ',');
00036     std::getline(ss, classCode);
00037
00038     classCode.erase(
00039         std::find_if(classCode.rbegin(), classCode.rend(),
00040                      [](unsigned char ch) { return !std::isspace(ch); })
00041             .base(),
00042         classCode.end());
00043
00044     auto it = students.find(studentCode);
00045     if (it != students.end()) {
00046       it->second.addClass(myUc(ucCode, classCode));
00047     } else {
00048       myStudent newStudent(studentCode, studentName);
00049       newStudent.addClass(myUc(ucCode, classCode));
00050       students[studentCode] = newStudent;
00051     }
00052
00053     // verify if the uc exists in the count tree
00054     auto it_count = count.find(ucCode);
```

```
00055
00056     // if not exists, add the class with one student in the count tree
00057     if (it_count == count.end()) {
00058       std::vector<classQtd> classVec;
00059       classVec.push_back({classCode, 1});
00060       count.emplace(ucCode, classVec);
00061     } else {
00062       // if uc exist, then verify if the class exists in the vector
00063       bool exist = false;
00064       for (auto &class_it : it_count->second) {
00065         // if exists, add +1 in the qtd
00066         if (class_it.classCode == classCode) {
00067           class_it.qtd++;
00068           exist = true;
00069           break;
00070         }
00071       }
00072       // if not exists, add the class with one student in the vector
00073       if (!exist) {
00074         it_count->second.push_back({classCode, 1});
00075       }
00076     }
00077   }
00078   file.close();
00079
00080   return students;
00081 }
```

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

A map of UCs and their associated classes.

Definition at line 94 of file read.cpp.

```
00094                                                                    {
00095   std::string line;
00096   std::map<std::string, std::vector<myUc» ucClasses;
00097
00098   std::ifstream file("schedule/classes_per_uc.csv");
00099   if (!file.is_open()) {
00100     errorMessageFile();
00101   }
00102
00103   bool header = true;
00104   while (std::getline(file, line)) {
00105     // testing
00106     // std::cout « "line" « std::endl;
00107
00108     if (header) {
00109       header = false;
00110       continue;
00111     }
00112     std::istringstream ss(line);
00113     std::string ucCode, classCode;
00114
00115     std::getline(ss, ucCode, ',');
00116     std::getline(ss, classCode, ',');
00117
```

```
00118     auto it = ucClasses.find(ucCode);
00119
00120     classCode.erase(
00121         std::find_if(classCode.rbegin(), classCode.rend(),
00122                      [](unsigned char ch) { return !std::isspace(ch); })
00123             .base(),
00124         classCode.end());
00125
00126     if (it != ucClasses.end()) {
00127       // exist
00128       myUc newUc;
00129       newUc.setUcCode(ucCode);
00130       newUc.setClassCode(classCode);
00131       it->second.push_back(newUc);
00132     } else {
00133       // doesnt exist
00134       std::vector<myUc> ucVector;
00135       myUc newUc;
00136       newUc.setUcCode(ucCode);
00137       newUc.setClassCode(classCode);
00138       ucVector.push_back(newUc);
00139       ucClasses[ucCode] = ucVector;
00140     }
00141
00142     bool exist = false;
00143
00144     // try to find the uc in the count tree
00145     auto it_count = count.find(ucCode);
00146
00147     // if found, verify if the class exists in the vector
00148     if (it_count != count.end()) {
00149       for (auto &class_it : it_count->second) {
00150         if (class_it.classCode == classCode) {
00151           exist = true;
00152         }
00153       }
00154       // if exist uc in the count tree, but not exist the class, add the class
00155       // with 0 students
00156       if (!exist) {
00157         it_count->second.push_back({classCode, 0});
00158       }
00159       // if not found, add the uc and class with 0 students
00160     } else {
00161       std::vector<classQtd> classVec;
00162       classVec.push_back({classCode, 0});
00163       count.emplace(ucCode, classVec);
00164     }
00165   }
00166   file.close();
00167
00168   return ucClasses;
00169 }
```

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"
00013
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()

```
int main ( )
```

Definition at line 7 of file main.cpp.

```
00007            {
00008
00009   menu();
00010
00011   return 0;
00012 }
```

4.31.1.2 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
00033   std::cout « "------------ Welcome to our app :) ------------" « std::endl;
00034   std::cout « "| 1) See database                             |" « std::endl;
00035   std::cout « "| 2) Change database                          |" « std::endl;
00036   std::cout « "| 3) Backup                                   |" « std::endl;
00037   std::cout « "| 4) Exit                                     |" « std::endl;
00038   std::cout « "-----------------------------------------------" « std::endl;
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:
00049     menuRequests();
00050     break;
00051   case 3:
00052     menuBackup();
00053     break;
00054   case 4:
00055     exit(0);
00056   default:
00057     errorMessage();
00058     break;
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            {
00027
00028   menuUpdate();
00029   system("clear");
00030
00031   int flag = 0;
00032
00033   std::cout « "------------ Welcome to our app :) ------------" « std::endl;
00034   std::cout « "| 1) See database                             |" « std::endl;
00035   std::cout « "| 2) Change database                          |" « std::endl;
00036   std::cout « "| 3) Backup                                   |" « std::endl;
00037   std::cout « "| 4) Exit                                     |" « std::endl;
00038   std::cout « "-----------------------------------------------" « std::endl;
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:
00049     menuRequests();
00050     break;
00051   case 3:
00052     menuBackup();
00053     break;
00054   case 4:
00055     exit(0);
00056   default:
00057     errorMessage();
00058     break;
00059   }
00060 }
```

4.33.1.2 menuAdd()

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

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   if (it->second.valideQtClasses()) {
00300     std::cout « "-----------------------------------------------" « std::endl;
00301     std::cout « " You have already 7 classes" « std::endl;
00302   } else {
00303     std::cout « "-----------------------------------------------" « std::endl;
00304     std::cout « "Enter UC code to see all classes: " « std::endl;
00305     std::cin » ucCode;
00306
00307     if (!verifyUcCode(ucCode, it)) {
00308       // checks if ucCode exists
00309       auto it_uc = ucs.find(ucCode);
00310
00311       if (it_uc == ucs.end()) {
00312         std::cout « "-----------------------------------------------"
00313                   « std::endl;
00314         std::cout « "UC code not found" « std::endl;
00315         menuTryAgain(1, it);
00316
00317       } else {
00318         std::cout « "-----------------------------------------------"
00319                   « std::endl;
00320         std::cout « "Uc. Code: " « it_uc->first « std::endl;
00321
00322         printFreeClasses(ucCode, count);
00323         std::cout « "-----------------------------------------------"
00324                   « std::endl;
00325         std::cout « "Enter class code to add: " « std::endl;
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) {
```

```
00336             addClassStudent(ucCode, classCode, it, stackAlter);
00337             printStudentClasses(it);
00338             std::cout « "\nSucessfully added" « std::endl;
00339
00340             saveOrReturn();
00341           }
00342         } else {
00343           std::cout « "-----------------------------------------------"
00344                     « std::endl;
00345           std::cout « "Class code not found" « std::endl;
00346           menuTryAgain(1, it);
00347         }
00348       }
00349     } else {
00350       std::cout « "-----------------------------------------------"
00351                 « std::endl;
00352       std::cout « "You are already enrolled in this UC" « std::endl;
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;
00552   system("clear");
00553   listAllBackups();
00554
00555   bool valide = printAllBackups();
00556   if (valide == true) {
00557     printChanges(selectBackupCode(0));
00558     menuChanges();
00559   } else {
00560     std::cout « "-----------------------------------------------" « std::endl;
00561     std::cout « "| 1) - Main menu                              |" « std::endl;
00562     std::cout « "-----------------------------------------------" « std::endl;
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.

```
00590                   {
00591
00592   int flag;
00593
00594   std::cout « "-----------------------------------------------" « std::endl;
00595   std::cout « "| 1) Return                                    |" « std::endl;
```

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

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

```
00133                     {
00134   int flag = 0;
00135
00136   system("clear");
00137   std::cout « "Change database" « std::endl;
00138   std::cout « "-----------------------------------------" « std::endl;
00139   std::cout « "| 1) Add                                 |" « std::endl;
00140   std::cout « "| 2) Remove                              |" « std::endl;
00141   std::cout « "| 3) Switch                              |" « std::endl;
00142   std::cout « "-----------------------------------------" « std::endl;
00143   std::cout « "Choose an option: ";
00144   std::cin » flag;
00145
00146   if (flag > 4 || flag == 0) {
00147     errorMessage();
00148   } else {
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 « "-----------------------------------------------" « std::endl;
00073   std::cout « "| 1) See Students                             |" « std::endl;
00074   std::cout « "| 2) See Classes and UC's                     |" « std::endl;
00075   std::cout « "| 3) See My Schedules                         |" « std::endl;
00076   std::cout « "-----------------------------------------------" « std::endl;
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     case 1:
00091       menuStudents(code, type);
00092       break;
00093     case 2:
00094       menuUcs(code, type);
00095       break;
00096     default:
00097       errorMessage();
00098       break;
00099     }
00100   } else {
00101     int filter;
00102     int order;
00103     std::string value;
00104     if (type == 2) {
00105       filter = selectFilter();
00106       value = selectValue();
00107     }
00108     switch (flag) {
00109     case 1:
00110       order = selectOrderStudents();
00111       menuStudents(value, type, filter, order);
00112       break;
00113     case 2:
00114       order = selectOrderUcs();
00115       menuUcs(value, type, filter, order);
```

```
00116       break;
00117     case 3:
00118       menuStudentCode(4);
00119       break;
00120     default:
00121       errorMessage();
00122       break;
00123     }
00124   }
00125 }
```

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.

```
00161                                {
00162   std::string registrationNumber;
00163   std::cout « "-------------------------------------------------" « std::endl;
00164   std::cout « "Enter your registration number: ";
00165   std::cin » registrationNumber;
00166
00167   auto it = students.find(registrationNumber);
00168
00169   if (it == students.end()) {
00170     std::cout « "-------------------------------------------------" « std::endl;
00171     std::cout « "| Registration number not found                |" « std::endl;
00172     std::cout « "-------------------------------------------------" « std::endl;
00173
00174     std::cout « "| 1) Try again                                  |" « std::endl;
00175     std::cout « "| 2) Exit                                       |" « std::endl;
00176
00177     int flag2;
00178
00179     std::cin » flag2;
00180
00181     switch (flag2) {
00182     case 1:
00183       system("clear");
00184       menuStudentCode(flag);
00185       break;
00186     case 2:
00187       exit(0);
00188     default:
00189       errorMessage();
00190       break;
00191     }
00192
00193     menuRequests();
00194   } else {
00195     // printStudentClasses(it);
00196   }
00197
00198   switch (flag) {
00199   case (1):
00200     menuAdd(it);
00201     break;
00202   case (2):
00203     menuRemove(it);
00204     break;
00205   case (3):
00206     menuSwitch(it);
00207     break;
00208   case (4):
```

```
00209     printStudentSchedules(it, classes);
00210     break;
00211   default:
00212     errorMessage();
00213   }
00214 }
```

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.

```
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);
00770   }
00771
00772   if (type == 1) {
00773     oneStudent = selectStudent(str, oneStudent);
00774     printStudent(oneStudent);
00775   } else {
00776     if (type == 2) {
00777       data = filterInfoStudent(filter, str, data);
00778     }
00779     data = orderInfoStudent(order, data);
00780     printStudents(data);
00781   }
00782 }
```

4.33.1.10 menuSwitch()

```
void menuSwitch (
            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.

```
00367                                                 {
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
00376   std::cout « "-----------------------------------------------" « std::endl;
00377   std::cout « "| 1) Switch UC                                 |" « std::endl;
00378   std::cout « "| 2) Switch Class                              |" « std::endl;
00379   std::cout « "-----------------------------------------------" « std::endl;
00380   std::cin » flag;
00381
00382   switch (flag) {
00383   case (1):
00384     std::cout « "-----------------------------------------------" « std::endl;
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
00401         std::cout « "-----------------------------------------------"
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) {
00409           validate = valideNewClass(ucCode, classCode, it, classes);
00410           if (!validate) {
00411             removeUcStudent(ucCode, it, stackAlter, count);
00412             addClassStudent(ucCode, classCode, it, stackAlter);
00413             printStudentClasses(it);
00414             std::cout « "\nSuccessfully switched" « std::endl;
00415             saveOrReturn();
00416           }
00417         } else {
00418           std::cout « "-----------------------------------------------"
00419                     « std::endl;
00420           std::cout « "Class code not found" « std::endl;
00421           menuTryAgain(3, it);
00422         }
00423       } else {
00424         std::cout « "-----------------------------------------------"
00425                   « std::endl;
00426         std::cout « "UC code not found" « std::endl;
00427         menuTryAgain(3, it);
00428       }
00429     } else {
00430       std::cout « "-----------------------------------------------"
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 « "-----------------------------------------------" « std::endl;
00439     std::cout « "Enter UC to change class: " « std::endl;
00440     std::cin » ucCode;
00441
00442     if (verifyUcCode(ucCode, it)) {
00443
00444       printFreeClasses(ucCode, count);
00445       std::cout « "-----------------------------------------------"
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
00452       if (check_class) {
```

```
00453         removeUcStudent(ucCode, it, stackAlter, count);
00454         validate = valideNewClass(ucCode, classCode, it, classes);
00455         if (!validate) {
00456           addClassStudent(ucCode, classCode, it, stackAlter);
00457           printStudentClasses(it);
00458           std::cout « "\nSuccessfully switched" « std::endl;
00459           saveOrReturn();
00460         }
00461       } else {
00462         std::cout « "-----------------------------------------------"
00463                   « std::endl;
00464         std::cout « "Class code not found" « std::endl;
00465         menuTryAgain(3, it);
00466       }
00467     } else {
00468       std::cout « "-----------------------------------------------"
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 }
```

4.33.1.11 menuTryAgain()

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

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

menuType	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                                                          {
00227   int flag;
00228   std::cout « "-----------------------------------------------" « std::endl;
00229   std::cout « "| 1) Try again                                  |" « std::endl;
00230   std::cout « "| 2) Exit                                       |" « std::endl;
00231   std::cout « "-----------------------------------------------" « std::endl;
00232   std::cin » flag;
00233
00234   switch (flag) {
00235   case 1:
00236     system("clear");
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   }
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) {
00801     for (const myUc &ucObj : ucVectorPair.second) {
00802       data.push_back(ucObj);
00803     }
00804   }
00805
00806   if (type == 1) {
00807     oneUc = selectUc(str, classes);
00808     printUcClasses(oneUc);
00809   } else {
00810     if (type == 2) {
00811       data = filterInfoUc(filter, str, data);
00812     }
00813     data = orderInfoUc(order, data);
00814     printUcs(data);
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.

```
00579                       {
00580   backupFile(selectBackupCode(1));
00581   menu();
00582 }
```

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.

```
00487                   {
00488   int flag = 0;
00489
00490   std::cout « "-------------------------------------------------" « std::endl;
00491   std::cout « "| 1) Save                                        |" « std::endl;
00492   std::cout « "| 2) Return                                      |" « std::endl;
00493   std::cout « "-------------------------------------------------" « std::endl;
00494   std::cout « "Choose an option: ";
00495   std::cin » flag;
00496
00497   errorCheck(flag);
00498
00499   switch (flag) {
00500   case 1:
00501     save();
00502     break;
00503   case 2:
00504     menuRequests();
00505     break;
00506   default:
00507     errorMessage();
00508     break;
00509   }
00510 }
```

4.33.1.17 selectBackupCode()

```
int selectBackupCode (
            int type )
```

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: ";
00535   } else if (type == 1) {
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                          {
00701   std::string str;
00702   std::cout « "-----------------------------------------------" « std::endl;
00703   std::cout « "| 1) Search by code                           |" « std::endl;
00704   std::cout « "-----------------------------------------------" « std::endl;
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
00724   std::cout « "-----------------------------------------------" « std::endl;
00725   std::cout « "| 1) Uc Code                                  |" « std::endl;
00726   std::cout « "| 2) Class Code                               |" « std::endl;
00727   std::cout « "-----------------------------------------------" « std::endl;
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;
00630   std::cout « "| 1) Sort by student code asc                 |" « std::endl;
00631   std::cout « "| 2) Sort by student code desc                |" « std::endl;
00632   std::cout « "| 3) Sort by student name asc                 |" « std::endl;
00633   std::cout « "| 4) Sort by student name desc                |" « std::endl;
00634   std::cout « "-----------------------------------------------" « std::endl;
00635   // 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;
00656   std::cout « "| 1) Sort by uc code asc                      |" « std::endl;
00657   std::cout « "| 2) Sort by uc code desc                     |" « std::endl;
00658   std::cout « "| 3) Sort by class code asc                   |" « std::endl;
00659   std::cout « "| 4) Sort by class code desc                  |" « std::endl;
00660   std::cout « "-----------------------------------------------" « std::endl;
00661   // add more order like - n° ucs,
00662   std::cout « "Choose an option: ";
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
00680   std::cout « "-----------------------------------------------" « std::endl;
00681   std::cout « "| 1) See one                                   |" « std::endl;
00682   std::cout « "| 2) See a particular group                   |" « std::endl;
00683   std::cout « "| 3) See all                                  |" « std::endl;
00684   std::cout « "-----------------------------------------------" « 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.

```
00743                           {
00744   std::string str;
00745
00746   std::cout « "Enter the value: ";
00747   std::cin » str;
00748   // errorcheck (str)
00749
00750   return str;
00751 }
```

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.

4.33.2.2 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);
00007 std::map<std::string, myUc> classes = readSchedules();
00008
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
00033   std::cout « "------------ Welcome to our app :) ------------" « std::endl;
00034   std::cout « "| 1) See database                             |" « std::endl;
00035   std::cout « "| 2) Change database                          |" « std::endl;
00036   std::cout « "| 3) Backup                                   |" « std::endl;
00037   std::cout « "| 4) Exit                                     |" « std::endl;
00038   std::cout « "-----------------------------------------------" « std::endl;
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:
```

```
00049     menuRequests();
00050     break;
00051   case 3:
00052     menuBackup();
00053     break;
00054   case 4:
00055     exit(0);
00056   default:
00057     errorMessage();
00058     break;
00059   }
00060 }
00061
00068 void menuSeeDatabase() {
00069   int flag = 0;
00070   int type;
00071
00072   std::cout « "-----------------------------------------------" « std::endl;
00073   std::cout « "| 1) See Students                              |" « std::endl;
00074   std::cout « "| 2) See Classes and UC's                      |" « std::endl;
00075   std::cout « "| 3) See My Schedules                          |" « std::endl;
00076   std::cout « "-----------------------------------------------" « std::endl;
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     case 1:
00091       menuStudents(code, type);
00092       break;
00093     case 2:
00094       menuUcs(code, type);
00095       break;
00096     default:
00097       errorMessage();
00098       break;
00099     }
00100   } else {
00101     int filter;
00102     int order;
00103     std::string value;
00104     if (type == 2) {
00105       filter = selectFilter();
00106       value = selectValue();
00107     }
00108     switch (flag) {
00109     case 1:
00110       order = selectOrderStudents();
00111       menuStudents(value, type, filter, order);
00112       break;
00113     case 2:
00114       order = selectOrderUcs();
00115       menuUcs(value, type, filter, order);
00116       break;
00117     case 3:
00118       menuStudentCode(4);
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;
00138   std::cout « "-----------------------------------------------" « std::endl;
00139   std::cout « "| 1) Add                                       |" « std::endl;
00140   std::cout « "| 2) Remove                                    |" « std::endl;
00141   std::cout « "| 3) Switch                                    |" « std::endl;
00142   std::cout « "-----------------------------------------------" « std::endl;
00143   std::cout « "Choose an option: ";
00144   std::cin » flag;
00145
00146   if (flag > 4 || flag == 0) {
00147     errorMessage();
```

```
00148   } else {
00149     menuStudentCode(flag);
00150   }
00151 }
00152
00161 void menuStudentCode(int flag) {
00162   std::string registrationNumber;
00163   std::cout « "-----------------------------------------------" « std::endl;
00164   std::cout « "Enter your registration number: ";
00165   std::cin » registrationNumber;
00166
00167   auto it = students.find(registrationNumber);
00168
00169   if (it == students.end()) {
00170     std::cout « "-----------------------------------------------" « std::endl;
00171     std::cout « "| Registration number not found               |" « std::endl;
00172     std::cout « "-----------------------------------------------" « std::endl;
00173
00174     std::cout « "| 1) Try again                                |" « std::endl;
00175     std::cout « "| 2) Exit                                     |" « std::endl;
00176
00177     int flag2;
00178
00179     std::cin » flag2;
00180
00181     switch (flag2) {
00182     case 1:
00183       system("clear");
00184       menuStudentCode(flag);
00185       break;
00186     case 2:
00187       exit(0);
00188     default:
00189       errorMessage();
00190       break;
00191     }
00192
00193     menuRequests();
00194   } else {
00195     // printStudentClasses(it);
00196   }
00197
00198   switch (flag) {
00199   case (1):
00200     menuAdd(it);
00201     break;
00202   case (2):
00203     menuRemove(it);
00204     break;
00205   case (3):
00206     menuSwitch(it);
00207     break;
00208   case (4):
00209     printStudentSchedules(it, classes);
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;
00229   std::cout « "| 1) Try again                                |" « std::endl;
00230   std::cout « "| 2) Exit                                     |" « std::endl;
00231   std::cout « "-----------------------------------------------" « std::endl;
00232   std::cin » flag;
00233
00234   switch (flag) {
00235   case 1:
00236     system("clear");
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   }
00251 }
```

```
00252
00262 void menuRemove(std::map<std::string, myStudent>::iterator &it) {
00263   printStudentClasses(it);
00264   std::string ucCode;
00265
00266   std::cout « "-----------------------------------------------" « std::endl;
00267   std::cout « "Enter UC code to remove " « std::endl;
00268   std::cin » ucCode;
00269   std::cout « "-----------------------------------------------" « std::endl;
00270
00271   bool remove = removeUcStudent(ucCode, it, stackAlter, count);
00272
00273   if (remove) {
00274     printStudentClasses(it);
00275     std::cout « "\nRemovido com sucesso" « std::endl;
00276     saveOrReturn();
00277   } else {
00278     std::cout « "-----------------------------------------------" « std::endl;
00279     std::cout « "UC code not found" « std::endl;
00280     menuTryAgain(2, it);
00281   }
00282 }
00283
00292 void menuAdd(std::map<std::string, myStudent>::iterator &it) {
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   if (it->second.valideQtClasses()) {
00300     std::cout « "-----------------------------------------------" « std::endl;
00301     std::cout « " You have already 7 classes" « std::endl;
00302   } else {
00303     std::cout « "-----------------------------------------------" « std::endl;
00304     std::cout « "Enter UC code to see all classes: " « std::endl;
00305     std::cin » ucCode;
00306
00307     if (!verifyUcCode(ucCode, it)) {
00308       // checks if ucCode exists
00309       auto it_uc = ucs.find(ucCode);
00310
00311       if (it_uc == ucs.end()) {
00312         std::cout « "-----------------------------------------------"
00313                   « std::endl;
00314         std::cout « "UC code not found" « std::endl;
00315         menuTryAgain(1, it);
00316
00317       } else {
00318         std::cout « "-----------------------------------------------"
00319                   « std::endl;
00320         std::cout « "Uc. Code: " « it_uc->first « std::endl;
00321
00322         printFreeClasses(ucCode, count);
00323         std::cout « "-----------------------------------------------"
00324                   « std::endl;
00325         std::cout « "Enter class code to add: " « std::endl;
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) {
00336             addClassStudent(ucCode, classCode, it, stackAlter);
00337             printStudentClasses(it);
00338             std::cout « "\nSucessfully added" « std::endl;
00339
00340             saveOrReturn();
00341           }
00342         } else {
00343           std::cout « "-----------------------------------------------"
00344                     « std::endl;
00345           std::cout « "Class code not found" « std::endl;
00346           menuTryAgain(1, it);
00347         }
00348       }
00349     } else {
00350       std::cout « "-----------------------------------------------"
00351                 « std::endl;
00352       std::cout « "You are already enrolled in this UC" « std::endl;
00353       menuTryAgain(1, it);
00354     }
00355   }
```

```
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
00376   std::cout « "-----------------------------------------------" « std::endl;
00377   std::cout « "| 1) Switch UC                                 |" « std::endl;
00378   std::cout « "| 2) Switch Class                              |" « std::endl;
00379   std::cout « "-----------------------------------------------" « std::endl;
00380   std::cin » flag;
00381
00382   switch (flag) {
00383   case (1):
00384     std::cout « "-----------------------------------------------" « std::endl;
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
00401         std::cout « "-----------------------------------------------"
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) {
00409           validate = valideNewClass(ucCode, classCode, it, classes);
00410           if (!validate) {
00411             removeUcStudent(ucCode, it, stackAlter, count);
00412             addClassStudent(ucCode, classCode, it, stackAlter);
00413             printStudentClasses(it);
00414             std::cout « "\nSuccessfully switched" « std::endl;
00415             saveOrReturn();
00416           }
00417         } else {
00418           std::cout « "-----------------------------------------------"
00419                     « std::endl;
00420           std::cout « "Class code not found" « std::endl;
00421           menuTryAgain(3, it);
00422         }
00423       } else {
00424         std::cout « "-----------------------------------------------"
00425                   « std::endl;
00426         std::cout « "UC code not found" « std::endl;
00427         menuTryAgain(3, it);
00428       }
00429     } else {
00430       std::cout « "-----------------------------------------------"
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 « "-----------------------------------------------" « std::endl;
00439     std::cout « "Enter UC to change class: " « std::endl;
00440     std::cin » ucCode;
00441
00442     if (verifyUcCode(ucCode, it)) {
00443
00444       printFreeClasses(ucCode, count);
00445       std::cout « "-----------------------------------------------"
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
```

```
00452       if (check_class) {
00453         removeUcStudent(ucCode, it, stackAlter, count);
00454         validate = valideNewClass(ucCode, classCode, it, classes);
00455         if (!validate) {
00456           addClassStudent(ucCode, classCode, it, stackAlter);
00457           printStudentClasses(it);
00458           std::cout « "\nSuccessfully switched" « std::endl;
00459           saveOrReturn();
00460         }
00461       } else {
00462         std::cout « "-----------------------------------------------"
00463                   « std::endl;
00464         std::cout « "Class code not found" « std::endl;
00465         menuTryAgain(3, it);
00466       }
00467     } else {
00468       std::cout « "-----------------------------------------------"
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 }
00479
00487 void saveOrReturn() {
00488   int flag = 0;
00489
00490   std::cout « "-----------------------------------------------" « std::endl;
00491   std::cout « "| 1) Save                                      |" « std::endl;
00492   std::cout « "| 2) Return                                    |" « std::endl;
00493   std::cout « "-----------------------------------------------" « std::endl;
00494   std::cout « "Choose an option: ";
00495   std::cin » flag;
00496
00497   errorCheck(flag);
00498
00499   switch (flag) {
00500   case 1:
00501     save();
00502     break;
00503   case 2:
00504     menuRequests();
00505     break;
00506   default:
00507     errorMessage();
00508     break;
00509   }
00510 }
00511
00518 void save() {
00519   keepAllChanges(students, stackAlter);
00520   exit(0);
00521 }
00522
00530 int selectBackupCode(int type) {
00531   int cdBkp;
00532
00533   if (type == 0) {
00534     std::cout « "Choose a backup to view changes: ";
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();
00556   if (valide == true) {
00557     printChanges(selectBackupCode(0));
00558     menuChanges();
00559   } else {
00560     std::cout « "-----------------------------------------------" « std::endl;
00561     std::cout « "| 1) - Main menu                              |" « std::endl;
00562     std::cout « "-----------------------------------------------" « std::endl;
00563     std::cin » flag;
00564
```

```
00565     if (flag == 1) {
00566       menu();
00567     } else {
00568       errorMessage();
00569     }
00570   }
00571 }
00572
00579 void restoreBackup() {
00580   backupFile(selectBackupCode(1));
00581   menu();
00582 }
00583
00590 void menuChanges() {
00591
00592   int flag;
00593
00594   std::cout « "-------------------------------------------------" « std::endl;
00595   std::cout « "| 1) Return                                      |" « std::endl;
00596   std::cout « "| 2) Main menu                                   |" « std::endl;
00597   std::cout « "| 3) Restore                                     |" « std::endl;
00598   std::cout « "-------------------------------------------------" « std::endl;
00599
00600   std::cin » flag;
00601
00602   switch (flag) {
00603   case (1):
00604     menuBackup();
00605     break;
00606   case (2):
00607     menu();
00608     break;
00609   case (3):
00610     restoreBackup();
00611     break;
00612   default:
00613     errorMessage();
00614     break;
00615   }
00616 }
00617
00626 int selectOrderStudents() {
00627   int flag = 0;
00628
00629   std::cout « "-------------------------------------------------" « std::endl;
00630   std::cout « "| 1) Sort by student code asc                    |" « std::endl;
00631   std::cout « "| 2) Sort by student code desc                   |" « std::endl;
00632   std::cout « "| 3) Sort by student name asc                    |" « std::endl;
00633   std::cout « "| 4) Sort by student name desc                   |" « std::endl;
00634   std::cout « "-------------------------------------------------" « std::endl;
00635   // 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 }
00643
00652 int selectOrderUcs() {
00653   int flag = 0;
00654
00655   std::cout « "-------------------------------------------------" « std::endl;
00656   std::cout « "| 1) Sort by uc code asc                         |" « std::endl;
00657   std::cout « "| 2) Sort by uc code desc                        |" « std::endl;
00658   std::cout « "| 3) Sort by class code asc                      |" « std::endl;
00659   std::cout « "| 4) Sort by class code desc                     |" « std::endl;
00660   std::cout « "-------------------------------------------------" « std::endl;
00661   // add more order like - n° ucs,
00662   std::cout « "Choose an option: ";
00663   std::cin » flag;
00664
00665   errorCheck(flag);
00666
00667   return flag;
00668 }
00669
00677 int selectType() {
00678   int flag = 0;
00679
00680   std::cout « "-------------------------------------------------" « std::endl;
00681   std::cout « "| 1) See one                                     |" « std::endl;
00682   std::cout « "| 2) See a particular group                      |" « std::endl;
00683   std::cout « "| 3) See all                                     |" « std::endl;
00684   std::cout « "-------------------------------------------------" « std::endl;
00685
00686   std::cout « "Choose an option: ";
```

```
00687   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;
00703   std::cout « "| 1) Search by code                           |" « std::endl;
00704   std::cout « "-----------------------------------------------" « std::endl;
00705   std::cout « "Enter the code: ";
00706   std::cin » str;
00707   // errorcheck (str)
00708
00709   return str;
00710 }
00711
00721 int selectFilter() {
00722   int flag = 0;
00723
00724   std::cout « "-----------------------------------------------" « std::endl;
00725   std::cout « "| 1) Uc Code                                  |" « std::endl;
00726   std::cout « "| 2) Class Code                               |" « std::endl;
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) {
00773     oneStudent = selectStudent(str, oneStudent);
00774     printStudent(oneStudent);
00775   } else {
00776     if (type == 2) {
00777       data = filterInfoStudent(filter, str, data);
00778     }
00779     data = orderInfoStudent(order, data);
00780     printStudents(data);
00781   }
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) {
00801     for (const myUc &ucObj : ucVectorPair.second) {
00802       data.push_back(ucObj);
00803     }
00804   }
00805
00806   if (type == 1) {
00807     oneUc = selectUc(str, classes);
00808     printUcClasses(oneUc);
00809   } else {
00810     if (type == 2) {
00811       data = filterInfoUc(filter, str, data);
00812     }
00813     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 (
            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 }
```

4.35.1.2 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
00033   std::cout « "------------ Welcome to our app :) ------------" « std::endl;
00034   std::cout « "| 1) See database                              |" « std::endl;
00035   std::cout « "| 2) Change database                          |" « std::endl;
00036   std::cout « "| 3) Backup                                    |" « std::endl;
```

```
00037   std::cout « "| 4) Exit                                          |" « std::endl;
00038   std::cout « "-------------------------------------------------" « std::endl;
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:
00049     menuRequests();
00050     break;
00051   case 3:
00052     menuBackup();
00053     break;
00054   case 4:
00055     exit(0);
00056   default:
00057     errorMessage();
00058     break;
00059   }
00060 }
```

4.35.1.4 menuAdd()

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

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   if (it->second.valideQtClasses()) {
00300     std::cout « "-------------------------------------------------" « std::endl;
00301     std::cout « " You have already 7 classes" « std::endl;
00302   } else {
00303     std::cout « "-------------------------------------------------" « std::endl;
00304     std::cout « "Enter UC code to see all classes: " « std::endl;
00305     std::cin » ucCode;
00306
00307     if (!verifyUcCode(ucCode, it)) {
00308       // checks if ucCode exists
00309       auto it_uc = ucs.find(ucCode);
00310
00311       if (it_uc == ucs.end()) {
00312         std::cout « "-------------------------------------------------"
00313                   « std::endl;
00314         std::cout « "UC code not found" « std::endl;
00315         menuTryAgain(1, it);
00316
00317       } else {
00318         std::cout « "-------------------------------------------------"
00319                   « std::endl;
00320         std::cout « "Uc. Code: " « it_uc->first « std::endl;
00321
00322         printFreeClasses(ucCode, count);
00323         std::cout « "-------------------------------------------------"
00324                   « std::endl;
00325         std::cout « "Enter class code to add: " « std::endl;
```

```
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) {
00336             addClassStudent(ucCode, classCode, it, stackAlter);
00337             printStudentClasses(it);
00338             std::cout « "\nSucessfully added" « std::endl;
00339
00340             saveOrReturn();
00341           }
00342         } else {
00343           std::cout « "-----------------------------------------------"
00344                     « std::endl;
00345           std::cout « "Class code not found" « std::endl;
00346           menuTryAgain(1, it);
00347         }
00348       }
00349     } else {
00350       std::cout « "-----------------------------------------------"
00351                 « std::endl;
00352       std::cout « "You are already enrolled in this UC" « std::endl;
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;
00552   system("clear");
00553   listAllBackups();
00554
00555   bool valide = printAllBackups();
00556   if (valide == true) {
00557     printChanges(selectBackupCode(0));
00558     menuChanges();
00559   } else {
00560     std::cout « "-----------------------------------------------" « std::endl;
00561     std::cout « "| 1) - Main menu                              |" « std::endl;
00562     std::cout « "-----------------------------------------------" « std::endl;
00563     std::cin » flag;
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.

```
00590                     {
00591
00592   int flag;
00593
00594   std::cout « "-----------------------------------------------" « std::endl;
00595   std::cout « "| 1) Return                                     |" « std::endl;
00596   std::cout « "| 2) Main menu                                 |" « std::endl;
00597   std::cout « "| 3) Restore                                   |" « std::endl;
00598   std::cout « "-----------------------------------------------" « std::endl;
00599
00600   std::cin » flag;
00601
00602   switch (flag) {
00603   case (1):
00604     menuBackup();
00605     break;
00606   case (2):
00607     menu();
00608     break;
00609   case (3):
00610     restoreBackup();
00611     break;
00612   default:
00613     errorMessage();
00614     break;
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                                                   {
00263   printStudentClasses(it);
00264   std::string ucCode;
00265
00266   std::cout « "-----------------------------------------------" « std::endl;
00267   std::cout « "Enter UC code to remove " « std::endl;
00268   std::cin » ucCode;
00269   std::cout « "-----------------------------------------------" « std::endl;
00270
00271   bool remove = removeUcStudent(ucCode, it, stackAlter, count);
00272
00273   if (remove) {
00274     printStudentClasses(it);
00275     std::cout « "\nRemovido com sucesso" « std::endl;
00276     saveOrReturn();
00277   } else {
00278     std::cout « "-----------------------------------------------" « std::endl;
00279     std::cout « "UC code not found" « std::endl;
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");
00137   std::cout « "Change database" « std::endl;
00138   std::cout « "-------------------------------------------------" « std::endl;
00139   std::cout « "| 1) Add                                         |" « std::endl;
00140   std::cout « "| 2) Remove                                      |" « std::endl;
00141   std::cout « "| 3) Switch                                      |" « std::endl;
00142   std::cout « "-------------------------------------------------" « std::endl;
00143   std::cout « "Choose an option: ";
00144   std::cin » flag;
00145
00146   if (flag > 4 || flag == 0) {
00147     errorMessage();
00148   } else {
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::endl;
00073   std::cout « "| 1) See Students                               |" « std::endl;
00074   std::cout « "| 2) See Classes and UC's                       |" « std::endl;
00075   std::cout « "| 3) See My Schedules                           |" « std::endl;
00076   std::cout « "-------------------------------------------------" « std::endl;
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     case 1:
00091       menuStudents(code, type);
00092       break;
00093     case 2:
00094       menuUcs(code, type);
00095       break;
00096     default:
00097       errorMessage();
00098       break;
00099     }
00100   } else {
00101     int filter;
00102     int order;
00103     std::string value;
00104     if (type == 2) {
00105       filter = selectFilter();
00106       value = selectValue();
00107     }
00108     switch (flag) {
```

```
00109     case 1:
00110       order = selectOrderStudents();
00111       menuStudents(value, type, filter, order);
00112       break;
00113     case 2:
00114       order = selectOrderUcs();
00115       menuUcs(value, type, filter, order);
00116       break;
00117     case 3:
00118       menuStudentCode(4);
00119       break;
00120     default:
00121       errorMessage();
00122       break;
00123     }
00124   }
00125 }
```

4.35.1.10 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.

```
00161                                 {
00162   std::string registrationNumber;
00163   std::cout « "-------------------------------------------------" « std::endl;
00164   std::cout « "Enter your registration number: ";
00165   std::cin » registrationNumber;
00166
00167   auto it = students.find(registrationNumber);
00168
00169   if (it == students.end()) {
00170     std::cout « "-------------------------------------------------" « std::endl;
00171     std::cout « "| Registration number not found                 |" « std::endl;
00172     std::cout « "-------------------------------------------------" « std::endl;
00173
00174     std::cout « "| 1) Try again                                  |" « std::endl;
00175     std::cout « "| 2) Exit                                       |" « std::endl;
00176
00177     int flag2;
00178
00179     std::cin » flag2;
00180
00181     switch (flag2) {
00182     case 1:
00183       system("clear");
00184       menuStudentCode(flag);
00185       break;
00186     case 2:
00187       exit(0);
00188     default:
00189       errorMessage();
00190       break;
00191     }
00192
00193     menuRequests();
00194   } else {
00195     // printStudentClasses(it);
00196   }
00197
00198   switch (flag) {
00199   case (1):
00200     menuAdd(it);
00201     break;
```

```
00202   case (2):
00203     menuRemove(it);
00204     break;
00205   case (3):
00206     menuSwitch(it);
00207     break;
00208   case (4):
00209     printStudentSchedules(it, classes);
00210     break;
00211   default:
00212     errorMessage();
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);
00770   }
00771
00772   if (type == 1) {
00773     oneStudent = selectStudent(str, oneStudent);
00774     printStudent(oneStudent);
00775   } else {
00776     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()

```
void menuSwitch (
              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.

```
00367                                                    {
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
00376   std::cout « "-----------------------------------------------" « std::endl;
00377   std::cout « "| 1) Switch UC                                 |" « std::endl;
00378   std::cout « "| 2) Switch Class                              |" « std::endl;
00379   std::cout « "-----------------------------------------------" « std::endl;
00380   std::cin » flag;
00381
00382   switch (flag) {
00383   case (1):
00384     std::cout « "-----------------------------------------------" « std::endl;
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
00401         std::cout « "-----------------------------------------------"
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) {
00409           validate = valideNewClass(ucCode, classCode, it, classes);
00410           if (!validate) {
00411             removeUcStudent(ucCode, it, stackAlter, count);
00412             addClassStudent(ucCode, classCode, it, stackAlter);
00413             printStudentClasses(it);
00414             std::cout « "\nSuccessfully switched" « std::endl;
00415             saveOrReturn();
00416           }
00417         } else {
00418           std::cout « "-----------------------------------------------"
00419                     « std::endl;
00420           std::cout « "Class code not found" « std::endl;
00421           menuTryAgain(3, it);
00422         }
00423       } else {
00424         std::cout « "-----------------------------------------------"
00425                   « std::endl;
00426         std::cout « "UC code not found" « std::endl;
00427         menuTryAgain(3, it);
00428       }
00429     } else {
00430       std::cout « "-----------------------------------------------"
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 « "-----------------------------------------------" « std::endl;
00439     std::cout « "Enter UC to change class: " « std::endl;
00440     std::cin » ucCode;
00441
00442     if (verifyUcCode(ucCode, it)) {
00443
00444       printFreeClasses(ucCode, count);
```

```
00445       std::cout « "-----------------------------------------------"
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
00452       if (check_class) {
00453         removeUcStudent(ucCode, it, stackAlter, count);
00454         validate = valideNewClass(ucCode, classCode, it, classes);
00455         if (!validate) {
00456           addClassStudent(ucCode, classCode, it, stackAlter);
00457           printStudentClasses(it);
00458           std::cout « "\nSuccessfully switched" « std::endl;
00459           saveOrReturn();
00460         }
00461       } else {
00462         std::cout « "-----------------------------------------------"
00463                   « std::endl;
00464         std::cout « "Class code not found" « std::endl;
00465         menuTryAgain(3, it);
00466       }
00467     } else {
00468       std::cout « "-----------------------------------------------"
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()

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

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

menuType	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                                                          {
00227   int flag;
00228   std::cout « "-----------------------------------------------" « std::endl;
00229   std::cout « "| 1) Try again                                  |" « std::endl;
00230   std::cout « "| 2) Exit                                       |" « std::endl;
00231   std::cout « "-----------------------------------------------" « std::endl;
00232   std::cin » flag;
00233
00234   switch (flag) {
00235   case 1:
00236     system("clear");
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   }
00251 }
```

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) {
00801     for (const myUc &ucObj : ucVectorPair.second) {
00802       data.push_back(ucObj);
00803     }
00804   }
00805
00806   if (type == 1) {
00807     oneUc = selectUc(str, classes);
00808     printUcClasses(oneUc);
00809   } else {
00810     if (type == 2) {
00811       data = filterInfoUc(filter, str, data);
00812     }
00813     data = orderInfoUc(order, data);
00814     printUcs(data);
00815   }
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.

```
00579                       {
00580   backupFile(selectBackupCode(1));
00581   menu();
00582 }
```

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.

```
00487                   {
00488   int flag = 0;
00489
00490   std::cout « "-------------------------------------------------" « std::endl;
00491   std::cout « "| 1) Save                                        |" « std::endl;
00492   std::cout « "| 2) Return                                      |" « std::endl;
00493   std::cout « "-------------------------------------------------" « std::endl;
00494   std::cout « "Choose an option: ";
00495   std::cin » flag;
00496
00497   errorCheck(flag);
00498
00499   switch (flag) {
00500   case 1:
00501     save();
00502     break;
00503   case 2:
00504     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::endl;
00703   std::cout « "| 1) Search by code                           |" « std::endl;
00704   std::cout « "-----------------------------------------------" « std::endl;
00705   std::cout « "Enter the code: ";
00706   std::cin » str;
00707   // 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   int flag = 0;
00723
00724   std::cout « "-----------------------------------------------" « std::endl;
00725   std::cout « "| 1) Uc Code                                  |" « std::endl;
00726   std::cout « "| 2) Class Code                               |" « std::endl;
00727   std::cout « "-----------------------------------------------" « std::endl;
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;
00630   std::cout « "| 1) Sort by student code asc                 |" « std::endl;
00631   std::cout « "| 2) Sort by student code desc                |" « std::endl;
00632   std::cout « "| 3) Sort by student name asc                 |" « std::endl;
00633   std::cout « "| 4) Sort by student name desc                |" « std::endl;
00634   std::cout « "-----------------------------------------------" « std::endl;
00635   // 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;
00656   std::cout « "| 1) Sort by uc code asc                      |" « std::endl;
00657   std::cout « "| 2) Sort by uc code desc                     |" « std::endl;
00658   std::cout « "| 3) Sort by class code asc                   |" « std::endl;
00659   std::cout « "| 4) Sort by class code desc                  |" « std::endl;
00660   std::cout « "-----------------------------------------------" « std::endl;
00661   // add more order like - n° ucs,
00662   std::cout « "Choose an option: ";
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;
00681   std::cout « "| 1) See one                                     |" « std::endl;
00682   std::cout « "| 2) See a particular group                     |" « std::endl;
00683   std::cout « "| 3) See all                                    |" « std::endl;
00684   std::cout « "-------------------------------------------------" « std::endl;
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
00746   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>
00008
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);
00023
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
```