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.

```
                                         {
        return startTime < other.startTime;
    }
```

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.

```
                                                    {
    return classCode < other.classCode;
  }
```

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 myStudient() `[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.

```
                                                    {
  std::vector<std::string> classCodes;
  for (const auto &uc : classes) {
    std::vector<classInfo> classInfoVec = uc.getClassInfoVec();
    for (const auto &classInfo : classInfoVec) {
      classCodes.push_back(classInfo.code);
    }
  }
  return classCodes;
}
```

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.

```
{ 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.

```
{ 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.

```
{ 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.

```
                                               {
  std::vector<std::string> ucCodes;
  for (const auto &uc : classes) {
    ucCodes.push_back(uc.getUcCode());
  }
  return ucCodes;
}
```

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.

```
                                                             {
  studentCode = sCode;
  studentName = sName;
}
```

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.

```
                              {
  if (classes.size() > 7) {
    return true;
  }
  return false;
}
```

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.

```
{ 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.

```
{ 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.

```
                                             {
  ucCode = ucC;
  classCode = classC;
}
```

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.

```
#include "student.h"

myStudent::myStudent(const std::string &sCode, const std::string &sName) {
  studentCode = sCode;
  studentName = sName;
}
myStudent::myStudent() {
  studentCode = "";
  studentName = "";
}

void myStudent::setStudent(std::string &sCode, std::string &sName) {
  studentCode = sCode;
  studentName = sName;
}

void myStudent::setStudentCode(std::string &n) { studentCode = n; }

void myStudent::setStudentName(std::string &n) { studentName = n; }

std::string myStudent::getStudentCode() const { return studentCode; }

std::string myStudent::getStudentName() const { return studentName; }

std::vector<std::string> myStudent::getUcCode() const {
  std::vector<std::string> ucCodes;
  for (const auto &uc : classes) {
    ucCodes.push_back(uc.getUcCode());
  }
  return ucCodes;
}

std::vector<std::string> myStudent::getClassCode() const {
  std::vector<std::string> classCodes;
  for (const auto &uc : classes) {
    std::vector<classInfo> classInfoVec = uc.getClassInfoVec();
    for (const auto &classInfo : classInfoVec) {
      classCodes.push_back(classInfo.code);
    }
  }
  return classCodes;
}

```

```
std::vector<myUc> &myStudent::getClasses() const { return classes; }

void myStudent::addClass(const myUc &myClass) { classes.push_back(myClass); }

bool myStudent::valideQtClasses() {
  if (classes.size() > 7) {
    return true;
  }
  return false;
}
```

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.

```
#ifndef MYSTUDENT_H
#define MYSTUDENT_H

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

#include "uc.h"

struct alter {
  std::string studentCode;
  std::string studentName;
  std::string type;
  std::string ucCode;
  std::string classCode;
};

class myStudent {
private:
  std::string studentCode;
  std::string studentName;
  mutable std::vector<myUc> classes;

public:
  // Constructor functions
  myStudent(const std::string &sCode, const std::string &sName);
  myStudent();

  // Setter functions
  void setStudent(std::string &sCode, std::string &sName);
  void setStudentCode(std::string &n);
  void setStudentName(std::string &n);

  // Getters functions
  std::string getStudentCode() const;
  std::string getStudentName() const;
  std::vector<std::string> getUcCode() const;
  std::vector<std::string> getClassCode() const;
  std::vector<myUc> &getClasses() const;

  // Others functions
  void addClass(const myUc &myClass);
  void changeClass(const myUc &myClass);
  void addUc(const myUc &myClass);
  void removeUc(const myUc &myClass);
  bool valideQtClasses();
};

#endif
```

4.5 src/classes/uc.cpp File Reference

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

4.6 uc.cpp

Go to the documentation of this file.

```
#include "uc.h"

myUc::myUc(const std::string &ucC, std::string &classC) {
  ucCode = ucC;
  classCode = classC;
}

myUc::myUc() {
  ucCode = "";
  classCode = {};
}

void myUc::SetUc(std::string &ucC, std::string &classC) {
  ucCode = ucC;
  classCode = classC;
}

void myUc::setUcCode(std::string &n) { ucCode = n; }

void myUc::setClassCode(std::string &n) { classCode = n; }

std::string myUc::getUcCode() const { return ucCode; }

std::string myUc::getClassCode() const { return classCode; }

std::vector<classInfo> myUc::getClassInfoVec() const { return classInfoVec; }

void myUc::addClass(const std::string &code) { classCode = code; }

void myUc::addClassInfo(std::string type, std::string day, int dayInt,
                        double startTime, double duration) {
  classInfo newClassInfo;
  newClassInfo.type = type;
  newClassInfo.day = day;
  newClassInfo.dayInt = dayInt;
  newClassInfo.startTime = startTime;
  newClassInfo.duration = duration;

  classInfoVec.push_back(newClassInfo);
}

bool myUc::compareUcCode(const myUc &a, const myUc &b) {
  return a.ucCode < b.ucCode;
}
```

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.

```
#ifndef MYUC_H
#define MYUC_H

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

struct classInfo {
  std::string code;
  std::string type;
  std::string day;
  int dayInt;
  double startTime;
  double duration;

  bool operator<(const classInfo &other) const {
    return startTime < other.startTime;
  }
};

struct classQtd {
  std::string classCode;
  int qtd;

  bool operator<(const classQtd &other) const {
    return classCode < other.classCode;
  }
};

class myUc {
private:
  std::string classCode;
  std::string ucCode;
  std::vector<classInfo> classInfoVec;

public:
  // Constructor functions
  myUc(const std::string &ucC, std::string &classC);
  myUc();

  // Setter functions
  void SetUc(std::string &ucC, std::string &classC);
  void setUcCode(std::string &n);
  void setClassCode(std::string &n);
  // void setClassCode(std::string &n);

  // Getters functions
  std::string getUcCode() const;
  std::string getClassCode() const;
  std::vector<classInfo> getClassInfoVec() const;

  // Others functions
  void addClass(const std::string &code);
  void addClassInfo(std::string type, std::string day, int dayInt,
                    double startTime, double duration);

  bool operator<(const myUc &other) const;
  static bool compareUcCode(const myUc &a, const myUc &b);
};

#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)

  Function to compare students based on their codes in ascending order.
- bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)

  Function to compare students based on their codes in descending order.
- bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)

  Function to compare students based on their names in ascending order.
- bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)

  Function to compare students based on their names in descending order.
- std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)

  Filter student information based on specified criteria.
- std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)

  Order a vector of students based on the specified criterion.

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

  Select students from a map based on a given student code.
- void organizerUcStudent (std::map< std::string, myStudent >::iterator &it)

  Organize the classes of a student based on Uc Code in ascending order.
- bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter, std::map< std::string, std::vector< classQtd > > &count)

  Remove a specific Uc Code from a student's classes.
- void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter)

  Add a new class to a student's record.
- void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector< classQtd > > &count, int type)

  Update the class count tree by adding or removing a class.
- bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

  Validate the addition of a new class to a student's schedule.
- std::map< int, std::set< classInfo > > orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

  Organize a student's classes by day.
- std::string weekDayString (int day)

  Convert a numeric day value to a corresponding day of the week string.
- bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)

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

4.11.1 Function Documentation

4.11.1.1 addClassStudent()

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

Add a new class to a student's record.

Parameters

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

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

Definition at line 222 of file dbStudents.cpp.

```
                                                    {

    myUc classe(ucCode, classCode);
    it->second.getClasses().push_back(classe);
    organizerUcStudent(it);
```

```
stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                 "add", ucCode, classCode});
}
```

4.11.1.2 compareStudentNameAsc()

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

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

Parameters

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

Returns

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

Definition at line 35 of file dbStudents.cpp.

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

4.11.1.3 compareStudentNameDesc()

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

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

Parameters

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

Returns

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

Definition at line 47 of file dbStudents.cpp.

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

4.11.1.4 compareStudentsCodeAsc()

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

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

Parameters

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

Returns

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

Definition at line 11 of file dbStudents.cpp.

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

4.11.1.5 compareStudentsCodeDesc()

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

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

Parameters

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

Returns

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

Definition at line 23 of file dbStudents.cpp.

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

4.11.1.6 filterInfoStudent()

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

Filter student information based on specified criteria.

Parameters

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

Returns

Vector of students matching the criteria.

Definition at line 63 of file dbStudents.cpp.

```
                                                              {
  std::vector<myStudent> filterStudents;
  switch (n) {
  case 1:
    // Filter by Uc Code
    for (const auto &student : students) {
      for (const auto &uc : student.getClasses()) {
        if (uc.getUcCode() == str) {
          filterStudents.push_back(student);
          break;
        }
      }
    }
    break;
  case 2:
    // Filter by Class Code
    for (const auto &student : students) {
      for (const auto &uc : student.getClasses()) {
        for (const auto &classInfo : uc.getClassInfoVec()) {
          if (classInfo.code == str) {
            filterStudents.push_back(student);
            break; // No need to check other class codes for this student
          }
        }
      }
    }
    break;
  default:
    errorMessage();
    break;
  }
  return filterStudents;
}
```

4.11.1.7 orderInfoStudent()

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

Order a vector of students based on the specified criterion.

Parameters

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

Returns

Ordered vector of students based on the specified criterion.

Definition at line 108 of file dbStudents.cpp.

```
                                                                            {
  switch (n) {
  case 1:
    // Order by Student Code Asc
    std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
    break;
  case 2:
    // Order by Student Code Desc
    std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
    break;
  case 3:
    // Order by Student Name Asc
    std::sort(students.begin(), students.end(), compareStudentNameAsc);
    break;
  case 4:
    // Order by Student Name Desc
    std::sort(students.begin(), students.end(), compareStudentNameDesc);
    break;
  default:
    errorMessage();
    break;
  }

  return students;
}
```

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 )
```

Organize a student's classes by day.

Parameters

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

Returns

Map of classes organized by day.

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

Definition at line 345 of file dbStudents.cpp.

```
                                                    {

  // map to order the classes
  // by day
  std::map<int, std::set<classInfo» orderClasses;

  // for each class of the
  // student, search in the
  // class tree and add the
  // classInfo in the
  // orderClasses map
  for (const auto &classe : it->second.getClasses()) {
    std::string value = classe.getUcCode() + classe.getClassCode();

    // student one class
    // pointer, verify if the
    // class exists in the
    // class tree
    auto it_class = classes.find(value);

    // if the class does not
    // exist, print error
    if (it_class == classes.end()) {
      std::cerr « "Error in "
                   "find class"
                « std::endl;
    } else {
      // if exists, add the
      // classInfo in the
      // orderClasses map
      for (auto &classInfo : it_class->second.getClassInfoVec()) {
        classInfo.code = classe.getUcCode();
        orderClasses[classInfo.dayInt].insert(classInfo);
      }
    }
  }
  return orderClasses;
}
```

4.11.1.9 organizerUcStudent()

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

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

Parameters

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

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

Definition at line 170 of file dbStudents.cpp.

```
                                                                  {

  std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
            myUc::compareUcCode);
}
```

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 )
```

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

Parameters

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

Returns

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

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

Definition at line 190 of file dbStudents.cpp.

```
                                                                    {

  bool remove = false;
  for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
    if (it->second.getClasses()[i].getUcCode() == ucCode) {
      stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                       "remove", ucCode,
                       it->second.getClasses()[i].getUcCode()});
      it->second.getClasses().erase(it->second.getClasses().begin() + i);
      remove = true;
      updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
                         count, 0);
    }
  }
  return remove;
}
```

4.11.1.11 selectStudent()

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

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

Parameters

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

Returns

Map of selected students with matching student codes.

Definition at line 145 of file dbStudents.cpp.

```
                                                                    {
  std::map<std::string, myStudent> selectedStudents;

  for (auto &studentPair : students) {
    const myStudent &mystudent = studentPair.second;
    if (str == mystudent.getStudentCode()) {
      selectedStudents[studentPair.first] = mystudent;
    }
  }

  return selectedStudents;
}
```

4.11.1.12 updateCountClasses()

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

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

Parameters

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

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

Definition at line 248 of file dbStudents.cpp.

```
                                           {

  auto it_count = count.find(ucCode);
  if (it_count != count.end()) {
    for (auto &classe : it_count->second) {
      if (classe.classCode == classCode) {
        if (type == 1) {
          classe.qtd++;
        } else {
          classe.qtd--;
        }
      }
    }
  }
}
```

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 )
```

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

Parameters

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

Returns

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

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

Definition at line 283 of file dbStudents.cpp.

```
                                                      {

  // call function to order the
  // classes of the student by
  // int day
  std::map<int, std::set<classInfo» orderClasses =
      orderStudentClass(it, classes);

  std::string value = ucCode + classCode;

  auto it_class = classes.find(value);

  if (it_class == classes.end()) {
    std::cout « "Error in "
                 "find class"
              « std::endl;
    return true;
  } else {
    // verify if has a class in
    // the same day and time
    for (const auto &class_info : it_class->second.getClassInfoVec()) {
      // get all classes of the
      // day of class
      const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];

      // and verify if the
      // student has a class in
      // the same time aula -> student classes
      // class_info -> class to add
      for (const auto &aula : classesOfDay) {

        if (aula.type != "T" && class_info.type != "T" &&
            class_info.startTime >= aula.startTime &&
            class_info.startTime < aula.startTime + aula.duration) {
```

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

4.11.1.14 verifyUcCode()

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

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

Parameters

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

Returns

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

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

Definition at line 430 of file dbStudents.cpp.

```
00431                                                             {
00432 
00433   for (const auto &classe : it->second.getClasses()) {
00434     if (classe.getUcCode() == ucCode) {
00435       return true;
00436     }
00437   }
00438 
00439   return false;
00440 }
```

4.11.1.15 weekDayString()

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

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

Parameters

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

Returns

Corresponding day of the week string.

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

Definition at line 393 of file dbStudents.cpp.

```
00393                                      {
00394   switch (day) {
00395   case 2:
00396     return "Monday";
00397     break;
00398   case 3:
00399     return "Tuesday";
00400     break;
00401   case 4:
00402     return "Wednesday";
00403     break;
00404   case 5:
00405     return "Thursday";
00406     break;
00407   case 6:
00408     return "Friday";
00409     break;
00410   case 7:
00411     return "Saturday";
00412     break;
00413   default:
00414     return "Error Day";
00415     break;
00416   }
00417 }
```

4.12 dbStudents.cpp

Go to the documentation of this file.

```
00001 #include "dbStudents.h"
00002
00003 // O(1)
00011 bool compareStudentsCodeAsc(const myStudent &student1,
00012                             const myStudent &student2) {
00013   return student1.getStudentCode() < student2.getStudentCode();
00014 }
00015 // O(1)
00023 bool compareStudentsCodeDesc(const myStudent &student1,
00024                              const myStudent &student2) {
00025   return student1.getStudentCode() > student2.getStudentCode();
00026 }
00027 // O(1)
00035 bool compareStudentNameAsc(const myStudent &student1,
00036                            const myStudent &student2) {
00037   return student1.getStudentName() < student2.getStudentName();
00038 }
00039 // O(1)
00047 bool compareStudentNameDesc(const myStudent &student1,
00048                             const myStudent &student2) {
00049   return student1.getStudentName() > student2.getStudentName();
00050 }
00051
00052 // O(n)
00053 // n = number of lines in the file
00062 std::vector<myStudent>
00063 filterInfoStudent(int n, std::string str,
00064                   const std::vector<myStudent> &students) {
00065   std::vector<myStudent> filterStudents;
00066   switch (n) {
00067   case 1:
00068     // Filter by Uc Code
00069     for (const auto &student : students) {
00070       for (const auto &uc : student.getClasses()) {
00071         if (uc.getUcCode() == str) {
00072           filterStudents.push_back(student);
00073           break;
00074         }
00075       }
00076     }
00077     break;
00078   case 2:
    // Filter by Class Code
    for (const auto &student : students) {
      for (const auto &uc : student.getClasses()) {
        for (const auto &classInfo : uc.getClassInfoVec()) {
          if (classInfo.code == str) {
            filterStudents.push_back(student);
            break; // No need to check other class codes for this student
          }
        }
      }
    }
    break;
  default:
    errorMessage();
    break;
  }
  return filterStudents;
}

// O(n*log(n))
// n = number of lines in the file
std::vector<myStudent> orderInfoStudent(int n,
                                        std::vector<myStudent> &students) {
  switch (n) {
  case 1:
    // Order by Student Code Asc
    std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
    break;
  case 2:
    // Order by Student Code Desc
    std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
    break;
  case 3:
    // Order by Student Name Asc
    std::sort(students.begin(), students.end(), compareStudentNameAsc);
    break;
  case 4:
    // Order by Student Name Desc
    std::sort(students.begin(), students.end(), compareStudentNameDesc);
    break;
  default:
    errorMessage();
    break;
  }

  return students;
}

// O(m)
// m = number of students
std::map<std::string, myStudent>
selectStudent(const std::string &str,
              const std::map<std::string, myStudent> &students) {
  std::map<std::string, myStudent> selectedStudents;

  for (auto &studentPair : students) {
    const myStudent &mystudent = studentPair.second;
    if (str == mystudent.getStudentCode()) {
      selectedStudents[studentPair.first] = mystudent;
    }
  }

  return selectedStudents;
}

// ------------------------------------------------ //

// O(m*log(m))
// m = number of UCs of the student
void organizerUcStudent(std::map<std::string, myStudent>::iterator &it) {

  std::sort(it->second.getClasses().begin(), it->second.getClasses().end(),
            myUc::compareUcCode);
}

// O(m)
// m = number of UCs of the student
bool removeUcStudent(std::string ucCode,
                     std::map<std::string, myStudent>::iterator &it,
                     std::stack<alter> &stackAlter,
                     std::map<std::string, std::vector<classQtd» &count) {

  bool remove = false;
  for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
    if (it->second.getClasses()[i].getUcCode() == ucCode) {
      stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                       "remove", ucCode,
                        it->second.getClasses()[i].getUcCode()});
        it->second.getClasses().erase(it->second.getClasses().begin() + i);
        remove = true;
        updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
                           count, 0);
      }
    }
    return remove;
}

// O(log(m))
// m = number of distinct student
void addClassStudent(std::string ucCode, std::string classCode,
                     std::map<std::string, myStudent>::iterator &it,
                     std::stack<alter> &stackAlter) {

  myUc classe(ucCode, classCode);
  it->second.getClasses().push_back(classe);
  organizerUcStudent(it);
  stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                   "add", ucCode, classCode});
}


// O(m*log(n)
// n = number of UCs
// m = number of classes

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

  auto it_count = count.find(ucCode);
  if (it_count != count.end()) {
    for (auto &classe : it_count->second) {
      if (classe.classCode == classCode) {
        if (type == 1) {
          classe.qtd++;
        } else {
          classe.qtd--;
        }
      }
    }
  }
}


// O(m*log(n)*k)
// n = number of UCs
// m = number of classes of the student
// k = number of type of classes (T,TP)

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

  // call function to order the
  // classes of the student by
  // int day
  std::map<int, std::set<classInfo» orderClasses =
      orderStudentClass(it, classes);

  std::string value = ucCode + classCode;

  auto it_class = classes.find(value);

  if (it_class == classes.end()) {
    std::cout « "Error in "
                "find class"
             « std::endl;
    return true;
  } else {
    // verify if has a class in
    // the same day and time
    for (const auto &class_info : it_class->second.getClassInfoVec()) {
      // get all classes of the
      // day of class
      const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];

      // and verify if the
      // student has a class in
      // the same time aula -> student classes
      // class_info -> class to add
      for (const auto &aula : classesOfDay) {

        if (aula.type != "T" && class_info.type != "T" &&
            class_info.startTime >= aula.startTime &&
             class_info.startTime < aula.startTime + aula.duration) {
           std::cout « "Error: "
                       "Incompatible"
                       " schedules"
                    « std::endl;
           return true;
         }
       }
     }
     return false;
   }
}

// O(m*log(n)*k)
// n = number of UCs
// m = number of classes
// k = number of type of classes (T,TP)
std::map<int, std::set<classInfo»
orderStudentClass(std::map<std::string, myStudent>::iterator &it,
                  std::map<std::string, myUc> &classes) {

  // map to order the classes
  // by day
  std::map<int, std::set<classInfo» orderClasses;

  // for each class of the
  // student, search in the
  // class tree and add the
  // classInfo in the
  // orderClasses map
  for (const auto &classe : it->second.getClasses()) {
    std::string value = classe.getUcCode() + classe.getClassCode();

    // student one class
    // pointer, verify if the
    // class exists in the
    // class tree
    auto it_class = classes.find(value);

    // if the class does not
    // exist, print error
    if (it_class == classes.end()) {
      std::cerr « "Error in "
                  "find class"
                « std::endl;
    } else {
      // if exists, add the
      // classInfo in the
      // orderClasses map
      for (auto &classInfo : it_class->second.getClassInfoVec()) {
        classInfo.code = classe.getUcCode();
        orderClasses[classInfo.dayInt].insert(classInfo);
      }
    }
  }
  return orderClasses;
}

std::string weekDayString(int day) {
  switch (day) {
  case 2:
    return "Monday";
    break;
  case 3:
    return "Tuesday";
    break;
  case 4:
    return "Wednesday";
    break;
  case 5:
    return "Thursday";
    break;
  case 6:
    return "Friday";
    break;
  case 7:
    return "Saturday";
    break;
  default:
    return "Error Day";
    break;
  }
}

// O(m)
// m = number of classes of the student
bool verifyUcCode(std::string ucCode,
                    std::map<std::string, myStudent>::iterator &it) {

  for (const auto &classe : it->second.getClasses()) {
    if (classe.getUcCode() == ucCode) {
      return true;
    }
  }

  return false;
}
```

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)

  Select students from a map based on a given student code.
- std::vector< myStudent > filterInfoStudent (int n, std::string str, const std::vector< myStudent > &students)

  Filter student information based on specified criteria.
- std::vector< myStudent > orderInfoStudent (int n, std::vector< myStudent > &students)

  Order a vector of students based on the specified criterion.
- bool removeUcStudent (std::string ucCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter, std::map< std::string, std::vector< classQtd > > &count)

  Remove a specific Uc Code from a student's classes.
- void addClassStudent (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::stack< alter > &stackAlter)

  Add a new class to a student's record.
- std::map< int, std::set< classInfo > > orderStudentClass (std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

  Organize a student's classes by day.
- bool valideNewClass (std::string ucCode, std::string classCode, std::map< std::string, myStudent >::iterator &it, std::map< std::string, myUc > &classes)

  Validate the addition of a new class to a student's schedule.
- void updateCountClasses (std::string ucCode, std::string classCode, std::map< std::string, std::vector< classQtd > > &count, int type)

  Update the class count tree by adding or removing a class.
- std::string weekDayString (int day)

  Convert a numeric day value to a corresponding day of the week string.
- bool verifyUcCode (std::string ucCode, std::map< std::string, myStudent >::iterator &it)

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

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

    Function to compare students based on their codes in ascending order.
- bool compareStudentsCodeDesc (const myStudent &student1, const myStudent &student2)

    Function to compare students based on their codes in descending order.
- bool compareStudentNameAsc (const myStudent &student1, const myStudent &student2)

    Function to compare students based on their names in ascending order.
- bool compareStudentNameDesc (const myStudent &student1, const myStudent &student2)

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

4.13.1 Function Documentation

4.13.1.1 addClassStudent()

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

Add a new class to a student's record.

Parameters

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

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

Definition at line 222 of file dbStudents.cpp.

```
                                                    {

    myUc classe(ucCode, classCode);
    it->second.getClasses().push_back(classe);
    organizerUcStudent(it);
    stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                     "add", ucCode, classCode});
}
```

4.13.1.2 compareStudentNameAsc()

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

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

Parameters

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

Returns

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

Definition at line 35 of file dbStudents.cpp.

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

4.13.1.3 compareStudentNameDesc()

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

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

Parameters

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

Returns

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

Definition at line 47 of file dbStudents.cpp.

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

4.13.1.4 compareStudentsCodeAsc()

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

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

Parameters

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

Returns

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

Definition at line 11 of file dbStudents.cpp.

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

4.13.1.5 compareStudentsCodeDesc()

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

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

Parameters

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

Returns

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

Definition at line 23 of file dbStudents.cpp.

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

4.13.1.6 errorMessage()

```
void errorMessage ( )
```

Definition at line 4 of file errorMsgs.cpp.

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

4.13.1.7 filterInfoStudent()

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

Filter student information based on specified criteria.

Parameters

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

Returns

Vector of students matching the criteria.

Definition at line 63 of file dbStudents.cpp.

```
                                                         {
 std::vector<myStudent> filterStudents;
 switch (n) {
 case 1:
  // Filter by Uc Code
  for (const auto &student : students) {
   for (const auto &uc : student.getClasses()) {
    if (uc.getUcCode() == str) {
     filterStudents.push_back(student);
     break;
    }
   }
  }
  break;
 case 2:
  // Filter by Class Code
  for (const auto &student : students) {
   for (const auto &uc : student.getClasses()) {
    for (const auto &classInfo : uc.getClassInfoVec()) {
     if (classInfo.code == str) {
      filterStudents.push_back(student);
      break; // No need to check other class codes for this student
     }
    }
   }
  }
  break;
 default:
  errorMessage();
  break;
 }
 return filterStudents;
}
```

4.13.1.8 orderInfoStudent()

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

Order a vector of students based on the specified criterion.

Parameters

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

Returns

Ordered vector of students based on the specified criterion.

Definition at line 108 of file dbStudents.cpp.

```
                                                                         {
 switch (n) {
 case 1:
  // Order by Student Code Asc
  std::sort(students.begin(), students.end(), compareStudentsCodeAsc);
  break;
 case 2:
  // Order by Student Code Desc
  std::sort(students.begin(), students.end(), compareStudentsCodeDesc);
  break;
 case 3:
  // Order by Student Name Asc
  std::sort(students.begin(), students.end(), compareStudentNameAsc);
  break;
 case 4:
  // Order by Student Name Desc
  std::sort(students.begin(), students.end(), compareStudentNameDesc);
  break;
  default:
    errorMessage();
    break;
  }

  return students;
}
```

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 )
```

Organize a student's classes by day.

Parameters

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

Returns

Map of classes organized by day.

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

Definition at line 345 of file dbStudents.cpp.

```
                                                    {

  // map to order the classes
  // by day
  std::map<int, std::set<classInfo» orderClasses;

  // for each class of the
  // student, search in the
  // class tree and add the
  // classInfo in the
  // orderClasses map
  for (const auto &classe : it->second.getClasses()) {
    std::string value = classe.getUcCode() + classe.getClassCode();

    // student one class
    // pointer, verify if the
    // class exists in the
    // class tree
    auto it_class = classes.find(value);

    // if the class does not
    // exist, print error
    if (it_class == classes.end()) {
      std::cerr « "Error in "
                  "find class"
                « std::endl;
    } else {
      // if exists, add the
      // classInfo in the
      // orderClasses map
      for (auto &classInfo : it_class->second.getClassInfoVec()) {
        classInfo.code = classe.getUcCode();
        orderClasses[classInfo.dayInt].insert(classInfo);
      }
    }
  }
  return orderClasses;
}
```

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 )
```

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

Parameters

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

Returns

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

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

Definition at line 190 of file dbStudents.cpp.

```
                                                                        {

  bool remove = false;
  for (unsigned i = 0; i < it->second.getClasses().size(); i++) {
    if (it->second.getClasses()[i].getUcCode() == ucCode) {
      stackAlter.push({it->second.getStudentCode(), it->second.getStudentName(),
                       "remove", ucCode,
                       it->second.getClasses()[i].getUcCode()});
      it->second.getClasses().erase(it->second.getClasses().begin() + i);
      remove = true;
      updateCountClasses(ucCode, it->second.getClasses()[i].getClassCode(),
                         count, 0);
    }
  }
  return remove;
}
```

4.13.1.11 selectStudent()

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

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

Parameters

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

Returns

Map of selected students with matching student codes.

Definition at line 145 of file dbStudents.cpp.

```
                                                                            {
  std::map<std::string, myStudent> selectedStudents;

  for (auto &studentPair : students) {
    const myStudent &mystudent = studentPair.second;
    if (str == mystudent.getStudentCode()) {
      selectedStudents[studentPair.first] = mystudent;
    }
  }

  return selectedStudents;
}
```

4.13.1.12 updateCountClasses()

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

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

Parameters

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

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

Definition at line 248 of file dbStudents.cpp.

```
                                        {

  auto it_count = count.find(ucCode);
  if (it_count != count.end()) {
    for (auto &classe : it_count->second) {
      if (classe.classCode == classCode) {
        if (type == 1) {
          classe.qtd++;
        } else {
          classe.qtd--;
        }
      }
    }
  }
}
```

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 )
```

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

Parameters

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

Returns

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

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

Definition at line 283 of file dbStudents.cpp.

```
                                                        {

  // call function to order the
  // classes of the student by
  // int day
  std::map<int, std::set<classInfo» orderClasses =
      orderStudentClass(it, classes);

  std::string value = ucCode + classCode;

  auto it_class = classes.find(value);

  if (it_class == classes.end()) {
    std::cout « "Error in "
                 "find class"
              « std::endl;
    return true;
  } else {
    // verify if has a class in
    // the same day and time
    for (const auto &class_info : it_class->second.getClassInfoVec()) {
      // get all classes of the
      // day of class
      const std::set<classInfo> &classesOfDay = orderClasses[class_info.dayInt];

      // and verify if the
      // student has a class in
      // the same time aula -> student classes
      // class_info -> class to add
      for (const auto &aula : classesOfDay) {

        if (aula.type != "T" && class_info.type != "T" &&
            class_info.startTime >= aula.startTime &&
            class_info.startTime < aula.startTime + aula.duration) {
          std::cout « "Error: "
                      "Incompatible"
                      " schedules"
                   « std::endl;
          return true;
        }
      }
    }
    return false;
  }
}
```

4.13.1.14 verifyUcCode()

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

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

Parameters

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

Returns

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

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

Definition at line 430 of file dbStudents.cpp.

```
                                                               {

  for (const auto &classe : it->second.getClasses()) {
    if (classe.getUcCode() == ucCode) {
      return true;
    }
  }

  return false;
}
```

4.13.1.15 weekDayString()

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

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

Parameters

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

Returns

Corresponding day of the week string.

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

Definition at line 393 of file dbStudents.cpp.

```
                                {
  switch (day) {
  case 2:
    return "Monday";
    break;
  case 3:
    return "Tuesday";
    break;
  case 4:
    return "Wednesday";
    break;
  case 5:
    return "Thursday";
    break;
  case 6:
    return "Friday";
    break;
  case 7:
    return "Saturday";
    break;
  default:
    return "Error Day";
    break;
  }
}
```

4.14 dbStudents.h

Go to the documentation of this file.

```
#ifndef DBSTUDENTS_H
#define DBSTUDENTS_H

#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
#include <list>
#include <map>
#include <set>
#include <stack>
#include <string>
#include <vector>

#include "../classes/student.h"

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);

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);

#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 11 of file dbUcs.cpp.

```
00011                                                                  {
00012   return uc1.getClassCode() < uc2.getClassCode();
00013 }
```

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 33 of file dbUcs.cpp.

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

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 22 of file dbUcs.cpp.

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

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 44 of file dbUcs.cpp.

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

4.15.1.5 filterInfoUc()

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

Filters UC information.

Parameters

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

Returns

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

Definition at line 58 of file dbUcs.cpp.

```
                                                                                   {
  std::vector<myUc> filterUc;
  switch (n) {
  case 1:
    // Filter by Uc Code
    for (const auto &uc : ucs) {
      if (uc.getUcCode() == str) {
        filterUc.push_back(uc);
      }
    }
    break;
  case 2:
    // Filter by Class Code
    for (const auto &uc : ucs) {
      for (const auto &classInfo : uc.getClassInfoVec()) {
        if (classInfo.code == str) {
          filterUc.push_back(uc);
          break;
        }
      }
    }
    break;
  default:
    errorMessage();
    break;
  }
  return filterUc;
}
```

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 95 of file dbUcs.cpp.
```
                                                                 {
  switch (n) {
  case 1:
    // Order by Uc Code Asc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
    break;
  case 2:
    // Order by Uc Code Desc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
    break;
  case 3:
    // Order by Class Code Asc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
    break;
  case 4:
    // Order by Class Code Desc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
    break;
  default:
    errorMessage();
    break;
  }
  return ucs;
}
```

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 130 of file dbUcs.cpp.
```
                                                                           {
  std::vector<myUc> selectedUcs;

  for (const auto &pair : classes) {
    auto ucObj = pair.second;

    if (ucObj.getUcCode() == str) {
      selectedUcs.push_back(ucObj);
    }
  }
  return selectedUcs;
}
```

4.16 dbUcs.cpp

Go to the documentation of this file.

```
#include "dbUcs.h"

// O(1)
bool compareClassesCodeAsc(const myUc &uc1, const myUc &uc2) {
  return uc1.getClassCode() < uc2.getClassCode();
}

// O(1)
bool compareUcsCodeASC(const myUc &uc1, const myUc &uc2) {
  return uc1.getUcCode() < uc2.getUcCode();
}

// O(1)
bool compareClassesCodeDesc(const myUc &uc1, const myUc &uc2) {
  return uc1.getClassCode() > uc2.getClassCode();
}

// O(1)
bool compareUcsCodeDesc(const myUc &uc1, const myUc &uc2) {
  return uc1.getUcCode() > uc2.getUcCode();
}

// Average: O(n)
// Best: O(1)
// n = number of lines in the file
std::vector<myUc> filterInfoUc(int n, std::string str, std::vector<myUc> &ucs) {
  std::vector<myUc> filterUc;
  switch (n) {
  case 1:
    // Filter by Uc Code
    for (const auto &uc : ucs) {
      if (uc.getUcCode() == str) {
        filterUc.push_back(uc);
      }
    }
    break;
  case 2:
    // Filter by Class Code
    for (const auto &uc : ucs) {
      for (const auto &classInfo : uc.getClassInfoVec()) {
        if (classInfo.code == str) {
          filterUc.push_back(uc);
          break;
        }
      }
    }
    break;
  default:
    errorMessage();
    break;
  }
  return filterUc;
}

// O(n*log(n))
// n = number of lines in the file
std::vector<myUc> orderInfoUc(int n, std::vector<myUc> &ucs) {

  switch (n) {
  case 1:
    // Order by Uc Code Asc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
    break;
  case 2:
    // Order by Uc Code Desc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
    break;
  case 3:
    // Order by Class Code Asc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
    break;
  case 4:
    // Order by Class Code Desc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
    break;
  default:
    errorMessage();
    break;
  }
  return ucs;
}

```

```
// O(n)
// n = number of lines in the file
std::vector<myUc> selectUc(const std::string &str,
                           const std::map<std::string, myUc> &classes) {
  std::vector<myUc> selectedUcs;

  for (const auto &pair : classes) {
    auto ucObj = pair.second;

    if (ucObj.getUcCode() == str) {
      selectedUcs.push_back(ucObj);
    }
  }
  return selectedUcs;
}
```

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.

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

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 58 of file dbUcs.cpp.

```
                                                                              {
  std::vector<myUc> filterUc;
  switch (n) {
  case 1:
    // Filter by Uc Code
    for (const auto &uc : ucs) {
      if (uc.getUcCode() == str) {
        filterUc.push_back(uc);
      }
    }
    break;
  case 2:
    // Filter by Class Code
    for (const auto &uc : ucs) {
      for (const auto &classInfo : uc.getClassInfoVec()) {
        if (classInfo.code == str) {
          filterUc.push_back(uc);
          break;
        }
      }
    }
    break;
  default:
    errorMessage();
    break;
  }
  return filterUc;
}
```

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 95 of file dbUcs.cpp.

```
                                                                  {

  switch (n) {
  case 1:
    // Order by Uc Code Asc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeASC);
    break;
  case 2:
    // Order by Uc Code Desc
    std::sort(ucs.begin(), ucs.end(), compareUcsCodeDesc);
    break;
  case 3:
    // Order by Class Code Asc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeAsc);
    break;
  case 4:
    // Order by Class Code Desc
    std::sort(ucs.begin(), ucs.end(), compareClassesCodeDesc);
    break;
  default:
    errorMessage();
    break;
  }
  return ucs;
}
```

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 130 of file dbUcs.cpp.

```
                                                                            {
  std::vector<myUc> selectedUcs;

  for (const auto &pair : classes) {
    auto ucObj = pair.second;

    if (ucObj.getUcCode() == str) {
      selectedUcs.push_back(ucObj);
    }
  }
  return selectedUcs;
}
```

4.18 dbUcs.h

Go to the documentation of this file.

```
#ifndef DBUCS_H
#define DBUCS_H

#include <algorithm>
#include <iostream>
#include <map>
#include <string>
#include <vector>

#include "../classes/uc.h"

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

#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 204 of file keepAllChanges.cpp.

```
                               {

  std::string path = "schedule/backup/" + backups[cdBkp];

  std::ifstream backup(path, std::ios::binary);

  if (!backup) {
    std::cerr « "Error opening file" « std::endl;
  }

  std::ofstream file("schedule/students_classes.csv", std::ios::binary);

  if (!file) {
    std::cerr « "Error opening file" « std::endl;
  }

  file « backup.rdbuf();
  file.close();
  backup.close();

  unsigned size = cdBkp;
  for (unsigned i = 0; i <= size; i++) {
    if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
      try {
        std::filesystem::remove("schedule/alter/" + backups[i]);
        std::filesystem::remove("schedule/backup/" + backups[i]);
      } catch (const std::filesystem::filesystem_error &e) {
        std::cerr « "Error to remove the file" « e.what() « std::endl;
      }
    } else {
      std::cout « "The file of changes not exist" « std::endl;
    }
  }
}
```

4.19.1.2 getSysdate()

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

Get the system date.

Returns

A string with the system date.

Definition at line 22 of file keepAllChanges.cpp.

```
                           {

```

```
  std::time_t date = std::time(0);
  std::tm *now = std::localtime(&date);

  return std::to_string(now->tm_year + 1900) + "-" +
         std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
         "-" + std::to_string(now->tm_hour) + ":" +
         std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
}
```

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 69 of file keepAllChanges.cpp.

```
                                                      {
  makeBackup();
  std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
                        ".csv",
                    std::ios::app);
  if (!alter.is_open()) {
    std::cerr « "Error opening file" « std::endl;
  }

  while (!stackAlter.empty()) {
    alter « "The student: " « stackAlter.top().studentCode « " - "
          « stackAlter.top().studentName « " " « stackAlter.top().type
          « " UC: " « stackAlter.top().ucCode
          « " Class: " « stackAlter.top().classCode « std::endl;
    stackAlter.pop();
  }

  std::ofstream file("schedule/students_classes.csv");

  if (!file.is_open()) {
    std::cerr « "Error opening file" « std::endl;
  }

  // Header
  file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;

  // Write the tree in the file
  for (auto it = students.begin(); it != students.end(); it++) {
    for (auto classe : it->second.getClasses()) {
      file « it->second.getStudentCode() « "," « it->second.getStudentName()
           « "," « classe.getUcCode() « "," « classe.getClassCode()
           « std::endl;
    }
  }
}
```

4.19.1.4 listAllBackups()

```
void listAllBackups ( )
```

List all backup files.

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

Definition at line 115 of file keepAllChanges.cpp.

```
                                {
  if (backups.size() == 0) {
    std::string way = "schedule/backup";
    for (const auto &in : std::filesystem::directory_iterator(way)) {
      if (std::filesystem::is_regular_file(in)) {
        backups.push_back(in.path().filename().string());
      }
    }
    std::sort(backups.begin(), backups.end(), orderVector);
  }
}
```

4.19.1.5 makeBackup()

```
void makeBackup ( )
```

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

Definition at line 38 of file keepAllChanges.cpp.

```
                        {
  std::ifstream file("schedule/students_classes.csv", std::ios::binary);

  if (!file) {
    std::cerr « "Error opening file" « std::endl;
  }

  std::string dateString = getSysdate();

  std::string backupName =
      "schedule/backup/students_classes-" + dateString + ".csv";
  std::ofstream backup(backupName, std::ios::binary);

  if (!backup) {
    std::cerr « "Error to create a backup file" « std::endl;
    return;
  }

  backup « file.rdbuf();
  file.close();
  backup.close();
}
```

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 13 of file keepAllChanges.cpp.

```
                                                                 {
  return str1 > str2;
}
```

4.19.1.7 printAllBackups()

```
bool printAllBackups ( )
```

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

Definition at line 132 of file keepAllChanges.cpp.

```
                               {
  if (backups.size() != 0) {
    std::cout « "Backups: " « std::endl;
    for (unsigned i = 0; i < backups.size(); i++) {
      std::cout « i « " - " « backups.at(i) « std::endl;
    }
    return true;
  } else {
    std::cout « "No backups" « std::endl;
    return false;
  }
}
```

4.19.1.8 printChanges()

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

Print the changes from backup files.

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

Parameters

cdBkp	The index of the backup files to print.

Definition at line 158 of file keepAllChanges.cpp.

```
                                     {
  unsigned size = cdBkp;
  for (unsigned i = 0; i <= size; i++) {
    std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);

    if (!file) {
      std::cerr « "Error opening file" « std::endl;
    }

    std::string line;

    while (std::getline(file, line)) {
      std::cout « "    " « line « std::endl;
    }
    file.close();
  }

  // // Write the tree in the file
  // for (auto it = students.begin(); it != students.end(); it++) {
  //   // for (auto classe : it->second.getClasses()) {
  //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
  //   «
  //   //   ","
  //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
  //   std::endl;
  //   //   file « it->second.getStudentCode() « "," «
  //   //   it->second.getStudentName() « ","
  //   //       « classe.getUcCode() « "," « classe.getClassCode() «
  //   //       std::endl;
  //   // }
  // }
}
```

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
00006 //O(1)
00013 bool orderVector(const std::string &str1, const std::string &str2) {
00014   return str1 > str2;
00015 }
00016
00017 //O(1)
00022 std::string getSysdate() {
00023
00024   std::time_t date = std::time(0);
00025   std::tm *now = std::localtime(&date);
00026
00027   return std::to_string(now->tm_year + 1900) + "-" +
00028          std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
00029          "-" + std::to_string(now->tm_hour) + ":" +
00030          std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
00031 }
00032
00033 //O(1)
00038 void makeBackup() {
00039   std::ifstream file("schedule/students_classes.csv", std::ios::binary);
00040
00041   if (!file) {
00042     std::cerr « "Error opening file" « std::endl;
00043   }
00044
00045   std::string dateString = getSysdate();
00046
00047   std::string backupName =
00048       "schedule/backup/students_classes-" + dateString + ".csv";
00049   std::ofstream backup(backupName, std::ios::binary);
00050
00051   if (!backup) {
00052     std::cerr « "Error to create a backup file" « std::endl;
00053     return;
00054   }
00055
00056   backup « file.rdbuf();
00057   file.close();
00058   backup.close();
00059 }
00060
00061 // O(m)
00062 // m = number of changes
00069 void keepAllChanges(std::map<std::string, myStudent> &students,
00070                     std::stack<alter> &stackAlter) {
00071   makeBackup();
00072   std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
00073                           ".csv",
00074                       std::ios::app);
00075   if (!alter.is_open()) {
00076     std::cerr « "Error opening file" « std::endl;
00077   }
00078
00079   while (!stackAlter.empty()) {
00080     alter « "The student: " « stackAlter.top().studentCode « " - "
00081           « stackAlter.top().studentName « " " « stackAlter.top().type
00082           « " UC: " « stackAlter.top().ucCode
00083           « " Class: " « stackAlter.top().classCode « std::endl;
00084     stackAlter.pop();
00085   }
00086
```

```
  std::ofstream file("schedule/students_classes.csv");

  if (!file.is_open()) {
    std::cerr « "Error opening file" « std::endl;
  }

  // Header
  file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;

  // Write the tree in the file
  for (auto it = students.begin(); it != students.end(); it++) {
    for (auto classe : it->second.getClasses()) {
      file « it->second.getStudentCode() « "," « it->second.getStudentName()
           « "," « classe.getUcCode() « "," « classe.getClassCode()
           « std::endl;
    }
  }
}

// O(m)
// m = number of backups files
// Ideally, use a script to maintain a maximum of 10
void listAllBackups() {
  if (backups.size() == 0) {
    std::string way = "schedule/backup";
    for (const auto &in : std::filesystem::directory_iterator(way)) {
      if (std::filesystem::is_regular_file(in)) {
        backups.push_back(in.path().filename().string());
      }
    }
    std::sort(backups.begin(), backups.end(), orderVector);
  }
}

// O(m)
// m = number of backups files
bool printAllBackups() {
  if (backups.size() != 0) {
    std::cout « "Backups: " « std::endl;
    for (unsigned i = 0; i < backups.size(); i++) {
      std::cout « i « " - " « backups.at(i) « std::endl;
    }
    return true;
  } else {
    std::cout « "No backups" « std::endl;
    return false;
  }
}

// O(m)
// m = number of backups files
// Best case: O(1) when the user select the newer backup
// Worst case: O(m) when the user select the older backup
void printChanges(int cdBkp) {
  unsigned size = cdBkp;
  for (unsigned i = 0; i <= size; i++) {
    std::ifstream file("schedule/alter/" + backups[i], std::ios::binary);

    if (!file) {
      std::cerr « "Error opening file" « std::endl;
    }

    std::string line;

    while (std::getline(file, line)) {
      std::cout « "   " « line « std::endl;
    }
    file.close();
  }

  // // Write the tree in the file
  // for (auto it = students.begin(); it != students.end(); it++) {
  //   // for (auto classe : it->second.getClasses()) {
  //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
  //   «
  //   //   ","
  //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
  //   std::endl;
  //   //   file « it->second.getStudentCode() « "," «
  //   //   it->second.getStudentName() « ","
  //   //        « classe.getUcCode() « "," « classe.getClassCode() «
  //   //        std::endl;
  //   // }
  // }
}

// O(m)
// m = number of backups files
// Best case: O(1) when the user select the newer backup
// Worst case: O(m) when the user select the older backup
void backupFile(int cdBkp) {

  std::string path = "schedule/backup/" + backups[cdBkp];

  std::ifstream backup(path, std::ios::binary);

  if (!backup) {
    std::cerr « "Error opening file" « std::endl;
  }

  std::ofstream file("schedule/students_classes.csv", std::ios::binary);

  if (!file) {
    std::cerr « "Error opening file" « std::endl;
  }

  file « backup.rdbuf();
  file.close();
  backup.close();

  unsigned size = cdBkp;
  for (unsigned i = 0; i <= size; i++) {
    if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
      try {
        std::filesystem::remove("schedule/alter/" + backups[i]);
        std::filesystem::remove("schedule/backup/" + backups[i]);
      } catch (const std::filesystem::filesystem_error &e) {
        std::cerr « "Error to remove the file" « e.what() « std::endl;
      }
    } else {
      std::cout « "The file of changes not exist" « std::endl;
    }
  }
}
```

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 204 of file keepAllChanges.cpp.

```
                              {

  std::string path = "schedule/backup/" + backups[cdBkp];

  std::ifstream backup(path, std::ios::binary);

  if (!backup) {
    std::cerr « "Error opening file" « std::endl;
  }

  std::ofstream file("schedule/students_classes.csv", std::ios::binary);

  if (!file) {
    std::cerr « "Error opening file" « std::endl;
  }

  file « backup.rdbuf();
  file.close();
  backup.close();

  unsigned size = cdBkp;
  for (unsigned i = 0; i <= size; i++) {
    if (std::filesystem::exists("schedule/alter/" + backups[cdBkp])) {
      try {
        std::filesystem::remove("schedule/alter/" + backups[i]);
        std::filesystem::remove("schedule/backup/" + backups[i]);
      } catch (const std::filesystem::filesystem_error &e) {
        std::cerr « "Error to remove the file" « e.what() « std::endl;
      }
    } else {
      std::cout « "The file of changes not exist" « std::endl;
    }
  }
}
```

4.21.1.2 getSysdate()

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

Get the system date.

Returns

A string with the system date.

Definition at line 22 of file keepAllChanges.cpp.

```
                          {

  std::time_t date = std::time(0);
  std::tm *now = std::localtime(&date);

  return std::to_string(now->tm_year + 1900) + "-" +
         std::to_string(now->tm_mon + 1) + "-" + std::to_string(now->tm_mday) +
         "-" + std::to_string(now->tm_hour) + ":" +
         std::to_string(now->tm_min) + ":" + std::to_string(now->tm_sec);
}
```

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 69 of file keepAllChanges.cpp.

```
                                                         {
  makeBackup();
  std::ofstream alter("schedule/alter/students_classes-" + getSysdate() +
                          ".csv",
                      std::ios::app);
  if (!alter.is_open()) {
    std::cerr « "Error opening file" « std::endl;
  }

  while (!stackAlter.empty()) {
    alter « "The student: " « stackAlter.top().studentCode « " - "
          « stackAlter.top().studentName « " " « stackAlter.top().type
          « " UC: " « stackAlter.top().ucCode
          « " Class: " « stackAlter.top().classCode « std::endl;
    stackAlter.pop();
  }

  std::ofstream file("schedule/students_classes.csv");

  if (!file.is_open()) {
    std::cerr « "Error opening file" « std::endl;
  }

  // Header
  file « "StudentCode,StudentName,UcCode,ClassCode" « std::endl;

  // Write the tree in the file
  for (auto it = students.begin(); it != students.end(); it++) {
    for (auto classe : it->second.getClasses()) {
      file « it->second.getStudentCode() « "," « it->second.getStudentName()
           « "," « classe.getUcCode() « "," « classe.getClassCode()
           « std::endl;
    }
  }
}
```

4.21.1.5 listAllBackups()

```
void listAllBackups ( )
```

List all backup files.

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

Definition at line 115 of file keepAllChanges.cpp.

```
                        {
  if (backups.size() == 0) {
    std::string way = "schedule/backup";
    for (const auto &in : std::filesystem::directory_iterator(way)) {
      if (std::filesystem::is_regular_file(in)) {
        backups.push_back(in.path().filename().string());
      }
    }
    std::sort(backups.begin(), backups.end(), orderVector);
  }
}
```

4.21.1.6 makeBackup()

```
void makeBackup ( )
```

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

Definition at line 38 of file keepAllChanges.cpp.

```
                    {
  std::ifstream file("schedule/students_classes.csv", std::ios::binary);

  if (!file) {
    std::cerr « "Error opening file" « std::endl;
  }

  std::string dateString = getSysdate();

  std::string backupName =
      "schedule/backup/students_classes-" + dateString + ".csv";
  std::ofstream backup(backupName, std::ios::binary);

  if (!backup) {
    std::cerr « "Error to create a backup file" « std::endl;
    return;
  }

  backup « file.rdbuf();
  file.close();
  backup.close();
}
```

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 13 of file keepAllChanges.cpp.

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

4.21.1.8 printAllBackups()

```
bool printAllBackups ( )
```

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

Definition at line 132 of file keepAllChanges.cpp.

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

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 158 of file keepAllChanges.cpp.

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

```
    while (std::getline(file, line)) {
      std::cout « "    " « line « std::endl;
    }
    file.close();
  }

  // // Write the tree in the file
  // for (auto it = students.begin(); it != students.end(); it++) {
  //   // for (auto classe : it->second.getClasses()) {
  //   //   // std::cout« it->second.getCode() « "," « it->second.getName()
  //   «
  //   //   ","
  //   //   // « classe.getUcCode() « "," « classe.getClassCode() «
  //   std::endl;
  //   //   file « it->second.getStudentCode() « "," «
  //   //   it->second.getStudentName() « ","
  //   //        « classe.getUcCode() « "," « classe.getClassCode() «
  //   //        std::endl;
  //   // }
  // }
}
```

4.22 keepAllChanges.h

Go to the documentation of this file.

```
#ifndef KEEPALLCHANGES_H
#define KEEPALLCHANGES_H

#include <algorithm>
#include <ctime>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <map>
#include <stack>
#include <string>
#include <vector>

#include "../classes/student.h"

void makeBackup();
bool orderVector(const std::string &str1, const std::string &str2);
void keepAllChanges(std::map<std::string, myStudent> &students,
                    std::stack<alter> &stackAlter);
std::string getSysdate();
void listAllBackups();
void printChanges(int cdBkp);
bool printAllBackups();
void backupFile(int cdBkp);

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

#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 202 of file print.cpp.

```
                                                                       {

  auto it_count = count.find(ucCode);
  std::list<std::string> free_classes;

  if (it_count != count.end()) {
    free_classes = valideFreeClass(it_count);
    std::cout « "   Classes: " « std::endl;

    if (!free_classes.empty()) {
      for (auto it_list = free_classes.begin(); it_list != free_classes.end();
           it_list++) {
        std::cout « "      " « *it_list « std::endl;
      }
    } else {
      std::cout « "      No classes available" « std::endl;
    }
  } else {
    std::cout « " Uc not found" « std::endl;
  }
}
```

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.

```
                                                                        {
  std::cout « "Student Code | Student Name"
            « std::endl;

  for (const auto &studentPair : students) {
    const myStudent &student = studentPair.second;
    std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
    std::cout « "    " « "Classes: " « std::endl;
    for(const auto &classe : student.getClasses()) {
      std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
    }

  }
}
```

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 69 of file print.cpp.

```
                                                                           {
  system("clear");
  std::cout « "\nCode: " « it->first « " - ";
  std::cout « "Name: " « it->second.getStudentName() « std::endl;
  std::cout « "Classes: " « std::endl;
  for (const auto &classe : it->second.getClasses()) {
    std::cout « "  " « classe.getUcCode() « " - " « classe.getClassCode()
              « std::endl;
  }
}
```

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 40 of file print.cpp.

```
                                                          {
  std::cout « "Student Code | Student Name"
            « std::endl;

  if (students.empty()) {
    std::cout « "Empty vector ucs" « std::endl;
  }

  for (const auto &student : students) {
    std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
    std::cout « "    " « "Classes: " « std::endl;
    for(const auto &classe : student.getClasses()) {
      std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
    }
  }
}
```

4.23.1.5 printStudentSchedules()

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

Print available free classes for a specific UC.

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

Parameters

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

Definition at line 244 of file print.cpp.

```
                                                              {
  auto orderClasses = orderStudentClass(it, classes);
  std::cout « "\nSchedules: " « std::endl;
  for (const auto &pair : orderClasses) {
    std::string day = weekDayString(pair.first);
    std::cout « "Day: " « day « std::endl;
    for (const auto &info : pair.second) {
      std::cout « info.code « " - ";
      std::cout « info.startTime « " to ";
      std::cout « info.startTime + info.duration « " - ";
      std::cout « info.type « std::endl;
    }
    std::cout « std::endl;
  }
}
```

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 92 of file print.cpp.

```
                                                        {
    std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
              « std::endl;

    for (const auto &classes : ucVector) {

      auto infoVec = classes.getClassInfoVec();
      for (const auto &classInfo : infoVec) {
        std::string type = classInfo.type;
        std::string day = classInfo.day;
        int dayInt = classInfo.dayInt;
        double startTime = classInfo.startTime;
        double duration = classInfo.duration;
        std::cout « classes.getUcCode() « " | " « classes.getClassCode()
                  « " | " « type « " | " « day « " | " « dayInt « " | "
                  « startTime « " | " « duration « std::endl;
      }
    }
}
```

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 121 of file print.cpp.

```
                                            {
    std::cout « "UcCode | ClassCode" « std::endl;

    for (const auto &uc : ucs) {
      std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
    }
}
```

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 142 of file print.cpp.

```
                                                                 {
  int min = INT_MAX;
  std::list<std::string> free_classes;

  // first verify the class with the minimum number of students
  for (auto &classe : it_count->second) {
    if (classe.qtd < min) {
      min = classe.qtd;
    }
  }
  // then verify if the class is able to accept new students and add to the
  // list
  for (auto &classe : it_count->second) {
    if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
      free_classes.push_back(classe.classCode);
    }
  }

  // return list
  return free_classes;
}
```

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 181 of file print.cpp.

```
                                                                        {
  auto it_count = count.find(ucCode);

  if (it_count != count.end()) {
    std::list<std::string> free_classes = valideFreeClass(it_count);
    for (auto it_list = free_classes.begin(); it_list != free_classes.end();
         it_list++) {
      if (*it_list == classCode) {
        return true;
      }
    }
  } else {
    std::cout « "Error in find uc" « std::endl;
  }
  return false;
}
```

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 
00029 //O(n)
00030 // n = number of lines in the file
00040 void printStudents(const std::vector<myStudent> &students) {
00041   std::cout « "Student Code | Student Name"
00042             « std::endl;
00043 
00044   if (students.empty()) {
00045     std::cout « "Empty vector ucs" « std::endl;
00046   }
00047 
00048   for (const auto &student : students) {
00049     std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
00050     std::cout « "    " « "Classes: " « std::endl;
00051     for(const auto &classe : student.getClasses()) {
00052       std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
00053     }
00054   }
00055 }
00056 
00057 // O(m)
00058 // Find the student in the map
00059 // m = number of students
00060 // m >= 7
00069 void printStudentClasses(std::map<std::string, myStudent>::iterator &it) {
00070   system("clear");
00071   std::cout « "\nCode: " « it->first « " - ";
00072   std::cout « "Name: " « it->second.getStudentName() « std::endl;
00073   std::cout « "Classes: " « std::endl;
00074   for (const auto &classe : it->second.getClasses()) {
00075     std::cout « "  " « classe.getUcCode() « " - " « classe.getClassCode()
00076               « std::endl;
00077   }
00078 }
00079 
00080 //O(n)
00081 // n = number of lines in the file
00092 void printUcClasses(const std::vector<myUc> &ucVector) {
00093   std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
00094             « std::endl;
00095 
00096   for (const auto &classes : ucVector) {
00097 
00098     auto infoVec = classes.getClassInfoVec();
00099     for (const auto &classInfo : infoVec) {
00100       std::string type = classInfo.type;
```

```
      std::string day = classInfo.day;
      int dayInt = classInfo.dayInt;
      double startTime = classInfo.startTime;
      double duration = classInfo.duration;
      std::cout « classes.getUcCode() « " | " « classes.getClassCode()
                « " | " « type « " | " « day « " | " « dayInt « " | "
                « startTime « " | " « duration « std::endl;
    }
  }
}

// O(n)
void printUcs(const std::vector<myUc> &ucs) {
  std::cout « "UcCode | ClassCode" « std::endl;

  for (const auto &uc : ucs) {
    std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
  }
}

// O(m)
// Iterator to UC
// m = number of classes in the UC
std::list<std::string> valideFreeClass(
    std::map<std::string, std::vector<classQtd»::iterator it_count) {
  int min = INT_MAX;
  std::list<std::string> free_classes;

  // first verify the class with the minimum number of students
  for (auto &classe : it_count->second) {
    if (classe.qtd < min) {
      min = classe.qtd;
    }
  }
  // then verify if the class is able to accept new students and add to the
  // list
  for (auto &classe : it_count->second) {
    if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
      free_classes.push_back(classe.classCode);
    }
  }

  // return list
  return free_classes;
}

// O(log(n) + m)
// n = number of UCs
// m = number of free classes

bool verifyClassCode(std::string classCode, std::string ucCode,
                     std::map<std::string, std::vector<classQtd» &count) {
  auto it_count = count.find(ucCode);

  if (it_count != count.end()) {
    std::list<std::string> free_classes = valideFreeClass(it_count);
    for (auto it_list = free_classes.begin(); it_list != free_classes.end();
         it_list++) {
      if (*it_list == classCode) {
        return true;
      }
    }
  } else {
    std::cout « "Error in find uc" « std::endl;
  }
  return false;
}

// O(log(n) + m)
// n = number of UCs
// m = number of free classes
void printFreeClasses(std::string ucCode,
                      std::map<std::string, std::vector<classQtd» &count) {

  auto it_count = count.find(ucCode);
  std::list<std::string> free_classes;

  if (it_count != count.end()) {
    free_classes = valideFreeClass(it_count);
    std::cout « "   Classes: " « std::endl;

    if (!free_classes.empty()) {
      for (auto it_list = free_classes.begin(); it_list != free_classes.end();
           it_list++) {
        std::cout « "      " « *it_list « std::endl;
      }
    } else {
      std::cout « "      No classes available" « std::endl;
    }
  } else {
    std::cout « " Uc not found" « std::endl;
  }
}

// orderStudentClass:
// O(m*log(n)*k)
// n = number of UCs
// m = number of classes
// k = number of type of classes (T,TP)

// O(m*k)
// m = number of classes
// k = number of type of classes (T,TP)

void printStudentSchedules(std::map<std::string, myStudent>::iterator &it,
                           std::map<std::string, myUc> &classes) {
  auto orderClasses = orderStudentClass(it, classes);
  std::cout « "\nSchedules: " « std::endl;
  for (const auto &pair : orderClasses) {
    std::string day = weekDayString(pair.first);
    std::cout « "Day: " « day « std::endl;
    for (const auto &info : pair.second) {
      std::cout « info.code « " - ";
      std::cout « info.startTime « " to ";
      std::cout « info.startTime + info.duration « " - ";
      std::cout « info.type « std::endl;
    }
    std::cout « std::endl;
  }
}
```

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.

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

4.25.1.2 printFreeClasses()

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

Definition at line 202 of file print.cpp.

```
                                                                            {

  auto it_count = count.find(ucCode);
  std::list<std::string> free_classes;

  if (it_count != count.end()) {
    free_classes = valideFreeClass(it_count);
    std::cout « "   Classes: " « std::endl;

    if (!free_classes.empty()) {
      for (auto it_list = free_classes.begin(); it_list != free_classes.end();
           it_list++) {
        std::cout « "      " « *it_list « std::endl;
      }
    } else {
      std::cout « "      No classes available" « std::endl;
    }
  } else {
    std::cout « " Uc not found" « std::endl;
  }
}
```

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.

```
                                                                {
  std::cout « "Student Code | Student Name"
            « std::endl;

  for (const auto &studentPair : students) {
    const myStudent &student = studentPair.second;
    std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
    std::cout « "    " « "Classes: " « std::endl;
    for(const auto &classe : student.getClasses()) {
      std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
    }

  }
}
```

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 69 of file print.cpp.

```
                                                                   {
  system("clear");
  std::cout « "\nCode: " « it->first « " - ";
  std::cout « "Name: " « it->second.getStudentName() « std::endl;
  std::cout « "Classes: " « std::endl;
  for (const auto &classe : it->second.getClasses()) {
    std::cout « "  " « classe.getUcCode() « " - " « classe.getClassCode()
              « std::endl;
  }
}
```

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 40 of file print.cpp.

```
                                                                {
  std::cout « "Student Code | Student Name"
            « std::endl;

  if (students.empty()) {
    std::cout « "Empty vector ucs" « std::endl;
  }

  for (const auto &student : students) {
    std::cout « student.getStudentCode() « " | " « student.getStudentName() « std::endl;
    std::cout « "    " « "Classes: " « std::endl;
    for(const auto &classe : student.getClasses()) {
      std::cout « "    " « classe.getUcCode() « " - " « classe.getClassCode() « std::endl;
    }
  }
}
```

4.25.1.6 printStudentSchedules()

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

Print available free classes for a specific UC.

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

Parameters

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

Definition at line 244 of file print.cpp.

```
                                                                    {
  auto orderClasses = orderStudentClass(it, classes);
  std::cout « "\nSchedules: " « std::endl;
  for (const auto &pair : orderClasses) {
    std::string day = weekDayString(pair.first);
    std::cout « "Day: " « day « std::endl;
    for (const auto &info : pair.second) {
      std::cout « info.code « " - ";
      std::cout « info.startTime « " to ";
      std::cout « info.startTime + info.duration « " - ";
      std::cout « info.type « std::endl;
    }
    std::cout « std::endl;
  }
}
```

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 92 of file print.cpp.

```
                                                    {
  std::cout « "UcCode | ClassCode | Type | Day | DayInt | StartTime | Duration"
            « std::endl;

  for (const auto &classes : ucVector) {

    auto infoVec = classes.getClassInfoVec();
    for (const auto &classInfo : infoVec) {
      std::string type = classInfo.type;
      std::string day = classInfo.day;
      int dayInt = classInfo.dayInt;
      double startTime = classInfo.startTime;
      double duration = classInfo.duration;
      std::cout « classes.getUcCode() « " | " « classes.getClassCode()
                « " | " « type « " | " « day « " | " « dayInt « " | "
                « startTime « " | " « duration « std::endl;
    }
  }
}
```

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 121 of file print.cpp.

```
                                          {
  std::cout « "UcCode | ClassCode" « std::endl;

  for (const auto &uc : ucs) {
    std::cout « uc.getUcCode() « " | " « uc.getClassCode() « std::endl;
  }
}
```

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 142 of file print.cpp.

```
                                                             {
  int min = INT_MAX;
  std::list<std::string> free_classes;

  // first verify the class with the minimum number of students
  for (auto &classe : it_count->second) {
    if (classe.qtd < min) {
      min = classe.qtd;
    }
  }
  // then verify if the class is able to accept new students and add to the
  // list
  for (auto &classe : it_count->second) {
    if (!(classe.qtd + 1 - min > equilibre) && classe.qtd + 1 <= max_students) {
      free_classes.push_back(classe.classCode);
    }
  }

  // return list
  return free_classes;
}
```

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 181 of file print.cpp.

```
                                                                    {
  auto it_count = count.find(ucCode);

  if (it_count != count.end()) {
    std::list<std::string> free_classes = valideFreeClass(it_count);
    for (auto it_list = free_classes.begin(); it_list != free_classes.end();
         it_list++) {
      if (*it_list == classCode) {
        return true;
      }
    }
  } else {
    std::cout « "Error in find uc" « std::endl;
  }
  return false;
}
```

4.25.1.11 workingMessage()

```
void workingMessage ( )
```

Definition at line 26 of file errorMsgs.cpp.

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

4.26 print.h

Go to the documentation of this file.

```
#ifndef PRINT_H
#define PRINT_H

#include <algorithm>
#include <climits>
#include <fstream>
#include <iostream>
#include <list>
#include <map>
#include <string>
#include <vector>

#include "../classes/student.h"
#include "../functions/dbStudents.h"

void workingMessage();
void errorMessage();

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

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

void printStudentSchedules(std::map<std::string, myStudent>::iterator &it,
                           std::map<std::string, myUc> &classes);
void printStudentClasses(std::map<std::string, myStudent>::iterator &it);
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);
bool verifyClassCode(std::string classCode, std::string ucCode,
                     std::map<std::string, std::vector<classQtd» &count);

#endif
```

4.27 src/inputoutput/read.cpp File Reference

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

Functions

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

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

  Read and process UC and class information from a CSV file.
- std::map< std::string, myUc > readSchedules ()

  Read and process class schedule information from a CSV file.

4.27.1 Function Documentation

4.27.1.1 readSchedules()

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

Read and process class schedule information from a CSV file.

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

Returns

A map of classes with their associated information.

Definition at line 185 of file read.cpp.

```
                                        { // O(n*log(n))
  std::string line;
  std::map<std::string, myUc> classes;
  std::map<std::string, int> dayToInt = {
      {"Sunday", 1},    {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
      {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};

  std::ifstream file("schedule/classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string classCode, ucCode, day, type;
    double startTime, duration;
    int dayInt = 0;

    std::getline(ss, classCode, ',');
    std::getline(ss, ucCode, ',');
    std::getline(ss, day, ',');
    ss » startTime;
    ss.ignore();
    ss » duration;
    ss.ignore();
    std::getline(ss, type);

    type.erase(std::find_if(type.rbegin(), type.rend(),
                            [](unsigned char ch) { return !std::isspace(ch); })
                   .base(),
               type.end());

    auto it1 = dayToInt.find(day);
    if (it1 != dayToInt.end()) {
      dayInt = it1->second;
    } else {
      std::cout « "Invalid day: " « day « std::endl;
    }

    // Check if the class code already exists in the map
    auto it2 = classes.find(ucCode + classCode);
    if (it2 != classes.end()) {
      it2->second.addClassInfo(type, day, dayInt, startTime, duration);
    } else {
      myUc newUcClass;
      newUcClass.setUcCode(ucCode);
      newUcClass.addClass(classCode);
      newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
      classes[ucCode + classCode] = newUcClass;
    }
  }
  return classes;
}
```

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 17 of file read.cpp.

```
                                                             {
  std::string line;
  std::map<std::string, myStudent> students;

  std::ifstream file("schedule/students_classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);

    std::string studentCode, studentName, ucCode, classCode;

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

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    auto it = students.find(studentCode);
    if (it != students.end()) {
      it->second.addClass(myUc(ucCode, classCode));
    } else {
      myStudent newStudent(studentCode, studentName);
      newStudent.addClass(myUc(ucCode, classCode));
      students[studentCode] = newStudent;
    }

    // verify if the uc exists in the count tree
    auto it_count = count.find(ucCode);

    // if not exists, add the class with one student in the count tree
    if (it_count == count.end()) {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 1});
      count.emplace(ucCode, classVec);
    } else {
      // if uc exist, then verify if the class exists in the vector
      bool exist = false;
      for (auto &class_it : it_count->second) {
        // if exists, add +1 in the qtd
        if (class_it.classCode == classCode) {
          class_it.qtd++;
          exist = true;
          break;
        }
      }
      // if not exists, add the class with one student in the vector
      if (!exist) {
        it_count->second.push_back({classCode, 1});
      }
    }
  }
  file.close();

  return students;
}
```

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 100 of file read.cpp.

```
                                                         {
  std::string line;
  std::map<std::string, std::vector<myUc» ucClasses;

  std::ifstream file("schedule/classes_per_uc.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    // testing
    // std::cout « "line" « std::endl;

    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string ucCode, classCode;

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

    auto it = ucClasses.find(ucCode);

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    if (it != ucClasses.end()) {
      // exist
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      it->second.push_back(newUc);
    } else {
      // doesnt exist
      std::vector<myUc> ucVector;
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      ucVector.push_back(newUc);
      ucClasses[ucCode] = ucVector;
    }

    bool exist = false;
    // try to find the uc in the count tree
    auto it_count = count.find(ucCode);
    // if found, verify if the class exists in the vector
    if (it_count != count.end()) {
      for (auto &class_it : it_count->second) {
        if (class_it.classCode == classCode) {
          exist = true;
        }
      }
      // if exist uc in the count tree, but not exist the class, add the class
      // with 0 students
      if (!exist) {
        it_count->second.push_back({classCode, 0});
      }
      // if not found, add the uc and class with 0 students
    } else {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 0});
      count.emplace(ucCode, classVec);
    }
  }
  file.close();
  return ucClasses;
}
```

4.28 read.cpp

Go to the documentation of this file.

```
#include "read.h"


// O(n*log(m))
// n = number of lines in the file
// m = number of differents students
std::map<std::string, myStudent>
readStudents(std::map<std::string, std::vector<classQtd» &count) {
  std::string line;
  std::map<std::string, myStudent> students;

  std::ifstream file("schedule/students_classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);

    std::string studentCode, studentName, ucCode, classCode;

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

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    auto it = students.find(studentCode);
    if (it != students.end()) {
      it->second.addClass(myUc(ucCode, classCode));
    } else {
      myStudent newStudent(studentCode, studentName);
      newStudent.addClass(myUc(ucCode, classCode));
      students[studentCode] = newStudent;
    }

    // verify if the uc exists in the count tree
    auto it_count = count.find(ucCode);

    // if not exists, add the class with one student in the count tree
    if (it_count == count.end()) {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 1});
      count.emplace(ucCode, classVec);
    } else {
      // if uc exist, then verify if the class exists in the vector
      bool exist = false;
      for (auto &class_it : it_count->second) {
        // if exists, add +1 in the qtd
        if (class_it.classCode == classCode) {
          class_it.qtd++;
          exist = true;
          break;
        }
      }
      // if not exists, add the class with one student in the vector
      if (!exist) {
        it_count->second.push_back({classCode, 1});
      }
    }
  }
  file.close();

  return students;
}

//O(n*log(m))
// n = number of lines in the file
// m = number of differents UCs
std::map<std::string, std::vector<myUc» // O(n*log(n))
readUcs(std::map<std::string, std::vector<classQtd» &count) {
  std::string line;
  std::map<std::string, std::vector<myUc» ucClasses;

  std::ifstream file("schedule/classes_per_uc.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    // testing
    // std::cout « "line" « std::endl;

    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string ucCode, classCode;

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

    auto it = ucClasses.find(ucCode);

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    if (it != ucClasses.end()) {
      // exist
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      it->second.push_back(newUc);
    } else {
      // doesnt exist
      std::vector<myUc> ucVector;
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      ucVector.push_back(newUc);
      ucClasses[ucCode] = ucVector;
    }

    bool exist = false;
    // try to find the uc in the count tree
    auto it_count = count.find(ucCode);
    // if found, verify if the class exists in the vector
    if (it_count != count.end()) {
      for (auto &class_it : it_count->second) {
        if (class_it.classCode == classCode) {
          exist = true;
        }
      }
      // if exist uc in the count tree, but not exist the class, add the class
      // with 0 students
      if (!exist) {
        it_count->second.push_back({classCode, 0});
      }
      // if not found, add the uc and class with 0 students
    } else {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 0});
      count.emplace(ucCode, classVec);
    }
  }
  file.close();
  return ucClasses;
}

//O(n*log(m))
// n = number of lines in the file
// m = number of differents classes
std::map<std::string, myUc> readSchedules() { // O(n*log(n))
  std::string line;
  std::map<std::string, myUc> classes;
  std::map<std::string, int> dayToInt = {
      {"Sunday", 1},    {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
      {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};

  std::ifstream file("schedule/classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string classCode, ucCode, day, type;
    double startTime, duration;
    int dayInt = 0;

    std::getline(ss, classCode, ',');
    std::getline(ss, ucCode, ',');
    std::getline(ss, day, ',');
    ss » startTime;
    ss.ignore();
    ss » duration;
    ss.ignore();
    std::getline(ss, type);

    type.erase(std::find_if(type.rbegin(), type.rend(),
                            [](unsigned char ch) { return !std::isspace(ch); })
                   .base(),
               type.end());

    auto it1 = dayToInt.find(day);
    if (it1 != dayToInt.end()) {
      dayInt = it1->second;
    } else {
      std::cout « "Invalid day: " « day « std::endl;
    }

    // Check if the class code already exists in the map
    auto it2 = classes.find(ucCode + classCode);
    if (it2 != classes.end()) {
      it2->second.addClassInfo(type, day, dayInt, startTime, duration);
    } else {
      myUc newUcClass;
      newUcClass.setUcCode(ucCode);
      newUcClass.addClass(classCode);
      newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
      classes[ucCode + classCode] = newUcClass;
    }
  }
  return classes;
}
```

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 185 of file read.cpp.

```
                                              { // O(n*log(n))
  std::string line;
  std::map<std::string, myUc> classes;
  std::map<std::string, int> dayToInt = {
      {"Sunday", 1},    {"Monday", 2}, {"Tuesday", 3}, {"Wednesday", 4},
      {"Thursday", 5}, {"Friday", 6}, {"Saturday", 7}};

  std::ifstream file("schedule/classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string classCode, ucCode, day, type;
    double startTime, duration;
    int dayInt = 0;

    std::getline(ss, classCode, ',');
    std::getline(ss, ucCode, ',');
    std::getline(ss, day, ',');
    ss » startTime;
    ss.ignore();
    ss » duration;
    ss.ignore();
    std::getline(ss, type);

    type.erase(std::find_if(type.rbegin(), type.rend(),
                            [](unsigned char ch) { return !std::isspace(ch); })
                   .base(),
               type.end());

    auto it1 = dayToInt.find(day);
    if (it1 != dayToInt.end()) {
      dayInt = it1->second;
    } else {
      std::cout « "Invalid day: " « day « std::endl;
    }

    // Check if the class code already exists in the map
    auto it2 = classes.find(ucCode + classCode);
    if (it2 != classes.end()) {
      it2->second.addClassInfo(type, day, dayInt, startTime, duration);
    } else {
      myUc newUcClass;
      newUcClass.setUcCode(ucCode);
      newUcClass.addClass(classCode);
      newUcClass.addClassInfo(type, day, dayInt, startTime, duration);
      classes[ucCode + classCode] = newUcClass;
    }
  }
  return classes;
}
```

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 17 of file read.cpp.

```
                                                            {
  std::string line;
  std::map<std::string, myStudent> students;

  std::ifstream file("schedule/students_classes.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);

    std::string studentCode, studentName, ucCode, classCode;

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

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    auto it = students.find(studentCode);
    if (it != students.end()) {
      it->second.addClass(myUc(ucCode, classCode));
    } else {
      myStudent newStudent(studentCode, studentName);
      newStudent.addClass(myUc(ucCode, classCode));
      students[studentCode] = newStudent;
    }

    // verify if the uc exists in the count tree
    auto it_count = count.find(ucCode);

    // if not exists, add the class with one student in the count tree
    if (it_count == count.end()) {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 1});
      count.emplace(ucCode, classVec);
    } else {
      // if uc exist, then verify if the class exists in the vector
      bool exist = false;
      for (auto &class_it : it_count->second) {
        // if exists, add +1 in the qtd
        if (class_it.classCode == classCode) {
          class_it.qtd++;
          exist = true;
          break;
        }
      }
      // if not exists, add the class with one student in the vector
      if (!exist) {
        it_count->second.push_back({classCode, 1});
      }
    }
  }
  file.close();

  return students;
}
```

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 100 of file read.cpp.

```
                                                         {
  std::string line;
  std::map<std::string, std::vector<myUc» ucClasses;

  std::ifstream file("schedule/classes_per_uc.csv");
  if (!file.is_open()) {
    errorMessageFile();
  }

  bool header = true;
  while (std::getline(file, line)) {
    // testing
    // std::cout « "line" « std::endl;

    if (header) {
      header = false;
      continue;
    }
    std::istringstream ss(line);
    std::string ucCode, classCode;

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

    auto it = ucClasses.find(ucCode);

    classCode.erase(
        std::find_if(classCode.rbegin(), classCode.rend(),
                     [](unsigned char ch) { return !std::isspace(ch); })
            .base(),
        classCode.end());

    if (it != ucClasses.end()) {
      // exist
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      it->second.push_back(newUc);
    } else {
      // doesnt exist
      std::vector<myUc> ucVector;
      myUc newUc;
      newUc.setUcCode(ucCode);
      newUc.setClassCode(classCode);
      ucVector.push_back(newUc);
      ucClasses[ucCode] = ucVector;
    }

    bool exist = false;
    // try to find the uc in the count tree
    auto it_count = count.find(ucCode);
    // if found, verify if the class exists in the vector
    if (it_count != count.end()) {
      for (auto &class_it : it_count->second) {
        if (class_it.classCode == classCode) {
          exist = true;
        }
      }
      // if exist uc in the count tree, but not exist the class, add the class
      // with 0 students
      if (!exist) {
        it_count->second.push_back({classCode, 0});
      }
      // if not found, add the uc and class with 0 students
    } else {
      std::vector<classQtd> classVec;
      classVec.push_back({classCode, 0});
      count.emplace(ucCode, classVec);
    }
  }
  file.close();
  return ucClasses;
}
```

4.30 read.h

Go to the documentation of this file.

```
#ifndef READ_H
#define READ_H

#include <algorithm>
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

#include "../classes/student.h"

void errorMessageFile();
void errorMessageLine(std::string);

std::map<std::string, myStudent>
readStudents(std::map<std::string, std::vector<classQtd» &count);
std::map<std::string, std::vector<myUc»
readUcs(std::map<std::string, std::vector<classQtd» &count);

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

#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.

```
            {

  menu();

  return 0;
}
```

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.

```
             {

  menuUpdate();
  system("clear");

  int flag = 0;

  std::cout « "------------ Welcome to our app :) ------------" « std::endl;
  std::cout « "| 1) See database                              |" « std::endl;
  std::cout « "| 2) Change database                           |" « std::endl;
  std::cout « "| 3) Backup                                    |" « std::endl;
  std::cout « "| 4) Exit                                      |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  switch (flag) {
  case 1:
    menuSeeDatabase();
    break;
  case 2:
    menuRequests();
    break;
  case 3:
    menuBackup();
    break;
  case 4:
    exit(0);
  default:
    errorMessage();
    break;
  }
}
```

4.32 main.cpp

Go to the documentation of this file.

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

void menu();

int main() {

  menu();

  return 0;
}
```

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) {
```

```
          addClassStudent(ucCode, classCode, it, stackAlter);
          printStudentClasses(it);
          std::cout « "\nSucessfully added" « std::endl;

          saveOrReturn();
        }
      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Class code not found" « std::endl;
        menuTryAgain(1, it);
      }
    }
  } else {
    std::cout « "-----------------------------------------------"
              « std::endl;
    std::cout « "You are already enrolled in this UC" « std::endl;
    menuTryAgain(1, it);
  }
 }
}
```

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.

```
                   {
  int flag;
  system("clear");
  listAllBackups();

  bool valide = printAllBackups();
  if (valide == true) {
    printChanges(selectBackupCode(0));
    menuChanges();
  } else {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| 1) - Main menu                              |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;
    std::cin » flag;

    if (flag == 1) {
      menu();
    } else {
      errorMessage();
    }
  }
}
```

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.

```
                    {

  int flag;

  std::cout « "-----------------------------------------------" « std::endl;
  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):
    printStudentSchedules(it, classes);
    break;
  default:
    errorMessage();
  }
}
```

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.

```
                                                                     {
  std::map<std::string, myStudent> oneStudent = students;
  std::vector<myStudent> data;

  for (const auto &studentPair : students) {
    data.push_back(studentPair.second);
  }

  if (type == 1) {
    oneStudent = selectStudent(str, oneStudent);
    printStudent(oneStudent);
  } else {
    if (type == 2) {
      data = filterInfoStudent(filter, str, data);
    }
    data = orderInfoStudent(order, data);
    printStudents(data);
  }
}
```

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) {
        removeUcStudent(ucCode, it, stackAlter, count);
        validate = valideNewClass(ucCode, classCode, it, classes);
        if (!validate) {
          addClassStudent(ucCode, classCode, it, stackAlter);
          printStudentClasses(it);
          std::cout « "\nSuccessfully switched" « std::endl;
          saveOrReturn();
        }
      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Class code not found" « std::endl;
        menuTryAgain(3, it);
      }
    } else {
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "You are not enrolled in this UC" « std::endl;
      menuTryAgain(3, it);
    }
    break;
  default:
    errorMessage();
    break;
  }
}
```

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.

```
                                                                  {
  int flag;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Try again                                 |" « std::endl;
  std::cout « "| 2) Exit                                      |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cin » flag;

  switch (flag) {
  case 1:
    system("clear");
    if (menuType == 1) {
      menuAdd(it);
    } else if (menuType == 2) {
      menuRemove(it);
    } else if (menuType == 3) {
      menuSwitch(it);
    }
    break;
  case 2:
    exit(0);
  default:
    errorMessage();
    break;
  }
}
```

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.

```
                                                           {
  std::vector<myUc> data;
  std::vector<myUc> oneUc;

  for (const auto &ucVectorPair : ucs) {
    for (const myUc &ucObj : ucVectorPair.second) {
      data.push_back(ucObj);
    }
  }

  if (type == 1) {
    oneUc = selectUc(str, classes);
    printUcClasses(oneUc);
  } else {
    if (type == 2) {
      data = filterInfoUc(filter, str, data);
    }
    data = orderInfoUc(order, data);
    printUcs(data);
  }
}
```

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.

```
{ 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.

```
                      {
  backupFile(selectBackupCode(1));
  menu();
}
```

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.
```
                                 {
  int cdBkp;

  if (type == 0) {
    std::cout « "Choose a backup to view changes: ";
  } else if (type == 1) {
    std::cout « "Choose a backup to restore: ";
  }

  std::cin » cdBkp;

  return cdBkp;
}
```

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.
```
                         {
  std::string str;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Search by code                           |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Enter the code: ";
  std::cin » str;
  // errorcheck (str)

  return str;
}
```

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.
```
                     {
  int flag = 0;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Uc Code                                  |" « std::endl;
  std::cout « "| 2) Class Code                               |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;

  std::cout « "Choose an option: ";
  std::cin » flag;
  errorCheck(flag);

  return flag;
}
```

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.

```
                           {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by student code asc                    |" « std::endl;
  std::cout « "| 2) Sort by student code desc                   |" « std::endl;
  std::cout « "| 3) Sort by student name asc                    |" « std::endl;
  std::cout « "| 4) Sort by student name desc                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - nº ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}
```

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.

```
                      {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by uc code asc                         |" « std::endl;
  std::cout « "| 2) Sort by uc code desc                        |" « std::endl;
  std::cout « "| 3) Sort by class code asc                      |" « std::endl;
  std::cout « "| 4) Sort by class code desc                     |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - nº ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}
```

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:
    menuRequests();
    break;
  case 3:
    menuBackup();
    break;
  case 4:
    exit(0);
  default:
    errorMessage();
    break;
  }
}

void menuSeeDatabase() {
  int flag = 0;
  int type;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) See Students                             |" « std::endl;
  std::cout « "| 2) See Classes and UC’s                     |" « std::endl;
  std::cout « "| 3) See My Schedules                         |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  if (flag != 3) {
    type = selectType();
  }
  // std::cout « type;

  if (type == 1) {
    std::string code = selectCode();
    switch (flag) {
    case 1:
      menuStudents(code, type);
      break;
    case 2:
      menuUcs(code, type);
      break;
    default:
      errorMessage();
      break;
    }
  } else {
    int filter;
    int order;
    std::string value;
    if (type == 2) {
      filter = selectFilter();
      value = selectValue();
    }
    switch (flag) {
    case 1:
      order = selectOrderStudents();
      menuStudents(value, type, filter, order);
      break;
    case 2:
      order = selectOrderUcs();
      menuUcs(value, type, filter, order);
      break;
    case 3:
      menuStudentCode(4);
      break;
    default:
      errorMessage();
      break;
    }
  }
}

void menuRequests() {
  int flag = 0;

  system("clear");
  std::cout « "Change database" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Add                                      |" « std::endl;
  std::cout « "| 2) Remove                                   |" « std::endl;
  std::cout « "| 3) Switch                                   |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Choose an option: ";
  std::cin » flag;

  if (flag > 4 || flag == 0) {
    errorMessage();
  } else {
    menuStudentCode(flag);
  }
}

void menuStudentCode(int flag) {
  std::string registrationNumber;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Enter your registration number: ";
  std::cin » registrationNumber;

  auto it = students.find(registrationNumber);

  if (it == students.end()) {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| Registration number not found               |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;

    std::cout « "| 1) Try again                                |" « std::endl;
    std::cout « "| 2) Exit                                     |" « std::endl;

    int flag2;

    std::cin » flag2;

    switch (flag2) {
    case 1:
      system("clear");
      menuStudentCode(flag);
      break;
    case 2:
      exit(0);
    default:
      errorMessage();
      break;
    }

    menuRequests();
  } else {
    // printStudentClasses(it);
  }

  switch (flag) {
  case (1):
    menuAdd(it);
    break;
  case (2):
    menuRemove(it);
    break;
  case (3):
    menuSwitch(it);
    break;
  case (4):
    printStudentSchedules(it, classes);
    break;
  default:
    errorMessage();
  }
}

void menuTryAgain(int menuType,
                  std::map<std::string, myStudent>::iterator &it) {
  int flag;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Try again                                |" « std::endl;
  std::cout « "| 2) Exit                                     |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cin » flag;

  switch (flag) {
  case 1:
    system("clear");
    if (menuType == 1) {
      menuAdd(it);
    } else if (menuType == 2) {
      menuRemove(it);
    } else if (menuType == 3) {
      menuSwitch(it);
    }
    break;
  case 2:
    exit(0);
  default:
    errorMessage();
    break;
  }
}
```

```

void menuRemove(std::map<std::string, myStudent>::iterator &it) {
  printStudentClasses(it);
  std::string ucCode;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Enter UC code to remove " « std::endl;
  std::cin » ucCode;
  std::cout « "-----------------------------------------------" « std::endl;

  bool remove = removeUcStudent(ucCode, it, stackAlter, count);

  if (remove) {
    printStudentClasses(it);
    std::cout « "\nRemovido com sucesso" « std::endl;
    saveOrReturn();
  } else {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "UC code not found" « std::endl;
    menuTryAgain(2, it);
  }
}

void menuAdd(std::map<std::string, myStudent>::iterator &it) {
  printStudentClasses(it);
  std::string ucCode;
  std::string classCode;
  bool check_class = false;

  // validates if the student is enrolled in more than 7 classes
  if (it->second.valideQtClasses()) {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « " You have already 7 classes" « std::endl;
  } else {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "Enter UC code to see all classes: " « std::endl;
    std::cin » ucCode;

    if (!verifyUcCode(ucCode, it)) {
      // checks if ucCode exists
      auto it_uc = ucs.find(ucCode);

      if (it_uc == ucs.end()) {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "UC code not found" « std::endl;
        menuTryAgain(1, it);

      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Uc. Code: " « it_uc->first « std::endl;

        printFreeClasses(ucCode, count);
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Enter class code to add: " « std::endl;
        std::cin » classCode;

        check_class = verifyClassCode(classCode, ucCode, count);

        if (check_class) {
          // validates that the class chosen by the student does not conflict
          // with the schedule of other classes
          bool validate = valideNewClass(ucCode, classCode, it, classes);

          if (!validate) {
            addClassStudent(ucCode, classCode, it, stackAlter);
            printStudentClasses(it);
            std::cout « "\nSucessfully added" « std::endl;

            saveOrReturn();
          }
        } else {
          std::cout « "-----------------------------------------------"
                    « std::endl;
          std::cout « "Class code not found" « std::endl;
          menuTryAgain(1, it);
        }
      }
    } else {
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "You are already enrolled in this UC" « std::endl;
      menuTryAgain(1, it);
    }
  }
```

```
}

void menuSwitch(std::map<std::string, myStudent>::iterator &it) {
  printStudentClasses(it);
  std::string ucCode, classCode;
  int flag;
  auto it_uc = ucs.begin();
  std::list<std::string> free_classes;
  bool validate = false;
  bool check_class = false;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Switch UC                                |" « std::endl;
  std::cout « "| 2) Switch Class                             |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cin » flag;

  switch (flag) {
  case (1):
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "Enter UC code to remove: " « std::endl;
    std::cin » ucCode;

    if (verifyUcCode(ucCode, it)) {

      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "Enter UC code to add: " « std::endl;
      std::cin » ucCode;

      it_uc = ucs.find(ucCode);

      if (it_uc != ucs.end()) {

        printFreeClasses(ucCode, count);

        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Enter class code to add: " « std::endl;
        std::cin » classCode;

        check_class = verifyClassCode(classCode, ucCode, count);

        if (check_class) {
          validate = valideNewClass(ucCode, classCode, it, classes);
          if (!validate) {
            removeUcStudent(ucCode, it, stackAlter, count);
            addClassStudent(ucCode, classCode, it, stackAlter);
            printStudentClasses(it);
            std::cout « "\nSuccessfully switched" « std::endl;
            saveOrReturn();
          }
        } else {
          std::cout « "-----------------------------------------------"
                    « std::endl;
          std::cout « "Class code not found" « std::endl;
          menuTryAgain(3, it);
        }
      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "UC code not found" « std::endl;
        menuTryAgain(3, it);
      }
    } else {
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "You are not enrolled in this UC" « std::endl;
      menuTryAgain(3, it);
    }

    break;
  case (2):
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "Enter UC to change class: " « std::endl;
    std::cin » ucCode;

    if (verifyUcCode(ucCode, it)) {

      printFreeClasses(ucCode, count);
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "Enter class code to add: " « std::endl;
      std::cin » classCode;

      check_class = verifyClassCode(classCode, ucCode, count);

      if (check_class) {
        removeUcStudent(ucCode, it, stackAlter, count);
        validate = valideNewClass(ucCode, classCode, it, classes);
        if (!validate) {
          addClassStudent(ucCode, classCode, it, stackAlter);
          printStudentClasses(it);
          std::cout « "\nSuccessfully switched" « std::endl;
          saveOrReturn();
        }
      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Class code not found" « std::endl;
        menuTryAgain(3, it);
      }
    } else {
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "You are not enrolled in this UC" « std::endl;
      menuTryAgain(3, it);
    }
    break;
  default:
    errorMessage();
    break;
  }
}

void saveOrReturn() {
  int flag = 0;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Save                                      |" « std::endl;
  std::cout « "| 2) Return                                    |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  switch (flag) {
  case 1:
    save();
    break;
  case 2:
    menuRequests();
    break;
  default:
    errorMessage();
    break;
  }
}

void save() {
  keepAllChanges(students, stackAlter);
  exit(0);
}

int selectBackupCode(int type) {
  int cdBkp;

  if (type == 0) {
    std::cout « "Choose a backup to view changes: ";
  } else if (type == 1) {
    std::cout « "Choose a backup to restore: ";
  }

  std::cin » cdBkp;

  return cdBkp;
}

void menuBackup() {
  int flag;
  system("clear");
  listAllBackups();

  bool valide = printAllBackups();
  if (valide == true) {
    printChanges(selectBackupCode(0));
    menuChanges();
  } else {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| 1) - Main menu                               |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;
    std::cin » flag;

    if (flag == 1) {
      menu();
    } else {
      errorMessage();
    }
  }
}

void restoreBackup() {
  backupFile(selectBackupCode(1));
  menu();
}

void menuChanges() {

  int flag;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Return                                    |" « std::endl;
  std::cout « "| 2) Main menu                                 |" « std::endl;
  std::cout « "| 3) Restore                                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;

  std::cin » flag;

  switch (flag) {
  case (1):
    menuBackup();
    break;
  case (2):
    menu();
    break;
  case (3):
    restoreBackup();
    break;
  default:
    errorMessage();
    break;
  }
}

int selectOrderStudents() {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by student code asc                  |" « std::endl;
  std::cout « "| 2) Sort by student code desc                 |" « std::endl;
  std::cout « "| 3) Sort by student name asc                  |" « std::endl;
  std::cout « "| 4) Sort by student name desc                 |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - n° ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}

int selectOrderUcs() {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by uc code asc                       |" « std::endl;
  std::cout « "| 2) Sort by uc code desc                      |" « std::endl;
  std::cout « "| 3) Sort by class code asc                    |" « std::endl;
  std::cout « "| 4) Sort by class code desc                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - n° ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}

int selectType() {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) See one                                   |" « std::endl;
  std::cout « "| 2) See a particular group                    |" « std::endl;
  std::cout « "| 3) See all                                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;

  std::cout « "Choose an option: ";
  std::cin » flag;
  errorCheck(flag);

  return flag;
}

std::string selectCode() {
  std::string str;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Search by code                            |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "Enter the code: ";
  std::cin » str;
  // errorcheck (str)

  return str;
}

int selectFilter() {
  int flag = 0;

  std::cout « "-----------------------------------------------" « std::endl;
  std::cout « "| 1) Uc Code                                   |" « std::endl;
  std::cout « "| 2) Class Code                                |" « std::endl;
  std::cout « "-----------------------------------------------" « std::endl;

  std::cout « "Choose an option: ";
  std::cin » flag;
  errorCheck(flag);

  return flag;
}

std::string selectValue() {
  std::string str;

  std::cout « "Enter the value: ";
  std::cin » str;
  // errorcheck (str)

  return str;
}

void menuStudents(std::string str, int type, int filter, int order) {
  std::map<std::string, myStudent> oneStudent = students;
  std::vector<myStudent> data;

  for (const auto &studentPair : students) {
    data.push_back(studentPair.second);
  }

  if (type == 1) {
    oneStudent = selectStudent(str, oneStudent);
    printStudent(oneStudent);
  } else {
    if (type == 2) {
      data = filterInfoStudent(filter, str, data);
    }
    data = orderInfoStudent(order, data);
    printStudents(data);
  }
}


void menuUcs(std::string str, int type, int filter, int order) {
  std::vector<myUc> data;
  std::vector<myUc> oneUc;

  for (const auto &ucVectorPair : ucs) {
    for (const myUc &ucObj : ucVectorPair.second) {
      data.push_back(ucObj);
    }
  }

  if (type == 1) {
    oneUc = selectUc(str, classes);
    printUcClasses(oneUc);
  } else {
    if (type == 2) {
      data = filterInfoUc(filter, str, data);
    }
    data = orderInfoUc(order, data);
    printUcs(data);
  }
}
```

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.

```
                        {
  if (n == 0) {
    std::cout « "ERROR: Invalid number" « std::endl;
    exit(0);
  }
}
```

4.35.1.2 errorMessage()

```
void errorMessage ( )
```

Definition at line 4 of file errorMsgs.cpp.

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

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.

```
             {

  menuUpdate();
  system("clear");

  int flag = 0;

  std::cout « "------------ Welcome to our app :) ------------" « std::endl;
  std::cout « "| 1) See database                             |" « std::endl;
  std::cout « "| 2) Change database                          |" « std::endl;
  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;
```

```
        std::cin » classCode;

        check_class = verifyClassCode(classCode, ucCode, count);

        if (check_class) {
          // validates that the class chosen by the student does not conflict
          // with the schedule of other classes
          bool validate = valideNewClass(ucCode, classCode, it, classes);

          if (!validate) {
            addClassStudent(ucCode, classCode, it, stackAlter);
            printStudentClasses(it);
            std::cout « "\nSucessfully added" « std::endl;

            saveOrReturn();
          }
        } else {
          std::cout « "-----------------------------------------------"
                    « std::endl;
          std::cout « "Class code not found" « std::endl;
          menuTryAgain(1, it);
        }
      }
    } else {
      std::cout « "-----------------------------------------------"
                « std::endl;
      std::cout « "You are already enrolled in this UC" « std::endl;
      menuTryAgain(1, it);
    }
  }
}
```

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.

```
                    {
  int flag;
  system("clear");
  listAllBackups();

  bool valide = printAllBackups();
  if (valide == true) {
    printChanges(selectBackupCode(0));
    menuChanges();
  } else {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| 1) - Main menu                              |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;
    std::cin » flag;

    if (flag == 1) {
      menu();
    } else {
      errorMessage();
    }
  }
}
```

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.
```
                       {

  int flag;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Return                                     |" « std::endl;
  std::cout « "| 2) Main menu                                  |" « std::endl;
  std::cout « "| 3) Restore                                    |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;

  std::cin » flag;

  switch (flag) {
  case (1):
    menuBackup();
    break;
  case (2):
    menu();
    break;
  case (3):
    restoreBackup();
    break;
  default:
    errorMessage();
    break;
  }
}
```

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.
```
                                                                           {
  printStudentClasses(it);
  std::string ucCode;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "Enter UC code to remove " « std::endl;
  std::cin » ucCode;
  std::cout « "------------------------------------------------" « std::endl;

  bool remove = removeUcStudent(ucCode, it, stackAlter, count);

  if (remove) {
    printStudentClasses(it);
    std::cout « "\nRemovido com sucesso" « std::endl;
    saveOrReturn();
  } else {
    std::cout « "------------------------------------------------" « std::endl;
    std::cout « "UC code not found" « std::endl;
    menuTryAgain(2, it);
  }
}
```

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) {
    case 1:
      order = selectOrderStudents();
      menuStudents(value, type, filter, order);
      break;
    case 2:
      order = selectOrderUcs();
      menuUcs(value, type, filter, order);
      break;
    case 3:
      menuStudentCode(4);
      break;
    default:
      errorMessage();
      break;
    }
  }
}
```

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.

```
                                {
  std::string registrationNumber;
  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "Enter your registration number: ";
  std::cin » registrationNumber;

  auto it = students.find(registrationNumber);

  if (it == students.end()) {
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| Registration number not found               |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;

    std::cout « "| 1) Try again                                |" « std::endl;
    std::cout « "| 2) Exit                                     |" « std::endl;

    int flag2;

    std::cin » flag2;

    switch (flag2) {
    case 1:
      system("clear");
      menuStudentCode(flag);
      break;
    case 2:
      exit(0);
    default:
      errorMessage();
      break;
    }

    menuRequests();
  } else {
    // printStudentClasses(it);
  }

  switch (flag) {
  case (1):
    menuAdd(it);
    break;
  case (2):
    menuRemove(it);
    break;
  case (3):
    menuSwitch(it);
    break;
  case (4):
    printStudentSchedules(it, classes);
    break;
  default:
    errorMessage();
  }
}
```

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.

```
                                                                    {
  std::map<std::string, myStudent> oneStudent = students;
  std::vector<myStudent> data;

  for (const auto &studentPair : students) {
    data.push_back(studentPair.second);
  }

  if (type == 1) {
    oneStudent = selectStudent(str, oneStudent);
    printStudent(oneStudent);
  } else {
    if (type == 2) {
      data = filterInfoStudent(filter, str, data);
    }
    data = orderInfoStudent(order, data);
    printStudents(data);
  }
}
```

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.

```
                                                                    {
    printStudentClasses(it);
    std::string ucCode, classCode;
    int flag;
    auto it_uc = ucs.begin();
    std::list<std::string> free_classes;
    bool validate = false;
    bool check_class = false;

    std::cout « "------------------------------------------------" « std::endl;
    std::cout « "| 1) Switch UC                                 |" « std::endl;
    std::cout « "| 2) Switch Class                              |" « std::endl;
    std::cout « "------------------------------------------------" « std::endl;
    std::cin » flag;

    switch (flag) {
    case (1):
      std::cout « "------------------------------------------------" « std::endl;
      std::cout « "Enter UC code to remove: " « std::endl;
      std::cin » ucCode;

      if (verifyUcCode(ucCode, it)) {

        std::cout « "------------------------------------------------"
                  « std::endl;
        std::cout « "Enter UC code to add: " « std::endl;
        std::cin » ucCode;

        it_uc = ucs.find(ucCode);

        if (it_uc != ucs.end()) {

          printFreeClasses(ucCode, count);

          std::cout « "------------------------------------------------"
                    « std::endl;
          std::cout « "Enter class code to add: " « std::endl;
          std::cin » classCode;

          check_class = verifyClassCode(classCode, ucCode, count);

          if (check_class) {
            validate = valideNewClass(ucCode, classCode, it, classes);
            if (!validate) {
              removeUcStudent(ucCode, it, stackAlter, count);
              addClassStudent(ucCode, classCode, it, stackAlter);
              printStudentClasses(it);
              std::cout « "\nSuccessfully switched" « std::endl;
              saveOrReturn();
            }
          } else {
            std::cout « "------------------------------------------------"
                      « std::endl;
            std::cout « "Class code not found" « std::endl;
            menuTryAgain(3, it);
          }
        } else {
          std::cout « "------------------------------------------------"
                    « std::endl;
          std::cout « "UC code not found" « std::endl;
          menuTryAgain(3, it);
        }
      } else {
        std::cout « "------------------------------------------------"
                  « std::endl;
        std::cout « "You are not enrolled in this UC" « std::endl;
        menuTryAgain(3, it);
      }

      break;
    case (2):
      std::cout « "------------------------------------------------" « std::endl;
      std::cout « "Enter UC to change class: " « std::endl;
      std::cin » ucCode;

      if (verifyUcCode(ucCode, it)) {

        printFreeClasses(ucCode, count);
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "Enter class code to add: " « std::endl;
        std::cin » classCode;

        check_class = verifyClassCode(classCode, ucCode, count);

        if (check_class) {
          removeUcStudent(ucCode, it, stackAlter, count);
          validate = valideNewClass(ucCode, classCode, it, classes);
          if (!validate) {
            addClassStudent(ucCode, classCode, it, stackAlter);
            printStudentClasses(it);
            std::cout « "\nSuccessfully switched" « std::endl;
            saveOrReturn();
          }
        } else {
          std::cout « "-----------------------------------------------"
                    « std::endl;
          std::cout « "Class code not found" « std::endl;
          menuTryAgain(3, it);
        }
      } else {
        std::cout « "-----------------------------------------------"
                  « std::endl;
        std::cout « "You are not enrolled in this UC" « std::endl;
        menuTryAgain(3, it);
      }
      break;
    default:
      errorMessage();
      break;
    }
  }
```

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.

```
                                                                  {
  int flag;
  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Try again                                  |" « std::endl;
  std::cout « "| 2) Exit                                       |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  std::cin » flag;

  switch (flag) {
  case 1:
    system("clear");
    if (menuType == 1) {
      menuAdd(it);
    } else if (menuType == 2) {
      menuRemove(it);
    } else if (menuType == 3) {
      menuSwitch(it);
    }
    break;
  case 2:
    exit(0);
  default:
    errorMessage();
    break;
  }
}
```

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.

```
                                                            {
  std::vector<myUc> data;
  std::vector<myUc> oneUc;

  for (const auto &ucVectorPair : ucs) {
    for (const myUc &ucObj : ucVectorPair.second) {
      data.push_back(ucObj);
    }
  }

  if (type == 1) {
    oneUc = selectUc(str, classes);
    printUcClasses(oneUc);
  } else {
    if (type == 2) {
      data = filterInfoUc(filter, str, data);
    }
    data = orderInfoUc(order, data);
    printUcs(data);
  }
}
```

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.

```
                        {
  backupFile(selectBackupCode(1));
  menu();
}
```

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.

```
                          {
    std::string str;
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| 1) Search by code                           |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "Enter the code: ";
    std::cin » str;
    // errorcheck (str)

    return str;
}
```

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.

```
                      {
    int flag = 0;

    std::cout « "-----------------------------------------------" « std::endl;
    std::cout « "| 1) Uc Code                                  |" « std::endl;
    std::cout « "| 2) Class Code                               |" « std::endl;
    std::cout « "-----------------------------------------------" « std::endl;

    std::cout « "Choose an option: ";
    std::cin » flag;
    errorCheck(flag);

    return flag;
}
```

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.

```
                        {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by student code asc                  |" « std::endl;
  std::cout « "| 2) Sort by student code desc                 |" « std::endl;
  std::cout « "| 3) Sort by student name asc                  |" « std::endl;
  std::cout « "| 4) Sort by student name desc                 |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - nº ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}
```

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.

```
                   {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) Sort by uc code asc                       |" « std::endl;
  std::cout « "| 2) Sort by uc code desc                      |" « std::endl;
  std::cout « "| 3) Sort by class code asc                    |" « std::endl;
  std::cout « "| 4) Sort by class code desc                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;
  // add more order like - nº ucs,
  std::cout « "Choose an option: ";
  std::cin » flag;

  errorCheck(flag);

  return flag;
}
```

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.

```
                    {
  int flag = 0;

  std::cout « "------------------------------------------------" « std::endl;
  std::cout « "| 1) See one                                   |" « std::endl;
  std::cout « "| 2) See a particular group                    |" « std::endl;
  std::cout « "| 3) See all                                   |" « std::endl;
  std::cout « "------------------------------------------------" « std::endl;

  std::cout « "Choose an option: ";
  std::cin » flag;
  errorCheck(flag);

  return flag;
}
```

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.

```
                           {
  std::string str;

  std::cout « "Enter the value: ";
  std::cin » str;
  // errorcheck (str)

  return str;
}
```

4.36 menu.h

Go to the documentation of this file.

```
#ifndef MENU_H
#define MENU_H

#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"

void errorMessage();
void errorCheck(int n);

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

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

void menu();
void menuSeeDatabase();
void menuRequests();
void menuRemove(std::map<std::string, myStudent>::iterator &it);
void menuAdd(std::map<std::string, myStudent>::iterator &it);
void menuSwitch(std::map<std::string, myStudent>::iterator &it);

void menuBackup();
void menuChanges();
void restoreBackup();
int selectBackupCode();

int selectOrderStudents();
int selectOrderUcs();
int selectType();
int selectFilter();
std::string selectCode();
std::string selectValue();

void saveOrReturn();
void save();

#endif
```