Lab(Student templated Class)

ESE 3005: EMBEDDED SYSTEMS ARCHITECTURE II

Lambton College in Toronto

Instructor: Takis Zourntos

STUDENT NAME & ID:

Vishal Hasrajani(C0761544)

Parth Patel(C0764929)

Goutham Reddy Alugubelly(C0747981)

Ratnajahnavi rebbapragada(C0762196)

INTRODUCTION :

In this project, we will use the different features of c++ including associative containers, iterators as well as the templated and regular classes.Here we have created a Student class that will align with the data provided using the text file(i.e Student_data.txt).We will display, add and delete the data from the database using the different functions.

We have created 2 projects in order to show the functionality like add(), delete() and display().

DESCRIPTION :

To start with, we will explain different parts of the code to clarify what is going on in this project.

In **students.h,** we have defined the structure for storing the data of a student.Data includes students ID, last name , first name, number of courses, courses and grade

```
typedef struct
{
     std::string key; // primary key/ID; unique and must be
included
     std::string ID;
     std::string last_name;
     std::string first_name;
     long num_courses;
     std::vector<std::string> courses;
     std::vector<double> grades;
```

```
} record_t;
```

Here we have defined a vector to store the courses and grades.Vectors are the dynamic arrays which can resize itself automatically on addition or deletion of any element.

Then we declared the Student class which includes different methods like get(),put(), end and a default constructor.

```
class Student
{
public:
     Student(void); // constructor
     // data on the student
     record_t data;
     int get(); // get the data from standard input and
keygen();
     void put() const; // send the record to standard
output;
     bool end(); // checks if record if student record is
the last one, updates last_record

private:
     std::string keygen(std::string, std::string);
};
```

keygen() is a key generator function which is kept private so that no one can change its data.

Now we will try to explain each function one by one in **students.c**

1. **Student::Student(void)**

```
Student::Student(void)
{
}
```

This is a student constructor which initializes the object .

2. **bool Student::end()**

```
bool Student::end()
{
     if (data.ID == END_MARKER)
     {
          return true;
     }
     return false;
}
```

This function will check whether the provided data in the text file has reached the END_MARKER(i.e. -9999) or not . If data.ID == END_MARKER then it will return true.Else it will return 0.

3. **int Student::get(void)**

```
int Student::get(void)
{
     long Nc; // stores the number of courses
```

```
    // get the student ID number (NOTE: this is not the
same as the "key" information)
    std::cin >> data.ID;

    // get the last name
    std::cin >> data.last_name;

    // get the first name
    std::cin >> data.first_name;

    // get the student courses
    std::cin >> data.num_courses;
    Nc = data.num_courses;

    // get the course names and the grades
    std::string course_name;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> course_name;
        Student::data.courses.push_back(course_name);
    }
    double grade;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> grade;
        data.grades.push_back(grade);
    }

    // generate a unique key
    data.key = keygen(data.ID, data.last_name);

    return 0; // in the future, can add error checking
}
```

So here we are taking the input from the standard output and storing it into the defined variables and vectors. Push_back is a method to push the data into the vector.

At last we are generating the key using the keygen() function. key is the combination of ID and Last name.

4. **void Student::put(void) const**

This is the function used to send the data to the standard output

```
void Student::put(void) const
{
     long Nc = data.num_courses;

     // put key
     std::cout << "key: " << data.key << std::endl;

     // put the student ID number (NOTE: this is not the
same as the "key" information)
     std::cout << "ID: " << data.ID << std::endl;

     // put the last name
     std::cout << "last name: " << data.last_name <<
std::endl;

     // put the first name
     std::cout << "first name: " << data.first_name <<
std::endl;

     // put  the student courses
     std::cout << "number of courses: " << Nc << std::endl;
```

```
    // put the course names and the grades



    std::string course_name;
    double grade;
    for (long j = 0; j != Nc; ++j)
    {
        course_name = data.courses[j];
        grade = data.grades[j];
        std::cout << "\t" << course_name << ":\t" << grade
<< std::endl;
    }

    // line spacing
    std::cout << std::endl << std::endl;
}
```

5. **std::string Student::keygen(std::string A, std::string B)**

```
std::string Student::keygen(std::string A, std::string B)
{
    std::string ret = A + "_" + B;

    return ret;
}
```

This function will add “_” between the ID and the Last name and will return the string.

Now coming to the templated classes . Templated classes can take any type of data defined by the user.

Let's start with **PROJ_CLASSES_H**

This is class prototype

```
/*************************************************************
*
 *            CLASS PROTOTYPE

*************************************************************
/
template<class T> class DB
{
public: // this is the interface to the class
     DB(void); // constructor
     std::size_t numelements() const;
     int add(void); // adds an element referenced in the
argument
     int del(void); // deletes an element specified by key
argument
     void display() const; // displays all the records in
the map

private: // these are needed by other methods of the class,
but not needed externally
     std::size_t DBi; // number of database elements
     std::map<std::string, T> DBmap; // our associative
container
};
```

Here we have created the **templated class T.** There are different methods declared in this class prototype which includes add(), del(), display().

Here map is the associative container which has a key-value pair to store.

Lets try to explain different methods in the source code of the templated class.

1) `template<class T>` **DB<T>::DB(void) :DBi(0)**

DB::constructor, default constructor, initializes storage

```
template<class T>
DB<T>::DB(void) :
          DBi(0)
{
     T record;
     record.get(); // retrieve a single record from
standard input
     while (!record.end())
     {
          DBmap[record.data.key] = record;
          record.get(); // retrieve a single record from
standard input
     }
     DBi = DBmap.size();
}
```

Initially DBi will be set to 0.

Then,It retrieves a single record from the standard input then it checks if it is not the end of the record (i.e -9999 or END_MARKER). If it's not the end of the list, then add the record to the DBmap .And it keeps on adding the records to the map until it reaches the END_MARKER.

DBi will store the size of the map .

.

.

.

2) `template<class T>` **std::size_t DB<T>::numelements() const:**

```
template<class T>
std::size_t DB<T>::numelements() const
{
     return DBi;
}
```

This function will return the number of elements in the database.

3) `template<class T>` **int DB<T>::add(void)**

```
// DB::add() method
template<class T>
int DB<T>::add(void)
{
     T token;
     // retrieve new element from standard input
     token.get();
     std::string key = token.data.key; // find the key for
this token

     // add the new element to the database
     DBmap[key] = token; // add this element to the map

     // check if the element can be found
     if ( (DBmap.find(key)->first) == (token.data.key) )
     {
          ++DBi;
```

```
          return 0;
     }
     else
          return 1; // error: element not added to database
}
```

This function adds the element to the map of the database.It will get the data then it will add the data to the map using the key and the token given to it.Here in the map, we can get the key using first and the second will give data stored.

So here we will check if the key matches the given key. If it matches, then increment the value of DBi(i.e. Number of elements in database).

3) template<class T>int DB<T>::del(void)

```
// DB::del() method
template<class T>
int DB<T>::del(void)
{
     std::string key="Z129347_Johandas";


     auto iter = DBmap.find(key); // auto infers variable
type from initializer type
     if (iter != DBmap.end())
     {
          // element is found
```

```
            DBmap.erase(iter);



            --DBi;
            return 0;
        }
        return 1; // error: element is not found
}
```

This function deletes the data from the database.It takes the key as an input and then finds the key in the map using the iterator.Once it finds the key, it will delete that record from the database and decrements the value of DBi.

4) `template<class T>` void DB<T>::display() const

```
// DB::display() method
template<class T>
void DB<T>::display() const
{
        T temp;
        auto c_iter = DBmap.begin(); // instead of:
std::map<std::string, T>::iterator c_iter
        while (c_iter != DBmap.end())
        {
            temp = c_iter->second; // c_iter->first
corresponds to the key, c_iter->second corresponds to the
value
            temp.put(); // writes the DBmap value to standard
```

```
output
          ++c_iter;
     }
}
```

This function will display all the records in the database.At first it checks if its end of the data .If it is not the end, then it uses the put method to display the record to the user(pointing to the second ,which is data)

So now we will try to Display, delete and add the record from/to the database.

proj_class_expt_templ.cpp

```
#include <iostream>
#include "proj_classes.h"   // main templated class
#include "students.h"       // Student class
#include <string>

using namespace std;
// only use "using namespace" in your main code file



int main()
{
     // declare a student-database object, load from
standard input
     DB<Student> myStudentDB;

     // experiment with adding and deleting object from
database
```

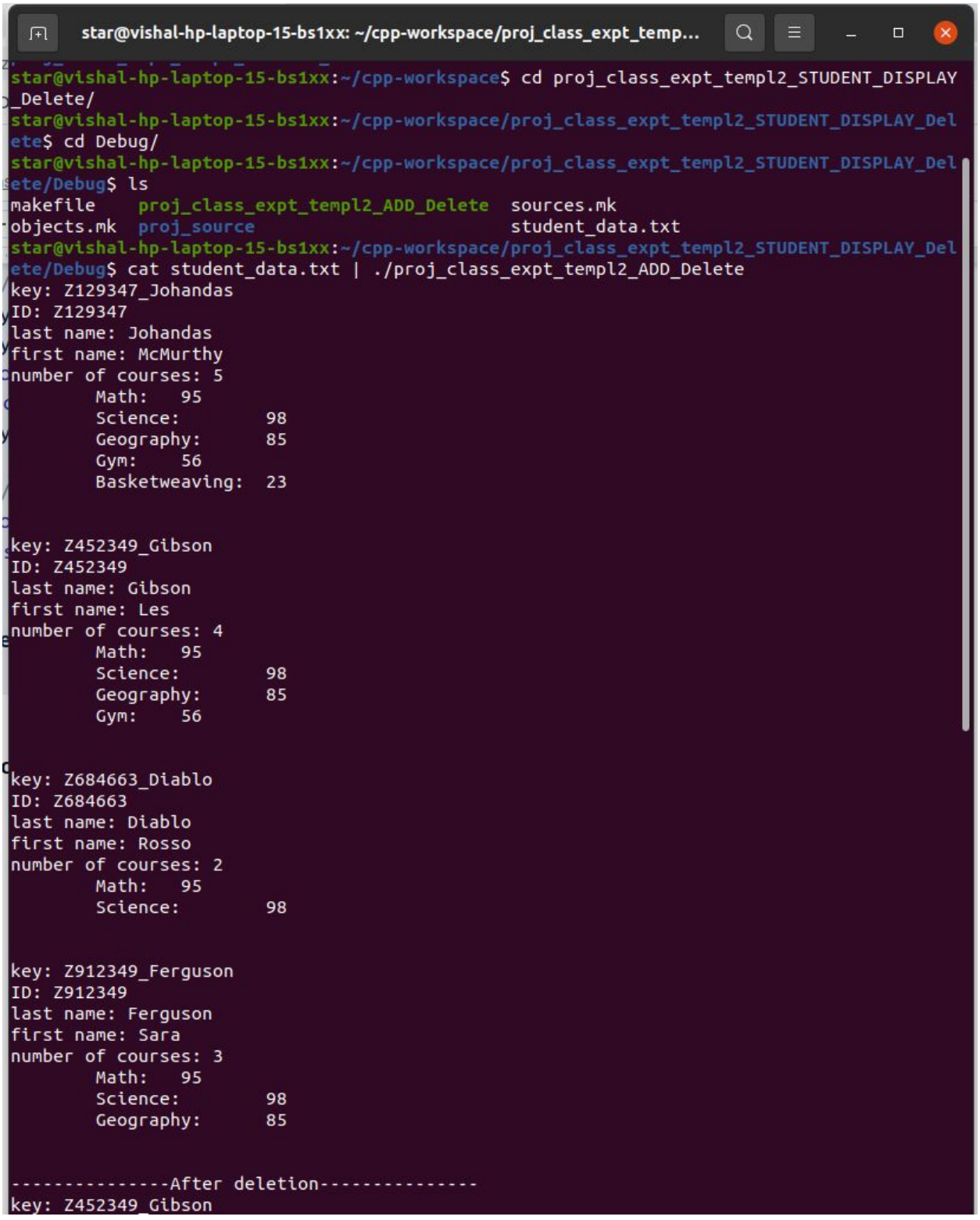
```
    // your code goes here

    // print out the data
    myStudentDB.display();
    myStudentDB.del();
    cout<< "---------------After
deletion---------------"<<endl;
    myStudentDB.display();

    // exit
    cout << endl << myStudentDB.numelements() << "
processed." << endl;


    return 0;
}
```

Text file---> student_data.txt

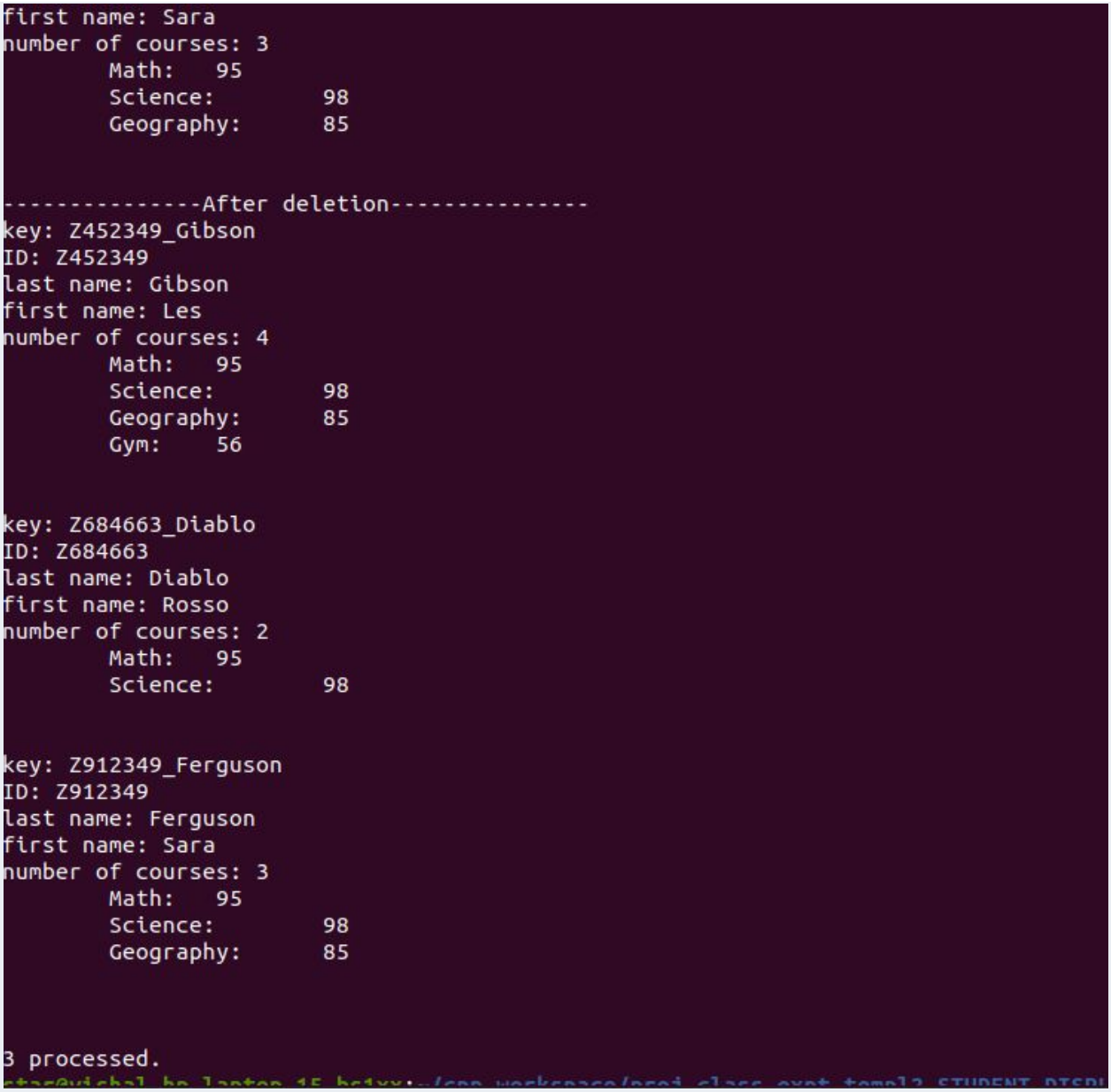
```
|
|
|
|
|
|
|
|
|
.
.
.
.
.
```

```
1
2 Z912349
3 Ferguson
4 Sara
5 3
6 Math    Science         Geography
7 95      98      85
8
9 Z129347
10 Johandas
11 McMurthy
12 5
13 Gym     Basketweaving   Poohing         Burping         Stopping
14 56      23      99      89      91
15
16 Z452349
17 Gibson
18 Les
19 4
20 Math    Science         Geography       Music
21 76      67              60              99
22
23 Z684663
24 Diablo
25 Rosso
26 2
27 Art     Farting
28 95      98
29
30 -9999
31 dummy
32 dummy
33 1
34 nonsense
35 0
```

Output of display () and del()



I have given the key (i.e. `std::string key="Z129347_Johandas";`)
So it will delete the record of `Z129347_Johandas.`

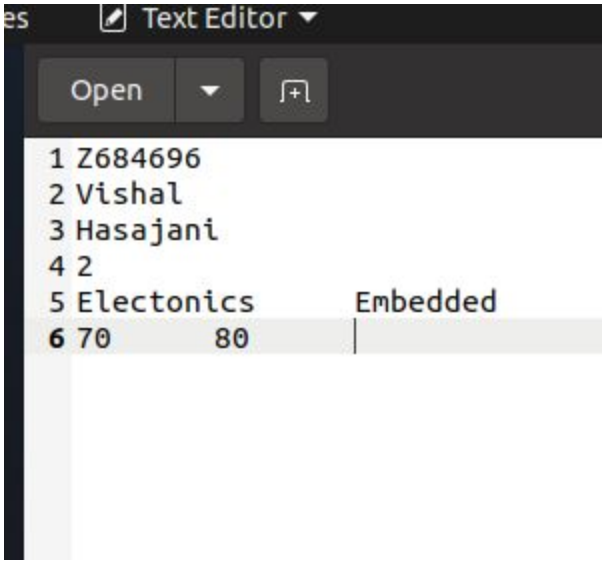


For adding the data to the database i have made the separate project with few changes in the templated class.h file

I have kept the default constructor empty for adding the element and I am using a separate text file (i.e. student_1_entry.txt)to add the data.
Default constructor for new project

```
template<class T>
DB<T>::DB(void) :
          DBi(0)
{
}
```

student_1_entry.txt

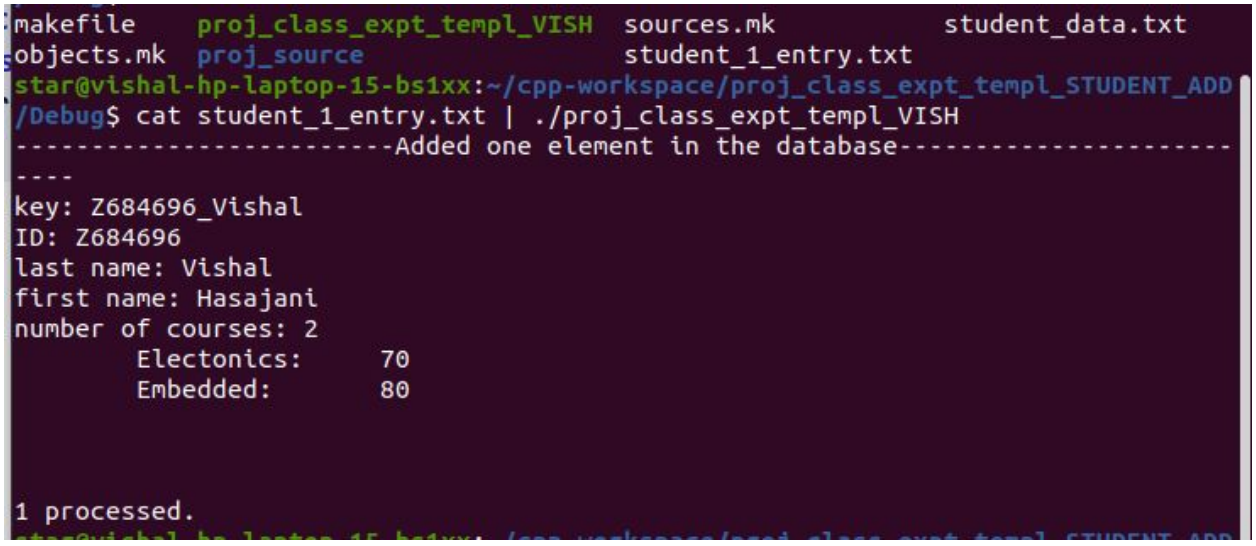


proj_class_expt_templ.cpp

```
#include <iostream>
#include "proj_classes.h"   // main templated class
#include "students.h"       // Student class
#include <string>
```

```
using namespace std;
// only use "using namespace" in your main code file
int main()
{
     // declare a student-database object, load from
standard input
     DB<Student> myStudentDB;
     cout<< "-------------------------Added one element in
the database--------------------------"<<endl;
     myStudentDB.add(); //adding element
     myStudentDB.display(); // displaying records
     // exit
     cout << endl << myStudentDB.numelements() << "
processed." << endl;
     return 0;
}
```

Output :



CONCLUSION:

Overall , we got to know about templated classes,regular classes ,vectors and maps in c++. It was a great pleasure to complete this project by understanding these advanced topics in c++.

APPENDIX :

Project 1:(For Display and Deletion)
students.h

```
/*
 * students.h
 *
 *  Created on: Dec. 2, 2020
 *      Author: takis
 */

#ifndef STUDENTS_H_
#define STUDENTS_H_

#define END_MARKER "-9999"

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

typedef struct
{
     std::string key; // primary key/ID; unique and must be included
     std::string ID;
     std::string last_name;
     std::string first_name;
     long num_courses;
```

```
    std::vector<std::string> courses;
    std::vector<double> grades;
} record_t;

/*
 * this defines a Student object
 */
class Student
{
public:
    Student(void); // constructor
    // data on the student
    record_t data;
    int get(); // get the data from standard input and
keygen();
    void put() const; // send the record to standard
output;
    bool end(); // checks if record if student record is
the last one, updates last_record

private:
    std::string keygen(std::string, std::string);
};

#endif /* STUDENTS_H_ */
```

students.cpp

```
/*
 * students.cpp
 *
 *  Created on: Dec. 2, 2020
 *      Author: takis
```

```
 */

#include <iostream>
#include <string>
#include <vector>
#include "students.h"

// Student constructor, initializes the object
Student::Student(void)
{
}

// Student::end() method
bool Student::end()
{
     if (data.ID == END_MARKER)
     {
          return true;
     }
     return false;
}

// Student::get() method, populates the data record from
standard input
int Student::get(void)
{
     long Nc; // stores the number of courses

     // get the student ID number (NOTE: this is not the
same as the "key" information)
     std::cin >> data.ID;

     // get the last name
```

```
    std::cin >> data.last_name;

    // get the first name
    std::cin >> data.first_name;

    // get the student courses
    std::cin >> data.num_courses;
    Nc = data.num_courses;
```

```
    // get the course names and the grades
    std::string course_name;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> course_name;
        Student::data.courses.push_back(course_name);
    }
    double grade;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> grade;
        data.grades.push_back(grade);
    }

    // generate a unique key
    data.key = keygen(data.ID, data.last_name);

    return 0; // in the future, can add error checking
}

// Student::put() method, sends data contents to standard
output
void Student::put(void) const
{
```

```
     long Nc = data.num_courses;

     // put key
     std::cout << "key: " << data.key << std::endl;

     // put the student ID number (NOTE: this is not the
same as the "key" information)
     std::cout << "ID: " << data.ID << std::endl;

     // put the last name
     std::cout << "last name: " << data.last_name <<
std::endl;

     // put the first name
     std::cout << "first name: " << data.first_name <<
std::endl;

     // put  the student courses
     std::cout << "number of courses: " << Nc << std::endl;

     // put the course names and the grades
     std::string course_name;
     double grade;
     for (long j = 0; j != Nc; ++j)
     {
          course_name = data.courses[j];
          grade = data.grades[j];
          std::cout << "\t" << course_name << ":\t" << grade
<< std::endl;
     }

     // line spacing
     std::cout << std::endl << std::endl;
```

```
}

// Student::keygen() method
std::string Student::keygen(std::string A, std::string B)
{
     std::string ret = A + "_" + B;

     return ret;
}
```

```
```

proj_classes.h

```
/*
 * proj_classes.h
 *
 *  Created on: Dec. 11, 2020
 *      Author: takis
 */


#ifndef PROJ_CLASSES_H_
#define PROJ_CLASSES_H_

#include <iostream>
#include <string>
#include <map>

/*************************************************************
*
 *            CLASS PROTOTYPE
```

```
**************************************************************
/
template<class T> class DB
{
public: // this is the interface to the class
     DB(void); // constructor
     std::size_t numelements() const;
     int add(void); // adds an element referenced in the
argument
     int del(void); // deletes an element specified by key
argument
     void display() const; // displays all the records in
the map

private: // these are needed by other methods of the class,
but not needed externally
     std::size_t DBi; // number of database elements
     std::map<std::string, T> DBmap; // our associative
container
};

/**************************************************************
*
 *             CLASS SOURCE CODE

**************************************************************
/

// DB::constructor, default constructor, initializes
storage
template<class T>
DB<T>::DB(void) :
```

```
          DBi(0)
{
     T record;
     record.get(); // retrieve a single record from
standard input
     while (!record.end())
     {
          DBmap[record.data.key] = record;
          record.get(); // retrieve a single record from
standard input
     }
     DBi = DBmap.size();
}

// DB::numelements()
template<class T>
std::size_t DB<T>::numelements() const
{
     return DBi;
}



// DB::del() method
template<class T>
int DB<T>::del(void)
{
     std::string key="Z129347_Johandas";


     auto iter = DBmap.find(key); // auto infers variable
type from initializer type
     if (iter != DBmap.end())
```

```
    {
        // element is found
        DBmap.erase(iter);
        --DBi;
        return 0;
    }
    return 1; // error: element is not found
}


// DB::display() method
template<class T>
void DB<T>::display() const
{
    T temp;
    auto c_iter = DBmap.begin(); // instead of:
std::map<std::string, T>::iterator c_iter
    while (c_iter != DBmap.end())
    {
        temp = c_iter->second; // c_iter->first
corresponds to the key, c_iter->second corresponds to the
value
        temp.put(); // writes the DBmap value to standard
output
        ++c_iter;
    }
}

#endif /* PROJ_CLASSES_H_ */
```

proj_class_expt_templ.cpp

```
//=============================================================
```

```
====================
// Name        : proj_class_expt_templ.cpp
// Author      : takis
// Version     :
// Copyright   : copyright (C) 2020 emad studio inc.
// Description : code to experiment with templated classes
//============================================================
====================

#include <iostream>
#include "proj_classes.h"   // main templated class
#include "students.h"       // Student class
//#include "toys.h"              // Toys class
#include <string>

using namespace std;
// only use "using namespace" in your main code file



int main()
{
     // declare a student-database object, load from
standard input
     DB<Student> myStudentDB;



     // print out the data
     myStudentDB.display(); //display all the records
     myStudentDB.del();// delete one record
     cout<< "---------------After
deletion---------------"<<endl;
```

```
    myStudentDB.display(); //display again

    // exit
    cout << endl << myStudentDB.numelements() << "
processed." << endl;


    return 0;
}
```

Project 2 : (Adding the elements to database)

Changes in 2 files (that are **proj_class_expt_templ.cpp** and **proj_classes.h**)

proj_class_expt_templ.cpp

```
//============================================================
==================
// Name        : proj_class_expt_templ.cpp
// Author      : takis
// Version     :
// Copyright   : copyright (C) 2020 emad studio inc.
// Description : code to experiment with templated classes
//============================================================
==================

#include <iostream>
#include "proj_classes.h"   // main templated class
#include "students.h"       // Student class
//#include "toys.h"             // Toys class
#include <string>
```

```
using namespace std;
// only use "using namespace" in your main code file


int main()
{
     // declare a student-database object, load from
standard input
     DB<Student> myStudentDB;



     // print out the data

     cout<< "-------------------------Added one element in
the database--------------------------"<<endl;
     myStudentDB.add(); //adding the record
     myStudentDB.display(); //display the record in
database

     // exit
     cout << endl << myStudentDB.numelements() << "
processed." << endl;
     return 0;
}
```

proj_classes.h

```
/*
 * proj_classes.h
 *
```

```
 *  Created on: Dec. 11, 2020
 *      Author: takis
 */


#ifndef PROJ_CLASSES_H_
#define PROJ_CLASSES_H_

#include <iostream>
#include <string>
#include <map>

/*************************************************************
*
 *            CLASS PROTOTYPE

*************************************************************
/
template<class T> class DB
{
public: // this is the interface to the class
     DB(void); // constructor
     std::size_t numelements() const;
     int add(void); // adds an element referenced in the
argument
     void display() const; // displays all the records in
the map

private: // these are needed by other methods of the class,
but not needed externally
     std::size_t DBi; // number of database elements
     std::map<std::string, T> DBmap; // our associative
container
```

```
};

/*************************************************************
*
 *            CLASS SOURCE CODE

*************************************************************
/

// DB::constructor, default constructor, initializes
storage
template<class T>
DB<T>::DB(void) :
          DBi(0)
{

}

// DB::numelements()
template<class T>
std::size_t DB<T>::numelements() const
{
     return DBi;
}

// DB::add() method
template<class T>
int DB<T>::add(void)
{
     T token;
     // retrieve new element from standard input
     token.get();
     std::string key = token.data.key; // find the key for
```

```
this token

     // add the new element to the database
     DBmap[key] = token; // add this element to the map

     // check if the element can be found
     if ( (DBmap.find(key)->first) == (token.data.key) )
     {
          ++DBi;
          return 0;
     }
     else
          return 1; // error: element not added to database
}



}

// DB::display() method
template<class T>
void DB<T>::display() const
{
     T temp;
     auto c_iter = DBmap.begin(); // instead of:
std::map<std::string, T>::iterator c_iter
     while (c_iter != DBmap.end())
     {
          temp = c_iter->second; // c_iter->first
corresponds to the key, c_iter->second corresponds to the
value
          temp.put(); // writes the DBmap value to standard
output
          ++c_iter;
```

```
    }
}
#endif /* PROJ_CLASSES_H_ */
```

Then rest 2 **students.cpp** and **student.h** remains the same as Project 1.