Lab(Toy 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 Toy class that will align with the data provided using the text file(i.e Toy_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 **toys.h,** we have defined the structure for storing the data of a Toy.Data includes manufacturing date,price, toy_name,category,rating and room.

```
typedef struct
{
     std::string key; // primary key
     std::string manuf_date;
     float price;
     std::string toy_name;
     std::vector<std::string> category;
     std::vector<double> ratings;
     std::string room;
} 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 Toy
{
public:
     Student(void); // constructor
     // data of the toy
     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 **toys.c**

1. **Toy::Toy(void)**

```
Toy::Toy(void)
{
}
```

This is a Toy constructor which initializes the object .

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

```
bool Toy::end()
{
     if (data.manuf_date == 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.manuf_date == END_MARKER then it will return true.Else it will return 0.

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

```
int Toy::get(void)
{
     long Nc=3; // stores the number of categories

     // get the Toy manufacturing date
```

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

    // get the price
    std::cin >> data.price;

    // get the toy name
    std::cin >> data.toy_name;



    // get the categories and ratings
    std::string cat_name;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> cat_name;
        Toy::data.category.push_back(cat_name);
    }
    double rating;
    for (long j = 0; j != Nc; ++j)
    {
        std::cin >> rating;
        data.ratings.push_back(rating);
    }

    std::cin >> data.room;
    // generate a unique key
    data.key = keygen(data.manuf_date, data.toy_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 manuf_date and Toy name.

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

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

```
void Toy::put(void) const
{
     long Nc = 3;

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

     // put the Manufacturing date
     std::cout << "Manufacture-Date: " << data.manuf_date
<< std::endl;

     // put the price
     std::cout << "price: " << data.price << std::endl;

     // put the Toy name
     std::cout << "Toy name: " << data.toy_name <<
std::endl;

     // put categories and ratings
     std::string cat_name;
     double rating;
     for (long j = 0; j != Nc; ++j)
     {
```

```
            cat_name = data.category[j];
            rating = data.ratings[j];
            std::cout << "\t" << cat_name << ":\t" << rating
<< std::endl;
     }
     // put Room for the toy
     std::cout << "Room: " << data.room << std::endl;
     // line spacing
     std::cout << std::endl << std::endl;
}
```

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

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

     return ret;
}
```

This function will add “_” between the manufacturing date and the Toy 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="04-30-2015_Hot_Wheels";


     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 <toys.h>       // Toy class
#include <iostream>
#include "proj_classes.h"   // main templated class
#include <string>



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



int main()
{
     // declare a Toy-database object, load from standard
input


     DB<Toy> myToyDB;


     // print out the data
     myToyDB.display();
```

```
    myToyDB.del(); //delete the record

    cout <<"-----------------After
deletion---------------------"<<endl;
    myToyDB.display();


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


    return 0;
}
```

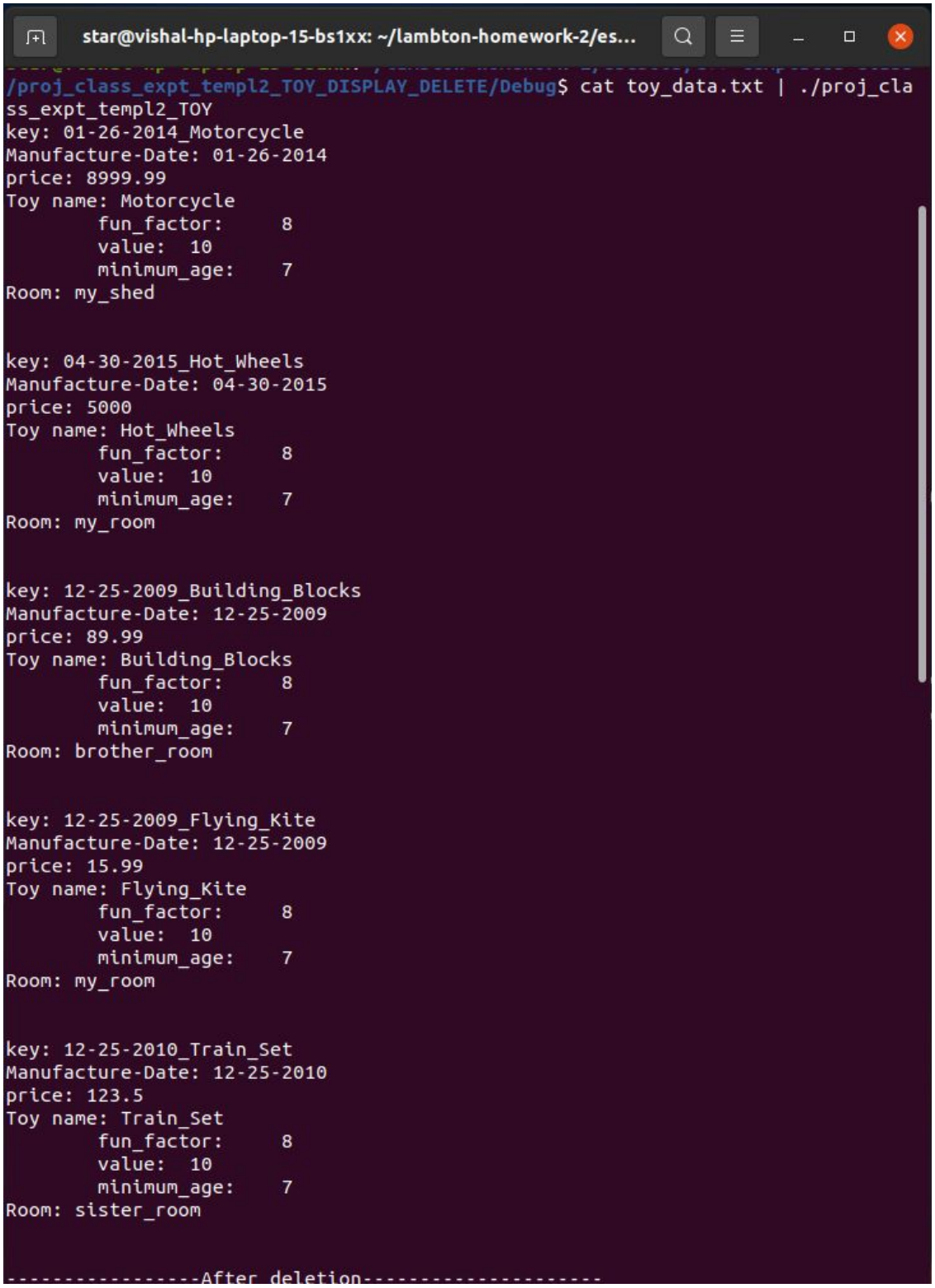
Text file---> toys_data.txt

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

.

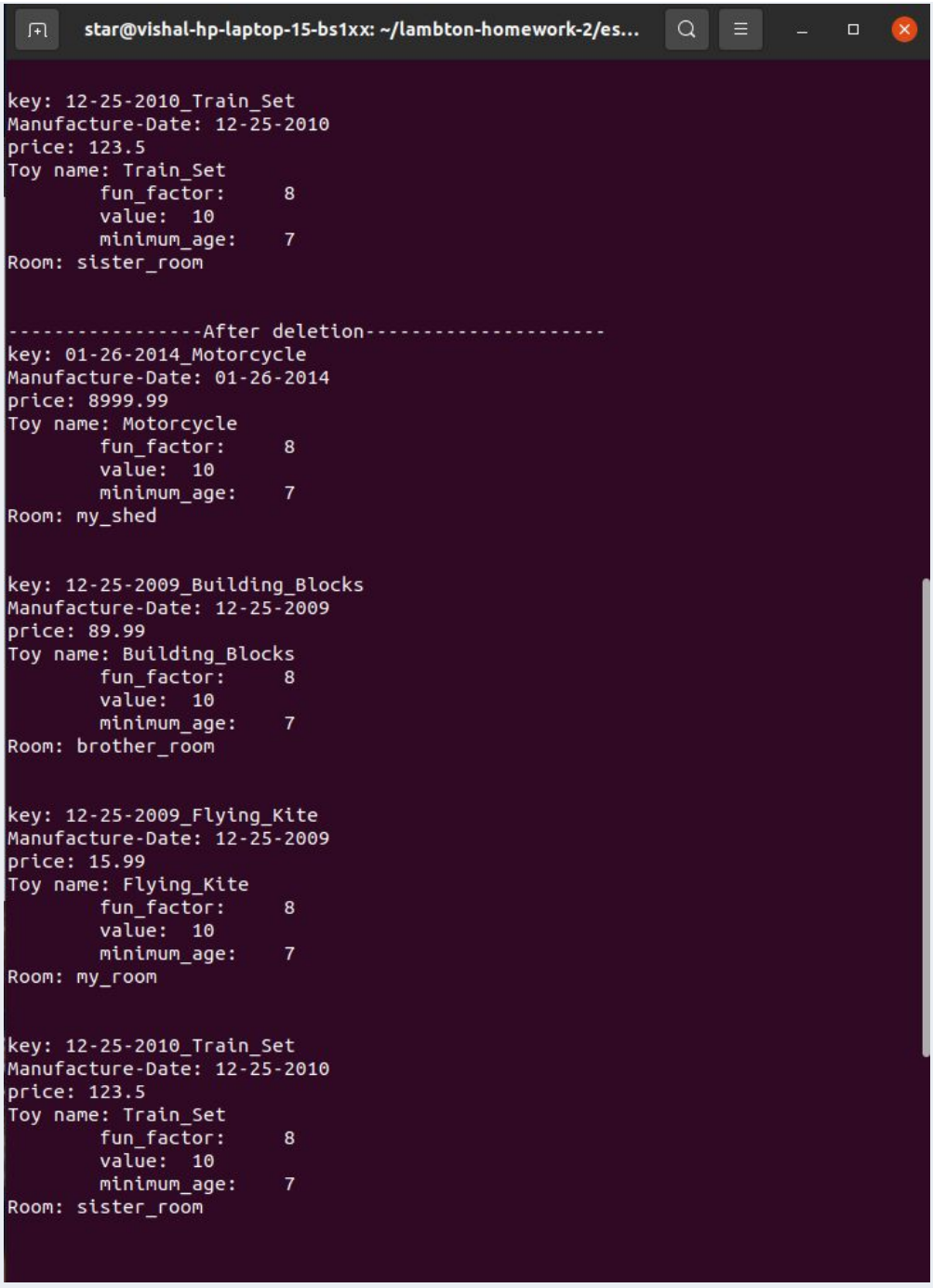

```
 1
 2 12-25-2009
 3 15.99
 4 Flying_Kite
 5 fun_factor      value   minimum_age
 6 8               10      7
 7 my_room
 8
 9 12-25-2009
10 89.99
11 Building_Blocks
12 fun_factor      value   minimum_age
13 9               8       8
14 brother_room
15
16 12-25-2010
17 123.50
18 Train_Set
19 fun_factor      value   minimum_age
20 10              7       10
21 sister_room
22
23 01-26-2014
24 8999.99
25 Motorcycle
26 fun_factor      value   minimum_age
27 10              9       16
28 my_shed
29
30 04-30-2015
31 5000.00
32 Hot_Wheels
33 fun_factor      value   minimum_age
34 7               3       5
35 my_room
36
37
38 -9999
39 0
40 dummy
41 fun_factor      value   minimum_age
42 0               0       0
43
```

Output of display () and del()



"04-30-2015_Hot_Wheels"

I have given the key (i.e.`std::string key="04-30-2015_Hot_Wheels";`)
So it will delete the record of `04-30-2015_Hot_Wheels.`

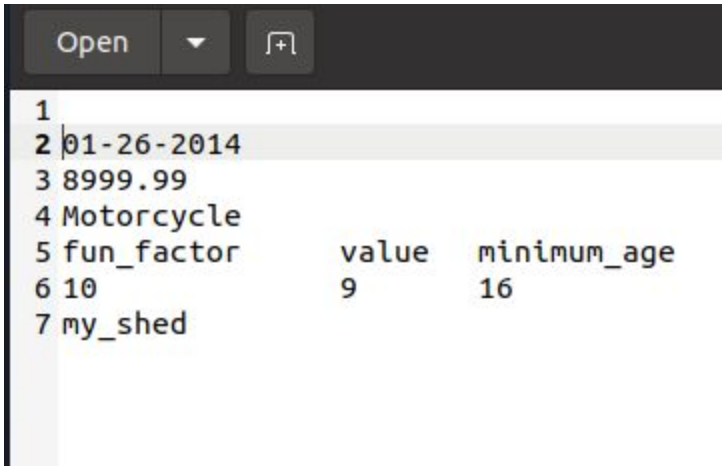


For adding the data to the database i have made the separate project with few changes in the templated class.h file

I have kept the default constructor empty for adding the element and I am using a separate text file (i.e. toy_1_entry.txt)to add the data.
Default constructor for new project

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

toy_1_entry.txt



proj_class_expt_templ.cpp

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

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

```
{
     // declare a Toy-database object, load from standard
input
     DB<Toy> myToyDB;
     cout<< "-------------------------Added one element in
the database---------------------------"<<endl;
     myToyDB.add(); //adding element
     myToyDB.display(); // displaying records
     // exit
     cout << endl << myToyDB.numelements() << " processed."
<< endl;
     return 0;
}
```

Output :

```
star@vishal-hp-laptop-15-bs1xx:~/cpp-workspace/proj_class_expt_templ2_TOY_ADD/De
bug$ cat toy_add_data.txt | ./proj_class_expt_templ2_TOY_ADD
-------------------------Added one element in the database----------------------
----
key: 01-26-2014_Motorcycle
Manufacture-Date: 01-26-2014
price: 8999.99
Toy name: Motorcycle
        fun_factor:     10
        value:  9
        minimum_age:    16
Room: my_shed
```

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)
toys.h

```
#ifndef TOYS_H_
#define TOYS_H_

#define END_MARKER "-9999"

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

typedef struct
{
     std::string key; // primary key
     std::string manuf_date;
     float price;
     std::string toy_name;
     std::vector<std::string> category;
     std::vector<double> ratings;
     std::string room;
} record_t;
```

```
/*
 * this defines a Toy object
 */
class Toy
{
public:
     Toy(void); // constructor
     // data of a Toy
     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 toy record is the last one,
updates last_record

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

#endif /* TOYS_H_ */
```

toys.cpp

```
#include <toys.h>
#include <iostream>
#include <string>
#include <vector>
```

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

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

// Toy::get() method, populates the data record from standard input
int Toy::get(void)
{
	long Nc=3; // stores the number of categories

	// get the Toy manufacturing date
	std::cin >> data.manuf_date;

	// get the price
	std::cin >> data.price;

	// get the toy name
	std::cin >> data.toy_name;



	// get the categories and ratings
	std::string cat_name;
	for (long j = 0; j != Nc; ++j)
```

```
        {
                std::cin >> cat_name;
                Toy::data.category.push_back(cat_name);
        }
        double rating;
        for (long j = 0; j != Nc; ++j)
        {
                std::cin >> rating;
                data.ratings.push_back(rating);
        }

        std::cin >> data.room;
        // generate a unique key
        data.key = keygen(data.manuf_date, data.toy_name);

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

// Toy::put() method, sends data contents to standard output
void Toy::put(void) const
{
        long Nc = 3;

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

        // put the Manufacturing date
        std::cout << "Manufacture-Date: " << data.manuf_date << std::endl;

        // put the price
        std::cout << "price: " << data.price << std::endl;

        // put the Toy name
        std::cout << "Toy name: " << data.toy_name << std::endl;
```

```
	// put categories and ratings
	std::string cat_name;
	double rating;
	for (long j = 0; j != Nc; ++j)
	{
		cat_name = data.category[j];
		rating = data.ratings[j];
		std::cout << "\t" << cat_name << ":\t" << rating << std::endl;
	}
	// put Room for the toy
	std::cout << "Room: " << data.room << std::endl;
	// line spacing
	std::cout << std::endl << std::endl;
}

// Toy::keygen() method
std::string Toy::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 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="04-30-2015_Hot_Wheels";


     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 <toys.h>      // Toy class
#include <iostream>
#include "proj_classes.h"   // main templated class
#include <string>

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

int main()
{
     // declare a Toy-database object, load from standard
input
     DB<Toy> myToyDB;
```

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


    // exit
    cout << endl << myToyDB.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 <toys.h>      // Toy class
#include <iostream>
#include "proj_classes.h"   // main templated class
#include <string>

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

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



    cout<< "-------------------------Added one element in
the database-------------------------"<<endl;
    myToyDB.add();
    myToyDB.display();

    cout << endl << myToyDB.numelements() << " processed."
<< endl;

    // exit
```

```
    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 **toys.cpp** and **toys.h** remains the same as Project 1.