**C++ Workshop – 150018**

**Homework Assignment #11**

**Files**

In this assignment we will perform random accesses to binary files.

A binary file (heretofor called the database) contains 100 identical sized records, each with a unique key (numbered from 1 to 100) corresponding to the record’s location in the file (the numbering is from 1 to 100 so that the record whose key is 1 is found in the beginning of the file).

Initially all records are empty, and during execution direct access to a record is done using its key. Since the entries are identical in size, the location of each entry in the file can be computed from the given key. It can be assumed that if the key stored in a record is 0, the record has not yet been created. Conversely, if the key has a value other than 0, then the record has already been created and contains actual information.

Write a program that will manage the enrollment of family members in activities offered in the neighborhood they live.

1. Define a class **Family** to store the information of each family:
   1. family id (int) – in the range 1-100 (this is the key of the record)
   2. last name (up to 20 chars)
   3. number of family members
   4. phone number (int)
   5. a Boolean array for each of the 6 activities offered in the neighborhood. A value true in the array indicates if at least one child in the family is registered for the activity, otherwise it is false.
2. Write a function **setFile** that initializes a binary file to store the data of 100 families. To do this, 100 blank family records should be placed in the file with the following data:
   1. family id: 0
   2. last name: an empty 20 length character string
   3. number of family members: 0
   4. phone number: 0
   5. a boolean array of size 6 initialized to false
3. Write the following function for a maintaining the database:
   1. Function **add** which adds a family. The function receives as input a file stream to a binary file. It then reads data for a family from standard input (console). Initially no children in the family are registered for any of the activities. If the family id given is legal but has so far not been registered in the database (i.e., the corresponding record has a family id of 0) a new family record is added to the database with the information that was read in.  
        
      If the family already exists in the database, then the exception ERROR: Family is already in the file should be thrown. If the family id is not valid (1-100), then the exception ERROR: Invalid family number should be thrown.
   2. Function **del** which deletes a family from the database. The function receives two parameters, a file stream to a binary file and a family id and deletes the corresponding record from the file whose id matches the given parameter. (Note – the implication of deleting an entry is to reset its family id to 0. There is no need to change other fields).  
      If the family id is not valid (1-100), then the exception ERROR: Invalid family number should be thrown.
   3. Function **update** to update a family’s record data. The function receives three parameters, a file stream to a binary file, a family id, and an STL queue. If the family id appears in the database, the function reads in 6 boolean values from standard input and updates the list of selected activities for the family.  
        
      Activities are limited to 10 children. Before performing the update, the program first checks that there is indeed room in a chosen activity (1-6) by iterating through all the families and checking whether there are less than 10 children currently registered for the activity. If there is space in the activity, the function updates the list of selected activities of the family. Make separate function for this, need to do 6 times  
        
      If there is no space for a requested activity, the function should add the family to a waiting list. For this purpose, an STL queue of **Family** should be used. A true value in the boolean vector for a family inserted onto the waiting list signifies the actvivity the family is waiting for. (Note, the same family can be enqueued multiple times if they were closed out of multiple activities.) so update to true but instead of putting back in file, put into queue  
        
      If the family does not exist in the database, then the exception ERROR: Family is not in the file should be thrown. If the family id is not valid (1-100), then the exception ERROR: Invalid family number should be thrown.
   4. Function **waiting** that handles waiting lists. The function receives two parameters, a file stream to a binary file and a waiting list. The queue The function should iterate over the list of families on the waiting list and check whether space has been freed for any of the activities they tried to register for (a true value in a family’s boolean vector means they are wait-listed for that activity). Use helper function made for update function To this end, the function should iterate over all the families registered in the database and check whether the total number registered in a activity is now less than 10 (due to updates during the program execution). If so, the function should print (to the screen) the family name, telephone number, and immediately afterwards input either a Y indicating that the family is still interested in registering for the activity, or N if the family is now longer interested in registrating. If the input was Y, the family record in the database should be updated accordingly. In any case the queue should be updated. Dequeue it
   5. A function **rishum** for checking registration of an activity. The function receives as a parameter a reference to a binary file, a second parameter which is a family id, and a third parameter which is an activity number (1 to 6). If the family id appears in the database, and the family is registered for the given activity, the function then returns true. Otherwise, the function will return false.  
        
      If the family does not exist in the database, then the exception ERROR: Family is not in the file should be thrown. If the family id is not valid (1-100), then the exception ERROR: Invalid family number should be thrown. If the activity number is not valid (1-6), then the exception ERROR: Invalid class number should be thrown.
   6. A function **print** for printing family information. The function receives as a parameter a reference to a binary file, and a second parameter that is a family id. If the family id exists in the data file, the function prints the family information to the standard output (screen) as follows: last name, number of persons, telephone number and for each activity, if a child is registered for the class (Y or N).  
        
      If the family does not exist in the database, then the exception ERROR: Family is not in the file should be thrown. If the family id is not valid (1-100), then the exception ERROR: Invalid family number should be thrown.
   7. A function **inClass** for printing the list of families registered for a particular activity. The function receives as a parameter a reference to a binary file, and a second parameter which is an activity number (1 to 6). The function should print to the standard output (screen) the names of the families (last name) registered for the activity whose number was received as a parameter.  
        
      If the activity number is not valid (1-6), then the exception ERROR: Invalid class number should be thrown.

Here is the main program:

int main()

{

Queue<Family> q;

fstream file;

file.open("families.txt", ios::binary | ios::in | ios::out);

if (!file)

{

cout << "ERROR: couldn't open file\n";

return 0;

}

setFile (file);

int choice;

int snum;

int cnum;

cout << "Choices are:\n0 to exit\n1 to add a family\n2 to delete a family\n3 to update rishum to classes\n4 to update waiting to classes \n5 to check rishum for a classas\n6 to print a family\n7 to print all the families that participate in a specific class\n";

cout << "enter 0-7:\n";

cin >> choice;

while (choice)

{

switch (choice)

{

case ADD://add to the file

try {add(file);}

catch (const char \* msg) { cout << msg; }

break;

case DEL://delete from file

cout << "enter number of family to delete:\n";

cin >> snum;

try { del(file, snum); }

catch (const char \* msg) { cout << msg; }

break;

case UPDATE://update the list of classes of a family

cout << "enter number of family to update:\n";

cin >> snum;

try { update(file, snum, q); }

catch (const char \* msg) { cout << msg; }

break;

case WAITING://update the list of classes of a waiting family

waiting(file, q);

break;

case RISHUM://check rishum to a specific class

cout << "enter number of family to check rishum:\n";

cin >> snum;

cout << "enter number of class to check rishum:\n";

cin >> cnum;

try

{

cout << "The family is" << (rishum(file, snum, cnum) ? " " : " not ") << "taking the class\n";

}

catch (const char \* msg) { cout << msg; }

break;

case PRINT://print the details of a specific family

cout << "enter number of family to print:\n";

cin >> snum;

try { print(file, snum); }

catch (const char \* msg) { cout << msg; }

break;

case CLASS://print the details of all the families that are taking a specific class

cout << "enter number of class to check rishum:\n";

cin >> cnum;

try { inClass(file, cnum); }

catch (const char \* msg) { cout << msg; }

break;

default:

cout << "ERROR: invalid choice\n";

}

cout << "\nenter 0-7:\n";

cin >> choice;

}

file.close();

return 0;

}

בהצלחה!