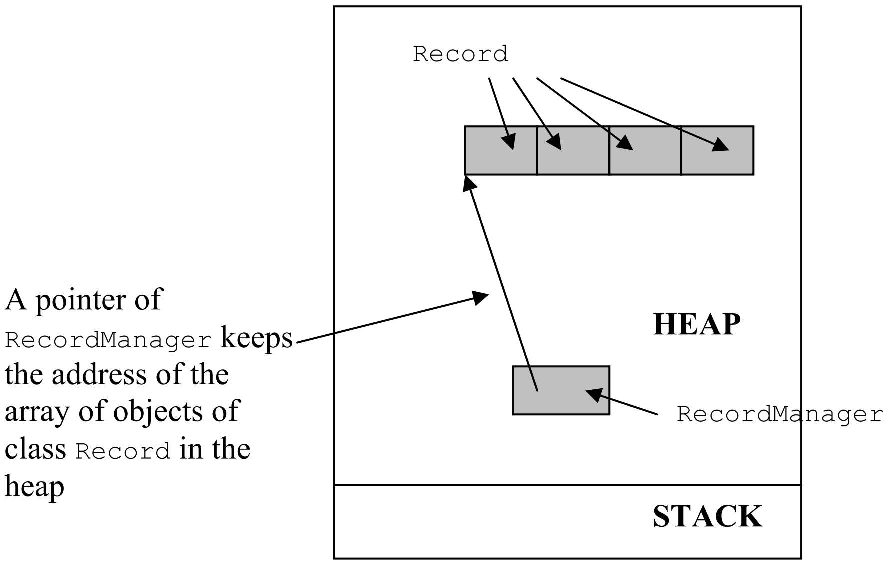
THE HONG KONG POLYTECHNIC UNIVERSITY
Department of Electronic and Information Engineering

Computer Programming (ENG236) - Homework 3

A. By using Visual Studio .NET, implement all member functions of the class `RecordManager` based on the class specification as given below. Note that the class `RecordManager` should instantiate an array of objects of class `Record` in the heap. The relationship between `RecordManager` and `Record` is shown in the diagram below:


Record
HEAP
RecordManager
STACK
A pointer of RecordManager keeps the address of the array of objects of class Record in the heap

The implementation of the class `Record` has been given below. You are only required to implement all member functions of the class `RecordManager`. When you have finished implementing those member functions, you should develop a console application such that it will:

1. Create an object of the class `RecordManager` in the heap with input parameters to be your surname (e.g. Chan) and the maximum number of `Record` objects (e.g. 10) you want to create in the system.
2. Call the member functions of `RecordManager` one by one so that the correct implementation of these member functions can be shown. You can call the functions any way you want as long as they can show their implementation meets all the requirements in the specification.

```
// The classes Record and RecordManager are defined below.
// The class Record has been implemented. You are required to implement the
//     member functions of RecordManager
// (Words started with $ refer to the member variables as stated in the
//    private sections)

class Record
{
public:
      Record() {vacant = true;}
      // When an object of Record is instantiated, do the following
      //    $vacant = true
      char * getID() {return studentID;}
      // Return the string $studentID[]
      int getmarks() {return marks;}
      // Return the student's $marks
      bool getvacant() {return vacant;}
      // Return the state of $vacant, i.e. true or false
      void setID(char * id) {strncpy(studentID,id,10);}
      // Copy the string id[] to $studentID[]
      void setmarks(int mk) {marks = mk;}
      // Set $marks = mk
      void setvacant(bool vac) {vacant = vac;}
      // Set the state of $vacant = vac
private:
      char studentID[10];           // Keep the studentID of a student
      int marks;                    // Keep the marks obtained by a student
      bool vacant;                  // True if the record is empty, false if
                                    // the record is used
};


class RecordManager
{
public:
      RecordManager(char * name, int num);
      // When the object is instantiated, do the following
      //    Copy the string name[] to $userName[]
      //    $recordNum = num
      //    Create an array of num objects of the class Record in the heap.
      //          The content of the class Record can be found above.
      //          The pointer of the array should be saved in $pRecordArr

      ~RecordManager();
      // Delete all records in the heap

      bool findRecord(char *id);
      // Find if a non-empty Record object with $studentID[] the same as id[].
      // If yes, return true, else return false

      int addRecord(char *id, int mk);
      // Find if a non-empty Record object with $studentID[] the same as id[].
      // If yes, copy mk into the $marks field of that Record object.
      // If no, find an empty Record object (with $vacant flag equal to true).
      //    Set the $vacant flag of that Record object to false
      //    Copy id[] into the $studentID[] field of that Record object
      //    Copy mk into the $marks field of that Record object
      // Return -1 if no empty Record object can be found, return 1 if an old
      //    Record object is updated; otherwise return 0

      bool delRecord(char *id);
      // Find a non-empty Record with $studentID[] the same as id[].
      // Set the $vacant flag of that Record object to true
      // Return false if there is no Record object having $studentID[] the same
      //    as id[]; otherwise return true

      int showRecord(char *id, int *pmk, int arrIndex);
      // Return by reference the content of Record[arrIndex] through the two
      //    input parameters id and pmk, i.e. to show the values of $studentID[]
      //    and $marks of Record[arrIndex] using pass by reference.
      // If Record[arrIndex] is vacant, return -1; else return 0


private:
      char userName[80];      // Store the username
      int recordNum;          // Keep the number of records
      Record * pRecordArr;    // Keep the pointer that points to the array in
heap
};
```

B. By using Visual Studio .NET, develop a static library, namely RecordManager.lib, based on the class specification as given in part A.

C. By using the static library you developed in part B, develop a console application in Visual Studio .NET such that it will first give a welcome message to the user and ask the user to enter his/her surname as well as the maximum number of records to be kept in the system. It will then repeatedly show the following menu:

```
      Student record management system:
      1. Add Record
      2. Edit Record
      3. Delete Record
      4. Show All Records
```

```
5. Quit
```

- If the user chooses 1, your program should allow the user to enter a `studentID` number (e.g. 01234567d) and his/her `marks` (e.g. 80).
  - If there is no record in the system with the same `studentID`, a new record will be created and these data will be saved into the record. A message "`The record of studentID is created.`" should be shown, where `studentID` is the input of the user.
  - If there is no empty record to save the data, a warning message "`No empty record is found. Please free up a record to store the data.`" should be shown.
  - If there is already the record of `studentID`, do NOT update the marks but a warning message "`The record of studentID exists.`" should be shown.

  Get back to the main menu after finishing the above operation.
- If the user chooses 2, your program should allow the user to enter a `studentID` number (e.g. 01234567d) and his/her `marks` (e.g. 80).
  - Your program will compare the number with all records in the system. The one that matches will have its marks modified with the marks the user entered. A message "`The record of studentID has been modified.`" should be shown, where `studentID` is the input of the user.
  - If no matching is found, a warning message "`The record of studentID is not found.`" should be shown, where `studentID` is the input of the user.

  Get back to the main menu after finishing the above operation.
- If the user chooses 3, your program should allow the user to enter a `studentID` number.
  - Your program will compare the number with all records in the system. The one that matches will be deleted. A message "`The record of studentID has been deleted.`" should be shown, where `studentID` is the input of the user.
  - If no match is found, a warning message "`The record of studentID is not found.`" should be shown, where `studentID` is the input of the user.

  Get back to the main menu after finishing the above operation.
- If the user chooses 4, your program should show the data of all records in the following format:

  ```
  StudentID   Marks
  01234567d   50
  02345678x   75
  04123456x   48
  05123456d   60
          :    :
          :    :
  ```

  Get back to the main menu after finishing the above operation.
- If the user chooses 5, a message "`Goodbye xxxxxxxx!!!`", where `xxxxxxxx` is the surname of the user, should be shown. <u>All</u> the data that have been created in the heap should be deleted; and then the program will quit.

When doing the computation as mentioned above, it is compulsory to use the member functions of your library whenever applicable.

Instructions

1. Use a new project for each question. It means that your files should be contained in 3 projects.
2. Try to explain your program as clear as possible using comments.
3. The program structure will be an important part. Never try to write your program with a single `main()` function.

```
/*=====================================================*\
| Homework 3 -- Part A                                  |
| Project name: HW3PA                                   |
| Test program for the recordManager object.            |
| This is for testing the member functions              |
| findRecord, addRecord, delRecord and showRecord.      |
|                                                       |
| Usage: partA                                          |
| Version: 1                                            |
| Date: Dec. 3, 2005                                    |
\*=====================================================*/

#include <iostream>
//#include <string.h>
using namespace std;

// (Words started with $ refer to the member variables as stated in the private sections)

class Record
{
public:
       Record() {vacant = true;}
       char * getID() {return studentID;}
       int getmarks() {return marks;}
       bool getvacant() {return vacant;}
       void setID(char * id) {strncpy(studentID,id,10);}
       void setmarks(int mk) {marks = mk;}
       void setvacant(bool vac) {vacant = vac;}
private:
       char studentID[10];        // Keep the studentID of a student
       int marks;                        // Keep the marks obtained by a student
       bool vacant;               // True if the record is empty, false if
                                         // the record is used
};

class RecordManager
{
public:
     RecordManager(char * name, int num);
     ~RecordManager();
     bool findRecord(char *id);
     int addRecord(char *id, int mk);
     bool delRecord(char *id);
     int showRecord(char *id, int *pmk, int arrIndex);
private:
     char userName[80];   // Store the username
     int recordNum;       // Keep the number of records
     Record * pRecordArr; // Keep the pointer that points to the array in heap
};

RecordManager::RecordManager(char * name, int num)
// When the object is instantiated, do the following
//   Copy the string name[] to $userName[]
//   $recordNum = num
//   Create an array of num objects of the class Record in the heap.
//        The content of the class Record can be found above.
//        The pointer of the array should be saved in $pRecordArr
{
     strcpy(userName,name);
     recordNum = num;
     pRecordArr= new Record[num];
}

RecordManager::~RecordManager()
// Delete all records in the heap
{
     delete [] pRecordArr;
     cout <<"Destructor of RecordManager object runs." <<endl;
}

bool RecordManager::findRecord(char *id)
// Find if a non-empty Record object with $studentID[] the same as id[].
// If yes, return true, else return false
{
     for (int i=0; i<recordNum; i++)
     {
          if ((!pRecordArr[i].getvacant()) && (strcmp(id, pRecordArr[i].getID())==0))
               return true;  // if current record is not vacant and its ID is the same
as the given ID, return true
     }
     return false;
}

int RecordManager::addRecord(char *id, int mk)
// Find if a non-empty Record object with $studentID[] the same as id[].
// If yes, copy mk into the $marks field of that Record object.
// If no, find an empty Record object (with $vacant flag equal to true).
// Set the $vacant flag of that Record object to false
// Copy id[] into the $studentID[] field of that Record object
// Copy mk into the $marks field of that Record object
// Return -1 if no empty Record object can be found, return 1 if an old
// Record object is updated; otherwise return 0
{
     int i, count=0;  //count stores the number of empty records
     for (i=0; i<recordNum; i++) //Go through the whole pRecordArr
     {
          if (pRecordArr[i].getvacant()) count++; // If the current record is vacant,
count ++
     }
     if (count == 0 && (!findRecord(id)))  // No record and no vacant slot
          return -1;
     else
     {
          if (findRecord(id)) // The record exists
          {
               for (i=0; i<recordNum; i++)
                    if (strcmp(id, pRecordArr[i].getID()) ==0)  // Look for the record
and update the mark
                         pRecordArr[i].setmarks(mk);
               return 1;  // An odd record is updated.
          }
          else // Record does not exist but vacant slot exists
               for (int i=0; i<recordNum; i++)
               {
                    if (pRecordArr[i].getvacant())  // Fill in the ID, mark and change
vacant to false
                    {
                         pRecordArr[i].setID(id);
                         pRecordArr[i].setmarks(mk);
                         pRecordArr[i].setvacant(false);
                         return 0;  // Leave immediately after filling in
                    }
               }
     }
}

bool RecordManager::delRecord(char *id)
// Find a non-empty Record with $studentID[] the same as id[].
// Set the $vacant flag of that Record object to true
// Return false if there is no Record object having $studentID[] the same
// as id[]; otherwise return true
{
     if (findRecord(id))  // When the record is found
     {
          for (int i=0; i<recordNum; i++)      // Go thorugh the whole pRecordArr
               if ((strcmp(pRecordArr[i].getID(), id)==0))
               {
                    pRecordArr[i].setvacant(true);   // change vacant to true
                    return true;  // Once found, set it to vacant and LEAVE
               }
     }
     else
          return false;     // Record not found
}

int RecordManager::showRecord(char *id, int *pmk, int arrIndex)
// Return by reference the content of Record[arrIndex] through the two
// input parameters id and pmk, i.e. to show the values of $studentID[]
// and $marks of Record[arrIndex] using pass by reference.
// If Record[arrIndex] is vacant, return -1; else return 0
```

```
{
     if (pRecordArr[arrIndex].getvacant())
          return -1;  // if the requested record is vacant, return -1
     else  // put the student ID and marks to reference *id and *pmk
     {
          strcpy(id,pRecordArr[arrIndex].getID());
          *pmk=pRecordArr[arrIndex].getmarks();
          return 0;
     }
}

int main()
{
     int Num=5, ret_no;
     bool ret_bo;
     RecordManager *Frank = new RecordManager("Leung", Num);
     char Id[10];
     int Mark=0;
     Frank->addRecord("123456a", 22);  // Test adding an empty record
     Frank->addRecord("123456c", 23);
     Frank->addRecord("123456b", 2);
     Frank->addRecord("123456d", 3);
     ret_no=Frank->addRecord("123456e", 4);
     cout<<"Adding to vacant record, return 0: "<<ret_no<<endl;
     ret_no=Frank->addRecord("123456a", 1);  // Test updating an odd record
     cout<<"Updating an old record, return 1: "<<ret_no<<endl;
     ret_no=Frank->addRecord("123456f", 25);  // Test adding record to a fully occupied
array
     cout<<"Array full, return -1: "<<ret_no<<endl;
     ret_bo=Frank->findRecord("123456c");  // Test findRecord()
     cout << "Test findRecord(), 123456c found, return 1: "<<ret_bo <<endl;
     ret_bo=Frank->findRecord("654321");  // Test findRecord()
     cout << "Test findRecord(), 654321 not found, return 0: "<<ret_bo <<endl;
     ret_bo=Frank->delRecord("123456c");  // Test delRecord()
     cout << "Test delRecord(), 123456c found, return 1: "<<ret_bo <<endl;
     ret_bo=Frank->delRecord("233333b");
     cout << "Test delRecord(), 233333b not found, return 0: "<<ret_bo <<endl;
     if (Frank->findRecord("123456c"))  // Test findRecord() and delRecord()
          cout << "This line should NOT be seen" <<endl;
     cout<<"Test showRecord(), expect to see: "<<endl;
     cout<<"Record 0\tStudent ID: 123456a\tMarks: 1"<<endl;
     cout<<"Record 2\tStudent ID: 123456b\tMarks: 2"<<endl;
     cout<<"Record 3\tStudent ID: 123456d\tMarks: 3"<<endl;
     cout<<"Record 4\tStudent ID: 123456e\tMarks: 4"<<endl;
     for (int i=0; i<Num;i++)
          if(Frank->showRecord(Id, &Mark, i)==0)  // Test showRecord()
          {
               cout<<"Record "<< i <<'\t';
               cout<<"Student ID: "<< Id <<'\t';
               cout<<"Marks: "<< Mark <<endl;
          }
     delete Frank;  // Test Destructor
}


// Part B
// Project name: RecordManager
// The following is stored in the file Record.h
#include <iostream>
//#include <string.h>
using namespace std;
class Record
{
public:
       Record() {vacant = true;}
       char * getID() {return studentID;}
       int getmarks() {return marks;}
       bool getvacant() {return vacant;}
       void setID(char * id) {strncpy(studentID,id,10);}
       void setmarks(int mk) {marks = mk;}
       void setvacant(bool vac) {vacant = vac;}
private:
       char studentID[10];        // Keep the studentID of a student
       int marks;                          // Keep the marks obtained by a student
       bool vacant;               // True if the record is empty, false if
```

```
                                        // the record is used
};

class RecordManager
{
public:
     RecordManager(char * name, int num);
     ~RecordManager();
     bool findRecord(char *id);
     int addRecord(char *id, int mk);
     bool delRecord(char *id);
     int showRecord(char *id, int *pmk, int arrIndex);
private:
     char userName[80];   // Store the username
     int recordNum;       // Keep the number of records
     Record * pRecordArr; // Keep the pointer that points to the array in heap
};


// The following can be stored in the file PartB.cpp
#include "Record.h"

RecordManager::RecordManager(char * name, int num)
{
     strcpy(userName,name);
     recordNum = num;
     pRecordArr= new Record[num];
}

RecordManager::~RecordManager()
{
     delete [] pRecordArr;
     cout <<"Destructor of RecordManager object runs." <<endl;
}

bool RecordManager::findRecord(char *id)
{
     for (int i=0; i<recordNum; i++)
     {
          if ((!pRecordArr[i].getvacant()) && (strcmp(id, pRecordArr[i].getID())==0))
               return true;
     }
     return false;
}

int RecordManager::addRecord(char *id, int mk)
{
     int i, count=0;  //count stores the number of empty records
     for (i=0; i<recordNum; i++) //Go through the whole pRecordArr
     {
          if (pRecordArr[i].getvacant()) count++;
     }
     if (count == 0 && (!findRecord(id)))  // No record and no vacant slot
          return -1;
     else
     {
          if (findRecord(id)) // The record exists
          {
               for (i=0; i<recordNum; i++)
                    if (strcmp(id, pRecordArr[i].getID()) ==0)
                         pRecordArr[i].setmarks(mk);
               return 1;  // An odd record is updated.
          }
          else // Record does not exist but vacant slot exists
               for (int i=0; i<recordNum; i++)
               {
                    if (pRecordArr[i].getvacant())
                    {
                         pRecordArr[i].setID(id);
                         pRecordArr[i].setmarks(mk);
                         pRecordArr[i].setvacant(false);
                         return 0;  // Leave immediately after filling in
                    }
               }
     }
```

```
}

bool RecordManager::delRecord(char *id)
{
     if (findRecord(id))  // When the record is found
     {
          for (int i=0; i<recordNum; i++)      // Go thorugh the whole pRecordArr
               if ((strcmp(pRecordArr[i].getID(), id)==0))
               {
                    pRecordArr[i].setvacant(true);   // change vacant to true
                    return true;  // Once found, set it to vacant and LEAVE
               }
     }
     else
          return false;     // Record not found
}

int RecordManager::showRecord(char *id, int *pmk, int arrIndex)
{
     if (pRecordArr[arrIndex].getvacant())
          return -1;  // if the requested record is vacant, return -1
     else  // put the student ID and marks to reference *id and *pmk
     {
          strcpy(id,pRecordArr[arrIndex].getID());
          *pmk=pRecordArr[arrIndex].getmarks();
          return 0;
     }
}


/*====================================================*\
| Homework 3 -- Part C                                 |
| Project Name: HW3PC                                  |
| Main program for the student record                  |
| management system using class RecordManager          |
| Users can create and modify a list of student        |
| records by Adding, editing and deleting records.     |
| We can also view all records in the list.            |
| Library needed: RecordManager.lib                    |
| Header file needed: Record.h                         |
|                                                      |
| Usage: PartC                                         |
| Version: 1                                           |
| Date: Dec. 3, 2005                                   |
\*====================================================*/

#include "Record.h"

void menu(char *, int); // The recurrent menu
void getIDnMarks(char *, int *); // Get student ID and marks; pass by reference

int main()
{
     char userName[80];
     int recordNum;
     cout<<"Welcome to the Student Record Management System.\n\n";
     cout<< "Please enter your surname:";
     cin>> userName;
     cout<< "How many records do you want to create:";
     cin>> recordNum;
     menu(userName, recordNum);
     return 0;
}

void menu(char *userName, int recordNum)
{
     RecordManager *User = new RecordManager(userName, recordNum);
     int f_reti;  //for storing integer results from functions
     int marks=0;
     char choice, studentID[10];
     do
     {// ============ MENU ============
          cout<<"\nStudent record management system:\n";
          cout<<"1. Add Record\n";
          cout<<"2. Edit Record\n";
          cout<<"3. Delete Record\n";
          cout<<"4. Show All Records\n";
          cout<<"5. Quit\n";
          cout<<"Please enter your choice: ";
          cin >> choice;
          switch (choice)
          {
               case '1':
               {
                    getIDnMarks(studentID, &marks);
                    if (!User->findRecord(studentID)) // Cannot find the record
                    {
                         f_reti=User->addRecord(studentID, marks); // Add a new record
                         if (f_reti==0)
                              cout<<"\nThe record of "<<studentID<<" is created.\n\n";
                         if (f_reti == -1)  // No empty slot to add
                              cout<<"\nNo empty record is found. Please free up a
record to store the data.\n\n";
                    }
                    else
                         cout<<"\nThe record of "<<studentID<<" exists.\n\n";
                    break;
               }
               case '2':
               {
                    getIDnMarks(studentID, &marks);
                    if (!User->findRecord(studentID)) // Cannot find the record
                         cout<<"\nThe record of "<<studentID<<" is not found.\n\n";
                    else
                    {
                         User->addRecord(studentID, marks); // Update record
                        cout<<"\nThe record of "<<studentID<<" has been modified.\n\n";
                    }
                    break;
               }
               case '3': // Read in the student ID and delete the record
               {
                    cout<<"Student number:";
                    cin>>studentID;
                    if (User->delRecord(studentID))  // Successful deletion
                         cout<<"\nThe record of "<<studentID<<" has been deleted.\n\n";
                    else
                         cout<<"\nThe record of "<<studentID<<" is not found.\n\n";
                    break;
               }
               case '4':
               {
                    cout<<"\n\tStudentID\tMarks\n";
                    for (int i=0; i<recordNum; i++)  // Check according to the indices
                         if(User->showRecord(studentID, &marks,i)==0) // will show
records if they are not vacant.
                              cout<<'\t'<<studentID<<'\t'<<marks<<endl;
                    break;
               }
               case '5':
               {
                    cout<< "Goodbye "<<userName<<"!!!\n";
                    delete  User; // quit and delete objects in the heap
                    break;
               }
               default: cout<<"\nInvalid input, please re-enter.\n\n"; break;
          }
     }  while (choice != '5');
}

void getIDnMarks(char *ID, int *marks)
{
     cout<<"Student number:";
     cin>>ID;
     cout<<"Marks:";
     cin>>*marks;
}
```