Assignment 2 Schedule of Activities

[Weight: 8 marks out of the final mark of this course]

Deadline: December 8, 2012 (Saturday of week 14) ***Exact hours of submission deadline will be announced at the course web.*

Note: This time there is only 1 submission deadline as stated above. (No interim assessment.)
For those with "creative design", demo is needed (around December 11, 2012)

Submission of report (similar to assignment 1): You may submit during demo Hardcopy or on/before Dec 15 softcopy (by email)

For late submissions, 20% of your original marks will be deducted if you hand in 1-day late, 40% for 2-day, 60% for 3-day and 80% for 4-day. Assignments handed in 5 days after the due date will not be accepted.

For penalty of dishonest cases or let others have a chance to copy your code, please refer to my lecture notes given in week 1.

Students must obtain the following results in sequence:

Phase 1 (100% correct in PASS) ==> Phase 2 (100% correct in PASS) ==> Your creative design Also upload to PASS

- If you can finish Phase 1 with good programming styles => B-/B
- If you can finish Phase 2 with good programming styles => up to B+
- If you can give a creative design after the above (good programming styles) => A- to A++ See Page 4

For "Good Programming Styles", please refer to *Useful Tips and Notes* at the course web.

Also Note:
- For phases 1 and 2, DO NOT use techniques or built-in functions which are not covered in our course (eg. **`templates`**, **`setiosflags( ios::left )`**, ..)
- DO NOT use the **`string`** class. But **`cstring`** functions like **`strcpy`**, **`strcmp`**, and **`strlen`** are allowed.

Phase 1

Implement an **`Activity`** class that represents activity entries like the contents in the file below:

Sample input file (t1.txt / (Available at course web)

```
09:00  30   Go to laundry shop
15:00  60   Swim
20:00  90   Movie
23:30  45   Long-distance call
08:20  30   Breakfast with Maggie
12:00  50   Lunch
19:00  45   Dinner at home
```

- Each line has 3 columns:
  Start time, duration (in minutes), and description
- The columns are separated by whitespace(s).

Your Activity class should include the following private data members:

(1) start time of activity (hour and minute in integer(s), 24-hours format)
(2) duration of activity in minutes (an integer)
(3) description of activity (a cstring of at most 25 characters, plus 1 more place for '\0')

You are given a driver program that inputs the activities from a text file and lists them, as shown on next page.

Driver program for this phase
(Available at course web)

DO NOT UPLOAD THIS DRIVER PROGRAM (main()) TO PASS

```
#include "activity.h"
#include <iostream>
using namespace std;

void main()
{
  //The activities array and count of activities (n)
  Activity activities[20];
  int n;

  cout << "[A] Input activities from file\n"
  cout << "==============================\n\
  readFileOfActivities(activities, n);
  cout << n << " activities are extracted.\n\n\n";

  cout << "[B] Display list of activities\n";
  cout << "==============================\n";

  for (int i=0;i<n;i++)
     cout << activities[i] << endl;
  cout << endl << endl;

  cout << "Thank you for using this program.\n\n";
}
```

Example 1 (Underlined contents are input by user)

```
[A] Input activities from file
==============================
Type the pathname of the input file: c:\t1.txt
7 activities are extracted.

[B] Display list of activities
==============================
09:00 - 09:30 Go to laundry shop (30 minutes)
15:00 - 16:00 Swim (1 hour)
20:00 - 21:30 Movie (1 hour 30 minutes)
23:30 - 00:15 Long-distance call (45 minutes)
08:20 - 08:50 Breakfast with Maggie (30 minutes)
12:00 - 12:50 Lunch (50 minutes)
19:00 - 19:45 Dinner at home (45 minutes)


Thank you for using this program.
Press any key to continue . . .
```

You should create the Activity class (activity.h and activity.cpp) accordingly, based on the required output format in Example 1.

Your activity.h should contain **at least** the followings:

```
..
#include <iostream>
using namespace std;

//The Activity Class
class Activity {
public:
  …
  … // (4) add appropriate constructor(s) here
  …

  // (5) and (6) Extraction and Insertion operators for the class
  friend ostream &operator <<(ostream &outs, const Activity &activity);
  friend istream &operator >>(istream &ins, Activity &activity);

private:
  …
  … // add private data members here (1) to (3) – see page 1
  …
};

// (7) Ask for the pathname of an input file that contains a list of activities.  Then read them.
void readFileOfActivities(Activity activities[], int &n);
..
```

Additional Assumption and Notes:

- You may assume that the file pathname entered by the user must be correct.
- You may assume that all contents in the file are correct (but, as usual, there may or may not be empty line(s) at the end)

[Continue on next page]

- You may add additional assumptions where needed.
- You may add additional functions and/or constants where needed. [No global variable for this phase.]
- A constructor of the **`Reservation`** class should setup all data members, which involve **`cstrings`**:
  - *If we want to initialize a* **`cstring`** *to empty, what should we do?*
  - *If we want to copy the contents from a* **`cstring`** *to another, we have at least 2 approaches:*
    - *(i) Use the* **`strcpy`** *function. How?*
    - *(ii) Write a function by ourselves. How?*

HINT: USE DEBUGGER!

Phase 2 Comparison, Insertion sort and Binary search

The following figure illustrates the additional features you need to implement in Phase 2. Read next page for all details.

Driver program for this phase (Available at course web)
[Do not submit to PASS]

```
#include "activity.h"
#include <iostream>
using namespace std;
void main()
{
   //The activities array and count of activities (n)
   Activity activities[20];
   int n;

   //For part D
   int id_found;
   int search_hh, search_mm;
   char dummy; //colon between hh and mm

   cout << "[A] Input activities from file\n";
   cout << "=============================\n";
   readFileOfActivities(activities, n);
   cout << endl;
   cout << n << " activities are extracted.\n\n\n";

   cout << "[B] Display list of activities\n";
   cout << "=============================\n";

   for (int i=0;i<n;i++)
         cout << activities[i] << endl;
   cout << endl << endl;

   cout << "[C] Sort activities\n";
   cout << "===================\n";
   insertion_sort(activities,n);
   for (int i=0;i<n;i++)
       cout << activities[i] << endl;

   cout << endl << endl;

   cout << "[D] Search activities\n";
   cout << "=====================\n";
   cout << "Type a start time to search: ";
   cin >> search_hh >> dummy >> search_mm;

   if (binary_search(activities,n,search_hh,search_mm,id_found))
   {   cout << "\n";
       cout << activities[id_found] << "\n\n\n";
   }
   else
       cout << "\nNo activity is found.\n\n\n";

   cout << "Thank you for using this program.\n\n";
}
```

Example 2 (Underlined contents are input by user)

```
[A] Input activities from file
=============================
Type the pathname of the input file: c:\t1.txt

7 activities are extracted.


[B] Display list of activities
=============================
09:00 - 09:30 Go to laundry shop (30 minutes)
15:00 - 16:00 Swim (1 hour)
20:00 - 21:30 Movie (1 hour 30 minutes)
23:30 - 00:15 Long-distance call (45 minutes)
08:20 - 08:50 Breakfast with Maggie (30 minutes)
12:00 - 12:50 Lunch (50 minutes)
19:00 - 19:45 Dinner at home (45 minutes)

[C] Sort activities
===================
08:20 - 08:50 Breakfast with Maggie (30 minutes)
09:00 - 09:30 Go to laundry shop (30 minutes)
12:00 - 12:50 Lunch (50 minutes)
15:00 - 16:00 Swim (1 hour)
19:00 - 19:45 Dinner at home (45 minutes)
20:00 - 21:30 Movie (1 hour 30 minutes)
23:30 - 00:15 Long-distance call (45 minutes)


[D] Search activities
=====================
Type a start time to search: 20:00

20:00 - 21:30 Movie (1 hour 30 minutes)


Thank you for using this program.

Press any key to continue . . .
```

Your tasks for Phase 2:

In activity.h and activity.cpp, add the following functions for comparison, sorting, and searching:

(1) // The > operator for the class – compare 2 activities' start times based on both hours and minutes
```
friend bool operator >(const Activity &activity1, const Activity &activity2);
```

(2) // The **insertion sort** function for sorting a list of activities, in ascending order of their start times
```
void insertion_sort(..);
```

(3) // The **binary search** function to search an activity inside an array, based on given start time (hh and mm)
// Return the search result (true/false), set the searched location (`idFound`) if the activity is found in the array.
```
bool binary_search(.., int &idFound);
```

Note that the basics of both (2) and (3) were taught in lecture07. Now you may implement them as friend or non-friend functions for array of reservations. For their parameters, you should refer to how they are called from `main()` as given on next page.

You may add additional functions and/or constants where needed. [No global variable for this phase.]

The driver program and sample run-down are given on next page.

You may assume that all activities' start times are different.

Suggestions for Creative Design

(1) In part [C], show formatted columns of activities; with column headings

(2) Display wrong entries and discard them during the extraction of the activity list from the input file.

For validation:
- hours must be within 0..23
- minutes must be within 0 to 59
- activity durations (in minutes) must be positive.

Example:
```
09:70 30  Go to laundry shop
25:00 60  Swim
20:00 90  Movie
23:30 45  Long-distance call
08:20 30  Breakfast with Maggie
12:00 50  Lunch
19:00 45  Dinner at home
```

The data of these 2 activities are invalid.
(Also consider the end time of activity.)

(3) Sorting based on different values (eg. description, duration) according to the user's choice
Hint: You may mark such a comparison-mode using a global variable (or a special class for such setting(s)). Then the **`> operator`** can refer to this comparison-mode during the sorting process.

(4) Extend the duration of an activity (add a member function which adds the activity with some minutes).

(5) Allow the user to add activities (check conflicts of timing)

(6) Show some statistics (say longest activity time, total time of activities, etc..)

(7) Others - *see what you can imagine and implement…*

If you can implement all of the above (or other designs with similar difficulties) with good programming styles, you will get A++. Grading will be based on how good and how much you can submit. You are also welcome to talk to me about it.

Reminder: - For Creative Design, you need to submit the whole program (including **`main()`**) to PASS.
- You may upload extra input files of test cases for testing / demonstration of your program.

-- end --