Project 3: A Scheduling Program

Project 3 is due 04-30-14

Projects in CSE 251 are individual activities. You may not collaborate or work with any other students on the projects. You may, of course, contact course staff with questions or for help. But, your work must be entirely your own.

You are to write a simple scheduling program for this project.

When you start the program, it will display the following menu:

```
cbowen@ubuntu:~/workspace251/Scheduler/Debug$ ./Scheduler
1 - Insert a new event
2 - Display all events
3 - Now?
4 - Delete expired
0 - Exit
Please select an option:
```

If the user enters a zero, the program exits.

In this document I have included some sample functions from the reference solution. You are welcome to use these functions in your solution. Do not assume they can always be pasted directly without modification due to the variable names or data structures you have chosen to use.

Menu Options

The following sections describe each of the menu options.

1 - Insert a new event

When this menu option is selected, the user enters an event like this:

```
Please select an option: 1
What is the event: CSE 251
Event date: 4/14/2011
Start time: 3:00pm
End time: 4:50pm
```

The event description is up to 50 characters and many include spaces. The date is in the form indicated: month/day/year. The user must use a full year (2011) as opposed to a 2 character year (11). Also, only dates after 1970 are valid in the time system we will use. The start time and end time are in 12 hour format with hours and minutes only. See the section on Dates and Times for details on how to input and utilize this information in the program.

When a new event is inserted it should be tested against all other events and an overlap indicated if one occurs. Two events A and B do not overlap if A ends before B starts or A starts after B ends. Here is an example of the output should an overlap occur:

```
Please select an option: 1
What is the event: Afternoon Tea
Event date: 4/14/2011
Start time: 4:00pm
End time: 4:15pm
Warning, this event overlaps:
 4/14/2011  3:00PM  4:50PM CSE 251
```

Overlapping events are **not** rejected, they are entered into the system. The warning message should include all events that overlap the newly entered event. The warning message must only appear if an overlap exists.

2 - Display all events

When this menu options is selected, all events are displays in start time order. Example output is:

```
Schedule:
 4/14/2011  3:00PM  4:50PM CSE 251
 4/14/2011  4:00PM  4:15PM Afternoon Tea
```

3 - Now?

When the Now menu option is selected, any events that are currently active should be displayed. An event is currently active if the current time is greater than or equal to the start time and less than or equal to the end time. As an example:

```
Please select an option: 3
Currently active events:
 4/14/2011  2:00PM  3:00PM Prepare for CSE 251
```

4 - Delete expired

If this menu option is selected, any events that have expired, meaning they ended before the current time, are deleted from the scheduler. The deleted events should be displayed:

```
Please select an option: 4
Deleting:
 4/14/2011 12:00PM  1:00PM Lunch
```

If there is nothing to delete, the program should indicate that there are no expired events:

```
Please select an option: 4
No expired events
```

Robust and Safe Input

All program input must be robust and safe. If a user enters an invalid value, ask again. There should be no way to crash the program or get it to behave in strange ways with invalid input. Some examples:

If someone enters a bad menu option, it should request the menu option again.

If someone enters a bad date or time, it should request the date or time again.

You do not need to have detailed error messages other than requesting the option again.

Dates and Times

This program makes use of C language features for dates and times. There are several ways the C represents a date and time. The two most common are **time_t** and **struct tm**.

time_t

The most basic representation of a date and time is the type **time_t**. The value of a time_t variable is the number of seconds since January 1, 1970, sometimes call the Unix epoch. This is the best way to internally represent the start and end times for an event because it is easy to compare these values. Two time_t values a and b can be compared using <:

```
if(a < b)
{
    printf("Time a is before time b\n");
}
```

It is recommended that you use a time_t value to represent the start date and time of the event and another to represent the end date and time. It is much easier to represent the date and time together than to have different values for them.

struct tm

While time_t represents a date and time as a single number, struct tm represents it as a struct with a lot of numbers:

```
struct tm
{
  int tm_sec;                   /* Seconds.     [0-60] (1 leap second) */
  int tm_min;                   /* Minutes.     [0-59] */
  int tm_hour;                  /* Hours.       [0-23] */
  int tm_mday;                  /* Day.         [1-31] */
  int tm_mon;                   /* Month.       [0-11] */
  int tm_year;                  /* Year - 1900.  */
  int tm_wday;                  /* Day of week. [0-6] */
  int tm_yday;                  /* Days in year.[0-365] */
  int tm_isdst;                 /* DST.         [-1/0/1]*/
};
```

Conversion

You can convert a time_t value to a struct tm value using the **localtime** function:

```
    struct tm startTM;
    time_t start;
    /* ... */

    startTM = *localtime(&start);
```

You can convert a struct tm value to a time_t value using the **mktime** function:

```
    struct tm startTM;
    time_t start;

    /* ... */

    start = mktime(&startTM);
```

You will see examples of the functions in the code below.

How to input a date

Your program will need to input a date. Here is an example of a safe and robust way to input a date:

```
time_t InputDate(char *prompt)
{
    char buffer[100];
    char *result;
    struct tm date;

    do
    {
        printf("%s", prompt);

        /* Get a line of up to 100 characters */
        fgets(buffer, sizeof(buffer), stdin);

        /* Remove any \n we may have input */
        if(strlen(buffer) > 0)
            buffer[strlen(buffer)-1] = '\0';

        result = strptime(buffer, "%m/%d/%Y", &date);

    } while(result == NULL);

    /* Convert to time_t format */
    date.tm_min = 0;
    date.tm_hour = 0;
    date.tm_sec = 0;
    date.tm_isdst = 1;

    return mktime(&date);
}
```

This function asks the user for a time with the given prompt. If the time is correctly entered, it converts it to type time_t, expressing a time and date of 12:00am on the entered date.

In order to use the strptime function, you will need to #include <time.h>. However, you must preceed the #include with this code:

```
#define __USE_XOPEN
#include <time.h>
```

Any time you #include <time.h> in your program, use the line #define __USE_XOPEN before it. Note that it begins with two underlines.

How to input a time

After entering the date, you can enter a time using this safe and robust function:

```
time_t InputTime(char *prompt, time_t date)
{
    char buffer[100];
    char *result;
    struct tm time;

    time = *localtime(&date);

    do
    {
        printf("%s", prompt);

        /* Get a line of up to 100 characters */
        fgets(buffer, sizeof(buffer), stdin);

        /* Remove any \n we may have input */
        if(strlen(buffer) > 0)
            buffer[strlen(buffer)-1] = '\0';

        result = strptime(buffer, "%I:%M%p", &time);

    } while(result == NULL);

    return mktime(&time);
}
```

This function is passed a date expressed as a time_t value, the output of the InputDate function. The time is entered and added to the date to produce a date and time value that is returned. Here is an example of how these functions can be used to input a date, start time, and end time.

```
    time_t date;
    time_t start;
    time_t end;

    /*...*/

    date = InputDate("Event date: ");
    start = InputTime("Start time: ", date);
    end = InputTime("End time: ", date);
```

How to output a time_t value.

To output a time or date, you convert it to a **struct tm**:

```
    struct tm startTM;
    time_t start;

    /* ... */

    startTM = *localtime(&start);
```

Once converted, the value in struct tm are:

```
struct tm
{
  int tm_sec;                     /* Seconds.     [0-60] (1 leap second) */
  int tm_min;                     /* Minutes.     [0-59] */
  int tm_hour;                    /* Hours.       [0-23] */
  int tm_mday;                    /* Day.         [1-31] */
  int tm_mon;                     /* Month.       [0-11] */
  int tm_year;                    /* Year - 1900.  */
  int tm_wday;                    /* Day of week. [0-6] */
  int tm_yday;                    /* Days in year.[0-365] */
  int tm_isdst;                   /* DST.         [-1/0/1]*/
};
```

Now I can get the hour in the example as startTM.tm_hour. There are a couple strange things about this struct. To get the actual year, use startTM.tm_year + 1900. To get the actual month number, use startTM.tm_mon + 1.

You are required to output a date in the form: 4/15/2011. This is month/day/year. You are required to output a time in the form: 12:35AM. Note that the hour in **struct tm** is in 24 hour time. You will need to convert the time to 12 hour time. There is no function to do this. You must do this conversion yourself.

The function ctime

The function **ctime** converts a time_t value into a string of the following form:

```
Thu Apr 14 14:00:00 2011
```

This is a handy way to test to see if a time is what you expect. For example, if I have a variable: **time_t timeDate** and want to know what its value is, I can add this code while debugging:

```
printf("The date/time is %s\n", ctime(&timeDate));
```

This is also the easiest way to save the start date/time and end date/time to a file.

Reading the output of ctime

To read the output of ctime from a file and convert it back to a time_t value, use this code:

```
        struct tm tim;
        time_t startTime;

        fgets(buffer, sizeof(buffer), file);
        strptime(buffer, "%a %b %d %H:%M:%S %Y", &tim);
        startTime = mktime(&tim);
```

This inputs the date/time string from a file into the character string buffer. It then used strptime to convert it to a struct tm structure. Then the funciton mktime converts the struct tm value to a time_t value.

File I/O

File I/O is optional this semester due to time constraints. If you want to look at it and/or implement the File I/O portion, click here to display it.

Program Organization

You must make your program modular. Do not put the entire program in one .c file. Use at least two different .c file (three if you implement File I/O), grouping functions into files and headers in logical ways (my reference solution used 4). For example, my solution has a file.c and file.h that contains the functions for file input and output and a display.c and display.h that contains the functions to display the schedule and events.

Include a Makefile to built your program. Your Makefile must include a **clean** target.

Additional Requirements

You are not allowed to use global variables in this project. Repeat: no global variables are allowed in this project.

The easiest solution for passing around the schedule is to put it into a struct. Then you can pass it by reference like this:

```
void DisplayEvents(Schedule *schedule)
{
    int i;

    printf("\nSchedule:\n");
    for(i=0;  i<schedule->numEvents;  i++)
    {
        DisplayEvent(&schedule->events[i]);
    }

    printf("\n");
}
```

Hints and Suggestions

It is much easier to always keep the events in order by start time than to try to sort them when they are displayed.

You can use linked lists for this assignment, but using an array created using malloc and expanded using realloc as we did in the transistors step assignment is probably easier.

It is easiest to just overwrite the schedule file when the schedule changes rather than trying to update the file.

Write one thing at a time and get it working. Don't write a lot of code and then try to get it to compile and then try to get it to run. You probably never will get it to work. Do as little at a time as you can. For example, I started out with a menu that only had one item in it and got that to work first. Some of the other functions I just tested by themselves until I had them working, then moved on to other things.

Testing using an input data file

It can get very tedious inputting events over and over again as you test the program. You can create a test input file containing all of the keystrokes you would do on input and use this for testing. For example, I created a file named input.dat with this content:

```
1
CSE 251
4/28/2011
3:00pm
4:50pm
1
Example Event
4/29/2011
1:00pm
2:00pm
2
0
```

To use this, I run the scheduler program like this:

```
cbowen@ubuntu:~/workspace251/Scheduler$ ./scheduler <input.dat
```

The < sign means get the input from this file instead of the keyboard. Now, this looks a bit strange as you run it, but it saves a lot of keystrokes. Since the file does not echo to the terminal, you won't see the input. Here's what my run looks like for this test input file:

```
cbowen@ubuntu:~/workspace251/Scheduler$ ./scheduler <input.dat
1 - Insert a new event
2 - Display all events
3 - Now?
4 - Delete expired
0 - Exit
Please select an option: What is the event: Event date: Start time: End time: 1 - Insert a new event
2 - Display all events
3 - Now?
4 - Delete expired
0 - Exit
Please select an option: What is the event: Event date: Start time: End time: 1 - Insert a new event
2 - Display all events
3 - Now?
4 - Delete expired
0 - Exit
Please select an option:
Schedule:
 4/28/2011  3:00PM  4:50PM CSE 251
 4/29/2011  1:00PM  2:00PM Example Event

1 - Insert a new event
2 - Display all events
3 - Now?
4 - Delete expired
0 - Exit
Please select an option: Thank you for using the scheduler
```

Be sure you send your data file with a zero like I did so the program exits. Otherwise, your program will loop forever expecting input that never arrives:

```
ase select an option: Please select an option: Please select an option: Please
select an option: Please select an option: Please select an option: Please sel
ect an option: Please select an option: Please select an option: Please select
 an option: Please se^C
```

You can prevent this from happening if you will add this code to the loop where you input the menu option:

```
        if(feof(stdin))
            return 0;
```

The feof function tests for the end of the file on the standard input. If we are at the end of the file (no more input), this returns the zero menu option and the program exits. You may need to use this in other places as well if you run into debugging problems.

Submission

When you are done, do the following:

A: Do a **make clean** to remove the executable and intermediate object files.

B: Type these commands, assuming your project is in a folder scheduler:

```
cd ..
tar cvzf project3.tar.gz scheduler
```

This will create a file project3.tar.gz. This is the file you will turn in.

Turn in project3.tar.gz via http://www.cse.msu.edu/handin.

CSE 251