CST8234 – C Language and Network Programming

Assignment #2 – Student Exam Scheduler

DUE: Sunday, November 30th at 11:59 p.m. (i.e. 1 minute to midnight)

Purpose

The purpose of this assignment is to:

- Explore C's user-defined types, which are the forerunner of classes in C++;
- Use bit-wise operations to speed the execution of a program;
- Gain experience in the construction of an executable program consisting of multiple components, which requires advanced planning and forethought

Description of the Problem to be Solved

Despite the speed of modern computers, computationally intensive problems remain. This ensures that there will always be a role for fast languages like C. In many cases, compiled languages (such as Java) are too slow to handle many of these problems, and C (and assembler) are the only way to solve such problems in a practical amount of time.

Consider, for example, the following problem. Imagine there are 1000 students taking a first year Computer Hardware course. These students are drawn from many disciplines within computer studies and they will each have their own very different exam schedules. But now assume that they all have to take their Computer Hardware exam in the same 2 hour block of time. How does a scheduler find that 2-hour block of time amid all the conflicting schedules (assuming such a block exists)? That's the problem your program will resolve.

Program Methodology

1. To begin, `typedef` a user-defined type--call it the `studentSchedule` type—that contains an array that holds all the exam times over a two week

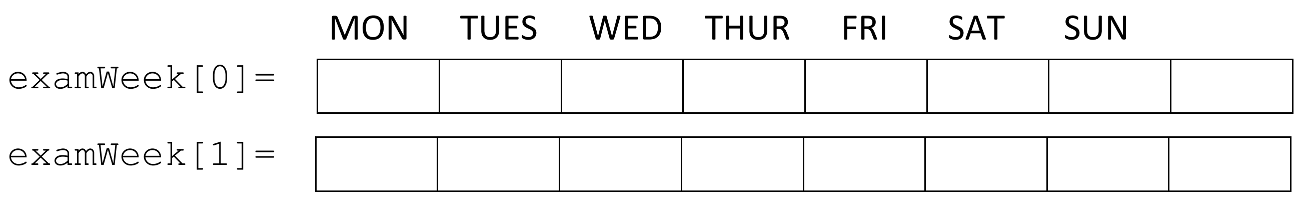
period in binary form, as an array of two `long long unsigned integer` values (see below for clarification).

2. Your program will need to declare an array of 1000 such types. Assume the student's number corresponds to the index of each element of the array.
3. The array that stores each exam schedule in the `studentSchedule` UDT could be defined as follows:

```
long long unsigned int examWeek[2] = ... ;
```

   but you should `typedef` this data type as a `lluInt` to simplify the coding that follows.

4. The actual encoding of the two week exam period works like this. Each `lluInt` encodes one week; each byte inside a `lluInt` encodes one eight hour day, with each bit standing for a one hour period between 9 am and 5 pm. If the time is booked, a 1 appears at the appropriate location in the array; if the time is free, a zero is stored. So imagine the two week exam period is saved as follows (where each rectangular block stands for one byte of data, or 8-hours of booking information):



   Note that the lowest byte corresponds to a non-existent day, so we will ignore this location in the calculations that follow.
   Written as binary values this means that each `examWeek` element will have an initial value of

```
  MON      TUES     WED     THUR      FRI      SAT      SUN
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

5. Whenever an hour is “booked”, a ‘1’ appears in the location corresponding to that hour. So if a student has a two hour exam from 9 a.m. to 11 a.m. on Tuesday morning, this would appear as:

   MON TUES WED THUR FRI SAT SUN

   00000000 **11**000000 00000000 00000000 00000000 00000000 00000000 00000000

   If the student’s exam schedule was all booked into one week, and it looked like this

Course	Day	Exam Time
C Programming:	M:	10:00 – 13:00
Web Development:	W:	15:00 – 17:00
Technical Writing:	Th:	12:00 -14:00
Data Structures:	F:	11:00-13:00
Computer Architecture:	T:	9:00-11:00

   then for that week, the value stored in the `examWeek` array would be:

   MON TUES WED THUR FRI SAT SUN

   0**111**0000 **11**000000 000000**11** 000**11**000 00**11**0000 00000000 00000000 00000000

   Of course, your program will need to store information over a *two week* period (using both `examWeek[0]` and `examWeek[1]`).

   Note: assume that exams can be booked on weekends, not just weekdays.

6. To reserve a time in one of the `lluInt` array elements, your code needs to be able to translate between the decimal values input by the user (the weeks, days, hours, and length when an exam is booked) and the binary value that needs to be stored in the array element. To translate between the two, consider the following code for reserving a block of time, given the

`day` of the week (Monday = 0, Tuesday = 1, …Sunday = 6), the starting `time` (using the 24 hour clock, i.e. 15:00 = 3 p.m.) , and the `length` of the booking, in hours.

```
#define lluBits (8*sizeof(long long int))

lluInt bookingMask = 0xFFFFFFFFFFFFFFFF;

// Set date and time for Tues exam at 10 a.m., for 3 hours
int day=1, time = 10, length = 3, initShift;

// Earliest starting time is 9 a.m.; so decr time by 9;
time -= 9;

// We need 3 1's in leftmost position, so shift away
// unwanted bits to the left.
//Note: this is much easier in Java! It has a >>> operator
initShift = lluBits - length;  //lluBits defined above; =64
bookingMask <<= initShift; // For 3 1's, shift left by 61
                           // =1110000000000000000000000...

// left fill with 0's up to starting time
bookingMask >>= time;      // =011100000000000000000000...

// left fill with 8 0's for each day up to date
bookingMask >>= (day*8);   // = 0000000001110000000000...

showBinary(bookingMask); // see code at end of document
```

The value of `bookingMask` is the value we wish to save into the `examWeek` array, for the appropriate week.

7. To store and retrieve this information, you'll want to put all the essential functions into a library called `examSchedule.c`. This should have a suitable .h file to 'advertise' the functions contained in the .c library to the outside world. `examSchedule.c` should contain the following four functions:

```
lluInt getBinExamTime(int studentNum, int day, int
hour, int length);
// A function that returns the binary value shown in
// the calculation above, given the day, hour, length
// of the exam, i.e. using the code provided

unsigned int setBookingTime(int studentNum, int week,
int day, int hour, int length);
// A function that takes the week (0 or 1), day, hour,
// and length of exam, and attempts to store this
// information into the examWeek array using the
// getBinExamTime function above.  You code must check
// for possible overbookings i.e. before you book a new
// exam time in the schedule, you must make certain
// there is no conflict with a pre-existing booking.
// Your code should return a 1 if successful, a 0
// otherwise.

lluInt getBinExamWeekInfo(int studentNum, int week);
// A function that returns the binary value stored in
// the examWeek array

void getUnscheduledTime();
// Finally, you'll need a function that calls the
// previous function for each of the 1000 students, and
// determines the times of during each week when no
// student has any exams (assuming such a time exists).
// You will first need to figure out which binary
// operation to perform on each of the two examWeek
// elements such that the result indicates when no
// classes are booked.  Next, you will need to
// extract from the resultant lluInt value the
// week, day, time and length using an operation that
// is essentially works the opposite of the
// getBinExamTime() function shown above.
```

This last function should output the information directly to the screen, e.g.

```
The following times are available for scheduling an
exam:

Week 0:
    M:  10:00-12:00
    W:  12:00-14:00
    Th: 13:00-17:00

Week 1
     (etc.)
```

<u>Calling your code from `main()`</u>

A 2-D array will be provided in a separate file with the reservation bookings for each of the 1000 students. The format of the array will be:

```
int examBookings[][5] = {
        {studentNumber, week, day, time, length},
        {studentNumber, week, day, time, length},
        {studentNumber, week, day, time, length},
        {studentNumber, week, day, time, length},
        etc.
}
```

where `studentNumber` indicates the number of the student (which is just the array index), and `week`, `day`, `time`, and `length` will indicate exam times for each student to book. Thus your code will need to load the exam reservations for each student by calling the **`setBookingTime()`** function (described above) for each row of the 2D array provided. Once the times are booked, call the **`getUnscheduledTime()`** function to output the free times available during the 2-week exam period for the Computer Hardware exam.

Marking

1. Code executes successfully.	**/1**
2. Followed all the instructions in this document (along with late additions or changes as posted) related to the functionality of the code, where such information was specified; did not attempt to subvert any of the general instructions nor the spirit of the assignment. This includes, e.g. using .h files where requested or implied to do so.	**/9**
3. Documentation. The code is clearly labelled according to the documentation standard provided.	**/4**
4. Clarity of code. Terse is good, but not to the point of incomprehensibility. Clearly, bloated, unnecessary code is to be avoided at all costs. If your code looks ugly, then it probably is ugly; it needs to be rewritten.	**/6**
5. Required pseudocode was provided that clearly reflects the actual operation of the code requested.	**/6**
6. The makefile functions as required and clearly specifies a rule for each dependency within your code, but does not imply unneeded dependencies, e.g. .h files which are already included inside .c files	**/4**
Total:	**30**

Submission Requirements

- All submitted code should be documented according to the 'CST8234 Documentation Standard' (available in Canvas)
- Your must provide a makefile that, when run, creates properly compiled and executable output. Your `main()` function, described above, should be in `Main.c`; `examSchedule.c` will contain the four functions listed above (along with any other support functions your deem necessary);

examSchedule.h will contain extern references to these four functions; and finally the file that contains the reservation information for each student (forthcoming).

Zip and ship your complete code to me using the link in Canvas no later than November 30th, at midnight. Your file name should have the following format:

CST8234_Assignment2_*YourLastName*_*YourFirstName*.zip

Notes

1. While C allows you to output numbers in hexadecimal and octal, it lacks a binary output display mechanism. The following code may therefore be of some use, if you want to view the binary contents of a `long long unsigned int`:

```
void showBinary(lluInt myLLuVal){
   lluInt mask = 0x8000000000000000;
   for (int ctr=0; ctr < lluBits; ctr++){
      printf((myLLuVal & mask)?"1":"0");
      mask >>= 1;
   }
   printf("\n");
}
```

2. All the code in this assignment has been tested on a PC that has 8-byte `long long unsigned int` data types. If `sizeof(long long int)` does not return 8 bytes on your PC, you may need to modify some of the code in this lab.

3. Throughout this assignment, you will need to employ bitwise ANDs (&), ORs (|) and XORs (^) to check or set the status of the bits in the `examWeek[]` arrays.

4. Each student will receive a file of student exam booking data that is unique. Therefore, your output may differ from your neighbour's.