Summary: This assignment explores the use of basic conditional execution, nested iteration, and memory accesses to analyze time/space data such as information collected by GPS-enabled applications In particular, you will write a program that analyzes two timelines, each of which gives a history of major cities in which a person has lived in chronological order. Your program will determine the total number of years in which both people lived in the same city at the same time.

Data in each timeline TL is stored as a sparse vector of ten elements in the following format:

```
   TL[i*2] = Year the person moved, for i=0,1,..., 4
 TL[i*2+1] = City to which the person moved, for i=0,1,..., 4
```

Each year is an integer in the range 1986 to 2015, inclusive, and each city is an integer between 0 and 9, inclusive. Assume each person has moved at most once in any given year. Also, assume that the total number of moves in each timeline is exactly five. For example, the timelines shown in Figure 1 are represented as

HarryTimeline[] = {1986, 2, 2003, 9, 2005, 3, 2007, 4, 2014, 7};

SallyTimeline[] = {1992, 4, 1996, 3, 2000, 2, 2005, 4, 2013, 3};

For this example, your program should compute 9 as the correct answer.

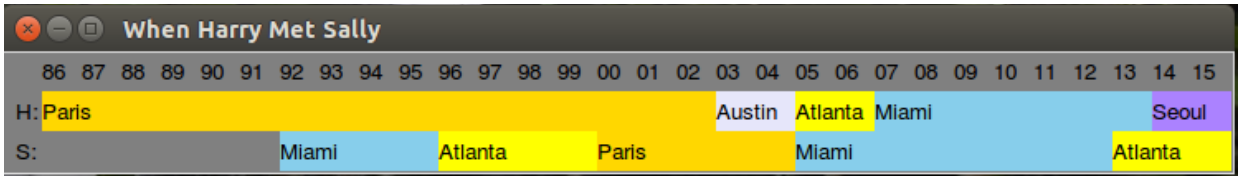

Figure 1: Timelines showing when Harry and Sally lived in certain cites. The total number of years in which they both lived in the same city at the same time is 9 (Paris for 3 and Miami for 6 years).

Note that this sparse vector representation is much more storage efficient than storing the city for each time point in the range 1986-2015.

For this assignment, you will write two programs, one in C and one in MIPS, as described below. You should design, implement, and test your own code. Otherwise you won't learn the things you need to know for later parts of projects one and two. **Any submitted project containing code not fully created and debugged by the student constitutes academic misconduct.**

HW2-1: For this part, design and implement a C program to compute and print the total number of years in which both people lived in the same city at the same time. If there is no year in which they are both in the same city, print 0.

As a starting point, use the file `HW2-1-shell.c`. This program includes a reader function `Load_Mem` that loads the values from a text file. You should use `gcc` under Linux/Unix to develop your program. Sample test files (`test1.txt, test0.txt, test9.txt`) are provided – the number in the name of each file indicates the correct answer. You should compile and run your program using the Linux command lines:

```
> gcc HW2-1.c -g -Wall -o HW2-1
> ./HW2-1 test1.txt
```

Name the file **`HW2-1.C`** and submit the file on T-square by **5:00pm on Friday, 16 September 2016**.

You can create additional test files for your C program using MiSaSiM in this way: run `HW2-2-shell.asm`, go to the end of the trace, and use the "Dump" memory menu button to save the memory to a text file with the correct answer (the value in $3) in the name of the file.

HW2-1: For this part, design and implement a MIPS version of your program. A description of the MIPS library routines you need is given below. Use the file `HW2-2-shell.asm` as a starting point. The result should be stored by your program in **$2** and reported as an answer by software interrupt **swi 590** as described below. You must use these register conventions or the automated grader will score your program incorrectly. Memory should only be used for input to your program. Your program must return to the operating system via the **`jr`** instruction. *Programs that include infinite loops or produce simulator warnings or errors will receive zero credit*. Name the file **`HW2-2.asm`** and submit the file **on T-square by 5:00pm on Friday, 16 September 2016.**

MIPS Library Routine: There are two library routines (accessible via the `swi` instruction).

SWI 586: Create Timelines: This routine generates and displays the timelines as shown in the example in Figure 1. It initializes memory with the 20 integers beginning at the specified base address (e.g., `Harry`). INPUTS: $1 should contain the base address of the 20 words (80 bytes total) already allocated in memory. OUTPUTS: the 20 words allocated in memory contain the 20 sparse vector elements (alternating year and city) to be used as input data.

As a debugging feature, you can load in a previously dumped testcase (pair of timelines) by putting -1 into register $2 before you call swi 586. This will tell swi 586 to prompt for an input txt file that contains a testcase (e.g., test9.txt).

SWI 590: Report Total Overlap: This routine allows you to report your answer and an oracle provides the correct answer so that you can validate your code during testing.

INPUTS: **$2** should contain the total number of years you computed as your answer.

OUTPUTS: **$3** gives the correct answer.

If you call **swi 590** more than once in your code, only the first answer that you provide will be recorded.