Notes on Computer Project #11

Comments about the assignment and responses to frequently asked questions will be added to this file as necessary.

***** comments added on 11/14/18 *****

1) As stated on the assignment handout, you are required to create a makefile which controls the translation of your program, and the name of your executable files must be "proj11".

Since you must link the instructor-supplied driver module with your support module, your makefile should include the appropriate steps to assemble your version of "proj11.support.s", as well as link the object code modules.

2) Please note the following suggestion:

  You may wish to create a stub for the required function, then translate, link and execute the program to explore the behavior of the driver module.

3) A couple of definitions about hockey statistics:

A player's points is defined as the sum of that player's goals and assists.

A player's points per game (a real number) is defined as number of points divided by number of games played. A player who has not participated in any games is defined to have zero points per game.

4) Be sure that you have the correct layout of both "struct player" and "struct table" in memory. For example, you must account for any bytes of padding to keep certain fields aligned on the appropriate byte boundary in memory.

5) Note that your program must work correctly for any properly formatted data set. An example of a properly formatted data file is available as:

```
  /user/cse320/Projects/project11.data
```

Clearly, you will need to develop several data files to test various aspects of your solution, since the simple instructor-supplied data file will not be sufficient for all of the cases that you will want to test.

6) Please note that the "r" option in the instructor-supplied driver module does not rely on any of your functions to perform its work.

7) Please note the function prototype below:

```
  int search( struct table*, unsigned long, struct player** );
```

The third argument to function "search" is a pointer to a pointer (an address where an address can be stored by the function).

Any function which calls function "search" is responsible for allocating four bytes of memory and sending the address of that four-byte area of memory as the third argument to "search".

Those four bytes can be allocated in the ".data" section (by a ".word" or ".skip" instruction) or in the run-time stack (by the "save" instruction which is the prologue for that function).

8) Please note the following from the assignment handout:

  The program will use an ordered table to maintain the data set, where each
  player's jersey number will serve as a unique key to identify that player.

That is, the table is ordered by jersey number, from smallest number to largest number. All operations on the table (such as inserts and deletes) must keep the table elements in the proper order.

When you insert a new item, you must keep the table in order. To do so, you need to "open up" a slot in the table at the right place, by moving some records down one slot each.

When you delete an item, you also need to keep the table in order. To do so, you need to move some records up one slot each (to write over the top of the deleted record).

If you are unable to maintain the table in order, your solution will still be worth partial credit. Under those circumstances, you can take some short cuts: put a new record at the end of the table (for "insert"), and use the last record to replace a record being removed (for "delete"). Of course, you'll also have to modify your "search" function so that it no longer assumes that the table is ordered by key.

To repeat: for full credit, your solution will maintain the table in order by key (jersey number). Solutions which do not maintain the table in order will be evaluated for partial credit.

9) Please note the function prototype below:

```
  int insert( struct table*, unsigned long, char*, int, int, int );
```

Note that function "insert" accepts six arguments. The first four arguments are passed in registers (as usual), but the other two arguments are passed in the stack.

10) Please recall the definition of a C-style character string: an array of type "char" with a null byte to mark the end of the sequence of characters. To copy a character string from one location in memory to another, load a byte from the source array and store that byte in the destination array, until you have copied all of the characters (including the null byte which marks the end of the source string).

11) The system function "memmove" is useful to copy blocks of data from one area of memory to another. See the example in:

```
  /user/cse320/Projects/project11.memmove.s
```

More information is available via "man memmove".

12) Each player's record contains the following field:

```
  float points_per_game;    /* points per game played */
```

A player's points per game (a real number) is defined as number of points divided by number of games played.

The number of points and the number of games played are integer values, so they will have to be converted into floating point values before computing the average. In C, you would do something like:

```
  average = float(total) / float(games);
```

See the example in:

```
  /user/cse320/Projects/project11.fdiv.s
```

The comments in the source code describe how to translate and execute the program.

--M. McCullen