Project 3: External Records

Due: Tue 10/07/03, Section 0101 (Chang) & Section 0301 (Macneil)
Wed 10/08/03, Section 0201 (Patel & Bourner)

Objective

The objective of this programming project is to gain experience writing more complex assembly language programs and to use indexed addressing modes.

Assignment

Your assembly language program for this project will work with an externally defined array of records. This array is defined in a C program as follows:

```
struct {
    char   realname[32] ;
    char   nickname[16] ;
    char   alignment[20] ;
    char   role[20] ;
    int    points ;
    int    level ;

} records[10] ;

int num_records = 10 ;
```

The records in the array have pre-initialized values not shown here. The full text of the C program is available on the GL file system at: `/afs/umbc.edu/users/c/h/chang/pub/cs313/records.c`

Your assembly language program must search through the array and find the record with the least number of points and the record with the alphabetically first nickname. It must then print out the `realname` field of these two records. E.g.,

```
Lowest Points: James Pressman
First Nickname: Dan Gannett
```

Implementation Notes

- The sample data in `records.c` contains 10 records, but your program should work with any number of records. The number of records is stored in the `int` variable `num_records`.
- In order to access the externally defined array and integer variable, you must have the following declaration in your assembly language program:

```
extern      records, num_records
```

- You must also make your own test cases. The example in `records.c` does not fully exercise your program. Your program will be graded based upon other test cases.
- You will need to link your assembly language program with the data defined in the C program:

```
gcc -c records.c
nasm -f elf report.asm
ld records.o report.o
```

- An important part of this project is deciding how to use indexed addressing to access the data in the records. Think this through carefully. A clean and logical approach to this problem will yield clean and logical code that is easier to construct and, more importantly, easier to debug.

- Your program should be reasonably robust and report errors encountered (e.g., empty array) rather than crashing.
- Note that the strings stored in the array are C-style null-terminated strings.
- Nicknames should be compared using dictionary ordering. For example, any string starting with the letter 'a' comes before any string that starts with 'b'. In the case that one string is a prefix of another, the shorter string come first. E.g., "egg" comes before "egghead".
- To access each field of the record, you should use an offset from the address of the record. You should use `%define` constants instead of magic numbers. E.g.,

```
%define NickOffset 32
%define AlignOffset 48
%define RoleOffset 68
%define PointsOffset 88
%define LevelOffset 92
%define RecSize 96
```

- Project 4 will be based upon Project 3, so keep in mind that you will need to extend/modify this program.

Turning in your program

Use the UNIX `submit` command on the GL system to turn in your project. You should submit at least 4 files: your assembly language program, at least 2 of your own test cases and a typescript file of sample runs of your program. The class name for submit is `cs313_0101`, `cs313_0102` or `cs313_0103` for respectively sections 0101 (Chang), 0201 (Patel & Bourner) or 0301 (Macneil). The name of the assignment name is `proj3`. The UNIX command to do this should look something like:

```
submit cs313_0103 proj3 report.asm myrec1.c myrec2.c typescript
```