COMP 3958: Lab 4

Submit a source file named `sort.ml`. You may use functions in the standard modules (except for functions in the `Str` module, which basically handle regular expressions). You are also not allowed to use imperative features like `for` and `while` loops. Your file must build without warnings or errors. Otherwise, you may not receive credit for this lab. Maximum score: 15

Implement a program in OCaml to sort records read either from standard input or from a file. We'll call our program `sort`. If `sort` is invoked

- without any argument on the command-line, it reads from standard input;
- with one or more arguments on the command-line, then the argument after the program name is the file to read; the remaining arguments are simply ignored.

(For simplicity, you may let the program crash with an exception if the specified file cannot be opened.)

We'll use the following type for records:

```
type record = {id: string; score: int}  (* ID and score *)
```

Each record has an ID and a score. The ID must start with the character `'A'` followed by 8 digits; the score must be between 0 and 100 inclusive. Each input line may contain information for one record. An input lines may be invalid, either because the information is incomplete (e.g., no score) or the ID or score is invalid. Invalid input lines are simply skipped.

The following shows some sample input lines:

```
A12345678  5  # top student
  A66666666  99  ! high score
 Abc 88     # invalid ID
   A55555555 89a  !! invalid score
  A44444444 100
 A77777777
```

Note that the last line above is invalid as there is no score.

The first word (if there is one) in a line is regarded as the ID, the next word (if there is one) is regarded as the score. Additional words that follow are regarded as part of the comment and are ignored.

To get a record, the program needs to make sure that an input line has at least two words, that the first word is a valid ID and the second word a valid score.

Records are stored in a list. After finishing reading records (on end-of-file), the program proceeds to sort the list of records and then display the records to standard output. The sorted output is in descending order of scores, and if multiple records have the same score, they are then in ascending order of IDs.

The following shows the format of the output:

```
100 A44444444
 99 A66666666
  5 A12345678
```

Put your code in a file named `sort.ml`. We'll be building your program using the command: `ocamlbuild sort.native`. Make sure that this builds without warnings or errors.

To facilitate testing (and in case your program does not quite work), there must be the following functions:

```
val is_valid_id : string -> bool     (* test whether a string is a valid ID *)
val is_valid_score : int -> bool     (* test whether an integer is a valid score *)
val sort_records : record list -> record list (* sort list of records in specified order *)
val parse : string -> record option  (* get a valid record from a string if possible *)
```

Be sure to comment any part of your program that you think may not be clear to the reader.