CW-B3 Audioscrobbler Recommendations

Released: December 07 2016
Deadline: December 21 2016
Weight: 20 %

This coursework carries 20 % towards your final mark. A zip file `cw-b3-start.zip` is provided containing some partially completed C source files. Complete these files and a TWO page report in PDF file format as specified below. Create a zip file `cw-b3.zip` containg the updated files from the original zip file, and the report as a PDF file `report.pdf`. Submit this zip file by clicking the "Submit CW-B3" link on Moodle.

The marking scheme is as follows:

- ▷ Task 1: 4 marks
- ▷ Task 2: 4 marks
- ▷ Task 3: 4 marks
- ▷ Task 4: 8 marks

Introduction

In this coursework, we will bring together the pieces from the previous courseworks and implement a music recommendation system, including some helper programs.

The zip file `cw-b3-start.zip` contains a Makefile for building (compiling) the code in this coursework. You can build the programs by typing `make` on the Linux command line.

Task 1 - Sorting generic lists

We will adapt the merge sorting function from CW-B2 so that it operates on a linked list that can contain any value. Each element in the linked list has a pointer to a value (e.g. an artist struct or a play struct), and a pointer to a next element, as explained in Lecture 14. The pointer to the next element in the linked list is given by the `next` field in the `list` struct, and the `val` field is a pointer to the value stored in an element.

```
struct list
{
   void *val;
   struct list *next;
};
```

(a) The file `list.c` which is provided with the coursework contains the definition of the functions `add_elt` and `free_list`. Explain the working of these functions. (report).

(b) We have implemented a function which constructs a linked list of elements from a text file, one element for each line in the file.

```
struct list *read_list(char *file_name, void *(*parse)(char *));
```

Explain in your own words the working of this function. What is the role of the second argument to this function? Give examples of values that can be used as the second argument (report).

(c) Explain in your own words how the merge sort algorithm works. Discuss how this algorithm relates to the concept divide and conquer (report).

(d) We have adapted the implementation of `merge_sort` from CW-B2 to generic linked lists.

```
struct list *sort(struct list *x, int (*comp)(void *a, void *b));
```

Explain in your own words the working of this function. What is the role of the second argument to this function? Give examples of values that can be used as the second argument (report).

Task 2 - Binary search

(a) We have implemented a function that takes a linked list of artists, and returns an array with pointers to the elements in the list.

```
struct artist **create_index(struct list *head, int n);
```

Explain in your own words the working of this function (report).

(b) Draw a diagram that illustrates the array returned by the `create_index` function when given a linked list of length five. The drawing should show all struct objects involved, and indicate pointers as arrows (report).

(c) Create a function which given an artist id and an array of pointers to artists uses binary search to find the artist with the given artist id.

```
struct artist *lookup_artist(struct artist **index, int n, int artist_id);
```

Task 3 - Utility programs

The purpose of this task is to create a set of programs that computes useful information about the Audioscrobbler data set. The programs should print an appropriate error message, free allocated resources (memory and file), and halt if one of the following error occurs: memory allocation error, file error, and incorrect number of arguments provided.

(a) Create a program `find_artist` which prints the name of the artist or "Not found" if there is no such artist in the artist data base. The program should take the following two arguments:

1) artist id
2) the name of the artist data file

(b) Create a program `sort_artists` which reads the Audioscrobbler data files and outputs the artists sorted according to one of four sorting criteria. The program should take the following three arguments:

1) the name of the artist data file,
2) the name of the play data file, and
3) one of the following sorting criteria:

   - ▷ `inc_pc`: sort according to increasing play count
   - ▷ `dec_pc`: sort according to decreasing play count
   - ▷ `inc_id`: sort according to increasing artist id
   - ▷ `ec_id`: sort according to decreasing artist id

(c) Create a program `query_plays` which takes two or three arguments. The first argument is the name of the play data file, and the remaining arguments are one of the following commands:

- ▷ `c <artistid>` counts all the plays in the data base, or if the optional argument `<artistid>` is provided, only the plays corresponding to this artist.
- ▷ `p <artistid>` prints all the plays in the data base, or if the optional argument `<artistid>` is provided, only the plays corresponding to this artist.

Task 4 - Artist similarity

The goal of the final task is to create a program which recommends artists that are similar to an artist with a given artist id. The program should use a *similarity function* which is provided with the coursework. Given two artists, the similarity function returns a value between 0 and 1, where 1 means that the artists are completely similar, and 0 means that the artists are completely dissimilar.

We consider two artists to be similar if they have many fans in common. A user is a *fan* of an artist if the artist is among the 100 most played artists of the user. To store the list of fans, we will extend the definition of the artist struct as follows. The struct field `fans` is a pointer to a generic linked list, where the `val` field in each element points to a play struct.

```
struct artist
{
  int id;
  char name[NAMELEN];
  int count;
  float distance;
  struct list *fans;
};
```

(a) We have constructed a function which adds the list of fans to each artist.

```
void register_fans(struct artist **index, int n, struct play *plays);
```

Explain in your own words the working of this function (report).

(b) Explain how the similarity function computes the similarity between two artists (report).

```
float similarity(struct artist *a, struct artist *b);
```

(c) Design a function which given an artist a, sorts the artist list according to increasing distance to the artist a.

```
struct list *sort_similarity(struct artist *a, struct list *artists);
```

Hint: Use the `similarity` function to compute the similarity value between artist a and each of the artists in the linked list. Store each similarity value in the field `similarity` in the artist struct. Then use the function `sort` from Task 1 (d).

(d) Implement an interactive program `recommend` which produces music recommendations. This task is open ended. You can decide how the user should interact with the program. Hint: Consider the first elements in the list generated by the function `sort_similarity`.

(e) Describe what your program `recommend` does, and its working (report).