moviedatabase.c

```
#include "moviedatabase.h"

#include <stdio.h>
#include <stdint.h>
#include <string.h> // strcmp

#include "llist.h"
#include "csv.h"
#include "alloc.h"
#include "film.h"

LinkedList* films = NULL;

void mdb_loadDB(const char* const path)
{
  size_t entries;
  LinkedList** csv = csv_read(path, &entries);

  if (!csv)
  {
    fprintf(stderr, "Failed to read database file\n");
    return;
  }

  if (!films)
    films = ll_new();
  if (!films)
  {
    fprintf(stderr, "Failed to allocate memory for film db\n");
    exit(EXIT_FAILURE);
  }

  for (size_t i = 0; i < entries; ++i)
  {
    LinkedList* entry = csv[i];
    LinkedIterator it = ll_it_begin(entry);
```

```
    char* title = (char*)ll_it_data(&it);
    uint16_t year = atoi((char*)ll_it_next(&it));
    char* rating = (char*)ll_it_next(&it);
    char* categories = (char*)ll_it_next(&it);
    uint16_t runtime = atoi((char*)ll_it_next(&it));
    double score = atof((char*)ll_it_next(&it));
    Film* film = film_new(title, year, rating_fromString(rating),
                          category_fromStrings(categories), runtime, score);
    ll_delete(entry);

    ll_push_back(films, film);
  }
  mt_free(csv);
}

void mdb_freeDB()
{
  if (!films) // No films? No work
    return;

  // delete each film manually
  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
    film_delete(ll_it_data(&it));
  // purge the linked list, all the data pointers are invalid now
  ll_purge(films);
}

void mdb_printAll()
{
  if (!films)
    return;

  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
    film_print(ll_it_data(&it));
}

int32_t alphanumeric(const void* const a, const void* const b)
{
  return strcmp(film_getTitle((Film*)a), film_getTitle((Film*)b));
}
int32_t chronological(const void* const a, const void* const b)
{
  int32_t ya = film_getYear((Film*)a);
  int32_t yb = film_getYear((Film*)b);
```

```
  // if equal, sort by title for consistent ordering
  return (yb - ya != 0) ? (yb - ya) : alphanumeric(a, b);
}
int32_t runtime(const void* const a, const void* const b)
{
  int32_t ra = film_getRuntime((Film*)a);
  int32_t rb = film_getRuntime((Film*)b);
  // if equal, sort by title for consistent ordering
  return (rb - ra != 0) ? (rb - ra) : alphanumeric(a, b);
}
int32_t score(const void* const a, const void* const b)
{
  double sa = film_getScore((Film*)a);
  double sb = film_getScore((Film*)b);
  if (sa < sb)
    return 1;
  if (sa > sb)
    return -1;
  // if equal, sort by title for consistent ordering
  return alphanumeric(a, b);
}

void task1()
{
  printf("\nTask 1 -- All films in chronological order:\n");
  ll_bsort(films, chronological);
  mdb_printAll();
}


void task2()
{
  printf("\nTask 2 -- 3rd longest Film-noir film:\n");
  LinkedList* noir = ll_new();

  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
  {
    Film* f = (Film*)ll_it_data(&it);
    if (film_hasCategory(f, FILM_NOIR))
      ll_push_front(noir, f);
  }

  ll_bsort(noir, runtime);
  LinkedIterator third = ll_at(noir, 2);
  film_print((Film*)ll_it_data(&third));
```

```
  ll_purge(noir);
}

void task3()
{
  printf("\nTask 3 -- 10th highest scoring Sci-fi film:\n");
  LinkedList* scifi = ll_new();

  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
  {
    Film* f = (Film*)ll_it_data(&it);
    if (film_hasCategory(f, SCI_FI))
      ll_push_front(scifi, f);
  }
  ll_bsort(scifi, score);
  LinkedIterator tenth = ll_at(scifi, 9);
  film_print(ll_it_data(&tenth));
  ll_purge(scifi);
}

void task4()
{
  printf("\nTask 4 -- Film with the highest score:\n");
  Film* highest = NULL;

  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
  {
    Film* f = (Film*)ll_it_data(&it);
    if (!highest || film_getScore(f) > film_getScore(highest))
      highest = f;
  }

  if (highest)
    film_print(highest);
}
void task5()
{
  printf("\nTask 5 -- Film with the shortest title:\n");
  Film* shortest = NULL;

  for (LinkedIterator it = ll_it_begin(films);
       ll_it_valid(&it); ll_it_next(&it))
  {
    Film* f = (Film*)ll_it_data(&it);
```

```
    if (!shortest || strlen(film_getTitle(f)) < strlen(film_getTitle(shortest)))
      shortest = f;
  }

  if (shortest)
    film_print(shortest);
}

void task6()
{
 printf("\nTask 6 -- Number of films in the database after deleting
all "
"R rated films:\n");
  LinkedIterator it = ll_it_begin(films);
  while (ll_it_valid(&it))
  {
    Film* f = (Film*)ll_it_data(&it);
    if (film_getRating(f) == R)
    {
      ll_erase(&it);
      film_delete(f);
      it = ll_it_begin(films); // erase invalidates the iterator
    }
    else
    {
      ll_it_next(&it);
    }
  }

 printf("After deleting all R rated films, there are %zu films.\n",
       ll_size(films));
}
```