Prácticas de Estructuras de Datos

David Arroyo Guardeño

Escuela Politécnica Superior de la Universidad Autónoma de Madrid

Tercera práctica I

- ✓ Entrega: semana del 12 de diciembre
- ✓ Contenido de la entrega (☠☠☠ fichero .tgz ☠☠☠)
  - ✗ Código fuente y Makefile
  - ✗ Directorio “libros_db” y todo su contenido
  - ✗ Memoria detallando el trabajo hecho y ejemplos que demuestren el funcionamiento del trabajo realizado
  - ✗ Descripción del diseño de alto nivel y de las pruebas efectuadas (funcionales y de liberación de recursos -valgrind!!!-)

Tercera práctica II

- Descripción del diseño a bajo nivel: comentarios en código y documentación (preferentemente mediante Doxygen)

Ejemplo manejo de ficheros binarios

http://arantxa.ii.uam.es/~darroyo/ejemplo_ficheros_binarios.c

Estructura base para el manejo de información: pares clave-valor

```
typedef struct
{
   char   nombre[MAX_NOMBRE];
   int  clave;
} registro_fichero;
```

Funciones base

```
/**
 *
 * Esta funcion annade la direccion relativa al indice para una cierta
     clave
 * @param indice Indice
 * @param clave Clave
 * @param dir_rel Direccion relativa
 */
void crear_indice(long indice[], int clave, long dir_rel )
{
  indice[clave] = dir_rel;
}


/**
 *
 * Esta funcion escribe en un fichero la estructura de entrada
 * @param fp Puntero al fichero en el que se desea escribir
 * @param regr Registro que se desea escribir en el fichero
 */
void escribir_registro(FILE * fp, registro_fichero regr)
{
  fwrite(&regr,sizeof(regr),1,fp);
}
```

Función principal I

```
fichero_reg = fopen("mfile","w");
  if(fichero_reg  == NULL){
    fprintf(stderr, "Error abriendo el fichero mfile en la linea %d del
        fichero %s\n",__LINE__,__FILE__);
    return EXIT_FAILURE;
  }

  dir_rel= 0 ;
  do{
    printf("Introduce el valor y la clave para el registro a annadir al
        fichero mfile\n");
    i = scanf("%s %d",frec.nombre,&frec.clave);
    if( i != 2 || errno){
      perror("Error al capturar datos de entrada");
      fclose(fichero_reg);
      return EXIT_FAILURE;
    }
    while(indice[frec.clave] != (-1)){
      printf("Ya existe una entrada con ese valor de clave\n");
      scanf("%s %d",frec.nombre,&frec.clave);
    }
    crear_indice(indice,frec.clave,dir_rel);
    escribir_registro(fichero_reg,frec);
    dir_rel =  ftell(fichero_reg); /* mediante esta operacion
```

Función principal II

```
                    establecemos la direccion relativa
                    para el siguiente registro como la
                    posicion actual del puntero a
                    fichero, medida en bytes y tomando como offset el
                        principio del fichero */

    printf("Pulsa 1 para continuar annadiendo registros al fichero\n");
    scanf("%d",&n);
}while(n == 1);
```

Main: salvamos el fichero con los índices y recuperamos registros I

```
ifile = fopen("fichero_indice","w");
  if(ifile == NULL){
    fprintf(stderr, "Error abriendo el fichero de indices en la linea %d
        del fichero %s\n",__LINE__,__FILE__);
    fclose(fichero_reg);
    return EXIT_FAILURE;
  }
  fwrite(indice,sizeof(indice),1,ifile);/*Escribe el indice completo en
      el fichero fichero_indice */

  fclose(fichero_reg);
  fclose(ifile);
  printf("Pulsa 1 si quieres recuperar un registro\n");
  scanf("%d",&n);
  if( n == 1){
    ifile=fopen("fichero_indice","r");
    if(ifile == NULL){
      fprintf(stderr,"Error al abrir el fichero de indices en la linea %d
          del fichero %s\n", __LINE__, __FILE__);
      return EXIT_FAILURE;
    }
```

Main: salvamos el fichero con los índices y recuperamos registros II

```
  fread(indice,sizeof(indice),1,ifile);/* Lee el indice completo del
      fichero de indices y lo escribe en el vector indice */

  fclose(ifile);
  fichero_reg = fopen("mfile","r");
  if(fichero_reg == NULL){
    fprintf(stderr,"Error al abrir el fichero de registros en la linea
        %d del fichero %s\n", __LINE__, __FILE__);
    return EXIT_FAILURE;
  }
}
printf("El contenido del fichero es\n");
while( (fread(&frec,sizeof(frec),1,fichero_reg)) != 0)
  printf("%s %d\n",frec.nombre,frec.clave);
do{
  printf("Introduce la clave del registro que quieres recuperar\n");
  scanf("%d",&clave);
  dir_rel = indice[clave]; /*obtiene la direccion relativa del registro
      almacenada en el vector de indices*/
  if( (fseek(fichero_reg,dir_rel,SEEK_SET))!= 0 ){
```

Main: salvamos el fichero con los índices y recuperamos registros III

```
    fprintf(stderr,"Error al recorrer el fichero de registros en la
        linea %d del fichero %s\n", __LINE__, __FILE__);
    fclose(fichero_reg);
    return EXIT_FAILURE;
  }
  fread(&frec,sizeof(frec),1,fichero_reg);
  printf("El valor almacenado en el registro recuperado es %s\n",frec.
      nombre);
  printf("Pulsa 1 para continuar recuperando registros\n");
  scanf("%d",&n);
} while(n == 1);
fclose(fichero_reg);
return EXIT_SUCCESS;
```

Ver el contenido de un fichero binario desde la shell I

```
hexdump fichero.bin
```

```
hexdump -C fichero.bin
```

```
xxd fichero.bin
```

Transformación de ASCII a binario:

```
echo -n "A" | xxd -b
```

Cambio de base:

```
man ascii
```

```
echo "obase=2;ibase=10;65" | bc
```

```
echo "obase=2;ibase=16;A" | bc
```

Ver el contenido de un fichero binario desde la shell II

```
echo "obase=16;ibase=2;1010" | bc
```

Ayuda:

```
man hexdump
```

```
man xxd
```

```
man bc
```

Introducción a esecuele I

1. Descargar de Moodle el fichero

   `esecuele_2016.tar`

2. Las fuentes sobre las que se trabajará están en la carpeta

   `esecuele/development`

   - ✗ type
   - ✗ parser
   - ✗ operation
   - ✗ database

3. Makefile

Introducción a esecuele II

- ✗ make esecuele: para compilar y enlazar vuestra implementación
- ✗ make wvs: para compilar y enlazar el ejemplo con la base de datos de bancos

4. Analizar el script `bank.bash`
5. Estudiar el código del fichero esecuele.c

Tareas I

1. Añadir los tipos double (DBL) y long (LNG)
   - ✗ `types/type.h`
   - ✗ Añadir dos nuevos casos a cada una de las funciones de la implementación (`types/types.c`) de esta interfaz
2. Implementar table.c/.h, record.c/.h $\rightarrow$ cliente de pruebas
3. Implementar las operaciones COUNT, UNION, LIMIT, OFFSET $\rightarrow$ validarlas empleando la base de datos de libros

Tareas II

4. Crear base de datos *libros_db* para almacenar los ficheros dados por las tablas de la base de datos que habéis creado en la práctica anterior. Hay que hacer uso de ESECUELE
   - ✗ Lista de libros comprados por “jack”
   - ✗ Número de libros comprados por “jack”
5. Opcional: implementar la operación JOIN sobre una sola columna, especificando en qué columna se quiere hacer el JOIN

esecuele.c I

1. `createdb` $\Rightarrow$ creación de una instancia de `database_t`
2. `define` $\Rightarrow$ se añaden una a una las tablas de la base de datos (según el esquema que figura en el enunciado de la práctica: formato de los registros)
   - ✗ Se crea un fichero binario por tabla
   - ✗ La cabecera contiene: número de columnas (un entero) seguido del tipo de cada columna
   - ✗ Tras la cabecera, aparecen cada uno de los registros
   - ✗ Por cada columna de un registro tenemos

esecuele.c II

- ⇒ Tamaño de la columna
- ⇒ Valor de la columna

3. `insert` ⇒ guarda en cada fichero de tablas los registros contenidos en los ficheros de datos dados como entrada
   - ✗ Para cada fichero de tablas, hay que saber dónde empiezan y dónde terminan los registros
   - ✗ Al abrir la base de datos se lee cada fichero de tablas y se carga en `database_t` (campo `tables`)

esecuele.c III

4. `query` $\Rightarrow$ para realizar consultas debe estar implementada la interfaz `operation`

database_copy

→ copia un fichero en una tabla

1. Recupero la instancia de `table_t` correspondiente con la info de cabecera de tabla

```
table = database_get_table(db, table_name);

types = table_types(table);
ncols = table_ncols(table);

row = malloc(sizeof (void*) * table_ncols(table));
c = 0;
```

database_copy I

```
while (fgets(line, MAX_LEN_ROW, file) != NULL){
    char* eol;
    /* lines starting with '#' are ignored */
    if (*line == '#') {
  continue;
    }
    /* removes the '\n' at the end of the line */
    eol = strchr(line, '\n');
    if (eol != NULL) {
  *eol = '\0';
    }
    i = 0;
    token = tokenizer(line);
    while (token != NULL && i < table_ncols(table)) {
  row[i] = value_parse(types[i], token);
  i++;
  token = tokenizer(NULL);
    }

    if (i != table_ncols(table)) {
  printf("%s\n", line);
    }

    /* get the position where the record will be inserted */
```

database_copy II

```
  pos = table_last_pos(table);
  /* insert the record at the end of the file */
  table_insert_record(table, row);
  /* updates the indexes */
  for (i = 0; i < ncols; i++) { /* for every column in the table */
index = database_get_index(db, table_name, i);
if (index != NULL) { /* if it is indexed, the index is updated */
  key = *((int*) row[i]);
  index_put(index, key, pos);
}
  }

  for (i = 0; i < ncols; i++) {
free(row[i]);
  }
  c++;
}
```

Comentarios sobre la interfaz table I

- ✓ El TAD asociado debe incorporar la info sobre la localización del fichero asociado:

```
if (strcmp(token, "TABLE")== 0)
```

en `esecuele.c` → `database_add_table`

- ✓ Además, los datos que devuelven las siguientes funciones definidas en la interfaz
  - ✗ `table_ncols`
  - ✗ `table_data_path`

Comentarios sobre la interfaz table II

- ✗ `table_types`
- ✗ `table_first_pos`
- ✗ `table_last_pos`

- ✓ `table_create` sólo se emplea para crear el fichero asociado a una tabla
- ✓ `table_open` crea e inicializa (de acuerdo con el fichero dado como entrada) una instancia de `table_t`

Comentarios sobre la interfaz record

- ✓ Sólo se emplea para “empaquetar” la información al leer del fichero
- ✓ Sólo `table_read_record` devuelve una instancia de `record_t`

Comentarios sobre la interfaz operation

Módulos a implementar:

- ✓ `operation_limit`
- ✓ `operation_offset`
- ✓ `operation_count`
- ✓ `operation_union`

En esecuele.c

```
void query(char* db_name) {
    database_t* db;
    char statement[MAX_LEN_STATEMENT];
    operation_t* operation;
    int i, ncols, nrows;
    type_t* types;

    /* opens de database */
    db = database_open(db_name);
    if (db == NULL) {
        return;
    }

    /* interactive prompt, ends with Ctrl+D*/
    printf("q> ");
    fflush(stdout);
    while (fgets(statement, MAX_LEN_STATEMENT, stdin) != NULL) {
        *(strchr(statement, '\n')) = '\0';

        /* the operation is parsed to create an operation */
        operation = parser_operation(statement, db);
        if (operation != NULL) {
            /* if the operation is valid */
            types = operation_types(operation);
            ncols = operation_ncols(operation);
            nrows = 0;
            /* for every result of the operation*/
```

TAD `operation_t`

```c
struct operation_ {
    void (* reset)(void* args);
    int (* next)(void* args);
    void* (* get)(int col, void* args);
    void (* close)(void* args);
    int ncols; /* number of columns of the operation */
    type_t* types; /* types of the columns of the operation */
    void* args;
};

typedef struct operation_ operation_t;
```

$\Rightarrow$ *Polimorfismo* mediante los punteros a función `reset`, `next`, `get`, `close`

Inicialización del TAD operation

- En `parser.c`, función `process_token`
- `esecuele.c` llama a `parser_operation`, y ésta a `process_token`

`operation_offset` ~ `operation_limit`, `operation_count` *I*

Desprecia los `offset` primero registros de una cierta consulta (una operación). Para llevarse a cabo necesita:

1. Operación sobre cuyos regitros opera
2. Número de registros recuperados
3. Número de registros a desechar

`operation_offset` ~ `operation_limit`, `operation_count` II

Mirad `operation_select.c` y Tened en cuenta qué devolver `operation_offset_get`, `operation_limit_get`, `operation_count_get` (en este caso se ejecuta una operación y se devuelve el número de registros!!!)

`operation_union` I

Devuelve los registros de dos operaciones, la cuales trabajan sobre las mismas columnas.
Necesita

- ✓ `operation1`
- ✓ `operation2`
- ✓ Un entero que indica qué operación se esta realizando
  1. `operation1`
  2. `operation2`

`operation_union` II

Tened en cuenta que desde esta función se llamará a `operation_next` para ejecutar cada una de las operaciones. En `operation_union_get` se llama a `operation_get` para obtener la columna correspondiente de un cierto registro para una de las dos operaciones.