Taller2Documentacion

1.0

Generado por Doxygen 1.9.1

Capítulo 1

Índice de estructura de datos

1.1. Estructura de datos

Lista de estructuras con una breve descripción:

Capítulo 2

Indice de archivos

2.1. Lista de archivos

Lista de todos los archivos con descripciones breves:

Capítulo 3

Documentación de las estructuras de datos

3.1. Referencia de la Estructura client

Estructura que almacena el fd del socket del cliente y los datos del mismo.

`#include <client.h>`

Campos de datos

- int fd
- struct sockaddr_in socketInfo

3.1.1. Descripción detallada

Estructura que almacena el fd del socket del cliente y los datos del mismo.

3.1.2. Documentación de los campos

3.1.2.1. fd

```
int fd
```

3.1.2.2. socketInfo

```
struct sockaddr_in socketInfo
```

La documentación para esta estructura fue generada a partir del siguiente fichero:

- types/client.h

3.2. Referencia de la Estructura node

Estructura que contiene los datos de conexion del cliente (fd y datos del socket). Empleada en la cola de clientes.

`#include <clientQueue.h>`

Campos de datos

- client_t * data
- struct node * next

3.2.1. Descripción detallada

Estructura que contiene los datos de conexion del cliente (fd y datos del socket). Empleada en la cola de clientes.

3.2.2. Documentación de los campos

3.2.2.1. data

```
client_t* data
```

3.2.2.2. next

```
struct node* next
```

La documentación para esta estructura fue generada a partir del siguiente fichero:

- dataStructures/clientQueue.h

3.3. Referencia de la Estructura record

Estructura empleada como nodo de la lista enlazada empleada en la tabla hash. Almacena un registro del archivo de datos sin procesar.

`#include <record.h>`

Campos de datos

- int sourceId
- int destId
- int hourOfDay
- float meanTravelTime
- struct record * next

3.3.1. Descripción detallada

Estructura empleada como nodo de la lista enlazada empleada en la tabla hash. Almacena un registro del archivo de datos sin procesar.

3.3.2. Documentación de los campos

3.3.2.1. destId

```
int destId
```

3.3.2.2. hourOfDay

```
int hourOfDay
```

3.3.2.3. meanTravelTime

```
float meanTravelTime
```

3.3.2.4. next

```
struct record* next
```

3.3.2.5. sourceId

```
int sourceId
```

La documentación para esta estructura fue generada a partir del siguiente fichero:

- types/record.h

3.4. Referencia de la Estructura recordLinkedList

Lista enlazada empleada para almacenar temporalmente los registros del archivo de datos sin procesar, y luego crear el archivo binario a partir de la misma.

`#include <record.h>`

Campos de datos

- record_t * head

3.4.1. Descripción detallada

Lista enlazada empleada para almacenar temporalmente los registros del archivo de datos sin procesar, y luego crear el archivo binario a partir de la misma.

3.4.2. Documentación de los campos

3.4.2.1. head

`record_t* head`

La documentación para esta estructura fue generada a partir del siguiente fichero:

- types/record.h

3.5. Referencia de la Estructura recordQuery

Estructura empleada para almacenar los datos de consulta del cliente.

`#include <record.h>`

Campos de datos

- int sourceId
- int destId
- int hourOfDay

3.5.1. Descripción detallada

Estructura empleada para almacenar los datos de consulta del cliente.

3.5.2. Documentación de los campos

3.5.2.1. destId

```
int destId
```

3.5.2.2. hourOfDay

```
int hourOfDay
```

3.5.2.3. sourceId

```
int sourceId
```

La documentación para esta estructura fue generada a partir del siguiente fichero:

- types/record.h

3.6. Referencia de la Estructura recordRead

Estructura empleada para escribir los datos de cada registro en el archivo binario.

```
#include <record.h>
```

Campos de datos

- int sourceId
- int destId
- int hourOfDay
- float meanTravelTime

3.6.1. Descripción detallada

Estructura empleada para escribir los datos de cada registro en el archivo binario.

3.6.2. Documentación de los campos

3.6.2.1. destId

```
int destId
```

3.6.2.2. hourOfDay

```
int hourOfDay
```

3.6.2.3. meanTravelTime

```
float meanTravelTime
```

3.6.2.4. sourceId

```
int sourceId
```

La documentación para esta estructura fue generada a partir del siguiente fichero:

- types/record.h

Capítulo 4

Documentación de archivos

4.1. Referencia del Archivo dataStructures/clientQueue.c

```
#include "clientQueue.h"
```

Funciones

- void enqueue (client_t *data)

  enqueue Crea una variable de tipo nodo_t, le inserta los datos del cliente y la enlaza al final de la cola.
- client_t * dequeue ()

  dequeue Remueve la variable de tipo nodo_t de la cabeza y retorna los datos del cliente si esta existe. De lo contrario, retorna NULL.

Variables

- node_t * head
- node_t * tail

4.1.1. Documentación de las funciones

4.1.1.1. dequeue()

```
client_t* dequeue ( )
```

dequeue Remueve la variable de tipo nodo_t de la cabeza y retorna los datos del cliente si esta existe. De lo contrario, retorna NULL.

Devuelve

Apuntador con los datos del cliente

4.1.1.2. enqueue()

```
void enqueue (
            client_t * data )
```

enqueue Crea una variable de tipo nodo_t, le inserta los datos del cliente y la enlaza al final de la cola.

Parámetros

client	Apuntador con los datos del cliente.

4.1.2. Documentación de las variables

4.1.2.1. head

```
node_t* head
```

4.1.2.2. tail

```
node_t* tail
```

4.2. Referencia del Archivo dataStructures/clientQueue.h

Contiene la definición de la estructura nodo y las funciones de la cola: enqueue y dequeue.

```
#include <stdlib.h>
#include <stdio.h>
#include "../types/client.h"
```

Estructuras de datos

- struct node

  Estructura que contiene los datos de conexion del cliente (fd y datos del socket). Empleada en la cola de clientes.

typedefs

- typedef struct node node_t

  Estructura que contiene los datos de conexion del cliente (fd y datos del socket). Empleada en la cola de clientes.

Funciones

- void enqueue (client_t *)

  enqueue Crea una variable de tipo nodo_t, le inserta los datos del cliente y la enlaza al final de la cola.
- client_t * dequeue ()

  dequeue Remueve la variable de tipo nodo_t de la cabeza y retorna los datos del cliente si esta existe. De lo contrario, retorna NULL.

4.2.1. Descripción detallada

Contiene la definición de la estructura nodo y las funciones de la cola: enqueue y dequeue.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan Lopez Castellanos - Víctor Alfredo Barragán Paez @title Cola para almacenar la información de los clientes

4.2.2. Documentación de los 'typedefs'

4.2.2.1. node_t

```
typedef struct node node_t
```

Estructura que contiene los datos de conexion del cliente (fd y datos del socket). Empleada en la cola de clientes.

4.2.3. Documentación de las funciones

4.2.3.1. dequeue()

```
client_t* dequeue ( )
```

dequeue Remueve la variable de tipo nodo_t de la cabeza y retorna los datos del cliente si esta existe. De lo contrario, retorna NULL.

Devuelve

Apuntador con los datos del cliente

4.2.3.2. enqueue()

```
void enqueue (
            client_t * data )
```

enqueue Crea una variable de tipo nodo_t, le inserta los datos del cliente y la enlaza al final de la cola.

Parámetros

client	Apuntador con los datos del cliente.

4.3. Referencia del Archivo p2-client.c

Programa que crea una conexión con el servidor y presenta una interfaz de consola para el ingreso de los datos del registro a buscar.

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>
#include "./utils/errorHandler.h"
#include "./types/record.h"
```

defines

- #define TRUE 1
- #define PORT 3535
- #define ORIGIN_INPUT 1
- #define DEST_INPUT 2
- #define HOUR_INPUT 3
- #define SEND_REQUEST 4
- #define EXIT 5

Funciones

- int sendRequest (int clientfd, recordQuery_t queryParams)

  sendRequest Función que envía la petición al servidor, con los datos de consulta, y retorna la respuesta.
- int main (int argc, char ∗argv[ ])

  main Función principal. Crea un socket de cliente, realiza la conexión con el servidor y administra el ingreso y envío de los datos de consulta junto con la finalización de la conexión.

4.3.1. Descripción detallada

Programa que crea una conexión con el servidor y presenta una interfaz de consola para el ingreso de los datos del registro a buscar.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Programa del cliente

4.3.2. Documentación de los 'defines'

4.3.2.1. DEST_INPUT

```
#define DEST_INPUT 2
```

4.3.2.2. EXIT

```
#define EXIT 5
```

4.3.2.3. HOUR_INPUT

```
#define HOUR_INPUT 3
```

4.3.2.4. ORIGIN_INPUT

```
#define ORIGIN_INPUT 1
```

4.3.2.5. PORT

```
#define PORT 3535
```

4.3.2.6. SEND_REQUEST

```
#define SEND_REQUEST 4
```

4.3.2.7. TRUE

```
#define TRUE 1
```

4.3.3. Documentación de las funciones

4.3.3.1. main()

```
int main (
            int argc,
            char * argv[] )
```

main Función principal. Crea un socket de cliente, realiza la conexión con el servidor y administra el ingreso y envío de los datos de consulta junto con la finalización de la conexión.

Parámetros

argv	El primer argumento de la entrada por terminal es la dirección IP del cliente que realizará la conexión con el servidor

4.3.3.2. sendRequest()

```
int sendRequest (
            int clientfd,
            recordQuery_t queryParams )
```

sendRequest Función que envía la petición al servidor, con los datos de consulta, y retorna la respuesta.

Parámetros

clientfd	Descriptor del socket del cliente
queryParams	Variable de tipo estructura que almacena los datos de consulta del cliente.

Devuelve

Respuesta del servidor (media de tiempos de viaje), -1 si ocurrió un error, 0 si el registro no se encontró.

4.4. Referencia del Archivo p2-many-clients.c

Programa de utilidad que realiza la prueba de conexión de varios clientes con el servidor y solicitudes con datos aleatorios.

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>
#include "./utils/errorHandler.h"
#include "./types/record.h"
```

defines

- #define PORT 3535
- #define NUM_THREADS 65

Funciones

- void * createClient ()

  createClient Función que ejecutan los hilos. Se encarga de establecer una conexión entre un nuevo cliente y el servidor y de realizar solicitudes con datos aleatorios hasta que el id de origen que envía el cliente sea 0, lo cual permite a ambas partes finalizar la conexión y desocupar el socket del cliente.
- int main ()

  main Función principal. Se encarga de crear los hilos que ejecutan la función createClient, así como de esperar a que estos terminen finalicen su ejecución.

4.4.1. Descripción detallada

Programa de utilidad que realiza la prueba de conexión de varios clientes con el servidor y solicitudes con datos aleatorios.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Programa de varios clientes

4.4.2. Documentación de los 'defines'

4.4.2.1. NUM_THREADS

```
#define NUM_THREADS 65
```

4.4.2.2. PORT

```
#define PORT 3535
```

4.4.3. Documentación de las funciones

4.4.3.1. createClient()

```
void * createClient ( )
```

createClient Función que ejecutan los hilos. Se encarga de establecer una conexión entre un nuevo cliente y el servidor y de realizar solicitudes con datos aleatorios hasta que el id de origen que envía el cliente sea 0, lo cual permite a ambas partes finalizar la conexión y desocupar el socket del cliente.

4.4.3.2. main()

```
int main ( )
```

main Función principal. Se encarga de crear los hilos que ejecutan la función createClient, así como de esperar a que estos terminen finalicen su ejecución.

4.5. Referencia del Archivo p2-server.c

Programa que crea un socker que actúa como servidor y administra un máximo de 32 conexiones con clientes al mismo tiempo, además de procesar las solicitudes de estos.

```
#include <signal.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <strings.h>
#include <unistd.h>
#include <pthread.h>
#include "./dataStructures/clientQueue.h"
#include "./search/searchRecord.h"
```

defines

- #define PORT 3535
- #define BACKLOG 100
- #define MAX_CLIENTS 32

Funciones

- void signalInterruptHandler (int)

  signalInterruptHandler Función empleada para cerrar recursos una vez se envía la señal de interrupción control + C
- void * threadFunction (void *)

  threadFunction Función que ejecuta cada hilo una vez es creado. Se encarga de desencolar clientes de la cola y hacer el llamado a la función que maneja la petición del cliente, siempre que exista en la cola.
- void * handleRequest (client_t *)

  handleRequest Función que maneja la petición del cliente, recibe los datos de consulta y llama a la función encargada de buscar el registro solicitado y devolver el tiempo medio de viajes.
- int main ()

  main Función principal. Crea un socket de servidor, crea los hilos, acepta las conexiones entrantes y encola los clientes que se conectan al servidor.

Variables

- pthread_t threads [MAX_CLIENTS]

  Arreglo de hilos pthread_t cuyo tamaño es el máximo de clientes que pueden establecer una conexión con el servidor al mismo tiempo.
- pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER

  Mutex empleado para evitar que los hilos y desencolen de la cola de clientes o escriban en el archivo log al mismo tiempo.
- pthread_cond_t threadCondition = PTHREAD_COND_INITIALIZER

  Condición de hilo empleada para bloquear el desencolamiento hasta que se encole un nuevo cliente.
- FILE * logFile
- int serverfd

4.5.1. Descripción detallada

Programa que crea un socker que actúa como servidor y administra un máximo de 32 conexiones con clientes al mismo tiempo, además de procesar las solicitudes de estos.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Programa del servidor

4.5.2. Documentación de los 'defines'

4.5.2.1. BACKLOG

```
#define BACKLOG 100
```

4.5.2.2. MAX_CLIENTS

```
#define MAX_CLIENTS 32
```

4.5.2.3. PORT

```
#define PORT 3535
```

4.5.3. Documentación de las funciones

4.5.3.1. handleRequest()

```
void * handleRequest (
            client_t * client )
```

handleRequest Función que maneja la petición del cliente, recibe los datos de consulta y llama a la función encargada de buscar el registro solicitado y devolver el tiempo medio de viajes.

Parámetros

Apuntador	a la estructura client_t con los datos de conexión del cliente.

4.5.3.2. main()

```
int main ( )
```

main Función principal. Crea un socket de servidor, crea los hilos, acepta las conexiones entrantes y encola los clientes que se conectan al servidor.

4.5.3.3. signalInterruptHandler()

```
void signalInterruptHandler (
            int s )
```

signalInterruptHandler Función empleada para cerrar recursos una vez se envía la señal de interrupción control + C

4.5.3.4. threadFunction()

```
void * threadFunction (
            void * arg )
```

threadFunction Función que ejecuta cada hilo una vez es creado. Se encarga de desencolar clientes de la cola y hacer el llamado a la función que maneja la petición del cliente, siempre que exista en la cola.

4.5.4. Documentación de las variables

4.5.4.1. logFile

```
FILE* logFile
```

4.5.4.2. mutex

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
```

Mutex empleado para evitar que los hilos y desencolen de la cola de clientes o escriban en el archivo log al mismo tiempo.

4.5.4.3. serverfd

```
int serverfd
```

4.5.4.4. threadCondition

```
pthread_cond_t threadCondition = PTHREAD_COND_INITIALIZER
```

Condición de hilo empleada para bloquear el desencolamiento hasta que se encole un nuevo cliente.

4.5.4.5. threads

```
pthread_t threads[MAX_CLIENTS]
```

Arreglo de hilos pthread_t cuyo tamaño es el máximo de clientes que pueden establecer una conexión con el servidor al mismo tiempo.

4.6. Referencia del Archivo search/searchRecord.c

```
#include "searchRecord.h"
```

Funciones

- float searchRecordMeanTravelTime (recordQuery_t *queryData)

  searchRecordMeanTravelTime Función que recibe los datos de consulta, hace la búsqueda de la posición del primer registro con el id de origen ingresado en una tabla hash almacenada en disco, para luego comparar los registros a partir de dicha posición hasta encontrar el registro buscado y retornar el tiempo medio de viajes o retornar cero en caso de no encontrar el registro.

4.6.1. Documentación de las funciones

4.6.1.1. searchRecordMeanTravelTime()

```
float searchRecordMeanTravelTime (
            recordQuery_t * queryData )
```

searchRecordMeanTravelTime Función que recibe los datos de consulta, hace la búsqueda de la posición del primer registro con el id de origen ingresado en una tabla hash almacenada en disco, para luego comparar los registros a partir de dicha posición hasta encontrar el registro buscado y retornar el tiempo medio de viajes o retornar cero en caso de no encontrar el registro.

Parámetros

queryParameters	Datos de la consulta (id de origen, id de destino y hora del día).

Devuelve

Tiempo medio de viajes.

4.7. Referencia del Archivo search/searchRecord.h

Realiza la búsqueda del tiempo medio de viajes del archivo proporcionado por Uber.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include "../utils/errorHandler.h"
#include "../types/record.h"
```

defines

- #define TRUE 1
- #define FALSE 0

Funciones

- float searchRecordMeanTravelTime (recordQuery_t *)

  searchRecordMeanTravelTime Función que recibe los datos de consulta, hace la búsqueda de la posición del primer registro con el id de origen ingresado en una tabla hash almacenada en disco, para luego comparar los registros a partir de dicha posición hasta encontrar el registro buscado y retornar el tiempo medio de viajes o retornar cero en caso de no encontrar el registro.

4.7.1. Descripción detallada

Realiza la búsqueda del tiempo medio de viajes del archivo proporcionado por Uber.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Función de búsqueda de registros

4.7.2. Documentación de los 'defines'

4.7.2.1. FALSE

```
#define FALSE 0
```

4.7.2.2. TRUE

```
#define TRUE 1
```

4.7.3. Documentación de las funciones

4.7.3.1. searchRecordMeanTravelTime()

```
float searchRecordMeanTravelTime (
            recordQuery_t * queryData )
```

searchRecordMeanTravelTime Función que recibe los datos de consulta, hace la búsqueda de la posición del primer registro con el id de origen ingresado en una tabla hash almacenada en disco, para luego comparar los registros a partir de dicha posición hasta encontrar el registro buscado y retornar el tiempo medio de viajes o retornar cero en caso de no encontrar el registro.

Parámetros

queryParameters	Datos de la consulta (id de origen, id de destino y hora del día).

Devuelve

Tiempo medio de viajes.

4.8. Referencia del Archivo types/client.h

Estructura que almacena los datos del cliente.

```
#include <netinet/in.h>
```

Estructuras de datos

- struct client

  Estructura que almacena el fd del socket del cliente y los datos del mismo.

typedefs

- typedef struct client client_t

  Estructura que almacena el fd del socket del cliente y los datos del mismo.

4.8.1. Descripción detallada

Estructura que almacena los datos del cliente.

Versión

1.0.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Estructura client

4.8.2. Documentación de los 'typedefs'

4.8.2.1. client_t

```
typedef struct client client_t
```

Estructura que almacena el fd del socket del cliente y los datos del mismo.

4.9. Referencia del Archivo types/record.h

Estructuras empleadas para representar la información de los registros suministrados por el archivo de Uber.

Estructuras de datos

- struct record

  Estructura empleada como nodo de la lista enlazada empleada en la tabla hash. Almacena un registro del archivo de datos sin procesar.
- struct recordLinkedList

  Lista enlazada empleada para almacenar temporalmente los registros del archivo de datos sin procesar, y luego crear el archivo binario a partir de la misma.
- struct recordRead

  Estructura empleada para escribir los datos de cada registro en el archivo binario.
- struct recordQuery

  Estructura empleada para almacenar los datos de consulta del cliente.

typedefs

- typedef struct record record_t

  Estructura empleada como nodo de la lista enlazada empleada en la tabla hash. Almacena un registro del archivo de datos sin procesar.
- typedef struct recordLinkedList recordList_t

  Lista enlazada empleada para almacenar temporalmente los registros del archivo de datos sin procesar, y luego crear el archivo binario a partir de la misma.
- typedef struct recordRead recordRead_t

  Estructura empleada para escribir los datos de cada registro en el archivo binario.
- typedef struct recordQuery recordQuery_t

  Estructura empleada para almacenar los datos de consulta del cliente.

4.9.1. Descripción detallada

Estructuras empleadas para representar la información de los registros suministrados por el archivo de Uber.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Estructuras auxiliares para los registros

4.9.2. Documentación de los 'typedefs'

4.9.2.1. record_t

```
typedef struct record record_t
```

Estructura empleada como nodo de la lista enlazada empleada en la tabla hash. Almacena un registro del archivo de datos sin procesar.

4.9.2.2. recordList_t

```
typedef struct recordLinkedList recordList_t
```

Lista enlazada empleada para almacenar temporalmente los registros del archivo de datos sin procesar, y luego crear el archivo binario a partir de la misma.

4.9.2.3. recordQuery_t

```
typedef struct recordQuery recordQuery_t
```

Estructura empleada para almacenar los datos de consulta del cliente.

4.9.2.4. recordRead_t

```
typedef struct recordRead recordRead_t
```

Estructura empleada para escribir los datos de cada registro en el archivo binario.

4.10. Referencia del Archivo utils/createHashTable.c

Programa de utilidad empleado para crear el archivo de la tabla hash y los datos de los registros procesados, a partir del archivo suministrado por Uber.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../types/record.h"
```

defines

- #define TRUE 1
- #define FALSE 0
- #define MAX_LINE_SIZE 150
- #define MAX_SOURCE_ID 1160
- #define MAX_ATTR_NEEDED 4
- #define SOURCE_ID 0
- #define DEST_ID 1
- #define HOD 2
- #define MEAN_TRAV_TIME 3

Funciones

- void freeLinkedList (record_t **head)

  freeLinkedList Función auxiliar empleada para liberar la memoria reservada para cada cada lista enlazada de la tabla hash en memoria.
- void addRecord (record_t **list, record_t newRecord)

  addRecord Función empleada para añadir los datos de un registro en una lista enlazada de la tabla hash.
- int main ()

  main Función principal. Procesa el archivo suministrado por Uber y los almacena en un archivo binario, además de guardar las posiciones en dicho archivo del primer registro por id de origen (tabla hash).

4.10.1. Descripción detallada

Programa de utilidad empleado para crear el archivo de la tabla hash y los datos de los registros procesados, a partir del archivo suministrado por Uber.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Programa createHashTable

4.10.2. Documentación de los 'defines'

4.10.2.1. DEST_ID

```
#define DEST_ID 1
```

4.10.2.2. FALSE

```
#define FALSE 0
```

4.10.2.3. HOD

```
#define HOD 2
```

4.10.2.4. MAX_ATTR_NEEDED

```
#define MAX_ATTR_NEEDED 4
```

4.10.2.5. MAX_LINE_SIZE

```
#define MAX_LINE_SIZE 150
```

4.10.2.6. MAX_SOURCE_ID

```
#define MAX_SOURCE_ID 1160
```

4.10.2.7. MEAN_TRAV_TIME

```
#define MEAN_TRAV_TIME 3
```

4.10.2.8. SOURCE_ID

```
#define SOURCE_ID 0
```

4.10.2.9. TRUE

```
#define TRUE 1
```

4.10.3. Documentación de las funciones

4.10.3.1. addRecord()

```
void addRecord (
            record_t ** list,
            record_t newRecord )
```

addRecord Función empleada para añadir los datos de un registro en una lista enlazada de la tabla hash.

Parámetros

list	Apuntador a un apuntador a la estructura record_t, la cabeza de la lista.
newRecord	Variable de tipo record_t que contiene los datos del registro leído y a ser ingresados en la lista.

4.10.3.2. freeLinkedList()

```
void freeLinkedList (
            record_t ** head )
```

freeLinkedList Función auxiliar empleada para liberar la memoria reservada para cada cada lista enlazada de la tabla hash en memoria.

Parámetros

head	Apuntador a un apuntador a la estructura record_t, la cabeza de la lista.

4.10.3.3. main()

```
int main ( )
```

main Función principal. Procesa el archivo suministrado por Uber y los almacena en un archivo binario, además de guardar las posiciones en dicho archivo del primer registro por id de origen (tabla hash).

4.11. Referencia del Archivo utils/errorHandler.c

```
#include "errorHandler.h"
```

Funciones

- void handleError (int returnCode, char ∗msg)

  handleError Función auxiliar que recibe el código retornado por el llamado a una función del sistema operativo e imprime la cadena suministrada como argumento en la salida stderr, además de finalizar la ejecución de la rutina donde se llame.

4.11.1. Documentación de las funciones

4.11.1.1. handleError()

```
void handleError (
            int returnCode,
            char * msg )
```

handleError Función auxiliar que recibe el código retornado por el llamado a una función del sistema operativo e imprime la cadena suministrada como argumento en la salida stderr, además de finalizar la ejecución de la rutina donde se llame.

Parámetros

codigo	Código de valor entero, el cual si es menor que 0 indica que ha ocurrido un error en el llamado a una función del sistema operativo.
mensaje	Cadena que se imprime en la salida de error estándar de recibirse un código de error.

4.12. Referencia del Archivo utils/errorHandler.h

Función de utilidad empleada para imprimir errores en la salida stderr error estándar.

```
#include <stdio.h>
#include <stdlib.h>
```

Funciones

- void handleError (int, char *)

  handleError Función auxiliar que recibe el código retornado por el llamado a una función del sistema operativo e imprime la cadena suministrada como argumento en la salida stderr, además de finalizar la ejecución de la rutina donde se llame.

4.12.1. Descripción detallada

Función de utilidad empleada para imprimir errores en la salida stderr error estándar.

Versión

1.0

Fecha

02/06/2021

Autor

Jonathan López Castellanos - Víctor Alfredo Barragán Paez @title Función handleError

4.12.2. Documentación de las funciones

4.12.2.1. handleError()

```
void handleError (
            int returnCode,
            char * msg )
```

handleError Función auxiliar que recibe el código retornado por el llamado a una función del sistema operativo e imprime la cadena suministrada como argumento en la salida stderr, además de finalizar la ejecución de la rutina donde se llame.

Parámetros

codigo	Código de valor entero, el cual si es menor que 0 indica que ha ocurrido un error en el llamado a una función del sistema operativo.
mensaje	Cadena que se imprime en la salida de error estándar de recibirse un código de error.

Índice alfabético