UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2019 - 2.do cuatrimestre

Algoritmos y Programación I (95.11)

Curso 1 - Ing. Cardozo

TRABAJO PRÁCTICO INTEGRADOR - Análisis forense de extracto bancario

Integrantes:

Becker, Gonzalo Agustín - #104291
<gonza.becker132@gmail.com >
Triberti Martínez, Gonzalo Nicolás - # 100126
<gonzalontm@gmail.com>

Fecha de entrega: 28/12/2019

Diagrama de flujo

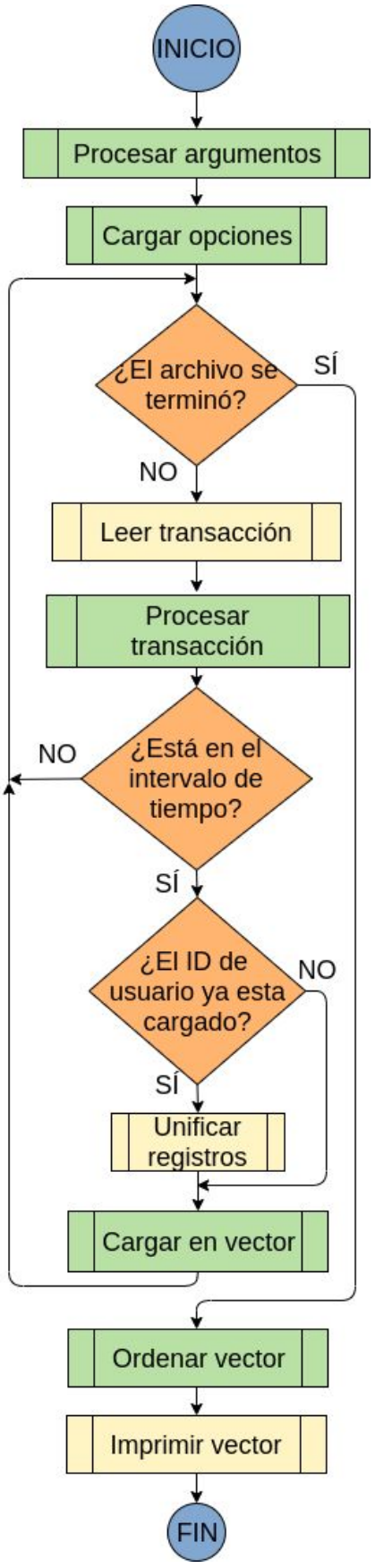
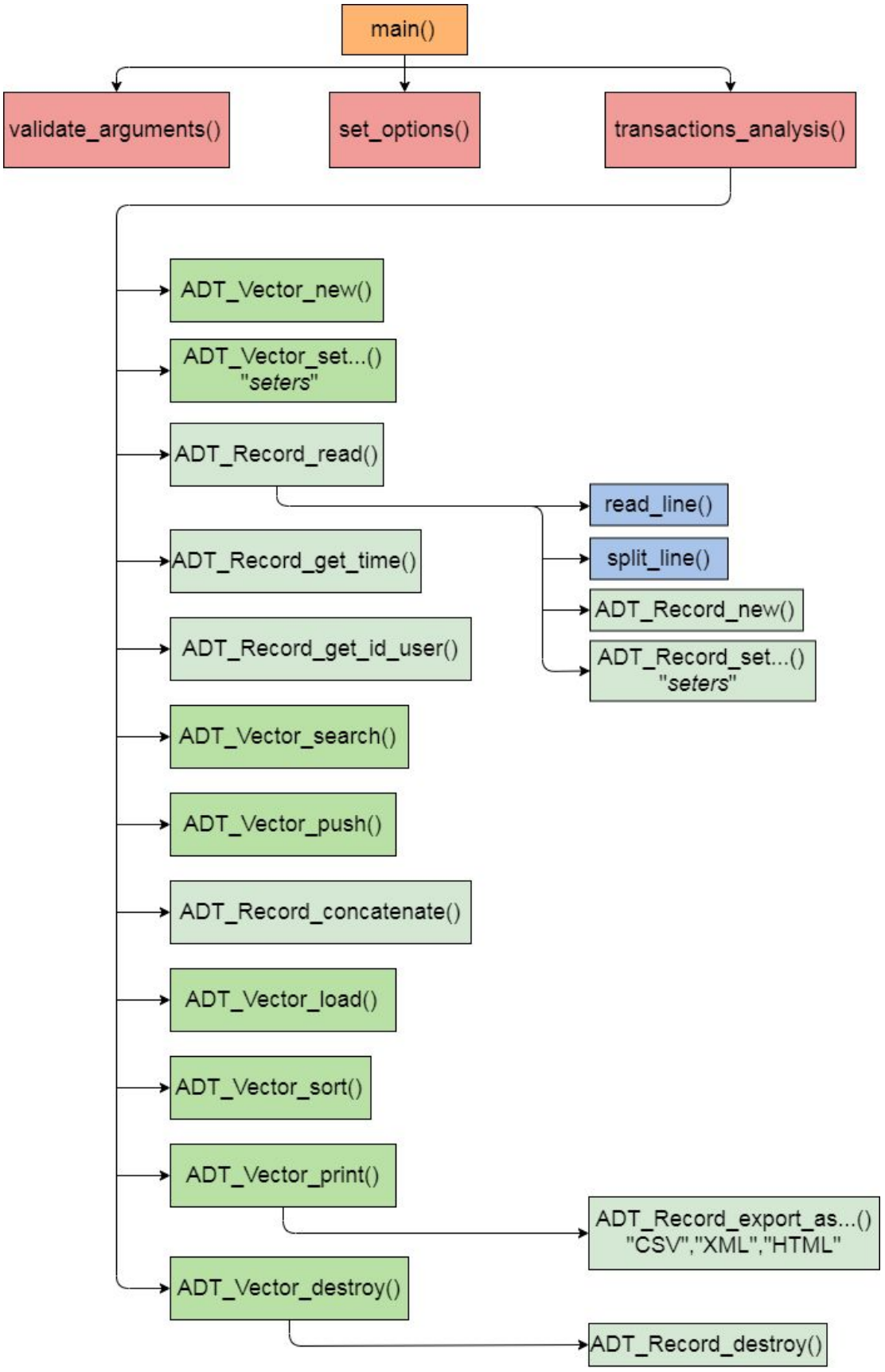


Diagrama de funciones



INFORME

Introducción

En el presente documento se expone un programa para procesar transacciones bancarias. Este consiste en la lectura de los registros de un archivo de texto de entrada, el procesamiento de estos registros y la exportación de ciertos valores a un archivo de texto de salida. Para facilitar su comprensión, se adjunta a continuación un diagrama de flujo y la estructura funcional del programa en cuestión. Se explican además las estrategias adoptadas durante su desarrollo, así como los problemas encontrados y sus respectivas soluciones. Además, se muestran los resultados de ejecución y se extraen conclusiones del trabajo realizado. Finalmente, se adjuntan los códigos fuente del programa.

Alternativas consideradas y estrategias adoptadas

En cuanto a los *CLA* (*Command Line Arguments,* por sus siglas en inglés), se decidió utilizar argumentos en línea de órdenes con un orden obligatorio y definido para interactuar con el programa. Esto simplifica considerablemente la función *validate_arguments*(). El prototipo de CLA a utilizar se especifica en el Anexo 1.

El presente programa se basa principalmente en la implementación de ADT (*Abstract Data Types*, por sus siglas en inglés). Para incrementar la eficiencia del programa, se optó por incluir una mayor cantidad de variables dentro de los ADT. De esta forma, puede observarse que *ADT_Record_t* incluye también el tiempo (en segundos de UNIX) como un campo, denominado *time*, además de los campos necesarios para poder realizar la exportación a un archivo de texto (*id_user, incomes y expense*s). Del mismo modo, es notorio que *ADT_Vector_t* no es simplemente un vector, si no que es una estructura que contiene al vector y otras variables que permiten que el código sea más simple y eficiente (por ejemplo, *un encabezado, un pie de página*, *una impresora*, etc.).

Se puede destacar que en los primeros bocetos del programa la información de formato y tiempo inicial/final ingresada por *CLA* se utilizaba directamente la variable *argv* en las funciones de bajo nivel. Este método resultó inseguro e incorrecto, por lo que se decidió crear una estructura global *options_t* en la cual se graban los datos para su posterior utilización. Así, se consigue una mayor adaptación al modelo IPO (*Input-Process-Output,* por sus siglas en inglés), mediante el cual se independizan los diferentes bloques del programa.

El lector podrá observar también que existen múltiples primitivas de los tipos de datos abstracto que no se utilizan. Sin embargo, las primitivas hacen el programa más compacto, ya que cuando se deseen realizar modificaciones, habrá una menor necesidad de crear más funciones.

Con relación a los métodos de ordenamiento, se procede en el siguiente programa a implementar el algoritmo QuickSort, el cual se puede implementar a partir de la función qsort localizada en la biblioteca <stdlib.h>. Inicialmente se utilizó un algoritmo de ordenamiento por selección, pero este resulta ineficiente, pues es de orden $\sigma(n^2)$. En cambio, el algoritmo QuickSort provee de un orden de operaciones de $\sigma(n\ log\ n)$, por lo que es más eficiente que las otras alternativas.

Resultados de ejecución

Cantidad de argumentos inválida:

```
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL$
./bank_analysis -fmt csv -out file_out.txt -in file_in.txt
CANTIDAD DE ARGUMENTOS INVALIDA
```

Banderas inválidas:

```
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL$
./bank_analysis -FMT csv -out file_out.txt -in file_in.txt -t1 10
0 -t2 1363522001
BANDERA DE FORMATO INVALIDA
```

Archivo de entrada vacío:

```
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL$
./bank_analysis -fmt csv -out file_out.txt -in file_empty.txt -t1
 100 -t2 1363522001
EL ARCHIVO DE ENTRADA ESTA VACIO
```

Intervalo de tiempo inválido:

```
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL$
 ./bank_analysis -fmt csv -out file_out.txt -in file_in.txt -t1
100 -t2 1
LOS TIEMPO INGRESADOS NO TIENEN INTERVALO VALIDO
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL$
```

Respuesta esperada en caso de finalizar correctamente:

```
gonza-ubuntu@gonza-ubuntu:~/Escritorio/MESA DE TRABAJO/TP FINAL
$ ./bank_analysis -fmt csv -out file_out.txt -in file_in.txt -t
1 100 -t2 1363522001
PROCESAMIENTO TERMINADO
```

Archivo de prueba utilizado:

```
ID_TRANSACCION;ID_USUARIO;FECHA;MONTO;DESCRIPCION
123456;25;15/10/2011 23:14:05;500;descripcion
123457;3;16/10/2011 23:14:06;-400;descripcion
123458;45;17/10/2011 23:14:07;100;descripcion
123459;30;18/10/2013 23:14:08;45;descripcion
123460;11;19/10/2011 23:14:09;0;descripcion
123461;5;19/10/2011 23:14:10;0;descripcion
123462;5;20/10/2011 23:14:11;0;descripcion
123463;7;21/10/2011 23:14:12;12;descripcion
123464;48;15/10/2012 23:14:13;13;descripcion
123465;30;15/01/2013 23:14:14;-14;descripcion
123466;22;15/02/2013 23:14:15;-15;descripcion
123467;30;15/03/2015 23:14:16;-16;descripcion
123468;71;15/04/2013 23:14:17;-12;descripcion
123469;52;15/05/2014 23:14:18;-18;descripcion
123470;12;15/06/2015 23:14:19;-19;descripcion
123471;30;15/07/2016 23:14:20;20;descripcion
123472;5;15/10/2017 23:14:21;21;descripcion
123473;4;15/10/2018 23:14:22;22;descripcion
123474;5;15/10/2019 23:14:23;23;descripcion
123475;2;15/10/2020 23:14:24;-24;descripcion
123476;7;15/10/2021 23:14:25;25;descripcion
123477;8;15/10/2022 23:14:26;26;descripcion
123478;6;15/10/2023 23:14:27;27;descripcion
```

Archivo de salida (CSV):

```
ID_USUARIO,INGRESOS,EGRESOS
30,65.00,30.00
12,0.00,19.00
52,0.00,18.00
22,0.00,15.00
71,0.00,12.00
48,13.00,0.00
```

Archivo de salida (XML):

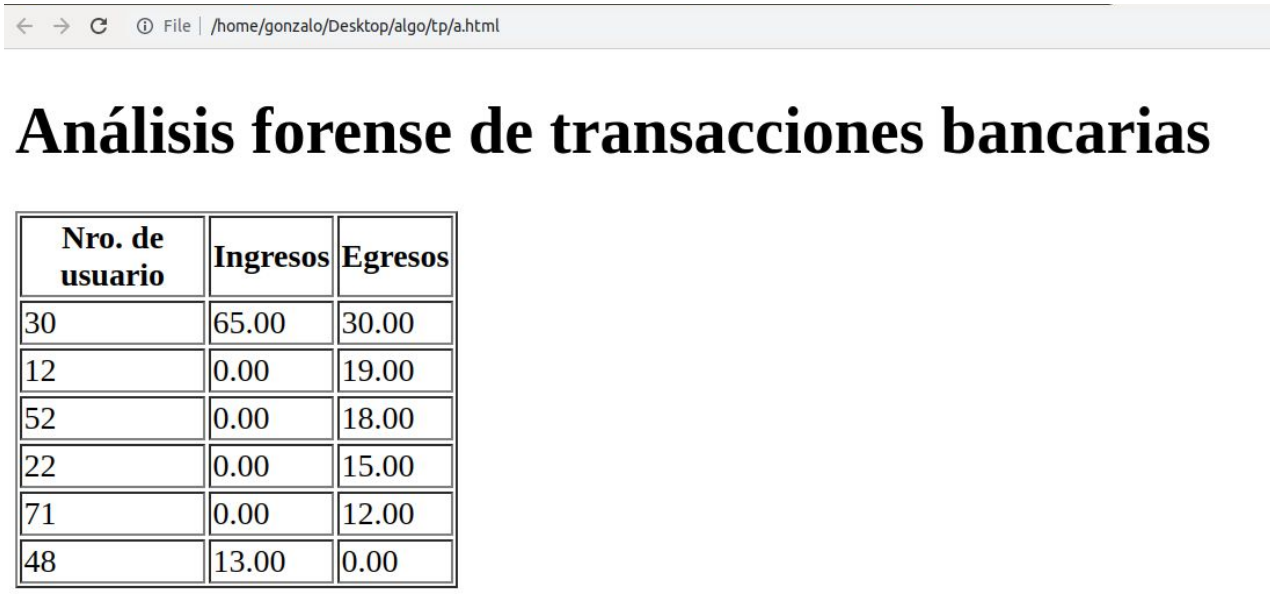
```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<usuarios>
	<usuario>
		<id>30</id>
		<ingresos65.00</ingresos>
		<egresos>30.00</egresos>
	</usuario>
	<usuario>
		<id>12</id>
		<ingresos0.00</ingresos>
		<egresos>19.00</egresos>
	</usuario>
	<usuario>
		<id>52</id>
		<ingresos0.00</ingresos>
		<egresos>18.00</egresos>
	</usuario>
	<usuario>
		<id>22</id>
		<ingresos0.00</ingresos>
		<egresos>15.00</egresos>
	</usuario>
	<usuario>
		<id>71</id>
		<ingresos0.00</ingresos>
		<egresos>12.00</egresos>
	</usuario>
	<usuario>
		<id>48</id>
		<ingresos13.00</ingresos>
		<egresos>0.00</egresos>
	</usuario>
</usuarios>
```

Archivo de salida (HTML):

```
<!DOCTYPE html>
<html>
	<head>
		<title>An&aacutelisis forense de transacciones bancarias</title>
	</head>
	<body>
		<h1>An&aacutelisis forense de transacciones bancarias </h1>
		<table border=1 width="30%">
			<thead>
				<tr>
					<th>Nro. de usuario</th>
					<th>Ingresos</th>
					<th>Egresos</th>
				</tr>
			</thead>
				<tr>
					<td>30</td>
					<td>65.00</td>
					<td>30.00</td>
				</tr>
				<tr>
					<td>12</td>
					<td>0.00</td>
					<td>19.00</td>
				</tr>
				<tr>
					<td>52</td>
					<td>0.00</td>
					<td>18.00</td>
				</tr>
				<tr>
					<td>22</td>
					<td>0.00</td>
					<td>15.00</td>
				</tr>
				<tr>
					<td>71</td>
					<td>0.00</td>
					<td>12.00</td>
				</tr>
				<tr>
					<td>48</td>
					<td>13.00</td>
					<td>0.00</td>
```

```
                    </tr>
                </table>
        </body>
</html>
```

Formato de página web a partir de HTML:



Problemas encontrados y soluciones implementadas

Uno de los principales problemas fue que inicialmente se implementaron primitivas muy generales, poco precisas y que realizaban diversas operaciones. Como solución se simplificaron estas primitivas hasta llegar a funciones simples y concisas que en conjunto realizan la misma operación. La correcta modularización permite que el programa sea más claro, entendible y robusto.

Se destaca también un error en la implementación de los punteros a función en la que estos no funcionaban correctamente. Para resolver este problema se optó por crear un programa secundario que verifique el correcto funcionamiento (técnica usualmente conocida como scaffold).

Gracias a esto y a la realización de un diagrama de funciones, se encontró una solución al problema, el cual era un error en la implementación de punteros sobre la variable *ADT_Vector_t*.

Por último, se realizaron múltiples cambios en la lógica del algoritmo en cuestión. Tal es el caso de la carga de datos en el vector. En efecto, en la versión final se carga cada registro

en la posición indicada por el ID, mientras que en versiones anteriores se cargaban los registros en posiciones consecutivas. Esta decisión fue tomada tras observar que, al cargar cada registro en posiciones consecutivas, se requeriría una mayor cantidad de procesamiento. Por eso, se optó por minimizar los tiempos de procesamiento, a costa de una mayor utilización de la memoria.

Conclusión

En conclusión, se destaca la eficiencia con la cual es posible desarrollar este programa en el lenguaje de programación C. Esto se debe a la posibilidad de utilizar memoria dinámica, tipos de dato abstractos y otras herramientas, que resultan sumamente útiles al momento de construir un algoritmo.

Los tipos de datos abstractos le otorgan al programa la cualidad de ser compactos, pues es posible codificar algoritmos desconociendo el real funcionamiento de cada primitiva. Esta modularización causa que el código sea mucho más fácil de comprender, pues no es necesario ahondar en el funcionamiento de cada primitiva. Además, si la sintaxis elegida es clara, es muy sencillo reconocer a qué elemento está haciendo referencia una cierta primitiva; por ejemplo, en el código adjunto podrán distinguirse dos tipos de primitivas, las que comienzan con *ADT_Record* y las que comienzan con *ADT_Vector*. Se puede decir entonces que los tipos de dato abstracto tienen un rol central en modularizar, optimizar y simplificar un programa, y que son, por lo tanto, un medio sumamente efectivo para lograr una reducción del esfuerzo en general del programador para desarrollar un sistema de *software*.

En cuanto al procesamiento, se destaca el rol central de la modularización al momento de unificar diferentes bloques. Esto se debe a que, con una correcta modularización, se simplifica el proceso de detectar el origen de los errores. También, es fundamental desarrollar diagramas de flujo y de funciones previo a implementar un cierto algoritmo, ya que muchas veces un simple cambio en la lógica utilizada puede simplificar otros segmentos del código.

Por último, es interesante mencionar la implementación de los punteros a función. Estos se utilizan para evitar hacer referencia a los datos que se encuentran contenidos en el vector. De esta forma, si se quiere destruir el contenido de un elemento del registro, se puede utilizar una función determinada para llevar a cabo esa acción, pasándola como un puntero a la función. Así, se logra destruir cualquier tipo de dato que esté en el vector, manteniendo la generalidad del tipo de dato abstracto *“Vector”*.

Bibliografía

[1] Enunciado del TP final – Análisis forense de extracto bancario, (link de página del campus).

Anexo 1: Makefile

Los programas se compilan a partir del siguiente Makefile:

```
CFLAGS = -Wall -ansi -pedantic -O2
CC=gcc

all: bank_analysis

bank_analysis:  bank_analysis.o  main.o  errors.o  record.o  vector.o  utils.o
date_support.o setup.o
	$(CC) $(CFLAGS) -o bank_analysis bank_analysis.o main.o errors.o record.o
vector.o utils.o date_support.o

bank_analysis.o: bank_analysis.c main.h types.h utils.h
	$(CC) $(CFLAGS) -o bank_analysis.o -c bank_analysis.c

main.o: main.c types.h main.h vector.h record.h utils.h user.h
	$(CC) $(CFLAGS) -o main.o -c main.c

errors.o: errors.c errors.h
	$(CC) $(CFLAGS) -o errors.o -c errors.c

vector.o: vector.c record.h vector.h types.h
	$(CC) $(CFLAGS) -o vector.o -c vector.c

record.o: record.c record.h types.h utils.h date_support.h main.h
	$(CC) $(CFLAGS) -o record.o -c record.c

date_support.o: date_support.c date_support.h utils.h errors.h
	$(CC) $(CFLAGS) -o date_support.o -c date_support.c

utils.o: utils.c utils.h
	$(CC) $(CFLAGS) -o utils.o -c utils.c

setup.o: setup.c utils.h
	$(CC) $(CFLAGS) -o setup.o -c setup.c

clean:
	rm *.o
```

Para compilar, es necesario que todos los archivos se encuentren en una misma carpeta, llamada makefile o Makefile, y se compilan desde la terminal a partir del comando *make*:
El programa de análisis bancario debe correrse a partir del siguiente comando:
./bank_analysis –fmt <format> -out <file out> -in <file in> -ti<initial time> -tf <final time>

Para el correcto funcionamiento el campo <*format*> debe tomar los valores de *CSV, XML o HTML*. Respecto a los tiempos ingresado, el tiempo inicial debe ser menor al tiempo final, y ambos deben ser mayor a cero (Nota: los tiempos ingresados deben estar en tiempo UNIX)

Anexo 2: Códigos fuente

Main

```
/*bank_analysis.c-------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "main.h"
#include "types.h"
#include "errors.h"
#include "utils.h"

setup_t *setup;

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

    FILE* f_in;
    FILE* f_out;

    status_t st;

    if((st=process_arguments(argc, argv))!=OK)
    {
        print_error(st);
        return st;
    }


    if((f_in = fopen(setup->input_file,"rt"))==NULL)
    {
        print_error(ERROR_OPEN_INVENTORY_FILE);
        return ERROR_OPEN_INVENTORY_FILE;
    }

    if((f_out = fopen(setup->output_file,"wt"))==NULL)
    {
        print_error(ERROR_OPEN_NEW_INVENTORY_FILE);
        fclose(f_in);
        return ERROR_OPEN_NEW_INVENTORY_FILE;
    }



    if((st=process_transactions(f_in,f_out))!=OK)
    {
      print_error(st);
        fclose(f_in);
```

```
        fclose(f_out);
        remove(setup->output_file);
        return st;
    }

    fclose(f_in);
    if(fclose(f_out)==EOF)
    {
        print_error(ERROR_DISK_SPACE);
        return ERROR_DISK_SPACE;
    }
    return OK;
}
```

HEADERS

```
/*main.h----------------------------------------*/

#ifndef MAIN__H
#define MAIN__H

#include <stdio.h>
#include "types.h"


#define MAX_ARGS 11
#define CMD_ARG_NAME 0
#define CMD_ARG_FORMAT_FLAG 1
#define CMD_ARG_FORMAT 2
#define ARG_FORMAT_FLAG "-fmt"
#define CMD_ARG_OUT_FILE_NAME_FLAG 3
#define CMD_ARG_OUT_FILE_NAME 4
#define ARG_OUT_FILE_NAME_FLAG "-out"
#define CMD_ARG_INPUT_FILE_NAME_FLAG 5
#define CMD_ARG_INPUT_FILE_NAME 6
#define ARG_INPUT_FILE_NAME_FLAG "-in"
#define CMD_ARG_INITIAL_TIME_FLAG 7
#define CMD_ARG_INITIAL_TIME 8
#define ARG_INITIAL_TIME_FLAG "-t1"
#define CMD_ARG_FINAL_TIME_FLAG 9
#define CMD_ARG_FINAL_TIME 10
#define ARG_FINAL_TIME_FLAG "-t2"

#define DOC_TYPE_STR_CSV "csv"
#define DOC_TYPE_STR_XML "xml"
#define DOC_TYPE_STR_HTML "html"

#define FILE_HEADER_CSV "header_CSV.txt"
#define FILE_HEADER_XML "header_XML.txt"
#define FILE_HEADER_HTML "header_HTML.txt"

#define FILE_FOOTER_CSV "footer_CSV.txt"
#define FILE_FOOTER_XML "footer_XML.txt"
#define FILE_FOOTER_HTML "footer_HTML.txt"

status_t process_arguments(int argc, char*argv[]);
status_t validate_arguments(int argc, char *argv[]);
status_t load_setup(char* argv[]);
status_t process_transactions(FILE* f_in,FILE* f_out);
status_t load_header(char **str);
status_t load_footer(char**str);

#endif
```

```
/*utils.h-------------------------------------------------------*/
#ifndef UTILS__H
#define UTILS__H

#include <stdio.h>
#include "types.h"

#define INIT_LINE_CHOP 50
#define CHOP_LINE_SIZE 50

#define DELIMITER_CSV_IN ';'
status_t read_file(FILE* f_in,char** str);
status_t read_line(FILE* f_in,char** line,bool_t* is_eof);
status_t split_line(char* record_line,char*** array_string,size_t* length, char
del);
char * strdup(const char *c);
status_t delete_array_string(char*** array_string,size_t* length);

#endif


/*user.h--------------------------------------------------------*/
#ifndef USER__H
#define USER__H
#include <stdio.h>

#define XML_OPEN_USER "<usuario>"
#define XML_CLOSE_USER "</usuario>"
#define XML_OPEN_ID_USER "<id>"
#define XML_CLOSE_ID_USER "</id>"
#define XML_OPEN_INGRESOS "<ingresos"
#define XML_CLOSE_INGRESOS "</ingresos>"
#define XML_OPEN_EGRESOS "<egresos>"
#define XML_CLOSE_EGRESOS "</egresos>"

#define DELIMITER_HTML '|'
#define DELIMITER_CSV_OUT ','

#define HTML_OPEN_ROW "<tr>"
#define HTML_CLOSE_ROW "</tr>"
#define HTML_OPEN_DATA "<td>"
#define HTML_CLOSE_DATA "</td>"

#endif
```

```
/*vector.h-------------------------------------------------*/
#ifndef VECTOR__H
#define VECTOR__H
#include <stdio.h>
#include "types.h"

#define VECTOR_INIT_CHOP 10
#define GROWTH_FACTOR 2

typedef struct{
     void** elements;
     size_t size;
     char* header;
     char* footer;
     destructor_t destructor;
     printer_t printer;
     comparator_t comparator;
}ADT_Vector_t;

status_t ADT_Vector_new(ADT_Vector_t** v);
status_t ADT_Vector_destroy(ADT_Vector_t** v);

status_t ADT_Vector_set_element_at_position(ADT_Vector_t** v,void * element,
size_t position);
status_t ADT_Vector_set_header(ADT_Vector_t* v, char* header);
status_t ADT_Vector_set_footer(ADT_Vector_t* v,char* footer);
status_t ADT_Vector_set_destructor(ADT_Vector_t* v,destructor_t destructor);
status_t ADT_Vector_set_printer(ADT_Vector_t* v,printer_t printer);
status_t ADT_Vector_set_comparator(ADT_Vector_t* v, comparator_t comparator);

void* ADT_Vector_get_element(ADT_Vector_t* v, size_t i);
size_t ADT_Vector_get_size(ADT_Vector_t* v);
char* ADT_Vector_get_header(ADT_Vector_t* v);
char* ADT_Vector_get_footer(ADT_Vector_t* v);
destructor_t ADT_Vector_get_destructor(ADT_Vector_t* v);
printer_t ADT_Vector_get_printer(ADT_Vector_t* v);

status_t ADT_Vector_search(ADT_Vector_t* v,size_t id,bool_t* is_exist);
status_t ADT_Vector_remove(ADT_Vector_t* v,size_t position,void** element);

status_t ADT_Vector_sort(ADT_Vector_t* v, comparator_t comparator);
status_t ADT_Vector_export(ADT_Vector_t* v,FILE* f_out);
status_t ADT_Vector_advance_elements(ADT_Vector_t *v, size_t *len);

#endif
```

```
/*record.h-------------------------------------------------*/
#ifndef RECORD__H
#define RECORD__H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"

#define DATE_STR_LENGTH 17
#define TOTAL_FIELDS 5
#define LINE_INIT_CHOP 100
#define LINE_INCREMENT 50

#define TRANSACTION_FIELD_POS 0
#define USER_FIELD_POS 1
#define DATE_FIELD_POS 2
#define AMMOUNT_FIELD_POS 3
#define DESCRIPTION_FIELD_POS 4

#define MAX_PRINTERS 3

typedef struct{
    size_t id_user;
    float incomes;
    float expenses;
    time_t unix_time;
}ADT_Record_t;

status_t ADT_Record_new(ADT_Record_t** r);
status_t ADT_Record_destroy(ADT_Record_t** r);

status_t ADT_Record_read(FILE* f_in,ADT_Record_t** r,bool_t* is_eof);

status_t ADT_Record_set_id_user(ADT_Record_t* r,size_t id_user);
status_t ADT_Record_set_incomes(ADT_Record_t* r,float incomes);
status_t ADT_Record_set_expenses(ADT_Record_t* r,float expenses);
status_t ADT_Record_set_time(ADT_Record_t* r,time_t time);

size_t ADT_Record_get_id_user(ADT_Record_t* r);
float ADT_Record_get_id_incomes(ADT_Record_t* r);
float ADT_Record_get_id_expenses(ADT_Record_t* r);
time_t ADT_Record_get_time(ADT_Record_t* r);

int ADT_Record_compare_by_expenses(const void** r1,const void** r2);

status_t ADT_Record_add(ADT_Record_t** r_old,ADT_Record_t* r_new);

status_t ADT_Record_export_as_CSV(ADT_Record_t* r,FILE* f_out);
status_t ADT_Record_export_as_XML(ADT_Record_t* r,FILE* f_out);
status_t ADT_Record_export_as_HTML(ADT_Record_t* r,FILE* f_out);
```

```
status_t _ADT_Record_build_record (char **array_string, ADT_Record_t **r);

extern printer_t printers[MAX_PRINTERS];

#endif

/*errors.h-------------------------------------------------*/
#ifndef ERRORS__H
#define ERRORS__H

#include <stdio.h>
#include "types.h"

#define MSG_OK "El programa finalizo"
#define MSG_ERROR "Se ha producido un error"
#define MSG_ERROR_OPEN_INVENTORY_FILE "NO SE PUDO ABRIR EL ARCHIVO DE
INVENTARIO"
#define MSG_ERROR_OPEN_NEW_INVENTORY_FILE "NO SE PUDO CREAR EL NUEVO ARCHIVO"
#define MSG_ERROR_DISK_SPACE "NO HAY SUFICIENTE ESPACIO PARA GUARDAR LOS
CAMBIOS"
#define MSG_ERROR_NULL_POINTER "SE RECIBIO UN PUNTERO NULO"
#define MSG_ERROR_INVALID_ARG_NUMBER "CANTIDAD DE ARGUMENTOS INVALIDA"
#define MSG_ERROR_INVALID_ARG_FMT_FLAG "BANDERA DE FORMATO INVALIDA"
#define MSG_ERROR_INVALID_ARG_OUT_FILE_NAME_FLAG "BANDERA DE ARCHIVO DE SALIDA
INVALIDA"
#define MSG_ERROR_INVALID_ARG_INPUT_FILE_NAME_FLAG "BANDERA DE ARCHIVO DE
ENTRADA INVALIDA"
#define MSG_ERROR_INVALID_ARG_INITIAL_TIME_FLAG "BANDERA DE TIEMPO INICIAL
INVALIDA"
#define MSG_ERROR_INVALID_ARG_FINAL_TIME_FLAG "BANDERA DE TIEMPO FINAL INVALIDA"
#define MSG_ERROR_INVALID_ARG_FORMAT "EL FORMATO INGRESADO ES INVALIDO"
#define MSG_ERROR_CONVERSION_ARG_FINAL_TIME "EL TIEMPO FINAL INGRESADO ES
INVALIDO"
#define MSG_ERROR_CONVERSION_ARG_INITIAL_TIME "EL TIEMPO INICIAL INGRESADO ES
INVALIDO"
#define MSG_ERROR_INVALID_ARG_TIMES "LOS TIEMPO INGRESADOS NO TIENEN INTERVALO
VALIDO"
#define MSG_ERROR_MEMORY "MEMORIA INSUFICIENTE"
#define MSG_ERROR_CONVERSION_ID "NO SE PUDO CONVERTIR EL ID"
#define MSG_ERROR_CONVERSION_MONEY "NO SE PUDO CONVERTIR EL MONTO"
#define MSG_ERROR_CONVERSION_TIME "NO SE PUDO CONVERTIR LA FECHA"
#define MSG_ERROR_INVALID_DATE "LA FECHA DEL REGISTRO ES INVALIDA"
#define MSG_ERROR_UNKNOWN_DOC_TYPE "EL FORMATO DE ARCHIVO RECIBIDO NO SE
ENCUENTRA REGISTRADO"

#define MAX_ERRORS 30
status_t print_error(status_t st);

#endif
/*types.h--------------------------------------------------*/
```

```
#ifndef TYPES__H
#define TYPES__H

#include <stdio.h>
#include <time.h>

typedef enum{
    OK,
    ERROR,
    ERROR_OPEN_INVENTORY_FILE,
    ERROR_OPEN_NEW_INVENTORY_FILE,
    ERROR_DISK_SPACE,
    ERROR_NULL_POINTER,
    ERROR_INVALID_ARG_NUMBER,
    ERROR_INVALID_ARG_FMT_FLAG,
    ERROR_INVALID_ARG_OUT_FILE_NAME_FLAG,
    ERROR_INVALID_ARG_INPUT_FILE_NAME_FLAG,
    ERROR_INVALID_ARG_INITIAL_TIME_FLAG,
    ERROR_INVALID_ARG_FINAL_TIME_FLAG,
    ERROR_INVALID_ARG_FORMAT,
    ERROR_CONVERSION_ARG_FINAL_TIME,
    ERROR_CONVERSION_ARG_INITIAL_TIME,
    ERROR_INVALID_ARG_TIMES,
    ERROR_MEMORY,
    ERROR_CONVERSION_ID,
    ERROR_CONVERSION_MONEY,
    ERROR_CONVERSION_TIME,
    ERROR_INVALID_DATE,
    ERROR_UNKNOWN_DOC_TYPE
}status_t;

typedef enum{
    TRUE,
    FALSE
}bool_t;

/*El tipo de documento se encuentra numerado (0-2) para
luego ser utilizadas las siguientes macros en un arreglo.*/

typedef enum{
    DOC_TYPE_CSV=0,
    DOC_TYPE_XML=1,
    DOC_TYPE_HTML=2
}doc_type_t;

typedef struct{
    doc_type_t doc_type;
    time_t initial_time;
    time_t final_time;
    char *input_file;
    char *output_file;
```

```
}setup_t;

typedef enum{
    DD_MM_YYYY_HH_MM_SS=0
}date_format_t;

typedef status_t (*destructor_t)(void**);
typedef status_t (*printer_t)(const void* ,FILE * );
typedef int (*comparator_t) (const void*, const void *);
typedef status_t (*date_parser_t) (char *str, time_t *t);

#endif

/*date_support.h-------------------------------------------------*/

#ifndef DATE_SUPPORT__H
#define DATE_SUPPORT__H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "utils.h"
#include "errors.h"

#define DATE_DELIMITER_SLASH '/'
#define DATE_DELIMITER_HYPHEN '-'
#define DATE_DELIMITER_DOT '.'

#define TIME_DELIMITER ':'
#define SEPARATION_DELIMITER ' '
#define TOTAL_DATE_FIELDS 6

#define MAX_STR_LEN 5

#define MAX_DATE_CONVERTERS 5

status_t validate_format_DD_MM_YYYY_HH_MM_SS(char *str);
status_t parse_DD_MM_YYYY_HH_MM_SS_date (char *str, time_t *unix_time);
status_t get_time(char *str,time_t *time, date_format_t format);

#endif
```

FUNCIONES

```
/*main.c-----------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "main.h"
#include "vector.h"
#include "record.h"
#include "utils.h"
#include "user.h"

/*La siguiente función es responsable del tratamiento de los argumentos pasados
por línea de órdenes. Incluye la validación de los mismos y la carga de estos en
una estructura de configuración*/

status_t process_arguments(int argc, char*argv[])
{
      status_t st;
      if((st=validate_arguments(argc, argv))!=OK)
            return st;

      if((st=load_setup(argv))!=OK)
            return st;

      return OK;
}

/*Esta función es la encargada de corroborar que los argumentos
pasados por línea de órdenes sean correctos*/

status_t validate_arguments(int argc,char* argv[])
{
    if(argv==NULL)
        return ERROR_NULL_POINTER;

    if(argc!=MAX_ARGS)
        return ERROR_INVALID_ARG_NUMBER;

    if(strcmp(argv[CMD_ARG_FORMAT_FLAG],ARG_FORMAT_FLAG))
        return ERROR_INVALID_ARG_FMT_FLAG;

    if(strcmp(argv[CMD_ARG_OUT_FILE_NAME_FLAG],ARG_OUT_FILE_NAME_FLAG))
        return ERROR_INVALID_ARG_OUT_FILE_NAME_FLAG;

    if(strcmp(argv[CMD_ARG_INPUT_FILE_NAME_FLAG],ARG_INPUT_FILE_NAME_FLAG))
        return ERROR_INVALID_ARG_INPUT_FILE_NAME_FLAG;

    if(strcmp(argv[CMD_ARG_INITIAL_TIME_FLAG],ARG_INITIAL_TIME_FLAG))
```

```
        return ERROR_INVALID_ARG_INITIAL_TIME_FLAG;

    if(strcmp(argv[CMD_ARG_FINAL_TIME_FLAG],ARG_FINAL_TIME_FLAG))
        return ERROR_INVALID_ARG_FINAL_TIME_FLAG;

    return OK;
}
/*La siguiente función se ocupa de guardar en una estructura parte de
los argumentos pasados por línea de órdenes para que posteriormente
sean utilizados*/

status_t load_setup(char* argv[])
{
      char*end_ptr;

      setup=(setup_t*)malloc(sizeof(setup_t));

      if(argv==NULL)
            return ERROR_NULL_POINTER;

      if(!strcmp(argv[CMD_ARG_FORMAT],DOC_TYPE_STR_CSV))
            setup->doc_type=DOC_TYPE_CSV;

      else if(!strcmp(argv[CMD_ARG_FORMAT],DOC_TYPE_STR_XML))
            setup->doc_type=DOC_TYPE_XML;

      else if(!strcmp(argv[CMD_ARG_FORMAT],DOC_TYPE_STR_HTML))
            setup->doc_type=DOC_TYPE_HTML;

    else
        return ERROR_INVALID_ARG_FORMAT;

    (setup->final_time)=(time_t)strtoul(argv[CMD_ARG_FINAL_TIME], &end_ptr, 10);
    if(*end_ptr)
        return ERROR_CONVERSION_ARG_FINAL_TIME;

    (setup->initial_time)=(time_t)strtoul(argv[CMD_ARG_INITIAL_TIME], &end_ptr,
10);
    if(*end_ptr)
        return ERROR_CONVERSION_ARG_INITIAL_TIME;

    if((setup->initial_time) > (setup->final_time))
        return ERROR_INVALID_ARG_TIMES;

    setup->input_file=argv[CMD_ARG_INPUT_FILE_NAME];

    setup->output_file=argv[CMD_ARG_OUT_FILE_NAME];

    return OK;
}
/*A continuación se encuentra el código de una función que lleva a
```

```
cabo la función principal del programa, la cual es, procesar un
archivo de transacciones a partir de opciones previamente cargadas,
para luego imprimirlo en un formato también previamente
especificado*/

status_t process_transactions(FILE* f_in,FILE* f_out)
{
    status_t st;
    ADT_Vector_t* vector;
    ADT_Record_t* record;
    void* element;
    size_t id;
    time_t time;
    bool_t eof;
    bool_t exists;
    char *line;
      char *header;
      char *footer;
      printer_t printer;

    if(f_in==NULL||f_out==NULL)
        return ERROR_NULL_POINTER;

    if((st=ADT_Vector_new(&vector))!=OK)
        return st;

    if((st=ADT_Vector_set_destructor(vector,(destructor_t)
ADT_Record_destroy))!=OK)
            return st;

      if((st=ADT_Vector_set_comparator(vector, (comparator_t)
ADT_Record_compare_by_expenses))!=OK)
            return st;

      if((st=load_header(&header))!=OK)
            return st;


      if((st=load_footer(&footer))!=OK)
            return st;

      printer=printers[setup->doc_type];

    if((st=ADT_Vector_set_header(vector,header))!=OK)
        return st;
    if((st=ADT_Vector_set_footer(vector,footer))!=OK)
        return st;

    if((st=ADT_Vector_set_printer(vector,printer))!=OK)
        return st;
```

```
    if((st=read_line(f_in,&line,&eof))!=OK)
        return st;

    while(eof==FALSE)
    {
        if((st=ADT_Record_read(f_in,&record,&eof))!=OK){
            free(line);
            return st;
        }

            if(eof==TRUE)
                  break;

        time=ADT_Record_get_time(record);

        if(setup->initial_time<time && time<setup->final_time)
        {

            id=ADT_Record_get_id_user(record);

            if((st=ADT_Vector_search(vector,id,&exists))!=OK){
                free(line);
                return st;
            }

            if(exists==TRUE)
            {

                if((st=ADT_Vector_remove(vector,id,&element))!=OK){
                    free(line);
                    return st;
                }

                if((st=ADT_Record_add((ADT_Record_t**)&element,record))!=OK){
                    free(line);
                    return st;
                }

                if((st=ADT_Vector_set_element_at_position(&vector,record,
record->id_user))!=OK){
                    free(line);
                    return st;
                }

            }else{

                  if((st=ADT_Vector_set_element_at_position(&vector,record,
record->id_user))!=OK){
                    free(line);
                      return st;
                  }
```

```
            }
          }
    }

    if((st=ADT_Vector_sort(vector,
(comparator_t)ADT_Record_compare_by_expenses))!=OK)
        return st;

    if((st=ADT_Vector_export(vector,f_out))!=OK)
        return st;

    if((st=ADT_Vector_destroy(&vector))!=OK)
        return st;

    return OK;
}

/*Esta función es la encargada de cargar un determinado
encabezado dependiendo del tipo de documento*/

      status_t load_header(char**str)
{
      FILE* f_header;
      status_t st;

      if(str==NULL||setup==NULL)
            return ERROR_NULL_POINTER;

      switch(setup->doc_type)
      {
      case DOC_TYPE_CSV:

              if((f_header = fopen(FILE_HEADER_CSV,"rt"))==NULL)
            return ERROR_OPEN_INVENTORY_FILE;

              if((st=read_file(f_header,str))!=OK){
                fclose(f_header);
                        return st;
            }

                  break;

            case DOC_TYPE_XML:

              if((f_header = fopen(FILE_HEADER_XML,"rt"))==NULL)
                  return ERROR_OPEN_INVENTORY_FILE;

              if((st=read_file(f_header,str))!=OK){
                fclose(f_header);
                        return st;
            }
```

```
                break;

            case DOC_TYPE_HTML:

              if((f_header = fopen(FILE_HEADER_HTML,"rt"))==NULL)
            return ERROR_OPEN_INVENTORY_FILE;

            if((st=read_file(f_header,str))!=OK){
                fclose(f_header);
                        return st;
            }

                break;

         default:

            return ERROR_UNKNOWN_DOC_TYPE;
       }

    return OK;
}

/*Esta función es la encargada de cargar un determinado
pie de pagina dependiendo del tipo de documento*/

status_t load_footer(char**str)
{
    FILE* f_footer;
    status_t st;


    if(str==NULL||setup==NULL)
            return ERROR_NULL_POINTER;

    switch(setup->doc_type)
    {
      case DOC_TYPE_CSV:

              if((f_footer = fopen(FILE_FOOTER_CSV,"rt"))==NULL)
                  return ERROR_OPEN_INVENTORY_FILE;

              if((st=read_file(f_footer,str))!=OK)
                        return st;

                  break;

            case DOC_TYPE_XML:

              if((f_footer = fopen(FILE_FOOTER_XML,"rt"))==NULL)
                  return ERROR_OPEN_INVENTORY_FILE;
```

```
            if((st=read_file(f_footer,str))!=OK)
                      return st;

                  break;

            case DOC_TYPE_HTML:

              if((f_footer = fopen(FILE_FOOTER_HTML,"rt"))==NULL)
                  return ERROR_OPEN_INVENTORY_FILE;

              if((st=read_file(f_footer,str))!=OK)
                      return st;

                  break;

        default:

            return ERROR_UNKNOWN_DOC_TYPE;
      }

      return OK;
}

/*vector.c-----------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include "types.h"
#include "vector.h"

/*Esta primitiva se ocupa de pedir memoria para un tipo de dato
abstracto definido como ADT_Vector_t , también para una cantidad
inicial de elements (campo del ADT)*/

status_t ADT_Vector_new(ADT_Vector_t** v)
{
    status_t st;

    if(v==NULL)
        return ERROR_NULL_POINTER;

    if(((*v)=(ADT_Vector_t*)malloc(sizeof(ADT_Vector_t)))==NULL)
        return ERROR_MEMORY;

    if(((*v)->elements=(void**)calloc(VECTOR_INIT_CHOP,sizeof(void*)))==NULL)
    {
        if((st=ADT_Vector_destroy(v))!=OK)
        return ERROR_MEMORY;
    }
    (*v)->size=VECTOR_INIT_CHOP;
```

```
    return OK;
}


/*La siguiente primitiva lleva a cabo la destrucción del tipo de
dato abstracto definido como ADT_Vector_t , liberando primero la
memoria de sus campos internos ( elements ) y luego la del ADT en
cuestión*/

status_t ADT_Vector_destroy(ADT_Vector_t** v)
{
    size_t i;
    status_t st;

    if(v==NULL)
        return ERROR_NULL_POINTER;

    for(i=0;i<(*v)->size;i++)
    {
        if(((*v)->elements[i])!=NULL)
        {
            if((st=(*((*v)->destructor))(&((*v)->elements[i])))!=OK)
                return st;
            (*v)->elements[i]=NULL;
        }
    }

    free((*v)->elements);
    (*v)->elements=NULL;
    (*v)->printer=NULL;
    (*v)->destructor=NULL;
    (*v)->comparator=NULL;
    free(*v);
    (*v)=NULL;

    return OK;
}


/*A continuación se encuentra el código de una primitiva que es la
encargada de agregar un element al tipo de dato abstracto definido
como ADT_Vector_t según una posición*/

status_t ADT_Vector_set_element_at_position(ADT_Vector_t** v,void * element,
size_t position)
{
    void **aux;
    size_t prev_size, i;

    if(v==NULL||element==NULL)
```

```
        return ERROR_NULL_POINTER;

    while(position>=((*v)->size))
    {

if((aux=(void**)realloc((*v)->elements,sizeof(void*)*(GROWTH_FACTOR*((*v)->size)
)))==NULL)
        {
            ADT_Vector_destroy(v);
            free(aux);
            return ERROR_MEMORY;
        }
        (*v)->elements=aux;
        prev_size=(*v)->size;
        (*v)->size=GROWTH_FACTOR*(*v)->size;

        for(i=prev_size; i <= (*v)->size; i++){
            ((*v)->elements)[i]=NULL;
        }
    }

    (*v)->elements[position]=element;

    return OK;
}

/*La siguiente primitiva es la encargada de la operación de
establecer, para un tipo de dato abstracto definido como
ADT_Vector_t , un valor para el campo header */

status_t ADT_Vector_set_header(ADT_Vector_t* v, char* header)
{
    if(v==NULL||header==NULL)
        return ERROR_NULL_POINTER;

    v->header=header;

    return OK;
}

/*A continuación se encuentra el código de una primitiva que se ocupa
de la operación de establecer, para un tipo de dato abstracto
definido como ADT_Vector_t , un valor para el campo footer */

status_t ADT_Vector_set_footer(ADT_Vector_t* v, char* footer)
{
    if(v==NULL||footer==NULL)
        return ERROR_NULL_POINTER;

    v->footer=footer;
```

```
    return OK;
}


/*Esta primitiva es la encargada de la operación de establecer, para
un tipo de dato abstracto definido como ADT_Vector_t , un valor para
el campo destructor */

status_t ADT_Vector_set_destructor(ADT_Vector_t* v,destructor_t destructor)
{
    if(v==NULL)
        return ERROR_NULL_POINTER;

    v->destructor=destructor;

    return OK;
}

/*La siguiente primitiva se ocupa de la operación de establecer, para
un tipo de dato abstracto definido como ADT_Vector_t , un valor para
el campo printer */

status_t ADT_Vector_set_printer(ADT_Vector_t* v,printer_t printer)
{
    if(v==NULL)
        return ERROR_NULL_POINTER;

    v->printer=printer;

    return OK;
}


/*La siguiente primitiva se encarga de la operación de establecer,
para un tipo de dato abstracto definido como ADT_Vector_t , un valor
para el campo comparator */

status_t ADT_Vector_set_comparator(ADT_Vector_t* v, comparator_t comparator){

      if(v==NULL)
            return ERROR_NULL_POINTER;

      v->comparator=comparator;
      return OK;
}


/*A continuación se encuentra el código de una primitiva que lleva a
cabo la operación de obtener, de un tipo de dato abstracto definido
como ADT_Vector_t , el valor del campo element de una determinada posición */
```

```
void* ADT_Vector_get_element_at_position(ADT_Vector_t* v, size_t i)
{
    if(v==NULL)
        return NULL;

    return (v->elements)[i];
}

/*La siguiente primitiva lleva a cabo la operación de obtener, de un
tipo de dato abstracto definido como ADT_Vector_t , el valor del campo
size */

size_t ADT_Vector_get_size(ADT_Vector_t* v)
{
    if(v==NULL)
        return ERROR_NULL_POINTER;

    return v->size;
}


/*La siguiente primitiva lleva a cabo la operación de obtener, de un
tipo de dato abstracto definido como ADT_Vector_t , el valor del campo
header */

char* ADT_Vector_get_header(ADT_Vector_t* v)
{
    return v->header;
}


/*A continuación se encuentra el código de una primitiva que es la
encargada de la operación de obtener, de un tipo de dato abstracto
definido como ADT_Vector_t , el valor del campo footer */

char* ADT_Vector_get_footer(ADT_Vector_t* v)
{
    return v->footer;
}


/*Esta primitiva lleva a cabo la operación de obtener, de un tipo de
dato abstracto definido como ADT_Vector_t , el valor del campo
destructor */

destructor_t ADT_Vector_get_destructor(ADT_Vector_t* v)
{
    if(v==NULL)
        return NULL;

    return v->destructor;
```

```
}

/*La siguiente primitiva es la encargada de la operación de obtener,
de un tipo de dato abstracto definido como ADT_Vector_t , el valor del
campo printer */

printer_t ADT_Vector_get_printer(ADT_Vector_t* v)
{
    if(v==NULL)
        return NULL;

    return v->printer;
}

/*A continuación se encuentra el código de una primitiva del tipo de
dato abstracto definido como ADT_Vector_t que se ocupa de determinar
si existe un element con cierto id cargado en el vector*/

status_t ADT_Vector_search(ADT_Vector_t* v,size_t position,bool_t* exists)
{
    if(v==NULL||exists==NULL)
        return ERROR_NULL_POINTER;

    *exists=FALSE;

      if((v->size)<=position){
            *exists=FALSE;
            return OK;
      }
      if((v->elements[position])!=NULL){
                  *exists=TRUE;
            return OK;
      }
    return OK;
}


/*Esta primitiva del tipo de dato abstracto definido como
ADT_Vector_t es la encargada de extraer el element ubicado en
position y devolverlo por interfaz, reemplazando el valor en el vector por
NULL*/

status_t ADT_Vector_remove(ADT_Vector_t* v,size_t position,void** element)
{
    if(v==NULL||element==NULL)
        return ERROR_NULL_POINTER;

    *element=(v->elements)[position];
    (v->elements)[position]=NULL;

    return OK;
```

```
}


/*La siguiente primitiva se encarga de ordenar el vector*/

status_t ADT_Vector_sort(ADT_Vector_t* v, comparator_t comparator)
{
    size_t total;
    status_t st;

    if((st=ADT_Vector_advance_elements(v, &total))!=OK)
            return st;

    qsort( v->elements , total, sizeof(void*), comparator);

    return OK;
}

/*A continuación se encuentra el código de una primitiva que lleva a cabo la
operación de imprimir una variable de tipo de dato abstracto definido como
ADT_Vector_t */

status_t ADT_Vector_export(ADT_Vector_t* v,FILE* f_out)
{
    size_t i;
    status_t st;

    if(v==NULL||f_out==NULL)
        return ERROR_NULL_POINTER;
    if((v->header)!=NULL)
        fprintf(f_out,"%s\n",v->header);

    for(i=0;i<(v->size);i++)
    {
        if((v->elements[i])!=NULL)
        {
            if((st=(*(v->printer))(v->elements[i],f_out))!=OK)
           return st;
        }
    }
    if((v->footer)!=NULL)
        fprintf(f_out,"%s\n", v->footer);

    return OK;
}


/*Esta primitiva se encarga de mover todos los elements del vector al
inicio*/

status_t ADT_Vector_advance_elements(ADT_Vector_t *v, size_t *len)
```

```
{
    size_t i,j;
    void *aux;
    *len=0;

    if(v==NULL||len==NULL)
        return ERROR_NULL_POINTER;

    for(i=0, j=v->size-1;i<v->size;i++)
    {
      if((v->elements)[i]==NULL)
      {

            while((v->elements)[j]==NULL){
                j--;
            }

            if(j<=i){
                *len=j+1;
                return OK;
            }

            aux=(v->elements)[i];
            (v->elements)[i]=(v->elements)[j];
            (v->elements)[j]=aux;
      }
    }
    *len=j;
    return OK;
}

/*record.c-----------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "record.h"
#include "utils.h"
#include "types.h"
#include "date_support.h"
#include "user.h"

printer_t printers [MAX_PRINTERS]={
    (printer_t)ADT_Record_export_as_CSV,
    (printer_t)ADT_Record_export_as_XML,
    (printer_t)ADT_Record_export_as_HTML
};


/*Esta primitiva es la encargada de pedir memoria para un tipo de
dato abstracto definido como ADT_Record_t */
```

```
status_t ADT_Record_new(ADT_Record_t** r)
{
    if(r==NULL)
        return ERROR_NULL_POINTER;

    if(((*r)=(ADT_Record_t*)malloc(sizeof(ADT_Record_t)))==NULL)
        return ERROR_MEMORY;

    return OK;
}

/*La siguiente primitiva se ocupa de destruir el tipo de dato
abstracto definido como ADT_Record_t */

status_t ADT_Record_destroy(ADT_Record_t** r)
{
    if(r==NULL)
        return ERROR_NULL_POINTER;

    free(*r);
    (*r)=NULL;

    return OK;
}


/*A continuación se encuentra el código de una primitiva que lleva a
cabo lectura de un ADT_Record_t a partir de un archivo de entrada*/

status_t ADT_Record_read(FILE* f_in,ADT_Record_t** r,bool_t* is_eof)
{
    status_t st;
    char* line;
    char** array_string;
    size_t length;

    if((st=read_line(f_in,&line,is_eof))!=OK){
        return st;
    }

    if(*is_eof==TRUE)
        return OK;

    if((st=split_line(line,&array_string, &length, DELIMITER_CSV_IN))!=OK){
        return st;
    }

    free(line);

    if((st=_ADT_Record_load_record(array_string, r))!=OK){
```

```
        delete_array_string(&array_string, &length);
        return st;
    }

    return OK;
}

/*Esta primitiva se ocupa de establecer, para un tipo de dato
abstracto definido como ADT_Record_t , un valor para el campo
id_user */

status_t ADT_Record_set_id_user(ADT_Record_t* r,size_t id_user)
{
    if(r==NULL)
        return ERROR_NULL_POINTER;
    r->id_user=id_user;
    return OK;
}

/*La siguiente primitiva lleva a cabo establecer, para un tipo de
dato abstracto definido como ADT_Record_t , un valor para el campo
incomes */

status_t ADT_Record_set_incomes(ADT_Record_t* r,float incomes)
{
    if(r==NULL)
        return ERROR_NULL_POINTER;

    r->incomes=incomes;
    return OK;
}

/*A continuación se encuentra el código de una primitiva que es la
encargada de establecer, para un tipo de dato abstracto definido como
ADT_Record_t , un valor para el campo expenses */

status_t ADT_Record_set_expenses(ADT_Record_t* r,float expenses)
{
    if(r==NULL)
        return ERROR_NULL_POINTER;

    r->expenses=expenses;
    return OK;
}


/*Esta primitiva lleva a cabo establecer, para un tipo de dato
abstracto definido como ADT_Record_t , un valor para el campo time */

status_t ADT_Record_set_time(ADT_Record_t* r,time_t new_time)
{
```

```
    if(r==NULL)
        return ERROR_NULL_POINTER;

    r->unix_time=new_time;
    return OK;
}

/*La siguiente primitiva es la encargada de obtener, de un tipo de
dato abstracto definido como ADT_Record_t , el valor del campo
id_user */

size_t ADT_Record_get_id_user(ADT_Record_t* r)
{
    if(r==NULL)
        return OK;

    return r->id_user;
}

/*A continuación se encuentra el código de una primitiva que se ocupa
de obtener, de un tipo de dato abstracto definido como ADT_Record_t ,
el valor del campo incomes */

float ADT_Record_get_incomes(ADT_Record_t* r)
{
    if(r==NULL)
        return OK;

    return r->incomes;
}

/*Esta primitiva es la encargada de obtener, de un tipo de dato
abstracto definido como ADT_Record_t , el valor del campo expenses */

float ADT_Record_get_expenses(ADT_Record_t* r)
{
    if(r==NULL)
        return OK;

    return r->expenses;
}

/*Esta primitiva es la encargada de obtener, de un tipo de dato
abstracto definido como ADT_Record_t , el valor del campo time */

time_t ADT_Record_get_time(ADT_Record_t* r)
{
    if(r==NULL)
        return OK;

    return r->unix_time;
```

```
}

/*La siguiente primitiva realiza la operación de comparar dos tipo de
dato abstracto por medio de su campo expenses */

int ADT_Record_compare_by_expenses(const void** r1, const void** r2)
{
    ADT_Record_t **aux_1, **aux_2;

      if(r1==NULL||r2==NULL)
         return 0;

    aux_1=(ADT_Record_t**)r1;
    aux_2=(ADT_Record_t**)r2;

      if((*aux_1)->expenses>(*aux_2)->expenses){
         return -1;

    }else if((*aux_1)->expenses<(*aux_2)->expenses){
         return 1;

    }else{
        return 0;
    }

    return 0;
}

/*A continuación se encuentra el código de una primitiva que lleva a
cabo la operación de unificar dos tipo de dato abstracto ADT_Record_t
con el mismo id , sumando los campos de incomes y expenses */

status_t ADT_Record_add(ADT_Record_t** r_old,ADT_Record_t* r_new)
{
    status_t st;
    float incomes_old,incomes_new,expenses_old,expenses_new;

    if(r_old==NULL||r_new==NULL)
        return ERROR_NULL_POINTER;

    incomes_old=ADT_Record_get_incomes(*r_old);

    incomes_new=ADT_Record_get_incomes(r_new);

    expenses_old=ADT_Record_get_expenses(*r_old);
    expenses_new=ADT_Record_get_expenses(r_new);

    incomes_new+=incomes_old;
    expenses_new+=expenses_old;

    if((st=ADT_Record_set_incomes(r_new,incomes_new))!=OK)
```

```
        return st;

    if((st=ADT_Record_set_expenses(r_new,expenses_new))!=OK)
        return st;

    if((st=ADT_Record_destroy(r_old))!=OK)
        return st;

    return OK;
}

/*Esta primitiva se ocupa de exportar en formato CSV un tipo de dato
abstracto definido como ADT_Record_t */

status_t ADT_Record_export_as_CSV(ADT_Record_t* r,FILE* f_out)
{
    if(r==NULL||f_out==NULL)
        return ERROR_NULL_POINTER;

    fprintf(f_out,"%ld%c%.2f%c%.2f\n",r->id_user,DELIMITER_CSV_OUT,
r->incomes,DELIMITER_CSV_OUT, r->expenses);

    return OK;
}

/*La siguiente primitiva lleva a cabo exportar en formato XML un tipo
de dato abstracto definido como ADT_Record_t */

status_t ADT_Record_export_as_XML(ADT_Record_t* r,FILE* f_out)
{
    fprintf(f_out,"\t%s\n",XML_OPEN_USER);

fprintf(f_out,"\t\t%s%ld%s\n",XML_OPEN_ID_USER,r->id_user,XML_CLOSE_ID_USER);

fprintf(f_out,"\t\t%s%.2f%s\n",XML_OPEN_INGRESOS,r->incomes,XML_CLOSE_INGRESOS);

fprintf(f_out,"\t\t%s%.2f%s\n",XML_OPEN_EGRESOS,r->expenses,XML_CLOSE_EGRESOS);
    fprintf(f_out,"\t%s\n",XML_CLOSE_USER);

    return OK;
}

/*La siguiente primitiva lleva a cabo exportar en formato HTML un
tipo de dato abstracto definido como ADT_Record_t */

status_t ADT_Record_export_as_HTML(ADT_Record_t* r,FILE* f_out)
{
    fprintf(f_out,"\t\t\t\t%s\n",HTML_OPEN_ROW);
    fprintf(f_out,"\t\t\t\t\t%s%ld%s\n",HTML_OPEN_DATA,r->id_user,
HTML_CLOSE_DATA);
```

```
    fprintf(f_out,"\t\t\t\t\t%s%.2f%s\n",HTML_OPEN_DATA,r->incomes,
HTML_CLOSE_DATA);
    fprintf(f_out,"\t\t\t\t\t%s%.2f%s\n",HTML_OPEN_DATA,r->expenses,
HTML_CLOSE_DATA);
    fprintf(f_out,"\t\t\t\t%s\n",HTML_CLOSE_ROW);

    return OK;
}

/*La siguiente primitiva es privada, y solamente debería ser utilizada por otras
primitivas. Su función es cargar un arreglo de cadenas en un registro.*/

status_t _ADT_Record_load_record (char **array_string, ADT_Record_t **r){

    status_t st;
    char* end_ptr;
    size_t id;
    float money;
    time_t time_record;
    if((st=ADT_Record_new(r))!=OK){
        return st;
    }

    id=(size_t)strtoul(array_string[USER_FIELD_POS], &end_ptr,10);
    if(*end_ptr){
        return ERROR_CONVERSION_ID;
    }

    if((st=ADT_Record_set_id_user(*r,id))!=OK){
        return st;
    }

    money=(float)strtod(array_string[AMMOUNT_FIELD_POS], &end_ptr);
    if(*end_ptr){
        return ERROR_CONVERSION_MONEY;
    }

    if(money>=0)
    {
        if((st=ADT_Record_set_incomes(*r,money))!=OK){
            return st;
        }
        if((st=ADT_Record_set_expenses(*r,0))!=OK){
            return st;
        }
    }

    if(money<0)
    {
        money=money*(-1);
        if((st=ADT_Record_set_expenses(*r,money))!=OK){
```

```
            return st;
        }
        if((st=ADT_Record_set_incomes(*r,0))!=OK){
            return st;
        }
    }

    if((st=get_time(array_string[DATE_FIELD_POS],&time_record,
DD_MM_YYYY_HH_MM_SS))!=OK){
        return st;
    }

    if((st=ADT_Record_set_time(*r,time_record))!=OK)
        return st;

    return OK;
}

/*utils.c-----------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "types.h"
#include "utils.h"


/*A continuación se encuentra el código de una función que es la
encargada de leer un archivo de un archivo de entrada para obtener
los encabezados y pies de paginas, también informa si se llegó al
fin de archivo o no*/

status_t read_file(FILE* f_in,char** str)
{
    size_t alloc_size,used_size;
    char c;
    char *aux,*aux2;

    if(str==NULL)
        return ERROR_NULL_POINTER;

    if((aux=malloc(INIT_LINE_CHOP*sizeof(char)))==NULL)
        return ERROR_MEMORY;

    alloc_size=INIT_LINE_CHOP;
    used_size=0;

    while((c=fgetc(f_in))!=EOF)
    {
        if(used_size>=alloc_size-1)
        {
```

```
            aux2=aux;

if((aux=realloc(aux,(alloc_size+CHOP_LINE_SIZE)*sizeof(char)))==NULL)
            {
                free(aux2);
                return ERROR_MEMORY;
            }
            alloc_size +=CHOP_LINE_SIZE;
        }
        aux[used_size]=c;
        used_size++;
    }
    aux[used_size]='\0';

    if((*str=(char*)malloc(sizeof(char)*used_size))==NULL)
    {
        free(aux);
        return ERROR_MEMORY;
    }

    *str=strdup(aux);
    free(aux);

    return OK;
}


/*A continuación se encuentra el código de una función que es la
encargada de leer una línea de un archivo de entrada, también informa
si se llegó al fin de archivo o no*/

status_t read_line(FILE* f_in,char** line,bool_t* is_eof)
{
    size_t alloc_size,used_size;
    int c;
    char *aux,*aux2;

    if(line==NULL||is_eof==NULL)
        return ERROR_NULL_POINTER;

    if((aux=malloc(INIT_LINE_CHOP*sizeof(char)))==NULL)
        return ERROR_MEMORY;

    alloc_size=INIT_LINE_CHOP;
    used_size=0;

    while((c=fgetc(f_in))!=EOF && c!='\n')
    {
        if(used_size>=alloc_size-1)
        {
            aux2=aux;
```

```
if((aux=realloc(aux,(alloc_size+CHOP_LINE_SIZE)*sizeof(char)))==NULL)
            {
                free(aux2);
                return ERROR_MEMORY;
            }
            alloc_size +=CHOP_LINE_SIZE;
            aux=aux2;
        }
        aux[used_size]=c;
        used_size++;
    }
    aux[used_size]='\0';

    *is_eof=FALSE;

    if(feof(f_in))
    {
        *is_eof=TRUE;
        free(aux);
        return OK;
    }

    if((*line=malloc(sizeof(char)*used_size))==NULL)
    {
        free(aux);
        return ERROR_MEMORY;
    }

    *line=strdup(aux);
    free(aux);

    return OK;
}

/*Esta función lleva a cabo el parseo de una línea en formato CSV
leída previamente*/

status_t split_line(char* record_line,char*** array_string,size_t* length, char
del)
{
    char *p, *q, *line, del_arr[2];
    size_t i;

    if(record_line==NULL||array_string==NULL||length==NULL)
        return ERROR_NULL_POINTER;

    if((line=strdup(record_line))==NULL)
            return ERROR_MEMORY;
```

```
    for(*length=0, i=0; line[i]!='\0'; i++)
    {
        if(line[i]==del)
            (*length)++;
    }

    (*length)++;

    if(((*array_string)= (char**)malloc((*length)*sizeof(char *)))==NULL)
    {
        free(line);
        *length=0;
        return ERROR_MEMORY;
    }

    del_arr[0]=del;
    del_arr[1]='\0';

    for(q=line,i=0;(p=strtok(q, del_arr))!=NULL; q=NULL,i++)
    {
        if(((*array_string)[i]=strdup(p))==NULL)
        {
            free(line);
            delete_array_string(array_string, &i);
            *length=0;
            return ERROR_MEMORY;
        }
    }

    free(line);
    return OK;
}

/*La siguiente función es la encargada de copiar una cadena*/

char * strdup(const char *c)
{
    size_t i;
    char *dup;

    if(c==NULL)
        return NULL;

    if((dup=(char*) malloc((strlen(c)+1)*sizeof(char)))==NULL){
        return NULL;
    }
    for(i=0; (dup[i]=c[i]); i++);
    return dup;
}

/*A continuación se encuentra el código de una función que se ocupa
```

```
de eliminar un arreglo de cadenas creado para la función
split_line ()*/

status_t delete_array_string(char*** array_strings, size_t *length)
{
    size_t j;

    if(array_strings==NULL||length==NULL)
        return ERROR_NULL_POINTER;

    for(j=0; j<(*length); j++)
        free(*array_strings[j]);

    free(*array_strings);
    *array_strings=NULL;
    *length=0;

    return OK;
}


/*date_support.c-------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "utils.h"
#include "date_support.h"
#include "errors.h"

date_parser_t date_parsers[MAX_DATE_CONVERTERS]={
      parse_DD_MM_YYYY_HH_MM_SS_date
};


/*Esta función se encarga de obtener el tiempo en UNIX de una cadena de
caracteres*/

status_t get_time(char *str,time_t *unix_time, date_format_t format)
{
      status_t st;

      if(date_parsers[format]!=NULL){
            if((st=(*date_parsers[format])(str, unix_time))!=OK)
                  return st;
      }

      return OK;
}
```

```
/*Esta función se encarga de convertir el tiempo de una cadena de caracteres en
tiempo UNIX*/

status_t parse_DD_MM_YYYY_HH_MM_SS_date (char *str, time_t *unix_time){

	struct tm date, aux;
	status_t st;
	char parr[TOTAL_DATE_FIELDS][MAX_STR_LEN], *end_ptr;

	if((st=validate_format_DD_MM_YYYY_HH_MM_SS(str))!=OK)
		return st;

	memcpy(parr[0], str, 2*sizeof(char));
	parr[0][2]='\0';

	memcpy(parr[1], str+3, 2*sizeof(char));
	parr[1][2]='\0';

	memcpy(parr[2], str+6, 4*sizeof(char));
	parr[2][4]='\0';

	memcpy(parr[3], str+11, 2*sizeof(char));
	parr[3][2]='\0';

	memcpy(parr[4], str+14, 2*sizeof(char));
	parr[4][2]='\0';

	memcpy(parr[5], str+17, 2*sizeof(char));
	parr[5][2]='\0';

	date.tm_mday=strtol(parr[0], &end_ptr, 10);

	if(*end_ptr)
		return ERROR_INVALID_DATE;

	date.tm_mon=strtoul(parr[1], &end_ptr, 10);

	if(*end_ptr)
		return ERROR_INVALID_DATE;

	date.tm_year=strtoul(parr[2], &end_ptr, 10)-1900;

	if(*end_ptr)
		return ERROR_INVALID_DATE;

	date.tm_hour=strtoul(parr[3], &end_ptr, 10);

	if(*end_ptr)
		return ERROR_INVALID_DATE;
```

```
	date.tm_min=strtoul(parr[4], &end_ptr, 10);

	if(*end_ptr)
		return ERROR_INVALID_DATE;

	date.tm_sec=strtoul(parr[5], &end_ptr, 10);

	if(*end_ptr)
		return ERROR_INVALID_DATE;

	date.tm_isdst=0;

	aux=date;

	if((*unix_time=mktime(&date))==-1){
		return ERROR_INVALID_DATE;
	}

	if(date.tm_mday!=aux.tm_mday)
		return ERROR_INVALID_DATE;

	if(date.tm_mon!=aux.tm_mon)
		return ERROR_INVALID_DATE;

	if(date.tm_year!=aux.tm_year)
		return ERROR_INVALID_DATE;

	if(date.tm_hour!=aux.tm_hour)
		return ERROR_INVALID_DATE;

	if(date.tm_min!=aux.tm_min)
		return ERROR_INVALID_DATE;

	if(date.tm_sec!=aux.tm_sec)
		return ERROR_INVALID_DATE;

	if(date.tm_isdst!=aux.tm_isdst)
		return ERROR_INVALID_DATE;

	return OK;
}


/*La siguiente función valida el formato de la fecha, y devuelve
el respectivo delimitador*/

status_t validate_format_DD_MM_YYYY_HH_MM_SS(char *str){

	char delim;
	if(str[2]!=str[5]){
```

```
            return ERROR_INVALID_DATE;
      }else
            delim=str[2];

      if(delim!=DATE_DELIMITER_SLASH && delim!=DATE_DELIMITER_HYPHEN &&
delim!=DATE_DELIMITER_DOT)
            return ERROR_INVALID_DATE;

      if(str[10]!=SEPARATION_DELIMITER)
            return ERROR_INVALID_DATE;

      if(str[13]!=TIME_DELIMITER || str[16]!=TIME_DELIMITER)
            return ERROR_INVALID_DATE;

      return OK;

}


/*errors.c-----------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include "errors.h"

const char *errores [MAX_ERRORS]={
    MSG_OK,
    MSG_ERROR,
    MSG_ERROR_OPEN_INVENTORY_FILE,
    MSG_ERROR_OPEN_NEW_INVENTORY_FILE,
    MSG_ERROR_DISK_SPACE,
    MSG_ERROR_NULL_POINTER,
    MSG_ERROR_INVALID_ARG_NUMBER,
    MSG_ERROR_INVALID_ARG_FMT_FLAG,
    MSG_ERROR_INVALID_ARG_OUT_FILE_NAME_FLAG,
    MSG_ERROR_INVALID_ARG_INPUT_FILE_NAME_FLAG,
    MSG_ERROR_INVALID_ARG_INITIAL_TIME_FLAG,
    MSG_ERROR_INVALID_ARG_FINAL_TIME_FLAG,
    MSG_ERROR_INVALID_ARG_FORMAT,
    MSG_ERROR_CONVERSION_ARG_FINAL_TIME,
    MSG_ERROR_CONVERSION_ARG_INITIAL_TIME,
    MSG_ERROR_INVALID_ARG_TIMES,
    MSG_ERROR_MEMORY,
    MSG_ERROR_CONVERSION_ID,
    MSG_ERROR_CONVERSION_MONEY,
    MSG_ERROR_CONVERSION_TIME,
    MSG_ERROR_INVALID_DATE,
    MSG_ERROR_UNKNOWN_DOC_TYPE
};

/*La siguiente función lleva a cabo el informe de un estado de error
```

```
pasado por interfaz*/

status_t print_error(status_t st)
{
    fprintf(stderr, "%s\n", errores[st]);
    return OK;
}

/*setup.c-------------------------------------------------------------*/
#include <stdio.h>
#include "types.h"

setup_t *setup;
```

ARCHIVOS COMPLEMENTARIOS

```
/*header_CSV.txt----------------------------------------------*/
ID_USUARIO,INGRESOS,EGRESOS

/*footer_CSV.txt----------------------------------------------*/

/*header_XML.txt----------------------------------------------*/
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<usuarios>

/*footer_XML.txt----------------------------------------------*/
</usuarios>

/*header_HTML.txt----------------------------------------------*/
<!DOCTYPE html>
<html>
      <head>
            <title>An&aacutelisis forense de transacciones
bancarias</title>
      </head>
      <body>
            <h1>An&aacutelisis forense de transacciones bancarias </h1>
            <table border=1 width="30%">
                  <thead>
                        <tr>
                              <th>Nro. de usuario</th>
                              <th>Ingresos</th>
                              <th>Egresos</th>
                        </tr>
                  </thead>


/*footer_HTML.txt----------------------------------------------*/
            </table>
      </body>
</html>
```