
Alg. y Prog. I (95.11)

Curso 1 - Ing. Cardozo

Trabajo práctico integrador: Análisis forense de extracto bancario

Alumnos:

Díaz Malca, David Vladimir (103152)
Peiretti, Pablo (103592)

Correo electrónicos:

dvladimirdm@hotmail.com
pablopei89@gmail.com

Fecha de entrega: 07/12/19

Universidad de Buenos Aires
Facultad de Ingeniería
Algoritmos y Programación I (95.11)

# TRABAJO PRÁCTICO INTEGRADOR – Análisis forense de extracto bancario

## 1) Objetivo

El objetivo de este T.P. consiste en desarrollar un aplicativo de consola con comandos en línea de órdenes y escrito en lenguaje ANSI C, que permita realizar un análisis de transacciones de un extracto bancario.

## 2) Alcance

Mediante el presente T.P. se busca que el alumno adquiera y aplique conocimientos sobre los siguientes temas:

- programas en modo consola;
- argumentos en línea de órdenes (*CLA*);
- modularización;
- *makefile;*
- archivos de texto y binarios;
- memoria dinámica;
- punteros a función;
- tipo de dato abstracto (T.D.A. Vector, T.D.A. *ad hoc*);
- estructura básica de un archivo CSV;
- manejo de fechas;

## 3) Desarrollo

En este T.P. se pide escribir un programa ejecutable que procese a partir de un extracto bancario las transacciones de ingreso y egreso de dinero de los distintos usuarios en un intervalo de tiempo determinado.

El programa ejecutable, denominado "`analisis_bancario.exe`" (WinXX) o "`analisis_bancario`" (Unix), debe ser invocado de la siguiente forma:

WinXX:
```
analisis_bancario –fmt <formato> -out <salida> -in <entrada> -ti
<tiempo_inicial> -tf <tiempo_final>
```

UNIX:
```
./analisis_bancario –fmt <formato> -out <salida> -in <entrada> -ti
<tiempo_inicial> -tf <tiempo_final>
```

Los comandos en línea de órdenes utilizados por la aplicación son los indicados a continuación.

Formato del índice a generar

| Comando | Descripción | Valor | Tipo de dato |
|---|---|---|---|
| -fmt | Formato del índice a generar | "csv" | Cadena de caracteres |
| | | "xml" | Cadena de caracteres |
| | | "html" | Cadena de caracteres* |

* Optativo

Archivo de salida <salida>

Es la ruta del archivo a crear con la información analizada.

Archivos de entrada <entrada>

Es la ruta del archivo de texto que contiene la información a analizar.

Tiempo inicial <tiempo_inicial>

Tiempo inicial, a partir del cual se debe realizar el análisis en segundos como tiempo Unix.

Tiempo inicial <tiempo_final>

Tiempo final, hasta el cual se debe realizar el análisis en segundos como tiempo Unix.

Nota: Se puede asumir que los comandos en línea de órdenes estarán en cualquier orden (en pares), si esta estrategia simplifica el desarrollo de la presente aplicación, pero se debe consignar la decisión en el informe.
El intervalo de tiempo dado entre el tiempo inicial y el tiempo final es un intervalo cerrado.

Formato de entrada

El archivo del extracto bancario a procesar tiene un formato CSV con las siguientes columnas:

*ID_TRANSACCION | ID_USUARIO | FECHA | MONTO | DESCRIPCION*
*123412 | 1 | 05/11/2011 10:00:00 | -10 | Compra supermercado*
*123413 | 2 | 05/11/2011 10:00:01 | -100 | Compra supermercado*
*123414 | 1 | 05/11/2011 10:00:02 | 200 | Transferencia*
*123415 | 3 | 05/11/2011 10:00:03 | -100 | Extraccion cajero*

| Nombre | Ejemplo | Descripción |
|---|---|---|
| Id Transacción | 12345 | Identificación de la transacción [size_t] |
| Id Usuario | 2153 | Identificación del usuario [size_t] |
| Fecha | *05/11/2011 10:00:00* | Fecha en la que ocurre la transacción en formato dd/mm/yyyy hh:mm:ss |
| | *05-11-2011 10:00:00* | Fecha en la que ocurre la transacción en formato dd-mm-yyyy hh:mm:ss |
| | *05.11.2011 10:00:00* | Fecha en la que ocurre la transacción en formato dd.mm.yyyy hh:mm:ss |
| Monto | 100 | Monto de la transacción [int] |
| Descripción | Transferencia | Descripción de la operación [string] |

Notas

1) El identificador del usuario tiene un valor máximo desconocido pero se sabe que es suficientemente chico como para que el programa pueda ser procesado por un ordenador hogareño.
2) El manejo del arreglo donde se compacten los datos deberá utilizar memoria dinámica.
3) Es importante observar que el formato de entrada puede tener las fechas expresadas en distintos formatos en el mismo archivo y, por lo tanto, se deberá poder manejar los distintos tipos de fechas.
4) El carácter delimitador del archivo es un '|'. En la descripción no habrá este tipo de caracteres.

Operación

El programa debe analizar el archivo de entrada y descartar aquellas líneas que se encuentran fuera del intervalo de tiempo dado. El objetivo es poder acumular los ingresos y egresos de cada usuario. Los ingresos estarán dados por los montos positivos y los egresos, por los montos negativos.

Formatos de salida

La salida del programa debe generar un archivo con los montos transaccionados por cada usuario, separando ingresos de egresos, ordenados de mayor a menor por la cantidad de egresos.

a) Formato CSV

El formato del documento CSV de salida a generar es:

```
<Id Usuario>,<Ingresos>,<Egresos>
...
<Id Usuario>,<Ingresos>,<Egresos>
```

*Ejemplo de archivo de salida*

*ID_USUARIO, INGRESOS, EGRESOS*
*3 , 0, 100*
*2, 0, 100*
*1, 200, 10*

b) Formato XML

Ejemplo equivalente de salida en XML

```
<?xml version="1.0" encoding="UTF-8"?>
<usuarios>
  <usuario>
    <id>3</id>
    <ingresos>0</ingresos>
    <egresos>100</egresos>
  </usuario>
  <usuario>
    <id>2</id>
    <ingresos>0</ingresos>
```

```
    <egresos>100</egresos>
  </usuario>
  <usuario>
    <id>3</id>
    <ingresos>200</ingresos>
    <egresos>10</egresos>
  </usuario>
</usuarios>
```

c) Formato HTML (optativo)

Formato libre de página web, a elección del alumno.

## 4) Restricciones

La realización de este programa está sujeta a las siguientes restricciones:

− Se debe recurrir al uso de punteros a función a fin de parametrizar la impresión de los archivos de salida.
− Se deben utilizar funciones y una adecuada modularización.
− Se debe construir un proyecto mediante la utilización de *makefile*.
− Se deben documentar las funciones implementadas.
− Hay otras cuestiones que no han sido especificadas intencionalmente en este requerimiento, para darle al desarrollador la libertad de elegir implementaciones que, según su criterio, resulten más favorables en determinadas situaciones. Por lo tanto, se debe explicitar cada una de las decisiones adoptadas y el o los fundamentos considerados para ellas. Sin embargo, las alternativas adoptadas por el desarrollador no podrán contradecir la presente especificación.

## 5) Entrega del Trabajo Práctico

Se debe entregar personalmente el trabajo práctico. Esta documentación de desarrollo se debe presentar en forma impresa y encarpetada, de acuerdo con la numeración siguiente. Los alumnos de cada grupo deben estar presentes para recibir las correcciones y realizar una defensa de su trabajo.

1) Carátula. Incluir una dirección de correo electrónico.
2) Enunciado (el presente documento).
3) Diagrama de flujo básico y simplificado del programa (1 carilla A4).
4) Estructura funcional del programa desarrollado (diagrama con la jerarquía de funciones, 1 carilla A4, omitir las funciones nativas).
5) Explicación de las alternativas consideradas y las estrategias adoptadas (ambas cuestiones).
6) Resultados de la ejecución (corridas) del programa, en condiciones normales e inesperadas de entrada.
7) Reseña de los problemas encontrados en el desarrollo del programa y las soluciones implementadas para subsanarlos.
8) Conclusiones.
9) *Script* de compilación.
10) Código fuente.

Una vez aprobado el trabajo, cada alumno deberá tener el original, una copia o una

reimpresión del trabajo firmada por alguno de los docentes del curso, y llevarlas el día de la evaluación integradora.

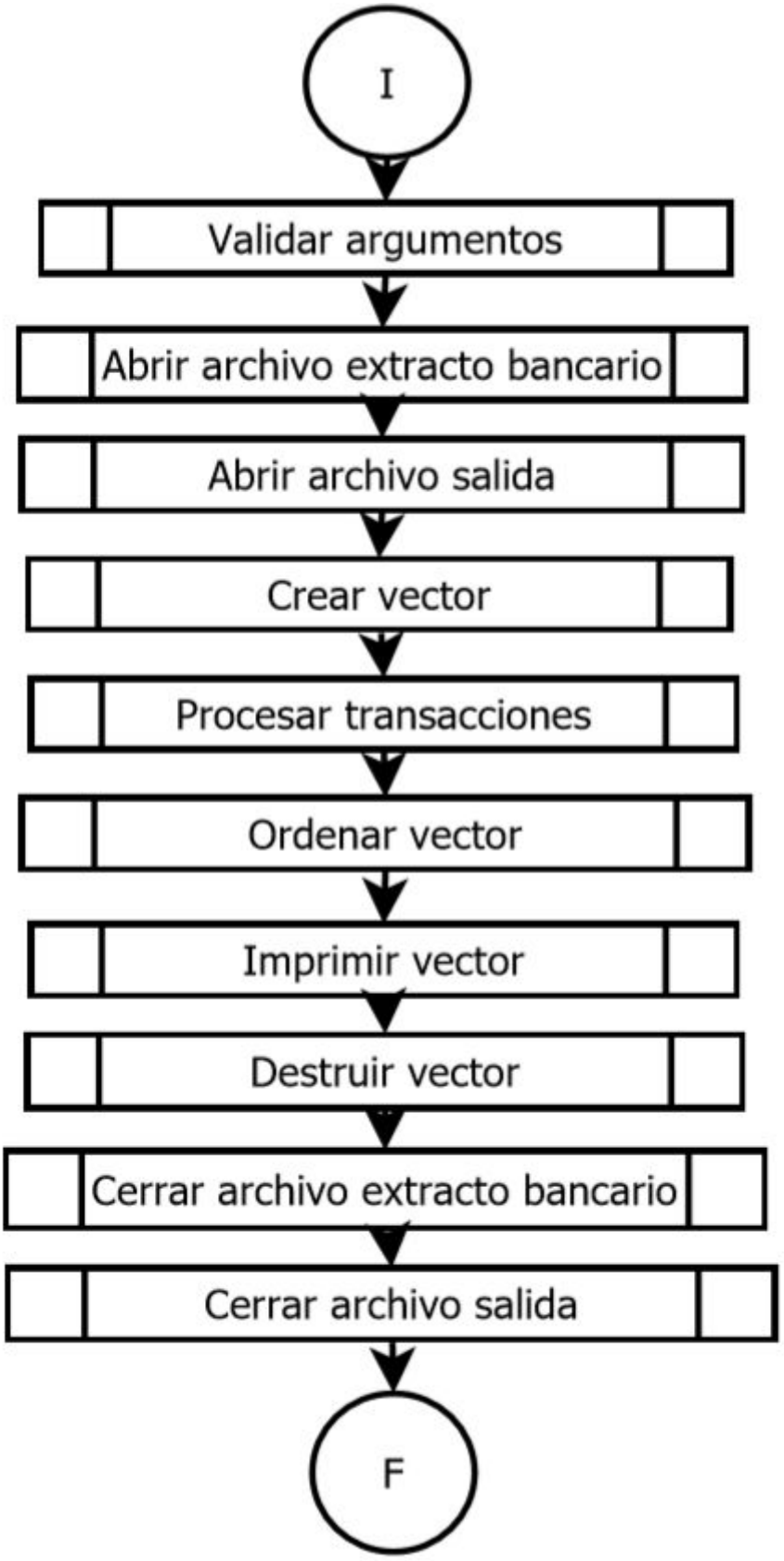
## 6) **Bibliografía**

Se debe incluir la referencia a toda bibliografía consultada para la realización del trabajo: libros, artículos, etc.

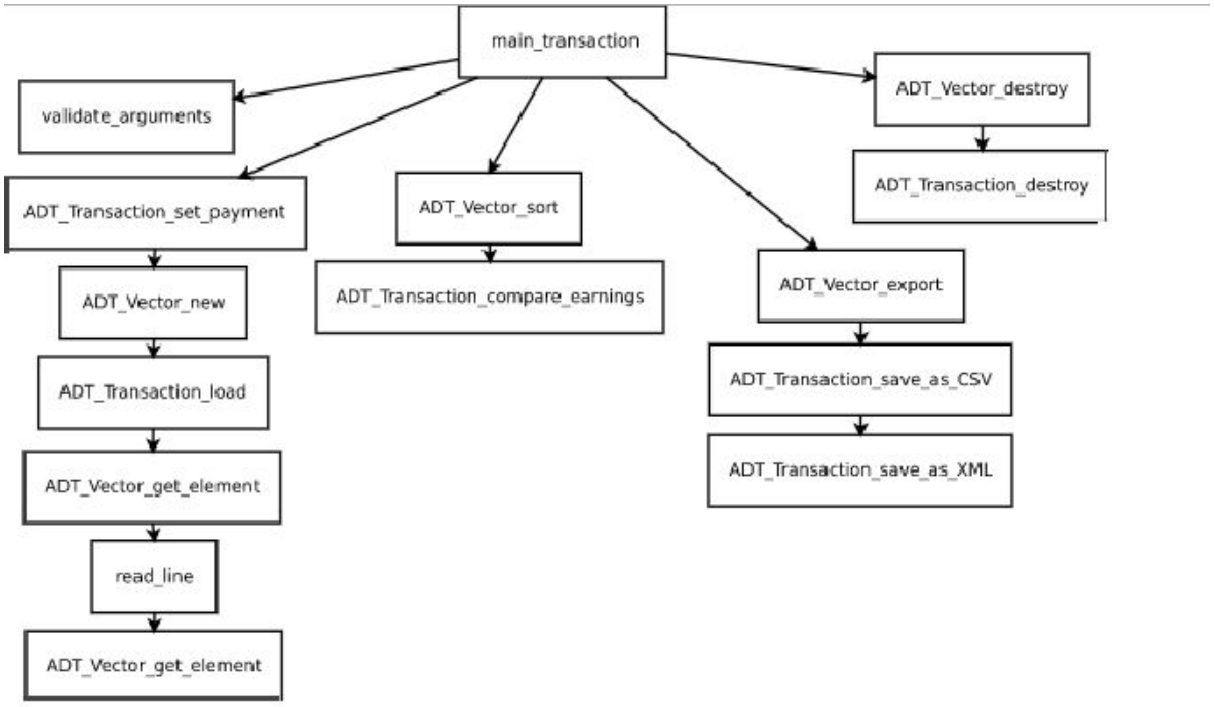
## 7) **Fecha de entrega**

La fecha límite de presentación del trabajo es el 05/12/2019 a las 19:00 h.

## 3) Diagrama de flujo



# 4) Estructura funcional




# 5) Explicación de las alternativas consideradas y las estrategias adoptadas

## Introducción

El objetivo planteado en este trabajo consiste en desarrollar un aplicativo de consola con comandos en línea de órdenes y escrito en lenguaje ANSI C, que permitan realizar un análisis de transacciones de un extracto bancario.

El trabajo comienza con los supuestos tenidos en cuenta, a continuación, desarrolla brevemente lo realizado, luego se encuentran las conclusiones y, al final del informe, la bibliografía.

## Desarrollo

En el presente caso de estudio se supuso que los argumentos en línea de orden se ingresaran ordenados debido a que facilita su procesamiento y mejora su eficiencia. Se

codificó una versión con los argumentos en desorden y se decidió no incluirla porque era una versión poco eficiente.

Se realizaron modularizaciones para facilitar la lectura. Se trata de una práctica sana de programación que consiste en separar las funciones y sus correspondientes encabezados en archivos separados de la función `main()`, así se logra el resultado esperado, ayudar al mantenimiento del software y la optimización en tiempo de compilación.

El caso de estudio fue realizado tras haber empleado un archivo de texto del tipo CSV (por sus siglas en inglés), es decir, archivos que están separados por un delimitador y su longitud de caracteres es variable. Existen otros tipos de archivos que fueron tenidos en cuenta a la hora de realizar el trabajo, por ejemplo los archivos de tipo binario o de ancho fijo. Los primeros contienen información de cualquier tipo codificada en binario, en cambio, los de ancho fijo son de tipo texto ,tienen longitud de caracteres fija y no están separados por ningún carácter delimitado.

En el trabajo de desarrollo un programa capaz de analizar las transacciones de un extracto bancario. Para lo cual, se diseñaron e implementaron los tipos de datos abstractos “ADT_Transaction_ t” y “ADT_Array”. El primero permite el manejo de las transacciones, por ejemplo, “ADT_Transaction_new” o “ADT_Transaction_get_id”. En cambio, el segundo, maneja un arreglo genérico, por ejemplo, “ADT_Array_new_element” o “ADT_Array_set_element”. Se incluyeron campos y primitivas en el tipo de dato abstracto que no son necesarios en este proyecto en particular, con el objetivo de mejorar la flexibilidad y reutilización de los mismos.

Además se desarrollaron otras funciones que ayudan a modularizar los programas, a la comprensividad del mismo y permiten su reutilización. Algunas de las mencionados son las funciones “read_line” , “split” y “clone_string” que al ser utilizadas en reiteradas ocasiones dentro del programa, se ahorra la necesidad de reescribir código.

El manejo de fechas se llevó a cabo en los archivos “date.c” y “date.h”, con el objetivo de que fuese flexible a otros tipos de formatos de fechas que se quisieran agregar.

Se utilizó un método de selección a la hora de ordenar el “ADT_Array” y se implementó con la función “ADT_Array_sort”. Se decidió emplear este método, ya que más allá de ser de orden cuadrático, las comparaciones no requieren mucho esfuerzo y tiene la ventaja de dejar los elementos en su orden definitivo.

Las validaciones necesarias usadas para que el programa sea robusto cumplen lo expresado por Ghezzi, Jazayeri y Mandrioli (1991: 21):

> *«A program is robust if it behaves “reasonably,” even in circumstances that were not anticipated in the requirements specification- for example, when it encounters incorrect input data or somo hardware malfunction. (1)...».*
>
> *«Un programa es robusto si se comporta de manera “razonable”, incluso en circunstancias que no han sido anticipadas en las especificaciones de los requisitos- por ejemplo, cuando encuentra datos de entrada incorrectos o algún mal funcionamiento del hardware.(...) (1)».*
> [Traducción propia]

## Resultados de ejecución

### Archivo de entrada

input.txt

```
1123|2|04/10/2019 10:00:00|-20|compra
4132|2|01/09/2019 10:00:00|-10|venta
1123|1|10/03/2019 10:00:00|3330|transferencia
1324|2|01/10/2019 10:00:00|130|pago
5341|5|01/10/2019 10:00:00|-120|compra
1235|6|01/10/2019 10:00:00|200|venta
5311|3|01/10/2000 10:00:00|-25|venta
1151|17|01/10/2009 10:00:00|-15|compra
1412|3|05/10/2019 10:00:00|13|transferencia
2161|15|03/10/2019 10:00:00|20|pago
14|9|01/10/1980 10:00:00|320|compra
41|10|01/10/2000 10:00:00|330|compra
51|11|01/12/1970 10:00:00|-130|compra
121|3|08/10/1970 10:00:00|3|devolucion
511|3|01/10/2019 10:00:00|30|compra
31|100|01/10/2019 10:00:00|10|compra
```

### Exportación como CSV

```
pei@pei-VirtualBox:~/algoritmos/tpentrega$ make all
make: No se hace nada para 'all'.
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt CSV -in input.txt -out output.txt -ti 1 -tf 10000000000000
pei@pei-VirtualBox:~/algoritmos/tpentrega$ cat output.txt
ID_USUARIO|INGRESOS|EGRESOS
1|3330|0
10|330|0
9|320|0
6|200|0
2|130|30
3|46|25
15|20|0
100|10|0
5|0|120
17|0|15
11|0|130
pei@pei-VirtualBox:~/algoritmos/tpentrega$
```

## Exportación como XML

```
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XML -in input.txt -out output.txt -ti 1 -tf 100000000000
pei@pei-VirtualBox:~/algoritmos/tpentrega$ cat output.txt
<?xml version="1.0" encoding="UTF-8"?">
<usuarios>
        <usuario>
                <id>1</id>
                <ingresos>3330</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>10</id>
                <ingresos>330</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>9</id>
                <ingresos>320</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>6</id>
                <ingresos>200</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>2</id>
                <ingresos>130</ingresos>
                <egresos>-30</egresos>
        </usuario>
        <usuario>
                <id>3</id>
                <ingresos>46</ingresos>
                <egresos>-25</egresos>
        </usuario>
        <usuario>
                <id>15</id>
                <ingresos>20</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>100</id>
                <ingresos>10</ingresos>
                <egresos>0</egresos>
        </usuario>
        <usuario>
                <id>5</id>
                <ingresos>0</ingresos>
                <egresos>-120</egresos>
        </usuario>
        <usuario>
                <id>17</id>
                <ingresos>0</ingresos>
                <egresos>-15</egresos>
        </usuario>
        <usuario>
                <id>11</id>
                <ingresos>0</ingresos>
                <egresos>-130</egresos>
        </usuario>
</usuarios>
```

## Fallas en el ingreso de argumentos

```
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XM -in input.txt -out output.txt -ti 1 -tf 10000000000000
El formato de salida es incorrecto
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XM -in input.txt -out output.txt -ti 1 -tf
Error en la invocacion, pocos arguments
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XM -in input.txt -out output.txt -ti 100 -tf 13
El tiempo inicial es mayor que el tiempo final
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XM -in input.txt -ou output.txt -ti 100 -tf 13
La funcion ha sido mal invocada
pei@pei-VirtualBox:~/algoritmos/tpentrega$
```

## Fallos en el archivo de entrada

input.txt

```
1123|2|04/10/2019 10:00:00|-20|compra
4132|2|09/2019 10:00:00|-10|venta
1123|1|10/03/2019 10:00:00|3330|transferencia
```

```
pei@pei-VirtualBox:~/algoritmos/tpentrega$ make all
make: No se hace nada para 'all'.
pei@pei-VirtualBox:~/algoritmos/tpentrega$ ./analisis_bancario -fmt XML -in input.txt -out output.txt -ti 90 -tf 1000000000000000
La fecha ingresada es muy corta
pei@pei-VirtualBox:~/algoritmos/tpentrega$
```

# Conclusiones

Un análisis posterior del caso de estudio deja como resultado una comparativa entre las ventajas de utilizar un tipo de archivo CSV frente a los archivos de texto de ancho fijo o binarios.

Los archivos de texto binarios son los más eficientes y los más sencillos de procesar. Sin embargo, no se utilizaron debido a su poca portabilidad, por ejemplo, si se encuentra representado en *little endian* y otra máquina lo lee en *big endian*, ésta lo leerá al revés.

Los archivos de ancho fijo permiten trabajar sin hacer uso de la asignación dinámica de la memoria. Por este motivo, su velocidad de procesamiento es muy alta.

Por último, los archivos CSV son propensos a errores y a la hora de leerlos se requiere manejo de memoria dinámica, por lo cual su velocidad de procesamiento es menor que la de los archivos de ancho fijo. Sin embargo, permiten un manejo mucho más eficiente de la memoria, ya que si los campos son de menor tamaño que el definido, no es necesario rellenarlos.

El caso de estudio fue realizado con un archivo de tipo CSV. Se decidió priorizar la eficiencia en el manejo de la memoria debido a que la longitud de los campos es desconocida y, al ser un un extracto bancario la cantidad de operaciones será muy grande. Además este archivo fue elegido considerando su portabilidad.

Entendiendo por portabilidad, lo expresado por Ghezzi, Jazayeri y Mandrioli (1991: 24):

> *«A software is portable if it can run on different enviroments.(...)(2)»*
>
> *«Un software es portable si puede ser ejecutado en distintos ambientes..(...)(2)».* [Traducción propia]

Tras el análisis de las características enunciadas resulta importante tomar en cuenta que la elección del tipo de archivo a usar en el momento de planeamiento y desarrollo recae en el contexto del proyecto que se quiere llevar a cabo. Sacrificar una cualidad para ganar otra según la posición en la que se encuentre y sus posteriores aplicaciones.

Otro aspecto fundamental, es la utilización de tipos de datos abstractos. Implementar este tipo de datos da la posibilidad de operar con una interfaz mínima, es decir, permite la separación de las partes que componen el programa. La ventaja principal de esta forma de operar es que el desarrollo se puede llevar a cabo en varios niveles, por lo cual se vuelve más sencillo y mantenible. También se puede ocultar la información que no resulte necesaria, por que se guarda la forma en la que está desarrollada y se vuelve más seguro.

Otra ventaja es la posibilidad de reutilización de los mismos, al hacerlo genéricos se pueden aplicar en otros proyectos.

# Bibliografía

1. 2. GHEZZI, C., JAZAYERI, M. y MANDRIOLI, D. *FUNDAMENTALS OF Software Engineering*. Prentice-hall, Inc. 1991. ISBN-0-13-820432-2.

# Código fuente

## main_transaction.h

```
#ifndef MAIN_TRANSACTION__H
#define MAIN_TRANSACTION__H

#include <stdio.h>

#include "process_transaction.h"
#include "config.h"
#include "error.h"
#include "types.h"

#define NUM_CMD_ARG 11
#define NUM_FLAGS 5
#define CMD_FORMAT "-fmt"
#define CMD_OUT "-out"
#define CMD_IN "-in"
#define CMD_FINAL_TIME "-tf"
#define CMD_INITIAL_TIME "-ti"
#define POS_CMD_FORMAT 1
#define POS_CMD_IN 3
#define POS_CMD_OUT 5
#define POS_CMD_INITIAL_TIME 7
#define POS_CMD_FINAL_TIME 9
#define POS_INPUT_FILE 4
#define POS_OUTPUT_FILE 6
#define FMT_NAME_CSV "CSV"
#define FMT_NAME_XML "XML"

status_t validate_arguments (int argc,char * argv[],config_t *
config);



#endif
```

## main_transaction.c

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

```
#include "main_transaction.h"
#include "process_transaction.h"
#include "config.h"
#include "error.h"
#include "types.h"

int main (int argc,char * argv[])
{
      status_t st;
      config_t config;
      FILE * transaction_file;
      FILE * output_file;


      if((st=validate_arguments(argc,argv,&config))!=OK)
      {
            print_error(st);
            return st;
      }

      if((transaction_file=fopen(config.input_file,"rt"))==NULL)
      {
            print_error(ERROR_OPEN_TRANSACTION_FILE);
            return ERROR_OPEN_TRANSACTION_FILE;
      }

      if((output_file=fopen(config.output_file,"wt"))==NULL)
      {
            fclose(transaction_file);
            print_error(ERROR_OPEN_OUTPUT_FILE);
            return ERROR_OPEN_OUTPUT_FILE;
      }


if((st=process_transaction(transaction_file,output_file,&config))!
=OK)
      {
            fclose(transaction_file);
            fclose(output_file);
            print_error(st);
            return st;
      }
```

```
	fclose(transaction_file);

	if(fclose(output_file)==EOF)
	{
		remove(output_file);
		print_error(ERROR_CLOSE_OUTPUT_FILE);
		return ERROR_CLOSE_OUTPUT_FILE;
	}

	return OK;

}


status_t validate_arguments (int argc,char * argv[],config_t *
config)
{
	char *tmp;

	if(argv==NULL)
		return ERROR_NULL_POINTER;

	if (argc<NUM_CMD_ARG)
		return ERROR_INVOCATION_FEW_ARGUMENTS;

	if(argc>NUM_CMD_ARG)
		return ERROR_INVOCATION_MANY_ARGUMENTS;


	/*VALIDO Y RECUPERO FORMATO*/
	if (strcmp(argv[POS_CMD_FORMAT],CMD_FORMAT))
		return ERROR_INVOCATION_FUNCTION;

	if(!strcmp(argv[POS_CMD_FORMAT+1],FMT_NAME_CSV)){
		config->format=FMT_CSV;
	}
	else if(!strcmp(argv[POS_CMD_FORMAT+1],FMT_NAME_XML))
		config->format=FMT_XML;

	if (strcmp(argv[POS_CMD_OUT],CMD_OUT))
		return ERROR_INVOCATION_FUNCTION;
	config->output_file=argv[POS_CMD_OUT+1];

	if (strcmp(argv[POS_CMD_IN],CMD_IN))
```

```
            return ERROR_INVOCATION_FUNCTION;
      config->input_file=argv[POS_CMD_IN+1];

      /*VALIDO Y RECUPERO TI*/
      if (strcmp(argv[POS_CMD_INITIAL_TIME],CMD_INITIAL_TIME))
            return ERROR_INVOCATION_FUNCTION;
      config->ti=strtoul(argv[POS_CMD_INITIAL_TIME+1],&tmp,10);
      if(*tmp)
            return ERROR_INITIAL_TIME_DATA;

      /*VALIDO Y RECUPERO TF*/
      if (strcmp(argv[POS_CMD_FINAL_TIME],CMD_FINAL_TIME))
            return ERROR_INVOCATION_FUNCTION;
      config->tf=strtoul(argv[POS_CMD_FINAL_TIME+1],&tmp,10);
      if(*tmp)
            return ERROR_FINAL_TIME_DATA;

      if(config->ti>config->tf)
            return ERROR_INVALID_TIME;



      return OK;


}
```

## ADT_Transaction.h

```
#ifndef ADT_TRANSACTION__H
#define ADT_TRANSACTION__H

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

#define NUM_DEL 2

#define FIELD_ID_TRANSACTION 0
#define FIELD_ID_USER 1
#define FIELD_DATE 2
#define FIELD_AMOUNT 3
#define FIELD_DESCRIPTION 4
```

```
#define USER_OPPENING_TAG "<usuario>"
#define USER_CLOSING_TAG "</usuario>"
#define ID_OPPENING_TAG "<id>"
#define ID_CLOSING_TAG "</id>"
#define EARNINGS_OPPENING_TAG "<ingresos>"
#define EARNINGS_CLOSING_TAG "</ingresos>"
#define EXPENSES_OPPENING_TAG "<egresos>"
#define EXPENSES_CLOSING_TAG "</egresos>"

#define OUTPUT_CSV_DELIMITOR '|'


typedef struct {
     size_t id_transaction;
     size_t id_user;
     int losses;
     int earnings;
     time_t seconds;
     char * description;
}ADT_Transaction_t;

status_t ADT_Transaction_new (ADT_Transaction_t **transaction);
status_t ADT_Transaction_destroy(ADT_Transaction_t **transaction);
/*GETTERS*/
status_t ADT_Transaction_get_id_user (ADT_Transaction_t
*transaction, size_t *id_user);
status_t ADT_Transaction_get_id_transaction (ADT_Transaction_t
*transaction, size_t *id_transaction);
status_t ADT_Transaction_get_description (ADT_Transaction_t
*transaccion, char** description);
status_t ADT_Transaction_get_seconds (ADT_Transaction_t
*transaction, time_t * date);
status_t ADT_Transaction_get_amount (ADT_Transaction_t
*transaction,int * amount);

/*SETTERS*/
status_t ADT_Transaction_set_id_transaction(ADT_Transaction_t
*transaction,size_t id);
status_t ADT_Transaction_set_id_user(ADT_Transaction_t
*transaction,size_t id);
status_t ADT_Transaction_set_description(ADT_Transaction_t
*transaction,char * description);
status_t ADT_Transaction_set_seconds(ADT_Transaction_t
*transaction,char seconds);
```

```
status_t ADT_Transaction_set_amount(int amount ,ADT_Transaction_t
*transaction);
status_t ADT_Transaction_add_amount( ADT_Transaction_t
*transaction,int amount);

status_t ADT_Transaction_load(FILE *fo,ADT_Transaction_t **
transaction,bool_t *eof,char  del);
/********* SAVE AS... *********/
status_t ADT_Transaction_save_as_XML(const ADT_Transaction_t
*bank_balance, FILE *fo);
status_t ADT_Transaction_save_as_CSV(const ADT_Transaction_t
*bank_balance, FILE *fo);
/*COMPARADORES*/
int ADT_Transaction_compare_earnings(const ADT_Transaction_t
*transaction1, const ADT_Transaction_t * transaction2);



#endif
```

## ADT_Transaction.c

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

#include "ADT_Transaction.h"
#include "utils.h"
#include "types.h"
#include "date.h"


/************************CREADORES*********************************
******/

status_t ADT_Transaction_new (ADT_Transaction_t **transaction)
{
     if (transaction==NULL)
          return ERROR_NULL_POINTER;

     if((*transaction=(ADT_Transaction_t
*)malloc(sizeof(ADT_Transaction_t)))==NULL)
          return ERROR_MEMORY;
```

```
	(*transaction)->id_transaction=0;
	(*transaction)->id_user=0;
	(*transaction)->earnings=0;
	(*transaction)->losses=0;
	(*transaction)->seconds=0;
	(*transaction)->description=NULL;

	return OK;
}

status_t ADT_Transaction_destroy(ADT_Transaction_t **transaction)
{
	if(transaction==NULL)
		return ERROR_NULL_POINTER;

	free((*transaction)->description);
	(*transaction)->description=NULL;

	free(*transaction);
	*transaction=NULL;

	return OK;
}



/*****************************************GETTERS*****************
******************/
status_t ADT_Transaction_get_id_user (ADT_Transaction_t
*transaction, size_t *id_user)
{
	if(transaction==NULL || id_user ==NULL)
		return ERROR_NULL_POINTER;

	*id_user=transaction->id_user;

	return OK;
}

status_t ADT_Transaction_get_id_transaction (ADT_Transaction_t
*transaction, size_t *id_transaction)
{
	if(transaction==NULL || id_transaction==NULL)
		return ERROR_NULL_POINTER;
```

```
	*id_transaction=transaction->id_transaction;

	return OK;
}

status_t ADT_Transaction_get_seconds (ADT_Transaction_t
*transaction, time_t * seconds)
{
	if(transaction==NULL)
		return ERROR_NULL_POINTER;
	*seconds=transaction->seconds;

	return OK;
}

status_t ADT_Transaction_get_description (ADT_Transaction_t
*transaction, char ** description)
{

	if(transaction==NULL || description ==NULL)
		return ERROR_NULL_POINTER;

	*description=transaction->description;

	return OK;
}

status_t ADT_Transaction_get_amount (ADT_Transaction_t
*transaction,int * amount)
{
	if(transaction==NULL)
		return ERROR_NULL_POINTER;

	*amount =transaction->earnings -transaction->losses;

	return OK;
}


/**********************************SETTERS************************
******************/
status_t ADT_Transaction_set_id_user(ADT_Transaction_t
*transaction,size_t id)
{
```

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

	transaction->id_user=id;

	return OK;
}

status_t ADT_Transaction_set_id_transaction(ADT_Transaction_t
*transaction,size_t id)
{
	if(transaction==NULL)
		return ERROR_NULL_POINTER;

	transaction->id_transaction=id;

	return OK;
}


status_t ADT_Transaction_set_seconds(ADT_Transaction_t
*transaction,char seconds)
{
	if(transaction==NULL)
		return ERROR_NULL_POINTER;

	transaction->seconds=seconds;

	return OK;
}


status_t ADT_Transaction_set_description(ADT_Transaction_t
*transaction,char * description)
{

	if(transaction==NULL || description==NULL)
		return ERROR_NULL_POINTER;

	transaction->description=description;

	return OK;
}
```

```
status_t ADT_Transaction_set_amount(int amount, ADT_Transaction_t
*transaction)
{

	if(transaction==NULL)
		return ERROR_NULL_POINTER;

	if(amount>0)
	{
		transaction->earnings=amount;
	}else {
		transaction->losses=(-amount);
	}

	return OK;

}

status_t ADT_Transaction_add_amount( ADT_Transaction_t
*transaction,int amount)
{
	if(transaction== NULL)
		return ERROR_NULL_POINTER;

	if(amount>0)
	{
		transaction->earnings+=amount;
	}else {
		transaction->losses-=amount;
	}

	return OK;

}
status_t ADT_Transaction_load(FILE *fo,ADT_Transaction_t **
transaction,bool_t *eof,char del)
{
	char *tmp;
	char dels[NUM_DEL]={del,'\0'};
	size_t n_fields;
	char ** fields;
	char * str;
	date_t fmt_date;
	time_t seconds;
	status_t st;
```

```
	int amount;


	if(fo==NULL||transaction==NULL||eof==NULL){
		return ERROR_NULL_POINTER;
	}

	if((st=read_line(fo,&str,eof))!=OK)
		return st;

	if(*eof==TRUE)
	{
		free(str);
		return OK;
	}

	if((st=split(str,dels,&fields,&n_fields))!=OK)
	{
		free(str);
		return st;
	}


	/* creo el ADT*/

	if((st=ADT_Transaction_new(transaction))!=OK)
	{
		free(str);
		destroy_strings(&fields,n_fields);
		return st;
	}

	/*cargo id transaccion*/

(*transaction)->id_transaction=strtoul(fields[FIELD_ID_TRANSACTION
],&tmp,10);
	if(*tmp)
	{
		free(str);
		destroy_strings(&fields,n_fields);
		return ERROR_INFORMATION_ID_TRANSACTION_INVALID;
	}

	/*cargo id usuario*/
```

```
(*transaction)->id_user=strtoul(fields[FIELD_ID_USER],&tmp,10);
	if(*tmp)
	{
		free(str);
		destroy_strings(&fields,n_fields);
		return ERROR_INFORMATION_ID_USER_INVALID;
	}

	/*cargo segundos*/
	if((st=parse_date(fields[FIELD_DATE],&fmt_date))!=OK)
		return st;
	switch(fmt_date)
	{
		case(DD_MM_YYYY):

if((convert_to_seconds_DD_MM_YYYY(fields[FIELD_DATE],&seconds))!=O
K)
			{
				free(str);
				destroy_strings(&fields,n_fields);
				return st;
			}
			break;
		default:/*no deberia llegar hasta aca pero no esta mal
agregarlo*/
			free(str);
			destroy_strings(&fields,n_fields);
			return ERROR_INFORMATION_DATE;
	}

	(*transaction)->seconds=seconds;

	/*cargo egreso e ingreso*/

	amount=(int)strtol(fields[FIELD_AMOUNT],&tmp,10);
	if(*tmp)
	{
		free(str);
		destroy_strings(&fields, n_fields);
		return ERROR_INFORMATION_AMOUNT_INVALID;
	}
	if((st=ADT_Transaction_set_amount(amount,*transaction))!=OK)
	{
		free(str);
```

```
            destroy_strings(&fields, n_fields);
            return st;
      }


      (*transaction)->description=fields[FIELD_DESCRIPTION];



      free(str);
      destroy_strings(&fields, n_fields);


      return OK;
}

status_t ADT_Transaction_save_as_XML(const ADT_Transaction_t
*bank_balance, FILE *fo)
{
      if( fo==NULL)
            return ERROR_NULL_POINTER;
      if(bank_balance==NULL)
            return OK;

      fprintf(fo,
"\t%s\n\t\t%s%lu%s\n\t\t%s%d%s\n\t\t%s%d%s\n\t%s\n"
            , USER_OPPENING_TAG
            , ID_OPPENING_TAG, bank_balance->id_user, ID_CLOSING_TAG
            , EARNINGS_OPPENING_TAG, bank_balance->earnings,
EARNINGS_CLOSING_TAG
            , EXPENSES_OPPENING_TAG, -(bank_balance->losses),
EXPENSES_CLOSING_TAG
            , USER_CLOSING_TAG);

      return OK;
}
status_t ADT_Transaction_save_as_CSV(const ADT_Transaction_t
*bank_balance, FILE *fo)
{
      if( fo==NULL)
            return ERROR_NULL_POINTER;
      if(bank_balance==NULL)
            return OK;
```

```
	fprintf(fo, "%lu%c%d%c%d\n" , bank_balance->id_user,
OUTPUT_CSV_DELIMITOR
                                                ,
bank_balance->earnings, OUTPUT_CSV_DELIMITOR
                                                , bank_balance->losses);

	return OK;
}
int ADT_Transaction_compare_earnings(const ADT_Transaction_t *
transaction1, const ADT_Transaction_t *    transaction2)
{
	return (transaction1->earnings - transaction2->earnings);
}
```

## ADT_Vector.h

```
#ifndef ADT_Vector__H
#define ADT_Vector__H

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


#define INIT_SIZE 25

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


status_t ADT_Vector_new(ADT_Vector_t ** arr);
status_t ADT_Vector_destroy(ADT_Vector_t **vector);
status_t ADT_Vector_export(const ADT_Vector_t *vector, FILE *fo);
status_t ADT_Vector_sort(ADT_Vector_t *vector);

/**********************getter**********************/
```

```
void * ADT_Vector_get_element(const ADT_Vector_t *vector, size_t
element);

/**********************setter**********************/
void ADT_Vector_set_destructor(ADT_Vector_t *vector, destructor_t
printer);
void ADT_Vector_set_printer(ADT_Vector_t *vector, printer_t
printer);
void ADT_Vector_set_header(ADT_Vector_t *vector, char * header);
void ADT_Vector_set_footer(ADT_Vector_t *vector, char * footer);
void ADT_Vector_set_compare(ADT_Vector_t *vector, compare_t
compare);
status_t ADT_Vector_set_element(ADT_Vector_t *vector, size_t
posicion,void *new_element);

#endif
```

## ADT_Vector.c

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

#include "ADT_Vector.h"
#include "utils.h"
#include "types.h"

/**********************CREADORES*******************************
******/
status_t ADT_Vector_new(ADT_Vector_t ** arr)
{
     if(arr==NULL)
          return ERROR_NULL_POINTER;

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

     if( ( (*arr)->elements = (void **) malloc (sizeof(void *) *
INIT_SIZE) )==NULL ){
          free(*arr);
          *arr = NULL;
          return ERROR_MEMORY;
```

```
	}

	(*arr)->size = INIT_SIZE;

	(*arr)->header = NULL;
	(*arr)->footer = NULL;
	(*arr)->printer = NULL;
	(*arr)->destructor = NULL;
	(*arr)->compare = NULL;

	return OK;
}

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

	if(vector==NULL)
		return ERROR_NULL_POINTER;

	if((*vector)->destructor==NULL)
		return ERROR_DESTRUCTOR_NOT_SET;

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

		(*vector)->elements[i]=NULL;
	}

	free((*vector)->elements);
	(*vector)->elements=NULL;

	free(*vector);
	*vector = NULL;

	return OK;
}

status_t ADT_Vector_sort(ADT_Vector_t *vector)
{
	size_t i,j, min;
	void *aux;
```

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

	if(vector->compare == NULL)
		return ERROR_COMPARE_NOT_SET;


	for(i=0; i < vector->size; i++){
		min=i;

		if(vector->elements[i]!=NULL)
		{

			for(j=i; j < vector->size; j++)
			{

				if (vector->elements[j]!=NULL)
				{
					if(
(vector->compare)(vector->elements[j], vector->elements[min]) >  0
)
					{
						min=j;
					}
					if(min!=i)
					{
					aux = vector->elements[min];
					vector->elements[min] =
vector->elements[i];
					vector->elements[i] = aux;
					}
				}


			}
		}

	}

	return OK;
}
```

```
status_t ADT_Vector_set_element(ADT_Vector_t *vector, size_t
posicion,  void *new_element)
{
      void **aux;
      if(vector==NULL || new_element ==NULL)
            return ERROR_NULL_POINTER;
      if( vector->size-1 < posicion)
      {

            if( ( aux= (void **) realloc (vector->elements,
sizeof(void *) * (posicion+1)) )==NULL )
                  return ERROR_MEMORY;

            vector->elements = aux;
            vector->size = posicion+1;
      }

      vector->elements[posicion] = new_element;
      return OK;
}

void * ADT_Vector_get_element(const ADT_Vector_t *vector, size_t
element)
{
      if(vector==NULL || element > vector->size)
            return NULL;
      return vector->elements[element];
}

status_t ADT_Vector_export(const ADT_Vector_t *vector, FILE *fo)
{
      status_t st;
      size_t i;

      if(vector==NULL)
            return ERROR_NULL_POINTER;

      if(vector->printer == NULL)
            return ERROR_PRINTER_NOT_SET;

      if(vector->header!=NULL){
            fprintf(fo, "%s\n", vector->header);
      }

      for(i=0; i < vector->size; i++){
```

```
            if( (st = ( *(vector->printer)) (vector->elements[i],fo)
)!=OK )
            {
                  return st;
            }
      }

      if(vector->footer!=NULL)
            fprintf(fo, "%s\n", vector->footer);

      return OK;

}


void ADT_Vector_set_destructor(ADT_Vector_t *vector, destructor_t
destructor)
{
      vector->destructor = destructor;
}
void ADT_Vector_set_compare(ADT_Vector_t *vector, compare_t
compare)
{
      vector->compare = compare;
}

void ADT_Vector_set_printer(ADT_Vector_t *vector, printer_t
printer)
{
      vector->printer = printer;
}

void ADT_Vector_set_header(ADT_Vector_t *vector, char * header)
{
      vector->header = header;
}

void ADT_Vector_set_footer(ADT_Vector_t *vector, char * footer)
{
      vector->footer = footer;
}
```

## utils.h

```
#ifndef UTILS__H
#define UTILS__H

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

#define INIT_CHOP 128
#define CHOP_SIZE 256

status_t read_line(FILE *fo,char ** s,bool_t *eof );
status_t split (const char *str,char * del,char ***parr,size_t
*n);
status_t destroy_strings(char *** s,size_t l);
status_t  clone_str(const char *s ,char **t);

#endif
```

## utils.c

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

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

status_t read_line(FILE *fo,char ** s,bool_t *eof )
{
      char c;
      size_t allow_size;
      size_t used_size;
      char * aux;


      if(fo ==NULL || s==NULL || eof==NULL)
            return ERROR_NULL_POINTER;

      if((*s=malloc(INIT_CHOP *sizeof(char)))==NULL)
            return ERROR_MEMORY;

      allow_size=INIT_CHOP;
      used_size=0;
```

```
	while((c=fgetc(fo))!=EOF && c!= '\n')
	{
		if(used_size==allow_size-1)
		{

if((aux=realloc(*s,(allow_size+CHOP_SIZE)*sizeof(char)))==NULL)
			{
				free(*s);
				*s=NULL;
				return ERROR_MEMORY;
			}
			allow_size+=CHOP_SIZE;

			*s=aux;
		}
		(*s)[used_size++]=c;
	}

	(*s)[used_size]='\0';
	*eof = (c==EOF) ? TRUE : FALSE;

	return OK;
}

status_t split (const char *str,char * del,char ***parr,size_t *n)
{
	char * line;
	char *q , *p;
	size_t i,j;
	status_t st;

	if(str==NULL || parr==NULL || n==NULL)
		return ERROR_NULL_POINTER;

	if((st=clone_str(str,&line))!=OK)
		return st;


	for(*n=0, i=0; line[i]; i++){
		for(j=0; del[j]; j++){
			if(line[i]==del[j])
				(*n)++;
		}
	}
```

```
	(*n)++;

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

	for(q=line,i=0;(p=strtok(q,del))!=NULL ;q=NULL,i++)
	{

		if((st=clone_str(p,&(*parr)[i]))!=OK)
		{
			free(line);
			destroy_strings(parr,i);
			*n=0;
			return st;

		}

	}



	free(line);
	return OK;
}


status_t clone_str(const char *s ,char **t)
{
	size_t i;

	if(s==NULL || t==NULL)
		return ERROR_NULL_POINTER;

	if((*t=malloc((strlen(s)+1)*sizeof(char)))==NULL)
		return ERROR_MEMORY;


	for(i=0;((*t)[i]=s[i]);i++);


	return OK;
```

```
}

status_t destroy_strings(char *** s,size_t l)
{

	size_t i;

	if(s==NULL)
		return ERROR_NULL_POINTER;

	for(i=0;i<l;i++)
	{
		free((*s)[i]);
		(*s)[i]=NULL;
	}

	free(*s);
	*s=NULL;

	return OK;
}
```

## error.c

```
#include <stdio.h>
#include "error.h"

static char * error []={
	MSG_OK,
	MSG_ERROR_NULL_POINTER,
	MSG_ERROR_INVOCATION_FEW_ARGUMENTS,
	MSG_ERROR_INVOCATION_MANY_ARGUMENTS,
	MSG_ERROR_INVOCATION_FUNCTION,
	MSG_ERROR_INITIAL_TIME_DATA,
	MSG_ERROR_FINAL_TIME_DATA,
	MSG_ERROR_INVALID_TIME,
	MSG_ERROR_OPEN_OUTPUT_FILE ,
	MSG_ERROR_OPEN_TRANSACTION_FILE,
	MSG_ERROR_CLOSE_OUTPUT_FILE,
	MSG_ERROR_MEMORY,
	MSG_ERROR_INFORMATION_ID_TRANSACTION_INVALID,
	MSG_ERROR_INFORMATION_ID_USER_INVALID,
	MSG_ERROR_INFORMATION_DATE,
	MSG_ERROR_INFORMATION_AMOUNT_INVALID,
```

```
      MSG_ERROR_DATE_FORMAT,
      MSG_ERROR_LONG_LENGTH_DATE,
      MSG_ERROR_SHORT_LENGTH_DATE,
      MSG_ERROR_DESTRUCTOR_NOT_SET,
      MSG_ERROR_COMPARE_NOT_SET,
      MSG_ERROR_PRINTER_NOT_SET,
      MSG_ERROR_INVALID_FORMAT

};

status_t print_error (status_t st)
{
      fprintf(stderr, "%s\n", error[st] );

      return OK;
}
```

## error.h

```
#ifndef ERROR__H
#define ERROR__H

#include "types.h"

#define MSG_OK "La operacion se realizo con exito"
#define MSG_ERROR_NULL_POINTER "Puntero nulo"
#define MSG_ERROR_INVOCATION_FEW_ARGUMENTS "Error en la
invocacion, pocos arguments"
#define MSG_ERROR_INVOCATION_MANY_ARGUMENTS "Error en la
invocacion, muchos argumentos"
#define MSG_ERROR_INVOCATION_FUNCTION "La funcion ha sido mal
invocada"
#define MSG_ERROR_INITIAL_TIME_DATA "El tiempo inicial es
incorrecto"
#define MSG_ERROR_FINAL_TIME_DATA "El tiempo final es incorrecto"
#define MSG_ERROR_INVALID_TIME "El tiempo inicial es mayor que el
tiempo final"
#define MSG_ERROR_OPEN_OUTPUT_FILE "No se pudo abrir el archivo de
salida"
#define MSG_ERROR_OPEN_TRANSACTION_FILE "No se pudo abrir el
archivo de transacciones"
#define MSG_ERROR_CLOSE_OUTPUT_FILE "No se pudo cerrar el archivo
de salida"
#define MSG_ERROR_MEMORY "Error en la memoria"
```

```
#define MSG_ERROR_INFORMATION_ID_TRANSACTION_INVALID "El id de la
transaccion es incorrecto"
#define MSG_ERROR_INFORMATION_ID_USER_INVALID "El id del usuario
es incorrecto"
#define MSG_ERROR_INFORMATION_DATE "La fecha ingresada es
incorrecta"
#define MSG_ERROR_INFORMATION_AMOUNT_INVALID "El monto ingresado
es incorrecto"
#define MSG_ERROR_DATE_FORMAT "El formato de la fecha es
incorrecto"
#define MSG_ERROR_LONG_LENGTH_DATE "La fecha ingresada es muy
larga"
#define MSG_ERROR_SHORT_LENGTH_DATE "La fecha ingresada es muy
corta"
#define MSG_ERROR_DESTRUCTOR_NOT_SET "No se ha configurado un
destructor"
#define MSG_ERROR_COMPARE_NOT_SET "No se ha configurado el
comparador"
#define MSG_ERROR_PRINTER_NOT_SET "No se ha configurado la forma
de imprimir"
#define MSG_ERROR_INVALID_FORMAT "El formato de salida es
incorrecto"


status_t print_error (status_t st);


#endif
```

## types.h

```
#ifndef TYPES__H
#define TYPES__H

typedef enum {
      OK=0,
      ERROR_NULL_POINTER=1,
      ERROR_INVOCATION_FEW_ARGUMENTS=2,
      ERROR_INVOCATION_MANY_ARGUMENTS=3,
      ERROR_INVOCATION_FUNCTION=4,
      ERROR_INITIAL_TIME_DATA=5,
      ERROR_FINAL_TIME_DATA=6,
      ERROR_INVALID_TIME=7,
      ERROR_OPEN_OUTPUT_FILE=8,
```

```
	ERROR_OPEN_TRANSACTION_FILE=9,
	ERROR_CLOSE_OUTPUT_FILE=10,
	ERROR_MEMORY=11,
	ERROR_INFORMATION_ID_TRANSACTION_INVALID=12,
	ERROR_INFORMATION_ID_USER_INVALID=13,
	ERROR_INFORMATION_DATE=14,
	ERROR_INFORMATION_AMOUNT_INVALID=15,
	ERROR_DATE_FORMAT=16,
	ERROR_LONG_LENGTH_DATE=17,
	ERROR_SHORT_LENGTH_DATE=18,
	ERROR_DESTRUCTOR_NOT_SET=19,
	ERROR_COMPARE_NOT_SET=20,
	ERROR_PRINTER_NOT_SET=21,
	ERROR_INVALID_FORMAT=22

}status_t;

typedef enum{
	FMT_CSV,
	FMT_XML
}format_t;

typedef enum{
	TRUE,
	FALSE
}bool_t;

typedef status_t (* printer_t)(const void *, FILE *);
typedef status_t (* destructor_t)(void *);
typedef int (*compare_t)(void *, void *);


#endif
```

## date.h

```
#ifndef DATE__H
#define DATE__H

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

```
#include "types.h"
#include "utils.h"

#define NUM_DEL_DATE 3
#define DATE_FIELDS 19
#define DAY_SPACE 2
#define MONTH_SPACE 2
#define YEAR_SPACE 4
#define HOUR_SPACE 2
#define MIN_SPACE 2
#define SEC_SPACE 2
#define DEL_DOT '.'
#define DEL_BAR '/'
#define DEL_MIDDLE_DASH '-'
#define POS_DEL_DD_MM 2
#define POS_DEL_MM_YYYY 5

typedef enum {
      DD_MM_YYYY,
      DDMMYYYY
}date_t;

status_t parse_date(char * date, date_t * fmt_date);
status_t convert_to_seconds_DD_MM_YYYY(char *str,time_t *
seconds);


#endif
```

## date.c

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

#include "types.h"
#include "utils.h"
#include "date.h"


status_t parse_date(char * date, date_t * fmt_date)
{
      char del[NUM_DEL_DATE]={DEL_DOT,DEL_MIDDLE_DASH,DEL_BAR};
```

```
	size_t i;

	if(fmt_date==NULL || fmt_date ==NULL)
		return ERROR_NULL_POINTER;

	/* ME FIJO QUE EL LARGO SEA VALIDO*/
	for (i=0;(date[i])!='\0';i++);

	if(i<DATE_FIELDS)
		return ERROR_SHORT_LENGTH_DATE;
	if(i>DATE_FIELDS)
		return ERROR_LONG_LENGTH_DATE;

	/*AVERIGUO EL FORMATO*/
	for(i=0;i<NUM_DEL_DATE;i++)
	{
		if(date[POS_DEL_DD_MM]==del[i] &&
date[POS_DEL_MM_YYYY]==del[i]){
			*fmt_date=DD_MM_YYYY;
			return OK;
		}
	}

	return ERROR_DATE_FORMAT;

}

status_t convert_to_seconds_DD_MM_YYYY(char *str,time_t * seconds)
{
	struct tm date;
	char *tmp;
	char
aux[DAY_SPACE+MONTH_SPACE+YEAR_SPACE+HOUR_SPACE+MIN_SPACE+SEC_SPAC
E+1];


	if(str==NULL)
		return ERROR_NULL_POINTER;

	/*COPIO LOS DIAS*/
	memcpy(aux,str,DAY_SPACE);
	aux[DAY_SPACE]='\0';

	date.tm_mday=(int)strtol(aux,&tmp,10);
		if(*tmp)
```

```
            return ERROR_INFORMATION_DATE;

      /*COPIO LOS MESES*/
      memcpy(aux,str+DAY_SPACE+1,MONTH_SPACE);
      aux[MONTH_SPACE]='\0';
      date.tm_mon=((int)strtol(aux,&tmp,10))-1;
      if(*tmp)
            return ERROR_INFORMATION_DATE;

      /*COPIO LOS AÑOS*/
      memcpy(aux,str+DAY_SPACE+MONTH_SPACE+2,YEAR_SPACE);
      aux[YEAR_SPACE]='\0';
      date.tm_year=((int)strtol(aux,&tmp,10))-1900;

      if(*tmp)
            return ERROR_INFORMATION_DATE;

      /*COPIO LAS HORAS*/

memcpy(aux,str+DAY_SPACE+MONTH_SPACE+YEAR_SPACE+3,HOUR_SPACE);
      aux[HOUR_SPACE]='\0';
      date.tm_hour=((int)strtol(aux,&tmp,10))-2;
      if(*tmp)
            return ERROR_INFORMATION_DATE;
            /*COPIO LOS MINUTOS*/

memcpy(aux,str+DAY_SPACE+MONTH_SPACE+YEAR_SPACE+HOUR_SPACE+4,MIN_S
PACE);
      aux[MIN_SPACE]='\0';
      date.tm_min=((int)strtol(aux,&tmp,10))-1;
      if(*tmp)
            return ERROR_INFORMATION_DATE;

      /*COPIO LOS SEGUNDOS*/

memcpy(aux,str+DAY_SPACE+MONTH_SPACE+YEAR_SPACE+HOUR_SPACE+MIN_SPA
CE+5,SEC_SPACE);
      aux[SEC_SPACE]='\0';
      date.tm_sec=((int)strtol(aux,&tmp,10))-1;
      if(*tmp)
            return ERROR_INFORMATION_DATE;


      *seconds=mktime(&date);
```

```
	return OK;
}
```

## config.h

```
#ifndef CONFIG__H
#define CONFIG__H

#include <stdio.h>

#include "types.h"


typedef struct {
	format_t format;
	size_t ti;
	size_t tf;
	char * input_file;
	char * output_file;
}config_t;

#endif
```

## process_transaction.h

```
#ifndef PROCES_TRANSACTION__H
#define PROCES_TRANSACTION__H

#include <stdio.h>

#include "ADT_Transaction.h"
#include "ADT_Vector.h"
#include "config.h"
#include "utils.h"
#include "types.h"

#define DEL '|'
#define CSV_HEADER "ID_USUARIO|INGRESOS|EGRESOS"

#define XML_HEADER "<?xml version=\"1.0\"
encoding=\"UTF-8\"?\">\n<usuarios>"
#define XML_FOOTER "</usuarios>"
```

```
status_t process_transaction(FILE *transaction_file, FILE
*output_file,config_t *config);


#endif
```

## process_transaction.c

```
#include <stdio.h>

#include "process_transaction.h"
#include "ADT_Transaction.h"
#include "ADT_Vector.h"
#include "config.h"
#include "utils.h"
#include "types.h"


status_t process_transaction(FILE *transaction_file, FILE
*output_file,config_t *config)
{

      ADT_Transaction_t * new_transaction;
      ADT_Transaction_t * current_transaction;
      ADT_Vector_t *array;
      size_t id_user;
      bool_t eof=FALSE;
      status_t st;
      int    new_amount;

      if(transaction_file==NULL || output_file ==NULL)
            return ERROR_NULL_POINTER;


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

      switch(config->format)
      {
            case FMT_CSV:
```

```
                ADT_Vector_set_printer(array,
&ADT_Transaction_save_as_CSV);
                ADT_Vector_set_header(array, CSV_HEADER);
                ADT_Vector_set_footer(array, NULL);
                break;
          case FMT_XML:
                ADT_Vector_set_printer(array,
&ADT_Transaction_save_as_XML);
                ADT_Vector_set_header(array, XML_HEADER);
                ADT_Vector_set_footer(array, XML_FOOTER);
                break;
          default:
                return ERROR_INVALID_FORMAT;

     }

     ADT_Vector_set_compare(array,
ADT_Transaction_compare_earnings);
     ADT_Vector_set_destructor(array, ADT_Transaction_destroy);


if((st=ADT_Transaction_load(transaction_file,&new_transaction,&eof
,DEL))!=OK)
     {
          ADT_Vector_destroy(&array);
          return st;
     }


     while(eof==FALSE)
     {

          if(new_transaction->seconds>config->ti &&
new_transaction->seconds<config->tf)
          {
                ADT_Transaction_get_id_user (new_transaction,
&id_user);
                current_transaction=ADT_Vector_get_element(array,
id_user);


                if(current_transaction==NULL)
                {
                     if((st=ADT_Vector_set_element(array,
id_user,new_transaction))!=OK)
```

```
                    {

ADT_Transaction_destroy(&new_transaction);
                         ADT_Vector_destroy(&array);
                         return st;
                    }

               }else if(current_transaction!=NULL)
               {

ADT_Transaction_get_amount(new_transaction,&new_amount);

ADT_Transaction_add_amount(current_transaction,new_amount);
               }

          }else{
               ADT_Transaction_destroy(&new_transaction);
          }




if((st=ADT_Transaction_load(transaction_file,&new_transaction,&eof
,DEL))!=OK)
          {
               ADT_Vector_destroy(&array);
               return st;
          }


     }

     /*******ORDENO**********/
     ADT_Vector_sort(array);

     /*******IMPRIMO*********/
     ADT_Vector_export(array,output_file);


     ADT_Vector_destroy(&array);

     return OK;
}
```

## makefile

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

all: analisis_bancario

analisis_bancario: main_transaction.o ADT_Vector.o
ADT_Transaction.o process_transaction.o utils.o error.o date.o
     $(CC) $(CFLAGS) -o analisis_bancario main_transaction.o
ADT_Vector.o ADT_Transaction.o process_transaction.o utils.o
error.o date.o

########################### ADTs ############################
ADT_Vector.o: ADT_Vector.c ADT_Vector.h types.h utils.h
     $(CC) $(CFLAGS) -o ADT_Vector.o -c ADT_Vector.c

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

#################### GENERAL FUNCTIONS ####################
date.o: date.c date.h types.h utils.h
     $(CC) $(CFLAGS) -o date.o -c date.c

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

process_transaction.o: process_transaction.c process_transaction.h
types.h utils.h ADT_Vector.h ADT_Transaction.h
     $(CC) $(CFLAGS) -o process_transaction.o -c
process_transaction.c

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

########################### MAIN ############################
main_transaction.o: main_transaction.c main_transaction.h types.h
error.h process_transaction.h config.h
     $(CC) $(CFLAGS) -o main_transaction.o -c main_transaction.c
```