UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Segundo cuatrimestre Año 2020/2021

TRABAJO PRÁCTICO N.º 2 - Validación de operaciones con tarjetas de crédito

MATERIA: Algoritmos y Programación I (95.11)
CURSO: 01 - Ing. Cardozo

FECHA DE ENTREGA: 31 de marzo de 2021

Alumnos

Apellido	Nombres	Padrón	Correo electrónico
Fernández	Ezequiel Oscar	107186	eze.f.tec@gmail.com
Marchan Gonzalez	Dayher Miguel	104735	dayhermarchan@gmail.com

Introducción

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 validar un archivo con transacciones con tarjetas de crédito y presentar un informe en diversos formatos.

Desarrollo

En este T.P. se pide escribir un programa ejecutable que procese y valide un archivo con operaciones con tarjeta de crédito de una cadena de supermercados utilizando el algoritmo de Luhn y genere archivos de salida en formato CSV y XML con las operaciones inválidas detectadas en cada sucursal del supermercado.

El programa ejecutable, denominado “luhn_analyzer.exe” (WinXX) o “luhn_analyzer” (Unix), debe ser invocado de la siguiente forma:

WinXX:

luhn_analyzer –fmt <formato> -out <salida> -in <entrada>

UNIX:

./luhn_analyzer –fmt <formato> -out <salida> -in <entrada>

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

COMANDO	DESCRIPCIÓN	VALOR	TIPO DE DATO
fmt	Formato del índice a generar	“csv”	Cadena de caracteres
fmt	Formato del índice a generar	“xml”	Cadena de caracteres

Archivo de salida <salida>

Es la ruta del archivo con los resultados del análisis que se desea generar.

Archivos de entrada <entrada>

Es la ruta del archivo de texto que contiene las operaciones con tarjeta de crédito en formato CSV.

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.

Operación

El programa debe analizar el archivo que contienen las transacciones con tarjeta de crédito, detectar las operaciones con números de tarjetas inválidos en cada sucursal y generar los archivos en los formatos de salida correspondientes.

Es importante aclarar que el programa debe considerar que la cadena de supermercados tiene una cantidad de sucursales variable y que puede crecer a lo largo del tiempo, con lo cual será necesario utilizar memoria dinámica para poder resolver el problema.

Para la validación de los números de tarjeta de crédito se deberá utilizar el algoritmo de Luhn.Este algoritmo detecta cualquier error de un único dígito, así como casi todas las transposiciones de dígitos adyacentes. Fue desarrollado por Hans Peter Luhn en 1954 trabajando para IBM, dando lugar a la patente U.S. Patent No. 2,950,048. En la actualidad su especificación se encuentra en la norma
ISO/IEC 7812-1.

Formato de entrada

El archivo de entrada se encuentra en formato CSV y contiene la información tabulada de la siguiente manera:

Transacción, Número de tarjeta de crédito, Sucursal, Monto, Descripción
213412, 4916288217067475, 123412, 250.60, Shampo
184535, 4916288217067475, 532123, 100.50, Manaos 1Lt
123956, 4916288217067475, 123412, 50.12, Leche
783619, 4916288217067475, 543256, 70.54, Yogurt
938179, 4916288217067475, 123412, 200.18, Cerveza
891234, 4916288217067475, 123555, 80.40, Doritos

Formato de salida

a) Formato CSV

Se trata de un archivo CSV sin encabezado, campos sin calificador y carácter separador “|”.

b) Formato XML

Ambos formatos de salida deberán estar ordenados por identificador de sucursal creciente y

monto de
las transacciones decreciente de forma tal de poder identificar rápidamente las transacciones inválidas
más importantes de cada sucursal.

Restricciones

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

− Se debe recurrir al uso de T.D.A. vector y/o lista para lograr almacenar las transacciones inválidas de cada sucursal.

− Se debe recurrir al uso de T.D.A. para almacenar los elementos de información correspondientes al mensaje de cada transacción.

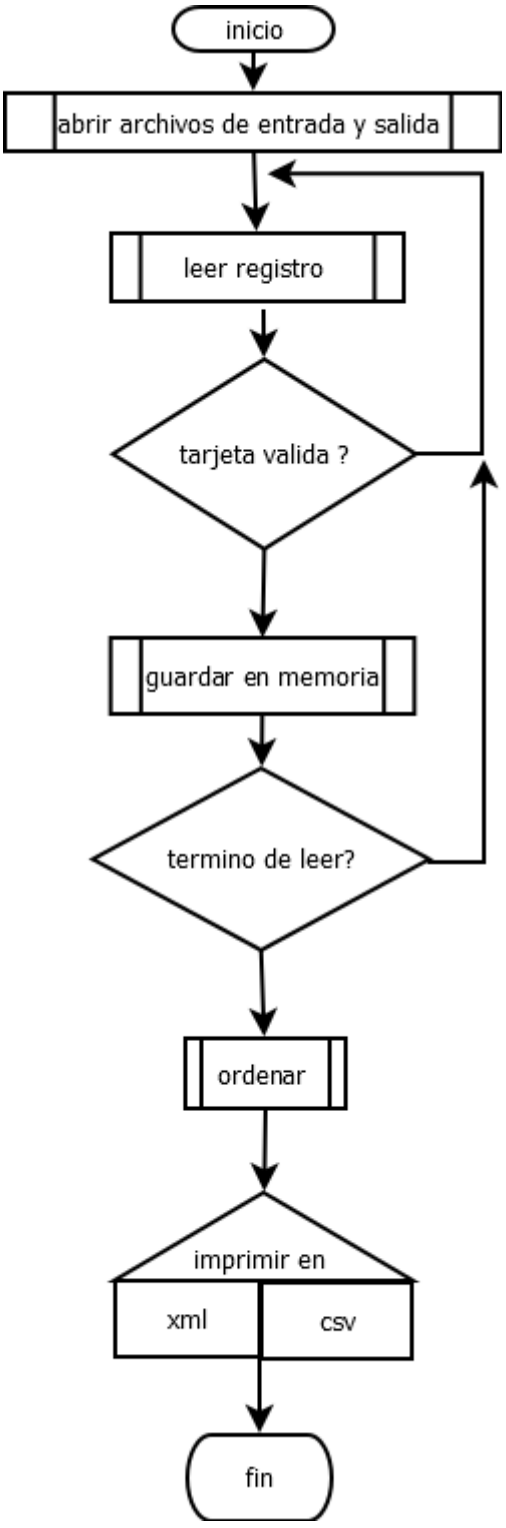
− Se debe recurrir al uso de punteros a función a fin de parametrizar la impresión de los archivos de salida.

− Se deben utilizar funciones y una adecuada modularización.

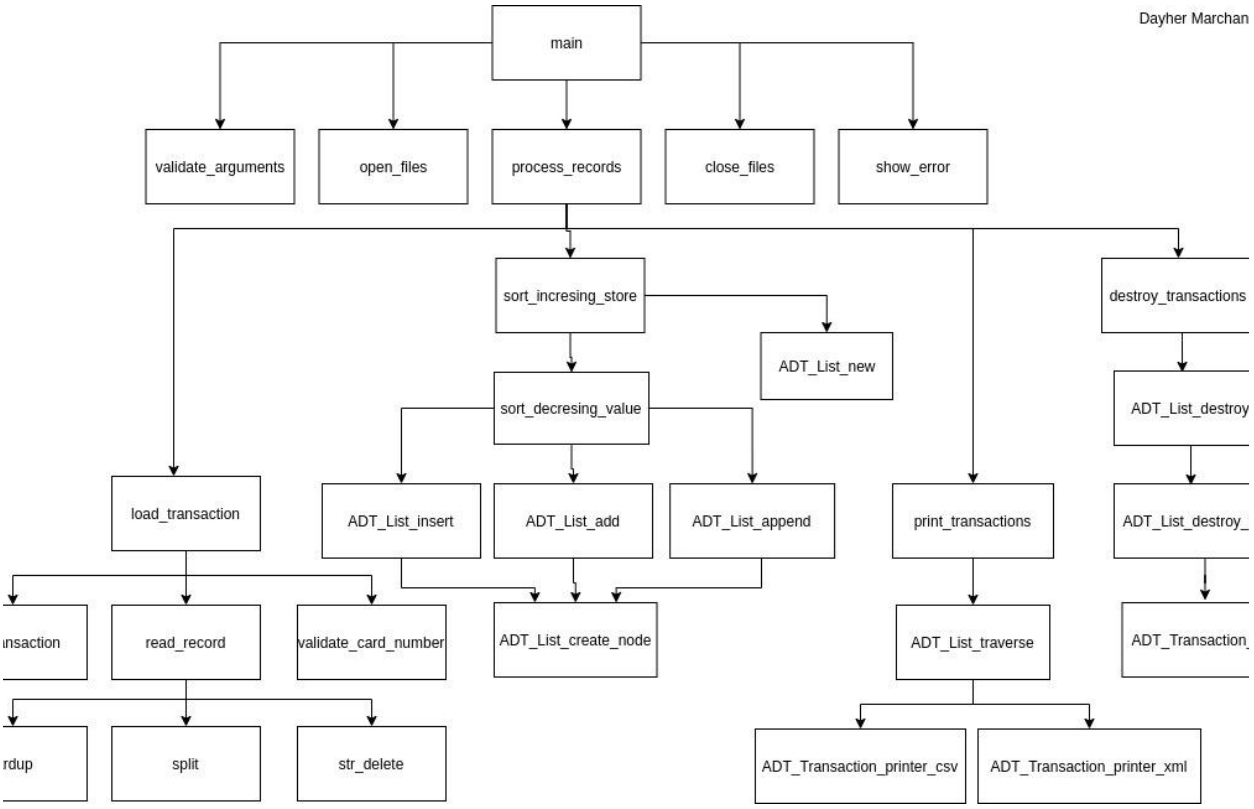
− Se debe construir un proyecto mediante la utilización de makefile.

− 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.

3) Diagrama de flujo del programa



4) Diagrama de funciones

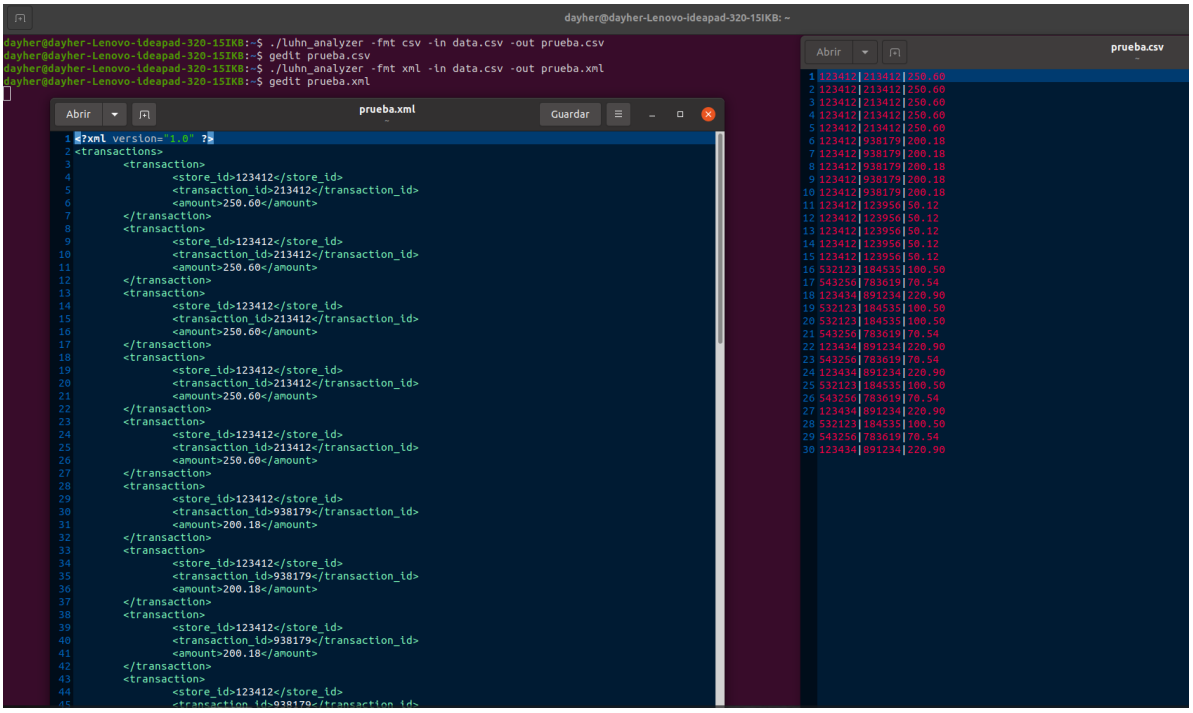


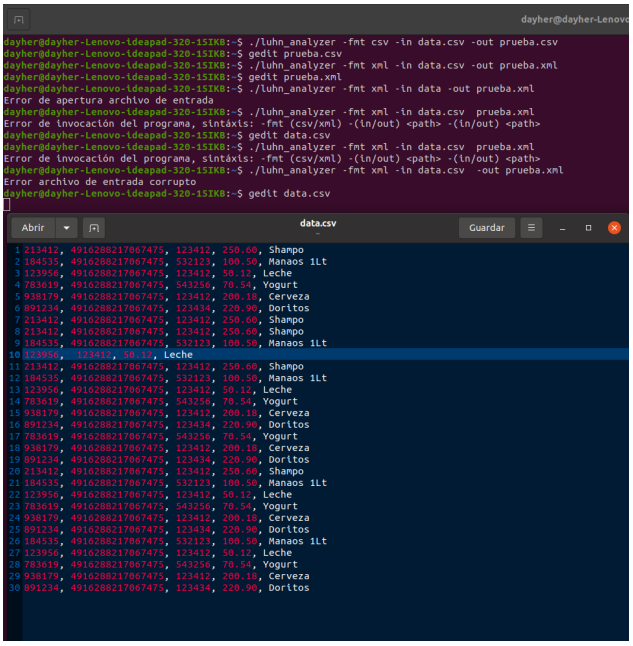
5) Alternativas consideradas y las estrategias adoptadas

La primera opción que se consideró fue la de almacenar los registros en memoria dinámica utilizando un arreglo dinámico de estructuras empleando la estrategia de crecimiento aritmética y luego validar cada registro, pero esto requería más instrucciones al programa que se evitaron realizando un ciclo dentro de la función de lectura de líneas del archivo de entrada, dentro del ciclo se válido el número de tarjeta de crédito, de esta forma sólo se almacenaban los datos de los registros deseados.

Debido a los requerimientos del programa los argumentos en línea de órdenes deben estar precedidos por un identificador (como -fmt) por lo que la alternativa de que el programa leyera los argumentos en pares en cualquier orden resultó factible.

6) Resultados de ejecución





7) Reseña de problemas encontrados y soluciones implementadas

Por la forma en la que los datos requieren ser ordenados, esto es por sucursal y luego por monto de la transacción, implementar un ADT_Lista supone comparar cada registro leído con todos los registros ya almacenados, esto es factible si la cantidad de registros con

número de tarjeta inválido es discreta, pero a mayor cantidad de estos, el número de veces que se compara con registros sucesivos que poseen el mismo identificador de sucursal aumentará tomando más tiempo para el programa. Se intentó ahorrar este tiempo separando cada sucursal como un nodo de una lista , que a su vez contiene una lista de transacciones agrupadas por el identificador de sucursal y ordenadas por monto de la transacción.

8) Conclusiones

El uso de macros en los mensajes imprimibles hace que sea más fácil un cambio de idioma del registro y los errores.

El uso de la modularización y la implementación hace que sea más organizado el manejo de datos

Para la compilación del programa se utilizó el aplicativo make el cual agiliza el proceso de compilación ya que no es necesario volver a compilar cada programa y subprograma.

9) Script de Compilación

```
luhn_analyzer: process.o files.o errors.o main.o io.o strings.o adt.o
	gcc process.o  adt.o files.o errors.o main.o io.o strings.o  -o luhn_analyzer

main.o: errors.h files.h process.h main.h main.c
	gcc -c main.c

adt.o: errors.h adt.h adt.c
	gcc -c adt.c

process.o: errors.h io.h process.h process.c adt.h
	gcc -c process.c

io.o: errors.h io.h strings.h io.c adt.h adt.c
	gcc -c io.c

strings.o: errors.h strings.c
	gcc -c strings.c

files.o: files.h errors.h files.c
	gcc -c files.c

errors.o: errors.h errors.c
	gcc -c errors.c
```