**UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPEL**

**CARRERA:**

**SOFTWARE**

**ASIGNATURA:**

**FUNDAMENTOS DE PROGRAMACIÓN**

**ALUMNOS:**

**ERAZO MOREIRA JOSUE DAMIAN**

**ESPINOSA FERNÁNDEZ ALEX GABRIEL**

**TEMA:**

**ELABORACIÓN DE UN PROGRAMA DE FACTURACION DE AUTOS EN C++ QUE INCLUYA FICHEROS Y ESTRUCTURAS**

**FECHA:**

**23/08/2016**

**“Ver a gran distancia es una cosa; ir hasta allí es algo bien distinto.”**

**(Constantin Brancusi)**

# Objetivo

Consolidar los conocimientos de codificación adquiridos a lo largo del periodo académico a través de la elaboración de un programa de facturación en lenguaje C para mejorar la lógica y lenguaje de programación lo que nos brindará más opciones solución a la hora de resolver problemas.

* 1. Utilizar correctamente la lógica de programación forjada en el transcurso del semestre
  2. Realizar investigaciones que contribuyan a un desarrollo eficiente del código fuente

# Marco Teórico

## Lenguaje C.

“Creado por Brian Kernighan y Dennis Ritchie a mediados de los años 70. El lenguaje C es un lenguaje para programadores en el sentido de que proporciona una gran flexibilidad de programación y una muy baja comprobación de incorrecciones, de forma que el lenguaje deja bajo la responsabilidad del programador acciones que otros lenguajes realizan por si mismos.”

(Esteban)

## Vectores

“Un vector, también llamado array(arreglo) unidimensional, es una estructura de datos que permite agrupar elementos del mismo tipo y almacenarlos en un solo bloque de memoria juntos, uno despues de otro. A este grupo de elementos se les identifica por un mismo nombre y la posición en la que se encuentran. La primera posición del array es la posición 0.

Podríamos agrupar en un array una serie de elementos de tipo enteros, flotantes, caracteres, objetos, etc.

Crear un vector en C++ es sencillo, seguimos la siguiente sintaxix: Tipo nombre[tamanyo];”

Ejm:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3 | int a[5]; // Vector de 5 enteros  float b[5]; // vector de 5 flotantes  Producto product[5]; // vector de 5 objetos de tipo Producto | |

(ronnyml, 2009)

## 2.3 Instrucciones Repetitivas

## A la forma de programación en la que solo se ejecuta una línea de código tras otra, se le conoce como programación secuencial, pero es mucho más eficiente la llamada programación estructurada, en la cual es posible ejecutar bloques o estructuras de código, las cuales internamente se siguen ejecutando secuencialmente, pero que pueden repetirse o ser invocadas en el momento en que se les necesite. El lenguaje C y también el C++, son lenguajes que soportan el paradigma de Programación Estructurada y por ello incorporan estructuras repetitivas, las cuales permiten romper un poco la secuencialidad de las sentencias. (Gómez, 2015)

## 2.5 Instrucción repetitiva for

## “La palabra for en ingles significa “por” o “para”, es por ello que podríamos leer o traducir lo que expresa un ciclo for como “para las condiciones dadas entre los paréntesis (()), hacer lo que se encuentra dentro de las llaves ({})”. Ya que esta estructura se compone de una cabecera que se inicia con la palabra reservada for, seguida por una serie de parámetros dados entre paréntesis y luego un cuerpo delimitado por llaves ({}), en el cual se encuentran las instrucciones que se pretenden repetir.

## Entre los paréntesis (()), se encuentran tres partes separadas por el símbolo punto y coma (;), la primera de ellas corresponde a la inicialización de una variable contador, que se encargara obviamente de contar los giros del ciclo, luego en la segunda parte, debe colocarse una expresión condicional como las utilizadas entre los paréntesis de las sentencias if, la cual se encargara de determinar cuándo se producirá o no, una nueva iteración del bucle y finalmente en la tercera parte se coloca otra expresión en la que se indica, cómo se efectuara el incremento o decremento de la variable contador cada vez que ocurra una nueva repetición.” (Gómez, 2015)

## 2.6 Función repetitiva While

## “La palabra inglesa While significa “mientras” en español, por lo tanto el bucle while o estructura repetitiva while, lógicamente indica la ejecución de las sentencias que se encuentran agrupadas en el cuerpo de esta estructura de control, el cual está delimitado por llaves ({}), solo si la condición ubicada entre los paréntesis (()), es evaluada como verdadera. El funcionamiento de esta estructura es muy similar al For, con la diferencia de no poseer una variable contador, en el resto de su funcionamiento es igual, presenta una cabecera que comienza por la palabra reservada while, seguida de entre paréntesis una expresión condicional que debe cumplirse para que se pueda generar una nueva iteración del ciclo.

## Este tipo de estructuras repetitivas, es especialmente útil cuando no se puede definir con precisión cuantas iteraciones o repeticiones deben efectuarse a un bloque de sentencias para poder dar resolución al problema que se esté afrontando.” (Gómez, 2015)

## Función repetitiva do while

## La palabra Do en español significa “hacer” por lo tanto este bucle se puede entender como: “hacer una serie de pasos mientras se cumpla la condición final”. En la parte superior de la estructura se coloca la palabra “do” seguida del bloque delimitado por llaves ({}) con las sentencias que se pretenden repetir y luego en la parte inferir se coloca la palabra “while” seguida de una expresión condicional ubicada entre paréntesis (()). A diferencia tanto del bucle For como del While, el Do-While termina con un punto y coma (;) al final de la estructura, luego del ultimo paréntesis ()) de la condición de parada del ciclo.

## El bloque de sentencias internas al ciclo, las que están dentro de las llaves ({}), nunca se ejecuten, pues si la condición de parada de la estructura repetitiva se evalúa como falsa en la cabecera del bucle (la parte superior), el ciclo culmina sin realizar ninguna iteración, en cambio con el bucle Do-While siempre se realiza al menos una iteración, es decir las sentencias internas, las que están agrupadas entre las llaves ({}), se ejecutan obligatoriamente una vez como mínimo, ya que la condición no se encuentra en la cabecera sino al pie (la parte inferior) de la estructura. (Gómez, 2015)

## Estructuras

## “Una estructura contiene varios datos. La forma de definir una estructura es haciendo uso de la palabra clave struct. Aqui hay ejemplo de la declaracion de una estructura:

|  |
| --- |
| **struct** mystruct  {  int int\_member;  double double\_member;  char string\_member[25];  } variable; |

## "variable" es una instancia de "mystruct" y no es necesario ponerla aquí. Se podria omitir de la declaracion de "mystruct" y más tarde declararla usando:

|  |
| --- |
| **struct** mystruct variable; |

## La estructura misma no tiene nombre (por la ausencia de nombre en la primera linea), pero tiene de alias "Mystruct". Entonces se puede usar así:

|  |
| --- |
| Mystruct variable; |

## Note que es una convencion, y una buena costumbre usar mayúscula en la primera letra de un sinónimo de tipo. De todos modos lo importante es darle algún identificador para poder hacer referencia a la estructura: podriamos tener una estructura de datos recursiva de algún tipo.”

## (Anónimo, 2016)

## Archivos

|  |
| --- |
| **2.9.1 El puntero FILE \*** |

Todas las funciones de estrada/salida estándar usan este puntero para conseguir información sobre el fichero abierto. Este puntero no apunta al archivo sino a una [estructura](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/estructuras.htm) que contiene información sobre él. Esta estructura incluye entre otras cosas información sobre el nombre del archivo, la dirección de la zona de memoria donde se almacena el fichero, tamaño del buffer.

|  |  |
| --- | --- |
|  | **2.9.2 Apertura del fichero para lectura: fopen** |

Ahora es preciso abrir el fichero. Para ello usamos la función fopen. Esta función tiene el siguiente formato:

*FILE \*fopen(const char \*nombre\_fichero, const char \*modo);*

En el ejemplo usábamos:

fichero = fopen("origen.txt","r");

El nombre de fichero se puede indicar directamente (como en el ejemplo) o usando una variable y se puede abrir de diversas formas. Esto se especifica con el parámetro **modo**. Los modos posibles son:

|  |  |
| --- | --- |
| r | Abre un fichero existente para lectura. |
| w | Crea un fichero nuevo (o borra su contenido si existe) y lo abre para escritura. |
| a | Abre un fichero (si no existe lo crea) para escritura. El puntero se sitúa al final del archivo, de forma que se puedan añadir datos si borrar los existentes. |

Se pueden añadir una serie de modificadores siguiendo a los modos anteriores:

|  |  |
| --- | --- |
| B | Abre el fichero en modo binario. |
| T | Abre el fichero en modo texto. |
| + | Abre el fichero para lectura y escritura. |

Ejemplos de combinaciones:

* rb+ - Abre el fichero en modo binario para lectura y escritura.
* w+ - Crea (o lo borra si existe) un fichero para lectura y escritura.
* rt - Abre un archivo existente en modo texto para lectura.

|  |  |
| --- | --- |
|  | **2.9.3 Comprobación de fichero abierto** |

Un aspecto muy importante después de intentar abrir un fichero es comprobar si realmente está abierto. El sistema no es infalible y pueden producirse fallos: el fichero puede no existir, estar dañado o no tener permisos de lectura. Si intentamos realizar operaciones sobre un puntero tipo FILE cuando no se ha conseguido abrir el fichero puede haber problemas.

Si el fichero no se ha abierto, el puntero fichero (puntero a FILE) tendrá el valor NULL, si se ha abierto con éxito tendrá un valor distinto de NULL. Por lo tanto, para comprobar si ha habido errores, nos fijamos en el valor del puntero:

|  |
| --- |
| if (fichero == NULL) {  printf( "No se puede abrir el fichero.\n" );  exit( 1 );  } |

Si fichero == NULL significa que no se ha podido abrir por alguna causa. Lo más conveniente es salir del programa. Para salir utilizamos la función exit(), donde el argumento 1 indica al sistema operativo que se han producido errores. Esta función precisa del archivo de cabecera stdlib.h.

|  |  |
| --- | --- |
|  | **2.9.4 Lectura de caracteres del fichero: getc** |
|  |  |

Ahora ya podemos empezar a leer el fichero. Para ello podemos utilizar la función getc, que lee los caracteres uno a uno. Se puede usar también la función getc (son equivalentes: la diferencia es que getcestá implementada como macro). Además de estas dos, existen otras funciones como [fgets](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/archivos.htm#fgets), [fread](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/archivos.htm#fread) que leen más de un carácter.

El formato de la función getc (y de fgetc) es:

*int getc(FILE \*fichero);*

En este caso lo usamos como:

|  |
| --- |
| letra = getc( fichero ); |

Tomamos un carácter de fichero, lo almacenamos en letra y el puntero  del fichero se coloca en el siguiente carácter.

|  |  |
| --- | --- |
|  | **2.9.5 Comprobación de fin de fichero: feof** |

Cuando entramos en el bucle while, la lectura se realiza hasta que se encuentre el final del fichero. La detección del final del fichero se puede llevar a cabo de dos formas:

* con la función [feof](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/archivos.htm#feof)
* comprobando si el valor de letra es [**EOF**](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/archivos.htm#eof)  (el último carácter de cualquier archivo).

En el ejemplo hemos usado la función feof. Esta función es de la forma:

*int feof(FILE \*fichero);*

Esta función comprueba si se ha llegado al final de fichero, en cuyo caso devuelve un valor distinto de 0. Si no se ha llegado al final de fichero devuelve un cero. Por eso lo usamos del siguiente modo:

|  |
| --- |
| while ( feof(fichero) == 0 ) |

Ó

|  |
| --- |
| while ( !feof(fichero) ) |

La segunda forma que comentaba arriba consiste en comprobar si el carácter leído es el de fin de fichero **EOF**:

|  |
| --- |
| while ( letra != EOF ) |

Sin embargo, cuando trabajamos con ficheros de texto no hay ningún problema, pero si estamos manejando un fichero binario podemos encontrarnos EOF antes del fin de fichero. Por eso es preferible usarfeof.

|  |  |
| --- | --- |
|  | **2.9.6 Cierre del fichero: fclose** |

Una vez realizadas todas las operaciones deseadas sobre el fichero hay que cerrarlo. Es importante no olvidar este paso pues el fichero podría corromperse. Al cerrarlo se vacían los *buffers* y se guarda el fichero en disco. Un fichero se cierra mediante la función fclose. Esta función tiene el siguiente formato:

*FILE \*fclose(FILE \* stream);*

Aquí stream es el puntero a FILE devuelto por [fopen](http://platea.pntic.mec.es/vgonzale/cyr_0204/cyr_01/control/lengua_C/archivos.htm#fopen) cuando el archivo se abrió. Si todo va bien, devuelve un cero, si hay problemas, devuelve otro valor. Estos problemas se pueden producir si el disco está lleno, por ejempl

|  |
| --- |
| if (fclose(fichero)!= 0)  printf( "Problemas al cerrar el fichero\n" ); |

|  |  |
| --- | --- |
|  | **2.9.7 Lectura de líneas del fichero: fgets** |

Esta función es muy útil para leer líneas completas desde un fichero. Su formato es:

*char \*fgets(char \*buffer, int longitud\_max, FILE \*fichero);*

Esta función lee desde el fichero hasta que encuentra un carácter '\n' o hasta que lee longitud\_max-1 caracteres y añade '\0' al final de la cadena (y retiene el carácter '\n'). La cadena leída la almacena enbuffer. Si se encuentra EOF antes de leer ningún carácter, o si se produce un error, la función devuelve NULL; en caso contrario devuelve la dirección de buffer.

# Planteamiento y Solución

* 1. **Análisis**

**3.2.1 Problema:** Como realizar un sistema de facturación de un patio de vehículos.

**3.2.2 Objetivo:** Elaborar un sistema de facturación de vehículos

**3.2.3 Datos de entrada:**

**3.2.4 Datos proceso:**

**3.2.5 Datos de salida:**

**3.2.6 Restricciones:**

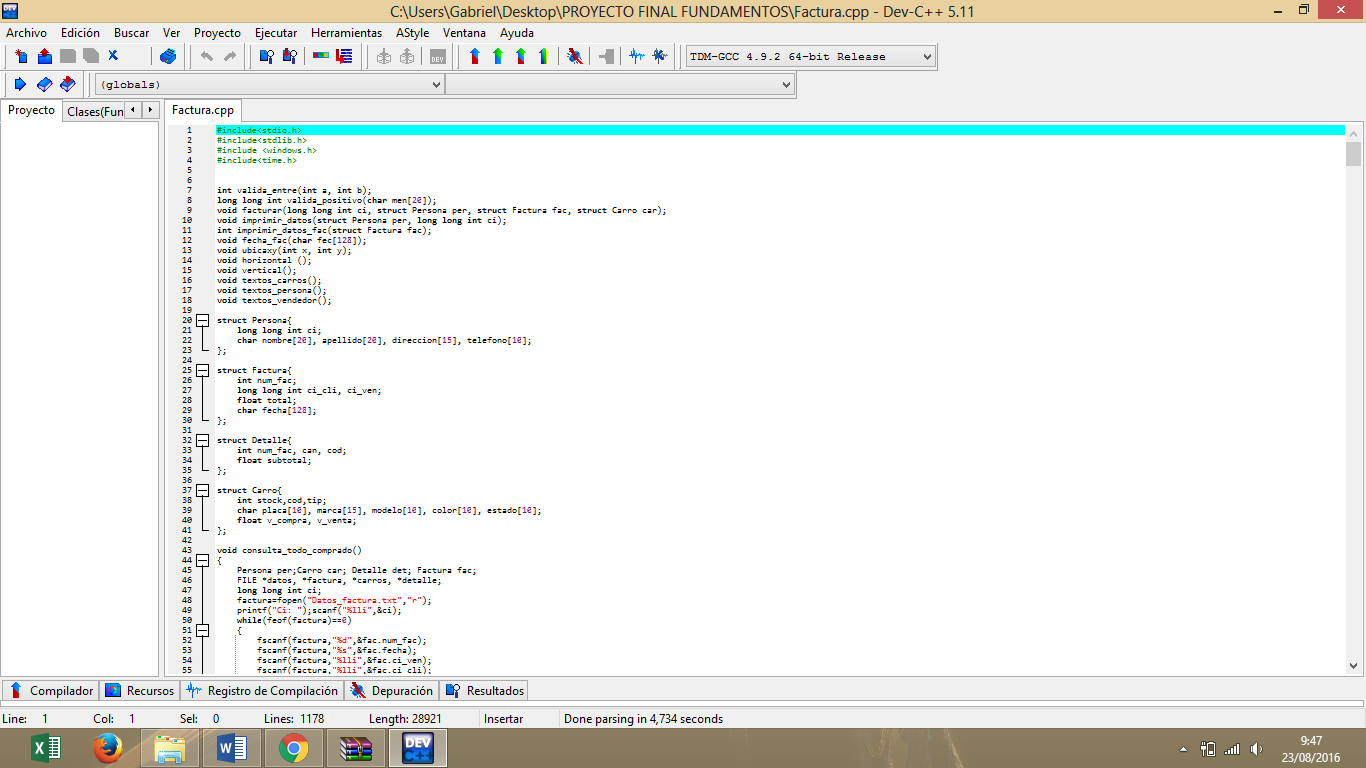
**3.2.7 Diseño:**

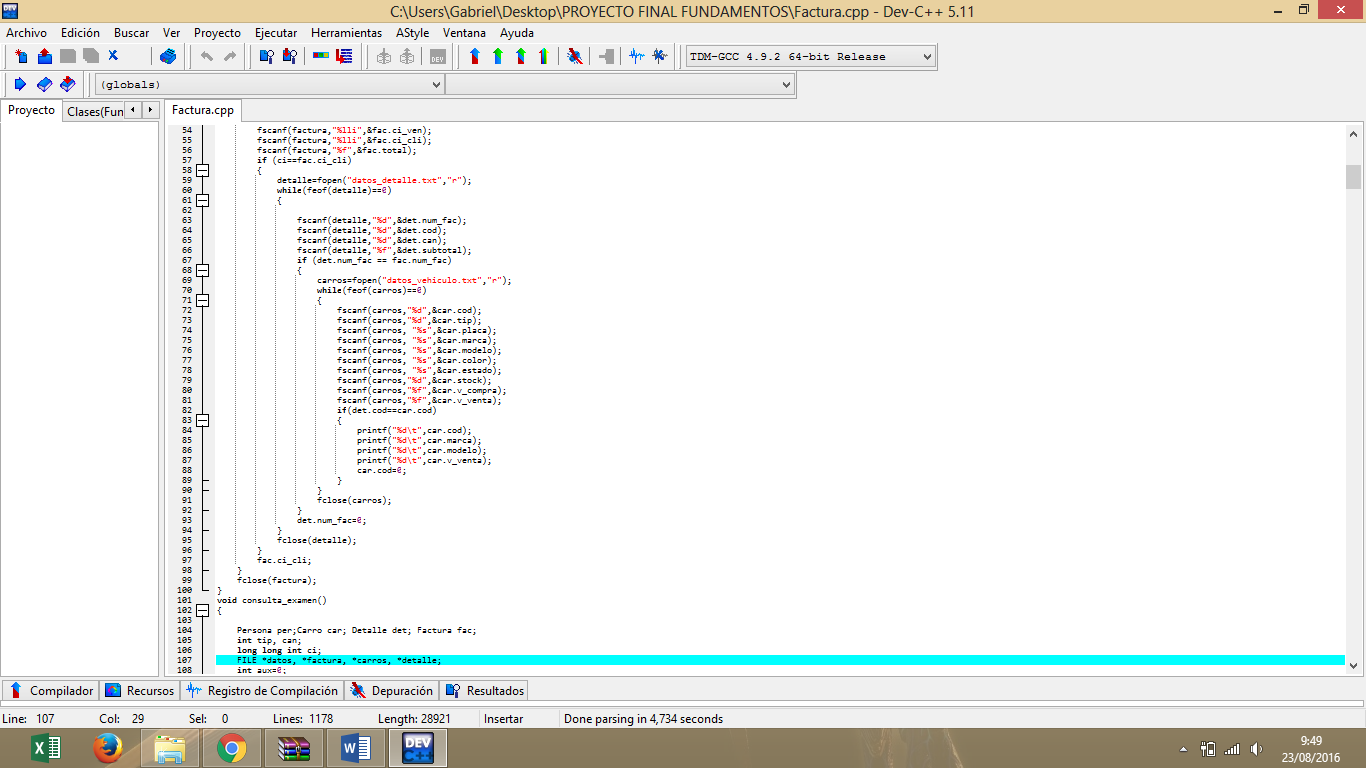
|  |
| --- |
| **MENU PRINCIPAL**   1. **ADMINISTRAR** 2. **FACTURAR** 3. **SALIR**   **\*\*OPCION A EJECUTAR** |

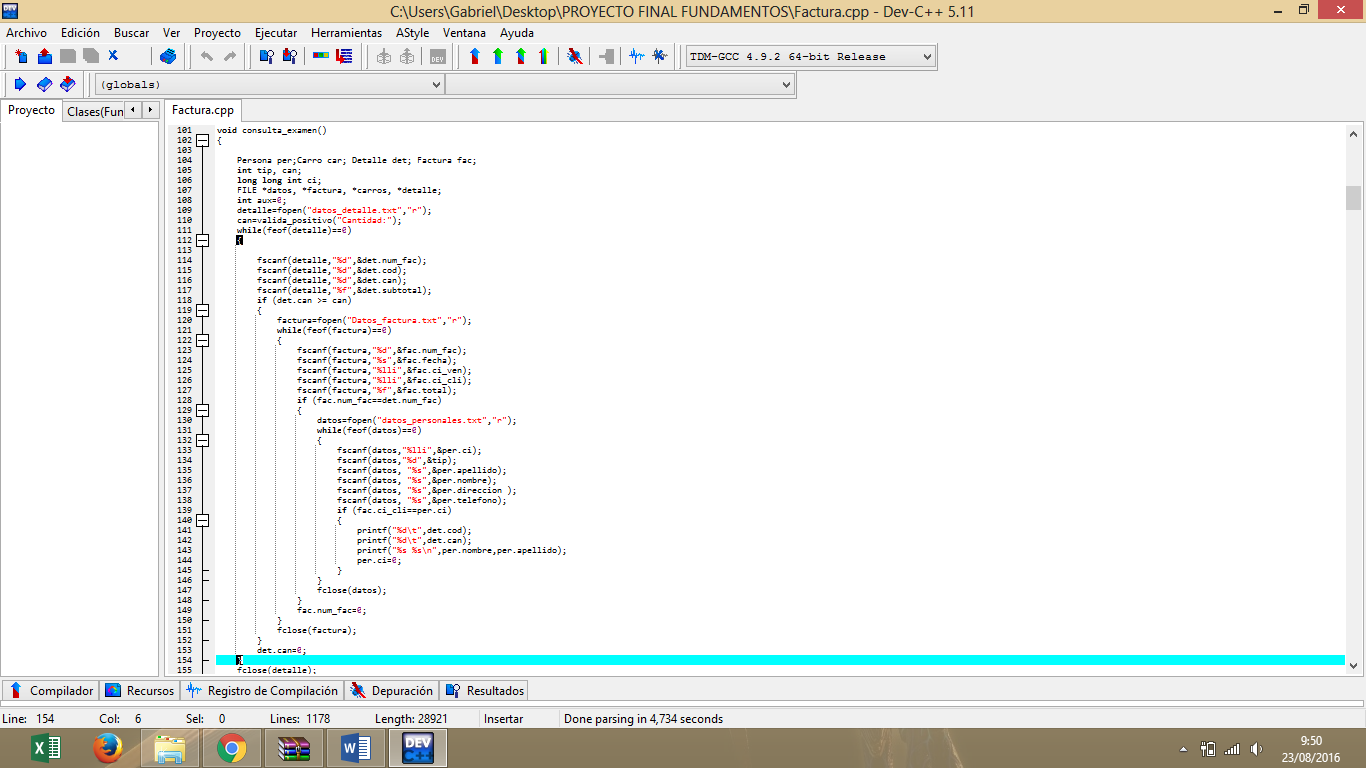
|  |
| --- |
| ADMINISTRADOR   1. INGRESAR 2. CONSULTAR 3. MODIFICAR 4. REGRESAR   \*\*OPCION A EJECUTAR |

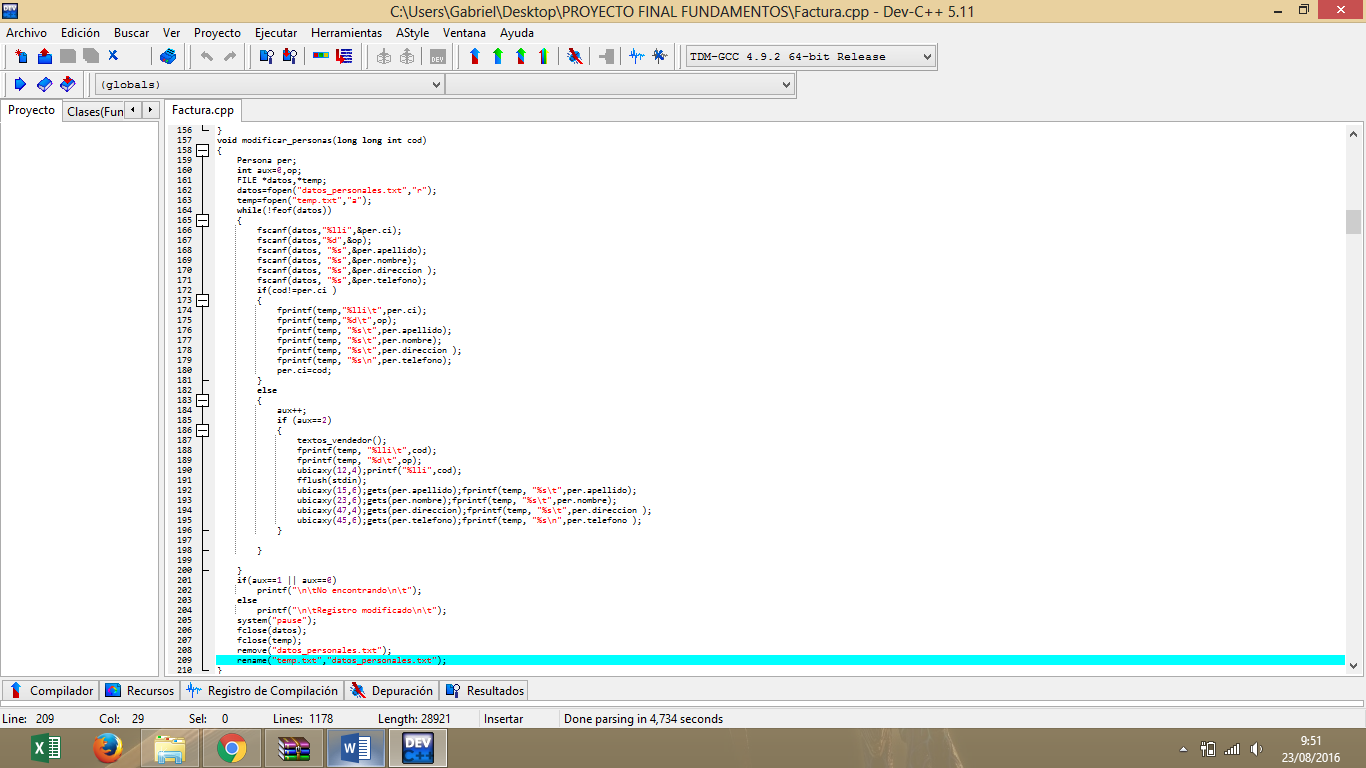
|  |
| --- |
| FACTURACION  IDENTIFICAR VENDEDOR  CEDULA:\_\_\_\_\_\_\_\_\_\_ |

1. Codificación

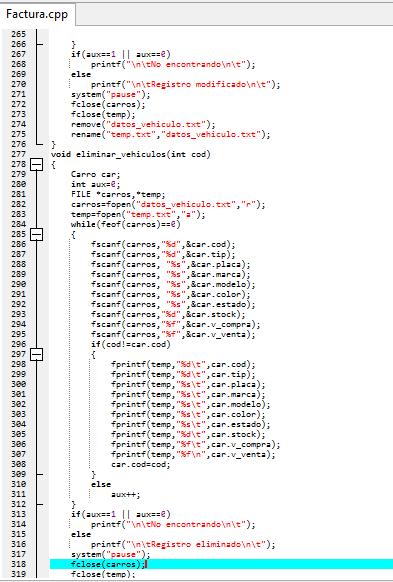












## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

# Conclusiones

## Gracias al desarrollo de un código fuente propio enfocado a un tema de

## Dominio popular como lo la facturación de mercaderías podemos concluir que:

## El dominio adecuado del lenguaje de programación C nos ha servido como herramienta para el correcto desarrollo de un programa cuyo fin es facilitar el trabajo del usuario.

## Con la ayuda de los conocimientos adquiridos en el trascurso del periodo estudiantil y con base en investigación personal nos vemos en la capacidad de estructurar un código fuente que tenga una funcionalidad eficiente para el ejecutor del programa.

## Recomendaciones

## Al implementar un tema no tratado con anterioridad tener en cuenta el correcto uso y sintaxis al momento de realizar la estructura del código

## Revisar el código de forma continua para que se pueda tener una forma más eficiente de realización es decir disminuir líneas de código y aumentar la utilidad.

# Bibliografía:

# Bibliografía

Anónimo. (04 de 02 de 2016). *wikilibros*. Obtenido de https://es.wikibooks.org/wiki/Programaci%C3%B3n\_en\_C/Estructuras\_y\_Uniones

Esteban, E. V. (s.f.). *informatica.uv.es*. Obtenido de http://informatica.uv.es/estguia/ATD/apuntes/laboratorio/Lenguaje-C.pdf

Gómez, N. (07 de 2015). *pixelesybytes.com*. Obtenido de http://www.pixelesybytes.com/2015/07/bucles-ciclos-do-while.html

Microsoft. (2016). *Microsoft*. Obtenido de Microsoft.

*programavideojuegos.blogspot.com*. (05 de 2013). Obtenido de http://programavideojuegos.blogspot.com/2013/05/juego-el-ahorcado-en-c.html

Pseint. (s.f.). *pseint.sourceforge.net*. Obtenido de http://pseint.sourceforge.net/index.php?page=pseudocodigo.php

Rancel, M. R. (2015). *aprendeaprogramar.com*. Obtenido de http://www.aprenderaprogramar.com/index.php?option=com\_content&view=article&id=934:funciones-para-cadenas-en-c-longitud-sizeof-stringh-y-strcpy-strlen-strcat-strcmp-ejemplos-cu00535f&catid=82:curso-basico-programacion-lenguaje-c-desde-cero&Itemid=210

ronnyml. (04 de 07 de 2009). *wordpress.com*. Obtenido de https://ronnyml.wordpress.com/2009/07/04/vectores-matrices-y-punteros-en-c/