

Universidad Nacional Autónoma de México

Facultad de Ingeniería



Asignatura: Estructura de Datos y Algoritmos I

Actividad 1: Repaso de lo que aprendí en la materia de Fundamentos de Programación

Alumno: Miranda González José Francisco

Fecha: Miércoles 24 de Febrero del 2021



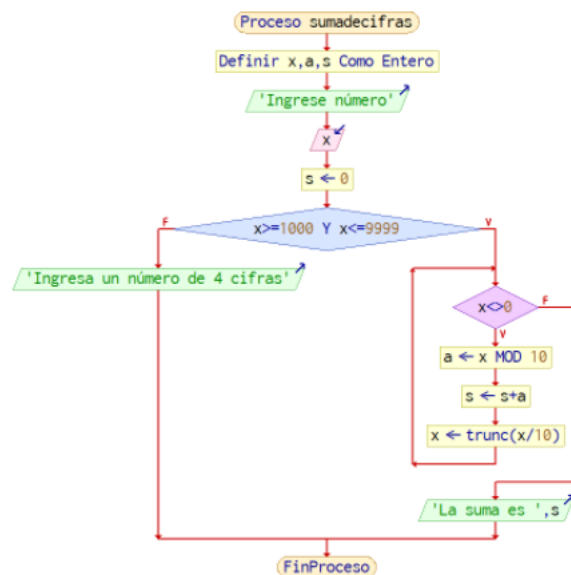
Repaso de lo que aprendí en la materia de Fundamentos de Programación

Aprendimos a utilizar herramientas de software que se ofrecen en Internet que permiten realizar actividades y trabajos académicos, tales como manejo de repositorios de almacenamiento y buscadores con funciones avanzadas. En este caso fuimos capaces de manejar el buscador de internet Google para encontrar información específica de manera correcta.

Conocimos la importancia de los sistemas operativos GNU/Linux y el manejo de su terminal, con comandos básicos pudimos explorar el sistema de archivos al igual que poder manejar de manera correcta a estos.

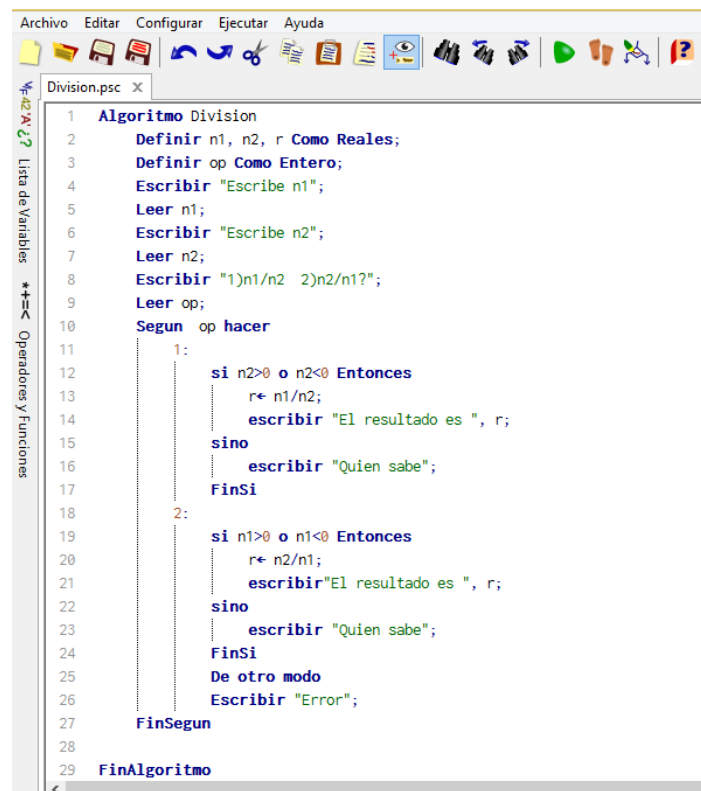
Posteriormente, realizamos la elaboración de ejercicios en los cuales a partir de un enunciado identificamos el conjunto de entradas y el de salidas con los cuales teníamos que elaborar un algoritmo que fuera capaz de resolver el problema. Continuando con estos algoritmos tuvimos que representarlos de una manera grafica mediante diagramas de flujo, estos, como su nombre dice muestran gráficamente el flujo de acciones a seguir para cumplir con una tarea específica. Durante este proceso aprendimos sus formas y gran importancia, ya que a partir de estos es posible codificar un programa en algún lenguaje de programación.

2. Determinar la suma de las cifras de un número entero positivo de 4 cifras. (ejem. 1254=12). Sólo números del 1000 al 9999.



Realizamos la elaboración de pseudocódigos que pudieran representar la solución algorítmica de un problema, revisamos su sintaxis en donde aprendimos que un pseudocódigo está limitado por las etiquetas de INICIO y FIN. En donde dentro de estas etiquetas se deben escribir todas las instrucciones del programa. Para indicar

lectura de datos se utiliza la etiqueta LEER. Para indicar escritura de datos se utiliza la etiqueta ESCRIBIR. Al igual que la declaración y tipos de variables entre otras cosas.

A screenshot of a C programming editor window titled 'Division.psc'. The window has a menu bar with 'Archivo', 'Editar', 'Configurar', 'Ejecutar', and 'Ayuda'. Below the menu is a toolbar with various icons. On the left side, there is a vertical toolbar with icons for 'Lista de Variables' and 'Operadores y Funciones'. The main area of the window contains C code for a division algorithm. The code is as follows:

```
1  Algoritmo Division
2  Definir n1, n2, r Como Reales;
3  Definir op Como Entero;
4  Escribir "Escribe n1";
5  Leer n1;
6  Escribir "Escribe n2";
7  Leer n2;
8  Escribir "1)n1/n2  2)n2/n1?";
9  Leer op;
10 Segun op hacer
11     1:
12         si n2>0 o n2<0 Entonces
13             r← n1/n2;
14             escribir "El resultado es ", r;
15         sino
16             escribir "Quien sabe";
17         FinSi
18     2:
19         si n1>0 o n1<0 Entonces
20             r← n2/n1;
21             escribir "El resultado es ", r;
22         sino
23             escribir "Quien sabe";
24         FinSi
25     De otro modo
26     Escribir "Error";
27 FinSegun
28
29 FinAlgoritmo
```

Continuando con el curso conocimos y usamos los ambientes y herramientas para el desarrollo y ejecución de programas en Lenguaje C, como editores y compiladores en diversos sistemas operativos.

A la hora de programar en C revisamos conceptos muy importantes que nos ayudaron con nuestros códigos.

Aprendimos a colocar los dos tipos de comentarios dentro de nuestros programas, los cuales no afectan en la ejecución del código.

El comentario por línea inicia cuando se insertan los símbolos `/**` y termina con el salto de línea (hasta donde termine el renglón). El comentario por bloque inicia cuando se insertan los símbolos `/*` y termina cuando se encuentran los símbolos `*/`.

Un punto muy importante que revisamos es que al momento de iniciar el programa se deben agregar todas las bibliotecas que se van a utilizar en el mismo, es decir, funciones externas necesarias para ejecutar el programa. En lenguaje C la biblioteca estándar de entrada y salida está definida en `'stdio.h'` (standard in out) y

provee, entre otras, funciones para lectura y escritura de datos que se verán a continuación.

Con respecto a las variables, pudimos declarar el tipo de dato que esta puede contener (Caracteres, Enteros, Flotantes, Dobles) al igual que el identificador (nombre o etiqueta) con el que se va a manejar el valor.

El especificador de dato se usa para guardar o imprimir el valor de una variable.

Entero: %d, %i, %ld, %li, %o, %x , Flotante: %f, %lf, %e, %g, Carácter: %c, %d, %i, %o, %x, Cadena de caracteres: %s

Aprendimos a manejar las diferentes funciones como printf, la cual es una función para imprimir con formato, es decir, se tiene que especificar entre comillas el tipo de dato que se desea imprimir. También supimos manejar a scanf, la cual es una función que sirve para leer datos de la entrada estándar (teclado), para ello únicamente se especifica el tipo de dato que se desea leer entre comillas y en qué variable se quiere almacenar. Al nombre de la variable le antecede un ampersand (&), esto indica que el dato recibido se guardará en la localidad de memoria asignada a esa variable.

Algo también muy importante que nos sirve en todos nuestros códigos son las teclas de escape que se revisaron en el semestre pasado, pues nos ayudan a imprimir con formato nuestro código.

\a: carácter de alarma, \b: retroceso, \f: avance de hoja, \n: salto de línea, \r: regreso de carro, \t : tabulador horizontal, \v: tabulador vertical, '\0': carácter nulo.

Aprendimos a utilizar los diferentes tipos de operadores, tales como:

operadores aritméticos: +, -, *, /, %

operadores lógicos a nivel de bits: >>, <<, &, |, ~

operadores de relación permiten comparar elementos numéricos, alfanuméricos, constantes o variables: ==, !=, <, >, <=, >=

operadores lógicos permiten formular condiciones complejas a partir de condiciones simples: !, &&, ||

Una vez que fuimos capaces de comprender todos los conceptos pasados pasamos al tema de estructuras de selección, en el cual pudimos utilizar estructuras de selección if, if-else, switch y ternaria (o condicional) para resolver problemas básicos,

Las estructuras de selección (o condicionales) nos permiten realizar una u otra acción con base en una expresión lógica.

Aprendimos a utilizar sus estructuras y a realizar diferentes códigos.

```

1  //ESTRUCTURAS SELECTIVAS
2
3  #include <stdio.h>
4
5  int main(){
6      int x;
7      x=5;
8
9      if(x==4){
10         //DIFERENTES OPERADORES DE COMPARACION:
11         // "==" , "<" , "<=" , ">" , ">=" , "!=" .
12         printf("\nEL VALOR DE X ES 5.\n");
13     }
14     else{
15         printf("\nEL VALOR DE X NO ES 5.\n");
16     }
17     system("pause");
18     return 0;
19 }
20

```

Aprendimos a utilizar las 3 estructuras de repetición que existen en el lenguaje C: while, do-while y for.

Las cuales nos permiten ejecutar un conjunto de instrucciones de manera repetida (o cíclica) mientras que la expresión lógica a evaluar se cumpla (sea verdadera).

```

8  #include <stdio.h>
9
10 int main(){
11     int i; //CONTADOR.
12
13     //EJEMPLO CON WHILE.
14     //EN WHILE SI LA CONDICION ES FALSA NO SE EJECUTA NADA.
15
16     i=10;
17
18     while(i<15){
19         printf("\nESTO SE VA A EJECUTAR 5 VECES.\n");
20         i++;
21     }
22     printf("\n\n");
23

```

Pudimos ser capaces de aprender las técnicas básicas de depuración en C, para revisar de manera precisa el flujo de ejecución de un programa y el valor de las variables.

Depurar es muy importante pues nos ayuda a optimizar el programa, ver si este tiene algún fallo o error de ejecución.

Revisamos el tema de los arreglos unidimensionales y multidimensionales.

Vimos que un arreglo es un conjunto de datos contiguos del mismo tipo con un tamaño fijo definido al momento de crearse. A cada elemento (dato) del arreglo se le asocia una posición particular, el cual se requiere indicar para acceder a un elemento en específico. Esto se logra a través del uso de índices. Los arreglos se utilizan para hacer más eficiente el código de un programa.

En cuestión de los unidimensionales, aprendimos la forma en la que estos se almacenan y la sintaxis para definirlo: `tipoDeDato nombre[tamaño]`.

También, dentro del mismo tema aprendimos la utilidad de los apuntadores. Estos son una variable que contiene la dirección de una variable, es decir, hace referencia a la localidad de memoria de otra variable. Debido a que los apuntadores trabajan directamente con la memoria, a través de ellos se accede con rapidez a un dato.

Aprendimos la sintaxis para declarar un apuntador y para asignarle la dirección de memoria de otra variable es de la siguiente manera:

```
TipoDeDato *apuntador, variable;
```

```
apuntador = &variable;
```

Revisamos que la declaración de una variable apuntador inicia con el carácter `*`. Cuando a una variable le antecede un ampersand, lo que se hace es acceder a la dirección de memoria de la misma.

Vimos que los apuntadores solo pueden apuntar a direcciones de memoria del mismo tipo de dato con el que fueron declarados; para acceder al contenido de dicha dirección, a la variable apuntador se le antepone `*`.

En cuestión de los arreglos multidimensionales aprendimos la sintaxis para definirlos: `tipoDato nombre [tamaño][tamaño]...[tamaño]`;

Casi finalizando el curso aprendimos el tema de Funciones. C permite tener dentro de un archivo fuente varias funciones, esto con el fin de dividir las tareas y que sea más fácil la depuración, la mejora y el entendimiento del código.

Revisamos la sintaxis básica para definir a una función:

```
valorRetorno nombre (parámetros){
```

```
// bloque de código de la función
```

```
}
```

Aprendimos que una función puede recibir parámetros de entrada, los cuales son datos de entrada con los que trabajará la función, dichos parámetros se deben definir dentro de los paréntesis de la función, separados por comas e indicando su tipo de dato.

También dentro de una función revisamos que el valor de retorno de una función indica el tipo de dato que va a regresar la función al terminar el bloque de código de la misma. El valor de retorno puede ser cualquiera de los tipos de datos vistos hasta el momento (entero, real, carácter o arreglo), aunque también se puede regresar el elemento vacío (void).

La firma de una función está compuesta por tres elementos: el nombre de la función, los parámetros que recibe la función y el valor de retorno de la función; finaliza con punto y coma (;).

```
5      #include <stdio.h>
6
7      int maximo();
8
9      int main(){
10         int x,y,max;
11
12         printf("\nVALOR DE X: ");
13         scanf("%i",&x);
14         printf("\nVALOR DE Y: ");
15         scanf("%i",&y);
16
17         max=maximo(x,y); //PARA QUE SE EJECUTE EN MAXIMO PONEMOS LAS VARIABLES DENTRO DEL PARANTESIS.
18                           //MAX SE CONVIERTE EN AUX max=aux
19                           //xxxxxxxx(x,y); =aux POR LA FUNCION DEL FUNAL.
20         printf("\nEL NUMERO MAYOR ES: %i",max);
21         printf("\n\n");
22
23         system("pause");
24         return 0;
25     }
26
27     int maximo(int a, int b){ //a=x ; b=y. CUALQUIER VARIABLE PUEDE SER IGUAL A LAS DEFINIDAS EN MAIN.
28         int aux; //DECLARAMOS AUX, QUE ES EN DONDE SE GUARDARA EL VALOR MAYOR.
29         if(a>b){
30             aux=a;
31         }
32         else{
33             aux=b;
34         }
35         return aux; //SE DEVUELVE EL VALOR GUARDADO EN AUX.
36     }
37 }
```

Como tema final aprendimos a elaborar programas que requerían el uso de archivos de texto plano en la resolución de problemas, entendimos a los archivos como un elemento de almacenamiento secundario.

Aprendimos a emplear las funciones para crear, leer, escribir y sobrescribir archivos de texto plano.

Revisamos la importancia de un apuntador a archivo, este es un hilo común que unifica el sistema de Entrada/Salida (E/S) con un buffer donde se transportan los datos.

Los apuntadores a un archivo se manejan en lenguaje C como variables apuntador de tipo FILE que se define en la cabecera stdio.h. La sintaxis para obtener una variable apuntador de archivo es la siguiente: FILE *F;

Aprendimos a como abrir y cerrar un archivo. La función fopen() abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su estructura es la siguiente: *FILE fopen(char *nombre_archivo, char *modo);

La función fclose() cierra una secuencia que fue abierta mediante una llamada a fopen(). Escribe la información que se encuentre en el buffer al disco y realiza un cierre formal del archivo a nivel del sistema operativo: int fclose(FILE *apArch);

Donde apArch es el apuntador al archivo devuelto por la llamada a fopen().

No menos importante revisamos las funciones fgets() y fputs(). Pueden leer y escribir, respectivamente, cadenas sobre los archivos. Las firmas de estas funciones son, respectivamente:

```
char *fgets(char *buffer, int tamaño, FILE *apArch);
```

```
char *fputs(char *buffer, FILE *apArch);
```

La función fputs() permite escribir una cadena en un archivo específico. La función fgets() permite leer una cadena desde el archivo especificado. Esta función lee un renglón a la vez.

Vimos que las funciones fprintf() y fscanf() se comportan exactamente como printf() (imprimir) y scanf() (leer), excepto que operan sobre archivo. Sus estructuras son:

```
int fprintf(FILE *apArch, char *formato, ...);
```

Donde apArch es un apuntador al archivo devuelto por una llamada a la función fopen(), es decir, fprintf() y fscanf() dirigen sus operaciones de E/S al archivo al que apunta apArch.

Por ultimo aprendimos a utilizar las funciones fread y fwrite.

fread permite leer uno o varios elementos de la misma longitud a partir de una dirección de memoria determinada (apuntador).

```
int fread(void *ap, size_t tam, size_t nelem, FILE *archivo)
```

fwrite permite escribir hacia un archivo uno o varios elementos de la misma longitud almacenados a partir de una dirección de memoria determinada.

```
int fwrite(void *ap, size_t tam, size_t nelem, FILE *archivo)
```

```
1  #include <stdio.h>
2
3  int main() {
4      FILE *fp;
5      char letra;
6      fp=fopen("archivoCaracter.txt", "r");
7      if(fp==NULL) //la condicion verifica que exista contenido (archivo) en el apuntador
8      {
9          printf("Error al abrir el archivo para leer\n");
10     }
11     else
12     {
13         printf("Encontre tu archivo!!!\n");
14         letra=fgetc(fp);
15         printf("\n El caracter leido es %c\n", letra);
16     }
17     fclose(fp);
18
19
20
```


A grandes rasgos explico en este trabajo todos los temas aprendidos en el curso de Fundamentos de Programación, al igual que la inclusión de algunos conceptos y ejemplos que a mi parecer ayudan demasiado a comprender cada uno de los temas.

Algunos de estos temas no los comprendí del todo, pues mi curso fue un tanto regular. Actualmente sigo estudiando los temas desde el inicio para tratar de comprenderlos de mejor manera.

