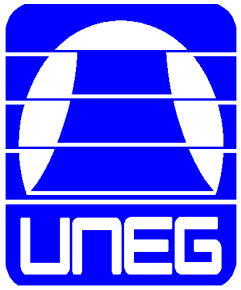

Unidad Curricular:

Técnicas de Programación I

II Semestre

Ing. Dubraska Roca

Correo: droca754@gmail.com

Teléfono: 04148965941

Whatsapp

Temario

- Del seudocódigo al lenguaje de programación C.
- Estructuras de Selección, Control y Repetición
- Funciones y Procedimientos
- Estructuras de Datos Lineales
- Métodos de Ordenamiento y Búsqueda (Algoritmos)
- Manejo de Archivos

Bibliografía

- **Aho, Alfred; Hopcroft, John y Ullman, Jeffrey.** (1974) The Design and Analysis of Computer Algorithms. Addison-Wesley. ISBN 0-201-000296.

- **Aho, Alfred; Hopcroft, John y Ullman, Jeffrey.** (1983) Data Structures and Algorithms. Addison-Wesley. ISBN: 0-201-000237.

- **Brassard, Gilles y Bratley Paul.** (2000) Fundamentos de Algoritmia. Prentice Hall; 1st. Edition. ISBN: 978-8489660007.

- **Carrillo, Antonio y Valdivia Joaquin** (2006). Abstracción y estructuras de datos en C++. Delta Publicaciones. ISBN: 9788496477261.

- **Deitel, Harvey y Deitel, Paul (2003).** Cómo programar en C++. Pearson Educación. ISBN: 970-2602548

Bibliografía

- **Joyanes, Luis** (1998). Fundamentos de Programación – 2b. McGraw – Hill Interamericana. ISBN: 978-8448106034.

- **Joyanes, Luis** (2007). Estructura de Datos en C++. McGraw – Hill Interamericana. ISBN: 978-8448156459.

- **Knuth, Donald.** (1998) The Art of Computer Programming. Volume 1-3. Addison-Wesley. ISBN: 978-0201485417.

- **Stroustrup, Bjarne** (1998). El lenguaje de programación C++, Addison Wesley. ISBN: 84-78290192.

Tópicos de uso en un Lenguaje de programación C

- **Estructura de un programa en C**
- **Variables. Tipos. Ámbito. Conversión. Modificadores**
- **Operadores Lógicos, Aritméticos, Relacionales**
- **Estructuras Condicionales**
- **Estructuras de Control**
- **Concepto de Función**

Estructura de un Programa

Un programa incluye secciones determinadas y un orden para las mismas.

En la siguiente forma:

Archivos de cabecera

Declaración de constantes y macros

Declaración de Variables Globales

Declaración de prototipos de Funciones

Desarrollo de otras funciones.

Desarrollo de la función principal

Estructura de un Programa

#include <stdio.h> **LIBRERÍA** para utilizar printf o scanf

#define MAX 20 para def. el valor de una constante

int a,b; Variables Globales
float f, k;

Desarrollo o Cuerpo de Funciones

int main() { } Función Principal

Estructura de una Función

Se pueden Desarrollar los cuerpos de las funciones a utilizar antes de su llamada en el main principal.

```
int funcionuno (float x)
{
//Variables locales a funcionuno
    int i, cant;

//Retorno de la funcion
    return (i*cant);
}
```

Estructura de un Programa

Declaración de prototipos de Funciones

```
void CargaGente ( );
int ValidaEdad();
```

Declaración de Variables Globales

```
int flag;
```

Desarrollo de la función principal

```
int main()
{
        int i, cant;
        printf("Bienvenido a Técnicas 1");
}
```

Variables

Tipos. Ámbito. Conversión. Modificadores.

- **Según el lugar donde se declaren las variables tendrán un ámbito**

- **Según el ámbito pueden ser utilizadas desde cualquier parte del programa o únicamente en la función donde han sido declaradas.**

Variables

- Locales: Cuando están declaradas dentro de la función
- Globales: Son conocidas a lo largo de todo el programa y se pueden utilizar desde cualquier parte del código
- De Registro: Son aquellas que se guardan en registros internos del microprocesador el acceso a ellas es más rápido y directo.
- Estáticas: Son aquellas variables locales donde se requiere mantener el valor entre una llamada y otra de una función. Se añade la palabra static delante del tipo.
- Externas: Se aplica a las variables globales cuyo valor se requiere cuando la compilación es por separado en pequeños módulos. Se añade la palabra extern delante del tipo.

3. VARIABLES Y CONSTANTES

Unidad básica de almacenamiento, la creación es la combinación de un identificador, un tipo y un ámbito. Todas las variables en C tienen que ser declaradas antes de ser usadas. El lugar donde se declaran las variables puede ser dentro o en la definición de una función, fuera de todas las funciones.

TIPOS	*RANGO*	*TAMAÑO*	*DESCRIPCIÓN*
char	-128 a 127	**1**	Para una letra o un dígito.
unsigned char	0 a 255	**1**	Letra o número positivo.
int	-32.768 a 32.767	**2**	Para números enteros.
unsigned int	0 a 65.535	**2**	Para números enteros.
long int	±2.147.483.647	**4**	Para números enteros
unsigned long int	0 a 4.294.967.295	**4**	Para números enteros
float	3.4E-38 decimales(6)	**6**	Para números con decimales
double	1.7E-308 decimales(10)	**8**	Para números con decimales
long double	3.4E-4932 decimales(10)	**10**	Para números con decimales

El nombre de las variables conocido como *identificadores* debe cumplir las siguientes normas. La longitud puede ir de un carácter a 31. El primero de ellos debe ser siempre una letra. No puede contener espacios en blanco, ni acentos y caracteres gramaticales. Hay que tener en cuenta que el compilador distingue entre mayúsculas y minúsculas.

SINTAXIS:

```
tipo nombre=valor_numerico;

tipo nombre='letra';

tipo nombre[tamaño]="cadena de letras",

tipo nombre=valor_entero.valor_decimal;
```

CONVERSION (*casting*): Las conversiones automáticas pueden ser controladas por el programador. Bastará con anteponer y encerrar entre parentesis, el tipo al que se desea convertir. Este tipo de conversiones solo es temporal y la variable a convertir mantiene su valor.

ARITMÉTICOS: Pueden aplicarse a todo tipo de expresiones. Son utilizados para realizar operaciones matemáticas sencillas, aunque uniéndolos se puede realizar cualquier tipo de operaciones. En la siguiente tabla se muestran todos los operadores aritméticos.

OPERADOR	*DESCRIPCIÓN*	*ORDEN*
-	Resta.	3
+	Suma	3
*	Multiplica	2
/	Divide	2
%	Resto de una división	2
-	signo (monario).	2
--	Decremento en 1.	1
++	Incrementa en 1.	1

LÓGICOS Y RELACIONALES: Los operadores relacionales hacen referencia a la relación entre unos valores y otros Los lógicos se refiere a la forma en que esas relaciones pueden conectarse entre si. Los veremos a la par por la estrecha relación en la que trabajan.

OPERADORES RELACIONALES		
OPERADOR	*DESCRIPCIÓN*	*ORDEN*
<	Menor que.	**5**
>	Mayor que.	**5**
<=	Menor o igual.	**5**
>=	Mayor o igual	**5**
==	Igual	**6**
!=	Distinto	**6**

OPERADORES LÓGICOS		
OPERADOR	*DESCRIPCIÓN*	*ORDEN*
&&	Y (AND)	**10**
||	O (OR)	**11**
!	NO (NOT)	**1**

A NIVEL DE BITS: Estos operadores son la herramienta más potente, pueden manipular internamente las variables, es decir bit a bit. Este tipo de operadores solo se pueden aplicar a las variables de tipo char, short, int y long. Para manejar los bits debemos conocer perfectamente el tamaño de las variables.

OPERADOR	*DESCRIPCIÓN*	*ORDEN*
&	Y (AND) bit a bit.	**7**
|	O (OR) Inclusiva.	**9**
^	O (OR) Exclusiva.	**8**
<<	Desplazamiento a la izquierda.	**4**
>>	Desplazamiento a la derecha.	**4**
~	Intercambia 0 por 1 y viceversa.	**1**

En las siguientes tablas se muestran todos los identificadores de formato y las constantes de carácter las que se utilizan para realizar operaciones automáticamente sin que el usuario tenga que intervenir en esas operaciones.

IDENTIFICADORES DE FORMATO	
IDENTIFICADOR	*DESCRIPCION*
%c	Carácter
%d	Entero
%e	N. Científica
%E	N. Científica
%f	Coma flotante

CONSTANTES DE CARÁCTER	
CONSTANTE	*DESCRIPCION*
\n	Salto de línea
\f	Salto de página
\r	Retorno de carro
\t	Tabulación
\b	Retroceso

%o	Octal
%s	Cadena
%u	Sin signo
%x	Hexadecimal
%X	Hexadecimal
%p	Puntero
%ld	Entero Largo
%h	Short
%%	signo %

\’	Comilla simple
\”	Comillas
\\	Barra invertida
\?	Interrogación

Existen especificadores de formato asociados a los identificadores que alteran su significado ligeramente. Se puede especificar la longitud mínima, el número de decimales y la alineación. Estos modificadores se sitúan entre el signo de porcentaje y el identificador.

```
% modificador identificador
```

El *especificador de longitud mínima* hace que un dato se rellene con espacios en blanco para asegurar que este alcanza una cierta longitud mínima. Si se quiere rellenar con ceros o espacios hay que añadir un cero delante antes del especificador de longitud.

```
printf("%f ",numero); //salida normal.

printf("%10f ",numero); //salida con 10 espacios.

printf("%010f ",numero); //salida con los espacios poniendo 0.
```

El *especificador de precisión* sigue al de longitud mínima(si existe). Consiste en un nulo y un valor entero. Según el dato al que se aplica su función varia. Si se aplica a datos en coma flotante determina el número de posiciones decimales. Si es a una cadena determina la longitud máxima del campo. Si se trata de un valor entero determina el número mínimo de dígitos.

```
printf("%10.4f ",numero); //salida con 10 espacios con 4 decimales.

printf("%10.15s",cadena); //salida con 10 caracteres dejando 15 espacios.

printf("%4.4d",numero); //salida de 4 dígitos mínimo.
```

El *especificador de ajuste* fuerza la salida para que se ajuste a la izquierda, por defecto siempre lo muestra a la derecha. Se consigue añadiendo después del porcentaje un signo menos.

```
printf("%8d",numero); // salida ajustada a la derecha.

printf("%-8d",numero); //salida ajustada a la izquierda.
```

SCANF: Es la rutina de entrada por consola. Puede leer todos los tipos de datos incorporados y convierte los números automáticamente al formato incorporado. En caso de leer una cadena lee hasta que encuentra un carácter de espacio en blanco. El formato general:

```
scanf("identificador",&variable_numerica o char);

scanf("identificador",variable_cadena);
```

La función `scanf()` también utiliza modificadores de formato, uno especifica el número máximo de caracteres de entrada y eliminadores de entrada.. Para especificar el número máximo solo hay que poner un entero después del signo de porcentaje. Si se desea eliminar entradas hay que añadir %*c en la posición donde se desee eliminar la entrada.

```
scanf("%10s",cadena);

scanf("%d%*c%d",&x,&y);
```

En muchas ocasiones se combinaran varias funciones para pedir datos y eso puede traer problemas para el buffer de teclado, es decir que asigne valores que no queramos a nuestras variables. Para evitar esto hay dos funciones que se utilizan para limpiar el buffer de teclado.

```
fflush(stdin);

fflushall();
```

OTRAS FUNCIONES E/S: El archivo de cabecera de todas estas funciones es STDIO.H. Todas estas funciones van a ser utilizadas para leer caracteres o cadenas de caracteres. Todas ellas tienen asociadas una función de salida por consola.

FUNCIONES	*DESCRIPCIÓN*
var_char=getchar();	Lee un carácter de teclado, espera un salto de carro.
var_char=getche();	Lee un carácter con eco, no espera salto de carro.
var_char=getch();	Lee un carácter sin eco, no espera salto de carro.
gets(var_cadena);	Lee una cadena del teclado.
putchar(var_char);	Muestra un carácter en pantalla.
puts(variables);	Muestra una cadena en pantalla.

IF-ELSE: La ejecución atraviese un conjunto de estados bolean que determinan que se ejecuten distintos fragmentos de código.

```
if (expresion-booleana)
     Sentencia1;
else
     Sentencia2;
```

```
if (expresion-booleana)
{
     Sentencia1;
     sentencia2;
}
else
     Sentencia3;
```

La cláusula `else` es opcional, la expresión puede ser de cualquier tipo y más de una (siempre que se unan mediante operadores lógicos).Otra opción posible es la utilización de if anidados, es decir unos dentro de otros compartiendo la cláusula else.

Un operador muy relacionado con la sentencia if es **?**. Es un operador ternario que puede sustituir al if. El modo de trabajo es el siguiente, evalúa la primera expresión y si es cierta toma el valor de la segunda expresión y se le pasa a la variable. Si es falsa, toma el valor de la tercera y la pasa a la variable

```
varible=condicion  ?  expresion2:expresion3;
```

EJEMPLO:

```
#include <stdio.h>

void main(void)
{
      int peso;
      clrscr();

      gotoxy(5,3);printf("Introducir edad: ");
      gotoxy(22,3);scanf("%d",&peso);

      if(peso<500)
      {
            gotoxy(5,5);
            printf("No es ni media Tn");
      }
      else
      {
            gotoxy(5,5);
            pritnf("Es más de media Tn");
      }
      getch();
}
```

Para probar la utilización de un if anidado realizar un programa que pida una edad. Si la edad es igual o menor a 10 poner el mensaje NIÑO, si la edad es mayor a 65 poner el mensaje JUBILADO, y si la edad es mayor a 10 y menor o igual 65 poner el mensaje ADULTO.

SWITCH: Realiza distintas operaciones en base al valor de una única variable o expresión. Es una sentencia muy similar a if-else, pero esta es mucho más cómoda y fácil de comprender. Si los valores con los que se compara son números se ponen directamente pero si es un carácter se debe encerrar entre comillas simples.

```
switch (expresión){
     case valor1:
          sentencia;
          break;
     case valor2:
          sentencia;
          break;
     case valor3:
          sentencia;
          break;
     case valorN:
          sentencia;
          break;
     default:
}
```

El valor de la expresión se compara con cada uno de los literales de la sentencia `case` si coincide alguno se ejecuta el código que le sigue, si ninguno coincide se realiza la sentencia `default` (opcional), si no hay sentencia `default` no se ejecuta nada.

}

WHILE: Ejecuta repetidamente el mismo bloque de código hasta que se cumpla una condición de terminación. Hay cuatro partes en cualquier bucle. *Inicialización, cuerpo, iteración* y *terminación*.

```
[inicialización;]
while(terminación){
     cuerpo;
     [iteración;]
}
```

DO-WHILE: Es lo mismo que en el caso anterior pero aquí como mínimo siempre se ejecutara el cuerpo una vez, en el caso anterior es posible que no se ejecute ni una sola vez.

```
[inicialización;]
do{
     cuerpo;
     [iteración;]
}while(terminación);
```

EJEMPLO: Este programa va sumando números y el resultado lo suma a otro, así hasta 100.000.

FOR: Realiza las mismas operaciones que en los casos anteriores pero la sintaxis es una forma compacta. Normalmente la condición para terminar es de tipo numérico. La iteración puede ser cualquier expresión matemática valida. Si de los 3 términos que necesita no se pone ninguno se convierte en un bucle infinito.

```
for (inicio;fin;iteración)
     sentencia1;
```

```
for (inicio;fin;iteración)
{
     sentencia1;
     sentencia2;
}
```

EJEMPLO: Este programa muestra números del 1 al 100. Utilizando un bucle de tipo FOR.

```
#include<stdio.h>

void main(void)
{
     int n1=0;
     for (n1=1;n1<=100;n1++)
          printf("%d\n",n1);
     getch();
}
```

EJEMPLO: Este programa muestra números del 1 al 100. Utilizando un bucle de tipo FOR.

```
#include<stdio.h>

void main(void)
{
      int n1=0;
      for (n1=1;n1<=100;n1++)
           printf("%d\n",n1);
      getch();
}
```

La sentencia **continue** lo que hace es ignorar las sentencias que tiene el bucle y saltar directa condición para ver si sigue siendo verdadera, si es así sigue dentro del bucle, en caso contrar directamente de el.

Gracias por su atención….