UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS Y ALGORITMOS I (1227)

Profesor: ***M.I. Marco Antonio Martínez Quintana***

Semestre 2021-2

Actividad Asíncrona #1 Lunes 7 de Junio

Nombre del alumno: Cadena Luna Iván Adrián

Grupo: 15

Fecha: (11/06/2021)

- Repase sus conocimientos de Lenguaje C adquiridos en Fundamentos de Programación y el primer tema de arreglos en C, de Estructura de Datos y Algoritmos I.

Estructura de un programa en C:

```
/*inclusión de bibliotecas, definición de constantes y */
/*macros, definición de prototipos de funciones*/

main() {
/*cuerpo de la función main
/*En esta función escribiremos el código principal de nuestro
programa*/
}

/*Definición de otras funciones*/
```

Palabras reservadas:

Las palabras reservadas son identificadores predefinidos que tienen significados especiales y no pueden usarse como identificadores creados por el usuario en los programas: *auto const double float int short struct unsigned break continue else for long signed switch void case default enum goto register sizeof typedef volatile char do extern if return static union while*

Tipo	Significado	Tamaño en bytes
`char`	caracter	1
`int`	entero	2 – 4
`short`	entero corto	2
`long`	entero largo	4
`unsigned char`	caracter sin signo	1
`unsigned`	entero sin signo	2 – 4
`unsigned char`	entero corto sin signo	2
`unsigned long`	entero largo sin signo	4
`float`	coma flotante, real	4
`double`	coma flotante largo	8

- **Variables**

  Identificador utilizado para representar un cierto tipo de

  información.Cada variable es de un tipo de datos determinado.

  Una variable puede almacenar diferentes valores en distintas partes delprograma.

  ! Debe comenzar con una letra o el carácter guión bajo (_).

  ! El resto puede contener letras, números o _

  Ejemplos de variables válidas: numero, _color, identificador_1

  Una declaración asocia un tipo de datos determinado a una o más variables.

  El formato de una declaración es:

  tipo_de_datos var1, var2, ..., varN;*Ejemplos:*

  • ***int*** *a, b, c;*

  • ***float*** *numero_1, numero_2;*

  • ***char*** *letra;*

  • ***unsigned long*** *entero;*

- **Expresiones y sentencias**

  Una expresión representa una unidad de datos simple, tal como un númeroo carácter.

  - **Operadores:**
    - + Suma
    - - Resta
    - * Multiplicación
    - / División
    - Mod Módulo (residuo de la división entera)

- **Punto y coma:**

  Indica que la línea de instrucción termina por ejemplo al declararvariables.

  Ejemplo:

```
int a,s,d,f; //indica terminación de variables
clrscr; //función limpiar pantalla
printf("hola"); //imprimir hola
```

- **Funciones**
  - ***printf***

    Permite imprimir información por la salida estándar (monitor).

    Formato: printf(formato acompañado o no de cadenas,

    argumentos);*Ejemplo:*

```
printf("Hola mundo\n");
printf("El numero 28 es %d\n", 28);
```

  - ***scanf***

    Permite leer datos del usuario: la función devuelve el número dedatos que se han leído correctamente.

    Formato: scanf(formato, argumentos);

    El símbolo & indica que el valor leído se almacenará en la direcciónde memoria de la variable correspondiente

    Ejemplo:

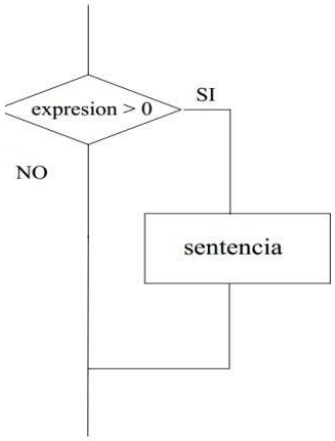
```
scanf("%f", &numero);
scanf("%c", &letra);
scanf("%f %d %c", &real, &entero, &letra);
scanf("%ld", &entero_largo);
```

- **Secuencias de control**

  Una secuencia de control es aquella por la cual los enunciados en un programa son ejecutados uno después del otro, en el orden en que aparecen escritos.

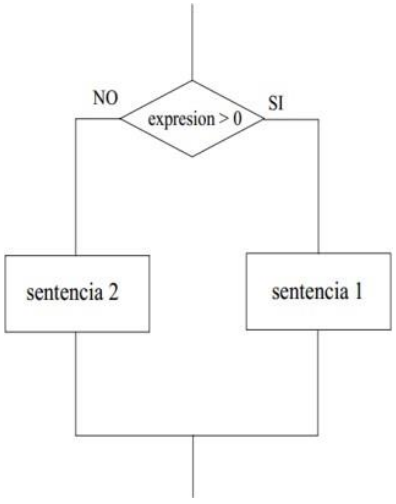
  - **if**

    Se utiliza para elegir entre cursos alternativos de acción. Es decir, si la condición definida por la expresión1 es verdadera, entonces se ejecuta la o las sentencias siguientes. Si la condición resulta falsa, seignora la o las sentencias siguientes, y se ejecuta el siguiente fragmento de código en su orden.



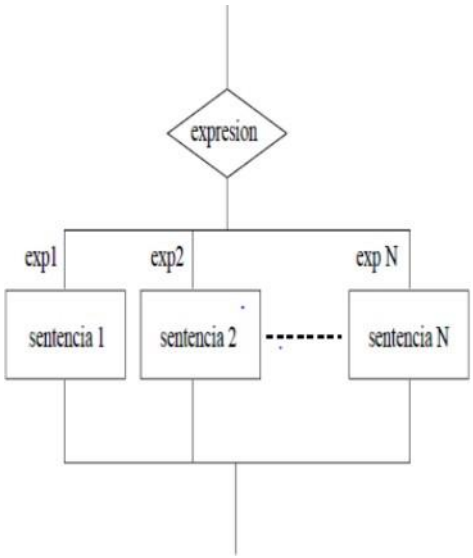
  - **if-else**

    Se utiliza de igual manera para elegir entre cursos alternativos de acción dependiendo de la condición definida por la expresión1; pero en este caso, si la condición resulta falsa, se pasa a ejecutar el siguiente fragmento de código indicado por else.



- **switch**

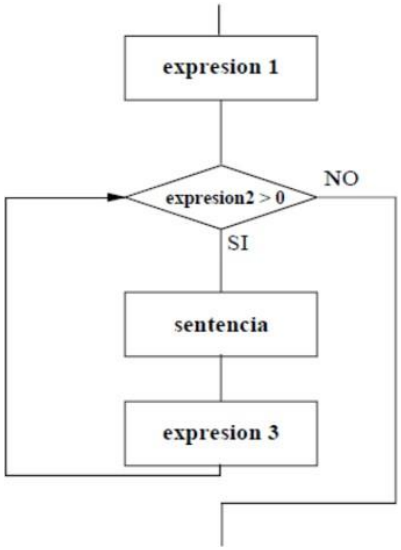
  En los casos que se utiliza esta secuencia de control, las variables o expresiones se probarán por separado contra cada uno de los valores constantes enteros que puede asumir, y se tomarán diferentes acciones.



- **for/contador**

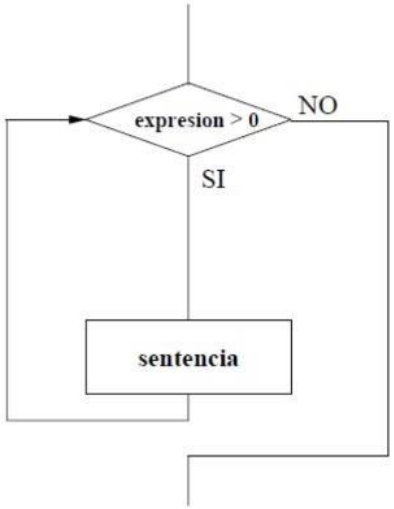
  La repetición controlada por contador se denomina a veces repetición definida, porque con anticipación **se sabe** con exactitud cuántas veces se ejecutará el ciclo.

  Se utiliza una variable de control para contar el número de repeticiones y es incrementada cada vez que se ejecuta el grupo de instrucciones; hasta que se ha ejecutado el número correcto de repeticiones, se termina el ciclo y la computadora continúa ejecutando el enunciado siguiente al de la estructura de repetición.



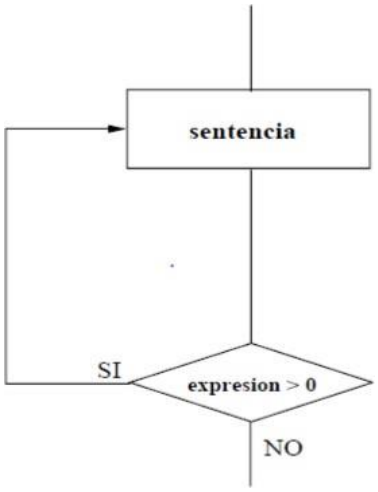
- **while/centinela**

  La repetición controlada por centinela a veces se denomina repetición indefinida, porque **no se sabe** con anticipación cuantas veces el ciclo se ejecutará.



- **do-while**

  En este caso, las sentencias se ejecutarán mientras el valor de la expresión sea verdadera, y al menos una vez después se evalúa la expresión. Normalmente las sentencias incluyen algún elemento que altera la expresión **proporcionando el fin de las repeticiones**.



- **Arreglos**

Un arreglo es una colección ordenada de elementos de un mismo tipo. Ordenada significa que cada elemento tiene una ubicación determinada dentro del arreglo y debemos conocerla para poder acceder a él.

Se definen de la siguiente manera:

```
<tipo>nombre variable [longitud];
```

Se asignan de la siguiente manera:

```
nombre variable[índice] = expresión del tipo <tipo >
```

Para tener acceso al contenido de un arreglo:

```
nombre variable[índice]
```

- **Apuntadores**

Los apuntadores también conocidos como **punteros** son variables que guardan direcciones de memoria. Proporcionan mucha utilidad al programador para accesar y manipular datos de maneras que no es posible en otros lenguajes, y también son útiles para pasarle parámetros a las funciones de tal modo que les permiten modificar y regresar valores a la rutina que las llama.

Los punteros, al igual que una variable común, pertenecen a un tipo, se dice que un puntero apunta a ese tipo al que pertenece. *Ejemplos:*

```
int* pint; //Declara un puntero a entero
char* pchar; //Puntero a char
fecha* pfecha; //Puntero a objeto de clase fecha
```

- **Estructuras**

Una estructura es un grupo de variables relacionadas de manera lógica, las cualespueden ser de diferentes tipos y declaradas en una sola unidad, donde la unidad es la estructura. Se forma utilizando la palabra reservada **struct**, seguida por un campo etiqueta opcional, y luego una lista de miembros dentro de la estructura. La etiqueta opcional se utiliza para crear otras variables del tipo particular de la estructura.

```
struct campo_etiqueta{
    tipo_miembro miembro_1;
    tipo_miembro miembro_2;
    .
    .
    .
    tipo_miembro miembro_n;
};
```

UN PROGRAMA ELEMENTAL

```c
#include <stdio.h>
main() {
 printf("Hola");
}

/*
Programa de Ejemplo
Fecha_
Autor_
*/
#include ____
#define ____
typedef ____
[Prototipos]
int main(void)
{
[variables] /* descripcion */
[instrucciones]
return 0;
}
```

TIPOS DE DATOS BÁSICOS

- Números enteros: **int**
- Números reales: **float** (simple precisión) y double (doble precisión)
- Caracteres: **char**
- Operaciones habituales: suma (+), resta (-), multiplicación (*), división (/), resto de la división o "módulo" (%).
- Operaciones abreviadas: incremento en una unidad (++), decremento en una unidad, incremento en varias unidades (x+=2), decremento en varias unidades (z-=3) y abreviaturas similares para las demás operaciones (x*=5; z/=4; y%=2;).

ESTRUCTURAS BÁSICAS

if

- Comprueba una condición.
- Ejemplo: if (x==5) printf("Vale 5");
- Tipos de condiciones posibles: Mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=), distinto de (!=), igual a (==).
- Si no se cumple la condición: else -> if (x==5) printf("Vale 5"); else printf("No vale 5");
- Para enlazar varias condiciones: y (&&), o (||), no (!): if ((x==5) && (y==1)) printf("Son 5 y 1");

while

- Repite mientras se cumpla una condición (la condición se comprueba antes de empezar a repetir).
- Ejemplo: while (x!=correcto) { printf("Vuelve a intentar"); scanf("%d", &x); }
- Tipos de condiciones y forma de enlazarlas: igual que para "if".

do-while

- Repite mientras se cumpla una condición (la condición se comprueba después de haber dado la primera "vuelta").
- Ejemplo: do {printf("Vuelve a intentar"); scanf("%d", &x); } while (x!=correcto);
- Tipos de condiciones y forma de enlazarlas: igual que para "if".

MANEJO BÁSICO DE PANTALLA Y TECLADO

printf

- Escritura formateada en pantalla
- Ejemplo: printf("El resultado es %d", x);
- Formatos más habituales: %d = número entero en decimal, %f = número real (coma flotante), %c = caracter, %s = cadena de texto, %x = número entero en hexadecimal
- Devuelve: el número de caracteres escritos

scanf

- Lectura formateada desde teclado
- Ejemplo: scanf("%d", &x);
- Formatos más habituales: similares a "printf"
- Devuelve: el número de datos leidos (0 = ninguno, EOF = error)
- Observaciones: en general, el nombre de la variable se debe preceder de & (no es necesario si se trata de un array)

BUCLES

Bucle **for**

```
for(inicialización, condición, instrucción_final)
    {
    [instrucciones]
}

Ejemplo: for(i=0; i<10; i++)
```

Bucle **while**

```
while (condición) {
    [instrucciones]
}
```

Bucle **do-while**

```
do {
    [instrucciones]
} while(condición);
```

Bucle **if**

```
case 1:
if (condición) {
    [instrucciones]
}

case 2:
if (condición) {
    [instrucciones_1]
} else {
    [instrucciones_2]
}

case 3:
if (condición_1) {
    [instrucciones_1]
} else if (condición_2) {
    [instrucciones_2]

...

} else if (condición_n) {
    [instrucciones_n]
} else {
    [instrucciones]
}
```

SINTAXIS DEL SWITCH

```
switch(expresión_entera) {
case constante_1:
    [instrucciones_1]
    break;
case constante_2:
    [instrucciones_2]
    break;
...
case constante_3:
    [instrucciones_3]
    break;
default:
    [instrucciones]
}
```

CADENAS

```
//Lectura:

scanf("%s",cadena);
//lee una palabra

gets(cadena);
//lee una frase hasta fin de linea

fgets(cadena, N, stdin);
/*lee una frase con control de tamaño.
También lee \n */


//Escritura:

printf("%s",cadena);
/*escribe una cadena por pantalla,
vale para frase o palabra */
```

ESTRUCTURAS

```
//Declaración de un tipo estructura
typedef struct persona {
    char nombre [N];
    int edad;
    long dni;
} PERSONA;

//Declaración de variables
PERSONA p; //una estructura
PERSONA *pp; //puntero a estructuras
PERSONA vec[20]; //vector de estructuras

//Acceso a los miembros
p.edad=27;
pp->edad=30;
vec[7].edad=37;
```

MANEJO DE FICHEROS

Para manejar ficheros, siempre deberemos realizar tres operaciones básicas:

- Abrir el fichero.
- Leer datos de él o escribir datos en él.
- Cerrar el fichero.
- Para declarar fichero: **FILE* fichero;**
- Para abrir un fichero: **fichero = fopen(nombreFichero, modo);** Los modos son:
  - **r** Abrir sólo para lectura.
  - **w** Crear para escribir. Sobreescribe el fichero si existiera ya (borrando el original).
  - **a** Añade al final del fichero si existe, o lo crea si no existe.
  - **+** Se escribe a continuación de los modos anteriores para indicar que también queremos modificar. Por ejemplo: r+ permite leer y modificar el fichero.
  - **t** Abrir en modo de texto.
  - **b** Abrir en modo binario.
- Obtener un carácter del fichero: **caracter = fgetc(fichero)**
- Escribir un carácter en el fichero: **fputc(caracter,fichero)**
- Escribir datos formateados en el fichero: **fprintf(fichero,"formato", argumento1, argumento2...)**
- Leer de fichero: **fscanf(fichero, "formato", argumento1,....)**
- Cerrar fichero: **fclose(fichero)**
- Distinto de 0 si acaba el fichero: **feof(fichero)**
- Leer linea en cadena: **fgets(cadena, cantidad , fichero)**
- Escribir cadena: **fputs(cadena, fichero)**
- Saltar a una posición: **fseek(fichero, salto, desde)** (el "desde" puede ser: SEEK_SET, SEEK_CUR, SEEK_END, para principio, actual o final del fichero

BIBLIOGRAFÍA CONSULTADA

Olivares, L. (2008). Manual de Programación en Lenguaje C++. Recuperado el 10 de junio de 2021, de https://paginas.matem.unam.mx/pderbf/images/mprogintc++.pdf