TEMA 25

PROGRAMACIÓN ESTRUCTURADA. ESTRUCTURAS BÁSICAS. FUNCIONES Y PROCEDIMIENTOS

INDICE:

Realizado por Cayetano Borja Carrillo

Tiempo de escritura: 1 hora y 45 minutos

1. INTRODUCCIÓN

En los inicios del desarrollo del *software*, la principal preocupación de los programadores era hacer que el programa funcionase y no se usaba ninguna metodología como tal. El estilo de programación era desestructurado y esto acarreaba diferentes problemas, entre ellos, que el código fuera difícil de entender y de mantener.

Como respuesta a esta problemática, surge a finales de la década de los 60 la programación estructurada, que se enfoca en organizar el código usando estructuras de control y técnicas de modularidad para crear programas más legibles y fáciles de mantener.

En este tema se desarrollan los aspectos fundamentales de la programación estructurada. Se trata de un tema de gran importancia dentro del campo de estudio del desarrollo del *software*, ya que, aunque existen otros paradigmas muy populares, éste sigue siendo uno de los más usados del mundo.

2. PROGRAMACIÓN ESTRUCTURADA

La programación estructurada está pensada no solamente para desarrollar programas fiables y eficientes, sino que además estos deben de estar escritos dando prioridad a los siguientes puntos:

- Claridad y simplicidad, dando como resultado la posibilidad de ser modificado por personas que no intervinieron en su diseño original.
- Permitir la reutilización del código gracias a su modularidad.
- Facilitar la depuración y corrección de errores.
- La capacidad de mantener el software.
- Reducir el tiempo de desarrollo.

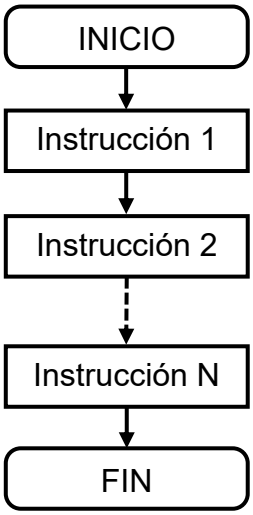
3. ESTRUCTURAS BÁSICAS

En la programación estructurada, las estructuras básicas son herramientas fundamentales que permiten a los programadores controlar el flujo de ejecución de un programa. Estas estructuras incluyen las estructuras secuenciales y de control (de selección e iterativas), así como estructuras de datos y bloques de código reutilizables, como funciones y procedimientos.

3.1. Estructura secuencial

Consiste en la ejecución de las instrucciones de un programa de forma ordenada y secuencial (una detrás de otra) hasta el fin del proceso. Una instrucción es una acción sobre los datos que se lleva a cabo inmediatamente, como puede ser la suma de 2 valores.

Para representar una estructura secuencial de forma gráfica, independientemente del lenguaje de programación que se vaya a usar para su codificación, se utilizan el pseudocódigo y/o el diagrama de flujo.

Pseudocódigo	Diagrama de flujo
INICIO Instrucción 1 Instrucción 2 … Instrucción N FIN	

Las sentencias que se pueden aplicar en una instrucción son:

Asignación

Consiste en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor. Los valores que se pasan deben ser compatibles con el tipo de datos de la variable, es decir, si la variable es numérica de tipo entero, sólo se le pueden asignar números enteros. Ejemplos de asignación:

Pseudocódigo	Código en lenguaje C

Variables:
entero: valor
INICIO
valor ←15
valor ← valor + 1
valor ← 3 * 4
FIN
	int main () {
	int valor;
	valor = 15;
	valor = valor + 1;
	valor = 3 * 4;
	}

Escritura o salida de datos

Consiste en enviar por un dispositivo de salida, como puede ser un monitor o una impresora, el contenido de una variable, un resultado o un mensaje. Ejemplos de salida de datos:

Pseudocódigo	Código en lenguaje C

Variables:
  entero: valor
INICIO
  Escribir “HOLA MUNDO”
  valor ←15
  Escribir “El valor es: “, <valor>
FIN
	int main () {
	  int valor;
	  printf (“HOLA MUNDO”);
	  valor = 15;
	  printf (“\nEl valor es: %d”, valor);
	}

Lectura o entrada de datos

Consiste en recibir desde un dispositivo de entrada, como puede ser un teclado, un determinado valor y almacenarlo en una variable. Ejemplos de entrada de datos:

Pseudocódigo	Código en lenguaje C

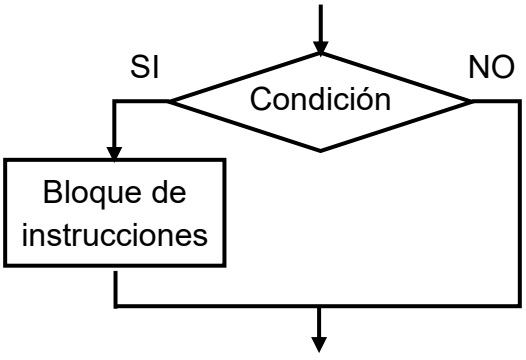
Variables:
  entero: valor
INICIO
  Escribir “Teclea un número”
  Leer valor
  Escribir “Has pulsado: “, <valor>
FIN
	int main () {
	  int valor;
	  printf (“Teclea un número: “);
	  scanf (“%d, &valor);
	  printf (“\nHas pulsado: %d”, valor);
	}

3.2. Estructuras de control de selección

Estas estructuras comparan una variable con uno o más valores y en base al resultado de esa comparación, el flujo tomará un camino u otro. Existen tres tipos básicos de estructuras de control de selección y son las siguientes:

Estructura “Si-Entonces” (*if-then*)

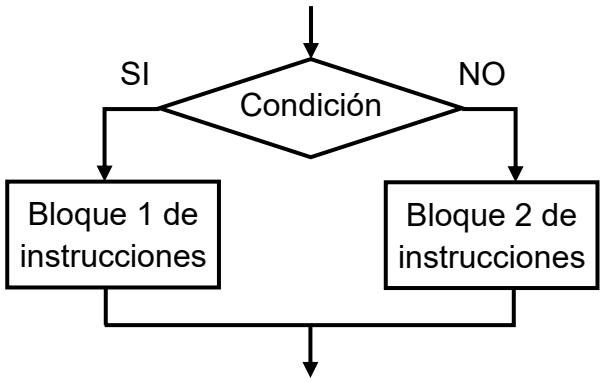
Si la condición evaluada se cumple, se ejecutan un conjunto de instrucciones, en caso contrario no se ejecutan y se continúa con la ejecución normal de instrucciones. Ejemplo:

Pseudocódigo	Diagrama de flujo
… Si <Condición> Entonces Bloque de instrucciones Fin Si …	

Estructura “Si-Entonces / Si-No” (*if-then-else*)

Si la condición evaluada se cumple, se lleva a cabo un conjunto de instrucciones, en caso contrario se ejecuta otro conjunto diferente de instrucciones. Ejemplo:

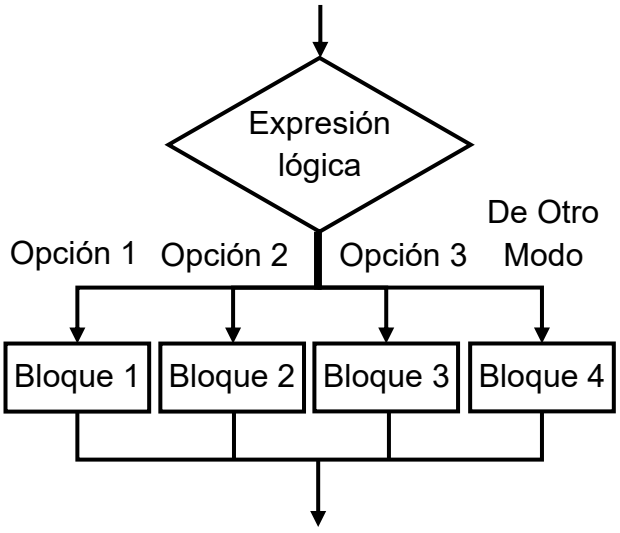
Pseudocódigo	Diagrama de flujo

…
Si <Condición> Entonces
  Bloque 1 de instrucciones
SiNo
  Bloque 2 de instrucciones
Fin Si
…
	

Estructura según (*switch case*)

Si evalúa la expresión y selecciona una ruta entre sus múltiples alternativas. En caso de no encontrar ningún camino que cumpla con la expresión, tomará la ruta predeterminada llamada “De otro modo”. Ejemplo:

Pseudocódigo	Diagrama de flujo

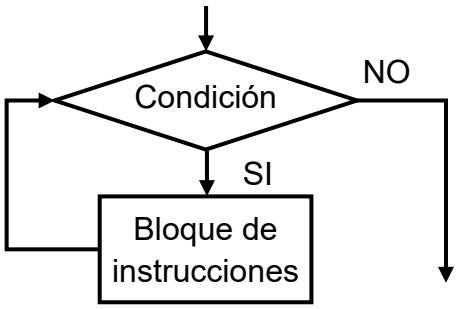
…
Segun <Expresión> hacer
  Opcion_1:
  Bloque 1
  Opcion_2:
  Bloque 2
  Opcion_3:
  Bloque 3
  De Otro Modo:
  Bloque 4
Fin Segun
…
	

3.3. Estructuras de control iterativas

Las estructuras iterativas o bucles son aquellas que permiten ejecutar un conjunto de instrucciones de forma repetitiva. Hay varios tipos de bucles, pero todos tienen un factor en común, que es el evaluar una condición que será la que determina si se ejecutan de nuevo las instrucciones que hay en la estructura o no (sale del bucle). Existen tres tipos básicos de estructuras de control iterativas y son:

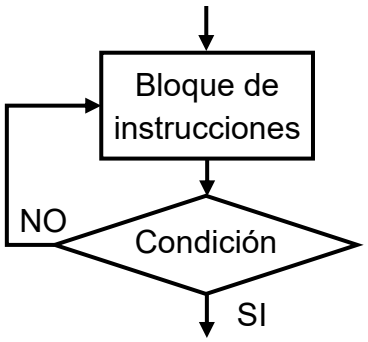
Estructura Mientras (*while*)

Esta estructura evalúa una condición y ejecutará un conjunto de instrucciones de forma cíclica mientras la condición se cumple. Ejemplo:

Pseudocódigo	Diagrama de flujo
… Mientras <Condición> Hacer Bloque de instrucciones Fin Mientras …	

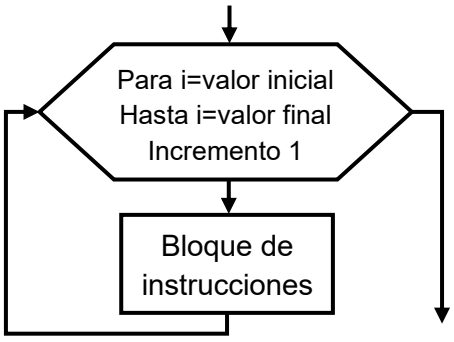
Estructura Repetir Hasta Que (*do-while*)

Esta estructura ejecuta un conjunto de instrucciones de forma cíclica hasta que la condición se cumpla. Ejemplo:

Pseudocódigo	Diagrama de flujo
… Repetir Bloque de instrucciones Hasta Que <Condición> …	

Estructura Para (for)

Esta estructura ejecuta un conjunto de instrucciones una cantidad determinada de veces. Es necesario indicar el valor inicial, el valor final y lo que se incrementará la variable evaluada en cada iteración. Ejemplo:

Pseudocódigo	Diagrama de flujo
… Para i←VInicial Hasta VFinal Con Incremento 1 Bloque de instrucciones Fin Para …	

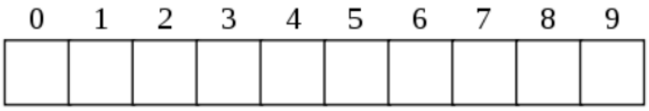
3.4. Estructuras de datos

Una estructura de datos es una forma de organizar una colección de datos con el fin de facilitar su manipulación. Las estructuras pueden ser estáticas o dinámicas, dependiendo de si el tamaño que ocupan en memoria es fijo o variable.

Array

Un *array* es una estructura de datos compuesta por un conjunto determinado de elementos del mismo tipo de datos (todos enteros, todos booleanos, etc.). Cada elemento del *array* tiene asociada una dirección única llamada índice, que determina su posición en el *array*.

La siguiente ilustración representa de forma gráfica un vector de 10 elementos. Nótese que los índices empiezan a contar desde el 0, pero esto puede variar dependiendo del lenguaje utilizado.



Registro

Un registro es una colección de datos que pueden ser de distinto tipo de datos, por ejemplo, un campo puede ser de tipo cadena de texto y otro de tipo entero.

Cadena de texto [20]	entero

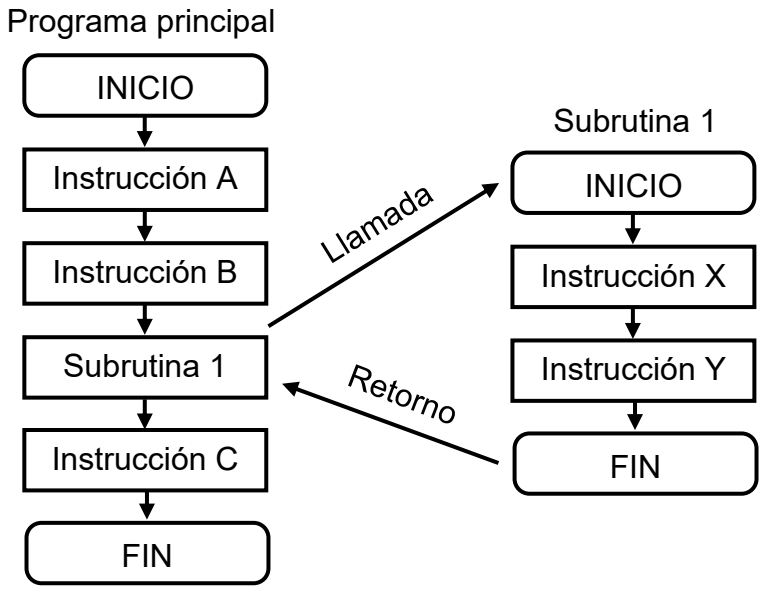
Debido a que cada registro puede tener una estructura distinta que depende de las necesidades del programador, no se pueden declarar tal cual y hay que diseñarlas.

Estructuras dinámicas

Una estructura dinámica es aquella donde el tamaño que ocupa en memoria puede cambiar durante el tiempo de ejecución. Esto es posible porque se basa en nodos que almacenan los datos y una o varias referencias a otros nodos (punteros). Dentro de esta categoría nos encontramos con las listas, pilas, colas, árboles y grafos.

4. FUNCIONES Y PROCEDIMIENTOS

Los procedimientos y las funciones son segmentos de código (subrutinas) separados del bloque principal. Pueden ser invocados en cualquier momento desde el programa principal o desde otra subrutina para resolver parte del problema. Al terminar su ejecución, se retorna al punto donde se hizo la llamada.



La diferencia entre procedimiento y función es que la función puede devolver un valor de un determinado tipo mientras que el procedimiento no.

Ambos son definidos mediante una cabecera con los siguientes elementos:

- Nombre que lo identifica.
- Lista de parámetros que la subrutina necesita para realizar su tarea. Cada parámetro está definido por un nombre y un tipo de datos asociado.
- Solo en el caso de la función, un tipo de datos de retorno.

Ejemplo: Crear en lenguaje C un programa que pida un número y muestre por pantalla el resultado tras sumarle 3. Realizar la suma en una función de tipo entero que devuelva el resultado de la operación y mostrar el mensaje mediante un procedimiento.

<table>
<tr><th>Programa principal</th><th>Función</th></tr>
<tr><td rowspan="3">

```
int sumarValores (int n1, int n2);
void mostrarMensaje (int res);

int main () {
    int num, resultado;
    printf (“Introduzca un número”);
    scanf (%d”, &num)
    resultado = sumarValores (num, 3);
    mostrarMensaje (resultado);
    return 0;
}
```

</td><td>

```
int sumarValores (int n1, int n2) {
    int resultado;
    resultado = n1 + n2;
    return resultado;
}
```

</td></tr>
<tr><th>Procedimiento</th></tr>
<tr><td>

```
void mostrarMensaje (int valor) {
    printf (“El resultado es: %d”, valor);
}
```

</td></tr>
</table>

Como se puede observar en el ejemplo, los elementos de la cabecera de la función son los siguientes:

- Nombre: sumarValores
- Lista de parámetros: n1 y n2, ambos de tipo entero.
- Tipo de datos de retorno: entero.

A continuación, vamos a ver las distintas características que pueden tener los parámetros.

4.1. Parámetros formales y actuales

Llamamos parámetros formales a aquellos que se encuentran en la cabecera de la definición de la subrutina. Por otro lado, llamamos parámetros actuales a aquellos que se utilizan en la llamada o invocación de la subrutina.

Si observamos el ejemplo anterior, los parámetros de las 2 primeras líneas (n1, n2 y res) son formales ya que se encuentran en la definición y los parámetros que se pasan a las subrutinas (num, 3 y resultado) son actuales.

4.2. Tipos de parámetros

Se pueden distinguir varios tipos de parámetros dependiendo de su función, es decir, de si se utilizan para introducir información en la subrutina o para devolverla. Se pueden clasificar en tres grupos:

- Parámetros de entrada: Sirven para introducir datos a la subrutina. Por ser sólo de entrada, si se modifica su valor, no tendría efecto en el exterior.

- Parámetros de salida: Sirven para devolver valores al exterior, por lo que deberán ser modificados a lo largo de la ejecución de la subrutina

- Parámetros de entrada y salida: Son parámetros que sirven tanto para introducir información a la subrutina como para devolver resultados.

4.3. Paso de parámetros

Existen varios métodos para pasar parámetros a las subrutinas; estos métodos van a depender directamente de cómo se almacenan físicamente los datos en la memoria.

- Paso por valor: Se crea una copia local de la variable dentro de la subrutina y durante la ejecución de ésta sólo se trabaja con la copia, por lo que cualquier cambio que haga sobre la variable no se verá reflejado en el nivel superior (donde se hizo la invocación).

- Paso por referencia: Se maneja directamente la variable, es decir, el valor que es proporcionado como parámetro apunta exactamente a la misma dirección de memoria que el valor inicial, por lo que los cambios realizados dentro de la subrutina le afectarán también fuera.

4.4. Tipos de variables

Las variables pueden clasificarse en:

- Variables locales: Se declaran dentro de la subrutina y no pueden utilizarse fuera de ella.
- Variables globales: Son declaradas en el nivel superior y su ámbito se extiende a todo el programa y subrutinas.

5. CONCLUSIÓN

La programación estructurada supuso un hito en la programación, ya que, desde el momento de su aparición se empezó a dar importancia, no sólo a la programación en sí, sino a cómo se programa.

Sus principios, adaptados a actualidad y embebidos en los nuevos paradigmas de programación, siguen siendo válidos hoy en día y aplicarlos supone tener códigos más reutilizables, programas más claros, facilidad a la hora de depurar programas y una reducción importante en el tiempo y costo de desarrollo.

6. BIBLIOGRAFÍA

- López Ureña, L. A. et al. (1997). *Fundamentos de Informática (1ª ed.)*. Ra-ma.
- Prieto Espinosa, A. et al. (2006). *Introducción a la informática (4ª ed.)*. McGraw-Hill.
- Brookshear, J. G. (2012). *Introducción a la computación (11ª ed.)*. Pearson Educación.
- Joyanes Aguilar, L. (2020). *Fundamentos de programación. Algoritmos, estructuras de datos y objetos (5ª ed.)*. McGraw-Hill.

7. NORMATIVA

Para el desarrollo de este tema, se ha tenido en cuenta la siguiente normativa, donde se especifican los contenidos, competencias y criterios de evaluación de los Ciclos Formativos y Bachillerato en Andalucía:

- Orden 16 de junio de 2011 (DAW/DAM). La parte correspondiente al módulo “Programación” y “Entornos de desarrollo”.
- Instrucción 13/2022 (Bachillerato). La parte correspondiente a la asignatura “Tecnologías de la Información y Comunicación”