Conceptos

Programa

Se le llama programa a la serie de instrucciones escritas en alguno de los lenguajes, por medio de los cuales se logra que la computadora realice todas las operaciones o decisiones señaladas en dichas instrucciones.

Podemos distinguir dos tipos de programa:

Programa fuente: es el conjunto de instrucciones escritas en algún lenguaje de computadora, las cuales han sido transcritos para ser interpretadas por algún dispositivo de lectura.

Programa objeto: es el conjunto de instrucciones que componen un programa fuente y que han sido traducidas al lenguaje maquina por medio del compilador correspondiente.

Un compilador es un programa que traduce el programa fuente a programa objeto. Un compilador independiente se requiere para cada lenguaje de programación. Éste efectúa sólo la traducción, no ejecuta el programa. Una vez compilado el programa, el resultado en forma de programa objeto será directamente ejecutable.

Pseudocódigo

El pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. El pseudocódigo no es realmente un código sino una imitación y una versión abreviada de instrucciones reales para las computadoras.

Algoritmos y Diagramas de flujo

Son los pasos a seguir para el desarrollo de un problema, existen básicamente dos tipos de elementos con los cuales es posible especificar un problema en forma esquemática y con una notación orientada a la computación, estas herramientas son los algoritmos y los diagramas de flujo que se complementan.

El concepto de algoritmo es muy importante dentro del área de computación, cuyo significado actual es similar a una receta, proceso, método, técnica, procedimiento o rutina para realizar una actividad, excepto que el algoritmo tiene una connotación ligeramente diferente.

Un algoritmo es un conjunto de reglas que determinan la secuencia de las operaciones a seguir para resolver un problema específico y que cumple con las siguientes cinco características:

1) Finitud: Un algoritmo debe terminar después de ejecutar un número finito de pasos.

2) Definición: Cada paso en un algoritmo debe estar definido con precisión, esto es, la acción a seguir no debe ser ambigua, sino rigurosamente especificada. Un algoritmo descrito en un lenguaje como inglés o español, en el cual una misma palabra puede significar varias cosas, puede no cumplir con este punto. Es por eso que se han definido los lenguajes de programación o lenguajes de computación para especificar algoritmos, ya que en ellos el significado de cada palabra es uno y sólo uno.

3) Entrada: Se considera como entrada el conjunto de datos o información requerida para resolver un problema dado. No cualquier grupo de datos se puede considerar como entrada en el procedimiento señalado.

4) Salida: La salida es un conjunto de resultados que se obtienen al aplicar el algoritmo al conjunto de datos de entrada.

5) Efectividad: Un algoritmo debe llevar a la solución del problema planteado, en otras palabras, se puede decir que todas las operaciones que efectúa el algoritmo, deben ser lo suficientemente simples para que en principio, se puedan ejecutar con papel y lápiz y al final obtener el resultado deseado.

Diagrama de flujo

Introducción

Un Diagrama de Flujo representa la esquematización gráfica de un algoritmo, el cual muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema . Su correcta construcción es sumamente importante porque, a partir del mismo se escribe un programa en algún Lenguaje de Programación. Si el Diagrama de Flujo está completo y correcto el paso del mismo a un Lenguaje de Programación es relativamente simple y directo. Es importante resaltar que el Diagrama de Flujo muestra lugares de origen y destino de los datos, transformaciones a las que son sometidos los datos, lugares en los que se almacenan los datos dentro del sistema, los canales por donde circulan los datos. Además de esto podemos decir que este es una representación reticular de un sistema, el cual lo contempla en términos de sus componentes indicando el enlace entre los mismos.

Al igual que el pseudocódigo, los diagramas de flujo son útiles para el desarrollo y la representación de algoritmos, aunque la mayor parte de los programadores prefieren el pseudocódigo. Los diagramas de flujo muestran con claridad cómo operan las estructuras de control utilizadas en la programación estructurada.

Un diagrama de flujo consta de un conjunto de símbolos con diferentes significados susceptibles de ser conectados entre si. En todo diagrama de flujo podemos encontrar los siguientes elementos:

a) Inicio de proceso.

b) Especificación de la alimentación de datos para efectuar el proceso.

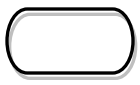
c) Acciones aplicables a los datos.

d) Obtención de resultados.

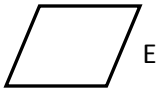
e) Fin del proceso.

Para cada una de estas actividades existen símbolos específicos que denotan los elementos o acciones que se tomarán en el proceso.

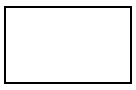
Nota: Estos han sido normalizados por el Instituto Norteamericano de Normalización (ANSI).



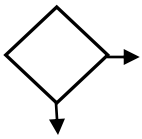
Inicio/fin de programa



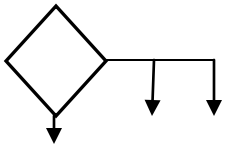
Entrada/Salida de datos



Proceso



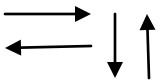
Decisión simple



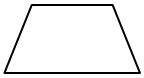
Decisión Múltiple

Proceso predefinido

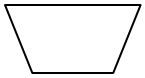
Conector



Dirección de flujo



Inicio de bucle



Fin del bucle

Programación estructurada

La programación estructurada significa escribir un programa de acuerdo a las siguientes reglas:

- El programa tiene un diseño modular.
- Los módulos son diseñados de modo descendente.
- Cada módulo se codifica utilizando las tres estructuras de control básicas: secuencial, alternativa repetitiva.

El termino programación estructurada se refiere a un conjunto de técnicas que aumentan considerablemente la productividad del programa reduciendo en elevado grado el tiempo requerido para escribir, verificar, depurar y mantener los programas. La programación estructurada utiliza un número limitado de estructuras de control que minimizan la complejidad de los programas y por consiguiente reducen los errores.

La programación estructurada es el conjunto de técnicas que incorporan:

- Recursos abstractos.
- Diseño descendente (top – Down).
- Estructuras básicas de control.

Recursos abstractos

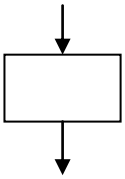
Descomponer un programa en términos de recursos abstractos – según Dijkstra – consiste en descomponer una determinada acción compleja en términos de un número de acciones más simples.

Diseño descendente

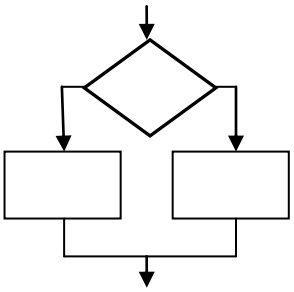
El diseño descendente es el proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento. La metodología descendente consiste en efectuar una relación entre las sucesivas etapas de estructuración, de modo que se relacionen unas con otras mediante entradas y salidas de información. Es decir, se descompone el problema en etapas o estructuras jerárquicas de forma que se puede considerar cada estructura desde dos puntos de vista: ¿Qué hace? Y ¿Cómo lo hace?.

Si se considera un nivel n de refinamiento, las estructuras se consideran de la siguiente manera:

El diseño descendente se puede ver en la siguiente figura.



Nivel n: desde el exterior ¿ lo que hace?



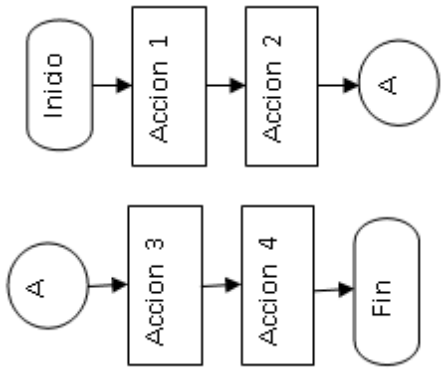
Nivel n+1: vista desde el interior ¿Cómo lo hace?

Estructuras básicas

Cualquier programa con un solo punto de entrada y un solo punto de salida puede resolverse con tres tipos de estructuras básicas de control: Secuencial, alternativa y repetitiva.

- Secuencial

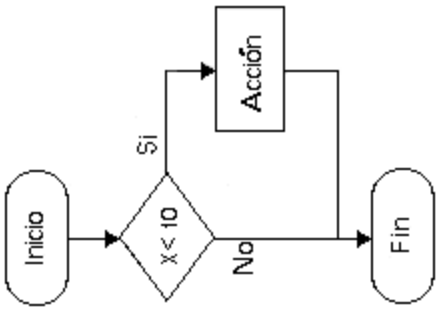
Es aquella que ejecuta las acciones sucesivamente unas a continuación de otras sin posibilidad de omitir ninguna, y naturalmente sin bifurcaciones (saltos a subrutinas). Todas estas estructuras tendrán una entrada y una salida.



- Alternativa

Es aquella en la que únicamente se realiza una alternativa dependiendo del valor de una determinada condición o predicado. Las estructuras alternativas también llamadas condicionales pueden ser de tres tipos: Simple, doble o múltiple

Alternativa simple (IF): Son aquellas en donde la existencia o cumplimiento de la condición implica la ruptura de la secuencia y la ejecución de una determinada acción. Se realiza una acción solo si cierta la sentencia a evaluar es verdadera.

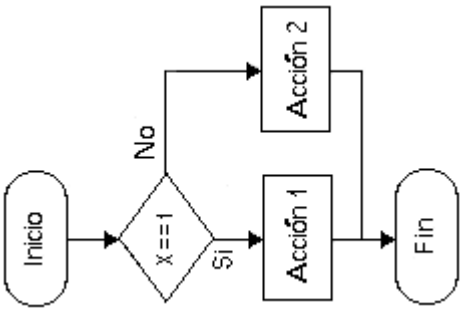


La "acción" se realiza solo sí X es menor que 10. Por ejem.

IF(x==0) x=1; //Sí x ya vale cero, reasignarle el valor a uno.

Alternativa doble (if-else)

Es aquella que permite la elección entre dos acciones o tratamientos en función de que se cumpla o no cierta condición. Sí la sentencia es verdadera, se realiza la acción 1 de lo contrario se realiza la acción 2.



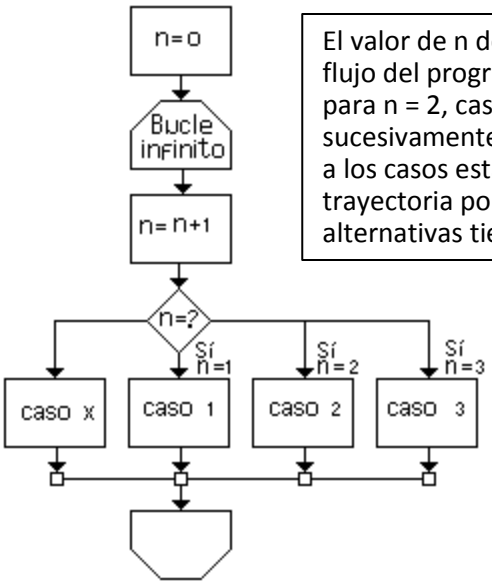
Sí x es igual a 1 se realizará la Acción 1. De lo contrario, se realiza la Acción 2. Por ejemplo;

If(x==1) y<<=1; /* Se lee, Sí x es igual a 1 hacer un corrimiento de y a la izquierda una posición */

Else y>>=1; /* De lo contrario recorrer a la derecha el valor de y una posición*/.

Alternativa múltiple (Switch)

En este caso se evalúa una variable entera. De acuerdo a su valor se determina la secuencia de instrucciones o acciones que se realizará. Esta estructura propuesta por Hoare, es similar a una estructura de casos del lenguaje Pascal o del Basic.



El valor de n determina la dirección del flujo del programa; caso 1 para n =1, caso 2 para n = 2, caso 3 para n= 3 y así sucesivamente. Sí el valor de n es diferente a los casos establecidos, se ejecuta una trayectoria por default (caso x). Todas las alternativas tienen un final de caso.

Ejemplo:

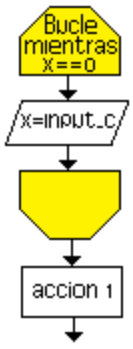
```
Switch(n){
Case 1:
Acción 1
Break;
Case 2:
Acción 2
Break;
Case 3:
Acción 3;
Break;
Default:
Acción x
Break;
```

Repetitiva (Bucle)

Son aquellas en las que las acciones se ejecutan un determinado número de veces y dependen de un valor predefinido o el cumplimiento de una determinada expresión lógica. Un bucle o lazo es el conjunto de acciones a repetir. En consecuencia es preciso disponer de estructuras algorítmicas que permitan describir una iteración de forma cómoda. Las tres estructuras más usuales dependiendo de que la condición se encuentre al principio o al final de la iteración son: Estructura *mientras, repetir hasta y estructura para*.

Estructura Mientras (while)

En el bucle "while" se evalúa la sentencia (p ejem. es x igual a cero?); sí esta es verdadera se ejecutan las instrucciones dentro del bucle(x=input_c). De lo contrario se realizan las instrucciones fuera del mismo (acción 1). El bloque final del bucle regresa al inicio para evaluar nuevamente la sentencia. Dentro del bucle debe existir una instrucción que cambie la condición de verdadera a falsa para que pueda salir del mismo.



// Evaluar sí la sentencia "x==0" es verdadera o falsa

//Sí la sentencia fue verdadera "leer la condición del puerto C" y asignar el valor a x

//Regresar de nuevo al inicio del bucle.

//Sí la sentencia es falsa realizar la acción 1

Ejemplo

```
While(x==1)
x=input(pin_a6);
Y = input(pin_a7);
```

Se lee:
Mientras que x sea igual a uno, asignar a x la condición del pin a6. De otra forma asignar a y la condición del pin_a7

Estructura Hacer – Mientras

El número de iteraciones del grupo de instrucciones se ejecuta una o varias veces hasta que la sentencia sea falsa. Debido a que la sentencia se evalúa al final del bucle, el grupo de instrucciones dentro del mismo se realiza por lo menos una vez. ***Do While***.

//Inicio del Bucle

// Leer la condición del puerto C y asignar su valor a x

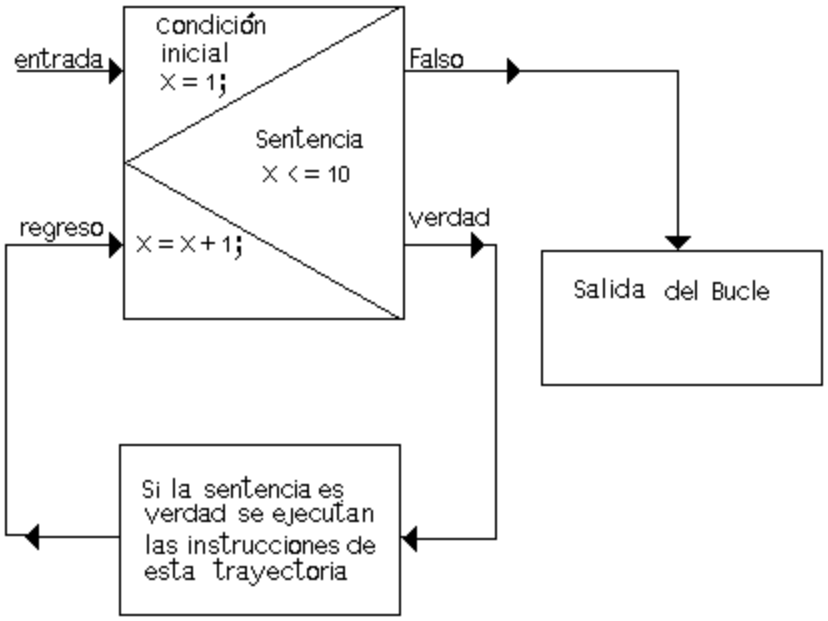
// Sí la sentencia (x==0) es verdadera, regresar al inicio de

// En caso de ser falsa, salir del bucle y realizar la acción 1

```
do{
x<<=1;
if(x==8)x=1;
output_a(x);
delay_ms(10);
if(x==1&&input(pin_a4))
y=3;}
while(y!=3);
output_toggle(pin_c4);
delay_ms(400);
```

Estructura Para (FOR)

En esta estructura se ejecuta las instrucciones dentro del bucle un cierto número de veces. Al ingresar al bucle se asigna un valor inicial a una variable(x=1;), luego se evalúa una sentencia(x<=10). Siempre que sea verdadera, se ejecutarán las instrucciones dentro del bucle y posteriormente se incrementa el valor de la variable(x=x+1). De nuevo se verifica sí la sentencia (x<=10) es verdadera o falsa. Cuando la sentencia sea falsa salimos del bucle for.



For(x=1;x<=10;x++){
Las instrucciones encerradas entre llaves {} se ejecutan mientras la sentencia (x<=10, x sea menor o igual a 10) sea verdadera}
Al hacerse falsa continúa con las instrucciones fuera del bucle.

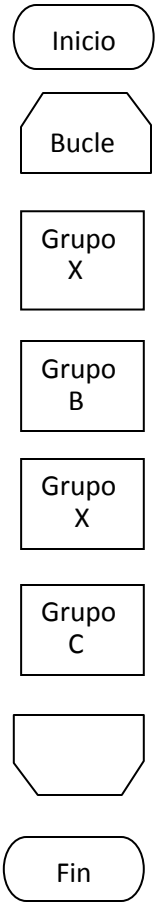
Proceso Predefinido

Cuando en un programa existe un conjunto de varias instrucciones que se repiten en diferentes partes del proceso; es recomendable el uso de macros o subrutinas. Una macro es un programa secundario que incluye toda esta secuencia de instrucciones en un espacio aparte de la memoria y que mediante una sola instrucción de llamada se solicita su ejecución. Con esto se logra una reducción en la cantidad de instrucciones del programa.

En los siguientes diagramas de flujo mostramos esta idea. En la estructura A tenemos un Grupo X de 100 instrucciones ordenadas en secuencia mientras que los Grupos B y C tienen 50 instrucciones. Debido a que el grupo X se repite en dos secciones diferentes, el programa contaría con 300 instrucciones.

En la estructura B se realiza una macro que incluye todas las instrucciones de Grupo X y mediante una instrucción de llamada se solicita que se ejecute cuando sea requerido. Esta estructura cuenta con 202 instrucciones que corresponden a las 100 instrucciones del Grupo X, 50 del grupo B, 50 del grupo C y 2 llamadas a la macro.

Estructura A



Estructura B