aprenderaprogramar.com

Instrucción IrA (GoTo). Saltos no naturales en el flujo normal de un programa. Pseudocódigo y diagramas de flujo. (CU00182A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel I

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº 81 del Curso Bases de la programación Nivel I

24

INSTRUCCIÓN IRA

La instrucción *IrA*, muy conocida por su expresión inglesa *GoTo*, es quizás la más polémica, poderosa y peligrosa herramienta para el control del flujo de programas. Polémica porque ella sola representa todas las virtudes y defectos que se le pueden achacar al control directo de flujos, convirtiéndose en símbolo de un estilo de programación. Poderosa porque todas las instrucciones de control directo de flujo podrían eliminarse para usar en su lugar el *IrA,* e igualmente cualquier estructura de repetición podría ser sustituida mediante el empleo de *IrA*. Y peligrosa porque su uso como base para la construcción de programas, o simplemente como herramienta habitual, se ha demostrado que ocasiona los ya comentados problemas de programas difíciles de leer, entender, corregir y con baja eficiencia.

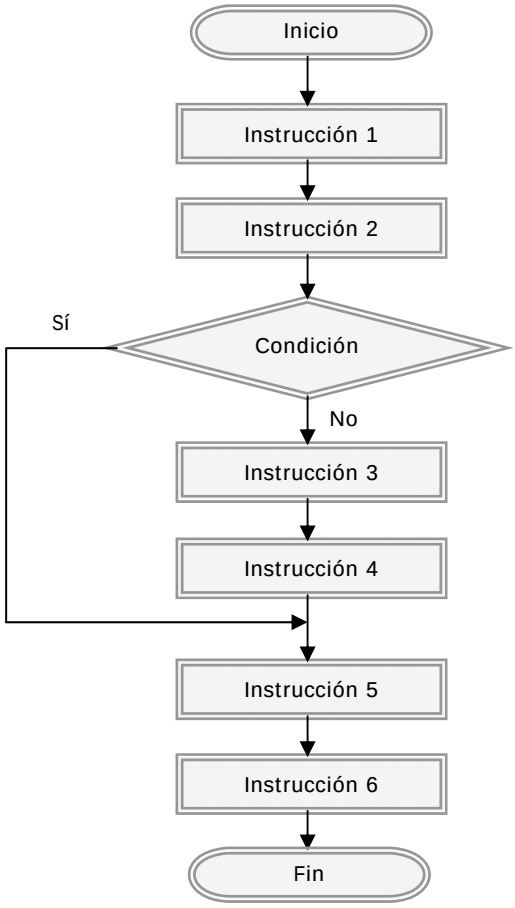
Centrándonos en sus propiedades, diremos que la instrucción *IrA* se usa para provocar un salto no natural entre dos puntos de un programa, sin tener en cuenta el orden de ejecución previsible. El punto donde debe continuar el flujo del programa se especifica poniendo un número de línea a continuación de *IrA*: por ejemplo *IrA 3, IrA 50, IrA 99, IrA 1000, IrA 5.4.*

IrA podría encontrarse en cualquier punto del programa comprendido entre el *Inicio* y el *Fin*. Normalmente irá después de evaluar una situación que es la desencadenante del salto. Los saltos que se consideran viables son: salto adelante, salto atrás y salto del interior de un bucle al exterior.

***Salto adelante*.**

Pseudocódigo y diagrama de flujo:

```
1. Inicio [Salto adelante]
      2. Instrucción 1
      3. Instrucción 2
      4. Si [Condición] Entonces
            IrA 7
         FinSi
      5. Instrucción 3
      6. Instrucción 4
      7. Instrucción 5
      8. Instrucción 6
9. Fin
```

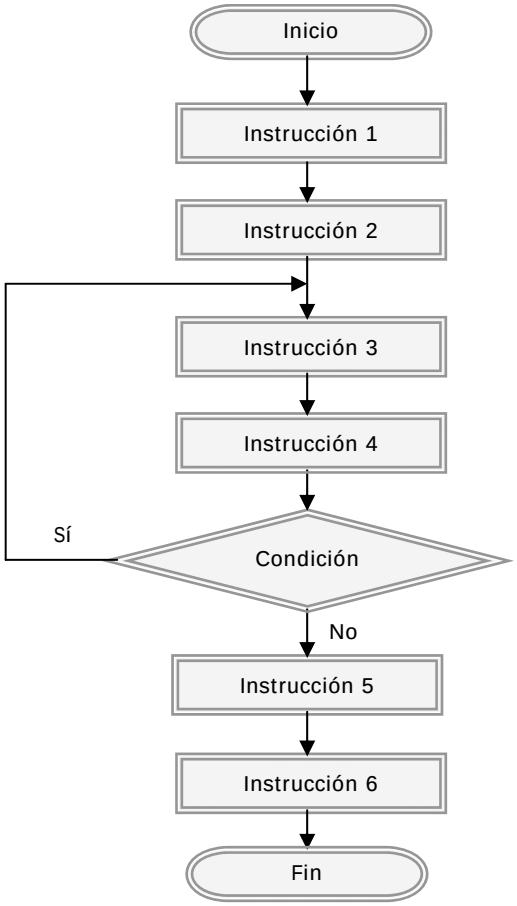


Nota: El salto sin evaluación es posible pero menos recomendable todavía si cabe.

***Salto atrás*.**

Pseudocódigo y diagrama de flujo:

1. **Inicio [Salto atrás]**
    2. Instrucción 1
    3. Instrucción 2
    4. Instrucción 3
    5. Instrucción 4
    6. **Si [condición] Entonces**
        IrA 4
       FinSi
    7. Instrucción 5
    8. Instrucción 6
9. **Fin**



Nota: El salto sin evaluación es posible pero no recomendable.

Nótese como en la hipótesis de salto adelante la circunstancia que se daba era el de un grupo de instrucciones que se dejaban de ejecutar, mientras que en el salto atrás es de un grupo de instrucciones que se repiten. Se entiende que durante la repetición (o las varias repeticiones si es el caso) la situación madura hasta en algún momento dar salida al bucle que se ha creado. De no ser así nos enfrentaríamos a un bucle infinito y bloqueo.

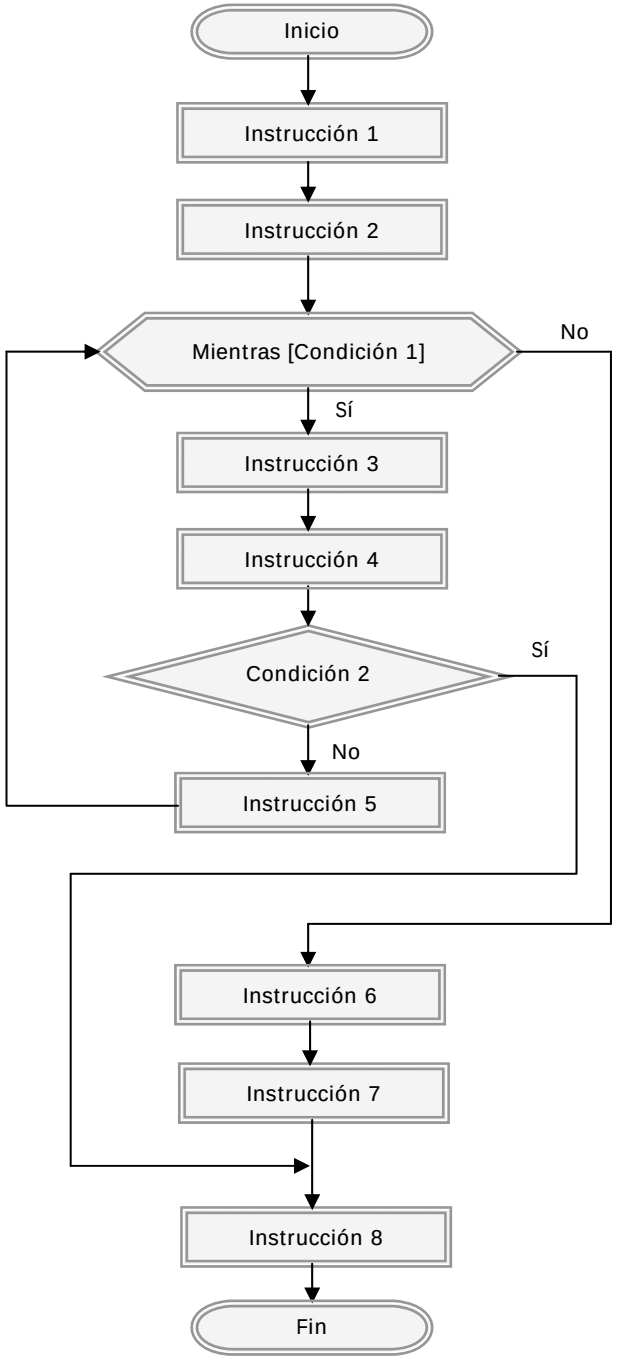
Estas estructuras que hemos visto son fácilmente sustituibles utilizando adecuadamente las instrucciones *Si ... Entonces, Mientras ... Hacer o Hacer ... Repetir Mientras.*

Un salto adelante sin evaluación vendría a emular un *Si ... Entonces* en el cual una de las opciones es siempre válida, mientras que un salto atrás sin evaluación vendría a ser equivalente a un *Hacer ... Repetir*. Ambas situaciones serían absurdas excepto si tratáramos de corregirlas a través de nuevos *IrA* que nos permitieran bien recuperar el código que ha quedado ignorado, bien salir del bucle en el cual nos hemos encerrado. Esto nos llevaría a una espiral de: “genero una situación indeseada ↔ corrijo complicando el flujo” que normalmente se complicaría hasta hacer incomprensible el programa.

Salto del interior de un bucle al exterior.

Pseudocódigo y diagrama de flujo:

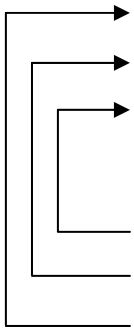
1. **Inicio [Salto desde bucle]**
2. Instrucción 1
3. Instrucción 2
4. **Mientras [Condición 1] Hacer**
   Instrucción 3
   Instrucción 4
   Si [Condición 2] Entonces
   IrA 7
   FinSi
   Instrucción 5
   Repetir
5. Instrucción 6
6. Instrucción 7
7. Instrucción 8
8. **Fin**



Nótese que esta estructura es posible reemplazarla, por ejemplo, usando un interruptor que se active en función de la condición 2 y después subordinando las instrucciones 5, 6, 7 al estado de dicho interruptor. Cuando el programa es muy complejo, manejándose decenas de variables e instrucciones, tendremos que valorar si introducir un *IrA* de este tipo reporta una mayor claridad o ahorro de código que usar las opciones habituales.

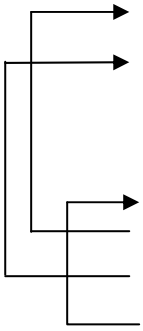
Se observa el paralelismo con el diagrama de flujo de la instrucción *SalirMientras*, con la ventaja de que no nos limita a continuar en la instrucción inmediata a la terminación del bucle.

Consideremos ahora la posibilidad de un salto hacia atrás, por ejemplo en vez de *IrA 7*, *IrA 2.* La posibilidad existe, si bien observamos que se genera un aparente bucle (definido por el *IrA*) superpuesto y "seccionando" al bucle *Mientras ... Repetir.* La situación de superposición o intersecciones de bucles es altamente indeseable por cuanto es causa habitual de errores y malfuncionamiento de los programas. De hecho es algo que ni siquiera habíamos planteado pues hemos venido siguiendo la estrategia de "programas ordenados". Y para que esto sea así, siempre hemos anidado los bucles uno dentro de otro.



Un bucle dentro de otro.

Válido



Un bucle superpone a otros.

No válido

En el caso que nos ocupa se podría "alegar" que en realidad no se trata de un bucle como tal sino de un mecanismo de seguridad que en el caso de que las entradas del usuario no sean válidas reinicia el proceso. Podría ser. Pero casi con seguridad, el funcionamiento del programa será mejor y su comprensión más clara si usamos una alternativa basada en bucles naturales, interruptores e instrucciones *Si ... Entonces.*

La variante del salto del exterior al interior de un bucle no la vamos a analizar, ni hacia delante ni hacia detrás, por el mismo motivo. Genera estructuras superpuestas y frecuentes errores además de malfuncionamientos.

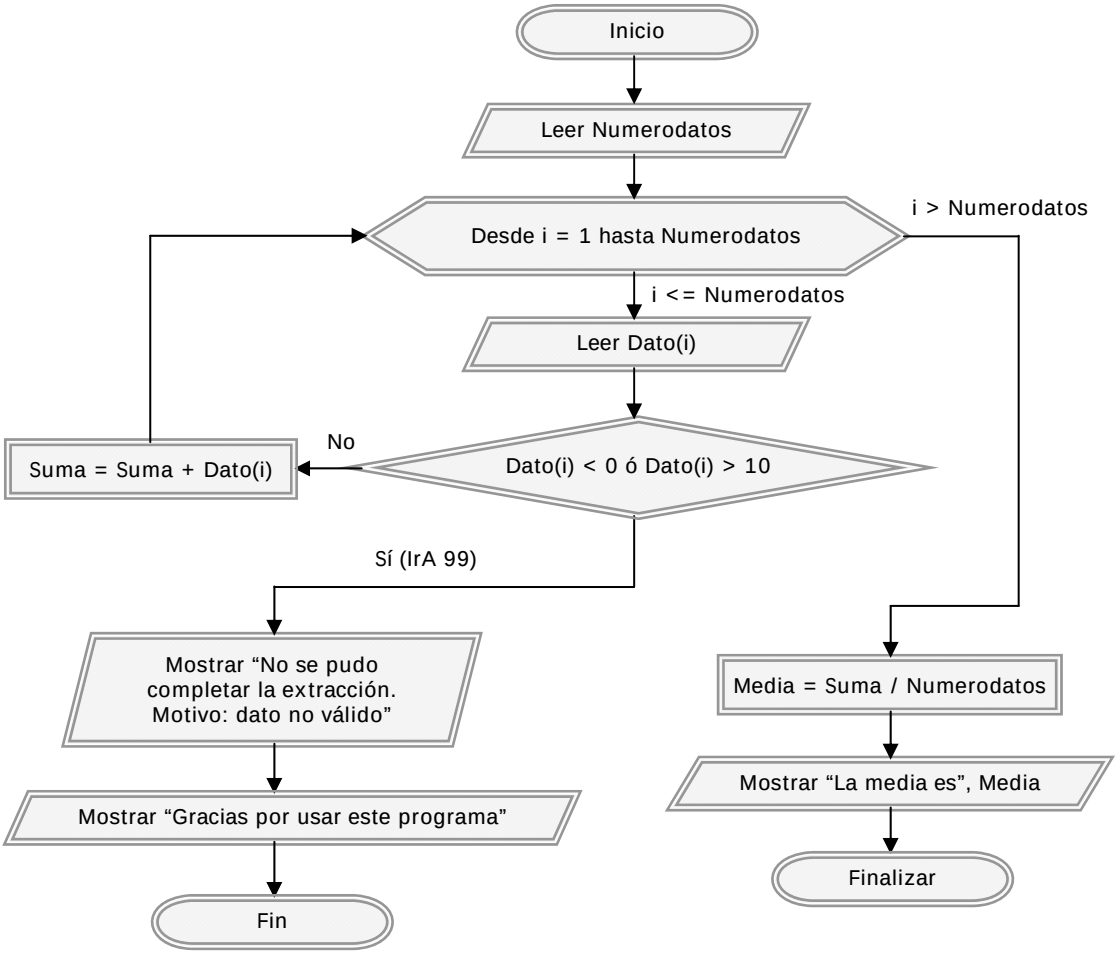
Ejemplo de uso de IrA.

Vamos a considerar una variante del ejemplo de aplicación de *SalirDesde.* En aquella ocasión, nos salíamos del proceso de extracción de datos si un dato se comprobaba erróneo (menor que cero o mayor que 10). En este caso haremos lo mismo con una instrucción *IrA.* El pseudocódigo:

```
1. Inicio [Ejemplo de IrA aprenderaprogramar.com]
     2. Leer Numerodatos [Establece el nº de datos a extraer]
     3. [El dato esperado en fichero debe valer entre cero y diez]
     4. Desde i = 1 hasta Numerodatos Hacer
                Leer Dato(i)
                Si Dato(i) < 0 ó Dato(i) > 10 Entonces [Dato no válido]
                        IrA 99 [Envío a Gestión de Errores]
                FinSi
                Suma = Suma + Dato(i)
        Siguiente
     5. Media = Suma / Numerodatos
```

```
        6.  Mostrar "La media es", Media
        7.  Finalizar
        99. [Gestión de Errores]
                99.1  Mostrar "No se pudo completar la extracción de datos. Motivo:
                                dato no válido. Revise archivo y vuelva a intentarlo"
                99.2  Mostrar "Gracias por utilizar este programa"
    100. Fin
```

Diagrama de flujo:



Comentarios: Si se compara este diagrama de flujo con el mostrado como ejemplo de aplicación de *SalirDesde* se comprueba que el resultado de ambos es el mismo si bien el diagrama de flujo difiere bastante. Vamos a analizar las situaciones que se dan en este ejemplo de aplicación de *IrA.*

1) En primer lugar observamos un salto atípico en la numeración, desde la línea 7 a la 99. Ya sabemos que la numeración es libre y que la estamos utilizando para facilitar la lectura del programa. Y esa es la intención de usar un número como 99: está avisando que esa parte del programa es algo especial. A su vez se han numerado como subordinadas y se han sangrado las líneas siguientes simplemente para facilitar la lectura: indicamos que esas líneas forman parte del bloque especial "Gestión de Errores". La última línea (100) se independiza de ese bloque. Adoptamos un número superior ya que no es posible poner un número inferior a uno previo porque romperíamos el sentido de la numeración.

2) El diagrama de flujo se ha construido mostrando dos ramas. Una sería la correspondiente a “Flujo normal” (se completa el bucle en su totalidad) y otra a “Gestión de errores” (no llega a completarse el bucle). Esto parece introducir cierta claridad a la hora de interpretarlo.

Hay que tener en cuenta que existe un final correspondiente a flujo normal y otro al que se llega a través de la gestión de errores. No necesariamente tendríamos que haber utilizado un *Finalizar*. Se podría haber ahorrado si en la línea 99 hubiéramos evaluado la existencia de errores para dar entrada a las líneas de gestión de errores, bien directamente o bien a través de un interruptor.

El interés de un esquema de este tipo no será tanto para programas cortos sino para programas largos con decenas de bucles y decenas de motivos o puntos donde puede generarse un error. En este caso la instrucción *IrA* nos puede permitir que confluyan en un mismo punto un tipo de situación, como es la relacionada con errores, que se pueda presentar a lo largo del programa. A su vez esto nos puede evitar el tener que estar subordinando continuamente distintas partes del programa a un interruptor o evaluación de error que impida su ejecución.

Finalizamos con las mismas recomendaciones que para el resto de instrucciones de control directo del flujo de programas:

- Evitar el uso de *IrA* siempre que sea posible.
- Recordar lo expuesto en la introducción a la modificación directa del flujo de programas.

Próxima entrega: CU00183A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=28&Itemid=59