UNLu

Unidad 1
Repaso de Introducción a la Programación

03. Estructuras de Decisión

11074 – Programación I

Material didáctico preparado para el dictado de la actividad académica.

Contenido
Aquí hablamos de las **estructuras de decisión** que son una de las formas de controlar el flujo de ejecución de un programa

Actividades
Te ofrecemos actividades a desarrollar para fijar mejor los conceptos

Participación
Te proponemos participar en foros de discusión y compartir tu aprendizaje

Universidad Nacional de Luján
Departamento de Ciencias Básicas
División Computación
www.unlu.edu.ar

Un poco de historia importante!

Vamos a comenzar este tema haciendo un poquito de historia. La historia es importante en este contexto porque nos enseña de dónde venimos y cómo mejoramos nuestros errores en el pasado, lo que nos permite no repetirlos.

¿Cómo se escribían los programas en el pasado? No nos vamos a ir muy atrás en la historia, porque la verdad es que al principio de todo los programas se hacían cableando un panel (literalmente) pero eso ya es historia muy vieja. Vamos a centrarnos en los programas que se escribían con teclado. En esa época el código se escribía sentencia a sentencia. Cada sentencia se enumeraba y el número de línea servía como identificador si en alguna parte se necesitaba hacer un salto en la secuencia de ejecución. Ese salto se hacía con una instrucción llamada GOTO (de *go to* o *ir a* en castellano) y a continuación se indicaba el número de línea al que se quería saltar. Entonces, para entender un programa de cientos de hojas, uno iba saltando de hoja en hoja para atrás y para adelante siguiendo los saltos de secuencia. Luego del quinto salto uno no sabía ni donde estaba. A este tipo de codificación se la denominó, posteriormente y como adjetivo peyorativo, código espagueti. El nombre se le puso porque se decía que el código se parecía a ese tipo de comida, donde los espaguetis se enredan formando una masa que nadie sabe cómo desenredar.

Entonces, aparece como un enfoque mágico lo que se denominó **Programación Estructurada**.

Programación Estructurada

La programación estructurada es un **paradigma** de programación cuyo objetivo es mejorar la claridad y calidad del código de un programa y, por consiguiente, mejorar los tiempos necesarios para desarrollar un programa de computadora.

La idea de este paradigma es que un código pueda disponer, únicamente, de subrutinas y tres estructuras básicas:

- **Secuencia:** ejecución de una instrucción tras otra.
- **Selección:** ejecución condicional de un conjunto de instrucciones. Es decir, esas instrucciones solo deben ejecutarse si cierta condición se cumple.
- **Iteración:** ejecución de un conjunto de instrucciones en forma repetitiva, un número determinado de veces.

Por otra parte, se indica que ya no es necesario y además, es una muy mala forma de programar, la utilización de saltos incondicionales (GOTO), que podría conducir a código espagueti, mucho más difícil de seguir y de mantener, y fuente de numerosos errores de programación.

El paradigma surge hace muchos años. Allá por la década del 1960, se hace presente en un trabajo de Böhm y Jacopini[1] y luego en 1968 se hace famoso en un trabajo de Edsger Dijkstra (considerado hoy el padre de la programación estructurada) denominado "La sentencia Go To, considerada perjudicial"[2]. Sus postulados se verían reforzados, a nivel teórico, por el teorema del programa estructurado. Además, a nivel práctico o de implementación, aparece un lenguaje de programación llamado ALGOL que define estructuras de control consistentes con el paradigma y muy bien formadas.

[1] Böhm, Jacopini. "Flow diagrams, turing machines and languages with only two formation rules" Comm. ACM, 9(5):366-371, May 1966

[2] Edsger Dijkstra (marzo de 1968). «Go To Statement Considered Harmful». Communications of the ACM (PDF) 11 (3): 147–148. doi:10.1145/362929.362947.

Cuadro de información

Puede que te resulte interesante leer sobre el teorema de Edsger Dijkstra. En Wikipedia hay una buena infografía de los fundamentos más importantes

https://es.wikipedia.org/wiki/Teorema_del_programa_estructurado

Entonces, un programa estructurado cuenta, como dijimos, de tres estructuras básicas, la de secuencia, la de selección y la de iteración.

El código que se ve debajo (que es el mismo que aparece en una guía anterior) muestra un conjunto de instrucciones en secuencia:

Código ...

```
Variables (
   ENTERO: primer_sumando, segundo_sumando, resultado
)

primer_sumando <- 5
segundo_sumando <- 3
resultado <- primer_sumando + segundo_sumando
```

Condición

Las estructuras de selección, también llamadas condicionales o de decisión, tienen por objetivo lograr que un conjunto de instrucciones secuenciales, se ejecuten sólo cuando cierta condición es verdadera.

Esto nos obliga a hablar entonces de **condición**. ¿Qué es una condición?

Una condición es una preposición y por lo tanto es posible decir de ella si es verdadera o falsa. Por ejemplo:

El 9 de Julio de este año cae (o cayó) viernes. No tengo idea en qué año estás leyendo esto pero no importa, siempre es posible revisar un almanaque y verificar si es cierto o no que ese día cayó viernes en ese año.

Las condiciones en programación se forman utilizando operadores relacionales. Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión es considerada verdadera, en caso contrario será falsa. Por ejemplo, 8 > 4 (ocho mayor que cuatro) es verdadera, se representa por el valor true del tipo básico boolean, en cambio, 8<4 (ocho menor que cuatro) es falsa, (false). Lamentablemente, el símbolo a utilizar para comparar, también depende del lenguaje de programación que estemos usando con lo cual, cuando aprendas un lenguaje debes averiguar cuáles son estos símbolos para ese lenguaje en particular. Solo a título de ejemplo colocamos aquí una tabla con los operadores relacionales que se utilizan tanto en el lenguaje C como en Python, Go y Java.

En la primera columna de la tabla, pusimos los símbolos del operador relacional, en la segunda el nombre de dicho operador y a continuación su significado mediante un ejemplo.

En esta tabla se utilizan, para los ejemplos, dos variables: a y b. Este tema fue cubierto en la guía didáctica anterior.

Operador	nombre	ejemplo	significado
<	menor que	*a < b*	*a* es menor que *b*
>	mayor que	*a > b*	*a* es mayor que *b*
==	igual a	*a == b*	*a* es igual a *b*
!=	no igual a	*a != b*	*a* no es igual a *b*
<=	menor que o igual a	*a <= 5*	*a* es menor que o igual a *b*
>=	mayor que o igual a	*a >= b*	*a* es menor que o igual a *b*

Con estos operadores relacionales ahora podemos comenzar a ver los tipos de estructuras de selección o condicionales

Estructuras de Selección

Como habíamos visto en las páginas previas, las estructuras de selección, condicionales o de decisión, son uno de los tipos de estructura que nos permite alterar la secuencialidad de ejecución del código de un programa, haciendo que una parte de éste se ejecute o no dependiendo del valor de verdad de cierta condición.

Por ejemplo, supongamos que deseo que un programa me indique si un número determinado es par o impar. Lo primero que debemos hacer es analizar el problema. ¿Que es un número determinado? Supongo que será un número que alguien debe darme. Entonces, mi programa deberá pedir un número y almacenarlo en una variable. Luego ¿Como se si es par o impar? Bueno, creo recordar que un número es par si al dividirlo por dos el resto de la división me da cero. Ahora debería encontrar un operador que me permita obtener el resto (o módulo) de la división entera. Habíamos visto en la guía anterior que algunos lenguajes utilizan el símbolo de porcentaje como operador para obtener el módulo. Si esto fuera así, la condición para saber si el número es par sería que el módulo de dividir la variable (supongamos "numero_a_evaluar") por 2 es igual a 0. Si esto es así, es par, sino, habiendo solo dos alternativas, es impar.

Nuestro programa entonces podría verse así:

Código ...

```
Variables REAL: numero_a_evaluar;

numero_a_evaluar <- Pedir("Dime el número a evaluar : ");
Si (numero_a_evaluar % 2 == 0) Entonces
      Escribir("El número ",numero_a_evaluar," es par ");
Si no
      Escribir("El número ",numero_a_evaluar," es impar ");
Fin Si
Escribir("Fin del Programa");
```

Si pudiéramos ejecutar este programa, se vería algo así (se muestra la salida con dos entradas distintas):

Ejecución

```
<<Primera Ejecución>>
Dime el número a evaluar : 4
El número 4 es par
Fin del Programa

<<Segunda Ejecución>>
Dime el número a evaluar : 5
El número 5 es impar
Fin del Programa
```

Si observamos este ejemplo de salida con los números 4 y 5, se observa que la salida que se verá en pantalla será o bien "El número <numero> es par" o "El número <numero> es impar" pero no ambas.

De esta forma, la estructura **Si .. Entonces** permite ***seleccionar*** cuál conjunto de instrucciones se ejecutarán, dependiendo del valor de verdad de la condición, que en este caso es ***numero_a_evaluar* % 2 == 0.**

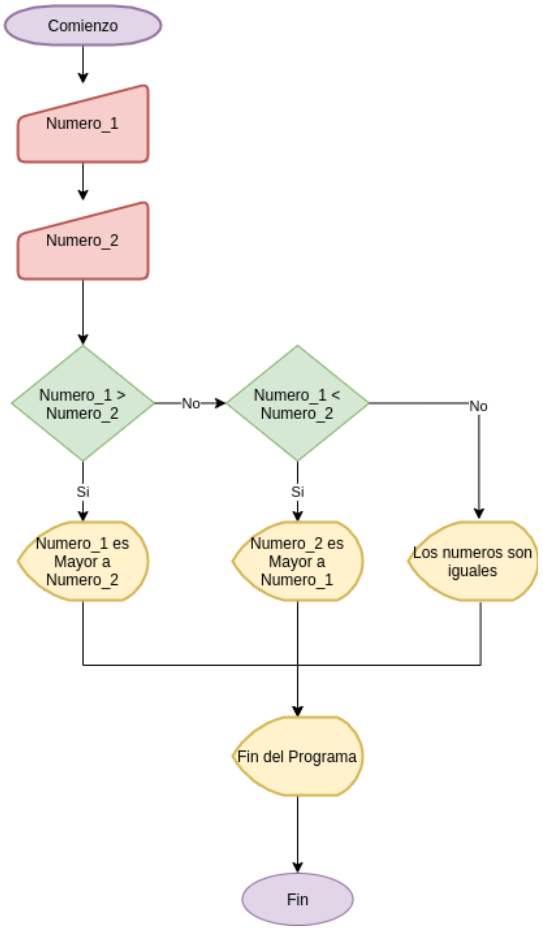
Se observa también que, si la condición no es verdadera, el bloque de código que se ejecutará es el que se encuentra luego de la instrucción **Si no**. Finalmente, se debe notar que, sin importar el valor de verdad de la condición, la frase "Fin del Programa" se observa siempre. Esto nos indica que el bloque de selección finaliza con **Fin Si**

Vamos a otro ejemplo, en este caso nos interesa saber cual es el número más grande de dos números dados. Ojo que si fueran iguales, quiero que me diga que no hay ninguno mayor que otro!!

Entonces de nuevo analizamos: Debemos pedirle a alguien que nos de dos números. Luego los comparamos. Si el primero es mayor que el segundo, ponemos que el primero es mayor, pero sino, hay dos posibilidades: Que sean iguales o que el segundo sea mayor que el primero.

Este problema no se puede resolver con un solo Si..Entonces. Pero las instrucciones de selección se pueden "anidar", esto es poner una dentro de otra.

A veces es bueno usar la técnica de Diagrama de Flujos para observar como quedaría armado nuestro programa. En Introducción a la programación vieron también esta técnica. Y vieron que un rombo representa una estructura de decisión como el Si .. Entonces.



Nuestro diagrama quedaría como se ve en la imágen de la derecha

Entonces podríamos escribir un programa así:

Código ...

```
Variables REAL: primer_numero, segundo_numero;

primer_numero <- Pedir("Dime el primer número : ");
segundo_numero <- Pedir("Dime el segundo número : ");
Si (primer_numero > segundo_numero) Entonces
      Escribir("El número ",primer_numero," es mayor a ",segundo_numero);
Si no
      Si (primer_numero < segundo_numero) Entonces
            Escribir("El número ",segundo_numero," es mayor a
            ",primer_numero);
      Si no
            Escribir("Los números ingresados son iguales");
      Fin Si
Fin Si
Escribir("Fin del Programa");
```

El bloque anidado, en este caso, está en el primer Si No. Podríamos haberlo escrito también así:

Código ...

```
Variables REAL: primer_numero, segundo_numero;

primer_numero <- Pedir("Dime el primer número : ");
segundo_numero <- Pedir("Dime el segundo número : ");
Si (primer_numero != segundo_numero) Entonces
      Si (primer_numero < segundo_numero) Entonces
            Escribir("El número ",segundo_numero," es mayor a
            ",primer_numero);
      Si no
            Escribir("El número ",primer_numero," es mayor a
            ",segundo_numero);
      Fin Si
Si no
      Escribir("Los números ingresados son iguales");
Fin Si
Escribir("Fin del Programa");
```

En este caso, primero se evalúa que los números no sean iguales, de forma tal que, luego de que la condición se evalúa como verdadera (no son iguales), quedan dos posibilidades, que sean que el primero sea mayor que el segundo o que el segundo sea mayor al primero. Por tanto el anidamiento se realiza en el Si Entonces.

Estructura de Selección Múltiple

Otro tipo de estructura de decisión, denominada Estructura de Decisión Múltiple permite realizar varias salidas posibles en función del valor de una variable.
En este caso la condición no se ve claramente, sino que puede inferirse. Veamos un ejemplo:
Supongamos que queremos hacer un algoritmo que indique, en función del número de día de la semana, que escriba su nombre, siendo 1 el Lunes, 2 el Martes y así hasta 7 el domingo.
Si lo hiciéramos con la estructura de selección simple tendríamos que:

```
Si (dia == 1) Entonces
      Escribir("Lunes")
Si no
      ....
```

y acá está el problema. Sino que pasa? Tengo 6 posibilidades más. Por lo tanto debo anidar 5 estructuras simples adicionales!
En lugar de hacer eso podemos hacer:

Código ...

```
Variables ENTERO: dia;

dia <- Pedir("Dime el número de dia (de 1 a 7) : ");
Según dia Sea
      1 : Escribir('Lunes');
      2 : Escribir('Martes');
      3 : Escribir('Miércoles');
      4 : Escribir('Jueves');
      5 : Escribir('Viernes');
      6 : Escribir('Sábado');
      7 : Escribir('Domingo');
      SiNo: Escribir('El número no está entre 1 y 7');
Fin Según
Escribir("Fin del Programa");
```

Si pudiéramos ejecutar el programa tres veces con las entradas 5 , 7 y 9 veríamos :

Ejecución

```
<<Primera Ejecución>>
Dime el número de dia (de 1 a 7) : 5
Viernes
Fin del Programa

<<Segunda Ejecución>>
Dime el número de dia (de 1 a 7) : 7
Domingo
Fin del Programa
```

```
<<Tercera Ejecución>>
Dime el número de dia (de 1 a 7) : 9
El número no está entre 1 y 7
Fin del Programa
```

Como se puede observar, mirando la salida, las instrucciones que se ejecutarán, dependerá del valor de la variable "dia" en este caso y solo se ejecutará el bloque de instrucciones que estén a continuación del valor especificado que coincida con el que tenga la variable.

Nuevamente, no hay un estándar respecto de cómo se usan estas instrucciones en un lenguaje particular. Por lo tanto depende del lenguaje que estés empleando como deberás escribir la instrucción de selección única y la múltiple.

Para ilustrar esto veamos algunos ejemplos:

Código en Python

```python
primer_numero = int(input("Dime el primer número :"))
segundo_numero = int(input("Dime el segundo número :"))
if primer_numero > segundo_numero:
   print("El número ",primer_numero," es mayor que ",segundo_numero)
else:
   if segundo_numero > primer_numero:
       print("El número ",segundo_numero," es mayor que ",primer_numero)
   else:
       print("Los números son iguales")
print("Fin del programa")
```

Código en Go

```go
package main

import "fmt"

func main() {
  var primerNumero int32
  var segundoNumero int32
  fmt.Print("Dime el primer número :")
  fmt.Scanf("%d\n",&primerNumero)
  fmt.Print("Dime el segundo número :")
  fmt.Scanf("%d\n",&segundoNumero)
  if (primerNumero > segundoNumero) {
        fmt.Printf("El numero %d es mayor que %d \n", primerNumero,
segundoNumero)
  } else if (primerNumero < segundoNumero) {
       fmt.Printf("El numero %d es mayor que %d \n", segundoNumero,
primerNumero)
  } else {
     fmt.Printf("Los números son iguales\n")
  }
```

```
  fmt.Printf("Fin del programa\n")
}
```

Ya que estamos viendo dos ejemplos implementados en dos lenguajes diferentes, observa cómo en el primer lenguaje (Python) no fue necesario declarar las variables antes de usarlas. Esto es porque Python utiliza tipado dinámico, en cambio en Go las variables se declaran usando la palabra reservada var y luego se puede especificar el tipo de dato que va a almacenar. Si no se lo hace, lo asume eligiendo el más compatible.

Respecto a la estructura de decisión simple que es lo que nos ocupa aquí, en **Python** la condición va luego del **if (si)** y entiende que lo que está luego de los dos puntos (:) e indentado está dentro de la condición. Para el **else**, es parecido. En cambio en **Go**, la condición va entre paréntesis y el bloque de instrucciones dentro de llaves. Por otra parte en ambos lenguajes tiene la posibilidad de definir un **else** que se combina con un nuevo **if** (como está escrito en el ejemplo Go). En el caso de Go es **else if** y en Python es **elif.** Esta instrucción adicional permite anidar más fácilmente instrucciones de dentro del **if**.

Para el caso de la selección múltiple, en Python no está implementada. En Go el ejemplo de los días de la semana sería así:

Código en Go

```
package main

import "fmt"

func main() {
  var dia int8
  fmt.Print("Dime el número de día ( entre 1 y 7 ) :")
  fmt.Scanf("%d\n",&dia)
  switch dia{
  case 1:
     fmt.Println("Lunes")
  case 2:
     fmt.Println("Martes")
  case 3:
     fmt.Println("Miércoles")
  case 4:
     fmt.Println("Jueves")
  case 5:
     fmt.Println("Viernes")
  case 6:
     fmt.Println("Sábado")
  case 7:
     fmt.Println("Domingo")
  default:
     fmt.Println("No es un día correcto")
  }
  fmt.Println("Fin del Programa")
}
```

Actividad 1

Sabiendo el lenguaje de programación que utilizarás en Programación I, investiga:

¿Cómo crearías un programa donde pudieras calcular el más grande de tres números dados?

Ahora, a participar!!

Escribe tu respuesta en el foro preparado a tal efecto. Recuerda que en este foro solo podrás ver las respuestas de tus compañeros luego de que hayas puesto la tuya.

Puedes comentar, una vez puesta tu respuesta, las de tus compañeros. Es importante que compartamos apreciaciones y pareceres.

Actividad 2

Sabiendo el lenguaje de programación que utilizarás en Programación I, investiga:

¿Cómo crearías un programa donde pudieras mostrar el nombre del mes en castellano, dado el número de mes, donde 1 es Enero, 2 es Febrero, etc?

Ahora, a participar!!

Escribe tu respuesta en el foro preparado a tal efecto. Recuerda que en este foro solo podrás ver las respuestas de tus compañeros luego de que hayas puesto la tuya.

Puedes comentar, una vez puesta tu respuesta, las de tus compañeros. Es importante que compartamos apreciaciones y pareceres.

Actividad 3

Investiga qué son los operadores lógicos, comparalo con lo que sepas de las tablas de verdad (que seguramente viste en otras asignaturas). ¿Es posible usando estos operadores simplificar el programa que hiciste en la actividad 1? ¿Cómo lo harías?

Ahora, a participar!!

Escribe tu respuesta en el foro preparado a tal efecto. Recuerda que en este foro solo podrás ver las respuestas de tus compañeros luego de que hayas puesto la tuya.

Puedes comentar, una vez puesta tu respuesta, las de tus compañeros. Es importante que compartamos apreciaciones y pareceres.

Aprovecha la oportunidad INVALUABLE que te da la virtualidad de manejar tus tiempos para comparar las respuestas de tus compañeros con la tuya. Fijate cómo resolvieron los problemas y de esa forma aprenderás mucho más. El intercambio favorece enormemente la experiencia de enseñanza-aprendizaje.