Capítulo 3

Datos

Objetivos

Anteriormente hemos visto cada una de las estructuras de control y con ellas hemos podido decidir, durante el algoritmo, cuál es el rumbo de acción a seguir.

Sin embargo, existen situaciones en las que es preciso representar información adicional específica del problema a resolver. Por ejemplo, podría resultar de interés saber la cantidad de veces que se apagó el horno mientras se estaba horneando una pizza o la cantidad de pasos realizados por el robot hasta encontrar una flor.

El objetivo de este capítulo es incorporar las herramientas necesarias para lograr representar y utilizar esta información.

Temas a tratar

- ✓ Conceptos de Control y Datos.
- ✓ Representación de los Datos.
- ✓ Variables
  - Sintaxis para la declaración de variables.
- ✓ Tipos de datos.
  - Tipo de dato numérico (numero).
  - Tipo de dato lógico (logico).
  - Tipo de dato alfanumérico (texto)
  - Tipo de dato semáforo - Concurrencia.
- ✓ Modificación de la información representada.
  - Asignación
  - Lectura
- ✓ Ejemplos.
- ✓ Conclusiones.

3.1 Conceptos de Control y Datos

Hasta ahora se ha insistido en las instrucciones que constituyen un algoritmo.

El orden de lectura de dichas instrucciones, en algoritmos como los del capítulo anterior, constituye el control del algoritmo.

Normalmente la lectura del control de un algoritmo, que también puede representarse gráficamente, indica el orden en que se irán ejecutando las instrucciones y como consecuencia de ello, qué es lo que ese algoritmo hace.

Retomemos el ejemplo del capítulo 2: Escriba un programa que le permita al robot recorrer la avenida 7 hasta encontrar una esquina que no tiene flores. Al finalizar debe informar en qué calle quedó parado. Por simplicidad, suponga que esta esquina seguro existe.

```
programa Cap2Ejemplo9
comenzar
  iniciar
  Pos(7,1)
  mientras HayFlorEnLaEsquina
    mover
  Informar( PosCa )
fin
```

Si quisiéramos saber cuántas cuadras recorrió el robot, necesitaríamos considerar dicha cantidad como un dato.

Un dato es un elemento u objeto de la realidad que los algoritmos representan y son capaces de modificar y procesar.

Cuando se resuelven problemas con computadora, muchas veces se modelan los objetos reales mediante objetos más abstractos, representables y entendibles sobre una computadora.

Es importante entender que el mecanismo de resolución de problemas involucra generalmente una transformación o un procesamiento de los datos, manejado por el control del algoritmo.

3.2 Representación de los Datos

Como se dijo anteriormente, los algoritmos modifican objetos de la realidad. La representación de dichos objetos estará dada por sus características principales o por la información que nos interese conocer de ellos.

En el caso del último ejemplo visto, sólo nos concentramos en representar las cuadras recorridas por el robot, y dejamos de lado la cantidad de flores y papeles de cada esquina

visitada. Es decir**, los datos a representar son aquellos de interés específico para resolver ese problema**.

Si además de contar la cantidad de cuadras recorridas, necesitáramos contar la cantidad de flores en todas las esquinas visitadas, sería necesario representar esta información dentro del algoritmo. Por lo tanto, se comenzará extendiendo la notación empleada en el algoritmo para dar lugar a las declaraciones de los datos que resultan relevantes al problema.

3.3 Variables

Además de su capacidad de movimiento y de recoger o depositar flores y papeles, el robot posee la habilidad de manejar datos que le permiten representar ciertos atributos de los problemas que debe resolver. Por ejemplo, es capaz de calcular la cantidad de flores que lleva en su bolsa o recordar si en una esquina dada había más flores que papeles.

En general**, durante la ejecución de un programa es necesario manipular información que puede cambiar** continuamente. Por este motivo, es imprescindible contar con un elemento que permita variar la información que se maneja en cada momento. **Este elemento es lo que se conoce como variable.**

Una variable permite almacenar un valor que puede ser modificado a lo largo del programa. Dicho valor representa un dato relevante para el problema a resolver. Dentro de un mismo programa pueden utilizarse tantas variables como sean necesarias.

3.3.1 Sintaxis para la declaración de variables

Se denominan **identificadores a los nombres descriptivos que se asocian a los datos (variables)** para representar, dentro del programa, su valor.

En el lenguaje del robot, existe una sección dentro del programa donde se declaran las variables que se van a utilizar. La sintaxis es la siguiente:

```
programa nombrePrograma
variables
  {indique aquí las variables a utilizar}
comenzar
  {sentencias que indican el recorrido del robot}
fin
```

Notemos que se ha incorporado una sección para la declaración de las variables entre la línea que contiene el nombre del programa y la palabra comenzar.

Ahora quedan distinguidos dentro del programa dos sectores: un sector superior donde se declaran las variables de interés y un sector inferior donde se detallan las instrucciones del

programa.

Si en el ejemplo antes visto se quisiera contar la cantidad de cuadras recorridas hasta encontrar una esquina sin flor, entonces deberíamos modificar el programa agregando en la zona de declaración de variables un dato que permita manejar y procesar esa información. Veamos como quedaría:

```
programa Cap2Ejemplo9
variables
  cantidadCuadras (*)
comenzar
  {sentencias}
fin
```

(*) Tener en cuenta que esta declaración de variables es aún incompleta dado que nos quedan por ver algunos temas adicionales antes de completarla.

3.4 Tipos de datos

Independientemente del lenguaje de programación que se utilice, la definición de un tipo de dato implica considerar tres aspectos fundamentales que lo caracterizan:

- Identificar **cuáles son los valores** que puede tomar un dato.
- **Definir cuáles son las operaciones** que pueden aplicarse sobre los datos de este tipo.
- Indicar **cuáles son las relaciones de orden** que permiten compararlos.

Se utilizará este mecanismo para definir los tipos de datos con los que trabaja DaVinci Concurrente.

3.4.1 Tipo de dato numérico (numero)

Los elementos de este tipo de dato son los números enteros

-3, -2, -1, 0, 1, 2, 3, ...

Una computadora sólo puede representar un subconjunto finito de valores, con lo cual **existirá un número máximo y un número mínimo**. Dichos valores estarán **determinados por la cantidad de memoria que se utilice para representarlo**. En el intérprete DaVinci Concurrente, el número máximo que se puede representar es la representación de un dato tipo entero en java y este varía según su implementación y arquitectura subyacente. En una máquina tipo de 32 bits tendremos como máximo $2^{31\ -1}$ = 2147483647 y como mínimo es -2^{31} +1= -2147483647.

Las operaciones válidas para los números son: **suma**, **resta, multiplicación y división entera**. Estas operaciones en el lenguaje DaVinci Concurrente se simbolizan como: **+, -, *, /** respectivamente.

Si el resultado de alguna operación sobrepasara el límite superior permitido para los elementos del tipo numero, se dice que ocurre un overflow y si se encuentra por debajo del mínimo se dice que ocurre underflow.

Algunos lenguajes detectan el overflow y el underflow como error, otros lo ignoran convirtiendo el valor resultado en un valor válido dentro del rango. En el caso del DaVinci Concurrente se intenta asignar el valor 0 (cero)

Las relaciones de orden entre números son: igual, menor, mayor, menor o igual, mayor o igual y distinto. En la tabla 3.1 se indica la sintaxis de cada uno de ellos así como un ejemplo de aplicación.

Relación	Sintaxis	Ejemplo
Igualdad	=	A = B
Menor	<	3 * C < D
Mayor	>	C > 2 * B
Menor o igual	<=	A < (2 * C + 4)
Mayor o igual	>=	(B + C) > D
Distinto	<>	A <> B

Tabla 3.1: Relaciones de Orden para números

Tener en cuenta que en la tabla 3.1 **A, B, C y D son variables numéricas**.

Para declarar una variable numérica deberá utilizarse el sector de declaraciones.

En el ejemplo de contar las cuadras, para completar la declaración del dato cantidadCuadras que nos había quedado pendiente, ahora realizamos lo siguiente:

```
programa Cap2Ejemplo9
variables
  cantidadCuadras: numero
comenzar
  {sentencias}
fin
```

Donde **numero indica al lenguaje que la variable será de tipo numérico**, y esto implica que sus valores entrarán en el rango especificado anteriormente y que podrá utilizar cualquiera de las operaciones válidas para el **tipo numero**.

Dado que el robot es capaz de realizar cálculos con sus variables numéricas, veamos otro ejemplo donde se declaran dos variables numéricas:

```
programa numerico
variables
  nro1 : numero
```

```
  nro2 : numero
comenzar
  nro1 := 23              (1)
  nro2 := 30              (2)
  Informar( nro1 * nro2 )  (3)
  Informar( 25 / 3 )       (4)
fin
```

La línea marcada con (1) utiliza el operador de asignación que veremos en una sección posterior. El operador de asignación en el lenguaje DaVinci Concurrente se representa con un dos puntos igual (:=) y permite dar valor a una variable. En el caso de (1) le está dando el valor 23, y en el caso de (2) le está dando el valor 30. En la línea marcada con (3) se abrirá en pantalla una ventana mostrando el valor 690 (resultante de la instrucción Informar) y la línea (4) visualizará el valor 8. **Notemos que la división es entera por lo que la parte fraccionaria, sin importar su valor, será ignorada.**

En la mayoría de los programas que se resolverán a lo largo de este curso, el uso de las variables numéricas se verá restringido a operaciones sobre números enteros positivos como por ejemplo: informar cantidad de pasos dados, cantidad de flores recogidas, cantidad de veces que ocurrió un determinado hecho, etc.

3.4.2 Tipo de dato lógico (logico)

Este tipo de dato lógico puede tomar uno de los dos siguiente valores: Verdadero o Falso. En DaVini Concurrente están denotados por **V y F**. Si se utilizan variables de tipo logico se puede asignar en ellas el resultado de cualquier expresión lógica o relacional.

```
programa variablesLogicas
variables
  esPositivo : logico
  identicos  : logico
comenzar
  iniciar
  identicos := (PosCa = PosAv)
  esPositivo := (PosCa > 1)
  informar(“es positivo: “,esPositivo)
fin
```

En este ejemplo, la variable identicos guardará el valor V (verdadero) si el robot se encuentra ubicado en una esquina en la cual el valor de la avenida coincide con el valor de la calle y guardará F (falso) para cualquier otra situación. Del mismo modo, la variable esPositivo guardará el valor F si está ubicado sobre la calle 1 y en cualquier otro caso, guardará V.

Veamos el siguiente problema que permite ejemplificar el uso de este tipo de variables.

Ejemplo 3.1: El robot debe ir de (1,1) a (1,2) y debe informar si en (1,1) hay flor o no.

```
programa Cap3Ejemplo1
variables
  habiaFlor : logico
comenzar
  iniciar
  {registremos si hay flor o no en (1,1) }
  habiaFlor := HayFlorEnLaEsquina
  mover
  Informar(habiaFlor)
fin
```

En este caso la variable lógica permite registrar V o F según si en (1,1) hay flor o no. Luego, al posicionar el robot en (1,2) el valor almacenado es informado.

El objetivo de este sencillo ejemplo es mostrar la posibilidad de la variable lógica de guardar el resultado de la evaluación de una condición (en este caso HayFlorEnLaEsquina) para usarlo con posterioridad. Esto también **puede aplicarse a cualquier proposición ya sea atómica o molecular.**

Analicemos el siguiente ejemplo:

Ejemplo 3.2: Recoger todas las flores de la esquina (10,1). Si logra recoger al menos una flor, ir a (10,10) y vaciar de flores la bolsa; sino informar que no se ha recogido ninguna flor en (10,1).

En este caso no se requiere conocer la cantidad de flores recogidas. Sólo se necesita saber si se ha recogido alguna flor en (10,1) o no. Por lo tanto, en lugar de utilizar una variable numérica, se utilizará una variable booleana para representar esta situación.

Realizaremos un esquema del algoritmo a utilizar para este problema:

```
programa Cap3Ejemplo2
comenzar
   {antes de empezar, analizar y recordar si en (10,1) hay flor o no}
 {tomar todas las flores de (10,1)}
  si (originalmente en (10,1) había flor)
     {ir a (10,10) y vaciar la bolsa}
  sino
     {informar que no se recogió ninguna  flor}
fin
```

En el lenguaje del robot esto se escribe de la siguiente forma:

```
programa Cap3Ejemplo2
variables
  florEn10: logico
comenzar
  iniciar
  Pos(10,1)
  {antes de empezar....}
  florEn10 := HayFlorEnLaEsquina
  {tomar todas las flores de (10,1)}
```

```
  mientras(HayFlorEnLaEsquina)
    tomarFlor
  si(florEn10)
    comenzar
    {ir a (10,10 ) y vaciar la bolsa}
    Pos(10,10)
    mientras(HayFlorEnLaBolsa)
      depositarFlor
    fin
  sino
    {informar Falso indicando que no se recogió ninguna}
    Informar (florEn10)
fin
```

3.4.3 Tipo de dato alfanumérico (texto)

Con este tipo de datos se puede representar cualquier secuencia de caracteres que no supere la longitud de 2^{31}-1

Los siguientes son asignaciones de secuencias válidas, nótese que cada secuencia tendrá al principio y al final el carácter doble comilla **"**

cadena1 := "esto es una secuencia válida: "

cadena2 := "12345"

cadena3 := cadena1 + cadena2

La manipulación de valores de este tipo de datos se podrá realizar por medio del operador **suma** o funciones nativas.

El operador suma retorna la concatenación de los dos operandos, como se mostró en el ejemplo anterior (**cadena3**)

Las funciones que trae de manera nativa el intérprete son:

longitud	Retorna la longitud de un texto pasado como argumento a la función. Sintaxis: longitud(en cadena:texto) retorna numero Ejemplo: cantidad := longitud("Hola mundo…")
sustraer	Retorna una sub-cadena de la cadena original pasada como argumento. La sub-

	cadena será la que surja de la extracción según las posiciones definidas en los argumentos. Sintaxis: sustraer(en cadena:texto; en desde:numero; en hasta:numero) retorna texto Ejemplos: subcadena := sustraer("original", 2, 3) informar(sustraer("hola mundo",1,4))
numeroATexto	Retorna en forma de texto el número pasado como argumento, si número no es válido se produce un error. Sintaxis: numeroATexto (en valor:numero) retorna texto Ejemplos: cadena := numeroATexto(2+5) cadena := sustraer(numeroATexto(10*10+1),1,2)
textoANumero	Retorna en forma de número el texto pasado como argumento, si no se puede convertir se produce un error. Sintaxis: textoANumero(en cadena:texto) retorna numero Ejemplo: n := textoANumero("100")
logicoATexto	Retorna en forma de texto el valor lógico pasado como argumento. Sintaxis: logicoATexto (en val:logico) retorna texto

	Ejemplo: cadena := logicoATexto(v)

Tabla 3.1: funciones primitivas aplicadas al tipo texto

Las relaciones de orden entre texto son: igual, menor, mayor, menor o igual, mayor o igual y distinto. En la tabla 3.2 se indica la sintaxis de cada uno de ellos.

Relación	Sintaxis
Igualdad	=
Menor	<
Mayor	>
Menor o igual	<=
Mayor o igual	>=
Distinto	<>

Tabla 3.2: Relaciones de Orden para textos

Se pueden hacer comparaciones al igual de los números enteros teniendo presente que estas se hacen independientemente si están en minúsculas o mayúsculas, y se utiliza el orden alfabético.

3.4.4 Tipos de datos - Concurrencia

DaVinci Concurrente soporta el uso de otros tipos de datos que no serán utilizados en este curso.

Tipo de dato semáforo*:*

Es un tipo de dato abstracto para trabajar con problemas resolubles mediante algoritmos concurrentes.

Este tipo de datos semáforo pude ser definido de dos maneras, de tipo binario o de tipo general.

semáforo binario: Los semáforos de este tipo pueden tomar solamente los valores 0 (cero) o 1 (uno).

semáforos generales: Los semáforos de este tipo pueden tomar cualquier valor entero positivo incluido el cero.

Las únicas operaciones con las cuales se los pude manipular son: "iniciarSemaforo", "avisar" y "esperar".

3.5 Modificación de la información representada

Hasta ahora sólo se ha manifestado la necesidad de conocer expresamente cierta información. Para tal fin se ha asociado un nombre, también llamado identificador, a cada dato que se desee

representar.

Cada uno de estos identificadores será utilizado como un lugar para almacenar la información correspondiente. Por lo tanto, será necesario contar con la posibilidad de guardar un valor determinado y posteriormente modificarlo.

3.5.1 Asignación

Se utilizará como lo indicamos en los ejemplos de la sección anterior la siguiente notación:

Identificador := valor a guardar

El operador **:=** se denomina **operador de asignación** y permite registrar o "guardar" el valor que aparece a derecha del símbolo en el nombre que aparece a izquierda de dicho símbolo.

En el ejemplo del capítulo anterior al comenzar el recorrido debe indicarse que no se ha caminado ninguna cuadra. Para esto se utilizará la siguiente notación:

cantidadCuadras := 0

A partir de esta asignación el valor del dato cantidadCuadras representará (o contendrá) el valor 0.

En ocasiones será necesario recuperar el valor almacenado para ello se utilizará directamente el identificador correspondiente.

Por ejemplo, la siguiente instrucción permitirá informar el último valor asignado a cantidadCuadras.

Informar (cantidadCuadras)

También, puede utilizarse el operador de asignación para modificar el valor de un identificador tomando como base su valor anterior (el que fue almacenado previamente). Por ejemplo:

cantidadCuadras := cantidadCuadras + 1

Como se explicó anteriormente, el operador **:=** permite asignar el valor que aparece a la derecha del símbolo al identificador que está a la izquierda del mismo. Sin embargo, la expresión que ahora aparece a derecha no es un valor constante sino que debe ser evaluado ANTES de realizar la asignación. El lado derecho indica que debe recuperarse el valor almacenado en cantidadCuadras y luego incrementarlo en 1. El resultado de la suma será almacenado nuevamente en cantidadCuadras.

Utilizando lo antes expuesto el ejemplo del Capítulo 2 Ejemplo 9 se reescribe de la siguiente manera:

```
programa Cap2Ejemplo9
variables
  cantidadCuadras: numero
comenzar
  iniciar
  { Hasta ahora no se camino ninguna cuadra }
```

```
  cantidadCuadras := 0                                    (1)
  mientras (hayFlorEnLaEsquina)
    comenzar
    { Incrementar en 1 la cantidad de cuadras dadas }
    cantidadCuadras := cantidadCuadras + 1                (2)
    mover
    fin
  { Informar la cantidad de cuadras dadas}
  Informar(cantidadCuadras)                               (3)
fin
```

Notemos que la instrucción (1) se encuentra fuera de la iteración por lo que se ejecutará una única vez al comienzo del algoritmo. Esta asignación inicial también se denomina inicialización del identificador. La instrucción (2) se ejecuta cada vez que el robot avanza una cuadra. Su efecto es recuperar el último valor de cantidadCuadras, incrementarlo en 1 y volver a guardar este nuevo valor en cantidadCuadras. De esta forma, el último valor almacenado será una unidad mayor que el valor anterior.

Observemos la importancia que tiene para (2) la existencia de (1). La primera vez que (2) se ejecuta, necesita que cantidadCuadras tenga un valor asignado previamente a fin de poder realizar el incremento correctamente. Finalmente, (3) permitirá conocer la cantidad de cuadras recorridas una vez que el robot se detuvo.

3.5.2 Lectura

Otro forma de alterar el contenido de una variables es a través de la lectura, esto es, **pedirle al usuario** que se encuentra ejecutando el programa que ingrese un valor para asignarle a la variable.

La lectura de variables de DaVinci concurrente se realiza mediante la sentencia **pedir**.

En el ejemplo siguiente se le solicita al usuario que ingrese un valor, luego se moverá el robot la cantidad de veces que el usuario indicó.

Ej:

```
programa Cap4EjemploLectura
variables cantidadcuadras:numero
comenzar
      iniciar
      pedir(cantidadCuadras)

      repetir cantidadCuadras
            mover
fin
```

Es importante tener en cuenta que existen lenguajes dinámicos, lo que significa que el tipo de dato asociado a una variable se realiza en tiempo de ejecución, permitiendo de esta manera

almacenar cualquier valor.

DaVinci concurrente es un lenguaje **fuertemente tipado**. La verificación de tipos de datos asociados a las variables se realiza previa a la ejecución del programa.

Cuando se ejecuta la sentencia **pedir(**variable**)** el usuario podrá ingresar cualquier valor, luego el intérprete verificará si el valor ingresado se corresponde con el tipo de la variable. En caso de que el valor no cumpla con el rango y/o tipo el intérprete asignará el mismo valor que toman las variables al momento de inicializarse.

3.6 Ejemplos

Ejemplo 3.3: El robot debe recorrer la avenida 1 hasta encontrar una esquina con flor y papel. Al finalizar el recorrido se debe informar la cantidad de cuadras recorridas hasta encontrar dicha esquina. Suponga que la esquina seguro existe.

Para poder resolverlo es necesario identificar los datos u objetos que se desean representar a través del algoritmo. En este caso interesa conocer la cantidad de cuadras hechas, y por lo tanto, es preciso registrar la cantidad de cuadras que se van avanzando hasta encontrar la esquina buscada. El algoritmo tendría la siguiente forma:

```
programa Cap3Ejemplo3
variables
  cuadras: numero
comenzar
  iniciar
  cuadras:=0
  mientras ~(hayFlorEnLaEsquina) | ~(hayPapelEnLaEsquina)
    comenzar
    {anotar que se caminó una cuadra mas}
    cuadras:=cuadras+1
    mover
    fin
  Informar (cuadras)
fin
```

¿Cómo modifico el algoritmo anterior si la esquina puede no existir?

Ejemplo 3.4: Recoger e informar la cantidad de flores de la esquina (1,1).

En este caso interesa conocer la cantidad de flores de la esquina (1,1) y por lo tanto es preciso registrar la cantidad de flores que se van tomando hasta que la esquina queda sin flores. El algoritmo tendría la siguiente forma:

```
programa Cap3Ejemplo4
variables
```

```
  flores: numero
comenzar
  iniciar
  flores:=0
  mientras (hayFlorEnLaEsquina)
    comenzar
    tomarFlor
    {registramos que se tomó una flor}
    flores:= flores+1
    fin
  Informar (flores)
fin
```

Ejemplo 3.5: Contar e informar la cantidad de flores de la esquina (1,1).

En este caso sólo nos interesa saber la cantidad de flores de la esquina, a diferencia del ejercicio anterior en donde debíamos además recoger las flores de la esquina (es decir dejar la esquina sin flores). Por lo tanto, se debe tener en cuenta que para poder contar las flores vamos a tener que tomarlas, y una vez que tomamos todas las flores de la esquina, debemos volver a depositarlas para que no se modifique la cantidad original de flores de la esquina.

Inicialmente el algoritmo tendría la siguiente forma:

```
programa Cap3Ejemplo5
variables
  flores:numero
comenzar
  iniciar
  flores:=0
  mientras (hayFlorEnLaEsquina)
    comenzar
    tomarFlor
    {registramos que se tomó una flor}
    flores:= flores+1
    fin
  {***Debemos depositar las flores juntadas de la esquina***}
  Informar (flores)
fin
```

El programa se reescribirá de la siguiente manera:

```
programa Cap3Ejemplo5
variables
  flores:numero
comenzar
  iniciar
  flores:=0
  mientras (hayFlorEnLaEsquina)
    comenzar
    tomarFlor
    {registramos que se tomó una flor}
```

```
    flores:= flores+1
    fin
  { depositamos las flores juntadas de la esquina}
  repetir flores
    depositarFlor
  Informar (flores)
fin
```

¿Puede ocurrir que el valor de flores permanezca en cero?

¿Si el valor de flores es cero, que ocurre con el repetir?

¿Se puede reemplazar la estructura de repetición "repetir flores", por

mientras(HayFlorEnLaBolsa)
 depositarFlor"

3.7 Conclusiones

Hasta aquí se ha presentado la sintaxis de un algoritmo que utiliza datos para la solución de un problema. También se ha mostrado cómo crear, inicializar y modificar estos datos en una solución.

Contar con la posibilidad de representar información adicional al problema mejora la potencia de expresión del algoritmo, ya que los atributos de los objetos con los que opera podrán estar reflejados en su interior.

También vimos que para representar esta información el lenguaje DaVinci Concurrente nos provee dos tipos de datos: el tipo numérico y el tipo lógico.

Hasta aquí se han presentado los elementos que componen un algoritmo: el control y los datos. De todo lo visto, entonces, podemos concluir que un algoritmo es una secuencia de instrucciones que utiliza y modifica la información disponible con el objetivo de resolver un problema.

Nota: para realizar este apunte se utilizó (con el permiso correspondiente) el material del curso de Ingreso a la Facultad de Informática de la UNLP. En el mismo se han realizado algunos agregados y modificaciones.