

## Capítulo 2

# Algoritmos y Lógica

## Introducción al lenguaje del Robot



## Objetivos

En este capítulo se verán con mayor profundidad algunos de los conceptos utilizados anteriormente para la definición de algoritmos.

Además se introducirá el concepto de lenguajes de expresión de problemas y los tipos de lenguajes existentes. Se presenta el ambiente de programación del robot R-Info que tiene un lenguaje especial, con el que comenzaremos a trabajar en la resolución de problemas.

Este capítulo permitirá aplicar lo visto sobre estructuras de control, pero en el lenguaje previsto para el ambiente del robot R-Info.

Además se introducirán y repasarán algunos conceptos básicos de la lógica proposicional para representar condiciones complejas utilizadas en las estructuras del ambiente del robot R-Info, aplicadas específicamente a problemas con el robot.



## **Temas a tratar**

- ✓ Lenguajes de Expresión de Problemas. Tipos de Lenguajes. Sintaxis y semántica en un Lenguaje.
  - ✓ Ambiente de programación del robot R-info. Operaciones sobre R-info. Estructura general de un programa. Estilo de Programación. Ambiente de programación.
  - ✓ Estructuras de control en el ambiente de programación del robot R-info.
  - ✓ Revisión del tema: Proposiciones atómicas y moleculares, simbolización y tablas de verdad
  - ✓ Conectivos lógicos: Conjunción, Disyunción y Negación. Utilización del paréntesis.
  - ✓ Conclusiones
  - ✓ Ejercitación

## 2.1 Lenguajes de Expresión de Problemas. Tipos de Lenguajes. Sintaxis y semántica en un Lenguaje.

En el capítulo anterior se ha utilizado un lenguaje casi natural para especificar las instrucciones que debían llevarse a cabo. Esto, si bien facilita la escritura del algoritmo para quien debe decir cómo resolver el problema, dificulta la comprensión de dicha solución por parte de quien debe interpretarla.

En algunos de los ejemplos presentados hasta el momento, seguramente el lector debe haber tenido diferentes interpretaciones ¿Por qué?

Fundamentalmente porque el lenguaje natural tiene varios significados para una palabra (es ambiguo) y porque admite varias combinaciones para armar un enunciado. Estas dos condiciones son “indeseables” para un lenguaje de expresión de problemas utilizable en Informática.

En el ejemplo 1.7, del capítulo anterior:



- ¿Qué sucede si la lámpara está en el centro de la habitación y la escalera no es de dos hojas?
- ¿Dónde se asegura que se dispone de lámparas nuevas?
- ¿”Alcanzar la lámpara” equivale a “tomar la lámpara con la mano para poder girarla”? ¿Cuándo se deja la lámpara usada y se toma la nueva para el reemplazo?

Por medio de estas preguntas nos damos cuenta que el significado de cada instrucción del lenguaje debe ser exactamente conocido y como consecuencia no se pueden admitir diferentes interpretaciones.

Un lenguaje de expresión de problemas contiene un conjunto finito y preciso de instrucciones o primitivas utilizables para especificar la solución buscada.

Se puede notar, que desde el punto de vista del diseño del algoritmo, el contar con un número finito de instrucciones posibles termina con el problema de decidir, de una forma totalmente subjetiva, el grado de detalle necesario para que los pasos a seguir puedan ser interpretados correctamente. El conjunto de instrucciones determinará cuales son los pasos elementales posibles que se utilizarán para el diseño de la solución.

Un lenguaje de expresión de problemas debe reunir las siguientes características:

- Debe estar formado por un número de instrucciones finito.
- Debe ser completo, es decir que todas las acciones de interés deben poder expresarse con dicho conjunto de instrucciones.
- Cada instrucción debe tener un significado (efecto) preciso.

- Cada instrucción debe escribirse de modo único.

### 2.1.1 Tipos de Lenguajes

No siempre los problemas se expresan con primitivas que representen un subconjunto preciso del lenguaje natural: se puede utilizar un sistema de símbolos gráficos (tales como los de los diagramas de flujo que se observarán en algunos textos de Informática), puede emplearse una simbología puramente matemática, puede crearse un lenguaje especial orientado a una aplicación, pueden combinarse gráficas con texto, etc.

De todos modos, cualquiera sea la forma del lenguaje elegido éste siempre respetará las características mencionadas anteriormente ¿Por qué?

Porque si se quiere que una máquina interprete y ejecute las órdenes del lenguaje, por más sofisticada que ella sea, requerirá que las órdenes diferentes constituyan un conjunto finito, que cada orden pueda ser interpretada de un modo único y que los problemas solubles por la máquina sean expresables en el lenguaje.

### 2.1.2 Sintaxis y Semántica en un Lenguaje

La forma en que se debe escribir cada instrucción de un lenguaje y las reglas generales de expresión de un problema completo en un lenguaje constituyen su sintaxis.

Por ejemplo hay lenguajes que en su sintaxis tienen reglas tales como:

- Indicar el comienzo y fin del algoritmo con palabras especiales.
- Indicar el fin de cada instrucción con un separador (por ejemplo punto y coma).
- Encerrar, entre palabras clave, bloques de acciones comunes a una situación del problema (por ejemplo todo lo que hay que hacer cuando la condición es verdadera dentro de la estructura de control de selección).
- Indentar adecuadamente las instrucciones.

El significado de cada instrucción del lenguaje y el significado global de determinados símbolos del lenguaje constituyen su semántica.

Dado que cada instrucción debe tener un significado preciso, la semántica de cada instrucción es una indicación exacta del efecto de dicha instrucción. Por ejemplo ¿Cuál sería el efecto de una instrucción del siguiente tipo:

si (condición)

.....

sino

.....

como la vista en el Capítulo 1?

El efecto sería:

1. Evaluar la condición.
2. Si la condición es Verdadera, realizar las acciones indicadas antes del **sino**.



3. Si la condición es Falsa realizar las acciones indentadas indicadas a continuación del sino.

## 2.2 Ambiente de programación del robot (R-info). Operaciones sobre R-info. Estructura general de un programa. Estilo de programación. Ambiente de programación.

A lo largo de este curso se trabajará con una máquina abstracta simple, un único robot móvil llamado **R-info**, controlado por un conjunto reducido de primitivas que permiten modelizar recorridos y acciones en una ciudad compuesta por calles y avenidas.

Una consideración importante que se debe hacer en este momento, es que este ambiente de programación permite abordar otros conceptos y operaciones que los que se presentan como contenidos de este curso. Por ejemplo, el ambiente permite declarar varios robots R-info que se desplazan por la ciudad (con diferentes características) y que serán utilizados para explicar los conceptos básicos de la programación concurrente y paralela. Estos conceptos se verán en asignaturas de los años superiores de la carrera.

**En resumen, en este curso utilizaremos un único robot R-info que se desplazará en una única área de una ciudad compuesta por 100 avenidas y 100 calles.**

El robot **R-info** que se utiliza posee las siguientes capacidades básicas:

1. Se mueve.
2. Se orienta hacia la derecha, es decir, gira 90 grados en el sentido de las agujas del reloj.
3. Dispone de sensores visuales que le permiten reconocer dos formas de objetos preestablecidas: flores y papeles. Los mismos se hallan ubicados en las esquinas de la ciudad.
4. Lleva consigo una bolsa donde puede transportar flores y papeles. Está capacitado para recoger y/o depositar cualquiera de los dos tipos de objetos en una esquina, pero de a uno a la vez. La bolsa posee capacidad ilimitada.
5. Puede realizar cálculos simples.
6. Puede informar los resultados obtenidos.

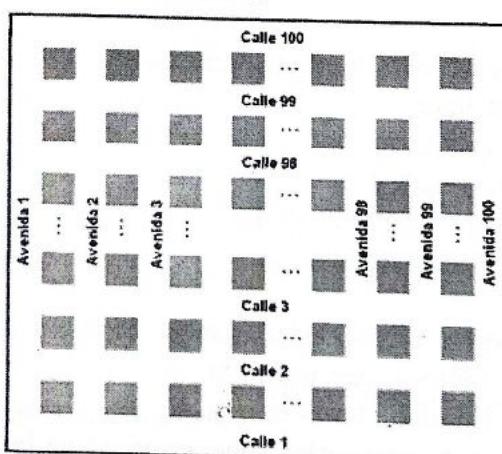


Figura 2.1: La ciudad del robot



La ciudad, en la que **R-info** se desplaza, está formada por calles y avenidas. Se denominan avenidas a las arterias verticales y calles a las arterias horizontales.

Como lo muestra la figura 2.1, la ciudad está formada por 100 avenidas y 100 calles.

Cada una de las esquinas está determinada por la intersección de una avenida y una calle. Debe considerarse a las arterias como rectas y a la esquina como el punto de intersección entre dichas rectas. La esquina se representará por dos coordenadas: la primera indicará el número de avenida y la segunda el número de calle. Por ejemplo, la esquina (2,4) es la intersección de la avenida 2 y la calle 4.

Las flores y los papeles se encuentran siempre en las esquinas. Pueden existir varias flores y varios papeles en cada esquina. En la búsqueda de reducir el problema del mundo real a los aspectos básicos que debe cubrir el robot en el ambiente, se han realizado las siguientes abstracciones:

- La ciudad queda reducida a un ámbito cuadrado de 100 calles y 100 avenidas;
- El andar del robot queda asociado con un paso que equivale a una cuadra de recorrido;
- Se reducen los datos en el modelo para tratar sólo con flores y papeles;
- Se aceptan convenciones (el robot solo inicia sus recorridos en la posición (1,1) de la ciudad);
- Se supone que el robot ve y reconoce las flores y los papeles. No es de interés de este curso discutir cómo realiza ese reconocimiento.

Es interesante analizar el grado de exactitud del modelo y su relación con los objetivos a cumplir. Está claro que en este ejemplo no se modeliza exactamente la ciudad ni los objetos que están en ella. Tampoco se representa adecuadamente el movimiento de un robot real, que posiblemente tenga que dar varios pasos para recorrer una cuadra. Se ignoran los detalles del proceso de reconocimiento de los objetos e incluso no se considera la posibilidad de que el robot confunda objetos.

Sin embargo, dado que el objetivo planteado es escribir programas que permitan representar recorridos con acciones simples (contar, limpiar, depositar) el modelo esencial es suficiente y funciona correctamente.

## 2.2.1 Operaciones en el ambiente del robot R-info

El conjunto de acciones que **R-info** puede realizar es muy reducido. Cada una de estas acciones corresponde a una instrucción, entendible por él y que debe tener un modo único de expresión, para que la máquina la interprete correctamente; y un significado único, a fin de poder verificar que el resultado final de la tarea se corresponde con lo requerido. De esta manera se desplaza, toma, deposita, evalúa algunas condiciones sencillas y puede visualizar información.

Este conjunto de instrucciones elementales que se detallan en la tabla 2.1 permite escribir programas con un objetivo bien definido, que tendrán una interpretación y una ejecución única por **R-info**. En dicha tabla se indica para cada instrucción su sintaxis, es decir, cómo debe escribirse, y su semántica, esto es cómo se interpreta esa orden en el lenguaje de **R-info**.



Sintaxis	Semántica
Iniciar (robot, posición)	Instrucción primitiva que posiciona al robot en la esquina indicada orientado hacia el norte. <b>En este curso siempre debemos posicionar al robot en la esquina (1,1) para comenzar su ejecución.</b>
derecha	Instrucción primitiva que cambia la orientación del robot en 90° en sentido horario respecto de la orientación actual.
mover	Instrucción primitiva que conduce al robot de la esquina en la que se encuentra a la siguiente, respetando la dirección en la que está orientado. Es responsabilidad del programador que esta instrucción sea ejecutada dentro de los límites de la ciudad. En caso contrario se producirá un error y el programa será abortado.
tomarFlor	Instrucción primitiva que le permite al robot recoger una flor de la esquina en la que se encuentra y ponerla en su bolsa. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos una flor en dicha esquina. En caso contrario se producirá un error y el programa será abortado.
tomarPapel	Instrucción primitiva que le permite al robot recoger un papel de la esquina en la que se encuentra y ponerlo en su bolsa. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos un papel en dicha esquina. En caso contrario se producirá un error y el programa será abortado.
depositarFlor	Instrucción primitiva que le permite al robot depositar una flor de su bolsa en la esquina en la que se encuentra. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos una flor en dicha bolsa. En caso contrario se producirá un error y el programa será abortado.
depositarPapel	Instrucción primitiva que le permite al robot depositar un papel de su bolsa en la esquina en la que se encuentra. Es responsabilidad del programador que esta instrucción sea ejecutada solo cuando haya al menos un papel en dicha bolsa. En caso contrario se producirá un error y el programa será abortado.
PosAv	Identificador que representa el número de avenida en la que el robot está actualmente posicionado. Su valor es un número entero en el rango 1..100 y no puede ser modificado por el programador.
PosCa	Identificador que representa el número de calle en la que el robot está actualmente posicionado. Su valor es un número entero en el rango 1..100 y no puede ser modificado por el programador.
HayFlorEnLaEsquina	Proposición atómica cuyo valor es V si hay al menos una flor en la esquina en la que el robot está actualmente posicionado, ó F en caso contrario. Su valor no puede ser modificado por el programador.
HayPapelEnLaEsquina	Proposición atómica cuyo valor es V si hay al menos un papel en la esquina en la que el robot está actualmente posicionado, ó F en caso contrario. Su valor no puede ser modificado por el programador.
HayFlorEnLaBolsa	Proposición atómica cuyo valor es V si hay al menos una flor en la bolsa del robot, ó F en caso contrario. Su valor no puede ser modificado por el programador.
HayPapelEnLaBolsa	Proposición atómica cuyo valor es V si hay al menos un papel en la bolsa del robot, ó F en caso contrario. <i>Su valor no puede ser modificado por el programador.</i>

Pos	Instrucción que requiere dos valores Av y Ca, cada uno de ellos en el rango 1..100, y posiciona al robot en la esquina determinada por el par (Av,Ca) sin modificar la orientación del robot.
Informar	Instrucción que permite visualizar en pantalla el contenido almacenado en alguna variable.

Tabla 2.1: Sintaxis del robot R-info

Es importante remarcar que la sintaxis en este lenguaje es sensible a mayúsculas y minúsculas. No es lo mismo escribir “depositarFlor” que “DepositarFlor” ó “depositarflor”. De las tres formas anteriores sólo “depositarFlor” es correcta.



Haciendo clic en el siguiente link bajar el *Ambiente de R-info* (el archivo está comprimido): [Ambiente R-info](#)

## 2.2.2 Estructura general de un programa

Un programa escrito en el lenguaje del robot comienza con la palabra clave **programa**, la cual debe estar seguida por un identificador que determina el nombre del programa.

El cuerpo del programa principal es una secuencia de sentencias, delimitada por las palabras claves **comenzar** y **fin**.

Dentro del programa se debe realizar un conjunto de declaraciones antes de comenzar a escribir el código propiamente dicho para el problema que se quiere resolver.

1. Inicialmente se dispondrá de un sector para declarar las diferentes áreas que se pueden utilizar (en este curso sólo se declarará un área que comprende la ciudad de 100 avenidas y 100 calles)
2. luego se deben declarar los diferentes tipos de robot que se desea utilizar para resolver cada problema y que estarán desplazándose por la ciudad junto al conjunto de instrucciones que cada tipo de robot debe utilizar (en este curso sólo se declarará un tipo de robot),
3. a continuación habrá otro espacio asignado para los módulos (se verá en capítulos sucesivos) y
4. por último antes de comenzar con el programa se indicarán las variables que se asocian a cada tipo de robot declarado previamente (para este curso en esta área sólo existirá la declaración de un robot).
5. Finalmente, entre las palabras **comenzar** y **fin** se escribe el código que indica en cual área se puede mover el robot (en este curso será la ciudad completa) y una instrucción que indica que el robot comienza a ejecutar las órdenes definidas.

Resumiendo lo explicado anteriormente, la estructura de un programa es la siguiente:

```
programa nombre_del_programa
areas
    se declara una única área que comprende toda la ciudad
robots
    se declara un único tipo de robot "robot1", junto al código correspondiente al
    programa que se quiere realizar
variables
    se declara una variable que representa al robot, será llamada R-info
comenzar
    se asigna el área donde se desplazará R-info (en este curso toda la ciudad)
    se indica el inicio para que cada robot se ejecute (en este curso solo se indica el
    comienzo de ejecución de R-info).
fin
```

### 2.2.2.1 Comentarios Lógicos

Los problemas para poder ser resueltos, deben entenderse, es decir interpretarse adecuadamente. Esto requiere un enunciado preciso de los mismos.

A su vez las soluciones que se desarrollan y que se expresan en un lenguaje preciso y riguroso, también deben ser entendidas por otros. ¿Por qué?

- Porque no siempre se ejecuta la solución.
- Porque a veces se debe modificar la solución por un pequeño cambio del problema, y para esto se debe “leer” rápida y correctamente la solución anterior.
- Porque en ocasiones se comparan soluciones de diferentes programadores y se debe tratar de entenderlas en un tiempo razonable.

Para que las soluciones sean claras y legibles, se deben agregar comentarios aclaratorios adecuados.

Un comentario dentro de un algoritmo no representa ni un dato, ni una orden. Sin embargo quienes desarrollan sistemas informáticos coinciden en que un algoritmo adecuadamente escrito debería interpretarse sólo leyendo los comentarios que el autor intercala a medida que construye su solución.

El término comentario lógico se refiere a que no se debe escribir un texto libre, sino expresar en forma sintética la función de una instrucción o un bloque de instrucciones del algoritmo, tratando de reflejar la transformación de los datos que se logra como consecuencia de su ejecución.

Normalmente los comentarios se intercalan en el algoritmo con algún símbolo inicial que indique que se trata de un comentario. En el ambiente de programación de R-info los comentarios se indican por medio de llaves.

De la experiencia en la disciplina Informática se aprende que el mayor esfuerzo y costo asociado con los sistemas de software es su mantenimiento, es decir corregir y ajustar dinámicamente el algoritmo o programa inicial a nuevas situaciones. Para reducir el costo de este mantenimiento es fundamental documentar adecuadamente los desarrollos,



y un aspecto importante de esa documentación consiste en escribir comentarios lógicos adecuados a medida que se desarrolla la solución.

### 2.2.3 Estilo de programación

La programación en el ambiente de R-info utiliza ciertas reglas sintácticas adicionales las cuales son muy estrictas relacionadas con la indentación y el uso de mayúsculas y minúsculas.

Estas reglas, aunque puedan resultar algo incómodas para el programador, buscan formar en el estudiante un buen estilo de programación.

El objetivo de un estilo de programación es mejorar, en mayor grado, la legibilidad del código de forma tal que resulte sencillo entenderlo, modificarlo, adaptarlo y reusarlo, ayudando así a maximizar la productividad y minimizar el costo de desarrollo y mantenimiento.

Al escribir un programa se deben respetar las siguientes reglas de indentación:

- La palabra clave **programa** debe comenzar en la primer columna.
- Las palabras claves **comenzar** y **fin** de un programa deben comenzar en la misma columna que la palabra clave **programa**.
- Las sentencias del cuerpo del programa debe comenzar dos columnas más a la derecha que las palabras claves que lo delimitan: **comenzar** y **fin**.
- Las sentencias que pertenecen al cuerpo de una estructura de control deben comenzar dos columnas más a la derecha que la palabra clave que identifica a la estructura de control.
- La indentación es la única forma de indicar si una sentencia pertenece o no a la estructura de control en cuestión.

Por otro lado, todas las palabras claves definidas por el ambiente de programación del robot R-info, así como las primitivas y las estructuras de control, deben ser escritas siempre con letras minúsculas, excepto que su nombre esté compuesto por más de una palabra, en cuyo caso, de la segunda palabra en adelante, cada una comienza con mayúscula. Por ejemplo: tomarFlor, mientras, numero.

Por el contrario, las variables y los procesos del sistema deben comenzar cada palabra que compone su nombre con una letra mayúscula y las demás minúsculas. Por ejemplo: PosAv, HayFlorEnLaBolsa, Pos, Informar.

Se recomienda la utilización de comentarios lógicos adecuados que faciliten el seguimiento del algoritmo planteado.

### 2.2.4 Ambiente de programación

Se denomina ambiente de programación a la herramienta que permite cubrir las distintas etapas en el desarrollo de un programa, que van desde la codificación del algoritmo en un lenguaje de programación hasta su ejecución, a fin de obtener los resultados esperados.

Cada ambiente de programación trabaja sobre un lenguaje específico. En particular, el ambiente de R-info utiliza la sintaxis del robot descripta previamente. Para codificar un algoritmo en el lenguaje del robot R-info es necesario realizar las siguientes tres etapas:

1. Escribir el programa: se escriben los pasos a seguir utilizando la sintaxis descripta.
2. Compilar el programa: para lograr que la computadora ejecute el programa escrito en la etapa anterior es necesario traducirlo a un lenguaje que la computadora comprenda. Esta etapa de denomina compilación y permite detectar los errores de sintaxis.
3. Ejecutar el programa: una vez que el programa ha sido compilado, puede ejecutarse. El ambiente de programación del robot R-info permite visualizar durante la ejecución, el recorrido que realiza el robot R-info dentro de la ciudad.

A continuación se describe el funcionamiento del ambiente a fin de poder mostrar cada una de las etapas mencionadas anteriormente. En la figura 2.2 se muestra la pantalla inicial del ambiente de programación del robot R-info.

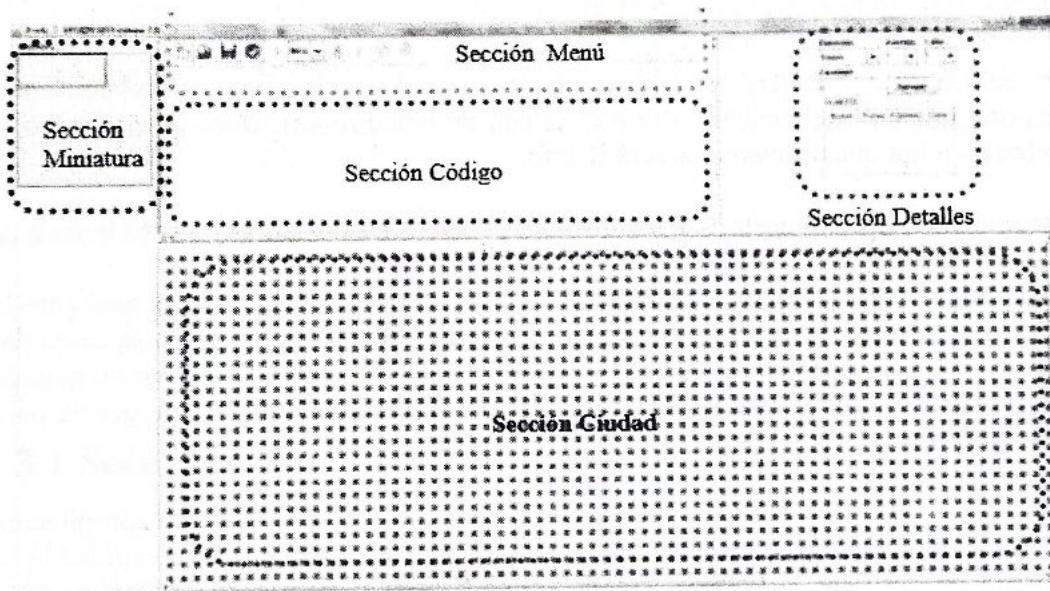


Figura 2.2: Ambiente de programación de R-info

Como se puede apreciar en la figura 2.2, este ambiente está dividido en cinco secciones: Sección Miniatura, Sección Menú, Sección Código, Sección Ciudad, y por último Sección Detalles.

**Sección Miniatura:** en esta sección se visualiza un cuadrado que representa la ciudad con sus avenidas y calles y un rectángulo más pequeño que indica que parte de la ciudad se está visualizando en la Sección Ciudad. Desplazando este rectángulo dentro del cuadrado podrás observar distintas avenidas y calles de la ciudad. En este curso nuestra ciudad estará compuesta por 100 avenidas y 100 calles.



**Sección Menú:** en esta sección se encuentra el conjunto de opciones que se pueden realizar. Entre las más utilizadas están: crear un nuevo programa, abrir un programa ya escrito, y guardar un programa, compilar y ejecutar un programa hecho. A medida que se utilice el ambiente de programación se podrá observar que existen otras operaciones, aunque las anteriores mencionadas seguramente son las únicas que se usarán en este curso.

**Sección Código:** en esta sección se visualiza el código correspondiente al programa con el que se está trabajando; puede ser un programa ya escrito o uno que se esté escribiendo en ese momento.

**Sección Detalles:** en esta sección se puede observar la información relevante al programa con el que se está trabajando, como la cantidad de flores y papeles de las esquinas y los robots que se encuentran en la ciudad (en este curso utilizaremos un solo robot R-info).

**Sección Ciudad:** en esta sección se puede observar el funcionamiento del programa a medida que ejecuta las instrucciones del mismo, una vez cumplida la etapa de compilación.

## 2.2.5 Comenzando a trabajar

Para comenzar a trabajar, se debe empezar a escribir el código del robot R-info (Recordar que en este curso sólo se trabaja con un único robot). Durante este curso el nombre de robot que utilizaremos será R-info.

El programa está compuesto por las instrucciones explicadas anteriormente junto a la sintaxis ya presentada.

Si bien no es imprescindible, se recomienda salvar el programa ingresado mediante la opción “Guardar” de la Sección Menú. Allí se deberá indicar el nombre que se desea dar al programa. Los programas que se ejecutan dentro del ambiente de programación poseen extensión *.lme*. Esta acción permitirá posteriormente editar el programa para volver a utilizarlo.

Luego de escrito el programa, es necesario realizar el proceso de compilación seleccionando la opción “Compilar” de la Sección Menú. El proceso de compilación se encargará de verificar la sintaxis del programa escrito y en caso de existir errores, visualizará los mensajes correspondientes.

Posteriormente, el programa ha sido correctamente escrito, puede ejecutarse mediante la opción “Ejecutar” de la Sección Menú. En la Sección Ciudad es posible ver cómo el robot R-info efectúa el recorrido indicado.

Es posible que se desee indicar la cantidad inicial de flores y papeles tanto para la ciudad como para la bolsa del robot R-info. Esto es posible modificando los valores correspondientes en la Sección Detalles. Además, en la misma sección se puede cambiar el color (rojo) asignado por defecto al robot R-info. La nueva configuración se hará efectiva solo cuando el programa se ejecute nuevamente.

Teniendo en cuenta todos los conceptos que hemos visto hasta este momento, la estructura de un programa que tiene un robot llamado R-info, el cual debe caminar dos cuadras a partir de la esquina (1,1) sería la siguiente:

```
programa Cap2Completo
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    mover
    mover
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Durante este curso, debes declarar una única área ciudad que tenga 100 avenidas y 100 calles

Durante este curso, utilizaremos un solo robot

Durante este curso, debes declarar una única variable de tipo robot llamada R-info

Durante este curso, el robot R-info sólo podrá desplazarse en el área que comprende toda la ciudad

Se indica la esquina inicial desde donde el robot comienza su ejecución. Durante este curso será siempre desde la esquina (1,1)

Un punto importante a tener en cuenta cuando se desarrollen los programas es que durante este curso sólo se debe modificar el código correspondiente al único robot R-info dependiendo de las acciones que se quieren realizar. Es decir, no deben definirse nuevas áreas ni robots como tampoco cambiar el tamaño de la ciudad.

## 2.3 Estructuras de Control

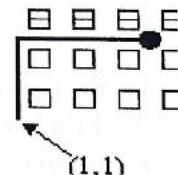
En esta sección se detallará la sintaxis correspondiente a las estructuras de control utilizadas por el robot R-info. El funcionamiento de cada una de esas estructuras se explicó en forma detallada en el capítulo 1.

### 2.3.1 Secuencia

Está definida por un conjunto de instrucciones que se ejecutarán una a continuación de otra.

**Ejemplo 2.1:** Programé al robot para que camine desde (1,1) a (1,3) y desde allí a (4,3).

```
programa Cap2Ejemplo1
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    mover
    mover
    derecha
    mover
    mover
    mover
fin
variables
```



```
R-info: robot1
comenzar
  AsignarArea(R-info, ciudad)
  Iniciar(R-info, 1, 1)
fin
```

La instrucción *Iniciar* ubica al robot R-info en la esquina (1,1) orientado hacia el norte (hacia arriba). Luego debe comenzar la ejecución de las instrucciones correspondientes al robot R-info. Por lo tanto, avanza dos cuadras en línea recta por lo que queda posicionado en la calle 3. Dobra a la derecha para seguir avanzando por la calle 3 y camina tres cuadras por lo que finaliza el recorrido parado en (4,3). R-info queda mirando hacia el este.

Note que no existe en el lenguaje del robot una instrucción que permita detenerlo. Esto ocurrirá naturalmente al terminar el programa, es decir cuando encuentra la palabra *fin* correspondiente al programa.

**Ejemplo 2.2:** Programe al robot para que recorra la avenida 4 desde la calle 3 hasta la calle 6. Al finalizar debe informar en qué esquina quedó parado.

```
programa Cap2Ejemplo2
  areas
    ciudad: AreaC(1,1,100,100)
  robots
    robot robot1
  comenzar
    Pos(4, 3)
    mover
    mover
    mover
    Informar(PosAv, PosCa)
  fin
  variables
    R-info: robot1
  comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1, 1)
  fin
```

La instrucción *Pos(4,3)* permite que el robot R-info “salte” desde (1,1) hasta (4,3). A partir de allí, camina tres cuadras en línea recta, realizando el recorrido solicitado. Al terminar el programa el robot quedará ubicado en la esquina (4,6). La instrucción *Informar(PosAv, PosCa)* muestra los valores retornados por las instrucciones *PosAv* y *PosCa*.



- Programe al robot para que recorra la calle 6 desde la avenida 11 a la avenida 13.
- Programe al robot para que recorra la avenida 17 desde la calle 31 hasta la calle 25.

### 2.3.2 Selección

Esta estructura permite al robot seleccionar una de dos alternativas posibles. La sintaxis es la siguiente:

**si** (condición)

    acción o bloque de acciones a realizar en caso de que la condición sea verdadera

**sino**

    acción o bloque de acciones a realizar en caso de que la condición sea falsa

Con respecto a la indentación necesaria para identificar las acciones a realizar en cada caso, se utilizarán dos posiciones a partir del margen izquierdo como puede apreciarse en los ejemplos que aparecen a continuación.

**Ejemplo 2.3:** Programe al robot para que recorra la calle 1 desde la avenida 1 a la 2 depositando, si puede, una flor en cada esquina. Además debe informar el número de avenida en las que no haya podido depositar la flor.

```
programa Cap2Ejemplo3
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    derecha
    si HayFlorEnLaBolsa      {Evalúa la primera esquina}
        depositarFlor
    sino
        Informar(PosAv)
    mover
    si HayFlorEnLaBolsa      {Evalúa la segunda esquina}
        depositarFlor
    sino
        Informar(PosAv)
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Notemos que es la indentación la que permite reconocer que la instrucción *mover* no pertenece a la primera selección. Al terminar el recorrido el robot quedará parado en (2,1). Además en caso de que haya flores en la bolsa, el robot ha depositado una sola flor en cada esquina.

En caso de no necesitar realizar acciones cuando la condición es falsa, puede omitirse la palabra *sino* junto con las instrucciones correspondientes; por lo que la sintaxis a utilizar sería la siguiente:

**si** (condición)

    acción o bloque de acciones a realizar en caso de que la condición sea verdadera

**Ejemplo 2.4:** Programe al robot para que recorra la avenida 15 desde la calle 12 a la calle 14 recogiendo, de ser posible, un papel en cada esquina.



```
programa Cap2Ejemplo4
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(15,12)
    si HayPapelEnLaEsquina
        tomarPapel
    mover
    si HayPapelEnLaEsquina
        tomarPapel
    mover
    si HayPapelEnLaEsquina
        tomarPapel
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```



- Programe al robot para que, si puede, deposite un papel en (1,2) y una flor en (1,3).
- Programe al robot para que intente recoger una flor de la esquina determinada por la calle 50 y la avenida 7. Solo si lo logra debe ir a la calle 51 y avenida 8 e intentar recoger allí otra flor. Al finalizar debe informar en qué esquina quedó parado.

### 2.3.3 Repetición

Cuando se desea realizar una acción o un conjunto de acciones un número fijo de veces, por ejemplo, N, se puede utilizar la siguiente estructura:

**repetir N**  
*acción o bloque de acciones a realizar*

Es importante remarcar que la cantidad de veces que se repite el bloque de acciones debe ser conocida de antemano. Una vez iniciada la repetición la ejecución no se detendrá hasta no haber ejecutado la cantidad de veces indicada por N, el conjunto de acciones indicado. Si se analizan con más detalle algunos de los ejemplos anteriores se verá que pueden resolverse utilizando una repetición.

**Ejemplo 2.5:** Programe al robot para que camine desde (1,1) a (1,3) y desde allí a (4,3).

Este problema fue resuelto utilizando una secuencia en el ejemplo 2.1. Ahora será implementado utilizando la repetición.

```
programa Cap2Ejemplo5
areas
    ciudad: AreaC(1,1,100,100)
robots
```

```
robot robot1
comenzar
    repetir 2
        mover
        derecha
    repetir 3
        mover
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Comparemos los programas Cap2Ejemplo1 y Cap2Ejemplo5 verificando que el recorrido realizado en ambos casos es el mismo. A continuación se muestra otra solución al problema planteado en el ejemplo 2.3 utilizando una repetición:

El Ejemplo 2.3 decía: Programe al robot para que recorra la calle 1 depositando, si puede, una flor en cada esquina. Además debe informar el número de avenida de aquellas esquinas en las que no haya podido depositar la flor.

```
programa Cap2Ejemplo6
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    derecha
    repetir 2
        si HayFlorEnLaBolsa
            depositarFlor
        sino
            Informar(PosAv)
        mover
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```



- ¿Por qué al finalizar el recorrido, el robot no queda posicionado en el mismo lugar que en el ejemplo 2.3?  
¿Perjudica en algo este hecho a lo que debe ser informado?

A continuación se muestra una variante del ejemplo 2.4:

**Ejemplo 2.7 (variante del 2.4):** Programe al robot para que recorra la avenida 15 desde la calle 12 a la 14 recogiendo, de ser posible, un papel en cada esquina.



```
programa Cap2Ejemplo7
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(15,12)
    repetir 2
        si HayPapelEnLaEsquina
            tomarPapel
        mover
        si HayPapelEnLaEsquina      (*)
            tomarPapel
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```



- ¿Las acciones realizadas por el robot en los programas Cap2Ejemplo4 y Cap2Ejemplo7 son iguales?
- ¿Es posible incluir la instrucción (\*) en la repetición? Si es así indique la manera de hacerlo y qué diferencias encuentra con el programa Cap2Ejemplo7.

**Ejemplo 2.8:** Programe al robot para que recorra la calle 4 dejando una flor en cada esquina.

Para resolver este problema es necesario analizar las 100 esquinas que forman la calle 4. El recorrido en el robot R-info será efectuado de la siguiente forma:

```
programa Cap2Ejemplo8
comenzar
    {ubicar al robot en (1,4) orientado hacia la derecha}
    {Recorrer las primeras 99 esquinas de la calle 4}
        {depositar la flor (solo si tiene)}
        {avanzar a la próxima esquina}
    {Falta ver si se puede depositar la flor en la esquina (1,100)}
fin
```

En la sintaxis del ambiente del robot R-info, este algoritmo se traduce en el siguiente programa:

```
programa Cap2Ejemplo8
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    derecha
    Pos(1,4)          {ubicar al robot en (1,4) orientado hacia la derecha}
```

```

repetir 99           {recorrer las primeras 99 esquinas de la calle 4}
    si HayFlorEnLaBolsa
        depositarFlor
        mover
    si HayFlorEnLaBolsa      {falta la esquina (1,100)}
        depositarFlor
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

Se puede notar que la sentencia *Pos(1,4)* no modifica la orientación del robot R-info. En todos los casos la indentación es lo que permite definir los bloques de instrucciones. Por ejemplo, la última selección no pertenece a la repetición. Esto queda claramente indicado al darle a ambas estructuras de control la misma indentación (el mismo margen izquierdo).

### 2.3.4 Iteración

Cuando la cantidad de veces que debe ejecutarse una acción o bloque de acciones depende del valor de verdad de una condición, puede utilizarse la siguiente estructura de control.

**mientras** (condición)

*acción o bloque de acciones a realizar mientras la condición sea verdadera*

A continuación se muestran algunos ejemplos que permiten representar el funcionamiento de esta estructura.

**Ejemplo 2.9:** 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.

Se debe tener en cuenta que no se conoce la cantidad de cuadras a recorrer para poder llegar a la esquina buscada. Para resolver este recorrido es preciso inspeccionar las esquinas una a una hasta lograr hallar la que no tiene flores. La solución es la siguiente:

```

programa Cap2Ejemplo9
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(7,1)
    mientras HayFlorEnLaEsquina
        mover
        Informar(PosCa)
    fin
variables
    R-info: robot1

```



```
comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1, 1)
fin
```

En este último ejemplo, para visualizar el número de calle donde el robot quedó parado debe utilizarse *PosCa* ya que no es posible calcular el valor a priori. Note que la ubicación de las flores en las esquinas de la ciudad puede variar entre una ejecución y otra.

Además en el ejemplo se puede observar que cada vez que el robot evalúa la condición “*HayFlorEnLaEsquina*” avanza una cuadra, si la condición es verdadera.

**Ejemplo 2.10:** Programe al robot para que deposite en (1,1) todas las flores que lleva en su bolsa.

```
programa Cap2Ejemplo10
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    mientras HayFlorEnLaBolsa
        depositarFlor
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1, 1)
fin
```

**Ejemplo 2.11:** Programe al robot para que recoja todas las flores y todos los papeles de la esquina determinada por la calle 75 y avenida 3. El código en el ambiente del robot R-info sería de la forma:

```
programa Cap2Ejemplo11
comenzar
    {Ubicar al robot en la esquina que se quiere limpiar}
    {Recoger todas las flores}
    {Recoger todos los papeles}
fin
```

Dado que en una esquina no se conoce a priori la cantidad de flores y/o papeles que puede haber será necesario utilizar dos iteraciones: una para recoger las flores y otra para recoger los papeles, de la siguiente forma:

```
programa Cap2Ejemplo11
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(3,75)
    {Recoger todas las flores}
    mientras HayFlorEnLaEsquina
```



```
tomarFlor
{Recoger todos los papeles}
mientras HayPapelEnLaEsquina
    tomarPapel
fin
variables
R-info: robot1
comenzar
AsignarArea(R-info,ciudad)
Iniciar(R-info,1,1)
fin
```

**Ejemplo 2.12:** Programe al robot para que camine desde (4,2) hasta (4,4) y luego hasta (7,4). El código en el ambiente del robot R-info sería de la forma:

```
programa Cap2Ejemplo12
comenzar
{Posicionar al robot }
{Avanzar dos cuadras }
{Doblar a la derecha }
{Avanzar tres cuadras }
fin
```

Este algoritmo puede ser implementado de diferentes formas. Analice estas dos opciones:

<pre>programa Cap2Ejemplo12A areas ciudad: AreaC(1,1,100,100) robots robot robot1 comenzar Pos(4,2) repetir 2     mover derecha repetir 3     mover fin variables R-info: robot1 comenzar AsignarArea(R-info,ciudad) Iniciar(R-info,1,1) fin</pre>	<pre>programa Cap2Ejemplo12B areas ciudad: AreaC(1,1,100,100) robots robot robot1 comenzar Pos(4,2) mientras (PosCa&lt;=4)     mover derecha mientras (PosAv&lt;=6)     mover fin variables R-info: robot1 comenzar AsignarArea(R-info,ciudad) Iniciar(R-info,1,1) fin</pre>
--	--



- ¿El robot realiza la misma cantidad de pasos en ambos programas?
- ¿Cuál solución prefiere Ud.? Justifique su respuesta pensando en que ahora debe hacer que el robot camine desde (3,5) hasta (3,7) y desde allí hasta (6,7).

## Análisis de la sintaxis del robot

La tabla 2.2 resume la sintaxis del robot ejemplificada hasta el momento.

Sintaxis del robot	
Informar	HayFlorEnLaEsquina
mover	HayPapelEnLaEsquina
derecha	HayFlorEnLaBolsa
tomarFlor	HayPapelEnLaBolsa
tomarPapel	PosAv
depositarFlor	PosCa
depositarPapel	
Pos	

Tabla 2.2: Resumen de la sintaxis del robot



- ¿Qué función cumple la instrucción iniciar?
- ¿Por qué todos los programas del robot deben comenzar con esta instrucción?
- ¿Qué diferencia hay entre las instrucciones PosAv y PosCa y la instrucción Pos?
- Suponga que el robot se encuentra en (1,1) y se desea que salte a (3,4). ¿Es posible realizar las siguientes asignaciones para lograrlo?

PosAv := 3

PosCa := 4

Justifique su respuesta. Indique la manera correcta de resolver este problema.

- ¿Es posible para el robot depositar una flor sin verificar primero si tiene al menos una flor en su bolsa?
- ¿El robot está capacitado para decir si en una misma esquina hay varios papeles?

## 2.4 Proposiciones atómicas y moleculares, simbolización y tablas de verdad

Cuando se emplearon condiciones para definir las acciones a tomar en la selección y la iteración no se indicó la forma en que pueden combinarse. Como el lector habrá notado, todos los ejemplos desarrollados hasta el momento fueron lo suficientemente simples como para poder ser resueltos con una pregunta sencilla. Sin embargo, en un problema real, esto no es así y se requiere combinar expresiones para poder representar una situación a evaluar. Por este motivo se introducirán y repasarán algunos conceptos básicos de la lógica proposicional que permitirán clarificar este aspecto, aplicados específicamente a problemas con el robot.

Dos de las estructuras de control ya vistas, selección e iteración, requieren para su funcionamiento, la evaluación de una condición. Estas condiciones se corresponden con lo que en términos de lógica se conoce como proposiciones.

Una proposición es una expresión de la cual tiene sentido decir si es verdadera o falsa, o sea es posible asignarle un valor de verdad (verdadero o falso, pero no ambos).

### Ejemplos de proposiciones

$1 + 4 = 5$  (*Verdad*)

La Pampa es una nación. (*Falso*)

Un triángulo es menor que un círculo. (*No se le puede asignar un valor de verdad, por lo tanto no es proposición*)

El color azul vale menos que una sonrisa. (*ídem anterior*)

Hay una flor en la esquina (*será verdadero o falso dependiendo de si la flor se encuentra o no en la esquina*)

#### 2.4.1 Proposiciones atómicas y moleculares

En Lógica, el término atómico se utiliza con su significado habitual: “algo que no puede ser dividido nuevamente”.

Una proposición es considerada atómica si no puede ser descompuesta en otras proposiciones.

##### Ejemplos

La casa es roja.

Hoy es lunes.

He llegado al final del recorrido.

Estoy ubicado a 3 metros de altura.

Cuando en una expresión se unen varias proposiciones atómicas se forma una proposición molecular o compuesta. Dicha unión se realiza mediante conectivos lógicos o términos de enlace.

Estos términos de enlace son de gran importancia. Tanto es así, que se estudiarán algunas reglas muy precisas para el uso de esta clase de términos.

Los términos de enlace a utilizar son los siguientes: “y”, “o”, “no”. Los dos primeros se utilizan para conectar proposiciones atómicas; en tanto que el conectivo “no”, solamente se coloca frente a una proposición atómica.

##### Ejemplos

No es cierto que la luna esté hecha de queso verde.

La vaca está cansada y no dará leche.

Hace calor o hay mucha humedad.

Hay papel en la bolsa y hay papel en la esquina.

Resumiendo:

Una proposición es atómica si no tiene conectivos lógicos, en caso contrario es molecular.

## 2.4.2 Simbolización

Así como en Matemática se simbolizan las cantidades para facilitar el planteo y solución de problemas, también en este caso es importante simbolizar las proposiciones atómicas, las moleculares y los conectivos lógicos con el objeto de facilitar las operaciones.

Conejivo	Simbolización en el ambiente de programación de R-info
y	&
o	
no	~

Tabla 2.3: Conejivos lógicos o términos de enlace

Se utilizarán letras minúsculas para simbolizar las proposiciones atómicas.

### Ejemplos de simbolización de proposiciones atómicas

1. Ayer fue un día ventoso.

*Si se considera  $p$  = “ayer fue un día ventoso”, esta proposición puede ser simbolizada como:  $p$ .*

2. Ese pájaro vuela muy alto.

*Si se llama  $q$  = “ese pájaro vuela muy alto”, la proposición se simboliza como:  $q$ .*

3. PosCa < 100.

*Si se llama  $r$  = “PosCa < 100”, la proposición se simboliza como:  $r$ .*

A continuación se aplicará este mecanismo de simbolización a las proposiciones moleculares.

El proceso para simbolizarlas consiste en tres pasos:

1. Determinar cuáles son las proposiciones atómicas que la componen.
2. Simbolizar las proposiciones como se explicó anteriormente.
3. Unir las proposiciones con los conectivos ya vistos. Por tal motivo, debe definirse un símbolo para cada uno de los conectivos. La tabla 2.3 muestra la simbolización a utilizar en cada caso.

### Ejemplos de simbolización de proposiciones moleculares

- Juan es estudiante y es jugador de fútbol.

$p$  = “Juan es estudiante”

$q$  = “Juan es jugador de fútbol”

Simbolizando  $p \& q$

- No es cierto que PosCa = 100.

$p$  = “PosCa=100”

Simbolizando  $\sim p$

- Hay flor en la esquina y hay papel en la bolsa, o hay papel en la esquina.

$p = \text{"Hay flor en la esquina"}$

$q = \text{"hay papel en la bolsa"}$

$r = \text{"hay papel en la esquina"}$

Analicemos, para resolver correctamente el ejemplo anterior debe tenerse en cuenta la aparición de la coma, la cual separa las dos primeras proposiciones de la tercera. Cuando se obtiene la simbolización debe respetarse ese orden. Por lo tanto la simbolización sería:  $(p \& q) | r$

- No hay flor en la bolsa, pero hay flor en la esquina.

$p = \text{"hay flor en la bolsa"}$

$q = \text{"hay flor en la esquina"}$

Simbolizando  $(\sim p \& q)$

Notemos que la palabra *pero* actúa como el conectivo lógico “y”.

### 2.4.3 Tablas de verdad. Repaso

Como se explicó previamente, una proposición es una expresión de la cual tiene sentido decir si es verdadera o falsa.

Para poder analizar cualquier proposición molecular y determinar su valor de verdad, es usual hacerlo a través de lo que se conoce como tabla de verdad.

La tabla de verdad de una proposición molecular es, como su nombre lo indica, una tabla donde se muestran todas las combinaciones posibles de los valores de verdad de las proposiciones atómicas que la conforman.

#### 2.4.3.1 Conjunción. Tabla de verdad

Dadas dos proposiciones cualesquiera  $p$  y  $q$ , la proposición molecular  $p \& q$  representa la conjunción de  $p$  y  $q$ .

La conjunción de dos proposiciones es cierta únicamente en el caso en que ambas proposiciones lo sean.

Dadas dos proposiciones cualesquiera  $p$  y  $q$ , si ambas son verdaderas, entonces  $p \& q$ , que representa la conjunción de  $p$  y  $q$ , es verdadera. Cualquier otra combinación da como resultado una proposición molecular falsa. La tabla 2.4 representa la tabla de verdad de la conjunción  $p \& q$  utilizando las cuatro combinaciones posibles de valores de verdad para  $p$  y  $q$ . Por lo tanto, si  $p \& q$  es una proposición verdadera entonces  $p$  es verdadera y  $q$  también es verdadera.

En Lógica se pueden unir dos proposiciones cualesquiera para formar una conjunción. No se requiere que el contenido de una de ellas tenga relación con el contenido de la otra.

p	q	p & q
V	V	V
V	F	F
F	V	F
F	F	F

Tabla 2.4: Conjunción ( $p \& q$ )

*Ejemplos:*

6 es un número par y divisible por 3.

$p$  = “6 es un numero par”

$q$  = “6 es divisible por 3”

$p$  es verdadera y  $q$  también, por lo tanto  $p \& q$  es verdadera.

Suponiendo que el robot se encuentra situado en la esquina (1,1)

$p$  = “PosCa = 1”

$q$  = “PosAv = 2”

$p$  es verdadera y  $q$  es falsa, por lo tanto  $p \& q$  es falsa.

El siguiente ejemplo muestra la aplicación de la conjunción en un algoritmo:

**Ejemplo 2.13:** el robot debe depositar, de ser posible, una flor en la esquina (45,70) solamente si en la esquina no hay flores.

```
programa Cap2Ejemplo13
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(45,70)
    si ((HayFlorEnLaBolsa) & ~ (HayFlorEnLaEsquina))
        depositarFlor
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

La selección utiliza la conjunción de dos proposiciones que, como se explicó anteriormente, para ser verdadera necesitará que ambas proposiciones lo sean simultáneamente. Esto es, basta con que una de ellas sea falsa para que no se deposite una flor en la esquina.

#### 2.4.3.2 Disyunción. Tabla de verdad

Dadas dos proposiciones cualesquiera  $p$  y  $q$ , la proposición molecular  $p \mid q$  representa la disyunción de  $p$  y  $q$ .

La disyunción entre dos proposiciones es cierta cuando al menos una de dichas proposiciones lo es.

La disyunción utiliza el término de enlace “o” en su sentido incluyente. Esto significa que basta con que una de las dos proposiciones sea verdadera para que la disyunción sea verdadera.

Dadas dos proposiciones cualesquiera  $p$  y  $q$ , su disyunción,  $p \vee q$ , será falsa cuando ambas proposiciones sean falsas. La tabla 2.5 representa la tabla de verdad de la disyunción  $p \vee q$  utilizando las cuatro combinaciones posibles de valores de verdad para  $p$  y  $q$ .

$p$	$q$	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

Tabla 2.3: Disyunción ( $p \vee q$ )

### Ejemplos

2 es primo o es impar

$p$  = “2 es primo”

$q$  = “2 es impar”

$p$  es verdadera,  $q$  es falsa. Se deduce que  $p \vee q$  es verdadera

Suponiendo que el robot se encuentra situado en la esquina (1,1)

$p$  = “PosCa = 8”

$q$  = “PosAv = 2”

$p$  es falsa,  $q$  es falsa. Se deduce que  $p \vee q$  es falsa.

El siguiente problema puede resolverse aplicando la disyunción:

**Ejemplo 2.14:** el robot debe caminar desde la esquina (45,70) a la esquina (45,74) solamente en el caso que la esquina (45,70) no esté vacía, es decir, la esquina tiene al menos una flor o un papel.

```
programa Cap2Ejemplo14
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(45,70)
    si ((HayFlorEnLaEsquina) | (HayPapelEnLaEsquina))
        repetir 4
            mover
        fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

La selección utiliza la disyunción de dos proposiciones que, como se explicó anteriormente, para ser verdadera solo requiere que una de ellas sea verdadera. Note que



si la esquina (45,74) tiene flor y papel, el robot avanza hasta la esquina (45,74). La única forma de que el robot no avance es que la esquina esté vacía, sin flor ni papel.

### 2.4.3.3 Negación. Tabla de verdad

Dada una proposición  $p$ , su negación  $\sim p$ , permitirá obtener el valor de verdad opuesto. El valor de verdad de la negación de una proposición verdadera es falso y el valor de verdad de la negación de una proposición falsa es verdadero.

Dada una proposición  $p$ , la tabla 2.4 representa los posibles valores de verdad de dos valores de verdad posibles de  $p$ .

$p$	$\sim p$
V	F
F	V

Tabla 2.4: Negación ( $\sim p$ )

Ejemplos

$p$  = “El número 9 es divisible por 3”

*La proposición  $p$  es verdadera.*

La negación de  $p$  es:  $\sim p$  = “El número 9 no es divisible por 3”

*Se ve claramente que  $\sim p$  es falsa.*

Suponiendo que el robot se encuentra situado en la esquina (1,1)

$p$  = “PosCa=1”

*La proposición  $p$  es verdadera.*

La negación de  $p$  es:  $\sim p$  = “ $\sim$ PosCa = 1”.

*Se deduce que  $\sim p$  es falsa.*

**Ejemplo 2.15:** el robot debe recorrer la calle 1 hasta encontrar una flor que seguro existe.

```
programa Cap2Ejemplo15
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    derecha
    mientras ~ (HayFlorEnLaEsquina)
        mover
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Note que la iteración utiliza la negación de la proposición atómica “hay flor en la esquina”. De esta forma, el robot detendrá su recorrido cuando encuentre una flor.

## 2.4.4 Utilización del paréntesis

Es frecuente encontrar proposiciones que tienen más de un término de enlace. En estos casos, uno de los términos de enlace es el más importante, o el término dominante, porque actúa sobre toda la proposición.

El operador “ $\sim$ ” es el que tiene mayor prioridad, es decir, es el primero es evalúa; seguido por “ $\&$ ” y “ $|$ ”. Estos dos últimos poseen igual prioridad. Ante una expresión que utilice varias conjunciones y/o disyunciones, se evaluaran de izquierda a derecha.

Lo mismo ocurre en Matemática. Si se considera la siguiente cuenta: “ $2 + 3 * 5$ ”, el resultado final, como es fácil de deducir, es 17.

La primera operación que se resuelve es el producto entre 3 y 5, y luego a su resultado se le suma 2. Ocurre así porque el operando \* (por), igual que el operando / (dividido), se resuelve antes que el operando + (mas), o que el operando - (menos).

Dada la siguiente proposición  $p \& \sim q$ , primero se resuelve la negación y luego la conjunción.

En determinados casos se tiene la necesidad de alterar este orden natural de resolución. Al igual que en Matemática, el uso de paréntesis permite resolver este problema.

### Ejemplo

$\sim p | q$  es una disyunción entre  $\sim p$  y  $q$  en tanto que:

$\sim(p | q)$  es la negación de una disyunción.

Como se puede observar el uso del paréntesis altera el tipo de proposición que debe resolverse, en este caso podría ser tratada como una disyunción o una negación dependiendo del uso o no de los paréntesis.

## 2.5 Conclusiones

El uso de un lenguaje de programación elimina la ambigüedad que se produce al expresarse en lenguaje natural permitiendo que cada instrucción tenga una sintaxis y una semántica únicas.

Se ha presentado un lenguaje de programación específico para el robot R-info que permite escribir programas que pueden ser ejecutados en una computadora.

Además, en este capítulo, se ha realizado una breve introducción a los conceptos de lógica proposicional y se han presentado numerosos ejemplos que muestran la utilidad de las proposiciones en el funcionamiento de las estructuras de control. Como pudo observarse, las proposiciones moleculares, formadas a través de los conectivos lógicos,



poseen una mayor potencia de expresión ya que permiten definir claramente el rumbo de acción a seguir.

Por otra parte, estos ejemplos también muestran las distintas posibilidades de combinar las estructuras de control y encontrar soluciones más generales a los problemas que se han planteado.

Lograr adquirir el conocimiento y la habilidad para desarrollar algoritmos utilizando estas estructuras es una tarea que requiere práctica. El lector no debería desanimarse si inicialmente advierte cierta dificultad en su uso. Se requiere un cierto tiempo de maduración para poder expresarse a través de esta terminología. El mecanismo para poder lograr adquirir las habilidades necesarias se basa en la ejercitación por lo que se recomienda la resolución de los problemas que se plantean a continuación.

Algoritmo 2.1. Algoritmo para calcular el área de un cuadrado.

```
1. Entrada: Se pide la medida de un lado de un cuadrado.
2. Proceso: Se calcula el área multiplicando la medida del lado por sí misma.
3. Salida: Se imprime el resultado.
```

Algoritmo 2.2. Algoritmo para calcular el área de un rectángulo.

```
1. Entrada: Se pide la medida de la base y la medida de la altura de un rectángulo.
2. Proceso: Se calcula el área multiplicando la medida de la base por la medida de la altura.
3. Salida: Se imprime el resultado.
```

Algoritmo 2.3. Algoritmo para calcular el área de un triángulo.

```
1. Entrada: Se pide la medida de la base y la medida de la altura de un triángulo.
2. Proceso: Se calcula el área multiplicando la medida de la base por la medida de la altura y dividir el resultado entre 2.
3. Salida: Se imprime el resultado.
```

Algoritmo 2.4. Algoritmo para calcular el área de un cuadrado.

```
1. Entrada: Se pide la medida de un lado de un cuadrado.
2. Proceso: Se calcula el área multiplicando la medida del lado por sí misma.
3. Salida: Se imprime el resultado.
```



# Capítulo 3

## Datos



### Objetivos

Los problemas resueltos anteriormente sólo buscan representar un conjunto de acciones a tomar dependiendo de la situación actual. Este es el objetivo de las estructuras de control. A través de ellas puede decidirse, 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 (boolean).
- ✓ Esquema de un Programa en el ambiente para el robot R-info.
- ✓ Modificación de la información representada.
- ✓ Ejemplos.
- ✓ Comparaciones.
- ✓ Representación de más de un dato dentro del programa.
- ✓ Conclusiones.
- ✓ Ejercitación.

## 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, ejercicio 9.

**Ejemplo 2.9:** 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 segura existe.

```
programa Cap2Ejemplo9
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(7,1)
    mientras (HayFlorEnLaEsquina)
        mover
        Informar(PosCa )
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Si se quisiera saber cuántas cuadras recorrió R-info, se necesitaría 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 modelizan 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 concentraremos en representar las cuadras recorridas por el robot R-info, 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 de la ejecución del programa. Dicho valor representa un dato relevante para el problema a resolver.

### 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 (código del robot), su valor.

Como se dijo en los comienzos del capítulo 2, en el ambiente lenguaje del robot R-info, existe un área o lugar dentro de cada robot (en nuestro curso R-info) donde se declaran las variables que se puedan necesitar para resolver los diferentes programas.

Un punto importante a considerar es que en este ambiente no existe la posibilidad de declarar variables en el programa (exceptuando la declaración del robot R-info), por el contrario sólo se permite declarar variables dentro del código del robot.

Versión: 2 - Capítulo 3 - Datos

Para declarar una variable dentro del ambiente del robot R-info se debe elegir un nombre que la identifique y un tipo de datos al cual pertenece dicha variable. Los tipos existentes en este ambiente se explicarán más adelante. La sintaxis dentro del robot R-info para declarar una variable es la siguiente:

```
robots
  robot robot1
variables
  nombreVariable: tipoVariable → Variable que puede ser
  comenzar
  ...
fin
```

Teniendo en cuenta esto, el programa completo quedaría de la forma:

```
programa Cap3EjemploVariables
areas
  ciudad: AreaC(1,1,100,100)
robots
  robot robot1
variables
  nombreVariable: tipoVariable
comenzar
  ...
fin
variables
  R-info: robot1
comenzar
  AsignarArea(R-info,ciudad)
  Iniciar(R-info,1,1)
fin
```

Notemos que se ha incorporado una sección para la declaración de las variables dentro del código del robot. Esta sección se encuentra entre la palabra robot y la palabra comenzar.

Ahora quedan distinguidos dentro del robot dos sectores: un sector superior donde se declaran las variables de interés y un sector inferior donde se detallan las instrucciones necesarias para que el robot R-info resuelva el problema planteado.

Si en el ejemplo antes visto se quisiera contar la cantidad de cuadras recorridas hasta encontrar una esquina sin flor, entonces deberíamos modificar el código del robot R-info agregando en la zona de declaración de variables un dato que permita manejar y procesar esa información.

Tener en cuenta que la declaración de variables que se presenta a continuación en la línea indicada con (\*) es aún incompleta dado que nos quedan por ver algunos temas adicionales antes de escribirla correctamente.

Veamos cómo quedaría:



```
programa Cap2Ejemplo9
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cantidadCuadras: (*)
comenzar
    ...
fin
variables
R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

## 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 dos tipos de datos con los que trabaja en el ambiente de programación del robot R-info.

### 3.4.1 Tipo de dato numérico (número)

Los elementos de este tipo de dato son los números enteros, como por ejemplo:

-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 ambiente de programación del robot R-info, el número máximo que se puede representar es  $2^{31} = 2147483647$  y el mínimo es  $-2^{31} = -2147483647$ .

Las operaciones válidas para los números son: suma, resta, multiplicación y división entera. Estas operaciones en el ambiente de programación del robot R-info se simbolizan como: +, -, \*, / respectivamente.

Si el resultado de alguna operación sobrepasa 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.

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	$\leq$	$A \leq (2 * C + 4)$
Mayor o igual	$\geq$	$(B + C) \geq D$
Distinto	$\diamond$	$A \neq B$

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 dentro del robot.

En el ejemplo de contar las cuadras (programa cap2Ejemplo9), para completar la declaración del dato cantidadCuadras que nos había quedado pendiente, ahora realizamos lo siguiente:

```

programa Cap2Ejemplo9
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cantidadCuadras: numero
comenzar
    ...
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

Donde numero indica en el ambiente de programación del robot que la variable será de tipo numérico, y esto implica que sus valores entrarán en el rango especificado

Versión: 2 - Capítulo 3 –Datos  
anteriormente y que podrá utilizar cualquiera de las operaciones válidas para el tipo número.

Dado que el robot es capaz de realizar cálculos con sus variables numéricas, analicemos otro ejemplo donde se declaran dos variables numéricas, prestando especial atención a las líneas numeradas:

```
programa numerico
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    nro1: numero
    nro2: numero
comenzar
    nro1 := 23                      {1}
    nro2 := 30                      {2}
    Informar ( nro1 * nro2 )        {3}
    Informar ( 25 / 3 )             {4}
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

La línea marcada con (1) utiliza el operador de asignación que veremos en una sección posterior. El operador de asignación en el ambiente de programación del robot R-info 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 (2) 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 (boolean)

Este tipo de dato lógico puede tomar uno de los dos siguientes valores: Verdadero o Falso. En el ambiente de programación del robot R-info están denotados por V y F. Si

se utilizan variables de tipo booleano se puede asignar en ellas el resultado de cualquier expresión lógica o relacional.

En el ejemplo que se presenta a continuación, se utiliza una variable *identicos* que 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.

Analicemos la solución presentada para este ejemplo:

```
programa numvariablesLogicas
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    esPositivo: boolean
    identicos: boolean
comenzar
    identicos := (PosCa = PosAv)
    esPositivo := (PosCa > 1)
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Veamos el otro problema que permite ejemplificar el uso del tipo de dato boolean:

**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
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    habiaFlor: boolean
comenzar
    habiaFlor := HayFlorEnLaEsquina
    mover
    Informar(habiaFlor)
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
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 (11,1). 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 (11,1).

En este caso no se requiere conocer la cantidad de flores recogidas. Sólo se necesita saber si se ha recogido alguna flor en (11,1) 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 que se deberá incluir en el robot R-info:

```
programa Cap3Ejemplo2
comenzar
{antesde empezar analiza y recuerda si en (11,1) hay flor o no}
{tomar todas las flores de (11,1)}
si(originalmente en (11,1) 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
areas
ciudad: AreaC(1,1,100,100)
robots
robot robot1
variables
florEn11: boolean
comenzar
Pos(11,1)
{antesde empezaranalizar y recordar si en (11,1) hay flor o no}
florEn11 := HayFlorEnLaEsquina
{tomar todas las flores de (11,1)}
mientras HayFlorEnLaEsquina
tomarFlor
si florEn11
Pos(10,10)           {ir a (10,10) y vaciar la bolsa}
mientras HayFlorEnLaBolsa
depositarFlor
sino
{informar F, se indica que no se recogió ninguna}
Informar (florEn11)
fin
variables
R-info: robot1
comenzar
AsignarArea(R-info,ciudad)
Iniciar(R-info,1,1)
```

### 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.

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 Cap2Ejemplo9 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 cantidadCuadrasrepresentará (o contendrá) el valor 0.

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

Por ejemplo, la instrucción

**Informar** (cantidadCuadras)

permitirá informar el último valor asignado a 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 cantidadCuadrasy luego incrementarlo en 1. El resultado de la suma será almacenado nuevamente en cantidadCuadras.

Utilizando lo antes expuesto el ejemplo del Cap2Ejemplo9 se reescribe de la siguiente manera:

**programa** cap2Ejemplo9B

Versión: 2 - Capítulo 3 --Datos

```

areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cantidadCuadras: numero
comenzar
    {Hasta ahora no se caminó ninguna cuadra}
    cantidadCuadras:=0
    mientras (hayFlorEnLaEsquina)
        {Incrementar en 1 la cantidad de cuadras dadas}
        cantidadCuadras:= cantidadCuadras + 1
        mover
        Informar(cantidadCuadras)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
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.

## 3.6 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. Una solución posible en el ambiente de programación del robot R-info:

```

programa cap3Ejemplo3
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1

```

```

variables
    cuadras: numero
comenzar
    cuadras:=0
    mientras ~(HayFlorEnLaEsquina) | ~(HayPapelEnLaEsquina)
        {anotar que se caminó una cuadra mas}
        cuadras:=cuadras+1
    mover
    Informar (cuadras)
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

En (1) indicamos que no se ha recorrido ninguna cuadra hasta el momento (por eso se le asigna el valor cero a la variable cuadras). En (2), se indica que mientras no haya flor ó no haya papel en la esquina, el recorrido debe continuar, esto es, la iteración terminará cuando encuentre una esquina con flor y papel. En (3) indicamos que se ha recorrido una cuadra más. En (4) informamos el último valor que quedó almacenado en cuadras, lo cual representa la cantidad de cuadras hechas en el recorrido.



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

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

Para poder resolverlo es necesario identificar los datos u objetos que se desean representar en el algoritmo.

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
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    flores: numero
comenzar
    flores:=0
    mientras (HayFlorEnLaEsquina)
        tomarFlor {registramos que se tomó una flor}
        flores:= flores+1
    Informar(flores)
fin

```

Versión: 2 - Capítulo 3 -Datos

```

variables
  R-info: robot1
comenzar
  AsignarArea(R-info, ciudad)
  Iniciar(R-info, 1, 1)
fin

```

En (1) indicamos que no se ha recogido ninguna flor hasta el momento. En (2) indicamos que se ha recogido una nueva flor de la esquina. En (3) informamos el último valor que quedó almacenado en flores. Hay que recordar que el punto (2) y la instrucción tomarFlor están dentro de una estructura de control de iteración (mientras) y esto se repetirá hasta que no haya más flores en la esquina, con lo cual en cada vuelta de esta iteración se incrementará en uno el valor de la variable flores.

### 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 implementado en el ambiente del robot R-infotendría la siguiente forma:

```

programa cap3Ejemplo5
areas
  ciudad: AreaC(1,1,100,100)
robots
  robot robot1
variables
  flores: numero
comenzar
  flores:=0
  mientras (HayFlorEnLaEsquina)
    tomarFlor {registramos que se tomó una flor}
    flores:= flores+1
    {Debemos depositar las flores juntadas de la esquina} (1)
    Informar (flores)
  fin
variables
  R-info: robot1
comenzar
  AsignarArea(R-info, ciudad)
  Iniciar(R-info, 1, 1)
fin

```

En (1) debemos encontrar la forma de poder depositar todas las flores que juntamos de la esquina. Para esto podemos ver que si en la esquina, por ejemplo, hubo 5 flores, entonces la variable flores tiene el valor 5, por lo tanto la variable flores contiene la cantidad de flores que debo depositar en la esquina.

Versión: 2 - Capítulo 3 - Datos

Resumiendo, sabemos cuántas flores hay que depositar en la esquina, esta cantidad es la que ha quedado almacenada en la variable flores. El programa se reescribirá de la siguiente manera:

```

programa cap3Ejemplo4B
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    flores: numero
comenzar
    flores:=0
    mientras (HayFlorEnLaEsquina)
        tomarFlor {registramos que se tomó una flor} {1}
        flores:= flores+1
    {depositamos las flores juntadas de la esquina} {2}
    repetir flores
        depositarFlor
    Informar (flores) {3}
fin

variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

En (1) indicamos que no se ha recogido ninguna flor hasta el momento. En (2) indicamos que se ha recogido una nueva flor de la esquina.

Para que puedas comprender la instrucción (3), tenemos que tener en cuenta dos cuestiones. La primera es que en (3) se utiliza una estructura de control repetitiva para depositar la cantidad de flores. Esto se puede hacer ya que conocemos el valor que queremos usar en la estructura repetir, ese valor será el almacenado en flores. La segunda es que esta instrucción repetitiva debe ubicarse fuera del mientras, ya que de lo contrario, por cada flor que recoge el robot volvería a depositar y por lo tanto la estructura de control mientras nunca terminaría. Otro efecto negativo de poner el repetir dentro del mientras es que la variable flores quedaría finalmente con un valor incorrecto. En (4) informamos el valor que contiene la variable flores. En este punto debemos prestar atención que haber utilizado la variable flores en el repetir no implica que la misma se haya modificado, esto es así, porque la única forma de modificar el contenido de una variable es por medio del operador:=.



- ¿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 “mientrasHayFlorEnLaBolsadedepositarFlor”?

**Ejemplo 3.6:** Recoger todos los papeles de la esquina (1,1) e informar la cantidad de papeles recogidos sólo si se han recogido al menos 5 papeles.

En este caso nos interesa saber la cantidad de papeles de la esquina, a diferencia del ejercicio anterior. Una vez que el robot ha juntado todos los papeles, si la cantidad de papeles obtenida es mayor o igual a 5 (al menos 5), debemos informar la cantidad recogida.

En el ambiente de programación del robot R-info lo podríamos escribir de la siguiente manera:

```

programa cap3Ejemplo6
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    papeles: numero
comenzar
    papeles:=0
    mientras (hayPapelEnLaEsquina)
        tomarPapel
        {registrarlos que se tomo un papel}
        papeles:=papeles+1
    si (papeles >= 5)
        Informar (papeles) {1}
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

Debemos tener en cuenta que la instrucción (1) está dentro de una instrucción de selección y por lo tanto sólo se ejecuta si la condición es verdadera, es decir, si la variable papeles quedó en un valor  $\geq 5$ .

**Ejemplo 3.7:** Recoger e informar todos los papeles de la avenida 1.

En este caso nos interesa saber la cantidad de papeles de la avenida. Una vez que hemos juntado todos los papeles de la avenida, debemos informar la cantidad recogida. El algoritmo debería tener en cuenta:

- 1) Cuál es la estructura de control para recorrer toda una avenida
- 2) Cómo contar todos los papeles de la esquina dónde se encuentra el robot.
- 3) Cómo contar los papeles de la última esquina, es decir, la (1,100).
- 4) Informar el valor de papeles recogidos en el recorrido.

La solución podría plantearse como sigue:

```
programa cap3Ejemplo7
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    papeles: numero
comenzar
    papeles:=0
    repetir 99           {1}
        mientras (HayPapelEnLaEsquina)      {2}
            tomarPapel
            {registramos que se tomo un papel}
            papeles:=papeles+1
        mover
        {procesamos la última esquina}
        mientras (HayPapelEnLaEsquina)      {3}
            tomarPapel
            {registramos que se tomo un papel}
            papeles:=papeles+1
        Informar (papeles)                  {4}
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

En (1) indicamos la estructura de control para recorrer toda una avenida. Como se sabe la cantidad de cuadras de una avenida completa es 99.

En (2) contamos todos los papeles de la esquina en donde se encuentra parado el robot.

En (3) contamos los papeles de la última esquina, es decir, la (1,100).

En (4) informamos el valor de todos los papeles recogidos en el recorrido, el cual se encuentra almacenado en papeles.



¿Qué modificaría en el algoritmo si se quisiera informar los papeles de cada esquina?

## 3.7 Representación de más de un dato dentro del algoritmo

En las secciones anteriores se presentó la necesidad de almacenar información adicional para responder a los requerimientos del problema. Es importante reconocer que los algoritmos desarrollados en los capítulos 1 y 2, sólo se referían al funcionamiento de las estructuras de control y no al manejo de los datos.

Contar con la posibilidad de asociar un identificador a un valor es equivalente a poder “registrar” un dato para recuperarlo o modificarlo posteriormente.

Como resumen de todo lo visto hasta el momento se presentará un ejemplo más complejo, repitiendo la explicación del proceso de modificación de la información y las posibles comparaciones a utilizar.

**Ejemplo 3.8:** El robot debe recoger e informar la cantidad de flores y papeles de la esquina (1,1).

En este ejercicio debemos representar dos datos, la cantidad de flores de la esquina y la cantidad de papeles de la esquina. Para esto, vamos a necesitar dos variables una que nos permita contar la cantidad de flores de la esquina y otra que nos permita contar la cantidad de papeles. El algoritmo quedaría de la siguiente forma:

```
programa cap3Ejemplo8
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    papeles: numero
    flores: numero
comenzar
    flores:=0 {1}
    papeles:=0 {2}
    mientras (HayPapelEnLaEsquina) {3}
        tomarPapel
        {registramos que se tomó un papel}
        papeles:=papeles+1
    mientras (HayFlorEnLaEsquina) {4}
        tomarFlor
        {registramos que se tomó una flor}
        flores := flores + 1
    Informar (flores) {5}
    Informar (papeles) {6}
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```



En (1) indicamos que no se ha recogido ningún papel todavía. Lo mismo ocurre con las flores en (2).

En (3) contamos todos los papeles que existen en la esquina. Lo mismo hacemos en (4) pero para las flores. En este punto podemos ver que la cantidad de papeles de la esquina va almacenándose en la variable papeles y la cantidad de flores en la variable flores.

En (5) y (6) se informa el valor de flores y papeles de la esquina respectivamente.

**Ejemplo 3.9:** Modifique el ejercicio anterior para que el robot evalúe cual de las cantidades (flores y papeles) resultó mayor e informe dicha cantidad.

Siguiendo el razonamiento anterior, luego de recoger y contar las flores y los papeles, deberemos analizar cuál de los dos valores obtenidos es el mayor e informar dicho valor.

El algoritmo quedaría de la siguiente forma:

```
programa cap3Ejemplo9
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    papeles: numero
    flores: numero
comenzar
    papeles:=0
    mientras (HayPapelEnLaEsquina)
        tomarPapel
        {registramos que se tomó un papel}
        papeles:=papeles+1
    mientras (HayFlorEnLaEsquina)
        tomarFlor
        {registramos que se tomó una flor}
        flores := flores + 1
    si (flores>papeles)                                {1}
        Informar(flores)
    sino
        Informar(papeles)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

En (1) se verifica si la cantidad de flores es mayor a la de papeles, por medio de una estructura de selección, en caso que la condición resulte verdadera se informa la cantidad de flores, en cambio, si la condición es falsa se informa la cantidad de papeles.

¿Qué informa si ambas variables contienen el mismo valor?





## 3.8 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 ambiente de programación del robot R-info nos provee dos tipos de datos: el tipo numero 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.



## Capítulo 5

# Programación Estructurada



### Objetivo

Este capítulo introduce una metodología que permitirá facilitar la resolución de problemas. La propuesta radica en descomponer las tareas a realizar en subtareas más sencillas de manera de no tener que escribir un programa como un todo.

Los problemas del mundo real, en general, resultan ser complejos y extensos. Si se pretende seguir con la forma de trabajo hasta aquí utilizada, se verá que resulta difícil cubrir todos los aspectos de la solución.

Por ejemplo, supongamos que se pide que el robot R-info recorra todas las avenidas de la ciudad juntando flores y papeles e informe la cantidad de flores por avenida y el total de papeles durante todo el recorrido, seguramente nos resultará más natural y sencillo pensar en términos de tareas. Por ejemplo, recorrer una avenida contando flores y papeles, informar cantidad de flores de la avenida, posicionarse en la avenida siguiente, poner la cantidad de flores en cero, y al finalizar el recorrido informar la cantidad de papeles.

Esta forma de trabajo es lo que se formalizará a partir de este capítulo y se presentarán sus beneficios.



### Temas a tratar

- ✓ Descomposición de problemas en partes
- ✓ Programación modular
- ✓ Conclusiones
- ✓ Ejercitación

## 5.1 Descomposición de problemas en partes

Una de las herramientas más útiles en la resolución de problemas con computadora es la descomposición de los problemas a resolver en subproblemas más simples. Esta descomposición, que se basa en el paradigma “**Divide y Vencerás**”, persigue un objetivo: cada problema es dividido en un número de subproblemas más pequeños, cada uno de los cuales a su vez, puede dividirse en un conjunto de subproblemas más pequeños aún, y así siguiendo. Cada uno de estos subproblemas debiera resultar entonces más simple de resolver. Una metodología de resolución con estas características se conoce como diseño Top -Down.

La figura 5.1 muestra la representación en forma de árbol de una descomposición Top-Down.

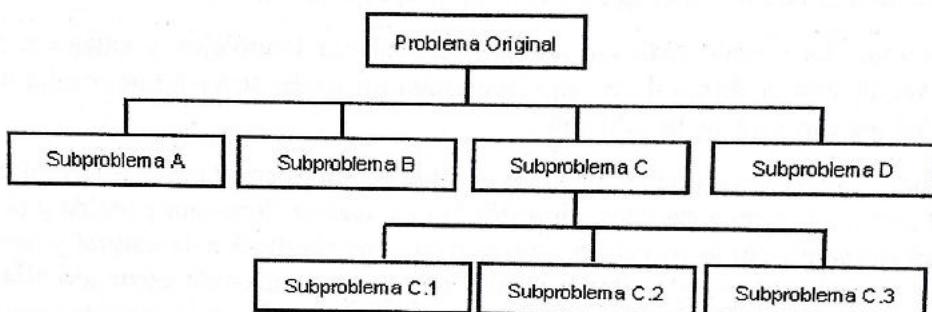


Figura 5.1: Diseño Top-Down

El nivel de descomposición al que se llega depende de los conocimientos de quien va a implementar la solución (obviamente, el nivel de detalle al que puede arribar un experto no es el mismo que al que llegará un novato).

Es decir, que con esta metodología, resolver el problema original se reduce a resolver una serie de problemas más simples o subproblemas. En cada paso del proceso de resolución, cada subproblema es refinado hasta llegar a un punto en que está compuesto de acciones tan simples que ya no tiene sentido seguir refinando.

Cuando se realiza esta descomposición debe tenerse en cuenta que los subproblemas que se encuentran a un mismo nivel de detalle pueden resolverse independientemente de los demás y que las soluciones de estos subproblemas deben combinarse para resolver el problema original

De la figura 5.1 podemos inducir que la resolución de los subproblemas C.1, C.2 y C.3 implica resolver el problema C. A su vez la resolución de los subproblemas A, B, C y D permitirán obtener la resolución del problema original. Es importante destacar que los subproblemas pueden ser resueltos de manera independiente entre sí y desarrollado por diferentes grupos de trabajo.



Haciendo clic en el siguiente link podés acceder a una animación con un ejemplo de *Modularización*: [Animación Modularización](#).

## 5.2 Programación modular

La metodología descripta en la sección anterior puede aplicarse al diseño de programas. Un buen diseño nos permitirá dividir nuestra solución en un número de piezas manejables llamadas **módulos**, cada uno de los cuales, tiene una tarea perfectamente definida.

Esta metodología, conocida como **modularización** ó **diseño Top Down**, es una de las técnicas más importantes para lograr un buen diseño de programa.

La **programación modular** es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa. La descomposición de un programa en módulos independientes más simples se conoce también como el método de “divide y vencerás”. Se diseña cada módulo con independencia de los demás y, siguiendo un método descendente, se llega hasta la descomposición final del problema en módulos en forma jerárquica.

En consecuencia, el programa se divide en módulos (partes independientes), cada uno de los cuales ejecuta una única actividad o tarea específica. Dichos módulos se analizan, codifican y ponen a punto por separado. Si la tarea asignada a cada módulo es demasiado compleja, este deberá descomponerse en otros módulos más pequeños. El proceso sucesivo de subdivisión continúa hasta que cada módulo tenga sólo una tarea específica que ejecutar.

Esta metodología de trabajo ofrece numerosas ventajas entre las que se pueden mencionar:

### • Independencia entre los módulos:

Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en distintas partes de un mismo programa. Esto reduce el tiempo de diseño y codificación del algoritmo.

Revisemos el siguiente ejemplo: El robot debe acomodar las esquinas de la ciudad de manera que en cada una quede la misma cantidad de flores que de papeles. Para hacerlo, recogerá lo que sobra. Por ejemplo, si en una esquina hay 10 flores y 14 papeles, se llevará 4 papeles para dejar 10 de cada uno. Las esquinas vacías serán consideradas equilibradas por lo que quedarán así.

Puede verse que el procesamiento de una esquina en particular es independiente del resto del programa. Podría encargarse su diseño e implementación a otra persona. Se denominará a esta persona Programador A. Como resultado de la tarea del programador A se obtendrá un módulo que iguala la cantidad de flores y de papeles haciendo que el

robot se lleve lo que sobra. El programador B será el encargado de usar este módulo en la resolución de este problema.

Es importante notar que al programador B no le interesa cómo hizo el programador A para igualar la cantidad de elementos de la esquina. Por su parte, el programador A desconoce cómo se utilizará el módulo que está implementando. Esto hace que ambos programadores trabajen con subproblemas más simples que el problema original. El programador A sólo se dedica al procesamiento de una esquina y el programador B sólo se preocupa porque el robot recorra todas las esquinas de la ciudad, dando por hecho el procesamiento de cada esquina.

### • Modificación de los módulos

Cada módulo tiene una tarea específica que debe llevar a cabo. En su interior sólo se definen acciones relacionadas con este fin. Por tal motivo, la modificación interna de un módulo no afectará al resto.

Volviendo al ejemplo anterior, suponga que debe modificarse el procesamiento de cada esquina de manera que el robot iguale la cantidad de elementos intentando depositar primero (igualará hacia el número superior) y si no tiene, recién entonces se llevará lo que sobra. Al tener el comportamiento de la esquina encerrado en un módulo, bastará con realizar allí las modificaciones para que todos los programas que lo utilicen se vean actualizados SIN necesidad de cambiar nada.

### • Reusabilidad de código

El desarrollo de un módulo es independiente del problema original que se desea resolver. Por ejemplo, podría definirse un módulo que permita al robot girar a la izquierda. Cuando esté implementado, podrá ser aplicado en múltiples recorridos. El concepto de reusabilidad hace hincapié en la ventaja de utilizar cada módulo directamente sin necesidad de volver a pensarla nuevamente. No importa cómo hace para quedar posicionado hacia la izquierda, lo que importa es que el módulo cumple con la función especificada. En otras palabras, para poder utilizar un módulo no interesa saber cómo está implementado internamente sino que alcanza con saber qué es lo que hace. Obviamente el creador o implementador del módulo deberá ocuparse de que ese módulo cumpla la función de la mejor manera posible. Es por ello que los que luego lo utilizarán pueden desentenderse del cómo lo hace y concentrarse en qué hace.

### • Mantenimiento del código

Una vez que el programa es puesto en marcha resulta común que aparezcan errores de diseño, ya sea porque no se interpretaron correctamente los requerimientos del usuario o porque han cambiado las especificaciones.

Cuando se desea realizar modificaciones, el uso de módulos es muy beneficioso ya que cada tarea se encuentra concentrada en un lugar del programa y basta con cambiar esta parte para llevar a cabo la actualización. Es más fácil encontrar dentro del programa, el lugar donde deben efectuarse las modificaciones.

Por ejemplo, inicialmente el módulo encargado de hacer girar al robot a la izquierda podría utilizar siete giros a derecha pero luego de verlo funcionar, se ve la conveniencia de realizar solo tres giros. Resulta claro ver que hay que ir al módulo que hace girar al robot y efectuar los cambios allí. Si no se utilizara un módulo con estas características, seguramente habría varios lugares dentro del programa donde habría que hacer modificaciones, aumentando así la posibilidad de error.

En general, en las soluciones modularizadas, un programa es un módulo en sí mismo denominado programa principal que controla todo lo que sucede y es el encargado de transferir el control a los submódulos de modo que ellos puedan ejecutar sus funciones y resolver el problema. Cada uno de los submódulos al finalizar su tarea, devuelve el control al módulo que lo llamó.

Un módulo puede transferir temporalmente el control a otro módulo; en cuyo caso, el módulo llamado devolverá el control al módulo que lo llamó, al finalizar su tarea.

A continuación se presenta la sintaxis a utilizar para la definición de módulos en el ambiente de programación del robot R-info.

```
proceso nombre del módulo
    comenzar
        { acciones a realizar dentro del módulo }
    fin
```

Como todo lo que se ha definido en este ambiente de programación existe en el mismo una sección especial para la declaración de los procesos llamada “procesos”. En esta sección se pueden declarar todos los procesos que el programador necesite para resolver el algoritmo.

Por lo tanto, el esquema general de un programa que utilice módulos es el siguiente:

```
programa ejemploProcesos
procesos
    proceso uno → Sección para definir los procesos
        comenzar
            .... Código del proceso
        fin
    areas
        ciudad: areaC(1,1,100,100)
    robots
        robot robot1
        comenzar
            uno
            mover
            uno
        fin
    variables
        R-info : robot1
        comenzar
            AsignarArea (R-info,ciudad)
            iniciar (robot1 , 1 , 1)
        fin
```

**Ejemplo 5.1:** Se desea programar al robot para que recoja todas las flores de la esquina (1,1) y de la esquina (1,2).

Solución sin modularizar	Solución modularizada
<pre> programa Cap5Ejemplo5.1 areas     ciudad: AreaC(1,1,100,100) robots     robot robot1 comenzar     mientras HayFlorEnLaEsquina         tomarFlor     mover     mientras HayFlorEnLaEsquina         tomarFlor     fin variables     R-info : robot1 Comenzar     AsignarArea (R-info, ciudad)     Iniciar (robot1 , 1 , 1) fin </pre>	<pre> programa Cap5Ejemplo5.1.2 procesos     proceso JuntarFlores {3} comenzar     mientras HayFlorEnLaEsquina         tomarFlor     fin areas     ciudad: AreaC(1,1,100,100) robots     robot robot1 comenzar     JuntarFlores {2}     mover {4}     JuntarFlores {5}     fin {6} variables     R-info : robot1 comenzar     AsignarArea (R-info, ciudad)     Iniciar (robot1 , 1 , 1) {1} fin </pre>

En la solución sin modularizar, nos vemos obligados a escribir dos veces el mismo código debido a que la tarea a realizar en las dos esquinas es la misma.

En la solución modularizada, en cambio, nos alcanza con invocar al módulo JuntarFlores cada vez que se quiera recoger flores en una esquina cualquiera. En este caso particular será necesario invocar a dicho módulo dos veces.

La ejecución del programa Cap5Ejemplo5.1.2 (solución modularizada) comienza por la línea (1). A continuación, la instrucción (2) realiza la llamada o invocación al proceso JuntarFlores. Esto hace que se suspenda la ejecución del código del robot R-info y el control pase a la línea (3) donde se encuentra el inicio del proceso. Cuando este termina su ejecución, el robot habrá recogido todas las flores de la esquina y el control volverá a la línea siguiente a la que hizo la invocación (4). El robot avanzará a la esquina (1,2) y en la línea (5) se llamará al mismo proceso nuevamente, lo que produce que se vuelva a ejecutar desde (3) nuevamente. Cuando este proceso haya finalizado, el control volverá a la línea (6) y el robot se detendrá llevando en su bolsa las flores de las esquinas (1,1) y (1,2).

**Ejemplo 5.2:** Se desea programar al robot para que recorra la avenida 1 juntando todas las flores y los papeles que encuentre.

Esto puede simplificarse si se utiliza la metodología Top-Down ya descripta. Este programa se puede descomponer en módulos, de modo que exista un módulo principal y diferentes submódulos como se muestra en la figura 5.2.



Figura 5.2:Diseño Top-Down del ejemplo 5.2

El código correspondiente al robot R-info sería de la siguiente forma:

Para cada esquina de la avenida 1

{*Juntar todas las flores de la esquina*}  
{*Juntar todos los papeles de la esquina*}  
{*Avanzar a la próxima esquina*}

Por lo tanto, el programa quedaría entonces de la siguiente manera:

```
programa Cap5Ejemplo5.2
procesos
    proceso JuntarFlores
    comenzar
        mientras HayFlorEnLaEsquina
            tomarFlor
        fin
    proceso JuntarPapeles
    comenzar
        mientras HayPapelEnLaEsquina
            tomarPapel
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    comenzar
        repetir 99
            JuntarFlores
            JuntarPapeles
            mover
            JuntarFlores
            JuntarPapeles
        fin
    variables
        R-info : robot1
    comenzar
        AsignarArea(R-info,ciudad)
        Iniciar (R-info , 1 , 1)
    fin
```

**Ejemplo 5.3:** Se desea programar al robot para que recorra las primeras 10 avenidas juntando todas las flores y los papeles que encuentre.



Figura 5.3: Diseño Top-Down del ejemplo 5.3

Solución 1	Solución 2
<pre> programa Cap5Ejemplo5.3.Solucion1 procesos   proceso JuntarFlores   comenzar     mientras HayFlorEnLaEsquina       tomarFlor     fin   proceso JuntarPapeles   comenzar     mientras HayPapelEnLaEsquina       tomarPapel     fin   areas   ciudad: AreaC(1,1,100,100)   robots     robot robot1     comenzar       repetir 10         repetir 99           JuntarFlores           JuntarPapeles           mover         {Falta última esquina}         JuntarFlores         JuntarPapeles         Pos(PosAv + 1 ,1)       fin   variables     R-info : robot1   comenzar     AsignarArea (R-info,ciudad)     Iniciar (R-info, 1 ,1)   fin </pre>	<pre> programa Cap5Ejemplo5.3.Solucion2 procesos   proceso JuntarFlores   comenzar     mientras HayFlorEnLaEsquina       tomarFlor     fin   proceso JuntarPapeles   comenzar     mientras HayPapelEnLaEsquina       tomarPapel     fin   proceso Avenida   comenzar     repetir 99       JuntarFlores       JuntarPapeles       mover       JuntarFlores       JuntarPapeles     fin   areas   ciudad: AreaC(1,1,100,100)   robots     robot robot1   comenzar     repetir 10       Avenida       Pos(PosAv + 1 ,1)     fin   variables     R-info : robot1   comenzar </pre>

	AsignarArea (R-info, ciudad) Iniciar (R-info , 1 ,1) fin
--	--

Como podemos observar, ambas soluciones resuelven el problema. Sin embargo en la solución 2, el programa principal resulta más legible debido a la utilización del módulo Avenida.

Por otra parte, hay que destacar que en la solución 2, desde el módulo Avenida se está invocando a los módulos JuntarFlores y JuntarPapeles. Para que el módulo Avenida pueda invocar correctamente a los módulos JuntarFlores y JuntarPapeles, estos deben estar previamente declarados.



- Analice el programa anterior realizando un seguimiento de la invocación de los procesos involucrados.

**Ejemplo 5.4:** Se desea programar al robot para que recorra las primeras 50 avenidas juntando las flores de las avenidas pares y los papeles de las avenidas impares. Al finalizar cada avenida debe depositar todos los elementos recogidos. Considere que inicialmente la bolsa está vacía.

Al igual que en los ejemplos anteriores, este programa se puede descomponer en módulos y submódulos. La idea principal de lo que debería hacer el robot es la siguiente:

#### Recorrido de Avenidas

{Realizar avenida impar}  
{Realizar avenida par}  
{Posicionamiento para la próxima avenida}

Notemos que la complejidad de los módulos no es la misma. En particular, el último módulo sólo implica reubicar al robot en otra esquina, por lo cual, su refinamiento no es necesario.

Si lo consideramos necesario, podemos continuar refinando los módulos que procesan las avenidas de la siguiente manera:

#### Módulo Realizar avenida impar

{Recorrer la avenida impar juntando papeles}  
{Depositar los papeles encontrados al finalizar la avenida impar}

#### Módulo Realizar avenida par

{Recorrer la avenida par juntando flores}  
{Depositar las flores encontradas al finalizar el recorrido}

En un nivel más de refinamiento, estos módulos pueden volver a descomponerse de la siguiente forma:



**Submódulo Recorrer la avenida impar juntando papeles**  
*{para cada esquina de la avenida impar}*  
*{recoger todos los papeles de la esquina}*  
*{avanzar una cuadra sobre la avenida impar}*

**Submódulo Recorrer la avenida par juntando flores**  
*{para cada esquina de la avenida par}*  
*{recoger todas las flores de la esquina}*  
*{avanzar una cuadra sobre la avenida par}*

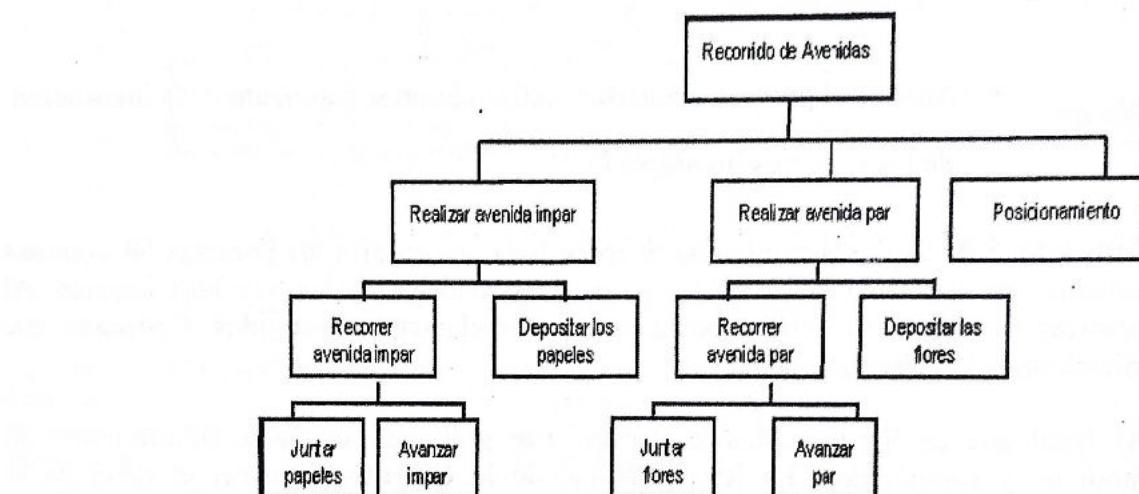


Figura 5.4: Diseño Top-Down del Ejemplo 5.4

La representación gráfica se muestra en la figura 5.4. Puede verse que a nivel de módulo se considera como elemento de trabajo a la avenida mientras que a nivel de submódulo se hace sobre las esquinas de cada avenida.

Recordemos que la descomposición Top-Down parte del problema general descomponiéndolo en tareas cada vez más específicas y el concentrarse en una tarea en particular es más simple que resolver el problema completo. En el ejemplo, es más simple resolver una única avenida par que intentar resolver las 25 avenidas pares en forma conjunta. Lo mismo ocurre con las impares.



En la resolución de este problema pueden utilizarse los procesos JuntarFlores y JuntarPapeles definidos anteriormente. Además se necesitarán algunos otros módulos: DejarPapeles, DejarFlores, RealizarAvenidaImpar y RealizarAvenidaPar.

A continuación se presenta el programa completo que resuelve el ejemplo 5.4.

```
programa Cap5Ejemplo5.4
procesos
    proceso JuntarFlores
    comenzar
        mientras HayFlorEnLaEsquina
            tomarFlor
        fin
    proceso JuntarPapeles
    comenzar
        mientras HayPapelEnLaEsquina
            tomarPapel
        fin
    proceso DejarFlores
    comenzar
        mientras HayFlorEnLaBolsa
            depositarFlor
        fin
    proceso DejarPapeles
    comenzar
        mientras HayPapelEnLaBolsa
            depositarPapel
        fin
    proceso RealizarAvenidaPar
    comenzar
        repetir 99
            JuntarPapeles
            mover
            DejarPapeles
        fin
    proceso RealizarAvenidaImpar
    comenzar
        repetir 99
            JuntarFlores
            mover
            DejarFlores
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
        comenzar
            repetir 25
                RealizarAvenidaImpar
                Pos(PosAv + 1 ,1)
                RealizarAvenidaPar
                Pos(PosAv +1 , 1)
            fin
    variables
        R-info : robot1
        comenzar
            AsignarArea (R-info,ciudad)
            Iniciar (R-info , 1 ,1)
        fin
```

Sobre el ejemplo anterior pueden analizarse los siguientes aspectos:

- Se han reusado los procesos JuntarFlores y JuntarPapeles evitando de esta manera el tener que volver a pensar e implementar la recolección de cada tipo de elemento dentro de una esquina específica.
- Si en el ejemplo 5.2 se hubiera escrito un único proceso que juntara todos los elementos de la esquina, no hubiera sido posible el reuso mencionado anteriormente. Esto lleva a tratar de escribir módulos lo suficientemente generales como para que puedan ser aplicados en distintas soluciones.
- Podemos notar que no todos los submódulos de la figura 5.4 se han convertido en procesos. Esto se debe a que algunos de ellos son lo suficientemente simples como para traducirse en una instrucción del robot. Por ejemplo, "Avanzar impar" se traduce en la instrucción mover.

**Ejemplo 5.5:** Programe al robot para que recorra el perímetro del cuadrado determinado por (1,1) y (2,2). Al terminar debe informar cuántos de los vértices estaban vacíos.

Para resolverlo es conveniente comenzar a analizar las distintas partes que componen el problema:

{Analizar el vértice y registrar si está vacío}  
{Avanzar 1 cuadra}  
{Girar a la izquierda}  
{Informar lo pedido}

Las primeras tres acciones se repiten cuatro veces para poder dar vuelta al cuadrado y la última se realiza cuando el robot ha terminado de recorrer el perímetro.

Note que la descomposición en módulos del problema no busca representar flujo de control del programa. En otras palabras, las estructuras de control no se encuentran reflejadas en la metodología Top-Down. Sólo se indican las partes necesarias para resolverlo, de manera de dividir el trabajo en tareas más sencillas. La unión de estos módulos, para hallar la solución, es una tarea posterior. Por este motivo, tampoco se repiten los módulos dentro de la descomposición planteada.

Todas las tareas indicadas en la descomposición anterior son lo suficientemente simples como para ser implementadas directamente. Una solución al problema podría ser la siguiente:

```
programa Cap5Ejemplo5.5
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
```

```

vacias : numero
comenzar
derecha
vacias := 0
repetir 4
{Analizar el vértice y registrar si está vacío}
si ~ HayFlorEnLaEsquina & ~HayPapelEnLaEsquina
vacias := vacias + 1
mover
{Girar a la izquierda}
repetir 3
derecha
Informar (vacias)
fin
variables
R-info : robot1
comenzar
AsignarArea (R-info, ciudad)
Iniciar (R-info , 1 ,1)
fin
```

Sin embargo, sería importante contar con un proceso que permitiera indicar al robot girar izquierda con la misma facilidad con que se indica que gire a derecha. Esto haría más legible los programas ya que el programa anterior podría escribirse de la siguiente forma:

```

programa Cap5Ejemplo5.5
areas
ciudad: areaC(1,1,100,100)
robots
robot robot1
variables
vacias : numero
comenzar
derecha
vacias := 0
repetir 4
{Analizar el vértice y registrar si está vacío}
si ~ HayFlorEnLaEsquina & ~HayPapelEnLaEsquina
vacias := vacias + 1
mover
{Girar a la izquierda}
izquierda
Informar (vacias)
fin
variables
R-info : robot1
comenzar
AsignarArea (R-info, ciudad)
iniciar (robot1 , 1 ,1)
fin
```



- Queda como ejercicio para el lector la definición del proceso izquierda.
- Una vez que haya incorporado a la solución anterior el proceso izquierda, indique la cantidad de veces que el robot gira a la derecha para completar el recorrido.



**Ejemplo 5.6:** Dados los recorridos de la figura 5.6 ¿Cuál es el módulo que sería conveniente tener definido?

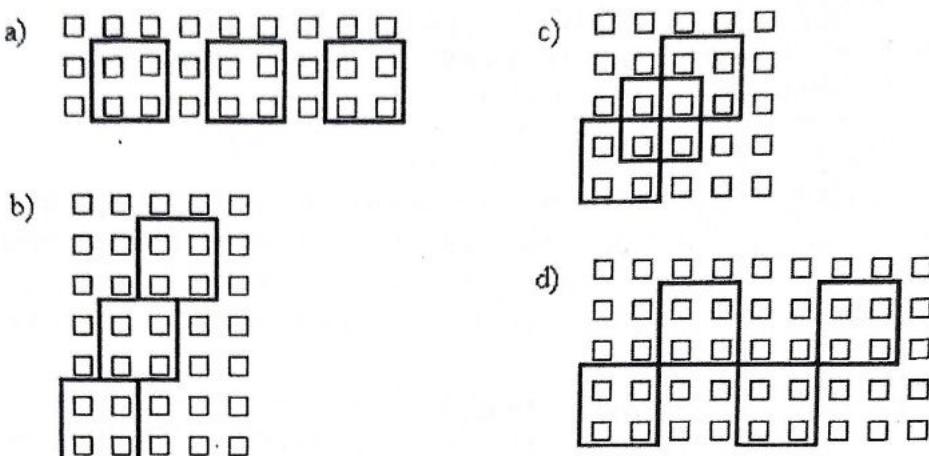


Figura 5.6: Recorridos

Como podemos observar todos los cuadrados tienen lado 2, entonces convendría tener definido un módulo que permita realizar un cuadrado de lado 2 ya que puede utilizarse en los cuatro recorridos. Como muestra la figura 5.6 cada recorrido consiste en realizar varios cuadrados de lado 2 con el robot posicionado en lugares diferentes.

El módulo a utilizar puede escribirse de la siguiente forma:

```
proceso cuadrado
comenzar
    repetir 4      {el cuadrado tiene 4 lados}
        repetir 2
            mover
    fin
```

En este momento puede apreciarse una de las principales ventajas de la modularización. Esta implementación del proceso cuadrado es independiente del recorrido en el cual será utilizado. Es más, este proceso puede ser verificado de manera totalmente independiente de la aplicación final.

El programador encargado de su desarrollo podría escribir el siguiente programa para verificar su funcionamiento:

```
programa Cap5Ejemplo5.6
procesos
    proceso cuadrado
    comenzar
        repetir 4
            repetir 2
                mover
            fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
        comenzar
```

```

        Pos (2,1)
        cuadrado
    fin
variables
R-info : robot1
comenzar
AsignarArea (R-info,ciudad)
Iniciar (R-info , 1 ,1)
fin

```

Una vez que se ha verificado que el funcionamiento del proceso cuadrado es correcto, su programador lo ubicará en un lugar común donde todos aquellos que lo requieran lo puedan acceder. Ahora, los usuarios del proceso cuadrado no sólo no tendrán la necesidad de implementar este recorrido sino que además cuentan con un proceso que funciona correctamente porque ya fue verificado previamente.

En este sentido, el ambiente de programación del robot R-info presenta algunas limitaciones ya que los interesados en el módulo deberán insertarlo dentro de su programa. Sin embargo, esto no es así en la mayoría de los lenguajes y este aspecto no debería invalidar las ventajas de la modularización.

Los siguientes programas resuelven el recorrido a) de la figura 5.6:

<pre> programa Cap5Ejemplo5.6.version1 procesos     proceso cuadrado     comenzar         repetir 4             repetir 2                 mover                 derecha         fin     areas         ciudad: AreaC(1,1,100,100)     robots         robot robot1         comenzar             Pos(2,1)             cuadrado             Pos(5,1)             cuadrado             Pos(8,1)             cuadrado         fin     variables         R-info : robot1     comenzar         AsignarArea (R-info,ciudad)         Iniciar (R-info , 1 ,1)     fin </pre>	<pre> programa Cap5Ejemplo5.6.version2 procesos     proceso cuadrado     comenzar         repetir 4             repetir 2                 mover                 derecha         fin     areas         ciudad: AreaC(1,1,100,100)     robots         robot robot1         comenzar             Pos(2,1)             repetir 3                 cuadrado                 Pos(PosAv+3,1)         fin     variables         R-info : robot1     comenzar         AsignarArea (R-info,ciudad)         Iniciar (R-info , 1 ,1)     fin </pre>
---	--

Una de las diferencias entre estas soluciones es que el programa RecorridoA-version1 utiliza solo la secuencia mientras que RecorridoA version2 utiliza la estructura de control repetir.



En ambos casos aparecen tres invocaciones al proceso cuadrado con el robot ubicado en una esquina distinta.

El módulo Cuadrado tiene las siguientes características:

1. El cuadrado tiene como vértice inferior izquierdo la esquina donde el robot está posicionado al momento de la invocación.
2. Una vez terminado el cuadrado, el robot vuelve a quedar parado en la misma esquina y orientado en el mismo sentido que cuando se comenzó la ejecución del módulo.



Justifique las siguientes afirmaciones:

- Para que el programa RecorridoA-version1 funcione correctamente sólo es preciso conocer la primera de estas características.
- En cambio, para que el programa RecorridoA-version2 funcione correctamente se requieren las dos.

A continuación se presentan una parte de la solución para el recorrido de la figura 5.6 b) (sólo se muestra el código del robot), en la solución de la izquierda se utiliza el modulo cuadrado y en la de la derecha no.

```
robots
robot robot1
comenzar
repetir 3
cuadrado
Pos(PosAv+1, PosCa+2)
fin
```

```
robots
robot robot1
comenzar
repetir 3
repetir 4
mover
mover
derecha
Pos(PosAv+1, PosCa+2)
fin
```

El programa RecorridoB version1 utiliza el proceso que realiza el cuadrado de lado 2, mientras que el programa RecorridoB version2 ha sido implementado igual que los ejercicios de los capítulos anteriores, es decir, sin utilizar ningún proceso.

Estos dos programas muestran algunos aspectos importantes:

1. El uso de la modularización no es obligatorio. En los capítulos anteriores se han resuelto problemas similares al del recorrido b) sin utilizar procesos. La metodología Top-Down no pretende afirmar que aquellas soluciones fueron incorrectas. Sólo muestra una forma alternativa que debería facilitar el diseño y desarrollo de los programas.
2. Si se presta atención al programa RecorridoB version1, puede verse que su implementación sólo se preocupa por posicionar al robot para formar el recorrido pedido. Mientras tanto, el programa RecorridoB version2 debe resolver ambos problemas: posicionar al robot y hacer el cuadrado.



3. Al proceso RecorridoB version1 no le preocupa cómo se hace el cuadrado. Da lo mismo que lo haga girando en el sentido de las agujas del reloj o girando en sentido contrario. En cambio el otro programa debe indicar claramente cómo hacer todo el recorrido.
4. Si en el futuro el robot contara con nuevas habilidades que le permitieran hacer el cuadrado de forma más eficiente (por ejemplo, algún día podría aprender a correr) los programas que utilizan el módulo cuadrado sólo tendrán que hacer referencia a la nueva implementación. Mientras tanto, los programas que hayan implementado explícitamente, en su interior, el recorrido para hacer el cuadrado de lado 2 deberán ser modificados uno por uno.

Quedan a cargo del lector las implementaciones de los programas que permitan al robot realizar los recorridos c y d.

**Ejemplo 5.7:** Programe al robot para realizar el recorrido de la figura 5.7.

Como puede apreciarse en el dibujo, el problema consiste en programar al robot para que de la vuelta a cada una de las manzanas indicadas. Como se explicó anteriormente, el uso de la metodología Top-Down no es obligatorio pero, si se opta por no utilizarla, habrá que enfrentar la solución del problema como un todo y el recorrido a realizar ya no es tan simple.

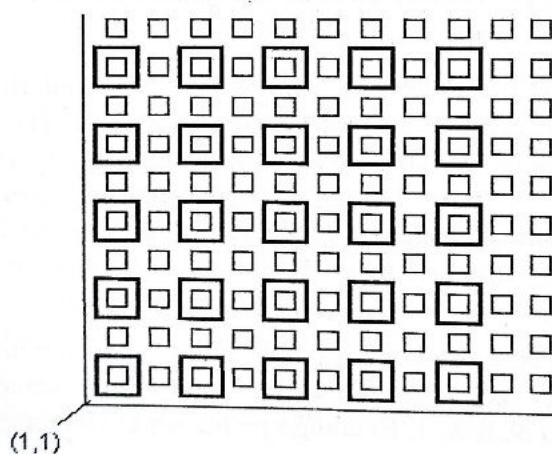


Figura 5.7: Recorrido del Ejemplo 5.7

Si se descompone el problema puede pensarse en cinco torres de cinco cuadrados cada una. Si se logra resolver una de las torres, luego solo habrá que repetir el proceso cinco veces. La figura 5.7 muestra la descomposición Top-Down del problema.

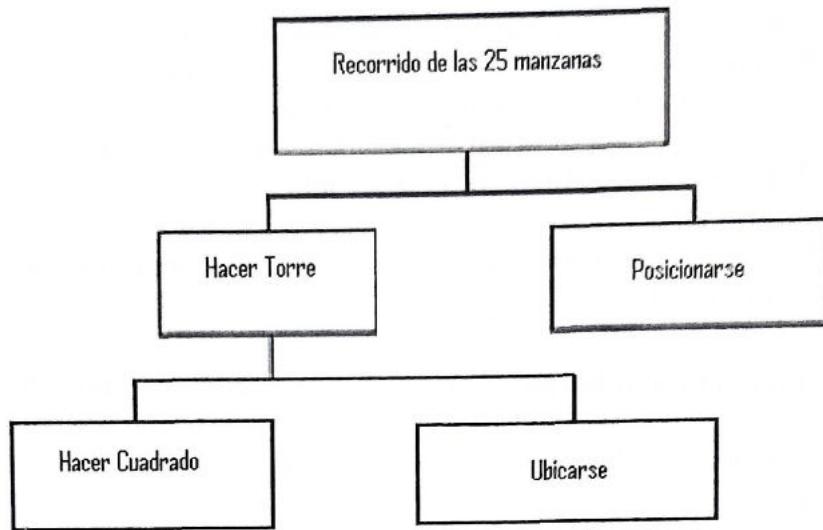


Figura 5.7: Descomposición Top-Down del Ejemplo 5.5

Si HacerTorre es el proceso que realiza una secuencia de cinco cuadrados como los de la figura 5.7, el código del robot podría escribirse de la siguiente forma:

```
robots
  robot robot1
comenzar
  repetir 5
    HacerTorre
    Pos(PosAv+2, 1)
  Fin
```

Para implementar cada torre de cinco cuadrados puede aplicar la misma metodología de descomposición del problema. Cada torre no es más que llamar cinco veces a un proceso que realice un cuadrado de lado 1. El código podría ser el siguiente:

```
proceso Torre
comenzar
  repetir 5
    HacerCuadrado
    Pos(PosAv, PosCa+2)
  fin
```

Como puede verse en el código anterior, el proceso HacerTorre realiza los cinco cuadrados a partir de la posición donde el robot se encuentra parado. Cada cuadrado de lado 1 tendrá su esquina inferior izquierda apoyada sobre la misma avenida. Notemos que para usar el proceso HacerCuadrado sólo importa saber que se realiza tomando como esquina inferior izquierda del cuadrado de lado 1, la esquina donde el robot está parado al momento de la invocación. A continuación se muestra el programa implementado en el ambiente de programación del robot R-info:



```
programa Cap5Ejemplo5.7
procesos
    proceso HacerCuadrado
    comenzar
        repetir 4
            mover
            derecha
    fin
    proceso HacerTorre
    comenzar
        repetir 5
            HacerCuadrado
            Pos(PosAv, PosCa+2)
    fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    repetir 5
        HacerTorre
        Pos(PosAv + 2, 1 )
    fin
variables
    R-info : robot1
comenzar
    AsignarArea (R-info,ciudad)
    Iniciar (R-info , 1 ,1)
fin
```

Es importante destacar que la ejecución del algoritmo comienza en el punto indicado con el número (1). Cuando el control llega a la invocación del proceso HacerTorre, indicada por (2), el algoritmo continúa ejecutándose en dicho proceso. Cuando llega al punto (3), es decir la invocación de HacerCuadrado, se suspende la ejecución del proceso HacerTorre y se realiza la primera vuelta a la manzana a partir de (1,1). Luego retorna a la instrucción siguiente de (3), posicionándose en la esquina (1,3), listo para hacer el próximo cuadrado. Esto se repite cinco veces. Una vez terminada la torre, el robot queda parado en (1,11). En ese momento el proceso HacerTorre termina y retorna a la instrucción siguiente de (2), posicionándose en la próxima avenida impar, listo para comenzar la segunda torre. Esto se repite cuatro veces más y el recorrido finaliza.



¿Dónde queda parado el robot al final del recorrido?

**Ejemplo 5.8:** Escriba el proceso Evaluar para que el robot recoja todos los papeles y las flores de la esquina donde se encuentra y deje una flor en caso de haber más flores que papeles; un papel, si hay más papeles que flores ó uno de cada uno en caso contrario.

Los pasos a seguir para implementar este proceso son los siguientes:

```
proceso Evaluar
    {contar la cantidad de flores}
    {contar la cantidad de papeles}
    si {hay más flores que papeles}
        {depositar una flor (si es que hay en la bolsa)}
    sino
```

```

    si {la cantidad de flores y papeles es la misma }
        {depositar uno de cada uno (si es que hay en la bolsa)}
    sino
        {depositar un papel (si es que hay en la bolsa)}

```

La implementación del proceso Evaluar será:

```

proceso Evaluar
variables
    CantF: numero
    CantP: numero
comenzar
    {cuenta las flores}
    CantF := 0
    mientras HayFlorEnLaEsquina
        tomarFlor
        CantF := CantF + 1
    {cuenta los papeles}
    CantP := 0
    mientras HayPapelEnLaEsquina
        tomarPapel
        CantP := CantP + 1
    {decide que depositar}
    si CantF > CantP
        si HayFlorEnLaBolsa
            depositarFlor
        sino
            si cantF = cantP
                si HayFlorEnLaBolsa & HayPapelEnLaBolsa
                    depositarPapel
                    depositarFlor
                sino
                    si HayPapelEnLaBolsa
                        depositarPapel
    fin

```

Como se puede observar en el ejemplo planteado, el módulo Evaluar utiliza dos variables propias para representar la cantidad de flores y papeles que hay en una esquina de la ciudad.

Los valores de estas variables permiten tomar posteriormente una decisión. Es importante destacar que estas variables sólo son útiles para el módulo y no son conocidas fuera del mismo.



- Analice el comportamiento del proceso Evaluar cuando se trata de una esquina vacía ¿Cuáles son las proposiciones atómicas que se evalúan? ¿Qué pasa si en la bolsa sólo tiene papeles?

**Ejemplo 5.9:** Programe al robot para que aplique el proceso Evaluar a cada una de las esquinas de la ciudad.

Tengamos en cuenta que para recorrer cada una de las esquinas de la ciudad, basta con recorrer todas las calles ó todas las avenidas. Cualquiera de estos recorridos nos asegura que el robot pasa por todas las esquinas de la ciudad. El diseño Top-Down para la solución de este problema puede ser:



Figura 5.8: Descomposición Top-Down del Ejemplo 5.9

```
programa Cap5Ejemplo5.9
procesos
    proceso Evaluar
    variables
        CantF: numero
        CantP: numero
    comenzar
        {cuenta las flores}
        CantF := 0
        mientras HayFlorEnLaEsquina
            tomarFlor
            CantF := CantF + 1
        {cuenta los papeles}
        CantP := 0
        mientras HayPapelEnLaEsquina
            tomarPapel
            CantP := CantP + 1
        {decide que depositar}
        si CantF > CantP
            si HayFlorEnLaBolsa
                depositarFlor
            sino
                si CantF = CantP
                    si HayFlorEnLaBolsa & HayPapelEnLaBolsa
                        depositarFlor
                        depositarPapel
                sino
                    si HayPapelEnLaBolsa
                        depositarPapel
        fin
    proceso RecorrerCalle
    comenzar
        repetir 99
            Evaluar
            mover
            Evaluar
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    comenzar
        derecha
        repetir 99
            RecorrerCalle
            Pos(1,PosCa +1)
```

```
    RecorrerCalle
    fin
variables
    R-info : robot1
comenzar
    AsignarArea (R-info, ciudad)
    Iniciar (R-info , 1 ,1)
fin
```

Esta solución recorre cada calle usando el módulo RecorrerCalle. Para cada calle se recorren todas sus esquinas ejecutando el módulo Evaluar. Notemos que en el módulo RecorrerCalle no se hace ninguna referencia a las variables usadas en el módulo Evaluar, como así tampoco en el cuerpo del programa principal.

### 5.3 Conclusiones

En este capítulo se ha presentado una metodología que ayuda a diseñar soluciones a problemas más complejos basada en la descomposición del problema original en subproblemas más sencillos. En particular se han exemplificado las siguientes ventajas:

- La descomposición realizada facilita la implementación del programa ya que las partes a desarrollar son mucho más simples que el todo.
- El programa principal, encargado de invocar los módulos, es más fácil de leer e interpretar.
- La existencia de procesos permite reusar código escrito anteriormente. Esto tiene la ventaja no sólo de no tener que volver a analizarlo y reescribirlo sino que se asegura que su funcionamiento será el adecuado.



### Ejercitación



## Capítulo 6

# Parámetros de entrada



## Objetivos

El objetivo de este capítulo es extender la sintaxis de definición de procesos a fin de permitir que se comparta información entre el módulo que llama y el módulo que es llamado.

Esto brindará la posibilidad de flexibilizar el comportamiento del proceso obteniendo, de esta forma, mejores resultados.



## Temas a tratar

- ✓ Comunicación entre módulos
- ✓ Declaración de parámetros
- ✓ Un ejemplo sencillo
- ✓ Ejemplos
- ✓ Restricción en el uso de los parámetros de entrada
- ✓ Conclusiones
- ✓ Ejercitación

## 6.1 Comunicación entre módulos

La metodología Top-Down se basa en la descomposición del problema original en partes más simples. Esto facilita su resolución dando origen a diversos módulos, cada uno de ellos con una función bien definida. Por otro lado, si es posible contar con un conjunto de subproblemas ya resueltos correctamente, estos podrán ser combinados para expresar soluciones más complejas.

Por ejemplo, podría ser útil contar con un proceso que permitiera conocer la cantidad de flores que el robot lleva en su bolsa, o tal vez podría desarrollarse un módulo que le permitiera al robot realizar un rectángulo cuyo alto y ancho se indicara durante la ejecución del programa.

Situaciones como las anteriores requieren que los procesos comparten información con el módulo que los invoca.

Los módulos desarrollados en el capítulo 5 no cuentan con esta posibilidad y su comportamiento es muy limitado ya que hacen siempre lo mismo. Por ejemplo, el proceso JuntarPapeles definido en 5.2 o el proceso Cuadrado definido en 5.6. Cada uno de estos procesos, para funcionar, sólo requieren que el robot esté posicionado en la esquina donde deben comenzar a ejecutarse. Al terminar, volverán a dejar al robot posicionado en ese mismo lugar.

Este tipo de comportamiento resulta muy acotado. Por ejemplo, ¿Qué pasaría si ahora hubiera que pedirle al proceso JuntarPapeles que retorne la cantidad de papeles que recogió? o ¿Qué pasaría si se quisiera realizar un cuadrado de lado 2 y otro de lado 5 utilizando el mismo proceso?

Cuando se quiere que el proceso interactúe con el módulo que lo llama es preciso compartir información.

La información es compartida entre módulos a través de los parámetros.

Se puede decir entonces que, se denomina parámetro a la información que se intercambia entre módulos.

En general, existen tres tipos de parámetros que interesan considerar:

- **Parámetro de entrada:** a través de este tipo de parámetro un proceso puede recibir información del módulo que lo llama. Por ejemplo, podría modificarse el proceso Cuadrado definido en el ejemplo 5.6 para que reciba información acerca del tamaño del cuadrado a realizar. De esta forma, el mismo proceso podría ser utilizado para realizar cuadrados de diferentes tamaños.
- **Parámetro de salida:** este tipo de parámetro permite que el proceso llamado genere información y pueda enviarla al módulo que lo llamó. Note que en el caso anterior, la información era generada por el módulo que llamaba, en cambio ahora, la información la genera el módulo llamado. Por ejemplo, podría ser muy útil contar con un proceso que permita conocer la cantidad de papeles que hay en una esquina. Este proceso tendría que contar los papeles y a través de un

parámetro de salida, permitir que el módulo que lo llamó pueda tener acceso a este valor.

- **Parámetro de entrada/salida:** este tipo de parámetro permite una comunicación más estrecha entre los módulos ya que la información viaja en ambos sentidos. A través de él, un dato, enviado por el módulo que llama, puede ser utilizado y modificado por el módulo llamado. Por ejemplo, el módulo JuntarPapeles podría recibir la cantidad de papeles juntados hasta el momento y actualizarla con el total de papeles de la esquina donde se encuentra parado.

En este curso vamos a trabajar con dos tipos de parámetros: los parámetros de entrada y los de entrada/salida.

## 6.2 Declaración de parámetros

La sintaxis a utilizar para definir un proceso con parámetros es la siguiente:

```
proceso nombre ( lista de parámetros )
variables
    {declare aquí las variables de este módulo}
comenzar
    {acciones a realizar dentro del módulo}
fin
```

La lista de parámetros, indicada a continuación del nombre del proceso, define cada uno de los parámetros a utilizar; es decir, la información a compartir entre el módulo llamado y el módulo que llama. Esta lista es opcional. Por ejemplo, los procesos definidos en el capítulo 5 no la utilizaban. En caso de utilizarse, la declaración de cada uno de los parámetros que la componen tiene tres partes:

1. En primer lugar debe indicarse la clase de parámetro a utilizar. Los dos tipos de parámetros que utilizaremos en la sintaxis del robot son los descriptos en la sección 6.1. Se utilizará la letra E para indicar un parámetro de entrada, y las letras ES para indicar un parámetro de entrada/salida.
2. Luego de definir la clase de parámetro a utilizar debe especificarse su nombre. Este identificador será utilizado por el proceso para recibir y/o enviar información.
3. Finalmente, a continuación del nombre del parámetro y precedido por ":" debe indicarse el tipo de dato al cual pertenece el parámetro. En la sintaxis del robot, las opciones aquí son *numero* o *boolean*.

Los parámetros se separan dentro de esta lista por ";". Por ejemplo, a continuación puede verse un módulo con su lista de parámetros:

```
proceso ProcesoDePrueba(E dato:numero; ES TodoBien:boolean)
variables
    {declare aquí las variables de este módulo}
comenzar
    {acciones a realizar dentro del módulo}
fin
```



dónde *dato* es un parámetro de entrada numérico, y *TodoBien* es un parámetro de entrada/salida lógico ó booleano. Estos parámetros indicados en el encabezado del proceso son denominados parámetros formales.

Se denominan parámetros formales a los indicados en el encabezado del proceso y se denominan parámetros actuales a los utilizados en la invocación del proceso.

Notemos la importancia de la letra que precede al nombre del parámetro. Por su intermedio puede conocerse la manera en que se compartirá la información, es decir, si es de entrada o de entrada/salida.

## 6.3 Un ejemplo sencillo

Analicemos la figura 6.1. En ella aparecen varios cuadrados de diferente tamaño. Básicamente, hay tres formas de resolver este tipo de problemas:

1. Sin utilizar modularización. Esta es la forma en que han sido resueltos los problemas en los capítulos 2 y 3. Sin embargo, hemos analizado en el capítulo 5 las ventajas de aplicar la modularización en el diseño de soluciones, por lo que resulta recomendable su utilización.
2. Utilizando procesos sin parámetros. Si se utilizara una idea similar al proceso Cuadrado definido en 5.6, debería haber tantos procesos diferentes como cuadrados de distinto tamaño de lado aparezcan en el recorrido.
3. Utilizando un único proceso cuadrado al cual pueda decírselle la longitud del lado a realizar en cada caso.

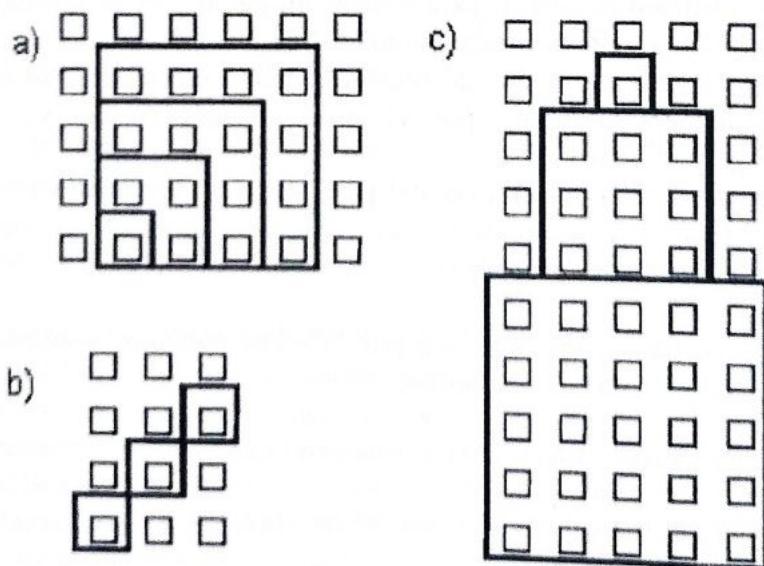


Figura 6.1: ¿Cómo se hace para que un mismo proceso sirva para realizar todos los cuadrados?

### Ejemplo 6.1: Programe al robot para que realice el recorrido a) de la figura 6.1.

Estos son los programas correspondientes al recorrido a) según las opciones 1 y 2 indicadas anteriormente:

Sin Modularizar	Con varios procesos cuadrado
<pre> programa muchosCuadradosV1 areas     ciudad: AreaC(1,1,100,100) robots     robot robot1 comenzar     {Cuadrado de lado 1}     repetir 4         mover         derecha     {Cuadrado de lado 2}     repetir 4         repetir 2             mover             derecha     {Cuadrado de lado 3}     repetir 4         repetir 3             mover             derecha     {Cuadrado de lado 4}     repetir 4         repetir 4             mover             derecha     fin variables     R-info : robot1 comenzar     AsignarArea(R-info,ciudad)     Iniciar(R-info, 1 , 1) fin </pre>	<pre> programa muchosCuadradosV2 procesos     proceso cuadrado1 comenzar     {Cuadrado de lado 1}     repetir 4         mover         derecha     fin     proceso cuadrado2 comenzar     {Cuadrado de lado 2}     repetir 4         repetir 2             mover             derecha     fin     proceso cuadrado3 comenzar     {Cuadrado de lado 3}     repetir 4         repetir 3             mover             derecha     fin     proceso cuadrado4 comenzar     {Cuadrado de lado 4}     repetir 4         repetir 4             mover             derecha     fin areas     ciudad: AreaC(1,1,100,100) robots     robot robot1 comenzar     cuadrado1     cuadrado2     cuadrado3     cuadrado4     fin variables     R-info : robot1 comenzar     AsignarArea(R-info,ciudad)     Iniciar(R-info, 1 , 1) fin </pre>

Ambas soluciones presentan los siguientes inconvenientes:

Son difíciles de generalizar: en el programa MuchosCuadradosV1 se ha realizado todo el recorrido sin descomponer el problema. Es decir, se ha analizado e implementado cada uno de los cuadrados secuencialmente. Además podemos observar que es bastante costosa su lectura.

En el programa MuchosCuadradosV2 se han utilizado cuatro procesos (que pudieron haber sido desarrollados previamente) pero si para cada tamaño de cuadrado a realizar es necesario escribir un proceso específico, se debe conocer de antemano el tamaño del cuadrado para poder escribir el proceso correspondiente. Además al aumentar la cantidad de cuadrados de diferente tamaño de lado, tendríamos que aumentar también la cantidad de procesos a escribir. No se está aprovechando la idea de que todos los cuadrados requieren del mismo recorrido, sólo sería preciso cambiar la longitud del lado. Es decir, todos se basan en repetir 4 veces: hacer el lado y girar a la derecha. Lo único que cambia entre un cuadrado y otro es la cantidad de cuadras que se deben recorrer en línea recta para hacer el lado correspondiente. En realidad, lo adecuado sería poder indicarle al proceso Cuadrado la longitud del lado que debe realizar. Para esto, el proceso cuadrado debería recibir un valor que represente la longitud del lado a través de un parámetro de entrada.

La solución toma la siguiente forma:

```
proceso cuadrado (E lado:numero) (1)
  comenzar
    repetir 4
      repetir lado
        mover
        derecha
    fin
```

En este caso, la lista de parámetros (lo que aparece entre paréntesis en la línea (1)) está formada por un único parámetro llamado *lado*. Según esta declaración, se trata de un parámetro de entrada pues su nombre se encuentra precedido por la letra E y corresponde al tipo *numero*. El identificador *lado* será utilizado por el proceso para recibir la información. Esto quiere decir que el proceso cuadrado podrá utilizar el parámetro *lado* en el cuerpo del proceso. Pero por tratarse de un parámetro de entrada, el módulo que llamó al proceso cuadrado no podrá recibir, a través de dicho parámetro, ninguna respuesta.

Cuando el proceso cuadrado sea invocado y reciba en *lado* la longitud del lado que debe realizar, este valor será utilizado por la línea (2) para efectuar el recorrido pedido. La resolución del recorrido a) de la figura 6.1 utilizando este proceso es la siguiente:



```
programa MuchosCuadradosV3
procesos
    proceso cuadrado (E lado : numero) {3}
        comenzar
            repetir 4 {4}
                repetir lado
                    mover
                    derecha
            fin
        areas
            ciudad: AreaC(1,1,100,100)
        robots
            robot robot1
            comenzar
                cuadrado(1) {1}
                cuadrado(2) {2}
                cuadrado(3)
                cuadrado(4)
            fin
        variables
            R-info : robot1
        comenzar
            AsignarArea(R-info,ciudad)
            Iniciar(R-info, 1 , 1)
        fin
```

La ejecución del programa *MuchosCuadradosV3* comienza por la línea (1). La primera invocación al proceso se realiza en (2). Notemos que ahora, a continuación del nombre del proceso se indica, en la invocación, el valor que recibirá el proceso cuadrado en el parámetro *lado*. Cuando el proceso cuadrado recibe el control, en la línea (3), *lado* tendrá el valor 1 y por lo tanto al ejecutarse realizará un cuadrado de lado 1. Es decir, en la línea (4) la instrucción repetir lado se reemplaza por repetir 1 ya que el parámetro lado ha tomado el valor 1.

Una vez que el proceso termina, la ejecución continúa en la línea (5) donde se vuelve a invocar al proceso cuadrado pero ahora se le pasa el valor 2 como parámetro. Esto es recibido por el proceso, en la línea (3), por lo que realizará un cuadrado de lado 2. Esto se repite dos veces mas invocando al proceso cuadrado con los valores 3 y 4 como parámetro.

En la solución anterior puede verse que:

- El código es más claro que en las dos versiones anteriores, facilitando de esta forma su lectura y comprensión. En el programa principal se nota claramente que se realizan cuatro cuadrados donde el primero tiene lado 1, el segundo lado 2, el tercero lado 3 y el cuarto lado 4.
  - El proceso cuadrado, a través de su parámetro de entrada, puede ser utilizado para realizar un cuadrado de cualquier tamaño. Esto resuelve el problema de generalización presentado en *MuchosCuadradosV2*.
-  Explique por qué si se cambia el nombre del parámetro formal, *lado*, el llamado en el código del robot R-info no cambia?

## 6.4 Ejemplos

**Ejemplo 6.2:** Programe al robot para que realice el recorrido c) de la figura 6.1.

Retomando el análisis realizado en el ejemplo anterior, se comenzará resolviendo este problema sin utilizar modularización. La modularización tiene que ver con el estilo de programación. Un programa que no utilice módulos tendrá algunas desventajas con respecto al que si los utiliza.

```

programa Cap6Ejemplo2TorreSinModulos
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    repetir 4
        repetir 5
            mover
            derecha
            Pos(2,6)
        repetir 4
            repetir 3
                mover
                derecha
                Pos(3,9)
            repetir 4
                repetir 1
                    mover
                    derecha
fin
variables
    R-info : robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info, 1 , 1)
fin

```

Una de las principales desventajas de esta solución es la falta de claridad en el programa ya que para comprender lo que hace es necesario analizar cada una de las líneas que lo componen. Sería más fácil de comprender si se escribiera como código del cuerpo del robot R-info las siguientes instrucciones:

```

comenzar
    iniciar
    {realizar un cuadrado de lado 5}
    Pos (2,6)
    {realizar un cuadrado de lado 3}
    Pos (3,9)
    {realizar un cuadrado de lado 1}
fin

```

Si consideramos esta solución, vemos que sería conveniente utilizar el módulo cuadrado con el parámetro de entrada *lado* visto anteriormente. Por lo tanto la solución puede reescribirse como sigue:

```
programa Cap6Ejemplo2v1
procesos
    proceso cuadrado (E lado : numero)
        comenzar
            repetir 4
                repetir lado
                    mover
                    derecha
            fin
        areas
            ciudad: AreaC(1,1,100,100)
        robots
            robot robot1
            comenzar
                cuadrado(5)
                Pos(2,6)
                cuadrado(3)
                Pos(3,9)
                cuadrado(1)
            fin
        variables
            R-info : robot1
        comenzar
            AsignarArea(R-info,ciudad)
            Iniciar(R-info, 1 , 1)
        fin
```

Como podemos observar, la primera vez que se invoca al módulo cuadrado desde el programa principal, el parámetro *lado* recibe el valor 5, luego será invocado con valor 3 y finalmente con el valor 1. A continuación se presenta otra opción de solución que muestra un programa en el cual el robot utiliza una variable llamada *long* (para representar el lado), el módulo cuadrado y el parámetro *lado*:

```
programa Cap6Ejemplo2v2
procesos
    proceso cuadrado (E lado : numero)
        comenzar
            repetir 4
                repetir lado
                    mover
                    derecha
            fin
        areas
            ciudad: AreaC(1,1,100,100)
        robots
            robot robot1
        variables
            long : numero
        comenzar
            long := 5
            cuadrado(long) {1}
            Pos(2,6)
            long := long - 2
            cuadrado(long) {2}
            Pos(3,9)
```

```

        long := long - 2                                {3}
        cuadrado(long)
    fin
variables
    R-info : robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info, 1 , 1)
fin

```

A partir de la solución presentada se pueden analizar algunos aspectos:

- Las invocaciones (1), (2) y (3) son idénticas, sólo debemos notar que cada una envía al módulo un valor de *long* actualizado. Inicialmente le enviará el valor 5, luego el valor 3 y finalmente el valor 1, como resultado de la actualización de la variable *long*. Siguiendo este razonamiento entonces ¿podríamos utilizar una estructura repetir 3 para que la legibilidad del cuerpo del programa mejore? La respuesta es SÍ pero debemos analizar algunos aspectos adicionales para poder lograrlo. En párrafos posteriores trataremos esto.
- Existe una relación entre la esquina donde comienza el recorrido del segundo cuadrado y el tamaño del lado del primer cuadrado. Lo mismo ocurre entre el tercer cuadrado y el segundo cuadrado.

La siguiente solución contempla los aspectos analizados anteriormente:

```

programa Cap6Ejemplo2v3
procesos
    proceso cuadrado (E lado : numero)                                {3}
    comenzar
        repetir 4
            repetir lado
                mover
                derecha
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    variables
        long : numero
    comenzar
        long := 5                                              {1}
        repetir 3
            cuadrado(long)                                     {2}
            Pos (PosAv + 1,PosCa + long )                     {4}
            long := long - 2                                  {5}
        fin
    variables
        R-info : tipol
    comenzar
        AsignarArea(R-info,ciudad)
        Iniciar(R-info, 1 , 1)
    fin

```

Sigamos la solución: en (1) se asigna el valor inicial 5 a *long* ya que el primer cuadrado de la figura es de este tamaño. Este valor es utilizado en (2) para llamar por primera vez al proceso cuadrado. El parámetro formal lado recibe en (3) el valor 5 y lo utiliza dentro del proceso para realizar el cuadrado correspondiente.

Cuando el proceso termina, retorna al robot el cual se encarga de hacer las modificaciones necesarias para realizar el próximo cuadrado, esto es: posicionar al robot (4) y decrementar la longitud del lado del cuadrado (5). En la línea (4) estamos repositionando al robot, esto significará ubicarlo en la esquina formada por: la avenida en la que se encontraba posicionado aumentada en 1 y en la calle en la que se encontraba posicionado aumentada en el tamaño del lado del cuadrado anterior.

La próxima invocación al proceso se hará con *long* valiendo 3, de donde en (3) *lado* recibirá este valor y hará el cuadrado de lado 3.

Finalmente, esto se repite una vez más realizando el cuadrado de lado 1.

Notemos que al terminar el programa, *long* vale -1 y el robot, a diferencia de las soluciones anteriores, está posicionado en (4,10).

En consecuencia, podemos afirmar que esta última solución resuelve el problema de generalidad de las dos soluciones anteriores. Por ejemplo, si quisieramos ahora realizar una torre de 9 cuadrados, las modificaciones resultarían mínimas. Esto se muestra en el siguiente código.

```

programa Cap6Ejemplo2v4
procesos
    proceso cuadrado (E lado : numero) {3}
        comenzar
            repetir 4
                repetir lado
                    mover
                    derecha
                fin
            areas
                ciudad: AreaC(1,1,100,100)
            robots
                robot robot1
            variables
                long : numero
            comenzar
                long := 18
                repetir 9
                    cuadrado(long)
                    Pos(PosAv + 1, PosCa + long )
                    long := long - 2
                fin
            variables
                R-info : robot1
            comenzar
                AsignarArea(R-info,ciudad)
                Iniciar(R-info, 1 , 1)
            fin

```

**Ejemplo 6.3:** Programar al robot para que realice el recorrido a) de la figura 6.1 utilizando una variable para indicar la longitud del lado de cada cuadrado.

Puede observarse que el proceso principal consiste en realizar un cuadrado, el cual va cambiando su tamaño, comenzando por un cuadrado de lado 1 hasta uno de lado 4.

El primer análisis del algoritmo es:

{recorrido de cuadrados con igual origen}  
 {realizar los cuatro cuadrados, donde para cada uno es necesario ...}  
 {hacer un cuadrado del lado correspondiente}  
 {incrementar el lado del cuadrado}

El cuadrado, como puede observarse, debe ir modificando su tamaño. Por lo tanto, es necesario definir un atributo que represente esta condición. Se utilizará para ello la variable *largo*. Poniendo en marcha puede escribirse el siguiente programa de la siguiente manera:

```
programa Cap6Ejemplo3
procesos
    proceso cuadrado (E lado : numero)
    comenzar
        {el lado tiene tantas cuadras como indica largo}
        repetir 4
            repetir lado
                mover
                derecha
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    variables
        largo : numero
    comenzar
        {valor inicial, el primero cuadrado es de uno}
        largo := 1
        repetir 4
            cuadrado(largo)
            {se incrementa el valor que indica el largo del lado}
            largo := largo + 1
        fin
    variables
        R-info : robot1
    comenzar
        AsignarArea(R-info,ciudad)
        Iniciar(R-info, 1 , 1)
    fin
```

A continuación se describe la ejecución del algoritmo. La segunda instrucción asigna el valor 1 a la variable *largo*, el cual se corresponde con el tamaño del lado del primer cuadrado. Cuando se llama al proceso cuadrado, se le manda al mismo la información correspondiente a la medida del lado. En esta primera invocación se le pasa el valor 1. Notemos que en el encabezado del proceso se encuentra definido un dato, *lado*, el cual se corresponde con la variable *largo* utilizada en la invocación; pero que, como se

observa, se lo llama con nombre diferente. El valor de la variable *largo* se copia en el dato *lado*. A partir de este momento, *lado* puede ser utilizado en el proceso cuadrado con el valor que recibió. Observemos también que cuando se indica que el valor de *largo* se copia en *lado* es equivalente a asignar a *lado* el valor de *largo*.

El proceso realiza un cuadrado de lado 1. Cuando el mismo finaliza, la instrucción siguiente a cuadrado consiste en aumentar en uno el valor de la variable *largo*, teniendo ahora el valor 2. Esto se repite tres veces más, completando el recorrido.

**Ejemplo 6.4:** Se desea programar al robot para que realice el recorrido de la figura 6.2

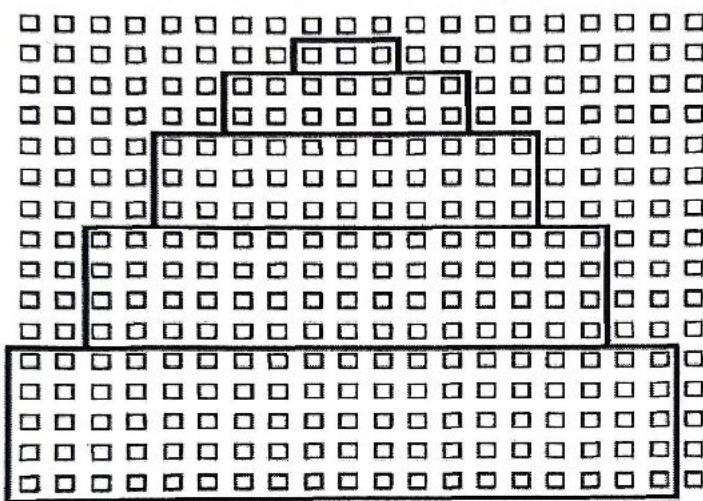


Figura 6.2: Torre de Rectángulos

Puede verse que la figura presenta diferentes rectángulos. El primero (el de más abajo) de 19 cuadras de base por 5 de alto, y el último (el de más arriba) de 3 por 1, o sea que cada rectángulo difiere con su superior en 4 cuadras de base y 1 cuadra en su altura.

El diseño Top-Down correspondiente al problema se muestra en la figura 6.3. El esquema del algoritmo quedará como:

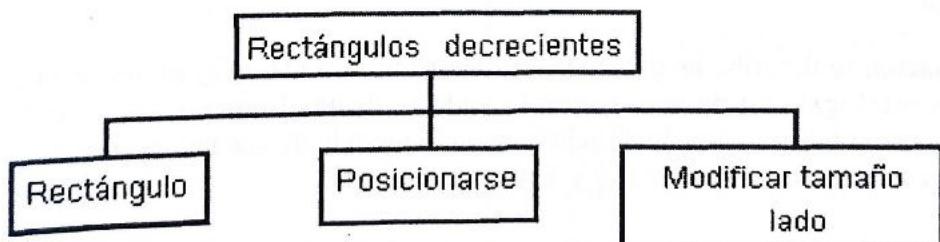


Figura 6.3: Descomposición Top-Down del ejemplo 6.4

{ torre de rectángulos }  
{ realizar los cinco rectángulos, como sigue ... }  
{ hacer un rectángulo }



{ posicionarse para el siguiente }  
{ modificar las dimensiones }

El rectángulo, a diferencia del cuadrado del ejemplo anterior, necesita dos elementos para indicar su tamaño. Por lo tanto, es necesario definir dos variables que representen esta condición. Para ello se utilizarán las variables *ancho* y *alto*.

Detallando lo anterior, la implementación de esta solución es:

```
programa Cap6Ejemplo4
procesos
    proceso Rectangulo (E base : numero; E altura : numero)
    comenzar
        repetir 2
            repetir altura
                mover
                derecha
            repetir base
                mover
                derecha
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    variables
        ancho, alto : numero
    comenzar
        ancho := 19 {1}
        alto := 5 {2}
        repetir 5
            Rectangulo(ancho,alto) {3}
            Pos(PosAv + 2, PosCa + alto) {4}
            ancho := ancho - 4 {5}
            alto := alto - 1 {6}
        fin
    variables
        R-info : robot1
    comenzar
        AsignarArea(R-info,ciudad)
        Iniciar(R-info, 1 , 1)
    fin
```

A continuación se describe la ejecución del algoritmo. En (1) y (2) se define el tamaño del primer rectángulo (el de más abajo). Cuando se llama al proceso *Rectangulo*, se le manda al mismo información correspondiente a la medida de sus lados. En esta primera invocación se le pasan los valores 19 y 5 respectivamente.

El proceso realiza un rectángulo de 19 por 5. Cuando finaliza, se devuelve el control al robot en la línea siguiente a (3) donde se posiciona al robot para realizar el siguiente rectángulo (4). Luego en (5) y (6) se modifican adecuadamente los valores que definen *alto* y *ancho* del próximo rectángulo.

En el problema planteado se observa que es necesario que el módulo reciba información. En este caso, el proceso Rectangulo recibe dos parámetros de entrada, *base* y *altura*, que son enviados por el robot a través de las variables *ancho* y *alto*, respectivamente.

**Ejemplo 6.5:** Supongamos que se dispone del siguiente proceso:

```
proceso escalon(E entra1 : numero, E entra2 : numero)
```

donde *entra1* corresponde a la altura y *entra2* corresponde al ancho de un escalón. Utilice el proceso anterior para resolver el recorrido de la figura 6.4.

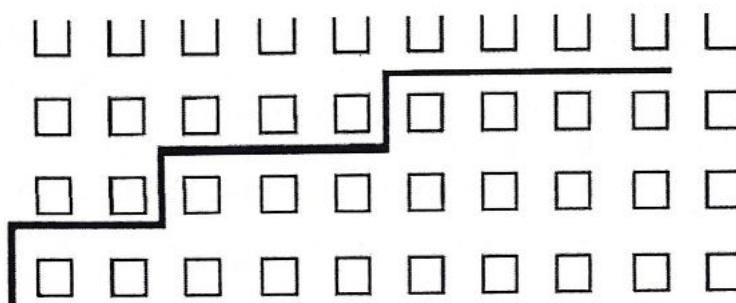


Figura 6.4: Recorrido en escalones

A continuación se presentan dos alternativas para resolver este problema:

```
programa Cap6Ejemplo5V1
procesos
    proceso escalon (E
entra1:numero;
                    E entra2:numero)
comenzar
    repetir entra1
        mover
        derecha
        repetir entra2
            mover
        repetir 3
            derecha
    fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    escalon(1,2)
    escalon(1,3)
    escalon(1,4)
fin
variables
    R-info : robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info, 1 , 1)
fin
```

```
programa Cap6Ejemplo5V2
procesos
    proceso escalon(E
entra1:numero;
                    E
entra2:numero)
comenzar
    repetir entra1
        mover
        derecha
        repetir entra2
            mover
        repetir 3
            derecha
    fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    long : numero
comenzar
    long := 2
    repetir 3
        escalon (1,long)
        long:= long + 1
    fin
variables
    R-info : robot1
comenzar
```



	AsignarArea(R-info, ciudad) Iniciar (R-info, 1, 1) <b>fin</b>
--	---

Compare las soluciones anteriores e indique:



¿Ambas soluciones realizan el mismo recorrido?

¿Cuál de las dos soluciones elegiría si debe continuar el recorrido en forma de escalera hasta completar 20 escalones donde cada escalón tiene una cuadra más de ancho que el anterior?

El ejemplo anterior pretende mostrar que para utilizar un proceso sólo es necesario conocer qué hace y no cómo lo hace. Además, toda la información necesaria para interactuar con el módulo se encuentra resumida en su primera línea, ya que allí aparece, no sólo el nombre del proceso sino además su lista de parámetros.

A continuación se indican dos posibles implementaciones del proceso escalón:

```
proceso escalon (E entral:numero;  
                  E entra2:numero)  
  
comenzar  
    repetir entral  
        mover  
        derecha  
    repetir entra2  
        mover  
        izquierda  
    fin
```

```
proceso escalon(E entral:numero;  
                 E entra2: numero)  
  
comenzar  
    Pos(PosAv+entra2, PosCa+Entral)  
    izquierda  
    repetir entra2  
        mover  
        izquierda  
    repetir entral  
        mover  
        Pos(PosAv+entra2, PosCa+Entral)  
    repetir 2  
        derecha  
fin
```



¿Es posible implementar el proceso escalón de manera que el primer parámetro represente la altura y el segundo el ancho? Notemos que podemos elegir cualquier nombre para los parámetros formales (no importa si no se llaman *entral* y *entra2*).

## 6.5 Restricción en el uso de los parámetros de entrada

En los ejemplos anteriores se han desarrollado procesos con parámetros de entrada y en todos los casos, la información recibida de esta forma ha sido utilizada como “de lectura”. En otras palabras, en ninguno de los ejemplos se ha asignado un valor sobre un parámetro de entrada.

Esta restricción es tanto conceptual como sintáctica. Desde el punto de vista conceptual, no tiene sentido modificar el valor de un parámetro de entrada ya que es información recibida desde el módulo que realizó la invocación, el cual no espera recibir ninguna respuesta a cambio. Desde el punto de vista sintáctico, no es posible asignar dentro del proceso un valor al parámetro de entrada.

A continuación se exemplificará esta restricción:

**Ejemplo 6.6:** Escriba un proceso que le permita al robot recorrer la avenida donde se encuentra parado, desde la calle 1 hasta la calle 10 dejando en cada esquina una flor menos que en la esquina anterior. La cantidad de flores de la primera esquina se recibe como parámetro y se garantiza que este valor es mayor que 10 (seguro tengo algo que depositar en cada una de las 10 esquinas). El proceso termina cuando el robot haya recorrido las 10 esquinas. El proceso será invocado con el robot ubicado al comienzo de la avenida con dirección norte. Por simplicidad considere que lleva en su bolsa la cantidad de flores necesarias.

Por ejemplo, si el robot debe dejar en la primera esquina 12 flores, el recorrido consistirá en depositar: 12 flores en la primera, 11 en la 2da., 10 en la 3ra.y 9 en la 4ta. Para esto es necesario que el proceso reciba como parámetro la cantidad de la primera esquina

A continuación se detalla una implementación que presenta un error referido a la restricción del parámetro de entrada:

```
proceso UnaMenosV1 ( E FloresIniciales : numero )
comenzar
{Cada esquina tendrá una flor menos que la anterior}
mientras (PosCa<11)
{Depositar las flores indicadas (seguro puede hacerlo)}
    repetir FloresIniciales
        depositarFlor
        mover
        FloresIniciales := FloresIniciales - 1
    Fin
(1)
```

El proceso anterior utiliza una iteración para controlar que el robot no intente depositar en la calle 11.

Por otro lado, la repetición deposita las flores sin preguntar si hay en la bolsa porque es una precondition de este ejemplo que el robot lleva la cantidad de flores necesarias.

El problema de esta implementación se encuentra en la línea (1) donde se asigna un valor en el parámetro de entrada. Esto NO es válido en la sintaxis del ambiente de programación del robot. Para poder hacerlo, será necesario recurrir a una variable auxiliar, que sólo será conocida dentro del proceso.

La implementación correcta es la siguiente:

```
proceso UnaMenosV2 ( E FloresIniciales : numero)
variables
    cuantas : numero
comenzar
    cuantas := FloresIniciales
    {Cada esquina tendrá una flor menos que la anterior}
    mientras (PosCa<11)
        repetir cuantas
            depositarFlor
    Fin
(1)
```



```
mover
cuantas := cuantas - 1
fin
```

De esta forma, el parámetro de entrada sólo es leído al inicio del proceso, en la línea (1) y a partir de allí se utiliza la variable local.

Si bien esta forma de utilizar los parámetros de entrada puede resultar restrictiva, es importante recordar que durante todos los ejemplos anteriores no fue preciso modificar el valor del parámetro. Esto no es casual, sino que se encuentra asociado a la función que cumple la información recibida por el proceso. Si se trata de información de entrada, es de esperar que su valor permanezca SIN modificación alguna dentro del módulo llamado.

El hecho de tener que conocer para cada esquina la cantidad de flores a depositar es independiente de la información recibida inicialmente referida a la cantidad de flores de la primera esquina. Si el proceso necesita manejar este dato, deberá utilizar sus propias variables para hacerlo.

- Se propone rehacer el ejemplo 6.6 para que el robot pueda aplicar esta distribución de flores a toda la avenida. Además se desconoce si inicialmente posee la cantidad de flores necesarias para hacerlo.

## 6.6 Conclusiones

En este capítulo se han presentado los parámetros de entrada como medio de comunicación entre módulos.

Este tipo de parámetros permite que el proceso llamado reciba información de quien lo llama. Dicha información podrá ser utilizada internamente por el proceso para adaptar su comportamiento.

Utilizando parámetros de entrada, la información viaja en un único sentido, desde el módulo que llama hacia el módulo llamado.



# Capítulo 7

## Parámetros de entrada/salida



### Objetivos

Continuando con los mecanismos de comunicación entre módulos se incorporarán en este capítulo los parámetros de entrada/salida que, como su nombre lo indica, permiten realizar un intercambio de información entre módulos, en ambos sentidos.

Este tipo de parámetros, si bien puede utilizarse en reemplazo de los parámetros de entrada, es recomendable utilizarlos solo en aquellos casos en que la comunicación entre los módulos lo justifique. De esta manera se reducirá la aparición de errores no deseados.



### Temas a tratar

- ✓ Introducción
- ✓ Ejemplos
- ✓ Otro uso de los Parámetros de Entrada/Salida.
- ✓ Conclusiones
- ✓ Ejercitación



## 7.1 Introducción

Un módulo utiliza un parámetro de entrada/salida cuando necesita recibir un dato, procesarlo y devolverlo modificado. También se utiliza para que un módulo pueda darle información al módulo que lo llamó.

El parámetro de entrada/salida permite realizar ambas operaciones sobre el mismo parámetro, ampliando de esta forma las posibilidades de comunicación.

Si bien este aspecto puede parecer ventajoso en primera instancia, es importante considerar que el uso de este tipo de parámetros resta independencia al módulo llamado ya que su funcionamiento depende de la información recibida.

## 7.2 Ejemplos

**Ejemplo 7.1:** Programe al robot para que informe la cantidad total de flores que hay en la avenida 4. No se debe modificar la cantidad de flores de cada esquina.

```
programa Cap7Ejemplo1
procesos
    proceso SumarFloresEsquina (ES flores : numero)           {2}
    variables
        aux : numero
    comenzar
        aux:= 0
        mientras HayFlorEnLaEsquina
            tomarFlor
            aux:=aux+1
            flores:=flores+1
        repetir aux
            depositarFlor
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
        variables
            totalFlores: numero
        comenzar
            Pos(4,1)
            totalFlores:=0
            repetir 99
                SumarFloresEsquina(totalFlores)
                mover
                SumarFloresEsquina(totalFlores)
                Informar(totalFlores)           {1}
            fin
        variables
            R-info : robot1
        comenzar
            AsignarArea(R-info,ciudad)
            Iniciar(R-info, 1 , 1)
        fin
{3}
```

En Cap7Ejemplo1, el proceso SumarFloresEsquina recibe, en cada invocación, el total de flores encontradas hasta el momento y sobre este valor, continúa acumulando las flores. Esto puede verse en (1), donde al realizar la invocación se utiliza *a totalFlores* como parámetro. En (2) se especifica que este parámetro es de entrada/salida. Es decir, al comenzar la ejecución del proceso SumarFloresEsquina, este recibe en *flores* el valor del parámetro actual *totalFlores*, sobre este valor continúa acumulando la cantidad de flores encontradas y al finalizar, el valor del parámetro formal *flores* será devuelto al programa principal a través de *totalFlores*, reflejando de esta forma las modificaciones realizadas dentro del proceso.

Es importante ver que el objetivo del proceso SumarFloresEsquina es modificar la cantidad de flores encontradas hasta el momento sumándole la cantidad de flores de la esquina actual.

**Ejemplo 7.2:** Programe al robot para que recorra todas las avenidas de la ciudad e informe la cantidad total de flores encontradas.

La descomposición Top-Down del problema se muestra en la figura 7.1

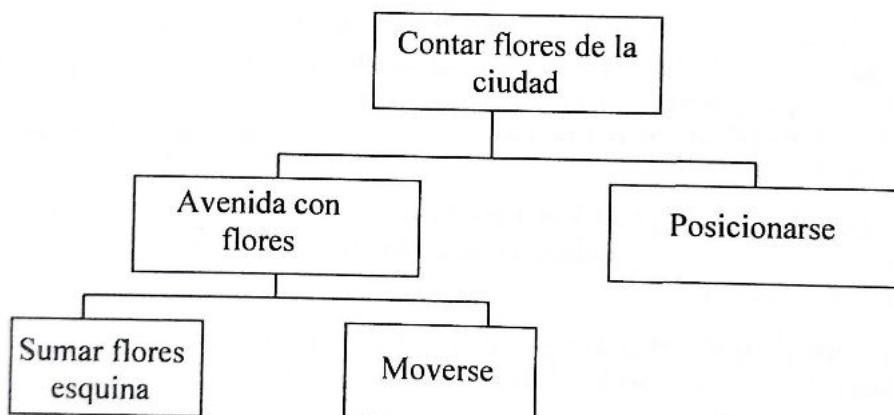


Fig. 7.1: Top-Down del ejemplo 7.2

El algoritmo es de la forma:

{Para cada avenida de la ciudad}  
    {Recorrer la avenida incrementando la cant. de flores encontradas}  
    {Posicionarse en la próxima avenida}  
    {Recorrer la última avenida}  
    {Informar el total de flores encontradas}

El programa utiliza dos módulos: uno para contar las flores de la esquina que ya fue definido en el ejemplo 7.1 y otro para recorrer la avenida.

El proceso que recorre la avenida es el siguiente:



```
proceso AvenidaConFlores( ES Total : numero )
variables
    cuantas : numero
comenzar
    repetir 99
        SumarFloresEsquina (cuantas)
        Total := Total + cuantas
        mover
        { esq. de la calle 100}
        SumarFloresEsquina (cuantas)
        Total := Total + cuantas
    fin
```

Como puede verse, posee un parámetro de entrada/salida para registrar el total de flores del recorrido. Cada vez que el proceso es invocado recibe como entrada la cantidad de flores encontradas hasta el momento, sobre este valor agrega las flores de esta avenida y lo devuelve modificado. El programa completo es el siguiente:

```
programa Cap7Ejemplo2
procesos
    proceso SumarFloresEsquina (ES flores : numero)
variables
    aux : numero
comenzar
    aux:= 0
    mientras HayFlorEnLaEsquina
        tomarFlor
        aux:=aux+1
        flores:=flores+1
    repetir aux
        depositarFlor
    fin
    proceso AvenidaConFlores(ES Total : numero)
variables
    cuantas : numero
comenzar
    repetir 99
        cuantas := 0
        SumarFloresEsquina(cuantas)
        Total := Total + cuantas
        mover
        {Esquina de la calle 100}
        SumarFloresEsquina(cuantas)
        Total := Total + cuantas
    fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    totalFlores: numero
comenzar
    totalFlores := 0 {1}
    repetir 99
        AvenidaConFlores(totalFlores) {2}
```

```

Pos(PosAv + 1 , 1)
AvenidaConFlores(totalFlores)
Informar(totalFlores) {3}
fin
variables
R-info : robot1
comenzar
AsignarArea(R-info,ciudad)
Iniciar(R-info, 1 , 1)
fin

```

En este código puede verse que, el determinar correctamente la cantidad de flores de la ciudad es responsabilidad tanto del robot R-info como del proceso AvenidaConFlores.

En (1) el robot asigna el valor 0 a la variable *TotFlores* como forma de representar que hasta el momento no se ha encontrado ninguna flor. En (2), al producirse la primer invocación al proceso avenida, se le envía el valor 0 que es recibido por el parámetro formal de entrada/salida, *total*. Durante la ejecución del proceso AvenidaConFlores, *total* se va incrementando con las flores encontradas en esa avenida. Al finalizar la avenida, se asigna este valor sobre el parámetro actual, *TotFlores*, permitiendo que el programa principal conozca la cantidad de flores encontradas en la avenida 1.

Luego de posicionarse en la avenida 2 se invoca nuevamente al proceso enviándole la cantidad de flores encontradas en la avenida 1. El proceso recibe esta cantidad y la incrementa con el total de flores de la avenida 2. Al finalizar, asigna nuevamente en *TotFlores* este valor permitiendo que el robot conozca la cantidad de flores encontradas en las primeras dos avenidas.

Esto se repite para las 98 avenidas restantes por lo cual en (3) se informará la cantidad de flores encontradas en todas las avenidas de la ciudad.

**Ejemplo 7.3:** Modifique la implementación del ejemplo 6.4 para que el robot informe al finalizar su recorrido, la cantidad total de vértices que tienen flores (al menos una).

El programa que sigue muestra la solución implementada:

```

programa Cap7Ejemplo3
procesos
    proceso Rectangulo (E base : numero; E altura : numero;
                        ES cantidad : numero)
    comenzar
        repetir 2
            si HayFlorEnLaEsquina
                cantidad := cantidad + 1
            repetir altura
                mover
                derecha
                si HayFlorEnLaEsquina
                    cantidad := cantidad +1
            repetir base
                mover
                derecha

```



```
fin
areas
    ciudad: AreaC(1, 1, 100, 100)
robots
    robot robot1
variables
    ancho, alto, cantVertices : numero
comenzar
    cantVertices := 0
    ancho := 19
    alto := 5
    repetir 5
        Rectangulo(ancho, alto, cantVertices) {1}
        Pos(PosAv + 2, PosCa + alto)
        ancho := ancho - 4
        alto := alto - 1
        Informar(cantVertices)
    fin
variables
    R-info : robot1
comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1, 1)
fin
```

En el punto (1), *cantVertices* toma el valor 0, indicando que aún no se han encontrado flores en los vértices de ningún rectángulo.

En el punto (2), se invoca al proceso *Rectangulo* con tres parámetros. Los dos primeros, *ancho* y *alto*, definen las dimensiones del rectángulo y el tercer parámetro *cantVertices* representa el total de vértices con flor encontrados en el recorrido.

En este caso, el proceso *Rectangulo* posee dos parámetros de entrada, *base* y *altura*, en los cuales recibe los valores de los parámetros actuales *ancho* y *alto* del programa principal, respectivamente. Además, el proceso *Rectangulo*, posee un parámetro de entrada/salida, *cantidad*, utilizado para recibir la cantidad de vértices con flor encontrados hasta el momento e incorporarle la cantidad hallada en este rectángulo. Al terminar al proceso, el valor final de *cantidad* será asignado a *cantVertices* del robot.

Es importante hacer notar que, en las sucesivas invocaciones al proceso *Rectangulo*, como el dato *cantVertices* se relaciona con el parámetro de entrada/salida *cantidad*, este dato entra al proceso con el valor que indica la cantidad de vértices con flores encontrada hasta el momento. Durante la ejecución de este proceso podría modificarse su valor y el efecto se verá reflejado nuevamente en la variable *cantVertices* del robot.

## 7.3 Otro uso de los parámetros de Entrada/Salida.

Los parámetros de entrada/salida, por permitir la comunicación en ambos sentidos, pueden ser utilizados para reemplazar a los parámetros de entrada (aunque no es recomendable) ó bien para que únicamente retornen valores.



**Ejemplo 7.4:** Programe al robot para que recorra la calle 10 e informe la cantidad total de esquinas que contienen exactamente 4 papeles.

Para resolver este problema podemos pensar en un proceso que cuenta los papeles de una esquina.

```
proceso ContarPapeles ( ES papeles
:numero)
comenzar
    papeles:= 0
    mientras HayPapelEnLaEsquina
        tomarPapel
        papeles := papeles + 1
    repetir papeles
        depositarPapel
fin
```

Como podemos observar, el proceso ContarPapeles recibe el parámetro de entrada/salida *papeles* y lo primero que hace es inicializarlo en 0 para poder saber cuántos papeles hay en la esquina donde está parado. Al terminar la ejecución del proceso el parámetro *papeles* contiene la cantidad de papeles de esa esquina.

A continuación se presenta la solución completa en el ambiente de programación del robot R-info:

```
programa Cap7Ejemplo4
procesos
    proceso ContarPapeles (ES papeles : numero)
    comenzar
        papeles := 0
        mientras HayPapelEnLaEsquina
            tomarPapel
            papeles := papeles + 1
        repetir papeles
            depositarPapel
    fin
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    totalEsquina4Papeles, papelesEsquina : numero
comenzar
    Pos(1,10)
    derecha
    totalEsquina4Papeles := 0 {1}
    repetir 99
        ContarPapeles(papelesEsquina) {2}
        si (papelesEsquina = 4) {3}
            totalEsquina4Papeles := totalEsquina4Papeles + 1
        mover
    {Falta esquina 100 , 10}
    si papelesEsquina = 4
        totalEsquina4Papeles := totalEsquina4Papeles + 1
```



```
    Informar(totalEsquina4Papeles)
    fin
variables
    R-info : robot1
comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1, 1)
    fin
```

En el punto (1) utilizamos una variable *totalEsquinas4Papeles* para saber cuántas esquinas tienen exactamente 4 papeles, inicializándola en 0 porque hasta ese momento no hemos contado nada. En (2), se invoca al proceso ContarPapeles con el parámetro formal *papelesEsquina* que no tiene ningún valor, pero como ya vimos, se inicializará en 0, ni bien comience a ejecutarse ese módulo. En el punto (3) se evalúa la cantidad de papeles de la esquina a través de *papelesEsquina* que devolvió el proceso ContarPapeles y si es 4 entonces se actualiza el contador *totalEsquinas4Papeles*.

**Ejemplo 7.5:** Programe al robot para que informe la cantidad de calles que contienen más de 50 flores.

Se puede utilizar el módulo desarrollado en el ejemplo 7.1, el código sería el siguiente:

```
programa Cap7Ejemplo5
procesos
    proceso SumarFloresEsquina (ES flores : numero) {1}
    comenzar
        / este proceso fue definido en el ejemplo 7.1 /
    fin
    proceso RecorrerCalle ( ES totalFlores : numero ) {2}
    comenzar
        totalFlores := 0
        repetir 99
            SumarFloresEsquina (totalFlores)
            mover
            SumarFloresEsquina (totalFlores)
        fin
    areas
        ciudad: AreaC(1,1,100,100)
    robots
        robot robot1
    variables
        floresCalle, totalCalle50Flores: numero
    comenzar
        derecha
        totalCalle50Flores := 0
        repetir 99
            RecorrerCalle(floresCalle)
            si floresCalle > 50
                totalCalle50Flores := totalCalle50Flores + 1
                Pos(1,PosCa +1)
            RecorrerCalle(floresCalle)
            si floresCalle > 50
                totalCalle50Flores := totalCalle50Flores + 1
            Informar(totalCalle50Flores)
        fin
    variables
        R-info : robot1
```

```
comenzar
    AsignarArea(R-info, ciudad)
    Iniciar(R-info, 1 , 1)
fin
```

Notemos que en (1) y (2), los parámetros son de entrada/salida. En (1) el módulo utiliza el parámetro flores se utiliza como entrada y salida porque sobre este dato se va acumulando el total de flores de todas las esquinas de una calle. En cambio en (2) el módulo utiliza el parámetro TotalFlores únicamente como salida. Para un uso correcto de este parámetro, el programador no debe olvidar la inicialización al comenzar el proceso, porque de lo contrario podrían obtenerse resultados erróneos.

En resumen, un proceso que utiliza parámetros de entrada/salida como únicamente de salida es totalmente independiente del módulo que lo invoca. Cuando se usa el parámetro en este sentido, se trata de información generada dentro del proceso que se desea dar a conocer al módulo que lo llamó. En este caso no se busca un intercambio de información en ambos sentidos, solo el proceso es quien exporta datos.



Haciendo clic en el siguiente link podés acceder a una animación con un ejemplo de análisis y resolución de un *Ejercicio con Parámetros*:  
[Animación Ejercicio con Parámetros](#)

## 7.4 Conclusiones

En este apunte se ha buscado introducir algunas ideas útiles a lo largo de la carrera en Informática:

1. Resultan de interés los problemas “solubles por computadora”, es decir expresables como algoritmos.
2. No solo se debe entender como son los problemas, sino aprender a modelizarlos y a resolverlos en forma ordenada y sistemática.
3. No basta con tener UNA solución. Normalmente existen varias. La elegida debe ser EFICIENTE y además la forma en que esté escrita debe ser CLARA y ENTENDIBLE.