

**Curso Inicial 2023**

**Expresión de Problemas  
y Algoritmos (EPA)**

**Facultad de Informática**



**UNIVERSIDAD NACIONAL DE LA PLATA**

## Contenido

Bienvenida	8
Capítulo 0 - Introducción	9
¿Qué voy a aprender al leer este material?	9
¿Cómo tengo que estudiar con este material?	9
¿Qué otros recursos facilitarán la lectura de este material?	10
Contenidos	11
Capítulo1- Resolución de problemas	13
Objetivos	13
Temas a tratar	13
1.1   Introducción	14
1.2   Etapas en la resolución de problemas con computadora	14
Análisis del problema	15
Diseño de una solución	15
Especificación de algoritmos	15
Escritura de programas	15
Verificación	15
1.3 Algoritmo	16
Ejemplo 1.1	16
Ejemplo 1.2	16
Ejemplo 1.3	17
Ejemplo 1.4	17
Ejemplo 1.5	17
Ejemplo 1.6	18
1.4 Pre y Postcondiciones de un algoritmo	19
En el ejemplo 1.1	19
En el ejemplo 1.2	19
En el ejemplo 1.4	19
1.5 Elementos que componen un algoritmo	19
1.5.1 Secuencia de Acciones	19
Ejemplo 1.7	20
Ejemplo 1.8	20

Facultad de Informática

1.5.2 Selección	21
Ejemplo 1.9	22
Ejemplos 1.10	23
1.5.3 Repetición	23
Ejemplo 1.11	24
Ejemplo 1.12	24
Ejemplo 1.13	24
1.5.4 Iteración	25
Ejemplo 1.14	25
1.6 Importancia de la indentación en las estructuras de control	26
Ejemplo 1.15	27
Ejemplo 1.16	27
Ejemplo 1.17	28
1.7 Conclusiones	28
Ejercitación	29
<b>Capítulo 2 -Algoritmos y Lógica Introducción al lenguaje del Robot</b>	<b>32</b>
Objetivos	32
Temas a tratar	32
2.1 Lenguajes de Expresión de Problemas. Tipos de Lenguajes. Sintaxis y semántica en un Lenguaje.	33
2.1.1 Tipos de Lenguajes	34
2.1.2 Sintaxis y Semántica en un Lenguaje	34
2.2 Ambiente de programación del robot (Rinfo). Operaciones sobre Rinfo. Estructura general de un programa. Estilo de programación. Ambiente de programación.	35
2.2.1 Operaciones en el ambiente del robot Rinfo	36
2.2.2 Estructura general de un programa	38
2.2.2.1 Comentarios Lógicos	39
2.2.3 Estilo de programación	40
2.2.4 Ambiente de programación	40
2.2.5 Comenzando a trabajar	42
2.3 Estructuras de Control	43
2.3.1 Secuencia	43
Ejemplo 2.1	44
Ejemplo 2.2	44

Facultad de Informática

2.3.2 Selección	45
Ejemplo 2.3	45
Ejemplo 2.4	46
2.3.3 Repetición	46
Ejemplo 2.5	47
Ejemplo 2.7	48
Ejemplo 2.8	49
2.3.4 Iteración	49
Ejemplo 2.9	50
Ejemplo 2.10	50
Ejemplo 2.11	50
Ejemplo 2.12	51
Análisis de la sintaxis del robot	52
2.4 Proposiciones atómicas y moleculares, simbolización y tablas de verdad	53
2.4.1 Proposiciones atómicas y moleculares	53
2.4.2 Simbolización	54
2.4.3 Tablas de verdad. Repaso	55
2.4.3.1 Conjunción. Tabla de verdad	56
2.4.3.2 Disyunción. Tabla de verdad	57
2.4.3.3 Negación. Tabla de verdad	58
2.4.4 Utilización del paréntesis	59
2.5 Conclusiones	60
Ejercitación	61
 Capítulo 3-Datos	62
Objetivos	62
Temas a tratar	62
 3.1 Conceptos de Control y Datos	63
3.2 Representación de los Datos	64
3.3 Variables	64
3.3.1 Sintaxis para la declaración de variables	64
3.4 Tipos de datos	66
3.4.1 Tipo de dato numérico (número)	66

Facultad de Informática

3.4.2 Tipo de dato lógico (boolean)	68
Ejemplo 3.1	69
Ejemplo 3.2	70
3.5 Modificación de la información representada	70
3.6 Ejemplos	72
Ejemplo 3.3	72
Ejemplo 3.4	73
Ejemplo 3.5	74
Ejemplo 3.6	76
Ejemplo 3.7	76
3.7 Representación de más de un dato dentro del algoritmo	78
Ejemplo 3.8	78
Ejemplo 3.9	79
3.8 Conclusiones	80
Ejercitación	81
<b>Capítulo 4- Repaso</b>	<b>83</b>
Objetivos	83
Temas a tratar	83
4.1 Repaso de variables	84
Ejemplo 4.1	84
4.2 Repaso de expresiones lógicas	85
Ejemplo 4.2	85
Ejemplo 4.3	86
4.3 Ejemplos	87
Ejemplo 4.4	87
Ejemplo 4.5	88
Ejemplo 4.6	89
4.4 Conclusiones	90
Ejercitación	91
<b>Capítulo 5- Programación Estructurada</b>	<b>95</b>
Temas a tratar	95
5.1 Descomposición de problemas en partes	96
5.2 Programación modular	97

Facultad de Informática

Ejemplo 5.1	100
Ejemplo 5.2	100
Ejemplo 5.3	102
Ejemplo 5.4	103
Ejemplo 5.5	106
Ejemplo 5.6	107
Ejemplo 5.7	110
Ejemplo 5.8	113
Ejemplo 5.9	114
5.3 Conclusiones	115
Ejercitación	116
<b>Capítulo 6- Parámetros de Entrada</b>	<b>118</b>
Temas a tratar	118
6.1 Comunicación entre módulos	119
6.2 Declaración de parámetros	120
6.3 Un ejemplo sencillo	121
Ejemplo 6.1	121
6.4 Ejemplos	124
Ejemplo 6.2	124
Ejemplo 6.3	128
Ejemplo 6.4	130
Ejemplo 6.5	131
6.5 Restricción en el uso de los parámetros de entrada	133
Ejemplo 6.6	133
6.6 Conclusiones	135
Ejercitación	136
<b>Capítulo 7-Parámetros de entrada/salida</b>	<b>138</b>
7.1 Introducción	139
7.2 Ejemplos	139
Ejemplo 7.1	139
Ejemplo 7.2	140
Ejemplo 7.3	142
7.3 Otro uso de los parámetros de Entrada/Salida.	143

---

Facultad de Informática	
Ejemplo 7.4	143
Ejemplo 7.5	145
7.4 Conclusiones	146
Ejercitación	147
Ejercitación adicional	148

# Bienvenida

*La Facultad de Informática desea darte la bienvenida a la Universidad Nacional de La Plata.*

Te felicitamos por haber elegido nuestra Facultad que, además de las carreras de grado, tiene una muy destacada producción científica y tecnológica y una amplia oferta de cursos de postgrado y actualización, así como convenios de capacitación con las empresas más destacadas del mercado informático mundial.

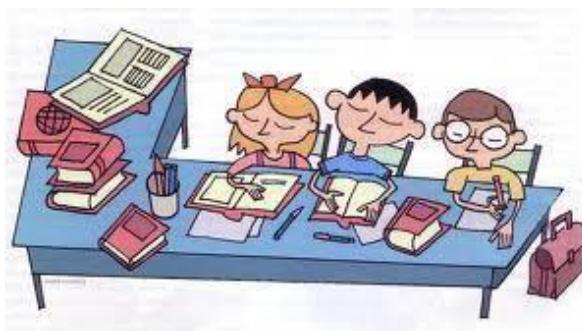
*Nuestra Facultad desea que participes en las actividades que se organizan, ya que la Informática es una disciplina en constante evolución, y donde las continuas innovaciones requieren de esfuerzo y dedicación como ingredientes esenciales para estar actualizado.*

Por otra parte, ante cualquier duda podés consultar la página de Internet de nuestra Facultad ([www.info.unlp.edu.ar](http://www.info.unlp.edu.ar)) donde trataremos de reflejar todos los datos que sean útiles para tu información.

*La Facultad de Informática se esfuerza por mantener un ambiente que fomente la interrelación entre los docentes, graduados, alumnos y no docentes.*

*Las autoridades de la Facultad de Informática quedan a tu disposición para cualquier duda y aclaración.*

*Ante cualquier consulta envíá un mail a: [ingreso@info.unlp.edu.ar](mailto:ingreso@info.unlp.edu.ar)*



# Capítulo 0

## Introducción



*¿Qué voy a aprender al leer este material?  
¿Cómo tengo que estudiar con este material?  
¿Cómo voy a evaluar mi aprendizaje  
¿Qué otros recursos facilitarán la lectura de este material?*

### ¿Qué voy a aprender al leer este material?

El objetivo de este material es que adquieras una metodología básica para la resolución de problemas utilizando una computadora.

Comenzaremos analizando el problema a resolver, luego se propondrá una especificación clara de la manera de solucionarlo y finalmente se expresará esa solución en un lenguaje de programación.

### ¿Cómo tengo que estudiar con este material?

Este curso constará de siete capítulos que presentan los conceptos básicos que resultan significativos para la materia Algoritmos Datos y Programas del 1<sup>er</sup> año de las Carreras de Informática.

Asimismo, **cada Capítulo se divide en Temas** que permiten administrar mejor tus tiempos de estudio, manteniendo la continuidad de cada tema.

- Cada Capítulo tiene como objetivo la introducción de temas específicos y están acompañados con actividades de lectura, análisis, comprensión y ejercitación.
- En los contenidos desarrollados encontrarás **toda** la información necesaria para alcanzar dichos objetivos.
- Cada uno de los Capítulos/ Temas tratados llevan asociados uno ó más ejercicios resueltos que te permitirán analizarlos con más detalle y de una forma guiada para la mejor comprensión del tema.
- Al final de cada Capítulo encontrarás la **ejercitación**. Consiste de un conjunto de preguntas o planteo de problemas a los que deberás responder de acuerdo a la información estudiada. Es importante que resuelvas esta ejercitación.
- Este material utilizará una iconografía particular para destacar las secciones importantes del contenido, por ejemplo, distinguiendo las definiciones, los recordatorios, el momento de reflexión, la ejercitación, las evaluaciones, etc.

Si bien contarás con todo el material necesario y las actividades propuestas hay otros aspectos a tener en cuenta. Estos aspectos se basan principalmente en tu **responsabilidad**.

Es sabido que un proceso de aprendizaje es básicamente un **compromiso** que el alumno asume consigo mismo.

Tené en cuenta que la **dedición** puesta, la **administración conveniente de los tiempos**, la **lectura cuidadosa**, el **espacio de consultas** con el docente, el repaso toda vez que lo consideres necesario, **colaborarán** para que el resultado de este curso resulte exitoso.

### ¿Qué otros recursos facilitarán la lectura de este material?

A continuación se presentan los íconos (representaciones gráficas de conceptos) que encontrarás en el material del Curso y que te orientarán a lo largo del estudio.



**Objetivos**



**Actividad de reflexión**



**Ejercitación**



**Temas a tratar**



**Enlace a  
Material adicional**

# Contenidos

## Capítulo 1: Resolución de Problemas

Introducción.

Etapas en la resolución de problemas con computadora.

Algoritmo.

Pre y Postcondición de un algoritmo.

Elementos que componen un algoritmo: Secuencia de Acciones, Selección, Repetición e Iteración.

Importancia de la indentación en las estructuras de control.

Conclusiones.

Ejercitación.

## Capítulo 2: Algoritmos y Lógica. Introducción al ambiente de Programación del robot R-info

Lenguajes de Expresión de Problemas: Objetivo, Concepto de Lenguajes de Expresión de Problemas, Tipos de Lenguajes, Sintaxis y Semántica en un Lenguaje.

El ambiente del Robot R-info.

Estructura general de un programa.

Operaciones sobre el ambiente del robot R-info.

Estructuras de Control: Secuencia, Selección, Repetición e Iteración en el ambiente de programación de R-info.

Repasso de proposiciones atómicas y moleculares, simbolización y Tablas de verdad.

Conectivos lógicos: Conjunción, Disyunción y Negación. Utilización del paréntesis.

Aplicación en el ambiente de programación de R-info.

Estilo de Programación y comentarios lógicos.

Ambiente de Programación R-info: Configuración Inicial.

Conclusiones.

Ejercitación.

## Capítulo 3: Datos y Aplicaciones

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 de programación del robot R-info.

Modificación de la información representada.

Ejemplos.

Comparaciones.

Representación de más de un dato dentro del programa.

Conclusiones.

Ejercitación.



## **Capítulo 4: Repaso**

Presentación, análisis y resolución de ejemplos.

Conclusiones.

Ejercitación.

## **Capítulo 5: Programación Estructurada**

Descomposición de problemas en partes.

Programación modular.

Ejemplos utilizando el lenguaje del ambiente del robot R-info.

Conclusiones.

Ejercitación.

## **Capítulo 6: Parámetros de entrada**

Comunicación entre módulos.

Declaración de parámetros.

Ejemplos.

Restricción en el uso de los parámetros de entrada.

Conclusiones.

Ejercitación.

## **Capítulo 7: Parámetros de entrada/salida**

Introducción.

Declaración de parámetros.

Ejemplos.

Conclusiones.

Ejercitación.

## **Ejercitación de repaso**

# Capítulo 1

## Resolución de problemas



### Objetivos

La resolución de problemas, utilizando como herramienta una computadora, requiere contar con la capacidad de expresión suficiente como para indicar a la máquina lo que debe llevarse a cabo.

Se comenzará resolviendo situaciones del mundo real tratando de utilizar determinados elementos que caracterizan a una secuencia de órdenes que una computadora puede comprender.

El tema central de este capítulo es la definición del concepto de algoritmo y los elementos que lo componen.



### Temas a tratar

- ✓ Introducción.
- ✓ Etapas en la resolución de problemas con computadora.
- ✓ Algoritmo.
- ✓ Pre y Postcondiciones de un algoritmo.
- ✓ Elementos que componen un algoritmo: Secuencia de Acciones, Selección, Repetición e Iteración.
- ✓ Importancia de la indentación en las estructuras de control.
- ✓ Conclusiones.
- ✓ Ejercitación.

## 1.1 Introducción

La Informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

La palabra ciencia se relaciona con una metodología fundamentada y racional para el estudio y resolución de los problemas. En este sentido la Informática se vincula especialmente con la Matemática.

Si se busca en el diccionario una definición en la palabra *problema* podrá hallarse alguna de las siguientes:

- Cuestión o proposición dudosa, que se trata de aclarar o resolver.
- Enunciado encaminado a averiguar el modo de obtener un resultado cuando se conocen ciertos datos.

La resolución de problemas mediante una computadora consiste en dar una adecuada formulación de pasos precisos a seguir.

Si se piensa en la forma en que una persona indica a otra como resolver un problema, se verá que habitualmente se utiliza un lenguaje común y corriente para realizar la explicación, quizás entremezclado con algunas palabras técnicas. Esto es un riesgo muy grande. Los que tienen cierta experiencia al respecto saben que es difícil transmitir el mensaje y por desgracia, con mucha frecuencia se malinterpretan las instrucciones y por lo tanto se ejecuta incorrectamente la solución obteniéndose errores.

Cuando de una computadora se trata, no pueden utilizarse indicaciones ambiguas. Ante cada orden resulta fundamental tener una única interpretación de lo que hay que realizar. Una máquina no posee la capacidad de decisión del ser humano para resolver situaciones no previstas. Si al dar una orden a la computadora se produce una situación no contemplada, será necesario abortar esa tarea y recomenzar todo el procedimiento nuevamente.

Además, para poder indicar a la computadora las órdenes que debe realizar es necesario previamente entender exactamente lo que se quiere hacer. Es fundamental conocer con qué información se cuenta y qué tipo de transformación se quiere hacer sobre ella.

A continuación se analizarán en forma general las distintas etapas que deben seguirse para poder llegar a resolver un problema utilizando una computadora como herramienta.

## 1.2 Etapas en la resolución de problemas con computadora

La resolución de problemas utilizando como herramienta una computadora no se resume únicamente en la escritura de un programa, sino que se trata de una tarea más compleja. El proceso abarca todos los aspectos que van desde interpretar las necesidades del usuario hasta verificar que la respuesta brindada es correcta. Las etapas son las siguientes:

## Análisis del problema

En esta primera etapa, se analiza el problema en su contexto del mundo real. Deben obtenerse los requerimientos del usuario. El resultado de este análisis es un modelo preciso del ambiente del problema y del objetivo a resolver. Dos componentes importantes de este modelo son los datos a utilizar y las transformaciones de los mismos que llevan al objetivo.

## Diseño de una solución

La resolución de un problema suele ser una tarea muy compleja para ser analizada como un todo. Una técnica de diseño en la resolución de problemas consiste en la identificación de las partes (subproblemas) que componen el problema y la manera en que se relacionan. Cada uno de estos subproblemas debe tener un objetivo específico, es decir, debe resolver una parte del problema original. La integración de las soluciones de los subproblemas es lo que permitirá obtener la solución buscada.

## Especificación de algoritmos

La solución de cada subproblema debe ser especificada a través de un algoritmo. Esta etapa busca obtener la secuencia de pasos a seguir para resolver el problema. La elección del algoritmo adecuado es fundamental para garantizar la eficiencia de la solución.

## Escritura de programas

Un algoritmo es una especificación simbólica que debe convertirse en un programa real sobre un lenguaje de programación concreto. A su vez, un programa escrito en un lenguaje de programación determinado (ej: Pascal, Ada, etc) es traducido automáticamente al lenguaje de máquina de la computadora que lo va a ejecutar. Esta traducción, denominada compilación, permite detectar y corregir los errores sintácticos que se cometan en la escritura del programa.

## Verificación

Una vez que se tiene un programa escrito en un lenguaje de programación se debe verificar que su ejecución produce el resultado deseado, utilizando datos representativos del problema real. Sería deseable poder afirmar que el programa cumple con los objetivos para los cuales fue creado, más allá de los datos particulares de una ejecución. Sin embargo, en los casos reales es muy difícil realizar una verificación exhaustiva de todas las posibles condiciones de ejecución de un sistema de software. La facilidad de verificación y la depuración de errores de funcionamiento del programa conducen a una mejor calidad del sistema y es un objetivo central de la Ingeniería de Software.

En cada una de las etapas vistas se pueden detectar errores lo cual lleva a revisar aspectos de la solución analizados previamente.

Dada la sencillez de los problemas a resolver en este curso, la primera etapa correspondiente al análisis del problema, sólo se verá reflejada en la interpretación del enunciado a resolver. Sin embargo, a lo largo de la carrera se presentarán diferentes

Versión: 2 - Capítulo 1 –Resolución de problemas

asignaturas que permitirán familiarizar al alumno con las técnicas necesarias para hacer frente a problemas de gran envergadura.

Con respecto a la segunda etapa, se pospondrá el análisis de este tema hasta el capítulo 5, ya que se comenzará a trabajar con problemas simples que no necesitan ser descompuestos en otros más elementales.

Por lo tanto, a continuación se trabajará sobre el concepto de algoritmo como forma de especificación de soluciones concretas para la resolución de problemas con computadora.

### 1.3 Algoritmo

La palabra algoritmo deriva del nombre de un matemático árabe del siglo IX, llamado Al-Khuwarizmi, quien estaba interesado en resolver ciertos problemas de aritmética y describió varios métodos para resolverlos. Estos métodos fueron presentados como una lista de instrucciones específicas (como una receta de cocina) y su nombre es utilizado para referirse a dichos métodos.

Un algoritmo es, en forma intuitiva, una receta, un conjunto de instrucciones o de especificaciones sobre un proceso para hacer algo. Ese algo generalmente es la solución de un problema de algún tipo. Se espera que un algoritmo tenga varias propiedades. La primera es que un algoritmo no debe ser ambiguo, o sea, que si se trabaja dentro de cierto marco o contexto, cada instrucción del algoritmo debe significar sólo una cosa.

Se presentan a continuación algunos ejemplos:

#### Ejemplo 1.1:

**Problema :** Indique la manera de salar una masa.

**Algoritmo 1:** Ponerle algo de sal a la masa

**Algoritmo 2:** Agregarle una cucharadita de sal a la masa.

El algoritmo 1 presenta una solución ambigua al problema planteado.

El algoritmo 2 presenta una solución adecuada al problema.

#### Ejemplo 1.2:

**Problema:** Determinar si el número 7317 es primo.

**Algoritmo 1:** Divida el 7317 entre sus anteriores buscando aquellos que lo dividan exactamente.

**Algoritmo 2:** Divida el número 7317 entre cada uno de los números 1, 2, 3, 4, ..., 7315, 7316. Si una de las divisiones es exacta, la respuesta es no. Si no es así, la respuesta es sí.

El algoritmo 1 no especifica claramente cuáles son los valores a lo que se refiere, por lo que resulta ambiguo.

El algoritmo 2 presenta una solución no ambigua para este problema. Existen otros algoritmos mucho más eficaces para dicho problema, pero esta es una de las soluciones correctas.

### Ejemplo 1.3:

**Problema:** Determinar la suma de todos los números enteros.

En este caso no se puede determinar un algoritmo para resolver este problema. Un algoritmo debe alcanzar la solución en un tiempo finito, situación que no se cumplirá en el ejemplo ya que los números enteros son infinitos.

Además de no ser ambiguo, un algoritmo debe detenerse. Se supone también que cuando se detiene, debe informar de alguna manera, su resultado. Es bastante factible escribir un conjunto de instrucciones que no incluyan una terminación y por lo tanto dicho conjunto de instrucciones no conformarían un algoritmo.

### Ejemplo 1.4:

**Problema:** Volcar un montículo de arena en una zanja.

**Algoritmo:** Tome una pala. Mientras haya arena en el montículo cargue la pala con arena y vuélquela en la zanja. Dejar la pala.

Este algoritmo es muy simple y no ambiguo. Se está seguro que en algún momento parará, aunque no se sabe cuántas paladas se requerirán.

Resumiendo, un algoritmo puede definirse como una secuencia ordenada de pasos elementales, exenta de ambigüedades, que lleva a la solución de un problema dado en un tiempo finito.

Para comprender totalmente la definición anterior falta clarificar que se entiende por “paso elemental”.

### Ejemplo 1.5:

Escriba un algoritmo que permita preparar una tortilla de papas de tres huevos.

El enunciado anterior basta para que un cocinero experto lo resuelva sin mayor nivel de detalle, pero si este no es el caso, se deben describir los pasos necesarios para realizar la preparación. Esta descripción puede ser:

*Mezclar papas cocidas, huevos y una pizca de sal en un recipiente  
Freír*

Esto podría resolver el problema, si el procesador o ejecutor del mismo no fuera una persona que da sus primeros pasos en tareas culinarias, ya que el nivel de detalle del algoritmo presupone muchas cosas.

Si este problema debe resolverlo una persona que no sabe cocinar, se debe detallar, cada uno de los pasos mencionados, pues estos no son lo bastante simples para un principiante.

De esta forma, el primer paso puede descomponerse en:

*Pelar las papas  
Cortarlas en cuadraditos  
Cocinar las papas  
Batir los huevos en un recipiente  
Aregar las papas al recipiente y echar una pizca de sal al mismo*

El segundo paso (freír) puede descomponerse en los siguientes tres:

- Calentar el aceite en la sartén*
- Verter el contenido del recipiente en la sartén*
- Dorar la tortilla de ambos lados*

Nótese además que si la tortilla va a ser realizada por un niño, algunas tareas (por ejemplo batir los huevos) pueden necesitar una mejor especificación.

El ejemplo anterior sólo pretende mostrar que la lista de pasos elementales que compongan nuestro algoritmo depende de quién sea el encargado de ejecutarlo.

Si en particular, el problema va a ser resuelto utilizando una computadora, el conjunto de pasos elementales conocidos es muy reducido, lo que implica un alto grado de detalle para los algoritmos.

Se considera entonces como un paso elemental aquel que no puede volver a ser dividido en otros más simples. De ahora en adelante se utilizará la palabra instrucción como sinónimo de paso elemental.

Un aspecto importante a discutir es el detalle que debe llevar el algoritmo. Esto no debe confundirse con el concepto anterior de paso elemental. En ocasiones, no se trata de descomponer una orden en acciones más simples sino que se busca analizar cuáles son las órdenes relevantes para el problema. Esto resulta difícil de cuantificar cuando las soluciones son expresadas en lenguaje natural. Analice el siguiente ejemplo:

### Ejemplo 1.6:

Desarrolle un algoritmo que describa la manera en que Ud. se levanta todas las mañanas para ir al trabajo.

- Salir de la cama*
- Quitarse el pijama*
- Ducharse*
- Vestirse*
- Desayunar*
- Arrancar el auto para ir al trabajo*

Nótese que se ha llegado a la solución del problema en seis pasos, y no se resaltan aspectos como: colocarse un calzado después de salir de la cama, o abrir la llave de la ducha antes de ducharse. Estos aspectos han sido descartados, pues no tienen mayor trascendencia. En otras palabras se sobreentienden o se suponen. A nadie se le ocurriría ir a trabajar descalzo.

En cambio existen aspectos que no pueden obviarse o suponerse porque el algoritmo perdería lógica. El tercer paso, “vestirse”, no puede ser omitido. Puede discutirse si requiere un mayor nivel de detalle o no, pero no puede ser eliminado del algoritmo.

Un buen desarrollador de algoritmos deberá reconocer esos aspectos importantes y tratar de simplificar su especificación de manera de seguir resolviendo el problema con la menor cantidad de órdenes posibles.

#### 1.4 Pre y Postcondiciones de un algoritmo

Precondición es la información que se conoce como verdadera antes de comenzar el algoritmo.

#### En el ejemplo 1.1:

**Problema:** Indique la manera de salar una masa.

**Algoritmo:** Agregarle una cucharadita de sal a la masa.

Se supone que se dispone de todos los elementos para llevar a cabo esta tarea. Por lo tanto, como precondición puede afirmarse que se cuenta con la cucharita, la sal y la masa.

Postcondición es la información que se conoce como verdadera al concluir el algoritmo si se cumple adecuadamente el requerimiento pedido.

#### En el ejemplo 1.2:

**Problema:** Determinar si el número 7317 es primo.

**Algoritmo:** Divida el número 7317 entre cada uno de los números 1, 2, 3, 4, ..., 7315, 7316. Si una de las divisiones es exacta, la respuesta es no. Si no es así, la respuesta es sí.

La postcondición es que se ha podido determinar si el número 7317 es primo o no.

#### En el ejemplo 1.4:

**Problema:** Volcar un montículo de arena en una zanja.

**Algoritmo:** Tome una pala. Mientras haya arena en el montículo cargue la pala con arena y vuélquela en la zanja. Dejar la pala.



- ¿Cuáles serían las precondiciones y las postcondiciones del algoritmo? La precondición es que se cuenta con la pala, la arena y está ubicado cerca de la zanja que debe llenar.

La postcondición es que el montículo quedó vacío al terminar el algoritmo.

#### 1.5 Elementos que componen un algoritmo

##### 1.5.1 Secuencia de Acciones

Una secuencia de acciones está formada por una serie de instrucciones que se ejecutan una a continuación de la otra.

Esto se muestra gráficamente en la figura 1.1

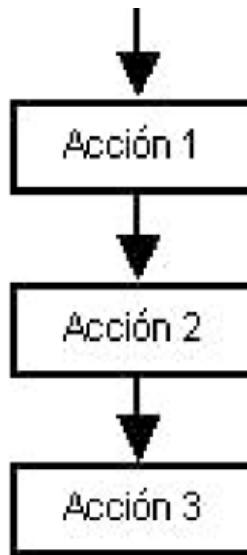


Figura 1.1: Secuencia

**Ejemplo 1.7:** Escriba un algoritmo que permita cambiar una lámpara quemada.

*Colocar la escalera debajo de la lámpara quemada*

*Tomar una lámpara nueva de la misma potencia que la anterior*

*Subir por la escalera con la nueva lámpara hasta alcanzar la lámpara a sustituir*

*Desenroscar la lámpara quemada*

*Enroscar la nueva lámpara hasta que quede apretada la nueva lámpara*

*Bajar de la escalera con lámpara quemada*

*Tirar la lámpara a la basura*

**Ejemplo 1.8:** Escriba un algoritmo que permita a un robot subir 8 escalones

*Levantar Pie Izquierdo*

*Subir un escalón*

*Levantar Pie Derecho*

*Subir un escalón*

*Levantar Pie Izquierdo*

*Subir un escalón*

*Levantar Pie Derecho*

*Subir un escalón*

*Levantar Pie Izquierdo*

*Subir un escalón*

*Levantar Pie Derecho*

*Subir un escalón*

*Levantar Pie Izquierdo*

*Subir un escalón*

*Levantar Pie Derecho*

*Subir un escalón*

Versión: 2 - Capítulo 1 –Resolución de problemas

Se denomina flujo de control de un algoritmo al orden en el cual deben ejecutarse los pasos individuales.

Hasta ahora se ha trabajado con flujo de control secuencial, es decir, la ejecución uno a uno de los pasos, desde el primero hasta el último.

Las estructuras de control son construcciones algorítmicas que alteran directamente el flujo de control secuencial del algoritmo.

Con ellas es posible seleccionar un determinado sentido de acción entre un par de alternativas específicas o repetir automáticamente un grupo de instrucciones.

A continuación se presentan las estructuras de control necesarias para la resolución de problemas más complejos.

### 1.5.2 Selección

La escritura de soluciones a través de una secuencia de órdenes requiere conocer a priori las diferentes alternativas que se presentarán en la resolución del problema. Lamentablemente, es imposible contar con esta información antes de comenzar la ejecución de la secuencia de acciones.

Por ejemplo, que ocurriría si en el ejemplo 1.7 al querer sacar la lámpara quemada, el portalámparas se rompe. Esto implica que el resto de las acciones no podrán llevarse a cabo por lo que el algoritmo deberá ser interrumpido. Si se desea que esto no ocurra, el algoritmo deberá contemplar esta situación. Nótese que el estado del portalámparas es desconocido al iniciar el proceso y sólo es detectado al intentar sacar la lámpara quemada. Por lo que usar solamente la secuencia planteada es insuficiente para expresar esta solución.

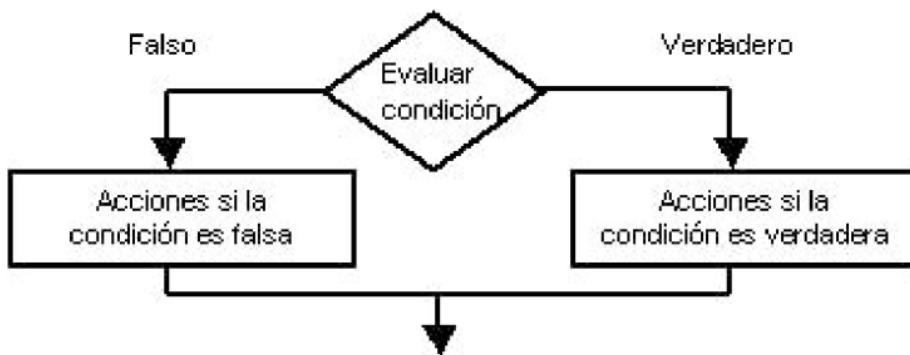


Figura 1.2: Estructura Si-Entonces-Sino

A través de la selección se incorpora, a la especificación del algoritmo, la capacidad de decisión. De esta forma será posible seleccionar una de dos alternativas de acción posibles durante la ejecución del algoritmo.

Por lo tanto, el algoritmo debe considerar las dos alternativas, es decir, qué hacer en cada uno de los casos. La selección se notará de la siguiente forma:

**si** (condición)

acción o acciones a realizar si la condición es verdadera (1)

sino

*acción acciones a realizar si la condición es falsa (2)*

donde “condición” es una expresión que al ser evaluada puede tomar solamente uno de dos valores posibles: verdadero o falso.

El esquema anterior representa que en caso de que la condición a evaluar resulte verdadera se ejecutarán las acciones de (1) y NO se ejecutarán las de (2). En caso contrario, es decir, si la condición resulta ser falsa, solo se ejecutarán las acciones de (2).

En la figura 1.2 se grafica la selección utilizando un rombo para representar la decisión y un rectángulo para representar un bloque de acciones secuenciales.

Analice el siguiente ejemplo:

**Ejemplo 1.9:** Su amigo le ha pedido que le compre \$1 de caramelos en el kiosco. De ser posible, prefiere que sean de menta pero si no hay, le da igual que sean de cualquier otro tipo. Escriba un algoritmo que represente esta situación.

### *Ir al kiosco*

*si (hay caramelos de menta)*

### *Llevar caramelos de menta*

(1)

sino

### *Llevar de cualquier otro tipo*

(2)

*Pagar 1 peso*

Los aspectos más importantes son:

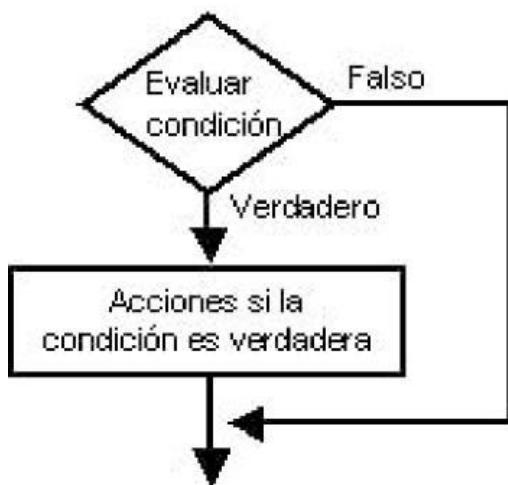


Figura 1.3: Estructura Si-Entonces

- No es posible saber si en el kiosco hay o no hay caramelos de menta ANTES de llegar al kiosco por lo que no puede utilizarse únicamente una secuencia de acciones para resolver este problema.
- La condición “hay caramelos de menta” sólo admite dos respuestas posibles: hay o no hay; es decir, verdadero o falso respectivamente.
- Si se ejecuta la instrucción marcada con (1), NO se ejecutará la acción (2) y viceversa.
- Independientemente del tipo de caramelos que haya comprado, siempre se pagará \$1. Esta acción es independiente del tipo de caramelos que haya llevado.

En algunos casos puede no haber una acción específica a realizar si la condición es falsa. En ese caso se utilizará la siguiente notación:

*si (condición)*

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

Esto se muestra gráficamente en la figura 1.3.

**Ejemplos 1.10:** Su amigo se ha puesto un poco más exigente y ahora le ha pedido que le compre \$1 de caramelos de menta en el kiosco. Si no consigue caramelos de menta, no debe comprar nada.

Escriba un algoritmo que represente esta situación.

*Ir al kiosco*

*si (hay caramelos de menta)*

*Pedir caramelos de menta por valor de \$1*

*Pagar \$1*

Con este último algoritmo, a diferencia del ejemplo 1.8, si la condición “hay caramelos de menta” resulta ser falsa, no se realizará ninguna acción.

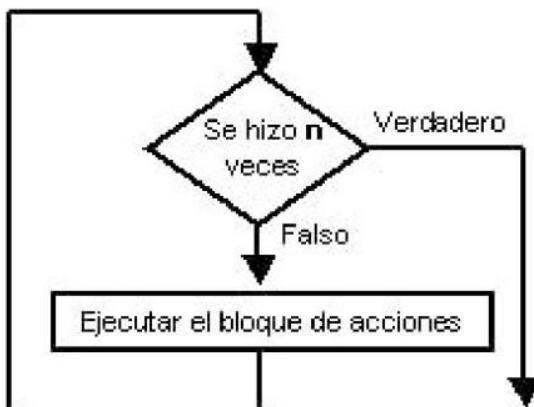


Figura 1.4: Estructura repetitiva

### 1.5.3 Repetición

Un componente esencial de los algoritmos es la repetición. La computadora, a diferencia de los humanos, posee una alta velocidad de procesamiento. A través de ella, es posible ejecutar, de manera repetitiva, algunos pasos elementales de un algoritmo. Esto puede considerarse una extensión natural de la secuencia.

La repetición es la estructura de control que permite al algoritmo ejecutar un conjunto de instrucciones un número de veces fijo y conocido de antemano.

La notación a utilizar es la siguiente y se muestra en la figura 1.4:

**repetir N**  
*Acción o acciones a realizar N veces.*

Se analizan a continuación algunos algoritmos que presentan repeticiones:

**Ejemplo 1.11:** Escriba un algoritmo que permita poner 4 litros de agua en un balde utilizando un vaso de 50 cc.

Al plantear una solución posible, se observa que hay dos pasos básicos: llenar el vaso con agua y vaciarlo en el balde. Para completar los cuatro litros es necesario repetir estas dos operaciones ochenta veces. Suponga que se dispone de un vaso, un balde y una canilla para cargar el vaso con agua.

*Tomar el vaso y el balde*  
**repetir 80**  
*Llenar el vaso de agua.*  
*Vaciar el vaso en el balde.*  
*Dejar el vaso y el balde.*

Nótese que, la instrucción “Dejar el vaso y el balde” no pertenece a la repetición. Esto queda indicado por la sangría o indentación utilizada para cada instrucción. Por lo tanto, se repetirán 80 veces las instrucciones de “Llenar el vaso de agua” y “Vaciar el vaso en el balde”.



Haciendo clic en el siguiente link podés acceder a una animación sobre la estructura Repetición: [Animación Repetición](#)

El ejemplo 1.8, que inicialmente se presentó como un ejemplo de secuencia, puede escribirse utilizando una repetición de la siguiente forma:

**Ejemplo 1.12:** Escriba un algoritmo que permita a un robot subir 8 escalones.

**repetir 4**  
*LevantaPieIzquierdo*  
*SubirUnEscalón.*  
*LevantaPieDerecho*



### *SubirUnEscalón*

Este algoritmo realiza exactamente las mismas acciones que el algoritmo del ejemplo 1.8. Las ventajas de utilizar la repetición en lugar de la secuencia son: la reducción de la longitud del código y la facilidad de lectura.

**Ejemplo 1.13:** Juan y su amigo quieren correr una carrera dando la vuelta a la manzana. Considerando que Juan vive en una esquina, escriba el algoritmo correspondiente.

*repetir 4*

*CorrerUnaCuadra*

*DoblarALaDerecha*

## 1.5.4 Iteración

Existen situaciones en las que se desconoce el número de veces que debe repetirse un conjunto de acciones. Por ejemplo, si se quiere llenar una zanja con arena utilizando una pala, será difícil indicar exactamente cuántas paladas de arena serán necesarias para realizar esta tarea. Sin embargo, se trata claramente de un proceso iterativo que consiste en cargar la pala y vaciarla en la zanja.

Por lo tanto, dentro de una iteración, además de una serie de pasos elementales que se repiten; es necesario contar con un mecanismo que lo detenga.

La iteración es una estructura de control que permite al algoritmo ejecutar en forma repetitiva un conjunto de acciones utilizando una condición para indicar su finalización.

El esquema iterativo es de la forma:

*mientras (condición)*

*Acción o acciones a realizar en caso de que la condición sea verdadera.*

Las acciones contenidas en la iteración serán ejecutadas mientras la condición sea verdadera. Es importante notar que, la primera vez, antes de ejecutar alguna de las acciones de la iteración, lo primero que se realiza es la evaluación de la condición. Sólo luego de comprobar que es verdadera se procede a ejecutar el conjunto de acciones pertenecientes al mientras.

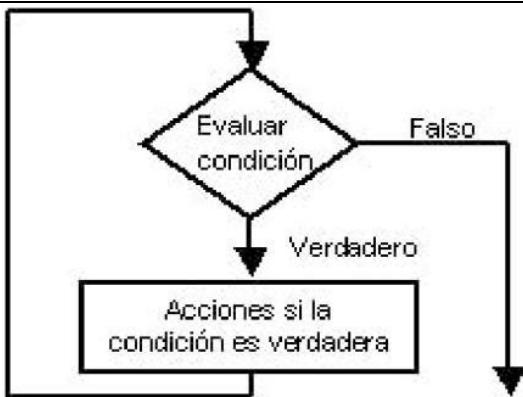


Figura 1.5: Estructura iterativa

Si inicialmente la condición resultara falsa, el contenido del mientras no se ejecutaría ni siquiera una sola vez. Este funcionamiento se muestra gráficamente en la figura 1.5.

Es importante que las acciones realizadas en el interior de la iteración modifiquen el valor de verdad de la condición a finde garantizar que la iteración terminará en algún momento.

Analicemos el siguiente ejemplo:

**Ejemplo 1.14:** Escriba un algoritmo que permita volcar un montículo de arena en una zanja utilizando una pala.

*Tomar la pala.*

*Ubicarse frente a la zanja.*

*mientras (no esté vacío el montículo de arena)*

*cargar la pala con arena*

*volcar la arena en la zanja*

*Dejar la pala.*



Haciendo clic en el siguiente link podés acceder a una animación sobre la estructura Iteración: [Animación Iteración](#)

La iteración indica que, mientras no se vacíe el montículo, se seguirá incorporando arena en la zanja. Cuando el montículo esté vacío, la condición será falsa y la iteración terminará. Es importante destacar, que si el montículo inicialmente estaba vacío, ninguna palada de arena será tomada del montículo ni incorporada a la zanja. Es decir, la condición se verifica ANTES de comenzar la iteración.

En este punto es apropiado hacerse la siguiente pregunta. ¿Qué sentido tiene introducir el concepto de iteración? Con toda seguridad, para los ejemplos antes mencionados no es necesario dicho concepto para establecer clara, simple o comprensiblemente las instrucciones del algoritmo.

Versión: 2 - Capítulo 1 –Resolución de problemas

Existe una razón bastante obvia para justificar esta estructura de control: es una realidad el hecho de que las computadoras requieren instrucciones detalladas y no ambiguas acerca de lo que deben hacer. Se debe, por lo tanto, dividir los algoritmos en pasos simples, de modo que las computadoras puedan efectuar sus cálculos. Si se quiere que algo sea realizado 80 veces, se le debe indicar que lo repita 80 veces. El empleo de las instrucciones de repetición, en este caso, permite hacer esto sin tener que escribir 80 líneas de instrucciones.

Por otro lado, el concepto de iteración es necesario para una mejor legibilidad o facilidad de lectura de los procesos algorítmicos. La iteración es un proceso fundamental en los algoritmos, y se debe ser capaz de pensar en términos de ciclos de iteración para poder construir los algoritmos.

### 1.6 Importancia de la indentación en las estructuras de control

Las instrucciones que pertenecen a una estructura de control deben tener una sangría mayor que la utilizada para escribir el comienzo de la estructura. De esta forma, podrá identificarse donde comienza y termina el conjunto de instrucciones involucradas en dicha estructura. A esta sangría se la denomina indentación.

Este concepto se aplica a las tres estructuras de control vistas previamente: selección, repetición e iteración.

El siguiente ejemplo muestra el uso de la indentación en la selección:

**Ejemplo 1.15:** Suponga que se planea una salida con amigos. La salida depende del clima: si llueve vos y tus amigos irán al cine a ver la película elegida, por el contrario si no llueve irán de pesca. Luego de realizar el paseo se juntarán a comentar la experiencia vivida. Escriba el algoritmo que resuelva esta situación.

*Juntarse en una casa con el grupo de amigos*

*Mirar el estado del tiempo.*

*si (llueve) (1)*

*elegir película*

*ir al cine*

*sino*

*preparar el equipo de pesca*

*ir a la laguna a pescar*

*Volver a la casa a comentar sobre el paseo (2)*

Como puede apreciarse, las acciones que deben ser realizadas cuando la condición es verdadera se encuentran desplazadas un poco más a la derecha que el resto de la estructura. Algo similar ocurre con las acciones a realizar cuando la condición es falsa. De esta forma puede diferenciarse lo que pertenece a la selección del resto de las instrucciones.

En el ejemplo anterior, la instrucción “Volver a la casa a comentar sobre el paseo” se realiza siempre sin importar si llovió o no. Esto se debe a que no pertenece a la selección. Esto queda de manifiesto al darle a las instrucciones (1) y (2) la misma indentación.

**Ejemplo 1.16:** Ud. desea ordenar una caja con 54 fotografías viejas de manera que todas queden al derecho; esto es, en la orientación correcta y la imagen boca arriba. Las fotografías ordenadas se irán guardando en el álbum familiar. Escriba el algoritmo que le permita resolver este problema.

*Tomar la caja de fotos y un álbum vacío.*

**repetir 54**

*Tomar una fotografía.*

*si (la foto está boca abajo)*

*dar vuelta la foto*

*si (la foto no está en la orientación correcta)*

*girar la foto para que quede en la orientación correcta*

*guardar la fotografía en el álbum*

*guardar el álbum*

Según la indentación utilizada, la repetición contiene a la acción de “Tomar una fotografía”, las dos selecciones y la instrucción “guardar la fotografía en el álbum”. Las instrucciones “Tomar la caja de fotos y el álbum” y “Guardar el álbum” no pertenecen a la repetición.

**Ejemplo 1.17:** Ud. se dispone a tomar una taza de café con leche pero previamente debe endulzarlo utilizando azúcar en sobrecitos. Escriba un algoritmo que resuelva este problema.

*Tomar la taza de café con leche.*

*Probar el café con leche*

**mientras** (no esté lo suficientemente dulce el café)

*Tomar un sobre de azúcar.*

*Vaciar el contenido del sobre en la taza.*

*Mezclar para que el azúcar se disuelva.*

*Probar el café con leche*

*Tomar el café con leche.*

Note que en este último ejemplo no se conoce de antemano la cantidad de sobrecitos de azúcar necesarios para endulzar el contenido de la taza. Además, la condición se evalúa antes de agregar el primer sobre. Según la indentación utilizada, la iteración incluye cuatro instrucciones. La acción “Tomar el café con leche” se ejecutará sólo cuando la iteración haya terminado, es decir, cuando la condición sea falsa.

## 1.7 Conclusiones

El uso de algoritmos permite expresar, de una forma clara, la manera en que un problema debe ser resuelto. Los elementos que lo componen son característicos de la resolución de problemas con computadora.

La ejercitación es la única herramienta para poder comprender y descubrir la verdadera potencialidad de las estructuras de control. Resulta fundamental alcanzar un total entendimiento del funcionamiento de estas estructuras para poder lograr expresar soluciones más complejas que los ejemplos aquí planteados.



## Ejercitación

1. Esta noche Juan se encuentra haciendo zapping sabiendo que hay un canal de televisión que está transmitiendo la película "30 años de felicidad". Luego de terminar de ver la película debe apagar el televisor.

Analice las siguientes soluciones:

### Solución 1:

Encender el televisor.  
Cambiar de canal hasta encontrar la película.  
Ver la película.  
Apagar el televisor.

### Solución 2:

Encender el televisor.  
**si** (está transmitiendo "30 años de felicidad")  
    ver la película.  
Apagar el televisor.

### Solución 3:

Encender el televisor.  
**repetir** 20  
    cambiar de canal.  
Ver la película "30 años de felicidad".  
Apagar el televisor.

### Solución 4:

Encender el televisor.  
**mientras** (no se transmite en el canal actual "30 años de felicidad")  
    cambiar de canal.  
Ver la película.  
Apagar el televisor.

- (a) Compare las soluciones 1 y 4.  
(b) Explique por qué las soluciones 2 y 3 son incorrectas.  
(c) ¿Qué ocurriría con la solución 4 si ningún canal estuviera transmitiendo la película?
2. Ud. desea comprar la revista "Crucigramas" que cada mes tiene reservada en el puesto de revistas que se encuentra en la esquina de su casa, al otro lado de la calle. Verifique que no pasen autos antes de cruzar. Indique, para cada uno de los siguientes algoritmos, si representa la solución a este problema. Justifique su respuesta.



**Algoritmo 1:**

Caminar hasta la esquina.  
**mientras** (no pasen autos)  
    Cruzar la calle  
    Comprar la revista "Crucigramas".

**Algoritmo 2:**

**mientras** (no llegue a la esquina)  
    dar un paso  
**mientras** (pasen autos)  
    esperar 1 segundo  
    Cruzar la calle.  
Llegar al puesto de revistas.  
Comprar la revista "Crucigramas".

**Algoritmo 3:**

**mientras** (no llegue a la esquina)  
    dar un paso.  
**mientras** (pasen autos)  
    esperar 1 segundo  
**mientras** (no llegue a la otra vereda)  
    dar un paso.  
Llegar al puesto de revistas.  
Comprar la revista "Crucigramas".

**Algoritmo 4:**

**repetir** 10  
    dar un paso.  
    Cruzar la calle.  
    Llegar al puesto de revistas.  
    Comprar la revista "Crucigramas".

3. Utilizando las estructuras de control vistas resolver:
  - a) Un algoritmo que, en caso de ser necesario, permita cambiar el filtro de papel de una cafetera. Considere que está frente a la cafetera y que dispone de un filtro suplemento.
  - b) Modifique la solución anterior para que cuando encuentre que el filtro de la cafetera está limpio, guarde el filtro suplemento en el lugar correspondiente.
4. Escriba un algoritmo que le permita trasladar 70 cajas de 30 kilos cada una, desde la Sala A hasta la Sala B. Considere que sólo llevará una caja a la vez porque el contenido es muy frágil. Para realizar el trabajo debe ponerse un traje especial y quitárselo luego de haber realizado el trabajo.
5. Modifique el algoritmo 4 suponiendo que puede trasladar 60 kilos a la vez.
6. Escriba un algoritmo que le permita guardar fotos en un álbum familiar. El álbum está compuesto por 150 páginas y se encuentra vacío. En cada página entran 10 fotos. El álbum se completa por páginas. Una vez que el álbum está completo, debe guardarse en la biblioteca. Considere que cuenta con fotos suficientes para completar el álbum. Para mayor simplicidad, las páginas se completan de un solo lado.



Versión: 2 - Capítulo 1 –Resolución de problemas

7. Modifique el algoritmo anterior si ahora no se conoce la cantidad de fotos que entran en una página. Se cuentan con fotos suficientes para completar el álbum.
8. Modifique el algoritmo del ejercicio 6) pero suponiendo ahora que no se sabe la cantidad de páginas que tiene el álbum. Se sabe que en cada página entran 10 fotos. Se cuentan con fotos suficientes para completar el álbum.
9. Modifique el algoritmo del ejercicio 6) pero suponiendo ahora que no se sabe la cantidad de páginas que tiene el álbum ni la cantidad de fotos que entran en cada página. Se cuentan con fotos suficientes para completar el álbum.

## 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.

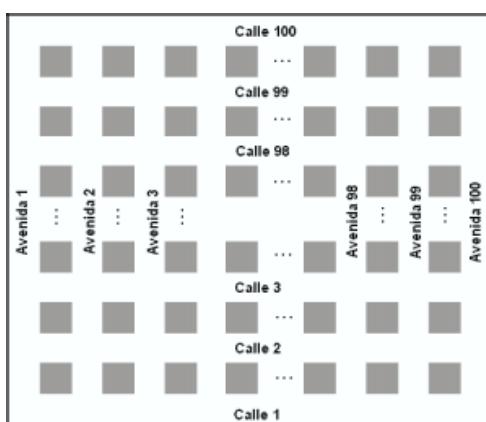


Figura 2.1: La ciudad del robot

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.

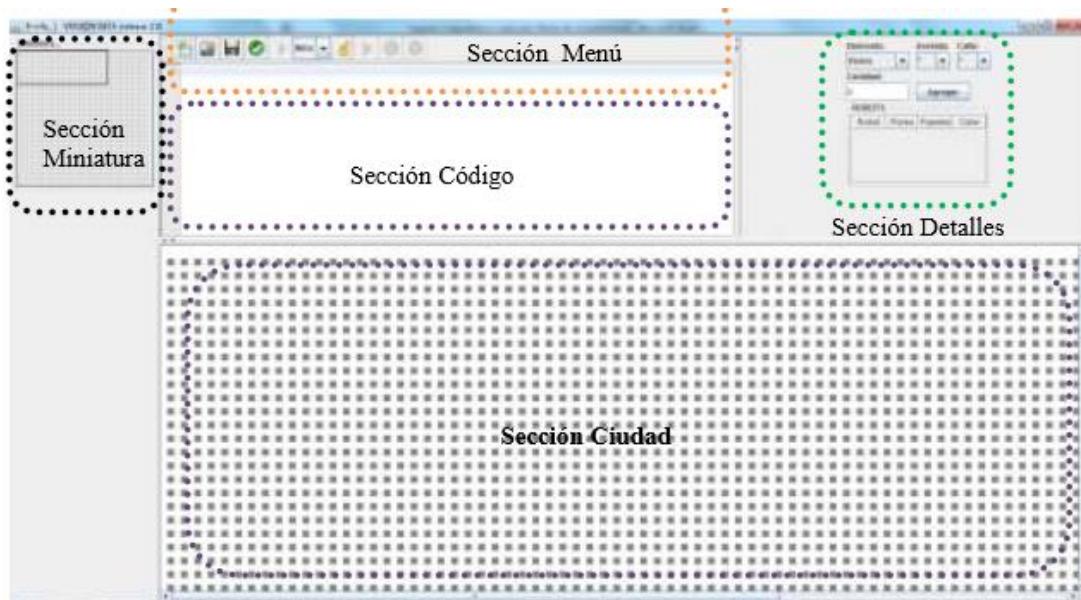


Figura 2.2: Ambiente de programación de R-info

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 *.lmp*. 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

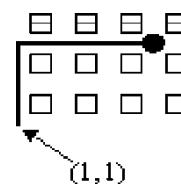
En esta sección se detallará la sintaxis correspondiente a las estructuras de control utilizadas por el robot R-info. El funcionamiento de cada una de esas estructuras se explicó en forma detallada en el capítulo 1.

### 2.3.1 Secuencia

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

**Ejemplo 2.1:** Programe 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.

Sintáxis 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.  
 $p = \text{“Juan es estudiante”}$   
 $q = \text{“Juan es jugador de fútbol”}$   
 Simbolizando  $p \ \& \ q$
- No es cierto que PosCa = 100.  
 $p = \text{“PosCa=100”}$   
 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.



<b>p</b>	<b>q</b>	<b>p &amp; q</b>
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.

$p = \text{"6 es un numero par"}$

$q = \text{"6 es divisible por 3"}$

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

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

$p = \text{"PosCa} = 1\text{"}$

$q = \text{"PosAv} = 2\text{"}$

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

<b>p</b>	<b>q</b>	<b><math>p \mid q</math></b>
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  $p \mid q$  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  $p \mid q$  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) mid (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 (45,70) 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.



## Ejercitación

1. Escriba un programa que le permita al robot recoger una flor de la esquina (2,84) si existe.
2. Escriba un programa que le permita al robot recorrer la calle 50 desde la avenida 65 hasta la avenida 23 depositando un papel en cada esquina. Debe avanzar hasta el final aunque durante el recorrido se quede sin papeles.
3. Escriba un programa que le permita al robot recorrer el perímetro del cuadrado determinado por (1,1) y (2,2).
4. Modificar el ejercicio 3. para que además recoja, de ser posible, un papel en cada esquina.
5. Escriba un programa que le permita al robot dejar todas las flores que lleva en su bolsa en la esquina (50,50).
6. Escriba un programa que le permita al robot recorrer la avenida 75 desde la calle 45 hasta la calle 15 recogiendo todas las flores que encuentre.
7. Escriba un programa que le permita al robot recorrer la avenida 10, depositando una flor en cada esquina. Si en algún momento del recorrido se queda sin flores en la bolsa, debe seguir caminando (sin depositar) hasta terminar la avenida.
8. Escriba un programa que le permita al robot recorrer la avenida 23 buscando una esquina sin papeles que seguro existe. Al encontrarla debe depositar, en esa esquina, todos los papeles que lleva en su bolsa. Informar en qué calle dejó los papeles.
9. Escriba un programa que le permita al robot recorrer la calle 17 depositando un papel en las avenidas impares. El recorrido termina cuando el robot llega a la esquina (100,17).
10. Programe al robot para que recorra las 5 primeras avenidas juntando en cada esquina todas las flores y papeles.
11. Programe al robot para que recorra el perímetro de la ciudad recogiendo todas las flores y papeles que encuentre y dejando en cada vértice solo un papel. Puede ocurrir que algún vértice quede vacío si el robot no tiene papeles en su bolsa para depositar.
12. Programe al robot para que recorra todas las calles depositando en cada esquina vacía un papel. En caso de no tener más papeles debe continuar el recorrido (sin depositar).

# 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 seguro 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, necesitaremos 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.

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
```

utilizada mientras se ejecuta el código del robot.

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 sobrepasara el límite superior permitido para los elementos del tipo numero, se dice que ocurre un overflow 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	$\neq$	$A \neq B$

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

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
```

Versión: 2 - Capítulo 3 –Datos

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 `cantidadCuadras` representará (o contendrá) el valor 0.

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

Por ejemplo, la 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 `cantidadCuadras` y 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

```

areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cantidadCuadras: numero
comenzar
    {Hasta ahora no se camino ninguna cuadra }
    cantidadCuadras:=0
    mientras (hayFlorEnLaEsquina)                                {1}
        {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 { 1 }

**mientras** ~ (HayFlorEnLaEsquina) | ~ (HayPapelEnLaEsquina) { 2 }

{anotar que se caminó una cuadra mas}

cuadras:=cuadras+1 { 3 }

move

Informar (cuadras) { 4 }

**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 modiflico 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 {registrarmos que se tomó una flor}
        flores:= flores+1
    Informar(flores)
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. 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.

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                                {1}
    mientras (HayFlorEnLaEsquina)
        tomarFlor {registramos que se tomó una flor}
        flores:= flores+1                      {2}
        {depositamos las flores juntadas de la esquina}
        repetir flores                         {3}
            depositarFlor
            Informar (flores)
    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 “mientrasHayFlorEnLaBolsadepositarFlor”?

**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
        {registramos 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.



# Ejercitación

1. Indique que hacen los siguientes programas considerando las diferentes situaciones que podrían presentarse:

- (a) i. todas las esquinas de la avenida 6 tienen al menos 1 flor
- ii. sólo la esquina (6,20) tiene flor.
- iii. ninguna esquina de la avenida 6 tiene flor

```
programa queHace1
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos (6,1)
    mientras ((HayFlorEnLaEsquina) & (PosCa< 100))
        mover
        tomarFlor
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
Fin
```

- (b) i. todas las esquinas de la avenida tienen al menos 1 flor y 1 papel.
- ii. sólo la esquina (6,20) tiene flor y ningún papel, las demás están vacías.
- iii. sólo la esquina (6,20) tiene papel y no tiene ninguna flor, las demás están vacías.
- iv. ninguna esquina de la avenida 1 tiene flor ni papel

```
programa queHace2
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    nro : numero
comenzar
    nro := 0
    repetir 10
        si~((HayFlorEnLaEsquina) | (HayPapelEnLaEsquina))
            mover
            nro := nro + 1
        Informar (nro)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
Fin
```

Versión: 2 - Capítulo 3 –Datos

- 
2. Programe al robot para que informe la cantidad de flores que hay en la calle 44.
    - I. Recogiendo todas las flores.
    - II. Sin modificar el contenido de cada esquina.
  3. Programe al robot para que informe la cantidad de esquinas vacías que hay en la ciudad.
  4. Escriba un programa que le permita al robot caminar por la calle 7 hasta encontrar 20 flores. Hay como máximo una flor por esquina. Seguro existen 20 flores.
  5. Escriba un programa que le permita al robot caminar por la calle 7 hasta encontrar 20 flores. Hay como máximo una flor por esquina. Pueden no haber 20 flores.
  6. Escriba un programa que le permita al robot caminar por la calle 7 hasta encontrar 20 flores. Puede haber más de una flor por esquina. Seguro existen 20 flores.
  7. El robot debe limpiar de papeles la calle 34. Al terminar el recorrido debe informar cuantas esquinas tenían originalmente exactamente 6 papeles.
  8. Programe al robot para que recorra la calle 2 hasta encontrar al menos 10 papeles. Pueden no haber 10 papeles.
  9. Programe al robot para que recorra la calle 2 hasta encontrar 10 papeles y 4 flores. Seguro existen dichas cantidades.
  10. Programe al robot para que recorra el perímetro de la ciudad e informe la cantidad de papeles recogidos en cada lado.

# Capítulo 4

## Repasso



### Objetivos

Hasta ahora se ha definido la manera de escribir programas utilizando el lenguaje del ambiente del robot R-info. También se ha presentado la sintaxis utilizada que permite trasladar al robot, recoger y/o depositar flores y papeles y saber si hay o no flores en la esquina o en la bolsa.

Por otro lado se han analizado diferentes situaciones que requieren la posibilidad de representar información específica del problema.

El objetivo de este capítulo es presentar, analizar y resolver diferentes ejemplos que permitirán la ejercitación de los temas vistos en los capítulos anteriores.



### Temas a tratar

- ✓ Presentación, análisis y resolución de ejemplos.
- ✓ Conclusiones.
- ✓ Ejercitación.

## 4.1 Repaso de variables

En los capítulos anteriores se ha definido la sintaxis de las acciones u órdenes que el robot puede llevar a cabo y se ha indicado como se representará y trabajará con la información relevante que presenta el problema a resolver utilizando el lenguaje del ambiente del robot R-info.

Como ya hemos visto:

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

Además, sabemos que dentro de un mismo programa pueden utilizarse tantas variables como sean necesarias para representar adecuadamente todos los datos presentes en el problema.

Sin embargo, de todos los ejemplos vistos en los capítulos 2 y 3 podríamos pensar que cada vez que un enunciado requiere informar una cantidad, es necesario recurrir a una variable. A través de un ejemplo, podemos observar que esto no siempre es así.

Analicemos el siguiente ejemplo:

**Ejemplo 4.1:** Programe al robot para que recorra la calle 45 deteniéndose cuando encuentre una esquina que no tiene flores, sabiendo que esa esquina seguro existe. Al terminar debe informar la cantidad de pasos dados.

Este problema admite dos soluciones. Una de ellas utiliza una variable para representar la cantidad de pasos que da el robot y la otra no.

<pre> <b>programa</b> cap4Ejemplo1a <b>areas</b>     ciudad: AreaC(1,1,100,100) <b>robots</b>     robot robot1 <b>comenzar</b>     <b>mientras</b> (HayFlorEnLaEsquina)         mover         Informar (PosAv-1)     <b>fin</b> <b>variables</b>     R-info: robot1 <b>comenzar</b>     AsignarArea(R-info,ciudad)     Iniciar(R-info,1,1) <b>fin</b> </pre>	<pre> <b>programa</b> cap4Ejemplo1b <b>areas</b>     ciudad: AreaC(1,1,100,100) <b>robots</b>     robot robot1 <b>variables</b>     pasos: numero <b>comenzar</b>     pasos:= 0 {1}     <b>mientras</b> (HayFlorEnLaEsquina)         mover         pasos:= pasos + 1         Informar (pasos) {2}     <b>fin</b> <b>variables</b>     R-info: robot1 <b>comenzar</b>     AsignarArea(R-info,ciudad)     Iniciar(R-info,1,1) <b>fin</b> </pre>
--	---

El ejemplo 4.1 demuestra que antes de decidir representar nueva información dentro del programa, es importante analizar si el robot no cuenta con la posibilidad de manejar los datos pedidos y de este modo evitar la declaración de un dato.



Analicemos:



Por qué en el programa Cap4Ejemplo1a (el que no usa la variable) se informa (PosAv-1)?

¿Qué ocurriría en Cap4Ejemplo1b si la línea (1) es reemplazada por pasos:=1 y la línea (2) es reemplazada por Informar ( pasos –1 ) ?

¿Cuál de las formas de resolver el problema le parece más adecuada? Justificar la respuesta.

## 4.2 Repaso de expresiones lógicas

Recordemos que las expresiones lógicas pueden formarse con variables y expresiones relacionales utilizando los operadores lógicos de la tabla 2.3.

Analicemos los siguientes ejemplos para ejercitarse la resolución de expresiones lógicas que combinan varias proposiciones:

**Ejemplo 4.2:** Programe al robot para que informe si en la esquina (7,4) hay solo flor o solo papel (pero no ambos).

```
programa cap4Ejemplo2
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(7,4)
    si (HayFlorEnLaEsquina & ~ HayPapelEnLaEsquina) | {1}
    (~ HayFlorEnLaEsquina & HayPapelEnLaEsquina) {2}
        Informar(V)
    sino
        Informar(F)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Como puede verse en este ejemplo, que en la selección se ha utilizado una disyunción de conjunciones. Es decir que basta con que una de las dos conjunciones sea verdadera para que toda la proposición lo sea.

Cada una de las conjunciones requiere que haya uno solo de los dos elementos: la primera pide que haya flor y no papel (1) y la segunda que haya papel y no flor (2). Obviamente, no pueden ser verdaderas al mismo tiempo. Pero basta con que solo una de ellas lo sea para que se informe V.

**Ejemplo 4.3:** Programe al robot para que recorra la avenida 16 buscando una flor que puede no existir. Al finalizar informar donde está (si la encontró) o F (falso) en caso contrario.

```

programa cap4Ejemplo3
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    Pos(16,1)
    {recorre la Av.16 buscando la flor}
    mientras ~ HayFlorEnLaEsquina & (PosCa < 100) {1}
        mover
        {ver si encontró la flor o no}
        si HayFlorEnLaEsquina
            Informar ( PosCa ) {2}
        sino
            Informar( F )
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

Como podemos observar, en la línea (1) se utiliza una proposición molecular para controlar la iteración. Ahora no alcanza con verificar solamente que la flor no exista ( $\sim \text{HayFlorEnLaEsquina}$ ) sino que además es necesario tener en cuenta que no se termine la avenida ( $\text{PosCa} < 100$ ).

Dado que ambas condiciones deben cumplirse simultáneamente, se las ha unido por una conjunción. Esta proposición molecular será verdadera cuando ambas proposiciones lo sean. La iteración puede leerse como: “mientras no encuentre la flor y a la vez, el robot no llegue a la calle 100, debe seguir avanzando”.

La iteración termina cuando la conjunción es falsa. Esto ocurre por tres motivos:

1. Encontró la flor durante el recorrido de la avenida. Es decir que la condición ( $\text{PosCa} < 100$ ) es verdadera pero la proposición ( $\sim \text{HayFlorEnLaEsquina}$ ) es falsa.
2. No encontró la flor pero llegó a la calle 100. Es decir que ( $\sim \text{HayFlorEnLaEsquina}$ ) es verdadera y ( $\text{PosCa} < 100$ ) es falsa.
3. Encontró la flor sobre la calle 100. En este caso ambas condiciones son falsas.

Por lo tanto, la iteración no necesariamente termina cuando la flor ha sido hallada y para poder informar lo solicitado en el enunciado del problema, será necesario distinguir lo que pasó. Esa es la función de la selección que aparece en la línea (2).

Analicemos:



Si se logra el mismo resultado reemplazando la condición que aparece en la línea (2) por PosCa=100. Justificar la respuesta.

Pensar en otra proposición que permita dar a la selección de (2) el mismo funcionamiento.

## 4.3 Ejemplos

Habiendo repasado los aspectos más importantes para la ejercitación propuesta para este capítulo, a continuación se presentan diferentes ejemplos que combinan los temas vistos hasta aquí. Es recomendable que prestemos especial atención a la definición y evaluación de proposiciones.

**Ejemplo 4.4:** Programe al robot para que recorra la calle 29 hasta encontrar una esquina vacía que puede no existir. En caso de encontrarla depositar en ella una flor. Si no pudo depositar (porque no tenía) informar F (falso).

El siguiente programa resuelve este problema:

```
programa cap4Ejemplo4
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    {ubicar el robot al comienzo de la calle 29}
    Pos(1,29)
    derecha
    {recorrer la calle hasta encontrar una esquina vacía o hasta terminar}
    mientras (HayFlorEnLaEsquina | HayPapelEnLaEsquina) & (PosAv < 100)
        mover
        {si la encontró depositar en ella una flor}
        si ~HayFlorEnLaEsquina & ~HayPapelEnLaEsquina
            si HayFlorEnLaBolsa
                depositarFlor
            sino
                Informar(F)
        fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Analicemos:



¿Cuales son los casos en los que la evaluación de la proposición molecular que maneja la iteración da como resultado falso?

¿Puede reemplazarse la selección anterior por la siguiente?:

```

si ~HayFlorEnLaEsquina & ~HayPapelEnLaEsquina & HayFlorEnLaBolsa
    depositarFlor
sino
    Informar (F)

```

**Ejemplo 4.5:** Programe al robot para que informe la cantidad de papeles que hay en la esquina (67,23) SIN modificar el contenido de la esquina.

Este problema es una variante del ejemplo 3.5, donde no se pide que se recojan los papeles sino sólo que informe la cantidad. Sabemos que para poder resolver esto será necesario juntar los papeles contando y luego depositar exactamente la cantidad de papeles recogidos. Notemos que no es lo mismo vaciar los papeles de la bolsa porque ella podría contener papeles ANTES de comenzar a recoger. El programa es el siguiente:

```

programa cap4Ejemplo5
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cantP: numero
comenzar
    Pos(67,23)
    {Indicar que aun no se ha recogido nada}
    cantP := 0
    mientras HayPapelEnLaEsquina
        tomarPapel
        cantP := cantP + 1
        {Ahora la esquina ya no tiene papeles}
        Informar (cantP)
        {Volver a dejar los papeles en la esquina}
        repetir cantP
            depositaPapel
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

```

Se ha utilizado una repetición para volver a poner los papeles en la esquina porque, luego de haberlos recogido, se conoce exactamente la cantidad de papeles que se quiere depositar. Además, para depositar no es necesario preguntar si hay papeles en la bolsa para satisfacer esta demanda porque se ha recogido la misma cantidad de papeles a través de la iteración. Es más, suponiendo que originalmente no hubiera habido papeles en (67,23), *cantP* valdrá cero en cuyo caso el repetir no ejecutará ninguna instrucción.



- Proponé otra forma de escribir el segmento de código que vuelve a poner los papeles en la esquina (último repetir).

**Ejemplo 4.6:** Programe al robot para que informe la cantidad de flores que hay en cada una de las esquinas de la avenida 1.

Para resolver este problema alcanzará con una única variable que represente la cantidad de flores de la esquina actual. Cada vez que llega a una esquina, el robot inicializará la variable en cero, anotará en ella cada vez que logre recoger una flor y finalmente informará su valor. Esto se debe repetir para cada esquina de la avenida 1. El programa será el siguiente:

```

programa cap4Ejemplo6
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    flores: numero
comenzar
    {Se recorrerán las primeras 99 esquinas}
    repetir 99
        {Indicar que aun no se ha recogido nada en esta esquina}
        flores := 0
        mientras HayFlorEnLaEsquina
            tomarFlor
            flores := flores + 1
            {Ahora la esquina ya no tiene flores}
            Informar(flores)
            {Pasar a la esquina siguiente}
            mover
            {Falta la esquina (1,100)}
            {Indicar que aun no se ha recogido nada en esta esquina}
            flores := 0
            mientras HayFlorEnLaEsquina
                tomarFlor
                flores := flores + 1
                {Ahora la esquina ya no tiene flores}
                Informar(flores)
            fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

Analicemos:

-  ¿Qué ocurriría si la línea (1) fuera trasladada *antes* del repetir, es decir antes de comenzar la repetición? ¿Qué valores informaría?
- ¿Por qué es necesario procesar por separado la esquina (1,100)? Vea que aparece fuera de la repetición en la línea (2).
- ¿Cómo modificaría el programa anterior para que el robot también pueda informar para cada esquina, el número de calle y la cantidad de flores que contiene?

## 4.4 Conclusiones

Se han presentado varios ejemplos que muestran el uso de los dos tipos de datos que puede manejar el robot: valores numéricos y valores booleanos. A través de ellos se ha mostrado la forma de mejorar la potencia de las soluciones ofrecidas, permitiendo que el robot registre valores para un procesamiento posterior.

También se ha definido y ejemplificado el uso de los conectivos lógicos permitiendo manejar las estructuras de control selección e iteración a través de proposiciones moleculares.



## Ejercitación

1. Indique qué hacen los siguientes programas considerando las diferentes situaciones que podrían presentarse:

- a) **programa** queHace1
- areas**
- ciudad: AreaC(1,1,100,100)
- robots**
- robot robot1
- comenzar**
- Pos (4,3)  
         si ((HayFlorEnLaEsquina) & ~ (HayPapelEnLaEsquina))  
             tomarFlor  
             Informar(V)  
         sino  
             Informar(F)
- fin**
- variables**
- R-info: robot1
- comenzar**
- AsignarArea(R-info, ciudad)  
         Iniciar(R-info, 1, 1)
- fin**
- b) **programa** queHace2
- areas**
- ciudad: AreaC(1,1,100,100)
- robots**
- robot robot1
- comenzar**
- Pos (6,1)  
         mientras ( HayFlorEnLaEsquina & (PosCa < 100))  
             mover  
             tomarFlor
- fin**
- variables**
- R-info: robot1
- comenzar**
- AsignarArea(R-info, ciudad)  
         Iniciar(R-info, 1, 1)
- fin**
- c) **programa** queHace3
- areas**
- ciudad: AreaC(1,1,100,100)
- robots**
- robot robot1
- comenzar**
- repetir 99  
             mientras (HayFlorEnLaEsquina)  
                 tomarFlor  
                 mover  
             mientras (HayFlorEnLaEsquina)  
                 tomarFlor
- fin**
- variables**
- R-info: robot1
- comenzar**

```

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

d) programa queHace4
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    nro : numero
comenzar
    nro := 0
    si ~( HayFlorEnLaEsquina | HayPapelEnLaEsquina )
        mover
        nro := nro + 1
        Informar (nro)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

e) programa queHace5
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    SinFlor : numero
comenzar
    SinFlor := 0
    Pos (1,20)
    derecha
    mientras ( HayFlorEnLaEsquina & (PosAv < 100))
        tomarFlor
        si ~(HayFlorEnLaEsquina)
            SinFlor := SinFlor + 1
        mover
        Informar (SinFlor)
    fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin

f) programa queHace6
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
variables
    cant : numero
comenzar
    cant:= 0
    mientras ( HayFlorEnLaEsquina & HayPapelEnLaEsquina )

```

---

```

        tomarFlor
        tomarPapel
        cant:= cant +1
fin
variables
    R-info: robot1
comenzar
    AsignarArea(R-info,ciudad)
    Iniciar(R-info,1,1)
fin
```

2. Programar al robot para que recorra la calle 3 desde la avenida 5 hasta la avenida 20 depositando un papel en cada esquina. Si durante el recorrido se queda sin papeles para depositar, debe detenerse.
3. Suponiendo que el robot cuenta con suficiente cantidad de flores y papeles en su bolsa, escribir un programa que le permita recorrer la calle 45 dejando en las avenidas pares solo una flor y en las impares solo un papel.
4. Programar al robot para que recorra la calle 20 e informe cuántas esquinas tienen sólo flores y cuántas tienen sólo papeles.
5. Programar al robot para que recorra el perímetro de la ciudad dejando un papel en aquellas esquinas que sólo tienen papel y una flor en las esquinas que tienen sólo flores. El recorrido debe finalizar al terminar de recorrer el perímetro.
6. Programar al robot para que recorra el perímetro de la ciudad buscando una esquina con exactamente 3 flores y 3 papeles, suponiendo que esta esquina existe. Debe informar cual es la esquina encontrada.
7. Idem 6. pero no se puede asegurar que tal esquina existe. En caso de encontrarla, informar cual es esa esquina.
8. Indique si son verdaderas o falsas las siguientes afirmaciones de acuerdo al programa ‘Que Hace7’. JUSTIFIQUE cada respuesta.

```

programa queHace7
areas
    ciudad: AreaC(1,1,100,100)
robots
    robot robot1
comenzar
    repetir 5
        mover
        derecha
        mientras ( (HayFlorEnLaEsquina | HayPapelEnLaEsquina) & (PosAv<100) )
            mover
            mientras (HayFlorEnLaEsquina)
                tomarFlor
            mientras (HayPapelEnLaEsquina)
                tomarPapel
    fin
variables
    R-info: robot1
comenzar
```

---

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

- a) Se puede asegurar que el robot pasará por la esquina (10, 6)
- b) Se puede asegurar que el robot pasará por la esquina (1, 6)
- c) El robot se puede caer de la ciudad.
- d) En todas las esquinas por las que pasó el robot hay flores o papeles.
- e) Al detenerse, se puede asegurar que el robot levantará flores y papeles.
- f) Al finalizar el recorrido, el robot tiene flores y papeles en la bolsa.

## 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.

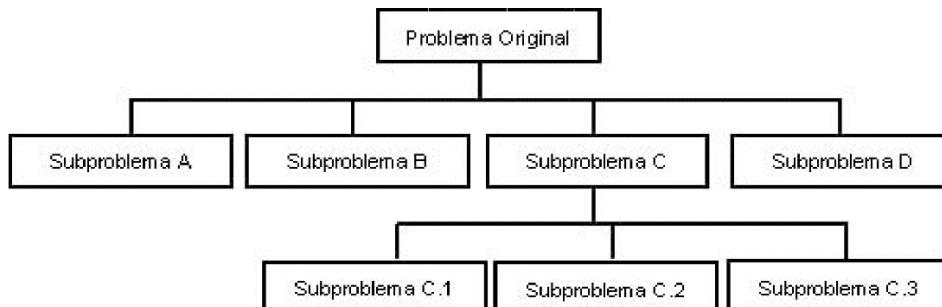


Figura 5.1: Diseño Top-Down

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 hacedor 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> <b>programa</b> Cap5Ejemplo5.1 <b>areas</b>   ciudad: AreaC(1,1,100,100) <b>robots</b>   <b>robot</b> robot1   comenzar     mientras HayFlorEnLaEsquina       tomarFlor     mover     mientras HayFlorEnLaEsquina       tomarFlor     fin <b>variables</b>   R-info : robot1 <b>Comenzar</b>   AsignarArea (R-info, ciudad)   Iniciar (robot1 , 1 , 1) <b>fin</b> </pre>	<pre> <b>programa</b> Cap5Ejemplo5.1.2 <b>procesos</b>   <b>proceso</b> JuntarFlores { 3 }   comenzar     mientras HayFlorEnLaEsquina       tomarFlor     fin <b>areas</b>   ciudad: AreaC(1,1,100,100) <b>robots</b>   robot robot1   comenzar     JuntarFlores { 2 }     mover { 4 }     JuntarFlores { 5 }     fin { 6 } <b>variables</b>   R-info : robot1 <b>comenzar</b>   AsignarArea (R-info, ciudad)   Iniciar (robot1 , 1 , 1) <b>fin</b> </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> <b>programa</b> Cap5Ejemplo5.3.Solucion1 <b>procesos</b>     <b>proceso</b> JuntarFlores     <b>comenzar</b>         mientras HayFlorEnLaEsquina             tomarFlor         <b>fin</b>     <b>proceso</b> JuntarPapeles     <b>comenzar</b>         mientras HayPapelEnLaEsquina             tomarPapel         <b>fin</b> <b>areas</b>     ciudad: AreaC(1,1,100,100) <b>robots</b>     <b>robot</b> robot1     <b>comenzar</b>         repetir 10             repetir 99                 JuntarFlores                 JuntarPapeles                 mover                 {Falta última esquina}                 JuntarFlores                 JuntarPapeles                 Pos(PosAv + 1 ,1)             <b>fin</b> <b>variables</b>     R-info : robot1 <b>comenzar</b>     AsignarArea (R-info,ciudad)     Iniciar (R-info, 1 ,1) <b>fin</b>   </pre>	<pre> <b>programa</b> Cap5Ejemplo5.3.Solucion2 <b>procesos</b>     <b>proceso</b> JuntarFlores     <b>comenzar</b>         mientras HayFlorEnLaEsquina             tomarFlor         <b>fin</b>     <b>proceso</b> JuntarPapeles     <b>comenzar</b>         mientras HayPapelEnLaEsquina             tomarPapel         <b>fin</b>     <b>proceso</b> Avenida     <b>comenzar</b>         repetir 99             JuntarFlores             JuntarPapeles             mover             JuntarFlores             JuntarPapeles         <b>fin</b> <b>areas</b>     ciudad: Areac(1,1,100,100) <b>robots</b>     <b>robot</b> robot1     <b>comenzar</b>         repetir 10             Avenida             Pos(PosAv + 1 ,1)         <b>fin</b> <b>variables</b>     R-info : robot1 <b>comenzar</b>   </pre>

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

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?

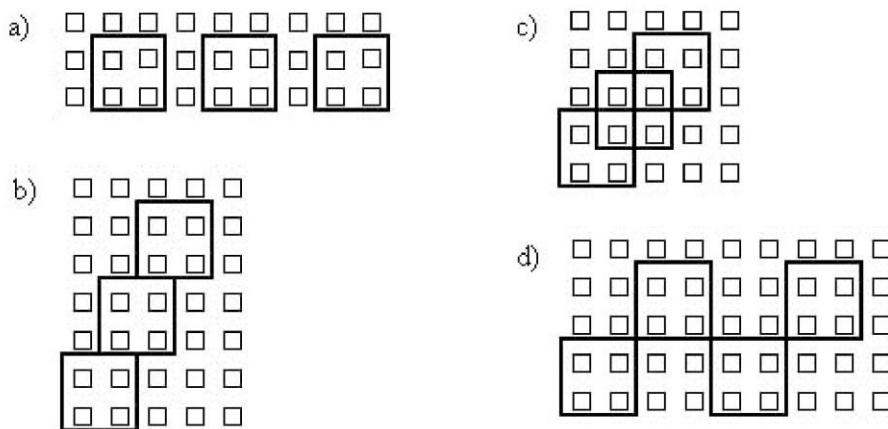


Figura 5.6: Recorridos

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

<pre><b>robots</b> <b>robot</b> robot1 <b>comenzar</b>     <b>repetir</b> 3         cuadrado         Pos(PosAv+1, PosCa+2)     <b>fin</b></pre>	<pre><b>robots</b> <b>robot</b> robot1 <b>comenzar</b>     <b>repetir</b> 3         <b>repetir</b> 4             mover             mover             derecha             Pos(PosAv+1, PosCa+2)     <b>fin</b></pre>
---	---

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.

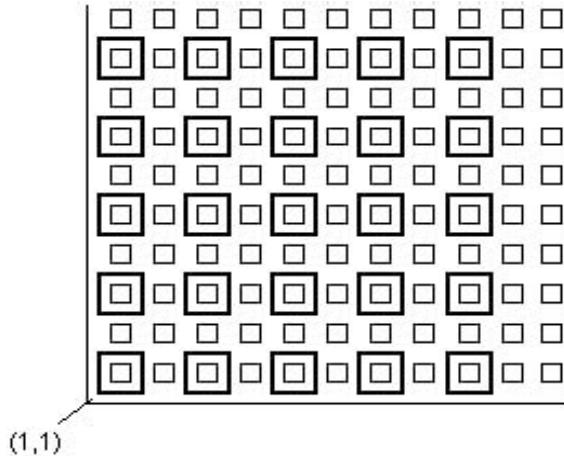


Figura 5.7: Recorrido del Ejemplo 5.7

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.

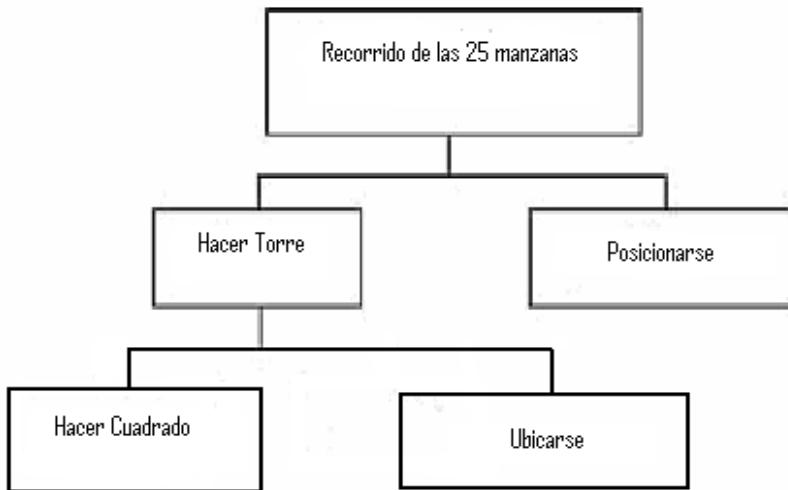


Figura 5.7: Descomposición Top-Down del Ejemplo 5.5

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:

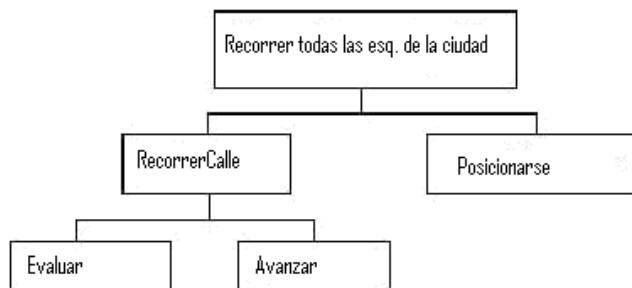


Figura 5.8: Descomposición Top-Down del Ejemplo 5.9

```

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 ejemplificado 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

1. Escriba un proceso que le permita al robot realizar un cuadrado de lado 2 girando en la dirección de las agujas del reloj.
2. Utilice el proceso desarrollado en 1. para realizar un programa para cada uno de los recorridos de la figura 5.9.

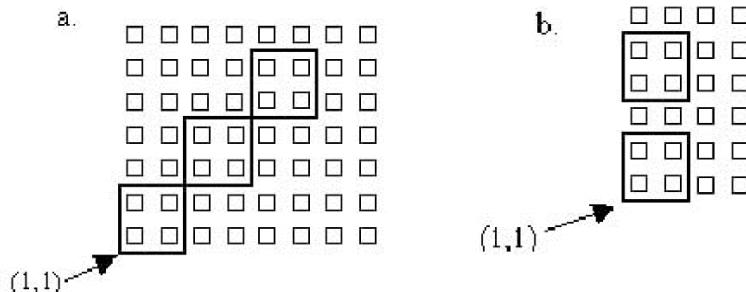


Figura 5.9: Recorridos usando cuadrados de lado 2

3. Escriba un proceso que le permita al robot realizar un rectángulo de base 5 y altura 3 girando en la dirección de las agujas del reloj a partir de la posición (1,1).
4. Programe al Robot para que realice los recorridos de la figura 5.10 utilizando el proceso desarrollado en 3.

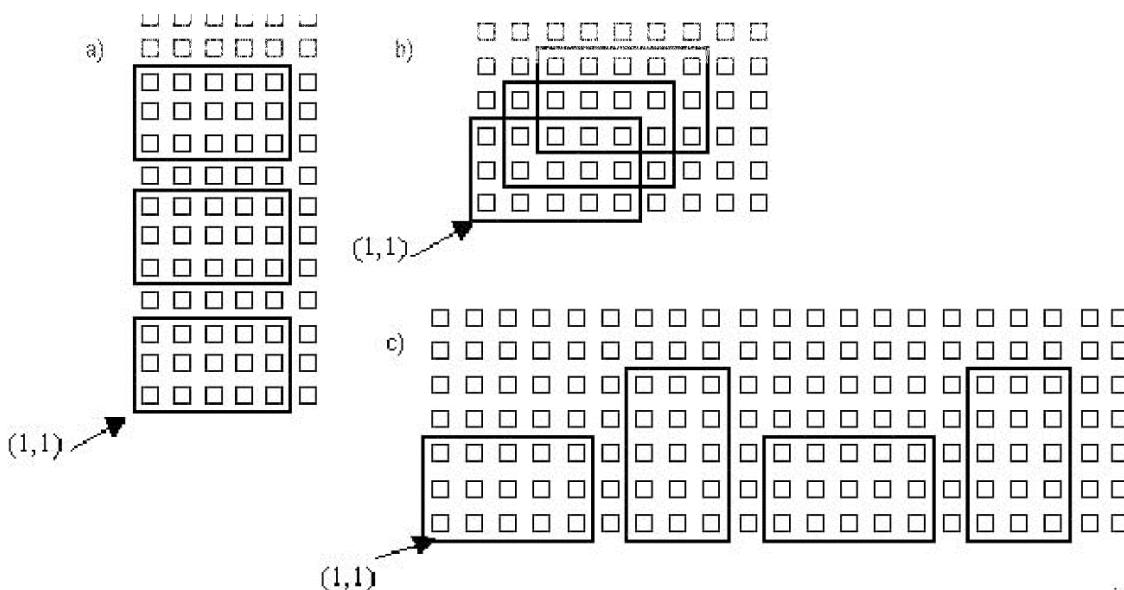


Figura 5.10: Recorridos usando rectángulos de 5x3.

5. Rehacer el recorrido del ejercicio 4.c) trasladando los papeles de cada esquina correspondientes a un lado del rectángulo al vértice siguiente en el recorrido. Por ejemplo, para el rectángulo con vértice en (1,1), los papeles de (1,2) y (1,3) deben ser trasladados a (1,4); los de la calle 4 entre las avenidas 2 y 5 deben ser reubicados en (6,4); y así siguiendo.
6. (a) Escriba un proceso que le permita al robot realizar un rectángulo de base 5 y altura 3 girando en la dirección contraria a la de las agujas del reloj.

- (b) Indique si se produce alguna modificación en los procesos de los ejercicios 4 y 5 si se reemplaza el módulo realizado en 3 por el implementado en 6.a.
7. (a) Escriba el proceso LimpiarEsquina que le permita al robot recoger todas las flores y todos los papeles de la esquina donde se encuentra parado.
- (b) Escriba un programa que le permita al robot recoger todas las flores y papeles de la avenida 89, utilizando los procesos implementados en 7 a).
- (c) Modifique el proceso 6.a) para que el robot realice el rectángulo indicado dejando a su paso todas las esquinas vacías. Para hacerlo debe utilizar el proceso LimpiarEsquina.
- (e) Rehacer el recorrido 4.b) utilizando el proceso definido en 7 c).
8. Programe al robot para que recorra la ciudad de la siguiente manera: primero debe recorrer la avenida 1 juntando todas las flores que encuentre, luego debe recorrer la calle 1 juntando todos los papeles que encuentre. Luego recorre la avenida 2 y la calle 2 de la misma manera y así siguiendo. Implemente un módulo para recorrer la avenida y otro módulo para recorrer la calle.
9. (a) Implemente un proceso para que el robot recorra una avenida juntando flores y se detenga cuando haya juntado 30 flores (seguro existe dicha cantidad).
- (b) Modifique el proceso implementado en (a) sabiendo que las 30 flores pueden no existir.
- (c) Implemente un programa que recorra todas las avenidas de la ciudad, utilizando el proceso implementado en (a).
10. (a) Implemente un proceso para que el robot recorra una calle y se detenga cuando encuentre un papel (seguro existe). Este proceso debe informar la cantidad de pasos dados hasta encontrar el papel.
- (b) Modifique el proceso implementado en (a) sabiendo que el papel puede no existir y en dicho caso debe informar 999.
- (c) Implemente un programa que recorra todas las calles de la ciudad, utilizando el proceso implementado en (b).
11. Programe al Robot para que realice el recorrido de la figura 5.11 utilizando un proceso que permita hacer un escalón.

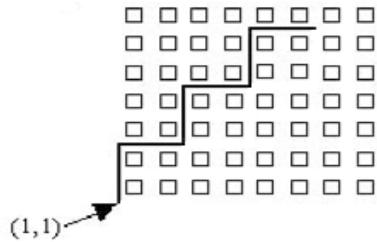


Figura 5.11: Recorrido en escalera de tres escalones.

# 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 compartan 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 decírsele la longitud del lado a realizar en cada caso.

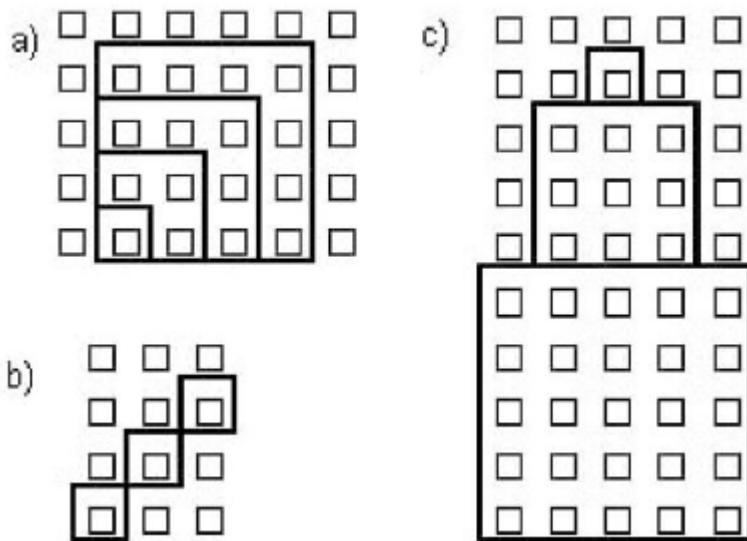


Figura 6.1: ¿Cómo se hace para que un mismo proceso sirva para realizar todos los cuadrados?

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

(2)

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
        cuadrado(long)                                {3}
    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 : tipoi
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

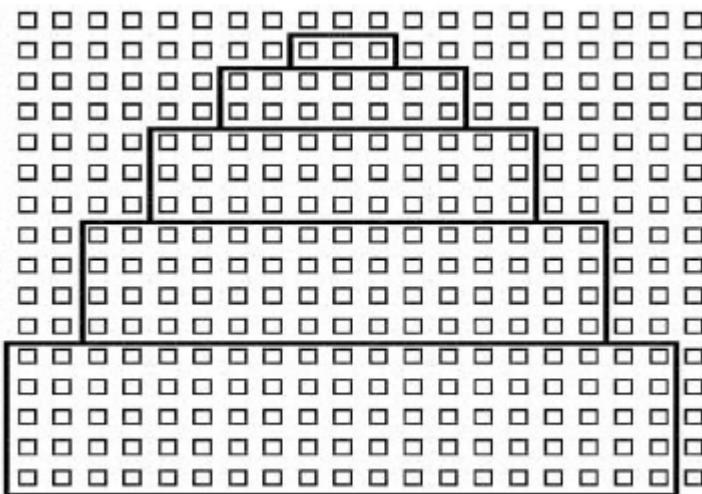


Figura 6.2: Torre de Rectángulos

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:

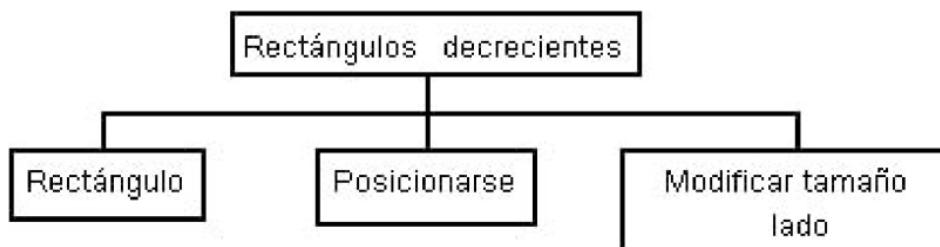


Figura 6.3: Descomposición Top-Down del ejemplo 6.4

```

{ 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.

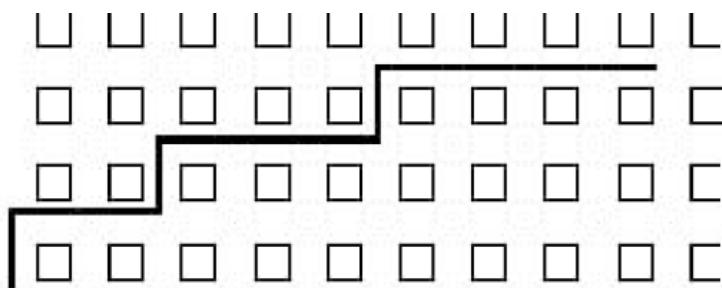


Figura 6.4: Recorrido en escalones

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:

<pre><b>proceso</b> escalon (<i>E entra1:numero</i>;  <i>E entra2:numero</i>)  <b>comenzar</b>  <b>repetir</b> entra1      mover      derecha  <b>repetir</b> entra2      mover      izquierda  <b>fin</b></pre>	<pre><b>proceso</b> escalon(<i>E entra1:numero</i>;  <i>E entra2: numero</i>)  <b>comenzar</b>      Pos(PosAv+entra2, PosCa+Entral)      izquierda  <b>repetir</b> entra2      mover      izquierda  <b>repetir</b> entra1      mover      Pos(PosAv+entra2, PosCa+Entral)  <b>repetir</b> 2      derecha  <b>fin</b></pre>
--	---



¿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 *entra1* 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 ejemplificará 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 precondición 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                                (1)
    {Cada esquina tendrá una flor menos que la anterior}
    mientras (PosCa<11)
        repetir cuantas
            depositarFlor

```

```
    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.

## Ejercitación

1. Escriba un proceso que le permita al robot realizar un cuadrado a partir de la esquina donde está parado, girando en la dirección de las agujas del reloj y recibiendo como dato la longitud del lado.
2. Utilice el proceso de 1. para realizar los recorridos de la figura 6.5 a partir de (1,1).

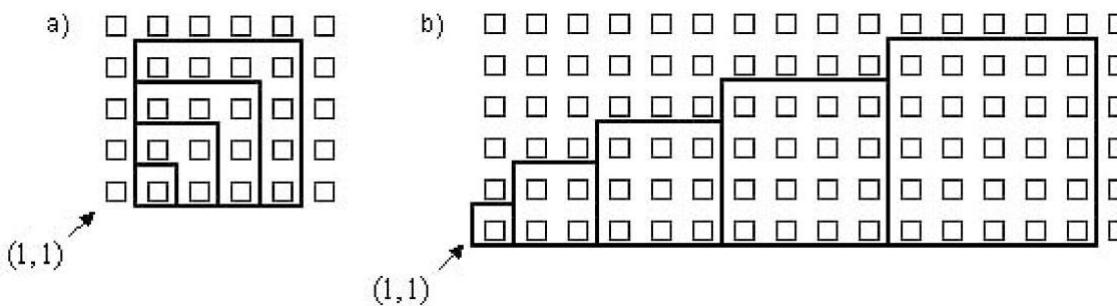


Figura 6.5: Recorridos con cuadrados

3. Escriba un proceso que le permita al Robot realizar un rectángulo a partir de la esquina donde está parado cuyas dimensiones, alto y ancho, se reciben.
4. Utilice el proceso realizado en 3. para que el Robot efectúe los recorridos de la figura 6.6 a partir de (1,1).

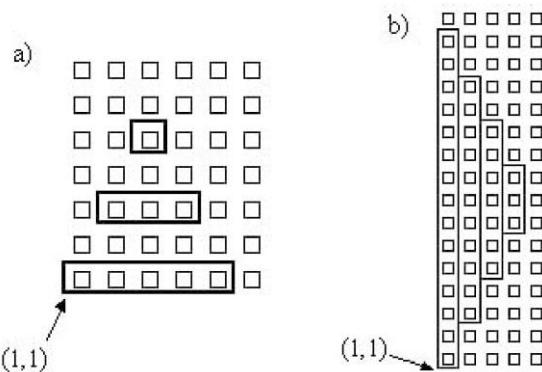


Figura 6.6: Recorridos con rectángulos

5. Programe al robot para que realice cada uno de los cuatro recorridos de la figura 6.7.
6. a) Escriba un proceso que le permita al robot recorrer una avenida cuyo número se ingresa como parámetro de entrada.

- b) Utilice el proceso de 6.a) para recorrer todas las avenidas de la ciudad.
- c) Utilice el proceso de 6.a) para recorrer las avenidas 5, 6, 7 ... 15.
- d) Utilice el proceso de 6.a) para recorrer las avenidas pares de la ciudad.

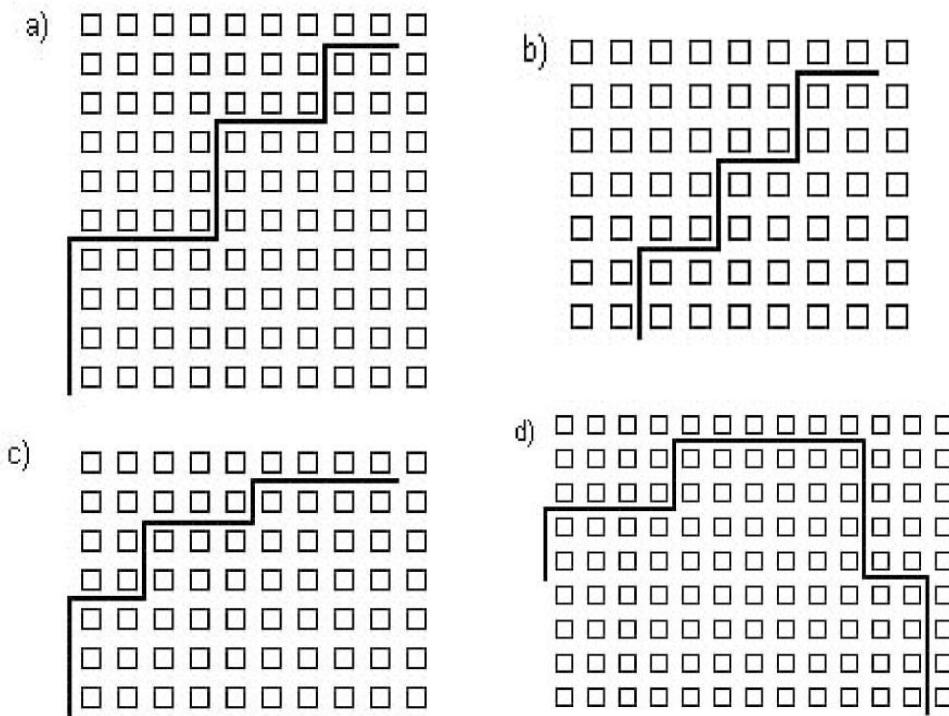


Figura 6.7: Recorrido con escalones.

7. Programe al robot para que realice un módulo CalleFlor que recorra una calle cuyo número se ingresa como parámetro, hasta juntar tantas flores como lo indica otro parámetro de entrada que este módulo recibe. La cantidad de flores seguro existe.
8. Programe al robot para que realice un módulo Avenida que recorra una avenida, cuyo número se ingresa como parámetro, hasta dar tantos pasos como los indicados por otro parámetro de entrada que este módulo recibe. Es decir, si recibe los valores 3 y 1, debe dar 1 paso en la avenida 3; si recibe 12 y 5 debe dar 5 pasos en la avenida 12; y así sucesivamente. En cambio, si recibe algún valor negativo no debe dar pasos. Considere que la cantidad máxima de pasos que podrá dar es 99, cualquier valor que reciba mayor que 99, implicará realizar sólo hasta 99 pasos. Los números de avenida seguro son entre 1 y 100.



# 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) {1}
                mover
                SumarFloresEsquina(totalFlores)
                Informar(totalFlores) {3}
            fin
        variables
            R-info : robot1
        comenzar
            AsignarArea(R-info,ciudad)
            Iniciar(R-info, 1 , 1)
        fin

```

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

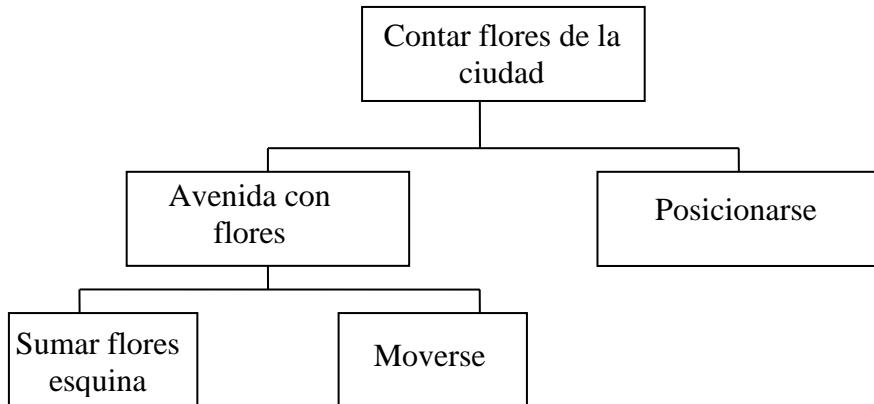


Fig. 7.1: Top-Down del ejemplo 7.2

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                                {1}
  ancho := 19
  alto := 5
  repetir 5
    Rectangulo(ancho,alto,cantVertices)           {2}
    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.



## Ejercitación

1. Escriba un programa que le permita al robot informar la cantidad total de flores y la cantidad total de papeles que hay en toda la ciudad. Para hacerlo, utilice un proceso que recorra una calle cuyo número recibe como parámetro y devuelva la información correspondiente.
2. El robot debe limpiar de flores las calles impares de la siguiente forma: toda flor que se encuentre en una calle impar debe ser trasladada a la calle par siguiente sobre la misma avenida. Por ejemplo si en (4,1) hay una flor, debe llevarse a (4,2). Al terminar el recorrido debe informar la cantidad total de flores que trasladó.
3. Escriba un programa para que el robot recorra la avenida 9 depositando en cada esquina lo que haga falta para que la cantidad de flores supere en 1 a la cantidad de papeles. Si no tiene en su bolsa lo necesario para hacerlo debe detener recorrido. Al finalizar debe informar la cantidad de esquinas que pudo completar adecuadamente. Si el recorrido quedo incompleto debe retornar a (9,1).
4. Programe al robot para que recorra las calles de la ciudad. Por cada calle determine si debe depositar una flor ó un papel en cada esquina dependiendo si el total de flores de la calle es mayor o igual que el total de papeles (deposita una flor por cada esquina) o un papel en caso contrario. Al terminar el recorrido de todas las calles debe informar cuantas de las calles fueron completadas con flores.
5. Escriba un programa que le permita al robot recorrer las calles impares de la ciudad. Cada calle debe recorrerse sólo hasta encontrar una esquina con alguna flor o algún papel o ambos, que seguro existe. Al finalizar cada calle debe informarse cuantos pasos se ha dado hasta encontrar la esquina.
6. Escriba un programa que le permita al robot recorrer cuadrados hasta encontrar un cuadrado con exactamente 3 flores y 2 papeles (seguro existe). El primer cuadrado es de lado 99 y los siguientes van decrementando en uno el tamaño del lado (98, 97 y así sucesivamente).



# Ejercicios Adicionales

1. Escriba un programa que le permita al robot recorrer todas las avenidas de la ciudad. Cada avenida debe recorrerse sólo hasta encontrar una esquina vacía (sin flor ni papel) que seguro existe. Además a medida que se recorre cada avenida debe informar si la misma tuvo a lo sumo 45 flores (hasta que encontró la esquina).

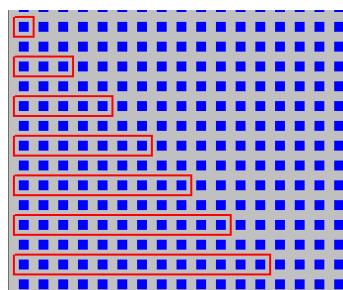
**Nota:** Se debe usar Modularización.

2. Escriba un programa que le permita al robot recorrer todas las avenidas de la ciudad. Al finalizar el recorrido debe informar la cantidad de esquinas con exactamente 20 flores y la cantidad avenidas con menos de 60 papeles.

**Nota:** Se debe usar Modularización y no modificar la cantidad de papeles/flores de las esquinas.

3. Escriba un programa que le permita al robot realizar el siguiente recorrido, comenzando en la esquina (1,1) juntando todas las flores y papeles de cada esquina. Al finalizar el recorrido debe informar la cantidad total de flores y de papeles que tiene en la bolsa.

**Nota:** Se debe usar Modularización.



(1,1)

4. Programe al robot para que recorra la ciudad por avenidas, juntando papeles, hasta encontrar una avenida con exactamente 25 flores. Cuando encuentra la avenida con exactamente 25 flores debe recorrer toda la calle 75 (desde la avenida 1) y dar tantos pasos como papeles juntó en todas las avenidas recorridas.

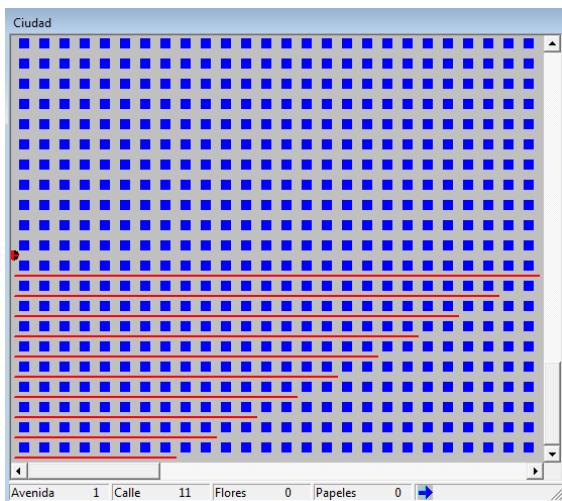
**Nota:** la avenida con 25 flores seguro existe. La cantidad de papeles juntados (entre todas las avenidas recorridas) seguro es menor a 100. Las esquinas pueden modificarse. Modularizar.

**Ejemplo:** suponga que el robot encuentra que la avenida 5 tiene exactamente 25 flores, y durante su recorrido (avenidas 1, 2, 3, 4 y 5) juntó 62 papeles. Entonces debe recorrer la calle 75 y dar 62 pasos.

5. Escriba un programa que le permita al robot recorrer 10 calles de la ciudad (como se muestra en la figura). En cada calle debe juntar las flores y los papeles. Al finalizar cada calle informar la cantidad de esquinas con el doble de flores que papeles. Al finalizar el recorrido debe informar la cantidad total de papeles y de flores

recogidas. El recorrido comienza en (1,1). En la primer calle se deben recorrer 8 avenidas, en la siguiente 2 avenidas más (es decir 10 avenidas) y así incrementar de a dos avenidas para las calles restantes.

**Nota:** se debe usar Modularización.



6. Realice un programa para que el robot recorra 15 cuadrados los cuales comienzan siempre en la esquina (1,1). En cada cuadrado debe juntar las flores y los papeles. Al finalizar los 15 cuadrados debe informar cuántos cuadrados tenían más de 20 flores.

**Notas:** Modularice. Se pide que como mínimo exista un módulo que realice un cuadrado. El primer cuadrado debe ser de lado 1, el segundo de lado 2 y así sucesivamente hasta llegar al cuadrado 15 el cual es de lado 15.

7. EXAMEN AÑO 2013: Escriba un algoritmo para que el robot recorra la avenida 8 juntando todas las flores y todos los papeles hasta encontrar una esquina vacía. Luego debe recorrer un rectángulo que comience en (1,1), donde el alto del rectángulo es igual a la cantidad de flores juntadas en la avenida 8 y el ancho o base es igual a la cantidad de papeles juntados en la avenida 8.

**Ejemplo:** si cuando el robot termina de recorrer la avenida 8 (porque encontró la esquina vacía) juntó 5 flores y 4 papeles, debe posicionarse en (1,1) y hacer un rectángulo donde el alto es 5 y el ancho es 4.

**Nota:** SE DEBE USAR MODULARIZACION (como mínimo debe haber un módulo para la avenida 8, y otro para el rectángulo). LA ESQUINA VACIA DE LA AVENIDA 8 SEGURO EXISTE. EL TOTAL DE FLORES Y DE PAPELES DE LA AVENIDA 8  $\leq 99$

8. EXAMEN AÑO 2013: Escriba un algoritmo para que el robot recorra las calles impares de la ciudad. Cada calle debe recorrerse hasta juntar al menos 10 flores. Una vez que ha recorrido todas las calles debe recorrer la avenida 10 la avenida 11 y la avenida 12 juntando todos los papeles. Al finalizar de recorrer las tres avenidas debe informar la cantidad total de papeles juntados.

**Nota:** SE DEBE USAR MODULARIZACION (como mínimo debe haber un módulo para las calles, y otro para las avenidas). SEGURO QUE CADA CALLE TIENE AL MENOS 10 FLORES.