

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 Rinfo 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 Rinfo.

Además se introducirán y repasarán algunos conceptos básicos de la lógica proposicional para representar condiciones complejas utilizadas en las estructuras del ambiente del robot Rinfo, 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 Rinfo. Operaciones sobre Rinfo. Estructura general de un programa. Estilo de Programación. Ambiente de programación.
- ✓ Estructuras de control en el ambiente de programación del robot Rinfo.
- ✓ Revisión del tema: Propositiones 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 (Rinfo). Operaciones sobre Rinfo. 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 **Rinfo**, 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 Rinfo 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 Rinfo que se desplazará en una única área de una ciudad compuesta por 100 avenidas y 100 calles.

El robot **Rinfo** 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.

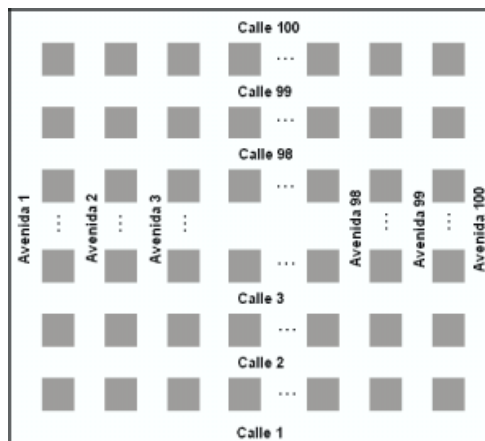


Figura 2.1: La ciudad del robot

La ciudad, en la que **Rinfo** 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 Rinfo

El conjunto de acciones que **Rinfo** puede realizar es muy reducido. Cada una de estas acciones corresponde a una instrucción, entendible por él y que debe tener un modo unívoco 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 **Rinfo**. 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 **Rinfo**.



Sintaxis	Semántica
Iniciar (robot,posición)	Instrucción primitiva que posiciona al robot en la esquina indicada orientado hacia el norte. En este curso siempre debemos posicionar al robot en la esquina (1,1) para comenzar su ejecución.
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(Avenida, Calle)	Instrucción que requiere dos valores Avenida y Calle, cada uno de ellos en el rango 1..100, y posiciona al robot en la esquina determinada por el par (Avenida,Calle) sin modificar la orientación del robot.
Informar	Instrucción que permite visualizar en pantalla el contenido almacenado en alguna variable o valores literales.

Tabla 2.1: Sintaxis del robot Rinfo

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, que estarán desplazándose por la ciudad, junto al conjunto de instrucciones que cada tipo de robot debe ejecutar (en este curso sólo se declarará un tipo de robot).
3. Habrá otro espacio asignado para los módulos (se verá en capítulos sucesivos).
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 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” cuyo código corresponde a la solución del problema que se está resolviendo

variables

se declara una variable que representa al robot, será llamada Rinfo

comenzar

se asigna el área donde se desplazará Rinfo (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 Rinfo).

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 Rinfo 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 ó 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 Rinfo utiliza ciertas reglas sintácticas adicionales las cuales son muy estrictas relacionadas con la indentación y el uso de mayúsculas y minúsculas.

Estas reglas, aunque pueda 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 mas 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 Rinfo, 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: Iniciar. 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 Rinfo utiliza la sintaxis del robot descripta previamente. Para codificar un algoritmo en el lenguaje del robot Rinfo 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 se 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 Rinfo permite visualizar durante la ejecución, el recorrido que realiza el robot Rinfo 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 Rinfo.

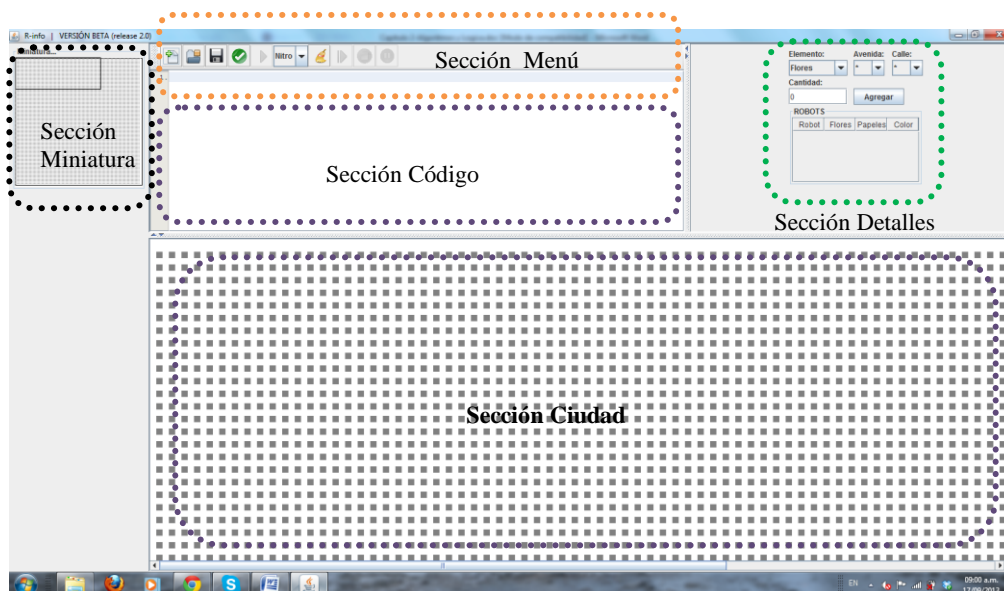


Figura 2.2: Ambiente de programación de Rinfo

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 qué 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 programa, cargar un programa ya escrito, crear un nuevo programa y guardarlo, compilar y/o ejecutar un programa hecho y reiniciar el programa. A medida que se utilice el ambiente de programación se podrá

observar que existen otras operaciones, aunque las antes mencionadas seguramente son las únicas que se usarán en este curso.

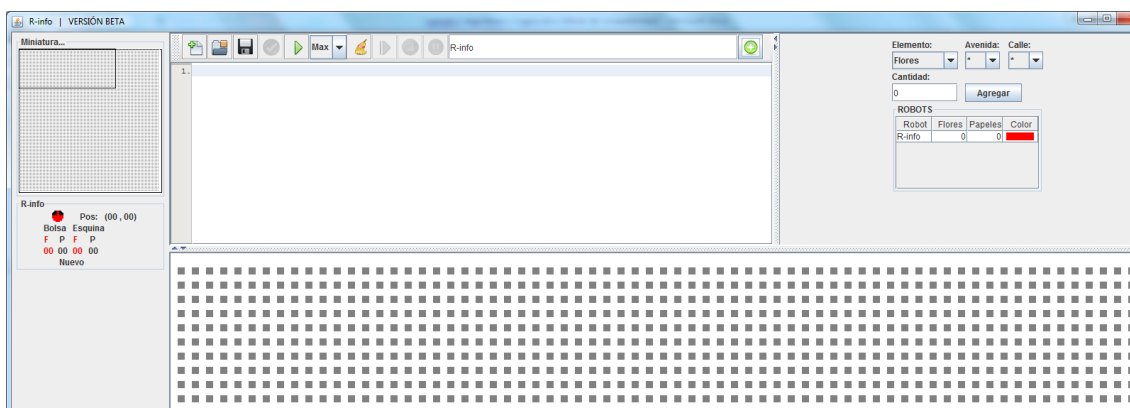
Sección Código: en esta sección se visualiza el código correspondiente al programa con el que se está trabajando; puede ser un programa ya escrito o uno que se esté escribiendo en ese momento.

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 Rinfo).

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 Rinfo (recordar que en este curso sólo se trabaja con un único robot). Durante este curso el nombre de robot que utilizaremos será Rinfo.



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 .lmre. Esta acción permitirá posteriormente editar el programa para volver a utilizarlo.

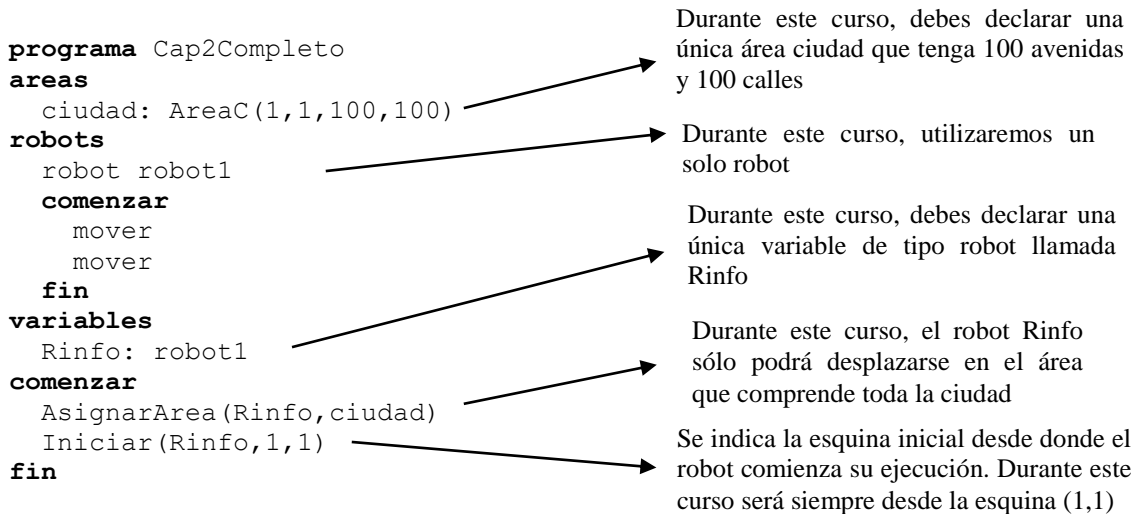
Luego de escrito el programa, es necesario realizar el proceso de compilación seleccionando la opción “Compilar” de la Sección Menú. El proceso de compilación se encargará de verificar la sintaxis del programa escrito y en caso de existir errores, visualizará los mensajes correspondientes.

Una vez que la compilación es realizada en la parte izquierda de la pantalla (debajo de la información miniatura) se visualizará la información del robot Rinfo

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 Rinfo 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 Rinfo. 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 Rinfo. 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 Rinfo, el cual debe caminar dos cuadras a partir de la esquina (1,1) sería la siguiente:



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 Rinfo 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 Rinfo. El funcionamiento de cada una de esas estructuras se explicó en forma detallada en el capítulo 1.

2.3.1 Secuencia

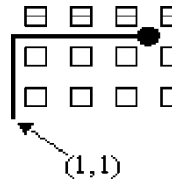
Está definida por un conjunto de instrucciones que se ejecutarán una a continuación de otra.

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
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo,ciudad)
  Iniciar(Rinfo,1,1)
fin

```



La instrucción *Iniciar* ubica al robot Rinfo en la esquina (1,1) orientado hacia el norte (hacia arriba). Luego debe comenzar la ejecución de las instrucciones correspondientes al robot Rinfo. Por lo tanto, avanza dos cuadradas en línea recta por lo que queda posicionado en la calle 3. Dobla a la derecha para seguir avanzando por la calle 3 y camina tres cuadradas por lo que finaliza el recorrido parado en (4,3). Rinfo 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
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo,ciudad)
  Iniciar(Rinfo,1,1)
fin

```

La instrucción $Pos(4,3)$ permite que el robot Rinfo “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
  Rinfo: robot1

  comenzar
    AsignarArea(Rinfo,ciudad)
    Iniciar(Rinfo,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
  Rinfo: robot1
  comenzar
    AsignarArea(Rinfo,ciudad)
    Iniciar(Rinfo,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 que 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
  Rinfo: robot1
  comenzar
    AsignarArea(Rinfo,ciudad)
    Iniciar(Rinfo,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 desde la avenida 1 hasta la avenida 2 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
```

```
Rinfo: robot1
comenzar
  AsignarArea(Rinfo, ciudad)
  Iniciar(Rinfo, 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
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo, ciudad)
  Iniciar(Rinfo, 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 que 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 Rinfo 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 Rinfo, este algoritmo se traduce en el siguiente programa:

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

Se puede notar que la sentencia *Pos(1,4)* no modifica la orientación del robot Rinfo. 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
    Rinfo: robot1
  comenzar
    AsignarArea(Rinfo,ciudad)
    Iniciar(Rinfo,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
    Rinfo: robot1
  comenzar
    AsignarArea(Rinfo,ciudad)
    Iniciar(Rinfo,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 Rinfo 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
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo,ciudad)
  Iniciar(Rinfo,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 Rinfo 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 Cap2Ejemplo12 areas ciudad: AreaC(1,1,100,100) robots robot robot1 comenzar Pos(4,2) repetir 2 mover derecha repetir 3 mover fin variables Rinfo: robot1 comenzar AsignarArea(Rinfo,ciudad) Iniciar(Rinfo,1,1) fin</pre>	<pre>programa Cap2Ejemplo12 areas ciudad: AreaC(1,1,100,100) robots robot robot1 comenzar Pos(4,2) mientras (PosCa<=3) mover derecha mientras (PosAv<=6) mover fin variables Rinfo: robot1 comenzar AsignarArea(Rinfo,ciudad) Iniciar(Rinfo,1,1) fin</pre>
--	--



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

Análisis de la sintaxis del robot

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

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

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 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 ó 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 ó 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 **ó** 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.

Conectivo	Simbolización en en ambiente de programación de Rinfo
y	&
o	
no	~

Tabla 2.3: Conectivos lógicos ó 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. $\text{PosCa} < 100$.

Si se llama r = “ $\text{PosCa} < 100$ ”, la proposición se simboliza como: r .

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

El proceso para simbolizarlas consiste en tres pasos:

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

Ejemplos de simbolización de proposiciones moleculares

Juan es estudiante y es jugador de fútbol.

p = "Juan es estudiante"

q = "Juan es jugador de fútbol"

Simbolizando $p \ \& \ q$

No es cierto que $\text{PosCa} = 100$.

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 = "Hay flor en la esquina"

q = "hay papel en la bolsa"

r = "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 = "hay flor en la bolsa"

q = "hay flor en la esquina"

Simbolizando $(\sim p \ \& \ q)$

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

2.4.3 Tablas de verdad. Repaso

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

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

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

2.4.3.1 Conjunción. Tabla de verdad

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

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

Dadas dos proposiciones cualesquiera p y q , si ambas son verdaderas, entonces $p \ \& \ q$, que representa la conjunción de p y q , es verdadera. Cualquier otra combinación da

como resultado una proposición molecular falsa. La tabla 2.4 representa la tabla de verdad de la conjunción $p \& q$ utilizando las cuatro combinaciones posibles de valores de verdad para p y q . Por lo tanto, si $p \& q$ es una proposición verdadera entonces p es verdadera y q también es verdadera.

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

p	q	$p \& q$
V	V	V
V	F	F
F	V	F
F	F	F

Tabla 2.4: Conjunción ($p \& q$)

Ejemplos:

6 es un número par y divisible por 3.
 p = “6 es un numero par”
 q = “6 es divisible por 3”
 p es verdadera y q también, por lo tanto $p \& q$ es verdadera.

Suponiendo que el robot se encuentra situado en la esquina (1,1)
 p = “PosCa=1”
 q = “PosAv=2”
 p es verdadera y q es falsa, por lo tanto $p \& q$ es falsa.

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

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

```
programa Cap2Ejemplo13
areas
  ciudad: AreaC(1,1,100,100)
robots
  robot robot1
comenzar
  Pos(45,70)
  si ((HayFlorEnLaBolsa) & ~(HayFlorEnLaEsquina))
    depositarFlor
  fin
variables
  Rinforobot1
comenzar
  AsignarArea(Rinforobot1,ciudad)
  Iniciar(Rinforobot1,1,1)
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 .

p	q	$p \mid q$
V	V	V
V	F	V
F	V	V
F	F	F

Tabla 2.5: Disyunción ($p \mid q$)

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) | (HayPapelEnLaEsquina))
    repetir 4
      mover
    fin
variables
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo,ciudad)
  Iniciar(Rinfo,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 2.6 representa los posibles valores de verdad de dos valores de verdad posibles de p .

p	$\sim p$
V	F
F	V

Tabla 2.6: 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
  Rinfo: robot1
comenzar
  AsignarArea(Rinfo, ciudad)
  Iniciar(Rinfo, 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 “~” es el que tiene mayor prioridad, es decir, es el que primero se evalúa; seguido por “&” y “|”. Estos dos últimos poseen igual prioridad. Ante una expresión que utilice varias conjunciones y/o disyunciones, se evaluarán 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 Teoremas de Lógica Proposicional

Es frecuente encontrar proposiciones que se necesita saber si son equivalentes entre sí. O bien, proposiciones que por su estructura no se comprenden fácilmente y requieren una simplificación o que sean expresadas de otra forma. Para estos casos, se enuncian a continuación algunos de los teoremas fundamentales de *Lógica Proposicional o Algebra de Bool* que suelen ser de ayuda para transformar una proposición en otra equivalente.

Teorema 1. El resultado de aplicar cualquiera de las tres operaciones antes definidas (negación, conjunción ó disyunción), a variables booleanas, es otra variable booleana y además el resultado es único.

Teorema 2. Ley de idempotencia. Tanto la suma como el producto de una variable booleana consigo misma da como resultado la misma variable:

$$\begin{aligned}p \mid p &= p \\p \& p &= p\end{aligned}$$

Teorema 3. Ley de involución. Una variable booleana negada dos veces, da como resultado la misma variable:

$$\sim (\sim p) = p$$

Teorema 4. Ley conmutativa. Se define respecto a la conjunción (y a la disyunción) y nos dice que el orden de los sumandos (factores) no altera el resultado:

$$\begin{aligned}\text{Respecto a la conjunción: } p \& q &= q \& p \\ \text{Respecto a la disyunción: } p \mid q &= q \mid p\end{aligned}$$

Teorema 5. Ley asociativa. Se define respecto a la conjunción (y a la disyunción) de la siguiente forma:

$$\begin{aligned}\text{Respecto a la conjunción: } p \& (q \& r) &= (p \& q) \& r = p \& q \& r \\ \text{Respecto a la disyunción: } p \mid (q \mid r) &= (p \mid q) \mid r = p \mid q \mid r\end{aligned}$$

Teorema 6. Ley distributiva.

$$\begin{aligned}\text{Respecto a la conjunción: } p \mid (q \& r) &= (p \mid q) \& (p \mid r) \\ \text{Respecto a la disyunción: } p \& (q \mid r) &= p \& q \mid p \& r\end{aligned}$$

Teorema 7. Ley de absorción.

$$\begin{aligned}p \mid (p \& q) &= p \\ p \& (p \mid q) &= p\end{aligned}$$

Teorema 8. Leyes De Morgan. Pueden ser generalizadas a "n" variables.

$$\begin{aligned}\text{Respecto a la conjunción: } \sim (p \& q) &= \sim p \mid \sim q \\ \text{Respecto a la disyunción: } \sim (p \mid q) &= \sim p \& \sim q\end{aligned}$$

2.6 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 Rinfo 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 un papel de la esquina (50,28) 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 (10,10).
4. Escriba un programa que le permita al robot recorrer el perímetro del cuadrado determinado por (1,1) y (10,10) recogiendo, de ser posible, un papel en cada esquina.
5. Escriba un programa que le permita al robot recorrer el perímetro del cuadrado determinado por (1,1) y (10,10) recogiendo, de ser posible, un papel en cada vértice.
6. Escriba un programa que le permita al robot dejar todas las flores que lleva en su bolsa en la esquina (50,50).
7. 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.
8. 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.
9. 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 que calle dejó los papeles.
10. 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).
11. Programe al robot para que recorra la calle 11 depositando, de ser posible, una flor en cada esquina. Si durante el recorrido se queda sin flores debe intentar depositar papeles. Si no tiene ninguna de las dos cosas debe seguir caminando hasta terminar la calle.
12. Programe al robot para que recorra las 5 primeras avenidas juntando en cada esquina todas las flores y papeles.
13. 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.
14. Programe al robot para que recorra todas las calles recogiendo, de ser posible, de cada esquina una flor y un papel.
15. Programe al robot para que recorra la calle 1 buscando una flor en las avenidas impares. Cada vez que encuentre una flor deberá realizar un cuadrado de lado 1 tomando como vértice inferior izquierdo de dicho cuadrado, la esquina donde encontró la flor.
16. El robot debe realizar una limpieza de la calle 17 muy especial. Debe dejar sólo flores en las avenidas impares y sólo papeles en las pares. Para no cargar muchas

cosas en su bolsa hará lo siguiente: todos los papeles de una avenida impar serán trasladados a la avenida par siguiente y todas las flores de una avenida par serán trasladadas a la avenida impar siguiente. Por simplicidad, considere que el robot comienza el recorrido con la bolsa vacía.

17. Programe al robot para que recorra las 10 primeras avenidas. Cada avenida debe recorrerse hasta encontrar una esquina con flor y sin papel. Suponga que seguro existe esa esquina en cada avenida.
18. Modifique el ejercicio 17, suponiendo que no necesariamente existe la esquina con flor y sin papel en cada avenida.