

PROGRAMACION LOGICA

herramienta de quinta generacion

NIVELES BASICO E INTERMEDIO

M. C. RICARDO FUENTES COVARRUBIAS

UNIVERSIDAD DE COLIMA

Primera actualizacion

Abril de 2000

CAPITULO	CONTENIDO	PAGINAS
1.	INTRODUCCIÓN	1
2.	INTERFAZ DE USUARIO	2
3.	SINTAXIS Y DIVISIONES DE PROGRAMA	4
4.	TIPOS DE DOMINIOS	6
5.	PREDICADOS ESTANDARES	11
6.	OBJETOS COMPUESTOS	17
7.	LISTAS	21
8.	ARCHIVOS	25
9.	GRAFICOS Y SONIDOS	29
10.	ELEMENTOS MATEMATICOS	39
11.	BASES DE DATOS DINAMICAS	43
12.	BASES DE DATOS EXTERNAS	46
13.	APLICACIONES AVANZADAS	51
14.	CONCLUSIONES Y BIBLIOGRAFIA	54
	APÉNDICE A. PROGRAMAS PARA EVALUACIÓN DE CONOCIMIENTOS	55
	APÉNDICE B. VISUAL PROLOG O WIN PROLOG SU RELACION CON TURBO PROLOG	57

PROLOGO

La presente obra es una contribución al avance de nuevas tecnologías generando aplicaciones a paradigmas surgidos en la década de los 60's y 70's que permanecieron en un proceso de estancamiento debido al poco avance en los sistemas hardware y que en su momento fueron material propicio para las novelas y películas de ciencia ficción que proliferaron de manera exagerada, pero que en este inicio de milenio el futuro nos empieza a alcanzar y muchos proyectos se han convertido en crudas realidades que se atreven a cuestionar incluso los procesos naturales de generación de la vida con esquemas de manipulación genética en la vida animal y vegetal y lo que en un principio fue acuñado como inteligencia artificial ha derivado en vida artificial, inteligencia mecánica, programación genética o redes neuronales.

La inteligencia artificial que en un principio constaba de áreas no muy claras e incluso no se consideraba como una disciplina formal, hoy en día es desde el punto de vista de este autor, el área de las ciencias computacionales hacia la cual se enfocan los desarrollos de nuevas tecnologías.

Con este trabajo se inicia una serie de libros cortos enfocados a cada una de las áreas de la Inteligencia artificial con un alto sentido didáctico orientado a niveles de licenciatura y maestría.

CAPITULO

1

INTRODUCCION

Prolog es un lenguaje orientado a la inteligencia artificial, es muy difícil indicar fechas o datos exactos que dan inicio a esta disciplina, empero, formalmente indicaremos que está asociada con la primera computadora que se diseñó, tomando como base este dato es obligado señalar que existen dos arquitecturas básicas: La maquina de cálculos de Blas pascal y la maquina analítica de Charles Babbage. Las cuales definen los dos modelos planteados para las computadoras modernas la primera y más usual dadas las condiciones de avance tecnológico imperantes en 1947 fueron planteadas por John Von Newman y la otra planteadas por Alan Turing misma que fue desechada por considerarla poco operante, sin embargo esta ultima concepción fue retomada en 1981 cuando el gobierno japonés decidió liberar fondos para apoyar el programa de computadoras de la quinta generación.

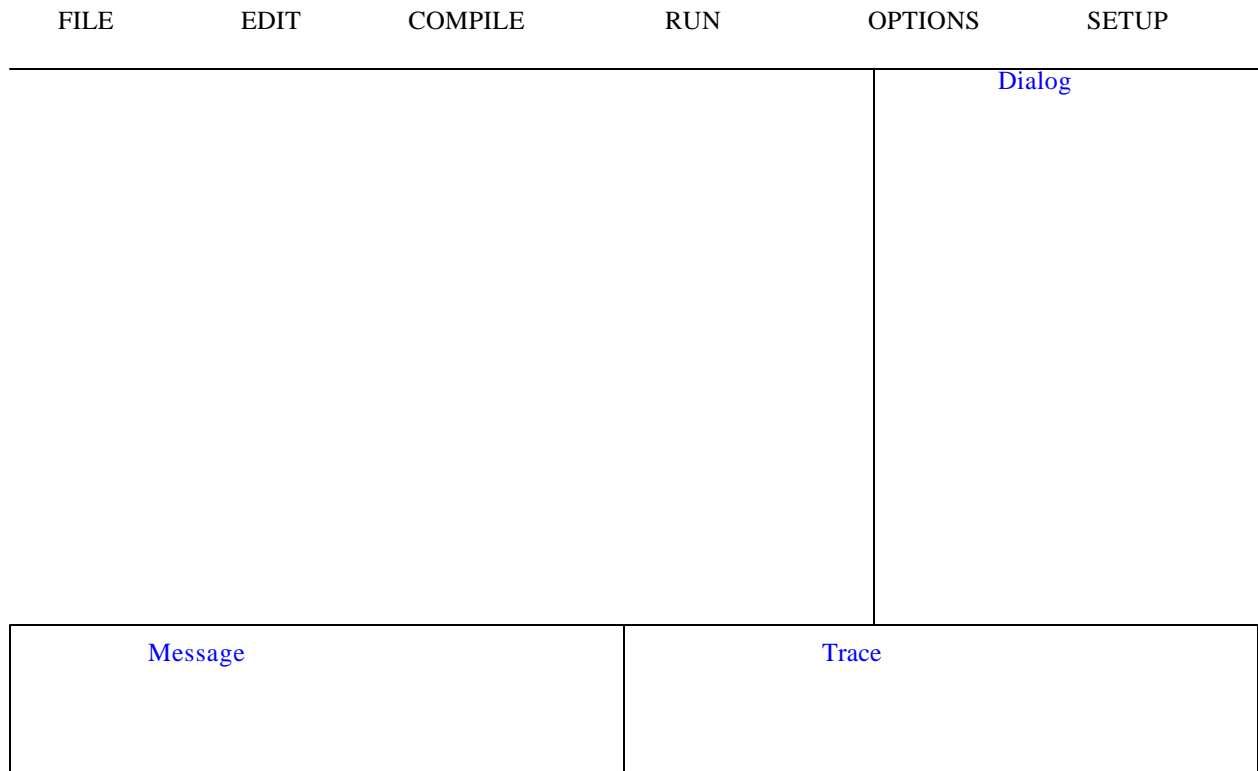
En este orden de ideas, es obligado indicar que cada arquitectura de computadora implica un nivel de programación diferente, la definida por Von Newman requiere programación algorítmica es decir, definiendo una serie de ordenes que tienen una secuencia lógica y la planteada por Alan Turing utilizando un esquema heurístico es decir, tomando en cuenta esquemas definidos de búsqueda y toma de decisiones a partir de una o varias funciones de evaluación.

Lo anterior nos lleva a la necesidad de indicar que para efectos del presente trabajo nos ocuparemos del lenguaje prolog, sin que este signifique ser la única opción pues los lenguajes orientados a la IA son muchos pero la mayoría han caído en desuso, y un estudio al respecto escapa del ámbito de esta obra.

Prolog es el resultado de los trabajos de varios investigadores y en un principio se concibió como programación lógica y surgieron así varios trabajos surgen primero con Robert Kowalski en Edimburgo en la parte teórica y Marteen Van Emden en la parte experimental y por el grupo de inteligencia artificial de la Universidad de Marsella, Francia en 1972, comandado por Alain Colmerauer para su implementación, sin embargo no trascendió hasta 1981 cuando fue adoptado como lenguaje oficial del programa japonés de computadoras de quinta generación. En la primera parte de esta obra utilizaremos el esquema de trabajo de Turbo Prolog ver. 2.0 la cual fue la ultima comercializada por la empresa Borland International, la justificación de su uso es la facilidad para la enseñanza de este curso, sin embargo en la segunda parte utilizaremos Visual Prolog versión 1.2.

CAPITULO 2 INTERFAZ DE USUARIO

La interfaz de usuario que Turbo Prolog presenta es la siguiente:



Existen cuatro ventanas de trabajo, la mayor es la ventana de edición, la utilizaremos para editar el código fuente de los programas. La ventana Dialog es en la que ejecutaremos los programas que contengan objetivos externos. La ventana Message despliega las ordenes relevantes realizadas en la ejecución de un programa y por ultimo la ventana de Trace despliega cada segmento de programa ejecutado cuando el programa se ejecuta en la modalidad paso a paso.

El modulo Files permite manipular los archivos de programa los cuales por default tendrán la extensión.pro: Crear uno nuevo, abrir archivo existente, grabar, grabar cómo, mostrar un directorio o cambiar de directorio y salida temporal a msdos.

La opción de menú Edit se utiliza para generar el código fuente o el programa a desarrollar.

El modulo Run permite ejecutar el programa fuente dependiendo del tipo de objetivos que utilizemos, si tenemos objetivos externos el programa se ejecutará en la ventana de Dialog.

El modulo Compile permite compilar el código fuente del programa tanto para memoria, obj, exe, link o como proyecto.

El modulo Options nos permite editar un archivo de tipo project, definir opciones de compilación o de enlace.

El modulo Setup permite redefinir entre otras cosas los tamaños de ventana, colores, directorios o cargar archivos de tipo sys.

CAPITULO

3

SINTAXIS Y DIVISIONES DE PROGRAMA

3.1 Sintaxis

Prolog es un lenguaje declarativo y modular estructurado pero no monolítico es decir, se crean dos tipos de secciones de programa: a) base de hechos y b) reglas. En su momento indicaremos el uso de cada uno.

Su fundamento se encuentra en la lógica matemática, con lógica proposicional o calculo de predicados.

Comentarios. Los comentarios en un programa Prolog se indican en medio de los símbolos: /* */

Variables. Cuando se definen variables estas deben iniciar con una letra mayúscula pues de lo contrario se les considerará como objetos. Además, una variable puede estar formada por letras, signos o números siempre y cuando la primera sea una letra. Toda variable deberá tener un valor asignado a este proceso se le llama instanciamiento, si no se define un valor para una variable entonces se considera como variable abierta. Una variable instanciada se representa como I mientras que una abierta se representa como O.

Variable blanca o anónima. Es un tipo de variable especial que se representa por el carácter de subrayado el cual puede asumir cualquier valor, es decir una variable anónima por defecto no contiene un valor sino que lo adquiere mediante un proceso de búsqueda.

Hechos. Así se llama a los objetos con su respectivo atributo ejemplo: Juan es amigo de Pedro se representaría así en prolog: amigo(juan,pedro), donde juan y pedro son objetos y amigo es su atributo, observe que juan y pedro tienen la primera letra en minúscula. Lo que puede inferirse, calcularse o determinarse prolog lo maneja mediante una maquina o motor de inferencia el cual se encarga de realizar las búsquedas.

También es muy común que a esta parte del programa se le identifique como base del conocimiento en la cual se definirán todos los objetos con su respectivo atributo agrupados en un mismo bloque de acuerdo con su declaración, es decir, si se tienen declarados los siguientes predicados:

```
Rey(persona)
Reina(persona)
```

Cuando se asignen objetos para cada atributo o predicado, estos deberán designarse en un mismo grupo de la siguiente manera:

```
Rey(felipe).
Rey(juan_carlos).
Reina(isabel).
Reina(sofia).
```

Pues el no hacerlo mostrará un mensaje de error en tiempo de compilación.

Aridad. Se llama así a la cantidad de objetos de que consta un atributo o predicado.

3.2 Divisiones de programa.

En principio las secciones de un programa prolog son las siguientes:

Domains. En esta se define el tipo de objetos que serán utilizados como argumentos de un predicado.

Predicates. Se utiliza para declarar los predicados o relaciones de un objeto.

Goal. En esta sección se definen los objetivos a lograr al ejecutar un programa, existen dos tipos de Objetivos: Internos y externos, los internos se encuentran definidos en la sección goal del programa mientras que los segundos se ejecutan en la ventana goal del entorno de trabajo.

Clauses. En esta sección es en donde prácticamente desglosaremos el programa a ejecutar, indicando los hechos y las reglas.

Operadores lógicos. Los operadores mas comunes son: AND el cual además se representa por una coma (,), OR que se representa por punto y coma (;), IF representado por dos puntos y un guión (:-), el símbolo de igualdad (=), y el de diferencia (<>).

CAPITULO

4

TIPOS DE DOMINIOS

Integer. Caracteres numéricos definidos en el rango de -32,768 a 32,767 (se almacenan en 16 bits de los cuales 15 representan el numero y el bit de mayor peso el signo).

Real. Caracteres numéricos comprendido en el rango de +/-E-307 a +/-E+308 (donde E representa el exponencial) su formato es: signo, numero, punto decimal, fracción, Exponencial, Signo para el exponente y exponente.

Char. Cualquier carácter de los 255 que conforman la lista de códigos ASCII.

String. Su forma es la misma que los de tipo String con la diferencia de que se representan entre comillas dobles.

Symbol. Consta de caracteres consecutivos los cuales pueden ser letras, números y símbolos de subrayado, se utiliza para representar objetos.

File. Existen dos tipos: los estándares y los Archivos estáticos o de usuario. Los primeros no se declaran y los segundos son los que se declaran en esta sección.

Database. Bases de datos dinámicas.

El primer programa:

*/*Ejemplo 1. Hechos */*

```
Domains
cosa=symbol
Predicates
  Tipo_de_animal(animal)
  Tipo_de_cosa(cosa)
Clauses
  Tipo_de_animal(perro).
  Tipo_de_animal(chivo).
  Tipo_de_animal(zorrillo).
  Tipo_de_animal(gato).
  Tipo_de_cosa(mesa).
  Tipo_de_cosa(computadora).
  Tipo_de_cosa(silla).
  Tipo_de_cosa(teléfono).
  Tipo_de_cosa(radio).
```

En la ventana de goal, se ejecuta el programa preguntando al motor de inferencia: Tipo_de_animal(perro) y el motor contestará: yes. Literalmente le abremos preguntado: hay un tipo de animal llamado perro?, buscará en la base de hechos su presencia, si lo encuentra contestará afirmativamente de lo contrario contestará en forma negativa.

Si preguntamos Tipo_de_animal(X), el motor instanciará a la variable X todos los objetos que tengan el atributo Tipo_de_animal y los mostrará.

Si se pregunta Tipo_de_animal(_), el motor de inferencia no buscará ningún objeto, solo el atributo y contestará afirmativa o negativamente.

4.1 REGLAS.

Las reglas se utilizan en PROLOG para significar que un hecho depende de uno o más hechos. Son la representación de las implicaciones lógicas del tipo $p \rightarrow q$ (si p entonces q).

- * Una regla consiste en una cabeza y un cuerpo, unidos por el signo ":-".
- * La cabeza esta formada por un único hecho.
- * El cuerpo puede ser uno o más hechos (conjunción de hechos), separados por una coma (","), que actúa como el "y" lógico.
- * Las reglas finalizan con un punto (".").

Con ellas se definen propiamente las condiciones de manipulación de una base de hechos utilizando adecuadamente los operadores lógicos necesarios. Ejecute los programas Ejemplo 2 y 3.

```
/* Ejemplo 2*/
Domains
  Nombre=symbol
Predicates
  Padre(nombre,nombre,nombre)
Clauses
  Padre(leonardo,karina,rosa).
  Padre(carlos,javier,rosa).
Todos:-
  Padre(X,Y,Z), fail. /* patrón de búsqueda del motor de inferencia */
```

Al ejecutar el programa se pregunta al motor de inferencias: padre(Quien,Quien2,Quien3) y éste mostrará instanciado a la variable Quien, Quien2, y Quien3, todos los objetos del atributo padre.

4.2 FALLO Y CORTE.

Son dos estructuras muy útiles en esquemas de búsqueda, fallo se utiliza para obligar al motor de inferencia hacer una búsqueda exhaustiva hasta completar los elementos requeridos para el instanciamiento de una variable con el patrón de búsqueda. La sintaxis es **fail**.

Corte limita la búsqueda del motor de inferencia al primer instanciamiento, es decir, cuando encuentra la primera solución termina la búsqueda, sin importar que aún existan mas soluciones. La sintaxis para corte es: !.

Al ejecutar el ejemplo 2 si se cambia fail, por !, el motor de inferencia instanciará la variable Quien, Quien2 y Quien3 solo en el primer instanciamiento .

4.3 BUSQUEDAS RELACIONALES.

Para ello es necesario utilizar los operadores relacionales:

Operador	Significado
<	Menor que
>	Mayor que
=	Igual a
<>	Diferente a
<=	Menor igual que
>=	Mayor igual que

En el siguiente ejemplo se muestra su aplicación comparando si persona es hombre o mujer para determinar si es padre o madre de alguien.

/* Ejemplo 3. Localiza los lazos familiares */

```
domains
  persona=symbol
predicates
  padre(persona,persona)
  hombre(persona)
  mujer(persona)
  hermanos(persona,persona)
  tio(persona,persona)
  madre(persona,persona)
clauses
  padre(adan,cain).
  padre(adan,abel).
  madre(eva,cain).
  madre(eva,abel).
  hombre(adan).
  hombre(cain).
  hombre(abel).
  mujer(eva).
  madre(M,H):-
    padre(M,H),mujer(M).
  hermanos(S,Y):-
```

```

    padre(M,S),hombre(M),padre(M,Y),S<>Y.
tio(X,S):-
    hermanos(X,Y),padre(Y,S).

```

Se recomienda Ejecutar el programa preguntando al motor de inferencia por padre(Quien, De_quien), y Así sucesivamente por los demás lazos familiares.

4.4 CREANDO UNA ESTRUCTURA TIPO "IF .. THEN"

Se logra mediante el uso de operadores lógicos tales como: and, or, not e if; observe su aplicación en el siguiente ejemplo.

/* Programa 4. Buscando la pareja adecuada */

```

domains
    persona = symbol

predicates
    varon(persona)
    no_varon(persona)
    fumador(persona)
    vegetariano(persona)
    deportista(persona)
    sofia_prefiere_dato(persona)
    rosa_prefiere_dato(persona)
    anselmo_prefiere_dato(persona)
    esther_prefiere_dato(persona)
goal
    makewindow(1,2,0,"",0,0,25,80),
    clearwindow,
    makewindow(2,2,5,"BUSQUEDA DE PAREJA ADECUADA",5,5,18,70),
    rosa_prefiere_dato(Y) and
    write("Un pareja posible para rosa es ",Y) and nl,
    anselmo_prefiere_dato(Z) and
    write("Un pareja posible para anselmo es ",Z) and nl,
    sofia_prefiere_dato(X) and
    write("Una pareja posible para sofia es ",X), nl,
    esther_prefiere_dato(A) and
    write("Una pareja posible para esther es ",A), nl.

clauses
    varon(raul).
    varon(jose).
    varon(memo).

```

```

varon(tomas).
varon(enrique).
no_varon(sofia).
fumador(sofia).
fumador(roberto).
fumador(tomas).
fumador(enrique).
fumador(raul).

```

```

vegetariano(jose).
vegetariano(tomas).
vegetariano(raul).
deportista(tomas).
deportista(enrique).
deportista(raul).
deportista(sofia).

```

```

sofia_prefiere_dato(X):-
    no_varon(X) and fumador(X) and vegetariano(X).
anselmo_prefiere_dato(Z):-
    varon(Z) and fumador(Z).
rosa_prefiere_dato(Y):-
    varon(Y) and vegetariano(Y) and deportista(Y).
esther_prefiere_dato(A):-
    varon(A),deportista(A),fumador(A).

```

Es posible que los resultados de la ejecución del programa no sean siempre muy satisfactorios, pero se pueden manipular si se escoge adecuadamente el o los operadores lógicos de acuerdo al patron de búsqueda deseado.

EJERCICIOS PROPUESTOS

1. Elabore un programa que permita un despliegue por pantalla formateando todos los tipos de dominios manejados por prolog (utilice el predicado writef().
2. Construya una base de hechos con el árbol genealógico de su familia
3. Construye una base de hechos con los tipos de animales según su clasificación en el reino animal.
4. Para los programas 2 y 3 defina una regla para desplegar el contenido de la base de hechos de acuerdo a un patrón de búsqueda y utilizando fail y !
5. Elabore un programa en el que despliegue en 4 ventanas con distintos atributos cada una, y una para cada tipo de animal que haya definido en el programa 3.
6. Capture el programa 4 de este libro y explique su ejecución.
7. Elabore un programa en Prolog en el cual defina objetos que sean instanciados por el usuario a través de peticiones utilizando el predicado read acorde al tipo de datos correspondiente.
8. Elabore un programa para redefinir los atributos de las ventanas del programa 5.
9. Elabore un programa en el cual defina una serie de reglas para las condiciones meteorológicas sol lluvia y nublado y sus respectivas alternativas en el supuesto para cada una de ellas.

10. Elabore un programa en el que se realice una búsqueda condicionada, por ejemplo en la base de hechos defina personas y sus respectivas edades y el usuario solicite al motor de inferencia la presentación de los que tengan cierta edad y otro atributo, el cual se encuentre en una regla.

CAPITULO 5 PREDICADOS ESTANDARES

5.1 PREDICADOS DE ESCRITURA

Write(). Permite desplegar una cadena de caracteres o el valor de una variable. Ejemplo:
write(" esto es una cadena") o Write(Cosa). En el segundo caso podemos definir a cosa como un argumento el cual no está delimitado en tamaño ni en numero de argumentos, de tal forma que podemos tener el siguiente arreglo: write(cosa1,cosa2,cosa3,...,cosaN) o bien write("el contenido de la variable cosa es: ",cosa1).

Tabla 1.

Caracteres de control:	
\	indica que el siguiente carácter es un carácter de control
\n	carácter de control para insertar una nueva línea
\t	carácter de control para un tab
\b	carácter de control para un espacio en blanco
\\	para escribir un slash al revés en la pagina

nl. Inserta una vuelta de carro inmediatamente después de un despliegue, posicionando al cursor al principio de la nueva línea. nl significa nueva línea. Ejemplo write("esto es una cadena"),nl.

Writef(Formato_de_cadena,A1,A2,A3..AN). Despliega una salida formateada su uso en términos generales es similar al predicado write. Los argumentos pueden ser cadenas de caracteres o variables instanciadas o abiertas. Los argumentos son los siguientes:

Formato	Descripción
---------	-------------

%	Posición en la cadena donde el valor será desplegado
-	Indica que el valor será desplegado con justificación hacia la izquierda, si no se indica entonces se justifica hacia la derecha.
m	Define la anchura mínima del campo a desplegar.
.p	Indica el numero mínimo de lugares después del punto decimal con el fin de definir en forma precisa las fracciones decimales en una cantidad. En este campo se pueden agregar cualquiera de las siguientes letras:

valor especificado	Descripción
f	Despliega números reales en notación decimal fija.
e	Despliega números reales en notación exponencial
g	Despliega números reales en la notación mas corta.
d	Caracteres o enteros como un numero decimal.
u	Caracteres o enteros como valor absoluto.
x	Caracteres o enteros como un numero hexadecimal.
c	Caracteres o enteros como un carácter. Como string o symbol.
R	Como referencia a un numero de una base de datos dinámica.
X	Como cadena o referencia a un numero hexadecimal largo.

/*Ejemplo 5. Uso del predicado writef()

```
goal
W=hola,
X=1.23e-3,
Y=3.141,
Z=129,
writef("Despliega un simbolo:%, \n y ahora numeros=\n%f\n%8.2\n %\n",W,X,Y,Z),
write("dame tu edad "),
readreal(Edad),
write("ahora tu sueldo"),
readreal(Sueldo),
writef("esta es tu edad:%fg, \n y tu sueldo=%3.2f \n ",Edad,Sueldo).
```

En la primer ocurrencia de writef(), se instancian valores a tres variables las cuales deberán estar fuera del ámbito de la cadena, es decir, se cierra la cadena y posteriormente se incluyen las variables, es muy importante el orden de las variables pues de lo contrario los valores no corresponderían con las variables.

Writedevice(nombre_de_archivo_simbolico). Indica al predicado write hacia que dispositivo enviará un despliegue de información.

5.2 PREDICADOS DE LECTURA.

Este tipo de predicados marcan diferencias debido al tipo de datos a introducir, sin embargo, cada dato a leer deberá instanciarse a una variable. Tome en consideración la tabla 11.

Tabla 2. Predicados de lectura.

Readint(variableentera). Lee un entero la variable deberá estar libre, y declarada como dominio integer.
Readreal(variablereal). Lee un real la variable deberá estar libre., y declarada como dominio real
Readchar(variablecarácter). Lee un carácter la variable deberá estar libre, y declarada como dominio char.
Readln(cadena). Lee una cadena, la variable deberá estar libre y declarada como dominio string.
Readterm(dominiotermino). Lee un objeto escrito mediante el predicado write, la variable termino será vinculada al objeto que se lea si dicho objeto está dentro de la sección de dominios.
Readdevice(nombre_de_archivo_simbolico)
Un ejemplo del uso de algunos de estos predicados lo tenemos en el ejemplo anterior.

5.3 PREDICADOS DE MANEJO DE VENTANAS.

Para crear una ventana se utiliza el predicado **Makewindow()**, con sus argumentos adecuados por ejemplo:

Makewindow(1,7,112," ventana numero 1",5,20,20,28).

Contiene los siguientes argumentos según su orden de ocurrencia:

Numero de ventana 1. Nos indica que se trata de la ventana activa numero 1.

Atributo de pantalla número 7. Establece que el área principal de la ventana deben ser caracteres en blanco sobre un fondo negro, para el caso en que se utiliza un video monocromo. Considere las 3, 4 y 5.

Tabla 3. Atributos de video monocromo.

Estilo	valor	escritura	fondo
En blanco	0	negro	negro
Normal	7	blanco	negro
video inverso	112	negro	blanco
Opciones:			
1. Subrayar caracteres en el color de escritura: añadir 1.			
2. Mostrar la parte blanca en alta intensidad: añadir 8.			
3. Parpadear carácter: añadir 128.			

Tabla 4. Atributos de video en color.

Color	Escritura	Fondo
Negro	0	0
Azul	1	16
Verde	2	32

Celeste	3	48
Rojo	4	64
Margenta	5	80
Marron	6	96
Blanco	7	112
Gris	8	na
Azul claro	9	na
Verde claro	10	na
Celeste claro	11	na
Rosa	12	na
Margenta claro	13	na
Amarillo	14	na
Blanco claro	15	na

Atributo de cuadro número 112. Indica que video inverso (caracteres negros sobre fondo blanco) bordearán la ventana.

Cabecera "ventana número 1". Es el titulo de la ventana si no desea incluirlo solo indique sus separadores así "".

Renglón, columna. Los valores 5, 20 especifican que la ventana comenzará en el renglón 5 columna 20.

Altura, anchura. Los valores 20, 28 indican que la ventana tendría 20 renglones de alta y 28 columnas de ancha.

Removewindow. Se utiliza para suprimir la ventana activa.

Shiftwindow(). Este predicado permite cambiar la ventana activa siendo el argumento indicado el Numero de ventana hacia la cual queremos mover la ventana activa.

Clearwindow. Se utiliza para limpiar la ventana entera de caracteres.

Cursor(renglón,columna). Permite posicionar el cursor en las coordenadas de la ventana actual.

Tabla 5. Predicados para manejo de ventanas.

<p>Scr_char(renglón,columna,caracter). Escribe el carácter con el atributo actual en la posición especificada.</p> <p>Scr_attr(renglón,columna,atributo). Pone los atributos de la posición especificada por los valores enteros de renglón, columna.</p> <p>Field_str(renglón,columna,longitud,cadena). Permite que un valor instanciado a cadena se despliegue en las coordenadas asignadas y con la longitud definida.</p> <p>Field_attr(renglón,columna,longitud,atributo). Permite poner o leer el atributo de un campo entero de una vez. Especifica también el campo mediante los argumentos de renglón, columna.</p>
--

/*Ejemplo 6. Manejo de ventanas*/

```

goal
makewindow(1,2,3,"",0,0,25,80),
write("pulsa una tecla..."),
readchar(_),
makewindow(2,2,3,"ventana 1",5,5,18,20),
write("pulsa una tecla..."),
readchar(_),
makewindow(3,2,3,"ventana 2",5,7,18,25),
write("pulsa una tecla..."),
readchar(_),
makewindow(4,2,3,"ventana 3",5,9,18,28),
write("pulsa una tecla..."),
readchar(_),
shiftwindow(1),
write("regresamos a la primer ventana").

```

El ejemplo anterior permite al usuario un desplazamiento desde la primer ventana hasta la ultima y regresando nuevamente a la primer ventana mediante el predicado shiftwindow.

/*Ejemplo 7. Atributos de ventanas*/

```

domains
cosa,lugar=symbol
predicates
e_s
arbol(cosa)
goal
e_s.
clauses
e_s:-
makewindow(1,7,112,"ejemplo de atributos de ventana",2,10,22,50),
clearwindow,
write("pulse una tecla"),
arbol(X),
field_attr(4,6,10,1),
readln(_),
field_str(4,6,10,X).
arbol(chile).
arbol(pino).
arbol(naranja).

```

5.4 RECURSIVIDAD.

En Prolog no existen las sentencias de control o los bucles, por tal motivo es diferente la forma en la cual se desarrolla una rutina de tipo recursiva es decir, que se ejecute una determinada cantidad de veces o hasta que se cumpla una determinada condición; sin embargo cuando se presenten las siguientes condiciones será muy necesario definir una regla de tipo recursiva.

- a) Cuando las relaciones son descritas en función de las mismas relaciones.
- b) Cuando los objetos Compuestos (siguiente capitulo) son partes de otros objetos compuestos.
- c) Cuando se define un patrón de búsqueda que debe satisfacer varias condiciones antes de concluir.

d) Cuando se implementa una regla tipo menú en el cual es necesario definir un esquema de tipo CASE.

Por ejemplo cuando existe una rutina para proporcionar una clave de acceso y se deben satisfacer varias condiciones antes de ingresar al sistema como en el ejemplo siguiente:

/*ejemplo 8. Rutina recursiva para ingresar al sistema mediante un password */

```
domains
    nombre,password=symbol
    predicates
    valido(nombre,password)
    login
    usuario(nombre,password)
    repite

goal
    makewindow(1,1,"",0,0,24,79),
    login.

clauses
    repite.
    repite:-
        repite.
    login:-
        clearwindow,
        valido(_,_),
        write("Acceso permitido"),nl.
    login:-
        repite,
        write("Ud. no tiene acceso"),nl,
        write("Intente otra vez"),nl,
        valido(_,_),
        write("Acceso permitido"),nl.
    valido(Nombre>Password):-
        write("Teclee su Nombre: "),
        readln(Nombre),nl,
        write("Teclee su Password: "),nl,
        readln>Password),nl,
        usuario(Nombre>Password).
    usuario(juan,abc).
    usuario(adan,bca).
    usuario(pedro,cba).
```

Este programa requiere que cada usuario proporcione un nombre y password o clave de acceso, los cuales se compararán en la base de hechos en el predicado usuario(), y si no se encuentran, el programa seguirá haciendo peticiones al usuario en forma recursiva hasta que los datos introducidos correspondan con los objetos definidos en el programa.

CAPITULO 6 OBJETOS COMPUESTOS

Así se le llama a un objeto que al mismo tiempo es predicado de otro objeto y su forma sintáctica es: **predicado(objeto,functor(objeto))**, o también , **predicado(objeto,objeto(functor(objeto,objeto)))**. Observe que el paréntesis inicial y los subsecuentes se cierran al final del objeto. Cuando se trabaja con un objeto compuesto el objeto compuesto recibe el nombre de **functor** y sus argumentos reciben el nombre de **componentes**.

Ejecute el ejemplo 9 y tome nota del resultado.

/ Ejemplo 9. Objetos compuestos */*

```
domains
  objetos = libro(nombre,autor); casa(); coche(modelo)
  pedro,nombre,autor=symbol
  modelo=integer
predicates
  tiene(pedro,objetos)
clauses
  tiene(pedro,libro("aplique turbo Prolog","Phillip Robinson")).
  tiene(pedro,casa).
  tiene(pedro,coche(89)).
```

Al solicitar al motor de ingerencia el siguiente objetivo:

```
tiene(pedro,Algo), al motor de inferencia contestará:
Algo=libro("aplique turbo Prolog","Phillip Robinson")
Algo=casa
Algo=coche(89)
3 solutions
```

Ejecute los siguientes dos ejemplos y tome en cuenta los resultados:

```
/* Ejemplo 10. Manejo de objetos compuestos */
```

```
domains
  persona      = persona(nombre,direccion)
  nombre       = nombre(primer,ultimo)
  direccion    = direccion(calle,ciudad,estado)
  calle        = calle(numero,nom_calle)
  ciudad,estado,nom_calle = string
  primero,ultimo = string
  numero      = integer

goal
  A = persona(nombre(juan,perez),direccion(calle(morelos,"5 "),estado,"JAL")),
  B = persona(nombre(_,perez),Direccion),
  C = persona(nombre(ana,diaz),Direccion),
  write(C).
```

En éste ejemplo el objeto compuesto persona contiene al functor direccion el cual contiene a los objetos calle, ciudad y estado, aunque es posible restringir los elementos del objeto, al definir tres variables A, B, y C con distintos argumentos.

```
/* Ejemplo 11. Se adicionan elementos por programa al objeto compuesto */
```

```
domains
  persona=pers(nom,edad,tel,trabajo)
  tel,edad=integer
  nom,trabajo=string

predicates
  leerpersona(persona)
  run

goal
  run.

clauses

leerpersona(pers(juan,25,22734,obrero)):- !. /*se adicionan elementos al objeto*/
run:-
  leerpersona(P),nl,
  cursor(10,8),
  write(P),nl,nl,
  cursor(11,8),
  write("Ejemplo de objeto compuesto"),
  readchar(_).
run:-
  cursor(11,8),
  write("Intente otra vez"),
```

```
readchar(_),
run.
```

El ejemplo 11, define el objeto compuesto en la regla leerpersona, en la que son introducidos los datos del objeto y posteriormente lo que instancia es el objeto pers en la variable P, a la cual se despliega su contenido con el predicado write(P).

/* Ejemplo 12. Objetos compuestos */

```
domains
  restaurant=comida(titulo,precio);
               ingredientes(nombre,cantidad);
               telefono(titulo,num_cuenta)
  persona,titulo,nombre=symbol
  precio=real
  cantidad,num_cuenta=integer

predicates
  compra(persona,restaurant)

clauses
  compra(juan,comida(chiles_rellenos,35.00)).
  compra(juan,ingredientes(chiles,2)).
  compra(juan,telefono("1800-llama",3)).
```

En este ejemplo si se plantea al motor de inferencia el siguiente objetivo: compra(Quien,Que), éste contestará:

```
Quien = juan, Que = comida("chiles_rellenos",35)
Quien = juan, Que = ingredientes("chiles",2)
Quien = juan, Que = telefono("1800-llama",3)
```

Si se utilizan variables anonimas con los siguientes objetivos, compra(Quien,_):

```
Quien = juan
Quien = juan
Quien = juan

Y Compra(_,Que)

Que = comida("chiles_rellenos",35)
Que = ingredientes("chiles",2)
Que = telefono("1800-llama",3).
```

EJERCICIOS PROPUESTOS:

1. A partir de las siguientes peticiones al motor de inferencia elabore el programa correspondiente:
Jefe(Quien,De_quien)
Muestra(Anciano)
Muestra(Puesto,Empleado)
Se recomienda crear una base de hechos con personas y edades, pero deberá crear una regla para definir con operadores condicionales para definir a un anciano (utópicamente) como aquel mayor de 65 años.
2. A partir de la siguiente declaración:
Domains
 Titulo, autor = symbol
 Paginas = integer
Complete el programa indicando al motor de inferencia que en tres ventanas diferentes despliegue los elementos de las base de hechos clasificados por a) titulo, paginas y autor; b) paginas, autor, titulo; c) autor, paginas, titulo.

Se recomienda utilizar el predicado writef() y los predicados de manejo de ventanas: makewindow(), clearwindow y shiftwindow().
3. Tomando en cuenta la siguiente declaración de dominio complete el programa definiendo un Functor:
Domains
 Articulos = libro(titulo,autor); caballo(nombre); bote; banamex(cantidad)
 Titulo, autor, nombre = symbol
 Cantidad = real
4. Elabore un programa en Prolog que muestre un functor sobre los elementos de que consta una bicicleta.
5. Elabore un programa en Prolog para seleccionar en una base de hechos que contenga nombres de personas, domicilios y raza solo aquellos que reúnan ciertas características (búsqueda por comparación), utilice operadores relacionales.
6. Elabore un programa en prolog que muestre un objeto compuesto que contenga las partes del cuerpo humano.
7. Elabore un programa en prolog que muestre un objeto compuesto que contenga las partes de un coche.
8. Elabore un programa en prolog que muestre un objeto compuesto que contenga distintos tipos de comida con sus respectivos ingredientes.
9. Elabore un programa en prolog que muestre un objeto compuesto que contenga distintas razas de animales con sus caracterisiticas mas relevantes.

CAPITULO

7

LISTAS

Una lista es la estructura siguiente al manejo de objetos compuestos y en prolog tiene una gran importancia porque nos permite manipular arreglos complejos.

Básicamente una lista es una colección de elementos ordenados separados por comas y colocados entre corchetes en la siguiente forma: [silla,casa,perro,ventilador], los tipos de elementos se determinan por el tipo de dominio asignado a los mismos como se tiene en los siguientes ejemplos:

[1,2,3,4] lista de enteros
[1.1,3.4,5.5] lista de reales
[l,i,s,t,a] lista de caracteres
[perro,silla,casa,computadora] lista de símbolos
["perro","silla","casa","computadora"] lista de cadenas
[] lista vacía

una lista se declara en la sección de dominios de la siguiente forma:

```
domains
    lista_de=cosas*
    cosas=symbol
```

Donde el asterisco significa "lista de".

Una lista se compone de por una cabeza y una cola separados por el símbolo de teclado "|", la cabeza es el miembro del extremo izquierdo y la cola es el resto de los elementos tal como se muestra a continuación: predicado([cabeza|cola]), debido a que las listas son otro tipo de objetos deben ir encerradas entre paréntesis como cualquier otro objeto.

Cuando se tiene una lista con un solo elemento tal como: lista([a]) la cabeza es a, y la cola es [], es decir la lista vacía siempre será la cola en estos casos.

Puesto que una lista es una secuencia de elementos ordenados, si se modifica el orden de una lista, esto la convierte automáticamente en otra lista por ejemplo:

[silla,perro,casa,computadora] es diferente a: [perro,silla,casa,computadora], porque los dos primeros elementos de la misma tienen distinto orden.

7.1 TRABAJANDO CON LISTAS.

/*Ejemplo 13. Muestra los elementos de la lista */

```
domains
    lista=integer*
predicates
    listaenteros(lista)
goal
    listaenteros(L),
    write("los elementos de la lista son: ",L).
clauses
```



```
listaenteros([1,2,3,4,5,6,7]):- !.
```

Los elementos de listaenteros son instanciados a la variable L y se despliegan con write(L).

/Ejemplo 14. Manipulando cabeza y cola*/

```
domains
  mamifero=symbol*
  animal=symbol
predicates
  animales(mamifero)
goal
  animales([Cabeza,Cola|Cola2]),
  write(Cabeza,Cola,Cola2).
clauses
  animales([hombre,mujer,nino,nina]).
```

En el ejemplo anterior si mueve el carácter de barra vertical en la sección de objetivos, moverá también la cabeza y la cola de la lista. Una vez que se definen los elementos de cabeza y cola, se instancian a sus respectivas variables y se despliega su contenido mediante el predicado write(Cabeza,Cola,Cola2).

/*Ejemplo 15. Realizando búsquedas en la lista */

```
domains
  lista = nombre* /*lista de nombres*/
  nombre = symbol

predicates
  miembro_de(nombre,lista)

clauses
  miembro_de(Nombre, [Nombre|_]). /*se instancia Nombre con elemento introducido por teclado y se*/
  /*inicializa cola con variable anónima*/
  miembro_de(Nombre, [_|Cola]) :-
    miembro_de(Nombre,Cola).
```

En la sección de objetivos ejecute el siguiente objetivo: miembro(juan,[maria,pedro,diana,juan]), con ello forzará al motor de inferencia a localizar a juan en la lista, si lo encuentra contestará: yes.

Ejecute también miembro(X,[juan,diana,pedro]) y observe que el motor desplegará los elementos de la lista, instanciados a la variable X.

7.2 UNIENDO UN FUNCTOR CON UNA LISTA

Es posible unir un functor con una lista y efectuar búsquedas selectivas, tal como se muestra en el siguiente ejemplo:

/*Ejemplo 16. Encontrando los elementos de una lista con un atributo que define un patrón de búsqueda.

Domains

```

Usuario=nombre*

Clave=integer
Nombre=n(primer,ultimo)
Primer,ultimo=symbol
Predicates
    Usuario(nombre,clave)

Clauses
    Usuario(n,(garcia,carlos),23),
    Usuario(n,(olmos,pedro),23),
    Usuario(n,(lomeli,laura),28),
    Usuario(n,(diaz,joaquin),33),
    Usuario(n,(hernandez,rosa),28),
    Usuario(n,(rodriguez,adriana),23),
Goal
    Findall(X,usuario(X,23),Usuario),
    Write(X,Usuario).

```

En el ejemplo anterior se utiliza el predicado findall, con el fin de vincular a la variable X, los elementos de la lista que reúnan el patrón de búsqueda, en este caso a todos los usuarios que tengan la clave 23.

7.3 UNIENDO LISTAS.

Es posible tener varias listas unidas en una sola, siempre y cuando se tome en consideración los siguientes requisitos:

1. Una lista unida es un predicado con varias listas relacionadas entre si:
predicado([lista1],[lista2],[lista3])
2. Para unir varias listas se debe unir lista1 con lista2 en lista3 de la siguiente forma:
unir([lista1],[lista2],[lista3])

Esto le indica al motor de inferencia que una lista1 y lista2 en lista3. Ejecute el ejemplo 16.

/*Ejemplo 17. Uniendo listas */

```

domains
    lista=symbol*
predicates
    junta(lista,lista,lista)
clauses
    junta([],Lista,Lista).
    junta([Cabeza|Cola_vieja],Lista,[Cabeza|Cola_nueva]):-
        junta(Cola_vieja,Lista,Cola_nueva).

```

Con el siguiente objetivo se ordenará al motor de inferencia que una dos listas en la tercera utilizando el predicado: junta([casa,muebles],[terreno,edificio], L3), El motor contestará: "casa", "muebles", "terreno", "edificio".

7.4 RECURSION.

Es posible definir un esquema recursivo haciendo que una regla realice llamadas a sí misma tal como se muestra en el siguiente ejemplo:

/ Ejemplo 18. Llamada recursiva */*

```
domains
    lista=integer*
predicates
    despliegalista(lista)
goal
    despliegalista([1,2,3,4,5,6]).
clauses
    despliegalista([]). /*inicializa la lista */
    despliegalista([Cabeza|Cola]):-
        write(Cabeza),nl, /*instancia en la variable cabeza el primer
                                elemento de la lista*/
        despliegalista(Cola). /*instancia en la variable Cola todos los
                                elementos después de la cabeza/
```

CAPITULO 8 ARCHIVOS

Son entidades lógicas en la programación de computadoras que se utilizan para coleccionar datos relacionados, para ubicarlos en un dispositivo de almacenamiento secundario.

8.1 DISPOSITIVOS ESTANDARES PREDEFINIDOS.

Son dispositivos que involucran entradas o salidas al sistema que no necesitan ser declarados, simplemente se usan redireccionando su salida o su entrada pues los dispositivos estándar son la pantalla y el teclado, para redireccionar es necesario utilizar el predicado `writedevice(nombre_de_archivosimbolico)` o `readdevice(nombre_de_archivosimbolico)`. Consulte la tabla 4.

Tabla 4.

Dispositivo predefinido	dispositivo de salida
Printer	puerto de impresión
Screen	monitor
Keyboard	teclado
Comx	el puerto de comunicaciones x

8.2 DECLARACION DE ARCHIVOS.

Pueden tener cualquier nombre siempre y cuando no sea el de alguna palabra reservada. La declaración en la sección de dominios será:

Domains

File=archivo1,archivo2,archivo3

Donde archivo1, archivo2 y archivo3, son archivos de memoria o simbólicos que deben ser instanciados a un archivo de disco.

8.3 PREDICADOS DE MANEJO DE ARCHIVOS.

Existfile(nombrearchivo_dos). Requiere una variable instanciada o vinculada. Comprueba si un archivo con el nombre del valor de la variable esta en el disco.

Deletefile(nombrearchivo_dos). Requiere una variable instanciada. Suprimirá el archivo que tiene el nombre del valor de la variable.

Renamefile(nombrearchivo_dos_antiguo, nombrearchivo_dos_nuevo). Este predicado requiere dos variables instanciadas. Encontrará el archivo con el valor de la segunda variable.

Eof. Se cumple si el carácter de la posición actual del archivo en el "nombre_de_archivosimbolico" es ASCII 26 (ctrl-z). Este código indica el final de un archivo.

Closefile(nombre_archivo_simbolico). Cierra el archivo con el nombre "nombre_archivo_simbolico". Si este archivo no está abierto el predicado se cumple de cualquier forma. Este predicado se utiliza para asegurar que cualquier archivo sobre el que se haya escrito o haya sido Modificado esté completo y así no se perderá ningún dato.

Filestr(nombre_archivo_dos,variable_cadena). Encuentra el archivo especificado por "nom_archivo_dos" y lee desde él caracteres hasta un límite de 64 k o hasta que se encuentre el carácter de fin de archivo, los caracteres son entonces asignados a la cadena "variablecadena".

8.4 APERTURA DE ARCHIVOS.

Openwrite(nombre_archivo_simbolico,nombre_archivo_dos). Se utiliza para abrir un archivo para escritura (enviar información a un archivo de disco), si ya hay un archivo con ese nombre, será actualizado con la nueva información.

Openread(nombre_archivo_simbolico,nombre_archivo_dos). Se utiliza para leer un archivo para lectura (enviar y recibir información de un archivo de disco).

Openappend(nombre_archivo_simbolico,nombre_archivo_dos). Se utiliza para abrir un archivo al que se quiere añadir información.

Openmodify(nombre_archivo_simbolico,nombre_archivo_dos). Se utiliza para abrir un archivo para escritura y lectura

8.5 POSICIONAMIENTO EN UN ARCHIVO.

Filepos(nombre_archivo_simbolico,posición,archivo,modo). Encuentra la posición del archivo y actualiza la posición del archivo, este predicado requiere un argumento que indica el modo. Para encontrar la posición del archivo "modo", debe ser instanciado a cero, "archivo_simbolico" debe ser instanciado y "posicion_archivo_simbolico" debe estar libre.

Para encontrar la posición del archivo, las tres variables deben estar instanciadas, "archivo_simbolico" dice sobre qué archivo se trabajará, "posicion_archivo" da un valor numérico para la posición y el "modo" dice a Prolog cómo debe interpretar el "numero_posicion_arch". El número puede contar desde el comienzo del archivo, desde la posición actual o hacia atrás desde el final del archivo.

MODO DESDE DONDE CONTAR POSICION DEL ARCHIVO

0	Comienzo del archivo
1	Posición actual en el archivo
2	Final del archivo moviéndose hacia atrás

/*Ejemplo 19. Escritura y lectura en un archivo*/

```
domains
file=archivo /*declaración de archivo simbolico*/
character=char
predicates
abrir  /* modulo de escritura */
leer   /* modulo de lectura */
lee_posiciones /* lee las posiciones en el archivo y despliega su
              contenido */
fin
clauses
abrir:-
    makewindow(1,0,0,"",0,0,25,80),
    clearwindow,
    makewindow(2,2,3,"escritura a un archivo",0,0,10,80),
    cursor(5,5),
    write("abre el archivo y graba una cadena de caracteres"),
    nl,
    openwrite(archivo,"c:\\prolog\\datos.dba"), /* instancia el archivo simbolico con el archivo de datos de
disco */
    writedevic(archivo), /*define el dispositivo de salida */
    write("esta es una prueba de escritura en datos.dba"),nl, /*información a grabar en archivo */
    closefile(archivo), /*cerrando el archivo simbólico */
    writedevic(screen),nl, /* redefine el dispositivo de salida */
    write("proceso de escritura terminado"),
    write("pulse una tecla"),
    readchar(_),
    removewindow.
leer:-
    makewindow(3,2,0,"",0,0,25,80),
    clearwindow,
    makewindow(2,2,3,"lectura de un archivo",0,0,10,80),
    cursor(5,5),
    write("abre el archivo y lee su contenido"),
    nl,
    openread(archivo,"c:\\prolog\\datos.dba"), /* abre el archivo simbolico instanciado al archivo de datos para su
lectura */
    lee_posiciones.
lee_posiciones:-
    readdevice(keyboard),nl,write("posicion no.? "), /* define el dispositivo de lectura*/
    readreal(X),
    readdevice(archivo), /*redefine el dispositivo de de lectura hacia el archivo simbólico */
    filepos(archivo,X,0),readchar(Y), /*lee la posición del archivo y */
    write(Y),lee_posiciones. /* despliega el contenido de la posición */
fin:-
    cursor(7,7),
    write("pulse una tecla para terminar"),nl,
    readchar(_).
```

CAPITULO

9

GRAFICOS Y SONIDO

9.1 GRAFICOS.

Prolog tiene un entorno gráfico muy potente y se divide de la siguiente forma:

1. Gráficos estáticos o también llamados, de puntos y líneas.
2. Gráficos de tortuga, también llamados gráficos dinámicos.
3. La caja de herramientas.

9.1.1 GRÁFICOS ESTÁTICOS.

Para iniciar el trabajo en el entorno gráfico es necesario cambiar del entorno por defecto, es decir del modo texto al modo gráfico, para ello se utiliza el predicado: `graphics(parammodo,paleta,fondo)`, este predicado limpia la pantalla, coloca el cursor en la esquina superior izquierda, pone la resolución, elige los colores disponibles y pone el color de fondo, dejando habilitado el modo gráfico.

Los predicados estándares para el entorno gráfico son:

```
Graphics(parammodo,paleta,color)
Dot(renglon,columna,color)
Line(renglon1,columna1,renglon2,columna2,color)
text
```

Para metros del predicado `graphics`

Parammodo

Valor	descripción	columnas	filas	Num. De color	Video
1	Medio	320	200	4	Cga
2	Alto	640	200	B/n	Cga
3	Medio	320	200	16	Ega
4	Alto	640	200	16	Ega
5	Aumentado	640	350	13	Ega

Paleta de resolución

valor	Color1	Color2	Color3
0	Verde	Rojo	Amarillo
1	celeste	Margenta	Blanco

Fondo

Valor	Color
0	Negro
1	Azul
2	Verde
3	Celeste
4	Rojo
5	Margenta
6	Marrón
7	Blanco
8	Gris
9	Azul claro
10	Verde claro
11	Celeste claro
12	Rojo claro
13	Margenta claro
14	Amarillo
15	Blanco claro

Uso de los predicados.

/* ejemplo 20. Gráfico de líneas y puntos */

```
goal
graphics(3,1,5), /* gráfico EGA aumentado, paleta 1 y margenta */
write("texto en modo grafico"),
line(0,0,31999,31999,3), /* línea descendente */
dot(10000,1000,2), /* punto arriba */
dot(24000,27000,2). /* punto abajo */
```

Como se observa en el ejemplo anterior el predicado write puede utilizarse en modo gráfico, así como los demás predicados de entrada – salida.

9.1.2 GRÁFICOS DE TORTUGA.

La tortuga es una herramienta de dibujo imaginaria sobre la pantalla los predicados indicados en la tabla siguiente le indican si debe moverse o no y si debe dejar un rastro, el sentido en que debe hacerlo.

Por defecto la tortuga inicia en el centro de la pantalla, orientada con la cabeza mirando imaginariamente hacia arriba, a partir de esta orientación se indican por programa los desplazamientos que debe seguir y el rastro a dejar o no sobre la pantalla, es de hacer notar que en caso de salir de las coordenadas de la pantalla no se ocasionará ningún error y es posible que la tortuga reaparezca en otra parte de la pantalla.

Predicados de control de tortuga

Control de la pluma	Valores
Pencolor(color)	Integer
Pendown	Levanta la pluma
Penup	La posiciona
Movimiento	
Forward(paso)	Adelante
Back(paso)	Atrás
Rotación	
Left(angulo)	Izquierda
Right(angulo)	Derecha

/* Ejemplo 21. Tortuga */

```
goal
write("modo texto"),
write("pulsa una tecla"),
readchar(_),
graphics(3,1,5), /*inicializa el entorno gráfico */
pencolor(14), /*define el color de la pluma (o del rastro de la tortuga)*/
pendown, /*baja la pluma*/
forward(10000), /*camina hacia adelante diez mil puntos*/
left(45), /*gira 45 grados hacia la izquierda*/
forward(10000), /*camina hacia adelante diez mil puntos */
line(0,0,31999,31999,3), /*se traza una línea (gráficos estáticos) */
dot(10000,1000,2), /*coloca un punto gráfico */
dot(24000,27000,2), /*coloca un punto gráfico */
write("modo grafico").
```

Al ejemplo anterior se añade un gráfico de tortuga y el resultado es un desplazamiento gráfico.

Por ultimo, si desea restablecer el esquema de trabajo en modo texto, inserte el predicado `text`, justo antes de terminar la ejecución del programa.

9.1.3 LA CAJA DE HERRAMIENTAS.

La caja de herramientas, es el entorno de trabajo en modo gráfico definido por Borland para turbo Prolog, mediante la interfaz gráfica de Borland (BGI), que es una librería de herramientas gráficas que consta de aproximadamente 70 predicados y soporta video monocromo, EGA, HERCULES, CGA y VGA.

A diferencia de los dos entornos gráficos anteriores, en los cuales se utilizó un sistema de coordenadas a efectos de definir posiciones en la pantalla, en la caja de herramientas se utilizan puntos gráficos o píxeles los cuales dependerán del tipo de tarjeta gráfica para indicar la cantidad de puntos disponibles.

Tarjeta	Identificador	Resolución
Video graphics array	VGA	320X200 en baja resolución 640X200 en alta resolución
Color graphics adapter	CGA	320X200 en baja resolución 640X200 en alta resolución
Enhanced graphics adapter	EGA	640X200
Hercules Graphics	Monochrome Adapter	
Multi color graphics adapter	MVGA	

Drivers gráficos.

Turbo prolog necesita una serie de drivers gráficos que deben estar presentes en el subdirectorio actual o en su defecto declarar el directorio en el cual se encuentran disponibles. Los drivers son archivos que tienen extensión BGI, los drivers que soporta turbo Prolog son:

ATT.BGI
CGA.BGI
EGAVGA.BGI
HERC.BGI
IBM8514.BGI
PC3270.BGI

Para inicializar el entorno gráfico el predicado `Initgraph()`, requiere de que se incluya como argumento el driver de la tarjeta gráfica incluida en la computadora en la cual se pretende ejecutar el programa, sin embargo y dado que uno de los requisitos que debe tener todo programa es la portabilidad, es decir, que pueda ser ejecutado en la mayor cantidad de computadoras posibles, el predicado `detectgraph()` se encarga de detectar el tipo de tarjeta disponible y elige el driver adecuado. La declaración de gráfico es la siguiente: `Initgraph(driver,modografico,nuevodriver,nuevomodografico,pathdriver)`.

En forma adicional turbo prolog dispone del archivo `GRAPDECL.PRO` que incluye declaraciones de constantes para el manejo de la tarjeta gráfica y su modo de despliegue, de tal forma, que solo deberá definirlo al principio del programa como un incluido, declarar el entorno gráfico y por ultimo crear el gráfico.

Predicados de configuración

`Setbkcolor()`. Define el color de fondo.
`Getbkcolor()`. Regresa el color de fondo actual.
`Setpalette()`. Define el color de la paleta.
`Getpalette()`. Regresa el color actual de la paleta.
`Setallpalette()`. Regresa todos los colores de la paleta.
`Setcolor()`. Toma todos los colores de la paleta y permite utilizar uno de ellos.
`Getcolor()`. Regresa el color actual.

Predicados de trabajo

Line(X0Y0,X1,Y1). Dibuja una línea desde las coordenadas iniciales hasta las finales.

Lineto(). Dibuja una línea en las coordenadas X,Y especificadas.

Setlinestyle(). Define el tipo de línea tomando como argumento un valor definido para los siguientes tipos de líneas:

Valor	descripción
0	sólida
1	punteada
2	centrada
3	en guión
4	definida por el usuario

Texto gráfico.

Turbo Prolog incluye librerías de texto ya definidas que pueden manipularse en forma restringida, alargando o recortando su tamaño o modificando la orientación de despliegue tal como, vertical u horizontal.

Font	valor	descripción
Ninguno	0	8X8 font de bit mapeado
Goth.chr	1	gótico artístico
Sans.chr	2	san-serif artistico
Litt.chr	3	small artístico
Trip.chr	4	triple artístico

Predicados para manejo de texto

Setusercharsize(). Modifica el tamaño del texto cuando es bit mapeado

Outtext(). Para insertar texto en la ubicación de pantalla actual.

Outtextxy(). Para insertar texto en un punto de la pantalla definido.

Predicados de diseño gráfico estándar

Con estos predicados se manipulan formas gráficas predefinidas como: rectángulos, arco, elipse, círculo, gráfico de pastel, gráfico de barras en tres dimensiones, etc. ejecute los dos siguientes programas.

```
/* Ejemplo 22. Manejo de gráficos estándares */
```

```
include "C:\\PROLOG\\GRAPDECL.PRO"
```

```
constants
```

```
    bgi_path = "c:\\prolog\\bgi"  
    g_driver = detect  
    g_mode = 0
```

```
predicates
```

```
    puertograf  
    set_color(integer)  
    grafico
```

```
goal
```

```
    puertograf,  
    set_color(_),  
    grafico,  
    closegraph().
```

```
clauses
```

```
puertograf:-
```

```
    initgraph(g_driver, g_mode, _, _, ""),  
    detectgraph(_, G_mode),  
    getMaxX(MaxX), getMaxY(MaxY),  
    setviewport(0, 0, MaxX, MaxY, 1),  
    clearviewport,  
    writef("el puerto por default es %, la resolucion es % por %",  
           G_mode, MaxX, MaxY), nl,  
    readchar(_, nl).
```

```
grafico:-
```

```
    getmaxX(MaxX), getMaxY(MaxY),  
    setviewport(0, 0, MaxX, MaxY, 1),
```

```
/* Dibuja a 90 grados un arco con radio de 50 */
```

```
    arc(150, 150, 0, 89, 50),  
    write("presione una tecla..."),  
    readchar(_),
```

```
/* consigue las coordenadas del arco y conecta los finales */
```

```
    getarcoords(X, Y, Xinicial, Yinicial, Xfinal, Yfinal),
```

```
/*dibuja una línea/
```

```
    line(100, 50, 350, 50),
```

```
/* Dibuja un circulo */
```

```
    circle(150, 150, 100),
```

```
/* Dibuja una elipse dentro del circulo */
```

```
    ellipse(150, 150, 0, 359, 100, 50),
```

```

/* Dibuja y rellena un grafico de pastel */
pieslice(100, 100, 0, 135, 49),
pieslice(100, 100, 135, 225, 49),
pieslice(100, 100, 225, 360, 49),

/* TRAZA UN GRAFICO DE BARRAS */
bar3d(300,300,400,450,50,50),

readchar(_),
closegraph.

set_color(1):-
    !, setbkcolor(3). % Set background color if CGA
set_color(_):-
    setcolor(3).

/* Ejemplo 23. manejo de texto grafico */

include "C:\\PROLOG\\GRAPDECL.PRO"

/*constants
    bgi_path = "c:\\prolog\\bgi" */

predicates
    set_color(integer)

goal
    initgraph(detect, 0, GraphDriver,GraphMode,""),
    set_color(GraphDriver),

    Titulo="ejemplo de tipos de letras",
    settxtjustify(center_text,center_text),
    setusercharsize(1,1,1,1),
    settxtstyle(triplex_font,vert_dir,user_char_size),
    textWidth(Titulo,Width),textHeight(Titulo,Height),
    setusercharsize(200,Width,100,Height),
    settxtstyle(triplex_font,vert_dir,user_char_size),

    outtextxy(300,88,Titulo),
    setusercharsize(300,Width,100,Height),
    settxtstyle(gothic_Font,hORIZ_Dir,user_char_size),

/* Dibuja un circulo */
circle(150, 150, 100),

/* Dibuja una elipse dentro del circulo */
ellipse(150, 150, 0, 359, 100, 50),

/* /* Dibuja y rellena un grafico de pastel */
pieslice(100, 100, 0, 135, 49),
pieslice(100, 100, 135, 225, 49),

```

```

pieslice(100, 100, 225, 360, 49),*/

/* TRAZA UN GRAFICO DE BARRAS */
bar3d(300,300,400,450,50,50),

readchar(_),
closegraph.

clauses
set_color(1):-
    !, setbkcolor(3). % Set background color if CGA
set_color(_):-
    setcolor(3).

/* Dibuja a 90 grados un arco con radio de 50 */
arc(150, 150, 0, 89, 50),

ARGUMENTOS
Coordenadas del centro, angulo inicial, angulo final, radio
/* consigue the coordenadas de el arco y conecta los finales */
/*getarcoords(X,Y,Xinicial,Yinicial,Xfinal,Yfinal),*/

/*dibuja una linea*/
line(100, 50, 350, 50),

ARGUMENTOS
Coordenadas de inicio y fin

/* Dibuja un circulo */
circle(150, 150, 100),

ARGUMENTOS
Coordenadas del centro y el radio
/* Dibuja una elipse dentro del circulo */
ellipse(150, 150, 0, 359, 100, 50),

ARGUMENTOS
Los mismos del arco pero con dos radios, uno para x y otro para y
/* Dibuja y rellena un grafico de pastel */
pieslice(100, 100, 0, 135, 49),

ARGUMENTOS
Mismos argumentos que la elipse

/* TRAZA UN GRAFICO DE BARRAS EN TRES DIMENSIONES */
bar3d(300,300,400,450,50,50),

ARGUMENTOS
Coordenadas de esquina sup. Izquierda, profundidad y bandera indicadora para colocar una linea de tope superior con su respectiva distancia.
/* TRAZA UN GRAFICO DE BARRAS NORMAL */
bar(300,300,400,450,50,50),

```

ARGUMENTOS

Coordenadas de esquina superior izquierda y esquina superior derecha

Predicados para manejo de pixeles e imágenes.

Getpixel

Este predicado regresa el color de un píxel xy. Su sintaxis es:

Getpixel(X,Y,Color)

Los tres argumentos deberán ser de tipo integer.

Putpixel

Traza un píxel en XY en el color pixelcolor

Su sintaxis es:

Putpixel(X,Y,Pixelcolor)

Los tres argumentos deberán ser de tipo integer.

Getimage

Salva una imagen de bit en una determinada region de memoria. Su sintaxis es:

Getimage(Left,Top,Right,Bottom,Bitmap)

Los argumentos deberán ser de tipo integer.

Imagesize

Regresa el numero de bytes requeridos para colocar una region rectangular de la pantalla. Su sintaxis es:

Imagesize((Left,Top,Right,Bottom,Size)

Los argumentos deberán ser de tipo integer.

Putimage

Pone una imagen de bit previamente salvada sobre la pantalla. Su sintaxis es:

Putimage(X,Y,Bitmap,Operación)

Los argumentos deberán ser de tipo integer.

9.2 SONIDOS.

El manejo de sonido en prolog esta restringido al trabajo con el altavoz y la programación de una tarjeta de sonido compatible con Sound Blaster, aunque esto ultimo escapa al contenido del presente trabajo.

Para el manejo de sonido se consideran dos predicados:

BEEP. No requiere de argumentos y únicamente produce un "pitido" con el altavoz.

Sound(duración, frecuencia), permite manipular la duración y la frecuencia del sonido a criterio del usuario. La duración esta dada en centésimas de segundo y la frecuencia en Hertz. Ejecute el siguiente programa.

```
/*Ejemplo 24. Manejo de sonidos */
predicates
ventana
melodia
alarma
parte1(integer)
parte2(integer)
parte3(integer)
bomba
clauses
melodia:-
sound(40,325),sound(40,325),sound(55,325),
sound(10,17900),
sound(40,325),sound(50,325),sound(55,323),
sound(10,17900),
sound(40,323),sound(50,348),sound(55,256),
sound(70,287),sound(70,323).
ventana:-
makewindow(1,2,3,"",0,0,25,80),
clearwindow,
makewindow(2,2,3,"Musica",0,5,10,60),
write("cancion navideña").
alarma:-
sound(60,650),nl,
sound(60,120),
alarma.
```

```
/*ejemplo 25 simula una bomba que cae*/
```

```
/*duración/
parte1(900):-
sound(10,900),!.
parte1(X):-
sound(5,X),
X1=X+20,
parte1(X1).
/*frecuencia*/
parte2(200):-!.
parte2(X):-
sound(5,X),
X1=X-50,
parte2(X1).

/*fin*/

parte3(400):- !.
```



```
parte3(X):-  
  sound(40,X),  
  X1=X-20,  
  parte3(X1).
```

```
bomba:-  
  parte1(100),  
  parte2(10000),  
  parte3(50).
```

CAPITULO

10

ELEMENTOS MATEMATICOS

10.1 PREDICADOS MATEMATICOS.

Los predicados matemáticos utilizados en Prolog utilizan el formato estándar: relación(objeto), (consulte la tabla 6). O bien el usuario puede crear sus propios predicados, pero deberá considerar que Prolog no maneja ciclos, sin embargo puede crear sus propias rutinas recursivas, consulte los siguientes ejemplos para aprender el uso de los predicados matemáticos estándares y de usuario.

Tabla 6.

Abs(X)	vincula X al valor absoluto de la X original
Sqrt(X)	vincula X a la raíz cuadrada X original
Random(X)	vincula X al numero real "aleatorio" tal que $0 \leq X < 1$
Log(X)	vincula X al logaritmo en base 10 la X original
Ln(X)	vincula X al logaritmo en base e elevado a la potencia X original
Exp(X)	vincula X al valor de e elevado a la potencia de la X original
Cos(X)	vincula X al coseno de la X original
Sin(X)	vincula X al seno de la X original
Tan(X)	vincula X a la tangente de la X original
Arctan(X)	vincula X a cotangente de la X original

/* ejemplo 25. Uso de predicados trigonométricos */

```
include "c:\\prolog\\grapdecl.pro /*incluido con librerias para manejo
de gráficos */
constants
bgi_drivers = "\\bgi"

domains
    file = archivo
    Div1,Div2 = real
    Lado1,Lado2,Lado3 = real
    Resultado,Resultado1 = real
    Resultados,Resultado2 = real

predicates
    presenta
    inicializar
    calculo
    dibujo

clauses
    inicializar:-
        detectgraph(Driver,Mode),
        initgraph(Driver,Mode,_,_,bgi_drivers).
    presenta:-
        outtextxy(210,10,"MANEJO DE PREDICADOS TRIGONOMETRICOS"),
        outtextxy(235,30,"LENGUAJE PROLOG").

    dibujo:-
        line(150,150,250,150),
        outtextxy(175,160,"lado 3"),
```

```

line(150,150,150,75),
outtextxy (80,110,"lado 1"),
line(150,75,250,150),
outtextxy(260,110,"lado 2"),
outtextxy(155,88,"B"),
outtextxy(227,140,"G").

```

calculo:-

```

outtextxy (20,180,"escriba el lado 1 del Triangulo"),
cursor(11,40),
readreal(Lado1),
outtextxy (20,215,"escriba el lado 2 del triangulo : "),
cursor(13,40),readreal(Lado2),
outtextxy (20,245,"escriba el lado 3 del triangulo : "),
cursor(15,40),readreal(Lado3),
Resultado = arctan(Lado3/Lado2),
Resultado1 = (180*Resultado)/3.1416,
Resultados = arctan(Lado1/Lado2),
Resultado2 = (180*Resultados)/3.1416,
write(" El angulo B es :",Resultado1),nl,
write (" El angulo G es :",Resultado2),nl,
Tangente = sin(Resultado1)/cos(Resultado1),
Cotangente = cos(Resultado1)/sin(Resultado1),
Secante = 1/cos(Resultado1),
Cosecante = 1/sin(Resultado1),
write (" Pulse cualquier tecla para grabar datos en el archivo"),
readchar(_),nl,
write (" Grabando datos en el archivo calculos.dba"),nl,
openwrite(archivo,"\\calculos.dbf"),
writedevise(archivo),
write("Valor del angulo A :",Resultado1),nl,
write("Valor del angulo B :",Resultado2),nl,
write (" La tangente del angulo B es : ",Tangente),nl,
write (" La cotangente del angulo B es : ",Cotangente),nl,
write (" La secante del angulo B es : ",Secante),nl,
write (" La cosecante del angulo B es : ",Cosecante),nl,
closefile(archivo),
write ("datos grabados").

```

goal

```

inicializar,
presenta,
dibujo,
calculo.

```

/* Ejemplo 26. Acumulador*/

/* el programa incrementa el valor inicial de 1 a 15*/

predicates

for(integer,integer,integer)

vuelta

clauses

for(Num,_,Num).

for(Inicio,Para,Num_final):-

Inicio1=Inicio+1,

```

Inicio1<=Para,
for(Inicio1,Para,Num_final).
vuelta:-
  for(1,15,X),
  write(X),nl,
  fail.
vuelta.
goal
makewindow(1,2,3,"",0,0,20,30),
clearwindow,
vuelta.

```

```

/* Ejemplo. 27. Elevando un numero a una potencia/
/* Ejecute el programa indicando un numero y la potencia a la que desea elevarlo/

```

```

domains
  potencia=real

```

```

predicates
  ventana
  potencia

```

```

clauses
  ventana:-
    makewindow(1,2,0,"",0,0,24,80),
    clearwindow,
    makewindow(2,2,3,"Exponenciacion",5,4,18,60).

```

```

potencia:-
  cursor(2,3), write("Proporcione el numero y la potencia deseada"),
  cursor(5,10), write("Numero: "), readreal(M),
  cursor(6,10), write("Potencia: "),readreal(N),
  cursor(8,10), write("El resultado es: "),Result = Exp(N*ln(M)),
  write (Result),
  cursor(12,1).

```

```

goal
  ventana,
  potencia.

```

```

/*Ejemplo 28. El factorial de un numero/
/*Ejecute el programa indicando el número al que desea determinar su factorial y la variable en la cual se
reflejará el resultado/

```

```

domains
  n,f=real
predicates
  factorial(n,f)
clauses
  factorial(1,1).

```

```
factorial(N,Res):-  
    N>0,  
    N1=N-1,  
    factorial(N1,FactN1),  
    Res=N*FactN1.
```

CAPITULO 11 BASES DE DATOS DINAMICAS

Hay varios tipos de archivos en prolog, él mas simple es la base de hechos en la cual el motor realiza las búsquedas; Los archivos, que se utilizan para aplicaciones estáticas en las que solo se requiere

almacenamiento secundario, actualizaciones, consultas y despliegues, las bases de datos dinámicas, las cuales se explicarán en este capítulo y las bases de datos externas.

Las bases de datos dinámicas son utilizadas para aplicaciones orientadas a la inteligencia artificial en las cuales se crea una base del conocimiento, que puede ser accedida desde el interior del programa fuente, dado que la base de datos allí se encuentra, o bien es posible que se encuentre almacenada en un archivo y el acceso sea desde programa.

El conocimiento colocado en la base de datos consiste en afirmar (assert) o desmentir (retract) un conjunto de hechos que pueden ser accedidos durante la ejecución del programa.

Los objetos de una base de datos son ubicados en forma secuencial como muchas otras cláusulas escritas en cualquier programa de Prolog, sin embargo dado que los datos son colocados en memoria, es necesario hacer un adecuado manejo de la misma, pues de lo contrario se corre el riesgo de saturar la memoria y como consecuencia el programa fallará.

11.1 MANIPULANDO BD.

Se utilizan tres predicados para manipular los objetos de una BD:

Asserta(hecho). Permite un ordenamiento descendente de los datos, colocando el hecho delante de cualquier otra cláusula con el mismo predicado.

Assertz(hecho). Permite un ordenamiento ascendente. Colocando el hecho después de cualquier otra cláusula con el mismo predicado.

Assert(hecho). Coloca los datos en memoria sin un orden definido.

Retract(hecho). Refresca la memoria, suprimiendo el primer hecho de la base de datos que coincide con el hecho del argumento.

Consult(NombreArchivoDos). Trae un archivo de predicados de una base de datos a memoria. El archivo debe ser de texto, el cual previamente fue colocado con el predicado save().

Save(NombreArchivoDos). Guarda las cláusulas de una BD en disco.

11.2. DECLARACIÓN.

Se agrega al programa en la sección de Dominios, definiendo los predicados de que consta, como se muestra enseguida:

```
Domains
Database
Persona(nombre,edada,direccion)
Ciudad(codpost,nomciudad)
```

En donde persona y ciudad son dos bases de datos a las cuales se dará tratamiento, tal como se da a un objeto compuesto.

/* Ejemplo 29. Manipulando bases de datos dinámicas */

```

domains
    nombre,direccion,nomciudad = string
    edad,codpost=integer
    database
        persona(nombre,edad,direccion)
        ciudad(codpost,nomciudad)

predicates
    ejecuta1
    ejecuta2
clauses
    ejecuta1:-
        asserta(persona("Rosa",26,"conocido")),
        asserta(persona("Juan",33,"conocido")),
        asserta(persona("Javier",20,"conocido")),
        asserta(ciudad(23456,"coquimatlan")),
        asserta(ciudad(23456,"pueblo juarez")).

    ejecuta2:-
        persona(X,R,T),
        ciudad(Y,Z),
        write(X," ",R," ",T),nl,
        write(Y," ",Z),nl,
        fail.

goal

    makewindow(1,2,3,"ejemplo de uso de base de datos internas",3,4,15,60),
    ejecuta1,
    ejecuta2.

```

/* Ejemplo 30. Ordenando Bases de datos */

```

database
    numero(integer)
    nombre(string)
    apellido(string)
predicates
    ejecuta1
    ejecuta2
clauses
    ejecuta1:-
        write("ordenamiento de los datos de la base"),
        nl,
        assert(numero(1)),
        assert(numero(2)),
        assert(numero(3)),
        assert(numero(4)),
        assert(numero(5)),
        assert(nombre(carlos)),
        assert(apellido(rodriuez)),

```

```

    fail.
    ejecuta1:-
numero(X),
write(X),nl,
fail;true,
retract(numero(X)),
fail;true.
ejecuta2:-
nombre(Y),
apellido(Z),
write(Y),nl,
write(Z),nl,
fail;true,
retract(nombre(Y)),
retract(apellido(Z)),
fail;true.

goal
makewindow(1,2,3,"Ejemplo de uso de BD",0,0,24,80),
clearwindow,
ejecuta1,
ejecuta2.

```

CAPITULO 12 BASES DE DATOS EXTERNAS

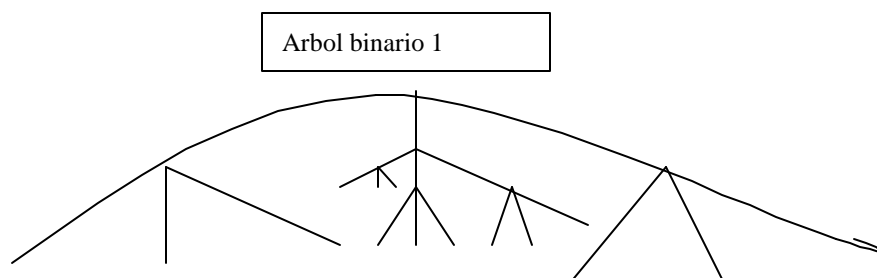
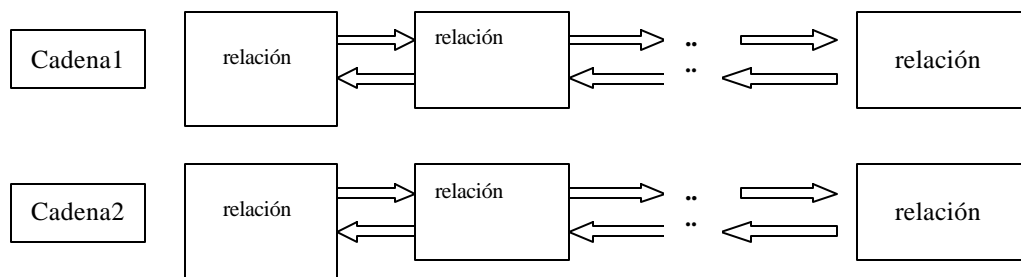
Las bases de datos dinámicas o internas, son muy útiles para el manejo de aplicaciones relativas principalmente a la inteligencia artificial, sin embargo, los requerimientos de la memoria RAM para una

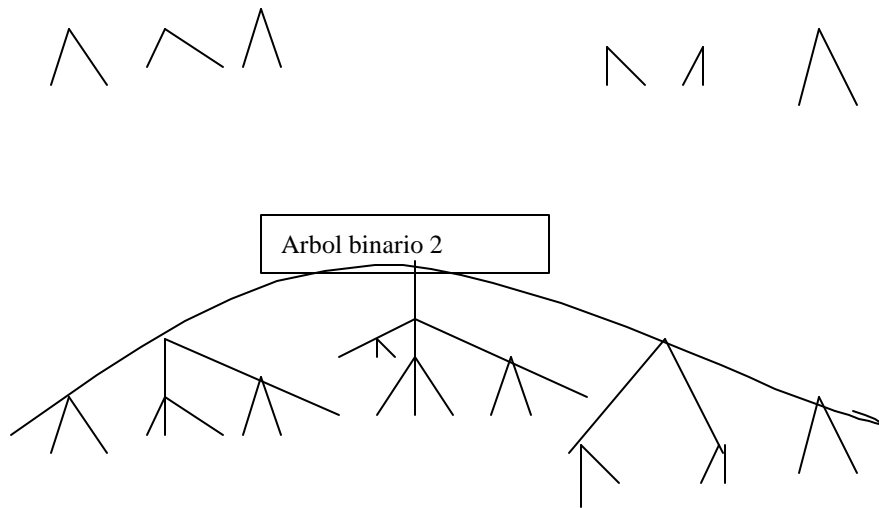
aplicación, podrían exceder la capacidad de cualquier computadora, lo cual conlleva a definir estrategias de administración de memoria para evitar que el programa tenga una terminación anormal.

Para paliar esta desventaja se incorpora al lenguaje un tipo de datos llamado base de datos externa que entre otras cosas permite realizar lo siguiente:

- Una aplicación con excesivo número de registros.
- Un sistema experto con muchas relaciones pero unos pocos registros con complicadas estructuras.
- Un sistema en el cual se puedan colocar grandes archivos de texto dentro de una base de datos.
- Un sistema en el cual se pueda tener varias bases de datos abiertas en forma simultánea y la facilidad de trabajar con la memoria extendida.

Una base de datos externa consiste de dos componentes: los datos colocados en cadenas y los correspondientes árboles binarios, los cuales pueden ser usados para acceder rápidamente a cada dato. Esquemáticamente una base de datos externa se representa así:





12.1 PREDICADOS.

- **Db_create.** Crea una nueva base de datos, su sintaxis es: `db_create (Dbase, Nombre, Lugar)`; donde las variables `Dbase` y `Nombre` corresponden al nombre interno y externo de los archivos.

El lugar en el cual se coloca el archivo externo está determinado por los siguientes valores:

`in_file` la base de datos externa está colocada en un archivo de disco.
`in_memory` la base de datos es colocada en la memoria principal.
`in_ems` la base de datos es colocada en la memoria extendida de la computadora.

Su declaración es:

Domains

Lugar = `in_file`; `in_file`; `in_ems`

Dos diferentes llamadas para el predicado `dbcreate` son:

`dbcreate(dbarchivo,"MYARCHIVO.DBA",in_file),`
`dbcreate(dbarchivo,"MYARCHIVO.DBA",in_memory).`

- **Db_open.** Abre una base de datos creada previamente, indentificandola por `Nombre` y `Lugar`, su sintaxis es:

`db_open (Dbase,Nombre,Lugar)`

Si `Lugar` está en memoria RAM o en la memoria extendida, `Nombre` es el nombre de la base de datos simbólica, si `Lugar` se declaro como `in_file`, `Nombre` es el nombre del archivo en disco.

- **Db_copy.** Sin importar la ubicación inicial de la base de datos externa, con este predicado es posible mover a una nueva ubicación física, su sintaxis es:

db_copy(Dbase,Nombre,Lugar)

Los usos de éste predicado son varios:

- Cargar una base de datos desde disco a memoria RAM y posteriormente grabarlo nuevamente.
- Copiar un archivo de mediano tamaño desde disco hacia la memoria extendida para un más rápido acceso.
- Cuando un archivo ocupe demasiado espacio en memoria, liberarla copiando el archivo hacia disco.

- Db_close. Este predicado cierra una base de datos externa, abierta, su sintaxis es:

db_close(Dbase).

- Db_delete. Cuando una base de datos ocupe demasiado espacio en memoria tanto RAM como extendida este predicado remueve dicha base de datos para de esta forma liberar espacio, su sintaxis es:

db_delete(Nombre,Lugar).

- Db_openinvalid. Permite al usuario abrir una base de datos que previamente fue marcada como invalida, su sintaxis es:

db_openinvalid(Dbase,Nombre,Lugar).

Si la computadora se inhibe, “resetea” o existe un “apagon” mientras un archivo está siendo actualizado todos los datos pueden perderse al no haber vaciado el buffer. Una bandera en la base de datos, puede indicar que hay un estado invalido después de actualizar.

Una base de datos es detectada como invalida después de una llamada de cualquiera de los predicados que cambian el contenido de la misma, eso incluye a:

chain_inserta	term_replace	btree_create
chain_inserz	term_delete	key_insert
chain_insertafter	chain_delete	key_delete

- Db_flush. Vacía los buffers y escribe su contenido en el destino apropiado de la base de datos, su sintaxis es:

db_flush(Dbase).

- Db_trees. Durante un retroceso (backtraking) en el recorrido de un árbol binario, une cada parte del mismo hacia la base de datos representada por Dbase, su sintaxis es:

nondeterm db_trees(Dbase,NomarbolB).

- Db_chains. Durante un retroceso, db_chains sucesivamente une Nomcadena hacia el nombre de cada cadena dentro de la base de datos Dbase, su sintaxis es:

nondeterm db_chains(Dbase,Nomcadena).

Los nombres de cadena son regresados en forma ordenada.

12.2 MANIPULANDO CADENAS.

Para insertar relaciones en una cadena de una base de datos externa se utilizan los predicados chain_inserta chain_inserz o chain_insertafter.

Chain_inserta y chain_inserz, corresponden a asserta y assertz respectivamente en bases de datos internas, su sintaxis es:

Chain_inserta(Dbase,Cadena,Dominio,Relacion,Referencia)

Chain_insertz(Dbbase,Cadena,Dominio,Relacion,Referencia)

Chain_inserta agrega la relación al principio de la cadena, mientras chain_insertz la inserta al final.

Por ejemplo, si dbase es declarado cómo dominio de db_selector, tal como:

Domains

Db_selector = dbase

Entonces en la siguiente llamada a chain_inserta

Chain_inserta(dba,cliente,persona, p(juan,"Madero 450",45), NuevaRef)

cliente es el nombre de la cadena y todos los clientes son colocados en una cadena.

12.3 PROGRAMAS EJEMPLO

/* Ejemplo 31.

*/

domains

db_selector = dba1 ; dba2

customers = customer(customer_name, address)

parts = part(part_name, ID, customer_name)

customer_name, part_name = symbol

ID = integer

address = string

predicates

access

goal

% create the databases dba1 and dba2

db_create(dba1, "dd1", in_memory),

db_create(dba2, "dd1.bin", in_file),

% insert customer facts into chain1 in dba1

chain_insertz(dba1, chain1, customers,
customer("Joe Fraser", "123 West Side"), _),

chain_insertz(dba1, chain1, customers,
customer("John Smith", "31 East Side"), _),

chain_insertz(dba1, chain1, customers,
customer("Diver Dan", "1 Water Way"), _),

chain_insertz(dba1, chain1, customers,
customer("Dave Devine", "123 Heaven Street"), _),

% insert parts facts into chain2 in dba2

chain_insertz(dba2, chain2, parts, part("wrench", 231, "John Smith"), _),

chain_insertz(dba2, chain2, parts, part("knife", 331, "Diver Dan"), _),

access,

db_close(dba1), db_close(dba2),

db_delete("dd1", in_memory),

db_delete("dd1.bin", in_file).

```
clauses
access:-
    chain_terms(dba1, chain1, customers, customer(Name, ADDR), _),
    chain_terms(dba2, chain2, parts, part(Part, Id, Name), _),
    write("send ", Part, " part num ", Id, " to ", Addr), nl,
    fail.
access.
```

CAPITULO

13

APLICACIONES AVANZADAS

13.1 PREDICADOS RELATIVOS AL DOS.

Predicados

uso

System(). Manda una cadena al DOS para la ejecución. Es la orden de propósito general del DOS para utilizar desde el interior de los programas en prolog. Por ejemplo: si tiene en la unidad A, llamado work.pro y quiere copiarlo en la unidad B, solo debe añadir la línea: system("copy a:work.pro b:").

Conversion de tipos.

Char_int(ParamCar,ParamEnt). Convierte un valor tipo caracter a su equivalente Ascii.

Str_car(ParamCadena,ParamCar). Convierte un valor cadena en caracter.

Str_int(cadena,Longitud). Funciona con los dominios "string" e "int".

Upper_lower(CadenaenMayuscula,CadenaenMinuscula). Convierte una cadena escrita en mayuscula, a minuscula y viceversa.

/*Ejemplo 32. Predicados del dos */

```
goal
makewindow(1,7,7,"imprime el contenido de un archivo",0,0,10,35),
system("type datos.dba>prn").
```

/*Ejemplo 33. Copiando un archivo */

```
goal
makewindow(1,7,7,"Archivo fuente",0,0,20,35),
write("cual archivo deseas copiar? "),cursor(3,8),readln(X),
makewindow(2,7,7,"Destino",0,0,40,20,35),
write("cual es el nombre de la nueva copia "),cursor(3,8),readln(Y),
concat(X,"",X1),concat(X,Y,Z),
concat("copy",Z,W),
makewindow(3,7,7,"Proceso",14,15,8,50),
write("copying..",X,"To",Y),
cursor(2,3),
System(W).
```

13.2 ESQUEMAS COMPLEJOS CON LISTAS

Para concluir el presente trabajo se presentan los siguientes ejemplos de usos complejos:

/* Ejemplo 34. Programa que utiliza una lista para almacenar las notas de una melodía y recorre la lista para interpretarlas */

constants

```

a = sound (15, 600)
b = sound (5, 50)
c = sound (15, 600)
d = sound (5, 50)
e = sound (30, 900)
f = sound (5, 50)
g = sound (15, 600)
h = sound (5, 50)
i = sound (30, 900)
domains
    nota = sound(integer, integer)
    canc = nota*
predicates
    ppal
    init (canc)
    play (canc)
    speak (nota)
goal
    ppal.
clauses
    ppal :-
        makewindow (1,3,3," FIME - Prolog ", 1,3,20,73),
        write ("Ejecución de unas notas musicales\n"),
        write ("Presione alguna tecla para iniciar ..."),
        readchar (_, nl, nl,
        init (Melodia),
        play (Melodia).

    init ([a,b,c,d,e,f,g,h,i]).

    play ([]).
    play ([P|Q]) :-
        write ("Nota : ", P), nl,
        speak (P),
        play (Q).
    speak (sound(F,D)) :-
        sound (F,D).

```

/*Ejemplo 35. Generando un retardo*/

```

predicates
    delay(real)
clauses
    delay(N):-
        N>0,!,
        N1=N-1,
        write(N1),
        delay(N1).
    delay(0).

```

```

goal
    makewindow(1,2,3,"",0,0,20,60),
    clearwindow,
    write("esto es un retardo"),nl,
    delay(52000),
    write("proceso concluido.").

/*Ejemplo 36. Generando números aleatorios */

predicates
    aleatorio(integer)
clauses
    aleatorio(X):-
        random(15,Z),
        X=Z,
        Write(Z),nl,
        Z<15,aleatorio(X).
Goal
    Makewindow(1,2,3,"",0,0,24,80),
    Clearwindow,
    Aleatorio( R ).

```

13.3 PROGRAMACION MODULAR.

La programacion modular construye un programa final o proyecto a partir de unos cuantos o varios modulos independientes.

Todos los modulos de un proyecto tienen que compartir una tabla de simbolos interna unica por tanto, cada modulo debe saber el proyecto en el que esta incluido

{ Simplificacion en la depuracion
Interface con otros lenguajes
Conservacion de las variables locales
Manejo de variables globales

ventajas

Requerimientos:

Dar una definicion del proyecto

Declarar las variables locales y globales

Si es necesario, declarar archivos incluidos

Asegurarse de que como maximo un modulo tiene un objetivo interno y que dicho archivo sea el principal.

Asegurarse de que los archivos adecuados para el enlace estan en los directorios apropiados.

Definición del proyecto:

1. Especifique los modulos que se utilizarán en el programa final compilado; Este archivo se crea con la opcion Edit PRJ file del menu Options dando el nombre del proyecto a esa lista y luego editando esa lista para que contenga todos los nombres de modulo. El nombre no debe incluir la extensión PRJ. El archivo del proyecto debera tener la extension PRJ.

2. Al compilar debe utilizarse la directiva de compilacion Project.

3. Una vez compilado el proyecto, se generará en forma automatica el programa ejecutable.

Considere el siguiente ejemplo el cual incluye dos archivos de programa (proyecto y inseta), un archivo incluido en el cual se declaran lo procedimientos globales (global), y por ultimo el archivo de proyecto en el cual se indican los archivos de programa que son parte del proyecto (proyecto.prj).

```
/* archivo principal: proyecto.pro*/
```

```
project "proyecto"
include "c:\\prolog\\global.pro"
domains
  opcion = integer
predicates
  menu
  toma_opcion ( opcion )
  procesa_opcion ( opcion )
```

```
clauses
  repetir. repetir :- repetir.
```

```
menu :-
  repetir,
  makewindow(1,2,2," MENU ",5,5,14,33),
  write(" Elija su opcion: \n\n",
    " (1) Leer un archivo\n",
    " (2) Escribir en un archivo\n",
    " (3) Borrar un archivo\n",
    " (4) Buscar un archivo\n",
    " (5) Salir "),
  cursor(0,17),
  toma_opcion (Opcion),
  removewindow,
```

```

procesa_opcion(Opcion),
Opcion >= 5.
toma_opcion(C1) :-
    readchar(C),
    C1 = C - '0',
    C1 <= 5, !,
    write(C1).
toma_opcion(C) :-
    beep, toma_opcion(C).
procesa_opcion(1) :-
    !,makewindow(11,2,2," Opcion 1 ",5,12,7,26),
    write("Leer un archivo"),
    readchar(_),
    removewindow.
procesa_opcion(2) :-
    !,makewindow(12,2,3," Opcion 2. Escribir en un archivo ",5,12,7,60),
    inserta,
    readchar(_),
    removewindow.
procesa_opcion(3) :-
    !,makewindow(13,3,4," Opcion 3 ",5,22,7,26),
    write("Borrar un archivo"),
    readchar(_),
    removewindow.
procesa_opcion(4) :-
    !,makewindow(14,4,3," Opcion 4 ",5,32,7,26),
    write("Crear un archivo"),
    readchar(_),
    removewindow.
procesa_opcion(5) :-
    !,makewindow(13,3,4," Opcion 5 ",5,42,7,20),
    write("Salir"),
    readchar(_),
    removewindow.

goal
    makewindow(1,2,3," Ejemplo 2 de manejo de archivos ",0,0,25,80),
    menu.

```

/*archivo secundario: inserta.pro */

```

project "proyecto"
include "c:\\prolog\\global.pro"
domains
file=visualiza
clauses
inserta:-
write("Inserta una cadena de caracteres"),
readln(Cadena),
openwrite(visualiza,"c:\\visualiza.dto"),
write(Cadena),nl,
Closefile(visualiza),

```

```
writedevic(screen),
write("Los datos fueron grabados correctamente."),nl.
```

```
/*archivo incluido: global.pro */
```

```
global predicates
inserta
nondeterm repetir
```

```
/* Archivo de proyecto: proyecto.prj */
```

```
proyecto
inserta
```

13.4 INTERFAZ CON OTROS LENGUAJES.

Es posible generar programas en C, Pascal o ensamblador que puedan compartir argumentos, sin embargo, es necesario tomar en cuenta que antes de llamar a rutinas hechas en otros lenguajes es necesario declararlas como predicados externos en Turbo Prolog, y entender los parámetros y secuencias para poder ejecutar cada programa. Los lenguajes con los cuales es posible interfazar son lenguaje C pascal y ensamblador.

Turbo C.

Para el enlazamiento debera configurar el codigo de acuerdo con las siguientes opciones:

1. Options/Compiler/Model/Large
2. Options/Compiler/Optimization/Jump optimization..On
3. Options/Compiler/Code Generation/Generate underbars.. Off

El codigo debera ser enlazado de acuerdo con la siguiente linea de commando generica:

```
Tlink INIT < pOBJ> <cOBJ> <.sym> [exe],[map],[usr] +prolog[+emu+mathl+cll]
```

Los argumentos de dicha linea de commando hacen lo siguiente:

Argumento	Que hace
Tlink	invoca tlink, el turbo enlazador de Borland
INIT	el archivo de inicializacion de turbo Prolog
POBJ	el archivo objeto generado con prolog
COBJ	el archivo objeto generado con turbo C
.sym	es el archivo simbolo de turbo Prolog
exe	opcional, es el nombre del archivo ejecutable
map	opcional, es el nombre del archivo tipo map
usr	opcional, se refiere a la lista de archivos de librerías de usuario

prolog	se refiere a Prolog.lib
emu	opcional, se refiere a la librería de emulación de punto flotante de C, Emu.lib
mathl	opcional, se refiere a la librería de C Math.lib
cl	opcional, se refiere a la librería de C, Cl.lib

Declarando predicados externos.

La declaración de predicados externos se realiza de la siguiente forma:

Global predicates

```
Agregar(integer, integer, integer) - (i, i, o), (i, i, i) lenguaje c
Scanner (string, string) - (i, o) lenguaje pascal
Triple(integer, real) - lenguaje asm
```

Adicionalmente es posible hacer llamadas a Turbo Prolog desde otros lenguajes. Si un predicados global es declarado que existe en otro lenguaje, y hay hay clausulas para ese predicado, Turbo Prolog generara una rutina "llamable" para ese lenguaje.

El siguiente Programa declara dos predicados gloales en C; mensaje y hola. El predicado mensaje puede ser llamado desde un modulo en C usando la funcion llamada mensaje_0 en el codigo fuente C.

```
include "c:\\prolog\\interfaz.c"
/*

*/

global predicates
    mensaje(string) - (i) language c
    hola_c - language c

clauses
    mensaje(S) :-
        makewindow(13,7,7,"",10,10,3,50),
        write(S), readchar(_),
        removewindow.

goal
    mensaje("Hola desde Turbo Prolog"),
    hola_c.
```

la seccion de objetivos del programa anterior llama a la función de C hola_c, la cual a su vez llama el predicado mensaje_0 para desplegar un mensaje.

```
/*
```

```

*/

void hola_c_0()
{
    mensaje_0("Hola desde Turbo C");
}

```

CONCLUSIONES

Esta obra muestra una parte del potencial que tiene Prolog; en un nivel avanzado se puede hablar de control de dispositivos por computadora, creación de juegos animados 3D, sistemas expertos interactuando en internet, bases de datos inteligentes, simulación en tiempo real, pero sobre todo trata de introducir al lector en el uso de este paradigma de programación que aun no se ha explotado con todas sus ventajas pues el esquema de trabajo es similar al que realiza el cerebro humano cuando realiza operaciones cognoscitivas, es decir el ser humano programa cada una de las actividades que realiza, en forma logica y no como el enfoque estructurado lo hace, es decir de una manera matemática, Aunque el enfoque orientado a objetos tambien trata de adoptar el esquema de de adquisición del conocimiento realizado por las personas.

Al inicio del nuevo milenio las tecnologías de desarrollo de sistemas tiende a incluir en cada lenguaje un modulo generador de codigo fuente por la urgencia de la solucion de los problemas informaticos actuales, bajo este tenor es digno mencionar que Visual Prolog no cuenta con un mecanismo generador de codigo fuente y si así fuera esto seria equivalente a definir un limite a la libertad de desarrollo lo cual no es posible asimismo, ya cuenta con la posibilidad de evitar cualquier error originado con el “bug” del milenio sin que se haya incluido un parche, es decir desde su diseño este problema fue considerado. Asimismo Turbo Prolog a pesar de que su ultima actualizacion fue hecha en 1988 nunca tubo problemas para el manejo de las fechas a menos que la computadora utilizada no tuviera un Bios actualizado lo cual en terminos generales no ocurre debido a la constante actualizacion de las tecnologías hardware por los fabricantes de computadoras.

REFERENCIAS BIBLIOGRAFICAS

- [1] Phillip R. Robinson, Aplique Turbo Prolog, Mc Graw Hill, 1987.
- [2] Kelly M. Rich & Phillip R. Robinson, Using Turbo Prolog, Mc Graw Hill, 1988.
- [3] Turbo Prolog Reference Guide, Borland International, 1988
- [4] Turbo Prolog User's Guide, Borland International, 1988.
- [5] Francis Giannesini, Henri Kanoui, Robert Pasero, Michel Van Caneghem, Prolog , Adisson Wesley, 1989.
- [7] Ivan Bratko, Prolog Programming for Artificial Intelligence, Adisson Wessley, 1990.
- [8] George F. Luger, William A. Stubblefield, Artificial Intelligence Structures and Strategies for Complex Problem Solving, Adisson Wesley, 1992.

CORREO ELECTRÓNICO

Si el lector desea hacer algún comentario, sugerencias o consultas puede contactar al autor mediante los siguientes correos electrónicos:

fuentesr@volcan.ucol.mx
fuentesr@hotmail.com
fuentesr_99@yahoo.com
fuentesr@palmera.colimanet.com

O bien consultar a las siguientes paginas de internet:

www.geocities.com/fuentesr_99
<http://www.members.tripod.com.mx/fuentesr>
www.icq.com/51813267

APÉNDICE

A

PROGRAMAS PARA EVALUACION DE CONOCIMIENTOS

1. Elabora un programa utilizando archivos en el que el usuario realice operaciones aritméticas de suma, resta, multiplicación y división y el resultado se almacene en un archivo de datos, pudiendo consultar el contenido de dicho archivo. Toma en cuenta que el programa amerita un menú de opciones.
2. Con gráficos estáticos y dinámicos elabora un programa que dibuje el croquis de una casa.
3. Elabora un programa en el que introduzcas en una lista un gráfico de tortuga.
4. Elabora un programa en el que utilizando gráficos de tortuga despliegues una computadora multimedia (con bocinas, teléfono y micrófono), toma en cuenta que deberás utilizar declaraciones de dominios, predicado y cláusulas.
5. Elabora un programa con listas unidas en el que muestres cinco semestres de una carrera uniando las materias que conforman cada semestre.
6. Elabora un programa en el cual calcules el valor de una resistencia mediante interacción con el usuario y los resultados sean grabados en un archivo de datos.
7. Elabora un programa en el que realices altas bajas, modificaciones y consultas de los datos de un archivo que sea generado por el usuario y en el cual los campos los datos de un control de almacén de una ferretería. Toma en cuenta que el programa amerita un menú de opciones.

8. Utilizando gráficos de tortuga elabora un programa en el cual despliegues el dibujo de una casa y un árbol.
9. Utilizando listas unidas elabora un programa en el que muestres los componentes del cuerpo humano clasificado por sus respectivos sistemas los cuales deberás unir y mostrar como una lista unida llamada persona.
10. Elabora un programa con objetos compuestos para introducir los sistemas que conforman un coche
11. Elabora el programa anterior utilizando listas unidas.
12. Utiliza la caja de herramientas para hacer una animación de una motocicleta saltando un precipicio
13. Elabora un programa que realice búsquedas en una base de datos lógica relativas a personas y sus relaciones una con otra ejemplo: Juan es hermano de rosa, es el resultado de una pregunta de usuario, toma en cuenta que deberás utilizar operaciones lógicas con sus operadores correspondientes.

APÉNDICE

B

VISUAL PROLOG, O WINPROLOG SU RELACION CON TURBO PROLOG

Existen muchos dialectos del lenguaje Prolog, sin embargo en la actualidad dos variantes son las que a nivel mundial estan tratando de sustituir al hueco que dejo en su momento Turbo Prolog cuando Borland dejo de generar nuevas actualizaciones, una vez que fueron evaluadas por el autor se determinaron marcadas diferencias entre si, aunque es indudable la potencia de ambos lenguajes. En forma resumida se describen las características de cada una, pues su estudio exhaustivo se reserva para un siguiente trabajo que se encuentra en preparación.

WINPROLOG:

a) Interfaz de usuario: rustica, austera y poco amigable, no cuenta con una caja de herramientas de desarrollo y el codigo es generado en su totalidad por el usuario.

Cuenta con un editor para generar el codigo fuente pero cuando el programa se ejecuta despliega el editor de ejecución de programa y en caso de errores son desplegados en el mismo editor esto puede prestarse confusiones si en la siguiente el programa se ejecuta correctamente. Consulte la figura numero 3 mostrada a continuación.

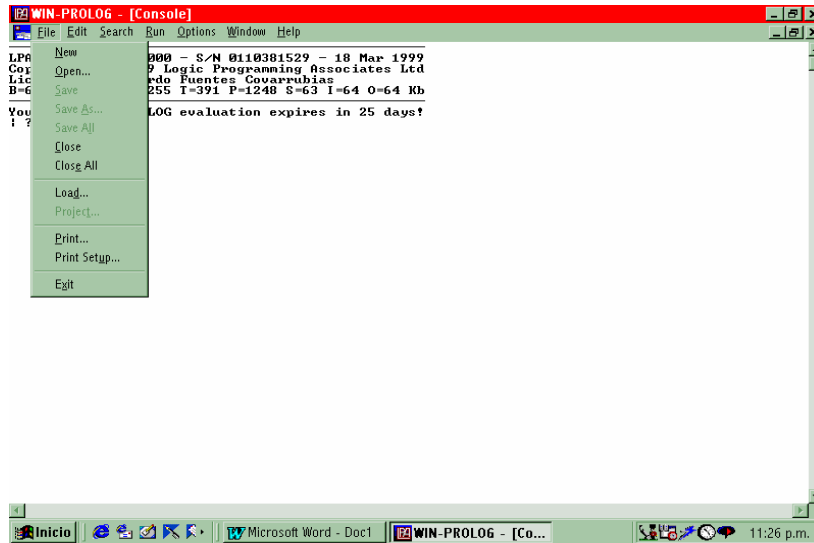


Figura 3. Interfaz de usuario.

b) Entorno de programación: es en un 80% similar a la ya estudiada en este trabajo, aunque no se hace tanto énfasis al entorno declarativo sino a su enfoque clausal es decir basado en cláusulas y predicados, es modular pues se utilizan reglas como en Turbo Prolog y es en base a una ejecución secuencial arriba-abajo en que se ejecuta cada regla.

Agrega un esquema de chequeo de sintaxis y el código fuente es compilado pero no se genera código ejecutable, por lo cual no es posible ejecutar las aplicaciones en forma independiente de WinProlog.

Cuando se ejecuta un programa en lugar de definir objetivos o “goals” se definen búsquedas o “query” después de haber sido compilado.

Un aspecto a resaltar es la potencia gráfica de WinProlog apoyada en las ventajas del Entorno Windows 95 o 98. Consulte las figuras 4 y 5, para ejemplo de lo explicado líneas arriba.

Por último si se desea migrar de Turbo Prolog a WinProlog se requiere regenerar la totalidad del código fuente.

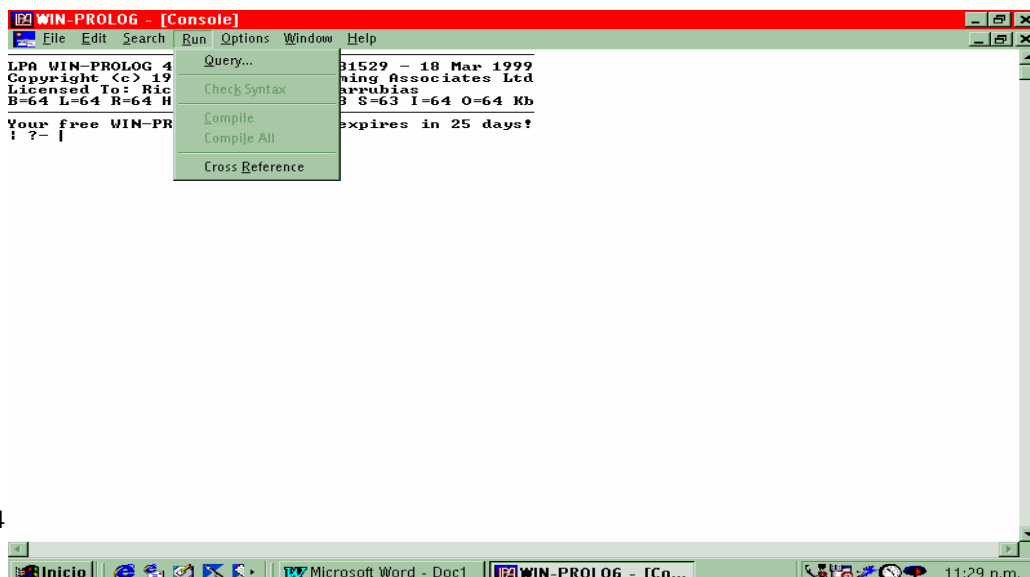


Figura 4. Interfaz de programación.

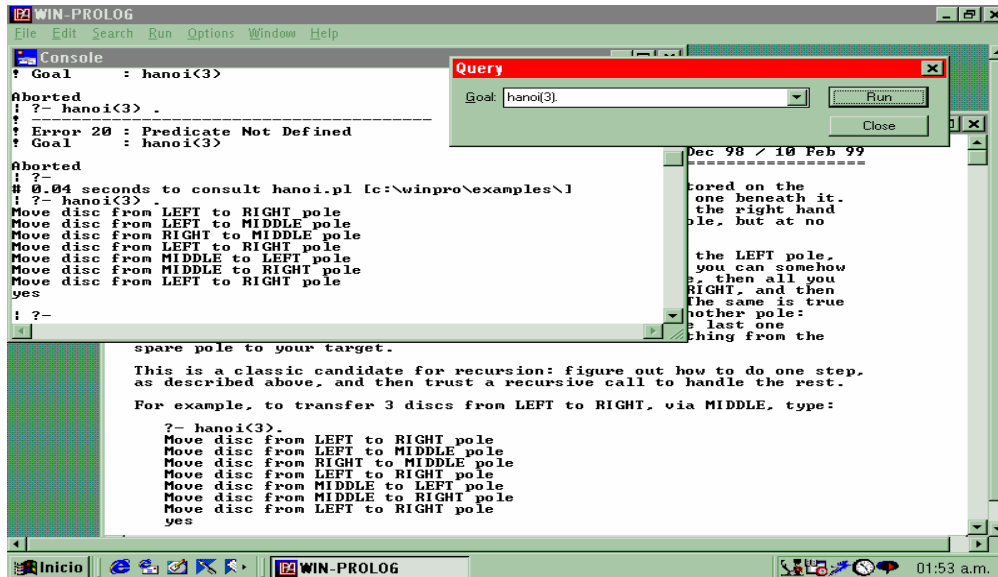


Figura 5. Entorno de programación.

VISUAL PROLOG.

a) Interfaz de usuario: amigable, similar en el primer nivel a WinProlog, cuenta con mas opciones de trabajo, similares a una caja de herramientas de desarrollo, el codigo es generado en un 60 o 70% por el usuario y el resto es agregado por WinProlog con solo declarar en forma de incluido los modulos correspondientes, que de fabrica proporciona el desarrollador Prolog Devenlopment Center de Dinamarca. Consulte la figura numero 6 mostrada a continuación.

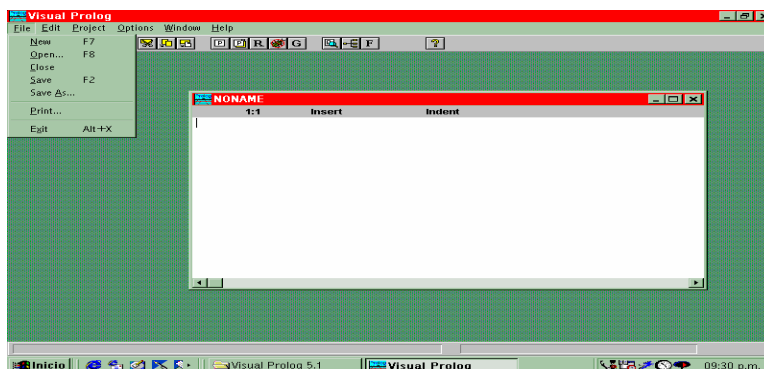


Figura 6. interfaz de usuario de Visual Prolog.

b) Interfaz de programación. En esencia un 100% similar a la ya estudiada en este trabajo, respetando los módulos de declaraciones como Predicados y clausulas con la salvedad de que puede existir mas de una seccion definida para predicados y clausulas, es importante el entorno declarativo, se utilizan reglas como en Turbo Prolog y los predicados definidos por el lenguaje son en esencia iguales a Turbo Prolog, lo cual facilita migrar de un entorno DOS a uno de 16 bits o 32, la ejecución de un programa sigue siendo secuencial arriba-abajo pero su orden de ejecución dependera de el orden de declaración de los objetivos internos.

Si una aplicación es extensa es decir conformada por varios archivos y con referencias a varios incluidos puede ser almacenada como un proyecto, se compila y es posible generar su respectivo codigo ejecutable lo cual permite independencia de trabajo con el lenguaje cuando un proyecto ha sido debidamente concluido, lo cual no es posible hacer con WinProlog.

Las aplicaciones que se generan con Visual Prolog tienen mayor potencia y van desde aplicaciones orientadas a internet, sistemas expertos o incluso sistemas para administración de reservaciones de aerolíneas o administración de trafico aereo, las aplicaciones para internet pueden ser ejecutadas en linea es decir cada usuario desde su terminal puede ejecutar un programa al mismo tiempo que otros usuarios si ocasionar algun conflicto en tiempo de ejecución.

Consulte las figuras 7, 8 y 9 para ejemplo de lo explicado lineas arriba.

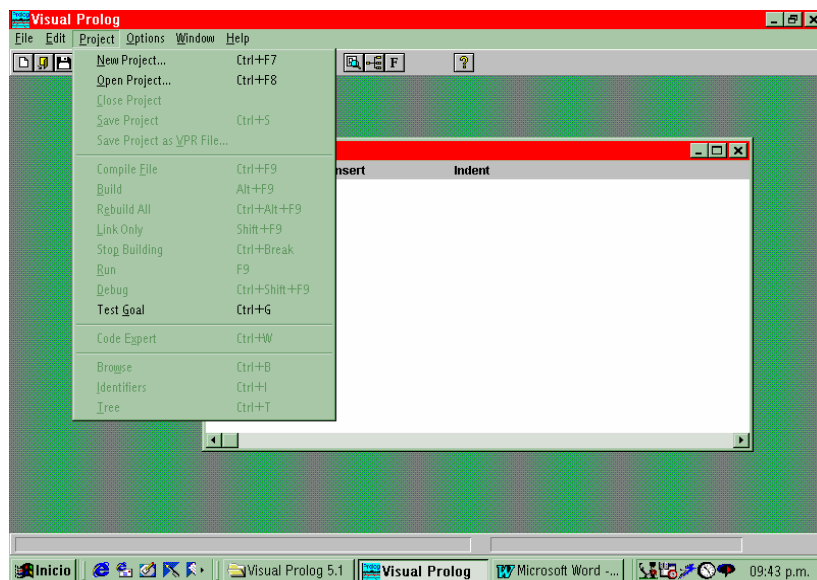


Figura 7. interfaz de programación.

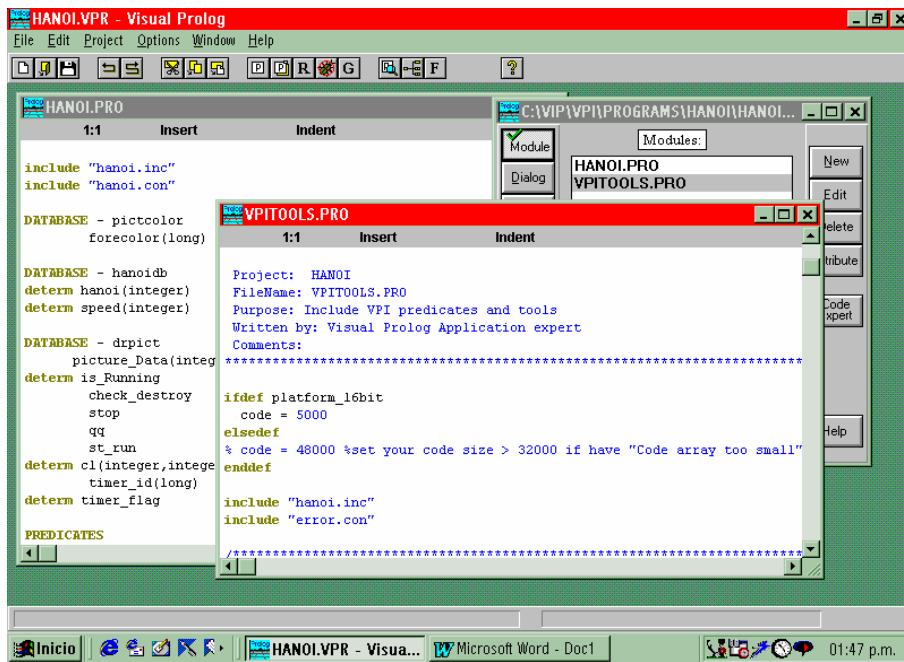


Figura 8. interfaz de programación.

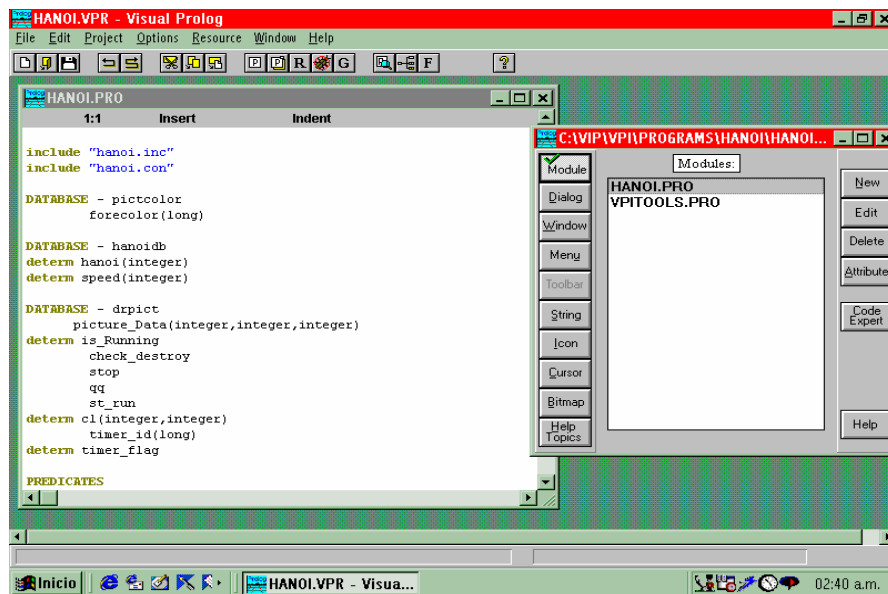


Figura 9. interfaz de programación.

Es posible ejecutar un programa en modo simple desde el editor de WINPROLOG, eligiendo new en la opcion File de la barra de opciones, e inmediatamente después generar el codigo fuente y para ejecutarlo elegir test goal en la opcion Project del menu de opciones tal y como se muestra en las siguientes figuras:

