Resolución de Problemas y Algoritmos

Clase 12
Estrategia de resolución por descomposición del problema.
Implementación de primitivas en Pascal: funciones.

Dr. Diego R. García

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca - Argentina

Motivación

Indique a que clase de elemento de Pascal corresponden los ejemplos de cada columna:

then	**integer**	**TRUNC**	**WRITE**
and	**real**	**EOF**	**READ**
program	**text**	**CHR**	**RESET**

Agregue un ejemplo más a cada columna de la tabla.

¿En qué lugar de un programa en Pascal se pueden utilizar los elementos de la tercera y cuarta columna?

Funciones y procedimientos predefinidos

Al programar en Pascal han usado primitivas predefinidas. Por ejemplo:

primitivas como sentencias (procedimientos predefinidos)

```
…
ASSIGN (F1,'mi-archivo');
WRITELN ('ingrese un número'); READLN (num);
RESET (F1);  REWRITE (F2);
while not EOF(F1) do begin
  READ (f1,elem);
  if TRUNC(elem) > SQR(num)
   then WRITE (F2, elem)
   else WRITELN ( TRUNC (elem) , ' menor que' , SQR (num) );
end;
…
```

primitivas en expresiones (funciones predefinidas)

¿Quién creó el código de estas primitivas predefinidas?

Resolución de Problemas y Algoritmos Dr. Diego R. García 3

Funciones en Pascal

Ejemplos de algunas funciones predefinidas :

- **EOF(F): recibe un manejador de archivo y retorna boolean**
- **TRUNC(R): recibe real y retorna integer**
- **SQRT(R): recibe real y retorna real**
- **CHR(I): recibe integer y retorna char**

Características de las Funciones en Pascal:

- **Se utilizan en una expresión.**
- **Reciben valores de algún tipo de dato de Pascal.**
- **Siempre retornan un valor de un tipo de Pascal.**

¿Podré construirme mis propias funciones?

Resolución de Problemas y Algoritmos Dr. Diego R. García 4

Procedimientos en Pascal

Ejemplos de algunos procedimientos predefinidos en Pascal:

- **assign (F, 'nombre');**
- **reset (F);**
- **read (A);**
- **readln;**

Reflexionemos ahora sobre procedimientos:

- **Se los usa en una sentencia (no en expresiones).**
- **Pueden recibir/retornar 0 o más valores.**

¿Podré construirme mis propios procedimientos?

Resolución de Problemas y Algoritmos Dr. Diego R. García 5

Resolución de problemas con primitivas

Hemos visto cómo resolver problemas simples, escribiendo un algoritmo y luego un programa usando asignaciones, condiciones y repeticiones.

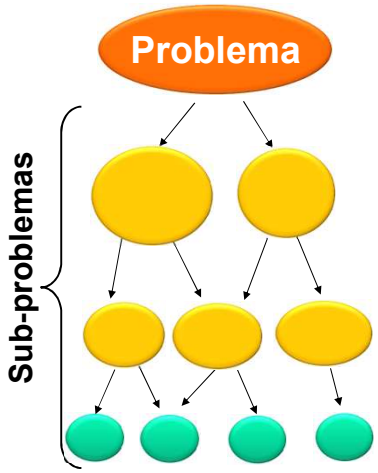
En lo que resta de la materia veremos:

1. **Técnicas para resolver problemas más complejos.**
2. **Cómo escribir algoritmos basados en primitivas y algoritmos que son primitivas.**
3. **Cómo implementar en Pascal primitivas (funciones y procedimientos) y poder usar las dos técnicas anteriores.**

Una primitiva es una operación o acción conocida, utilizada en un algoritmo o programa considerándola como básica.

Resolución de Problemas y Algoritmos Dr. Diego R. García 6

Conceptos: Técnica de resolución de problemas

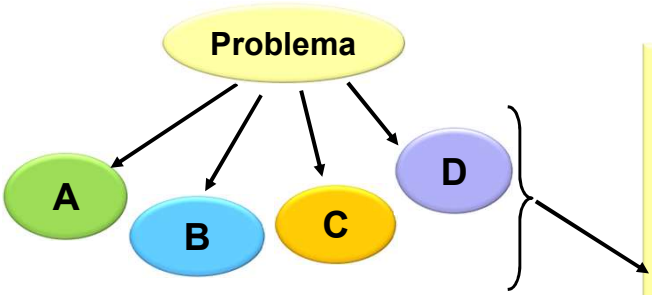


Técnica: Resolución de un problema por descomposición en problemas más simples.

- Cuando se intenta resolver un problema complejo puede abordarse la solución descomponiendo (dividiendo) el problema en partes (sub-problemas) más simples.
- De tal manera que la solución de las partes permita resolver el problema original.
- Si los sub-problemas siguen siendo complejos pueden también dividirse hasta llegar a problemas que no necesitan dividirse.

Resolución de Problemas y Algoritmos Dr. Diego R. García 7

División de un problema en sub-problemas



Metología: Para resolver un problema complejo se propone:

1) **dividir** en subproblemas,
2) **resolver** cada parte y luego
3) **para cada parte implementar primitivas** en Pascal: como funciones o procedimientos

```
Program SOLUCIÓN;
    Function A
    Procedure B
    Function C
    Procedure D
Begin
  …
End.
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 8

Funciones predefinidas para reales/enteros en Pascal

Función	Descripción	Recibe	Retorna
abs	Valor absoluto	real o integer	El mismo tipo recibido
arctan	arctan en radianes	real o integer	real
cos	cosine de un radián	real o integer	real
exp	***e*** a una potencia	real o integer	real
ln	Algoritmo natural	real o integer	real
round	redondeo	real	integer
sin	Seno de un radián	real o integer	real
sqr	Cuandrado (square)	real o integer	El mismo tipo recibido
sqrt	square root (raíz cuad.)	real o integer	real
trunc	truncado	real o integer	integer

http://wiki.freepascal.org/Standard_Functions

Resolución de Problemas y Algoritmos Dr. Diego R. García 9

Funciones predefinidas para reales/enteros en Pascal

Función predefinida	Datos que recibe	Datos que retorna
abs	real o integer	mismo tipo recibido
round	real	integer
sqr	real o integer	mismo tipo recibido
sqrt	real o integer	real
trunc	real o integer	integer

Estas funciones deben usarse en expresiones. Por ejemplo puedo hacer:

```
if abs(num) < 100
then resultado:=sqr(num);
```

Considere que queremos hacer una función "potencia" en Pascal, la cual permita calcular base ^{exponente} .
Por ejemplo, para obtener en resultado el valor " 2^3 " poder hacer:

resultado:=potencia(2,3);

El objetivo de este ejemplo es simplemente mostrar cómo se implementa una función en Pascal, por lo tanto, se implementará solamente para el caso en que base es un número entero y exponente un entero no negativo.

Resolución de Problemas y Algoritmos Dr. Diego R. García 10

Función para la potenciación (restringida)

Para simplificar la explicación, nuestra nueva primitiva "potencia" asume que **recibirá un número entero** para la base y un entero no negativo para el exponente; y **retornará un resultado entero**.

Algoritmo: potencia
Datos que recibe: base (entero) y exponente (entero no neg.)
Dato que retorna: resultado (entero)
resultado:=1;
Repetir exponente veces: resultado:=resultado*base.

Para implementar este algoritmo como una función en Pascal, hay una parte que ya hemos visto:

```
resultado := 1;
FOR aux:= 1 TO exponente DO resultado:= resultado * Base;
```

Veremos ahora como indicar el resto de los elementos para obtener una función en Pascal.

Resolución de Problemas y Algoritmos Dr. Diego R. García 11

Funciones en Pascal

A continuación se muestra una implementación para el algoritmo potencia (vea la sintaxis de funciones en los diagramas sintácticos.)

Algoritmo: potencia
Datos que recibe: base y exponente (enteros)
Dato que retorna: resultado (entero)
resultado:=1;
Repetir exponente veces: resultado:=resultado*base.

```
FUNCTION Potencia (Base, Exponente: integer): integer;
{Esta función retorna base elevado a la exponente}
VAR aux, resultado: integer;        // variables propias de la función
BEGIN
  resultado := 1;
  FOR aux:= 1 TO Exponente DO resultado := resultado * Base;
  Potencia:= resultado;
END;
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 12

Conceptos: en Pascal una función tiene:

1. Un **nombre** (con el cual se la invocará desde una expresión).
2. **Parámetros** (entre los cuales estarán los datos de entrada).
3. **Tipo del resultado** (que será el tipo de la función y determinará en que expresión podrá ser usada)
4. **Variables locales** (que son propias de la función).
5. Sentencias (también llamado "**cuerpo**" de la función).
6. **Asignación** de una expresión al **nombre** de la función (al menos una vez). Es la forma de retornar un valor.

```
                    1                2                3
FUNCTION Potencia (Base, Exponente: integer) : integer;
{retorna base elevado a la exponente}
VAR aux,resultado: integer;  4    // variables auxiliares
BEGIN
  resultado := 1;
5 FOR aux:= 1 TO Exponente DO resultado := resultado * Base;
  Potencia:= resultado;  6
END;
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 13

```
PROGRAM prueba; {prueba la función potencia}
VAR B,E, Pot :Integer;

FUNCTION Potencia(Base,Exponente:integer) : integer;
{utilidad: calcula "base" elevado a la "exponente"}
VAR aux,resultado: integer;
BEGIN
  resultado := 1;
  FOR aux:= 1 TO Exponente DO
        resultado := resultado * Base;
  Potencia:= resultado;
END;

BEGIN
 write('Ingrese base y exponente:');
 repeat readln(B,E); until E>=0;
 Pot:=Potencia(B,E);
 writeln(B,'a la',E,'=',pot);
 writeln('3 a la ', E+1, '=', potencia(3,E+1));
END.
```

Variables globales

Tipo de la función

Parámetros formales

Documentación de la intención de esta función

Variables Locales

Asignación del resultado

Llamada a la función desde una expresión

Para hacer la traza vea la próxima explicación

Resolución de Problemas y Algoritmos Dr. Diego R. García 14

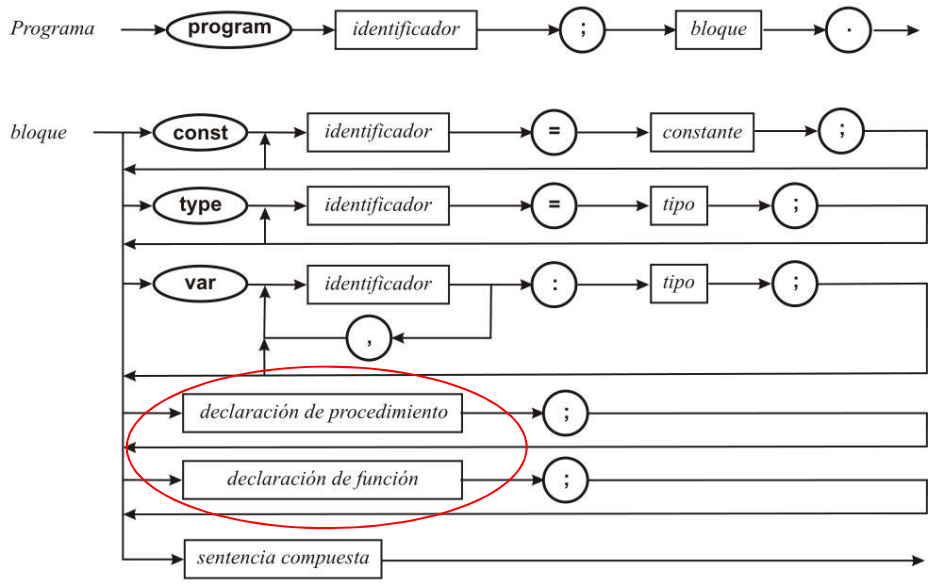
```
PROGRAM prueba; {prueba la función potencia}
VAR B,E, Pot :Integer;
FUNCTION Potencia(Base,Exponente:integer) : integer;
{utilidad: calcula "base" elevado a la "exponente"}
VAR aux,resultado: integer;
BEGIN
  resultado := 1;
  FOR aux:= 1 TO Exponente DO
        resultado := resultado * Base;
  Potencia:= resultado;
END;

BEGIN
 write('Ingrese base y exponente:');
 repeat readln(B,E); until E>=0;
 Pot:=Potencia(B,E);
 writeln(B,'a la',E,'=',pot);
 writeln('3 a la ', E+1, '=', potencia(3,E+1));
END.
```

Parámetros formales por valor:
reciben una copia de los valores de los parámetros efectivos

Parámetros efectivos.
Puedo utilizar cualquier constante, variable o expresión que sea asignación-compatible con el tipo del parámetro formal.

Resolución de Problemas y Algoritmos Dr. Diego R. García 15

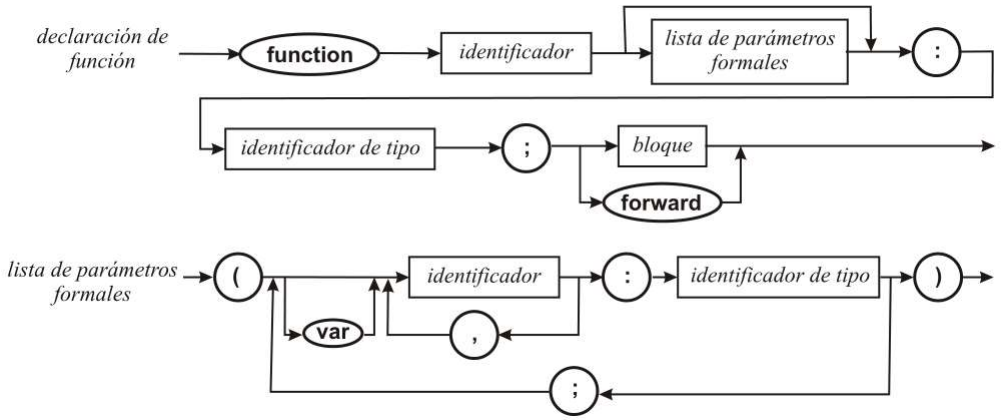
Diagrama sintáctico



Resolución de Problemas y Algoritmos Dr. Diego R. García 16

Diagrama sintáctico

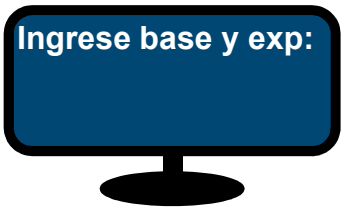
```
FUNCTION Potencia(Base,Exponente:integer) : integer;
{utilidad: calcula "base" elevado a la "exponente"}
VAR aux,resultado: integer;
BEGIN
  resultado := 1;
```



Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	?
E	?
pot	?

Cuando comienza la ejecución del programa se tienen las variables B, E, y pot, las cuales aun no tienen valor.



(La traza siguen a continuación en la otra hoja) →

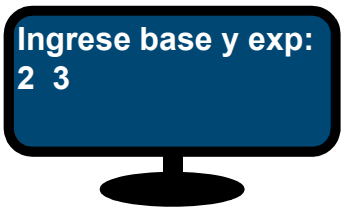
Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	?

llama a

potencia	

(1) Consideremos el caso de prueba donde el usuario ingresa los valores 2 y 3.
Cuando la ejecución llega a la asignación
pot:=potencia(B,E);
se llama a la función potencia con los valores de las variables B y E.



(La traza siguen a continuación en la otra hoja) →

Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	?

llama a

potencia	
base	2
exponente	3
aux	?
resultado	?

(2) Inmediatamente después de invocar a la función potencia aparecen en la traza los parámetros formales por valor (base y exponente) y las variables locales (aux y resultado) que son propias de la función.
El valor del parámetro efectivo B se copia al parámetro formal (por valor) base,
y el valor de E se copia a exponente.
De esta manera, los dos parámetros formales tendrán un valor inicial antes de ejecutar la primera sentencia de la función.
Observe , sin embargo, que las variables locales aux y resultado no tienen valor inicial.

(La traza siguen a continuación en la otra hoja) →

Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	?

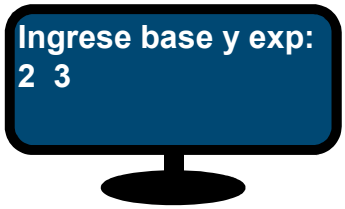
llama a

retorna 8

potencia	
base	2
exponente	3
aux	~~1,2~~,3
resultado	~~1,2,4~~,8

(3) Al ejecutarse la función potencia, la variable resultado inicia en 1. Como Aux es la variable de control de un FOR que se ejecuta 3 veces toma los valores 1, 2, y 3.
Por lo tanto, resultado toma los valores 2, 4 y 8.
La asignación Potencia:= resultado; tiene como efecto que la función retornará el valor de resultado, es esto, 8.

(La traza siguen a continuación en la otra hoja) →

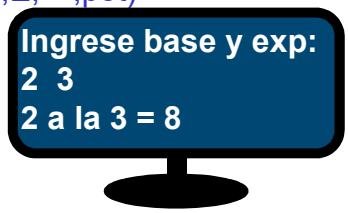


Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	8

(4) Al finalizar la ejecución de la función potencia, la ejecución del programa continúa en la sentencia pot:=potencia(B,E) de donde se había llamado a la función.
El resultado de la función (un 8) es asignado entonces a la variable pot.
Todas las variables de la función desaparecen (la memoria es liberada para que la use cualquier programa en ejecución).
Se ejecuta writeln(B,'a la',E,'=',pot)

(La traza siguen a continuación en la otra hoja) →



Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	8

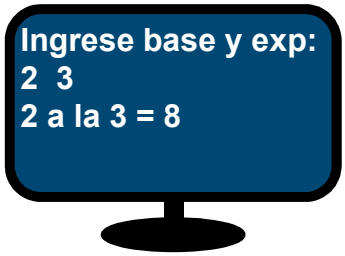
llama a

+1

potencia	
base	3
exponente	4
aux	?
resultado	?

(5) Cuando se ejecuta la sentencia
writeln('3 a la ', E+1, '=', potencia(3,E+1))
se llama a la función con los valores 3 y E+1.

(La traza siguen a continuación en la otra hoja) →



Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	8

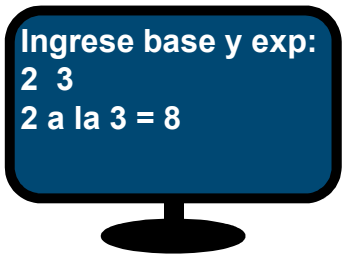
llama a

retorna 81

potencia	
base	3
exponente	4
aux	~~1,2,3~~,4
resultado	~~1,3,9,27~~,81

(6) Al ejecutarse la función potencia, la variable resultado inicia en 1. Como Aux es la variable de control de un FOR que se ejecuta 4. Por lo tanto, resultado toma los valores 3, 9, 27 y 81.
La asignación Potencia:= resultado; tiene como efecto que la función retornará el valor de resultado, es esto, 81.

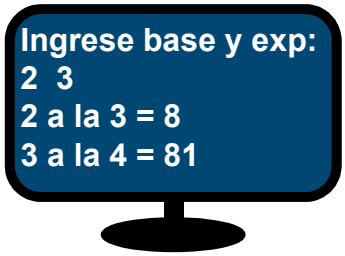
(La traza siguen a continuación en la otra hoja) →



Una forma de hacer la traza de Prueba_potencia

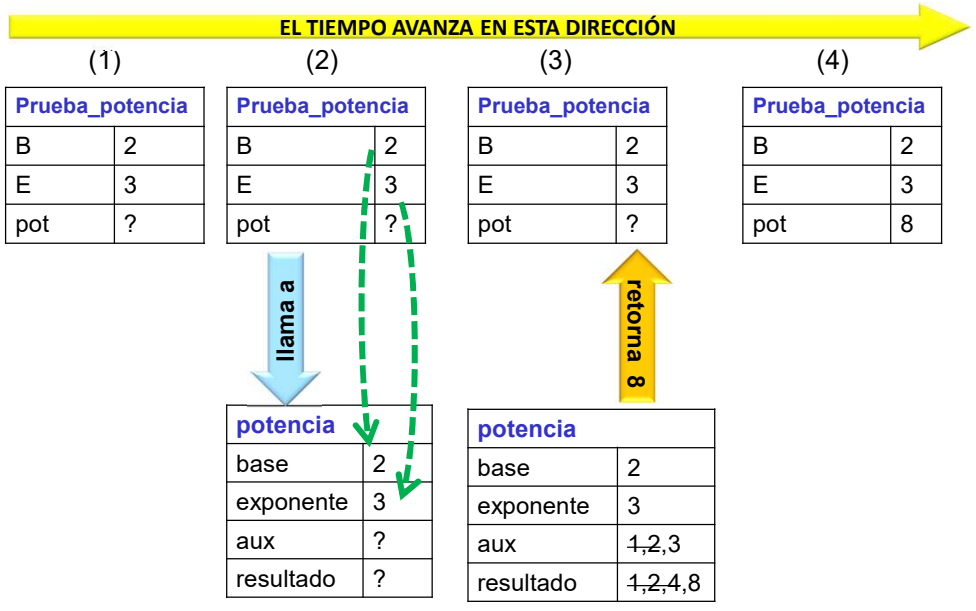
Prueba_potencia	
B	2
E	3
pot	8

(7) Al finalizar la ejecución de la función, la ejecución del programa continúa en la sentencia writeln('3 a la ', E+1, '=', potencia(3,E+1))
de donde se había llamado a la función.
El resultado de la función (81) es mostrado por pantalla.
La memoria usada por la función es liberada para que la use cualquier programa en ejecución).



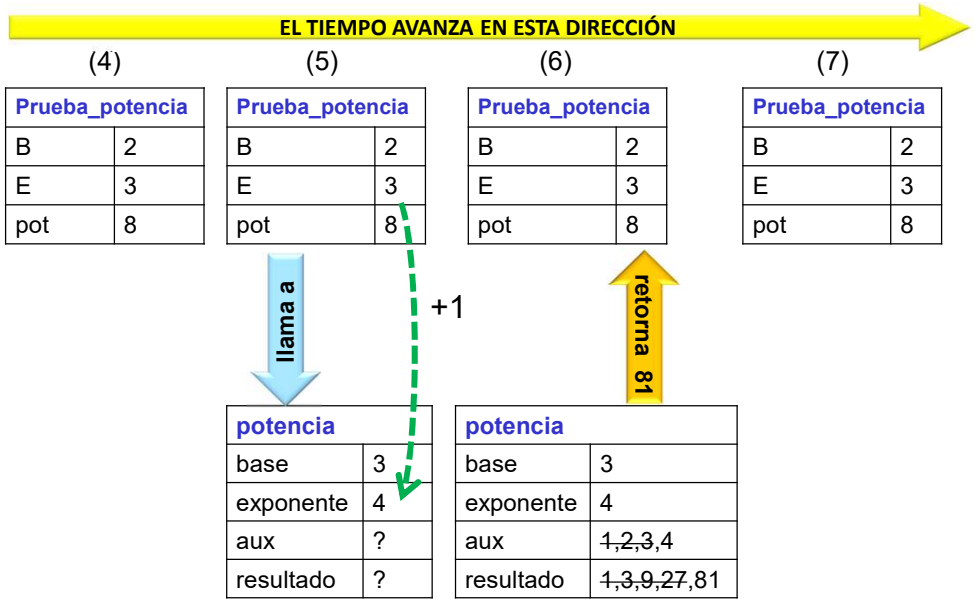
Resolución de Problemas y Algoritmos Dr. Diego R. García 25

Traza de Prueba_potencia



Resolución de Problemas y Algoritmos Dr. Diego R. García 26

Traza de Prueba_potencia



Resolución de Problemas y Algoritmos Dr. Diego R. García 27

Problema propuesto (lo más simple posible)

- Problema propuesto: Escribir un programa que solicite al usuario tres números entre 1 y 200: N1, N2, y N3. El programa deberá mostrar por pantalla todas las combinaciones de un número elevado con otro. Esto es, N1 elevado a la N2, N1 a la N3, N2 a la N3, N2 a la N1, N3 a la N1 y N3 a la N2.
- Con la función potencia puedo resolver una parte.
- Se puede también hacer una primitiva especial para la carga de datos: una función que lea y valide que el dato de entrada está entre los topes "menor" y "mayor". Por ejemplo:
  FUNCTION leer_y_validar(menor,mayor:integer):integer;
- De manera tal que pueda hacer llamadas de este estilo:

```
N1:=leer_y_validar(1,200);
N2:=leer_y_validar(1,200);
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 28

Función para leer y validar

```
FUNCTION leer_y_validar(menor,mayor:integer): integer;
{lee usando readln un valor del buffer de entrada hasta que el valor leído
esté entre los topes indicados como datos de entrada}
VAR aux: integer; // para leer los valores a ser validados
BEGIN
  repeat
     write('Ingrese entero entre ', menor, ' y ', mayor,':');
     readln(aux);
  until (aux>=menor) and (aux<=mayor);
  leer_y_validar:=aux; // retorno el valor leído y validado
END;
```



```
PROGRAM problema2;
CONST tope_inf=1; tope_sup=200;
VAR n1,n2,n3: integer;
  FUNCTION Potencia(Base,Exponente:integer) : integer;
  …
  FUNCTION leer_y_validar(menor,mayor:integer): integer;
  …
BEGIN
N1:=leer_y_validar( tope_inf, tope_sup);
N2:=leer_y_validar(tope_inf, tope_sup);
N3:=leer_y_validar(tope_inf, tope_sup);
write(N1,‘ a la ’,N2,’ es ’,potencia(N1,N2)  );
write(N1,‘ a la ’,N3,’ es ’,potencia(N1,N3)  );
write(N2,‘ a la ’,N1,’ es ’,potencia(N2,N1)  );
write(N2,‘ a la ’,N3,’ es ’,potencia(N2,N3)  );
write(N3,‘ a la ’,N1,’ es ’,potencia(N3,N1)  );
write(N3,‘ a la ’,N2,’ es ’,potencia(N3,N2)  );
END.
```

Hacer una traza

Funciones en Pascal

A continuación se muestra un algoritmo para identificar si una letra es mayúscula y una implementación con una función.

Algoritmo: esMayuscula
Datos que recibe: un carácter
Dato que retorna: verdadero/falso
si es una letra mayúscula retornar verdadero
de lo contrario retornar falso

```
FUNCTION EsMayuscula (letra :char): boolean;
{retorna verdadero si es letra mayúscula o false en caso contrario}
BEGIN   IF ('A' <= letra) and (letra <= 'Z' ) or (letra = chr(165)) // Ñ
          THEN EsMayuscula:=true
          ELSE EsMayuscula:=false
END;
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 31

Invocación a funciones (en expresiones)

```
FUNCTION EsMayuscula (letra :char): boolean;
{retorna verdadero si es letra mayúscula o false en caso contrario}
BEGIN   IF ('A' <= letra) and (letra <= 'Z' ) or (letra = chr(165)) // Ñ
          THEN EsMayuscula:=true
          ELSE EsMayuscula:=false
END;
```

Algunos ejemplos de invocación (siempre en expresiones):

```
VAR ch: char;   es_mayu:boolean;
 ...
read(ch);
es_mayu := EsMayuscula(ch);
writeln(EsMayuscula(ch));
IF (EsMayuscula(ch) or (ch='@'))
    THEN …
WHILE not EsMayuscula(ch) and not  EOF(…)  DO …
```

Resolución de Problemas y Algoritmos Dr. Diego R. García 32

Conceptos: invocación a funciones

Para llamar (invocar o usar) a una función se debe:

1) Utilizar la llamada en una expresión.
2) Coincidir la cantidad de parámetros; y el tipo de dato de cada uno de los parámetros efectivos debe ser asignación compatible con el parámetro formal correspondiente.
3) El tipo del resultado debe ser compatible con el tipo de la expresión en la que se lo llama.

Por ejemplo, considerando:

```
FUNCTION Potencia(Base,Exponente:integer) : integer;
```

todas estas llamadas son correctas:


```
aux:=Potencia(2,7);
write(Potencia(2,7));
write(potencia(1+1, 14 div 2));
if potencia(2,7) > tope then…
```


Conceptos: invocación a funciones

Para llamar (invocar o usar) a una función se debe:

1) Utilizar la llamada en una expresión.
2) Coincidir la cantidad de parámetros; y el tipo de dato de cada uno de los parámetros efectivos debe ser asignación compatible con el parámetro formal correspondiente.
3) El tipo del resultado debe ser compatible con el tipo de la expresión en la que se lo llama.

Dado `FUNCTION Potencia(Base,Exponente:integer) : integer;`

todas estas llamadas tienen errores de programación.


```
Potencia(2,7);
write(potencia(2.1,3));
write(Potencia(2));
write(potencia(2,3,1));
write(potencia('x',2));
if potencia(2,7) then…
```

- Error: no está en una expresión
- Error: el primer parámetro es de tipo real
- Error: falta un parámetro
- Error: sobra un parámetro
- Error: parámetro no compatible
- Error: tipo del resultado no compatible

Conceptos: retornar el valor de una función

- En toda función, es necesario que al menos una vez se ejecute una **asignación** que en su parte izquierda tenga el **nombre** de la función.
- Es la forma que tiene la función de retornar un valor.
- Si esto no ocurre se considera **un error de programación.**
- A continuación se incluyen ejemplos correctos y otros con errores.

```
FUNCTION Cubo (N:integer):integer;
BEGIN
Cubo:= N*N*N;
END;
```

CORRECTO

```
FUNCTION Cubo (N:integer):integer;
VAR aux: integer;
BEGIN
aux:= N*N*N;
END;
```

MAL

INCORRECTO: error de programación, falta la asignación de valor a la función.

Resolución de Problemas y Algoritmos Dr. Diego R. García 35

Conceptos: retornar el valor de una función

- En toda función, es necesario que al menos una vez se ejecute una **asignación** que en su parte izquierda tenga el **nombre** de la función.

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
    IF ('A' <= letra) and (letra <= 'Z' )
    THEN EsMayuscula:=true   ELSE EsMayuscula:=false
END;
```

CORRECTO

Error de programación: cuando no es mayúscula no se ejecuta la asignación de valor a la función. Ejemplo de prueba: 'a'.

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
    IF ('A' <= letra) and (letra <= 'Z' )
      THEN EsMayuscula:=true
END;
```

MAL

```
function esMayuscula (letra :char): boolean;
begin
 esMayuscula:= ('A' <= letra) and (letra <= 'Z' )
end;
```

CORRECTO

Importante: si existe aunque sea un solo caso de prueba para el cual la función no retorna valor, entonces hay un error de programación.

Resolución de Problemas y Algoritmos Dr. Diego R. García 36

Conceptos: retornar el valor de una función

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
    EsMayuscula:=false;
     IF ('A' <= letra) and (letra <= 'Z' )
     THEN EsMayuscula:=true
END;
```

CORRECTO

Puede haber muchas asignaciones que dan valor a la función.
Si se ejecuta más de una, la función retornará el valor asignado por la última en ejecutarse.

```
FUNCTION dia_mes (mes,anio :integer): integer;
BEGIN  {retorna la cantidad de días de un mes}
    dia_mes:=0;
   CASE  MES OF
     11,4,6,9: dia_mes :=30;
     2: IF EsBisiesto(anio) // llama a la función "EsBisiesto"
        THEN dia_mes := 29  ELSE dia_mes := 28;
     1,3,5,7,8,10,12: dia_mes :=31;
   END;
END;
```

CORRECTO

Importante: el nombre de una función NO ES UNA VARIABLE

Resolución de Problemas y Algoritmos Dr. Diego R. García 37

Conceptos: retornar el valor de una función

Importante: el **nombre de una función no es una variable**. Algunas veces se crea una confusión porque se le "asigna" un valor. Pero no puede usarse como una variable.

```
function esMayuscula (c :char): boolean;
begin
 esMayuscula:= ('A' <= c) and (c <= 'Z' )
end;
```

```
program …
…
resultado:= esMayuscula(ch);
IF resultado = true
then ….
```

identificador := expresión

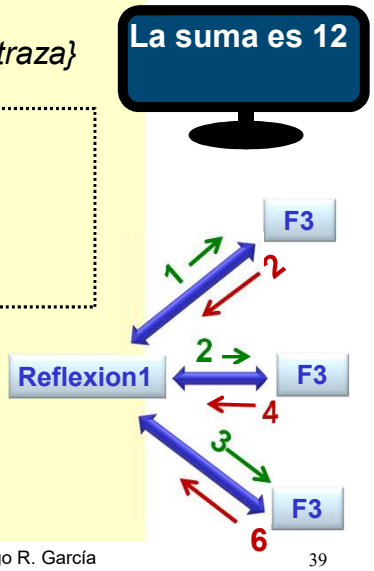
El nombre de una función a la izquierda del := se usa para darle valor a la función.

El nombre de una función usado en la expresión a la derecha del := es usado para llamar a la función

Resolución de Problemas y Algoritmos Dr. Diego R. García 38

Para reflexionar sobre el pasaje de parámetros, las variables locales y globales, con respecto a la dinámica en la ejecución realice la traza de los siguientes programas:

```
program Reflexion1;
 {Para reflexionar sobre la dinámica con una traza}
var i, tope, suma:integer;
    Function F3   (N:integer):integer;
    var local:integer; {F3 modifica local y N}
     begin
     local:= N; N:= N + local; F3:= N;
    end;
begin
tope:= 3;
Suma:=0; {F3 es llamada 3 veces}
for i:=1 to tope do suma:=suma+F3(i);
WriteIn('La suma es ', suma);
end.
```



Resolución de Problemas y Algoritmos Dr. Diego R. García 39

Una forma de hacer la traza de reflexión1

Reflexión1	
i	1
Tope	3
suma	0

llama a F3 — retorna 2

F3	
N	~~1~~, 2
local	1

(1) Reflexión1 llama a F3 y se copia el valor de i en el parámetro N

Reflexión1	
i	1
Tope	3
suma	2

(2) Al terminar la ejecución de F3 desaparecen las variables de F3, el programa recibe el valor 2 y suma es 2.

Reflexión1	
i	2
Tope	3
suma	2

llama a F3 — retorna 4

F3	
N	~~2~~, 4
local	2

(3) En Reflexión1 se incrementa i en 1, se llama nuevamente a F3 (N toma valor 2)

Reflexión1	
i	2
Tope	3
suma	6

(La traza siguen a continuación en la otra hoja) →

(4) Al terminar la segunda ejecución de F3, desaparecen las variables de F3, el programa recibe un 4 y suma es 2+4

EL TIEMPO AVANZA EN ESTA DIRECCIÓN

Resolución de Problemas y Algoritmos Dr. Diego R. García 40

(continuación de la traza)

Reflexión1	
i	3
Tope	3
suma	6

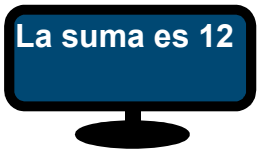
llama a F3

retorna 6

F3	
N	~~3~~, 6
local	3

(5) En Reflexión1 se incrementa i en 1, se llama nuevamente a F3 (se vuelven a crear las variables N y local, y el parámetro N toma valor 3)

Reflexión1	
i	3
Tope	3
suma	12

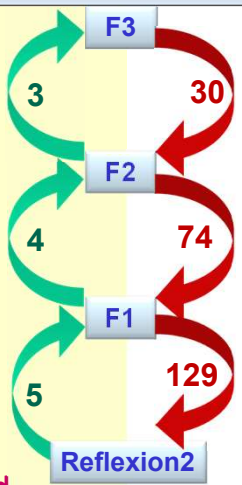


(6) Al terminar la tercera ejecución de F3, desaparecen las variables de F3, el programa recibe el valor 6 y suma es ahora 6+6 =12. Cómo i ya tiene el valor de tope (3) deja de repetir y muestra 12 en pantalla.

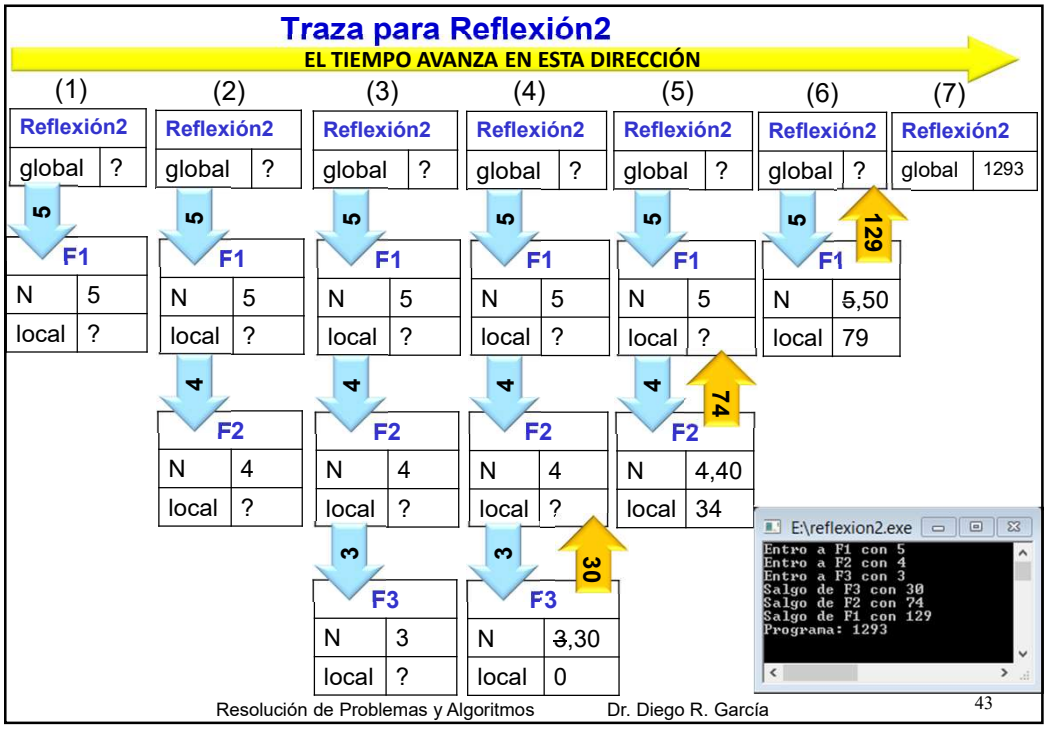
EL TIEMPO AVANZA EN ESTA DIRECCIÓN

Resolución de Problemas y Algoritmos Dr. Diego R. García 41

```
program reflexion2; var global:integer;
  Function F3 (N:integer):integer;
  var local:integer;
   begin writeln('Entro a F3 con ', N);
   local:= 0; N:= N * 10; F3:= N + local;
   writeln('Salgo de F3 con ', N + local); end;
  Function F2 (N:integer):integer;
  var local:integer;
   begin writeln('Entro a F2 con ', N);
   local:= N+ F3( N - 1); N:= N * 10; F2:= N + local;
   writeln('Salgo de F2 con ', N + local); end;
  Function F1 (N:integer):integer;
  var local:integer;
   begin writeln('Entro a F1 con ', N);
   local:=N+F2( N - 1); N:= N * 10; F1:= N + local;
   writeln('Salgo de F1 con ', N + local); end;
begin global:=3+F1(5)*10; write('Programa:',global); end.
```



42



Archivos como parámetros

- **Las funciones pueden recibir datos de tipo FILE (archivos) como parámetros. Ejemplos:**

  IF Pertenece_elemento(E, archivo1) then …

- En Pascal, es **obligatorio** que un **parámetro** de **tipo archivo** sea un parámetro **por** referencia.
- Los parámetros por referencia se indican en el código fuente usando la palabra reservada VAR antes del nombre del parámetro.
- Los parámetros por referencia deben ser de tipos idénticos, por lo tanto debe definirse un tipo nuevo para pasar archivos como parámetros: TYPE <identificador> FILE OF…

Repaso: compatibilidad entre parámetros

En un parámetro POR REFERENCIA, el tipo del parámetro formal debe ser idéntico al tipo del parámetro efectivo. Estos es, se cumple que:

a) Están declarados con el mismo identificador de tipo.

b) Los identificadores de tipo son diferentes (ej: `T1` y `T2`) pero han sido definidos como equivalentes por una declaración de la forma `T1 = T2.`

De esta forma es incorrecto:

```
VAR F1: FILE OF Integer;
...
FUNCTION ejemplo( VAR A: FILE OF Integer):boolean;
...
IF ejemplo(F1);
```

MAL

A y F1 NO SON DE TIPOS IDÉNTICOS

Resolución de Problemas y Algoritmos Dr. Diego R. García 45

Repaso: compatibilidad entre parámetros

En un parámetro POR REFERENCIA, el tipo del parámetro formal debe ser idéntico al tipo del parámetro efectivo. Estos es, se cumple que:

a) Están declarados con el mismo identificador de tipo.

De esta forma es correcto:

```
TYPE  TipoArchi = FILE OF Integer;
VAR F1: TipoArchi
...
FUNCTION ejemplo( VAR A: TipoArchi ):boolean;
...
IF ejemplo(F1);
```

BIEN

Ahora A y F1 si son de tipos idénticos

Resolución de Problemas y Algoritmos Dr. Diego R. García 46

Continuará …