

Resolución de Problemas y Algoritmos

Clase 12:
Funciones definidas por el programador.



Ada Byron King



Dr. Alejandro J. García

<http://cs.uns.edu.ar/~ajg>



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

Sobre el trabajo profesional futuro

Como profesional, un informático debe tener la capacidad de resolver problemas utilizando computadoras. Estos problemas pueden ser de:

- ✓ **muy gran escala:** como por ejemplo *mantener en órbita a la Estación Espacial Internacional (ISS)*.
- ✓ **gran escala:** por ejemplo desarrollar un sistema de reserva y venta de pasajes de una compañía aérea.
- ✓ **pequeña escala:** como por ejemplo *validar la identidad de un usuario mediante su nombre y clave; o controlar que una fecha ingresada en un formulario de una página web sea correcta.*

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 2

Reflexione un momento...

- ¿Cómo será el trabajo profesional si hay que desarrollar software a gran escala?
- ¿Trabajaré en grupo?
- ¿Tendré a cargo una parte?
- ¿Me relacionaré con otros?
- ¿Tardaré varios días en hacer parte ?

• Los lenguajes de programación evolucionaron para permitir que los programadores puedan hacer sus propias primitivas, compartir y “re-utilizar” el código lo más posible y de la mejor manera posible.

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 3

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.

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 4

Funciones y procedimientos predefinidos

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

```

...
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 como sentencias (procedimientos predefinidos)

primitivas en expresiones (funciones predefinidas)

¿Quién creó el código de estas primitivas predefinidas?

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 5

Funciones en Pascal

Ejemplos de algunas funciones predefinidas :

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

Reflexionemos sobre 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. Alejandro J. García 6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 “Resolución de Problemas y Algoritmos. Notas de Clase”. Alejandro J. García. Universidad Nacional del Sur. (c) 2015

Procedimientos en Pascal

Ejemplos de algunos procedimientos predefinidos en Pascal:

- **assign** (F, 'nombre');
- **reset** (F);
- **read** (A, B, C, D);
- **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. Alejandro J. García 7

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. Alejandro J. 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	Cuadrado (square)	real o integer	El mismo tipo recibido
sqrt	square root (raiz cuad.)	real o integer	real
trunc	truncado	real o integer	integer

http://wiki.freepascal.org/Standard_Functions

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 9

Funciones predefinidas para reales/enteros en Pascal

Función	Recibe	Retorna
abs	real o integer	mismo tipo recibido
arctan	real o integer	real
cos	real o integer	real
exp	real o integer	real
ln	real o integer	real
round	real	integer
sin	real o integer	real
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 sqr(num) < 100
then resultado:=sqr(num);
```

Hemos hecho un programa para hacer la potenciación (base exponente) pero la idea sería poder hacer esto:

```
resultado:=potencia(num,4);
```

Veremos a continuación como construir una función "potencia" que puede ser usada en cualquier expresión del programa .

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 10

Volviendo atrás, en la clase 5 teníamos....

```
PROGRAM CalculoPotencia;
{Calcula un número natural elevado a una potencia (también natural)}
VAR base, exponente, potencia, aux: integer;
BEGIN
writeln('Ingrese base y exp : ');
readln(base, exponente);
potencia := 1;
FOR aux := 1 TO exponente
DO potencia:=potencia * base;
write(base, ' a la ', exponente);
writeln(' es ', potencia);
END.
```

Base	exponente	aux	potencia
?	?	?	?
2	7	1	2
			2
			4
			8

Ingrese base y exp:
2 3
2 a la 3 es 8

A continuación se mostrará como construir una función "potencia" para ser usada como una primitiva en un programa.

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 11

Función para la potenciación

Una primitiva para la potenciación tendría dos datos de entrada (base y exponente) y retornaría como resultado la potencia, multiplicando exponente veces el valor de base.

Algoritmo: potencia
Datos de entrada: base y exponente
Dato de salida: resultado
 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 i:= 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. Alejandro J. García 12

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
"Resolución de Problemas y Algoritmos. Notas de Clase". Alejandro J. García. Universidad Nacional del Sur. (c) 2015

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 de entrada: base y exponente
Dato de salida: resultado
resultado:=1;
Repetir exponente veces: resultado:=resultado*base.

```

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 13

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.

```

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. 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:');
  readln(B,E);
  Pot:=Potencia(B,E);
  writeln(B,'a la ',E,'=',pot);
  writeln('2 a la ', E+1, ' =', potencia(2,E+1));
END.
    
```

Para hacer la traza vea la próxima explicación

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 15

Conceptos: parámetros formales y efectivos

```

PROGRAM Prueba_potencia;
VAR B,E, Pot :Integer;

FUNCTION Potencia (Base, Exponente:integer) : integer;
{utilidad: calcula "base" 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:');
  readln(B,E);
  Pot:=Potencia(B,E);
  writeln(B,'a la ',E,'=',pot);
  writeln('2 a la ', E+1, ' =', potencia(2,E+1));
END.
    
```

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 16

Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	?

(1) Cuando comienza la ejecución del programa se crean las variables B, E, y pot. 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 al función potencia.

Ingrese base y exp:
2 3

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 17

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 se crean los parámetros formales por valor (base y exponente) y las variables locales (aux y resultado) que son propias de la función. El valor de B se copia a base y el valor de E se copia a exponente. De esta manera los dos parámetros formales tienen valor inicial antes de ejecutar la primera sentencia de la función. Observe, sin embargo, que aux y resultado no tienen valor inicial antes de comenzar con la ejecución de las sentencias de la función.

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 18

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 "Resolución de Problemas y Algoritmos. Notas de Clase". Alejandro J. García. Universidad Nacional del Sur. (c) 2015

Una forma de hacer la traza de Prueba_potencia

Prueba_potencia	
B	2
E	3
pot	?

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

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

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 19

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, todas las variables de la función desaparecen (la memoria es liberada para que la use cualquier programa en ejecución). La ejecución continúa en el punto que había quedado en el programa prueba_potencia, esto es, en la asignación `pot:=potencia(B,E)`. El resultado de la función, un 8, es asignado entonces a la variable pot, y se muestran los valores por pantalla.

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 20

Traza de Prueba_potencia

EL TIEMPO AVANZA EN ESTA DIRECCIÓN →

Prueba_potencia (1)	
B	2
E	3
pot	?

Prueba_potencia (2)	
B	2
E	3
pot	?

Prueba_potencia (3)	
B	2
E	3
pot	?

Prueba_potencia (4)	
B	2
E	3
pot	8

Diagram showing the flow of control between the main program and the potencia function across four steps. A yellow arrow indicates the progression of time. A blue arrow labeled 'llama a' points from step (2) to the potencia function table. A green arrow points from the potencia function table back to step (3). A yellow arrow labeled 'retorna 8' points from step (3) to step (4). The potencia function table shows its internal state: base=2, exponente=3, aux=?, resultado=?.

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 21

Problema propuesto (lo más simple posible)

- Problema2: 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 al 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 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. Alejandro J. García 22

```

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

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 23

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;
    
```

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 24

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 "Resolución de Problemas y Algoritmos. Notas de Clase". Alejandro J. García. Universidad Nacional del Sur. (c) 2015

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 de entrada: un carácter
Dato de salida: 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. Alejandro J. García 25

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. Alejandro J. García 26

Conceptos: invocación a funciones

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

- Utilizar la llamada en una **expresión**.
- Coincidir la **cantidad de parámetros** y el **tipo de dato** de cada uno de los parámetros.
- 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 incorrectas y todas son consideradas errores de programación.
    
```

MAL

Potencia(2,7); ← **Error: no está en una expresión**

write(Potencia(2)); ← **Error: falta un parámetro**

write(potencia('x',2)); ← **Error: parámetro no compatible**

if potencia(2,7) then... ← **Error: tipo del resultado no compatible**

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 27

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);
  Writeln('La suma es ', suma);
end.
    
```

Resolución de Problemas y Algoritmos Dr. Alejandro J. García 28

Una forma de hacer la traza de reflexión1

Reflexión1	
i	1
Tope	3
suma	0

Reflexión1	
i	1
Tope	3
suma	2

Reflexión1	
i	2
Tope	3
suma	2

Reflexión1	
i	2
Tope	3
suma	6

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

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

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

(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. Alejandro J. García 29

(continuación de la traza)

Reflexión1	
i	3
Tope	3
suma	6

Reflexión1	
i	3
Tope	3
suma	12

(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)

(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. Alejandro J. García 30

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
“Resolución de Problemas y Algoritmos. Notas de Clase”. Alejandro J. García. Universidad Nacional del Sur. (c) 2015

```

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('progama:',global); end.
    
```

Traza para Reflexión2
EL TIEMPO AVANZA EN ESTA DIRECCIÓN

	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Reflexión2							
global	?	?	?	?	?	?	129
F1							129
N	5	5	5	5	5	5,50	
local	?	?	?	?	?	79	
F2							79
N	4	4	4	4	4,40		
local	?	?	?	?	34		
F3							30
N	3	3	3,30				
local	?	?	0				

Ada Augusta Byron King (1815-1852)

Lady Ada Lovelace, era matemática, y se enteró de los esfuerzos de Babbage y se interesó en su máquina analítica. Promovió activamente la máquina y escribió varios programas.

Diferentes historiadores concuerdan en que fue la primera programadora de computadoras de la historia.

Escribió el primer algoritmo para una computadora, el cuál contenía dos bucles, y otro que usaba variables.

En este momento se reconoce a Ada Byron como la primera persona en describir un lenguaje de programación, y la madre de la programación informática.

http://es.wikipedia.org/wiki/Ada_Lovelace

Lady Ada Augusta Byron King, condesa de Lovelace

Ada también preveía la capacidad de las computadoras para ir más allá de los simples cálculos de números. Mientras que otros, incluido el propio Babbage, se centran únicamente en las capacidades numéricas.

Su trabajo fue olvidado por muchos años, atribuyéndole el papel de transcritora de las notas de Babbage. Sin embargo, sus notas sobre programación y su idea de usar tarjetas perforadas para introducir programas, fueron aprovechadas mucho después.

Continuará ...

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 “Resolución de Problemas y Algoritmos. Notas de Clase”. Alejandro J. García. Universidad Nacional del Sur. (c) 2015