Resolución de Problemas y Algoritmos

Clase 12
Procedimientos y Funciones en Pascal
Resolución de problemas por división y composición: construcción de primitivas.

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

Solución 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 complejos.**
2. **Cómo escribir algoritmos basados en primitivas y algoritmos que son primitivas.**
3. **Cómo implementar en Pascal primitivas y poder usar las dos técnicas anteriores.**

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

Funciones y procedimientos predefinidos

Al programar en Pascal usamos primitivas predefinidas.
Por ejemplo:

```
...
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;
...
```


Algunas Funciones predefinidas

Funciones predefinidas:

- **Se utilizan en una expresión.**
- **Siempre retornan un valor de un tipo de Pascal.**

Ejemplos:

- **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**

¿Puedo construir mis propias nuevas funciones?

Algunos procedimientos predefinidos

Procedimientos predefinidos:

- **Se los usa en una sentencia.**
- **Pueden tener 0 o más parámetros.**

Ejemplos:

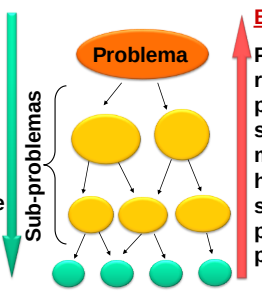
- **Writeln**
- **Readln**
- **reset (F)**
- **Rewrite (F)**
- **Assign (F, nombre)**

¿Puedo construir nuevos procedimientos?

Conceptos: técnicas "top-down" y "bottom-up"

Top-down:

División del problema principal en subproblemas más simples hasta llegar a problemas que no necesitan dividirse.



Bottom-up:

Por composición, resolviendo primero los subproblemas más simples hasta llegar a solucionar al problema principal.

Ejemplo de trabajo

Problema: Escriba un programa para calcular la suma hasta el término K-ésimo, donde en cada término de posición N, el numerador es 2^N y el denomidador N! Por ejemplo para K = 6

suma = $\underbrace{2/1 + 4/2 + 8/6 + 16/24 + 32/120 + 64/720}_{K=6}$

Solución: sumar (2^N / N!) desde N=1 hasta N=k

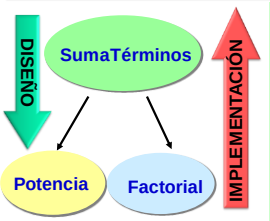
En Pascal, ¿tengo una primitiva para N! (factorial)?

¿tengo una primitiva para 2^N (potencia) ?

Como programador puedo construir nuevas primitivas.

Técnica: Para escribir el algoritmo y el programa se sugiere descomponer el problema en sub-problemas.

Descomposición del problema y primitivas

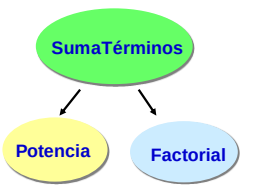



```
PRIMITIVA SumaTerminos
Dato Entrada : K {cant. términos }
Dato Salida : Suma
COMIENZO
  i ← 0    Suma ← 0
 Repetir K veces
  i ← i + 1
  Suma←Suma + Potencia(2,i)
                Factorial(i)
FIN ALGORITMO
```

```
PRIMITIVA Potencia
Datos Entrada : Base, Expo
Datos Salida : Pot
… calcula Base a la Expo…
```

```
PRIMITIVA Factorial
Datos Entrada : N {natural}
Datos Salida : Factorial de N
… calcula factorial de N…
```


Descomposición del problema y primitivas




```
Program suma;
   Potencia
   Factorial
   SumaTérminos
Begin
End.
```

PRIMITIVAS en Pascal:

1. **FUNCTION**
2. **PROCEDURE**


```
PROGRAM sumaK; {suma los primeros K términos}
VAR K: integer;
FUNCTION FACTORIAL (N:integer): integer;
… completar ….
FUNCTION POTENCIA (B,E:integer):integer;
… completar….
FUNCTION SumaTerminos ( tope:integer ) : REAL ;
VAR i: integer; suma: real;
BEGIN
  suma:= 0;
  FOR i:=1 TO tope DO
     suma:=suma+ potencia(2,i) / factorial(i) ;
  sumaTerminos:=suma;
END;
BEGIN
writeln('ingrese cantidad de términos'); readln(K);
writeln('la suma es: ', sumaTerminos(K));
END.
```

Variables globales

Parámetros formales

Tipo del Resultado

Variables Locales

Asignación del resultado

Llamada a la función

Tarea (muy importante) propuesta

- Complete el programa suma K y pase en la máquina.
- Realice algunas trazas en papel y luego ejecute para ver como funciona en la máquina.
- Puede poner algunos "writeln" para ver como se van llamando las funciones y como se modifican las variables.
- Por ejemplo:

writeln('ingreso a la función potencia');

writeln('salgo de sumaTermino con:', suma, tope);

En Pascal toda 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 usarla)
4. Variables locales (para uso interno solamente).
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;
VAR aux,P: integer;
BEGIN
   P := 1;
   FOR aux:= 1 TO Exponente DO P := P * Base;
   Potencia:= P;
END;
```

Para que una función retorne un valor, se debe ejecutar al menos 1 vez, una asignación que en su parte izquierda tenga el nombre de la función y a la derecha una expresión del mismo tipo que la función.

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

CORRECTO

```
FUNCTION Cubo (N:integer):integer;
VAR aux,P: integer;
BEGIN
  P := 1;
  FOR aux:= 1 TO 3 DO P := P * N;
  Cubo:= P;
END;
```

CORRECTO

Si no hay una asignación de ese tipo, o ninguna se ejecuta, entonces es un error de programación.

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

INCORRECTO (falta asignación de valor a la función)

```
FUNCTION Cubo (N:integer):integer;
VAR aux,P: integer;
BEGIN
  Cubo := 1;
  FOR aux:= 1 TO 3
     DO Cubo := Cubo * N;
END;
                    X
```

INCORRECTO ("Cubo" no es una variable)

Funciones en Pascal

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

1) Utilizarse su nombre en una expresión.

2) Coincidir la cantidad de parámetros y tipo de cada uno.

3) El tipo del resultado debe coincidir con el tipo de la expresión en la que se lo llama.

Invocación a funciones (en expresiones)

```
FUNCTION EsVocal (letra :char): boolean;
BEGIN
    CASE letra OF
     'A','E','I','O','U','a','e','i','o','u', : EsVocal:=true;
    ELSE EsVocal:=false;
    END;
END;
```

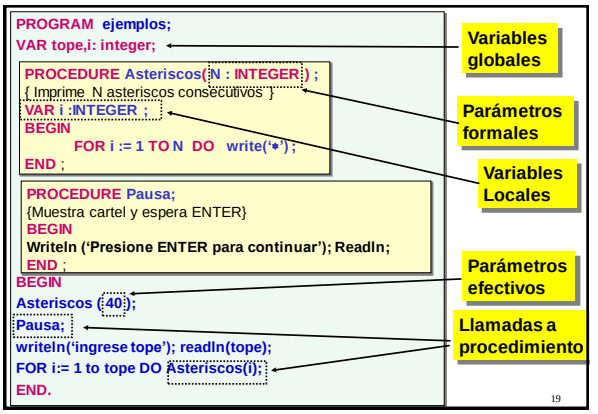
Algunos ejemplos de invocación:

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


PROCEDIMIENTOS (PROCEDURE)

- **En Pascal, además de las funciones, se pueden construir primitivas como "procedimientos".**
- **Su invocación se realiza como una sentencia y no desde una expresión.**
- **Pueden tener un número cualquiera de datos de entrada (cero o más), y un número cualquiera de datos de salida (cero o más).**
- **No tienen un tipo asociado, ni retornan obligatoriamente un valor.**
- **Ejemplo:**

```
PROCEDURE Pausa;
{Muestra un mensaje y espera ENTER}
BEGIN
Writeln ('Presione ENTER para continuar'); Readln;
END ;
```



Conceptos: diferencias entre…

Funciones

- Se invocan desde una expresión
- Al regresar de la invocación se sigue ejecutando la sentencia de la llamada.
- Tiene un tipo asociado
- Aunque no tenga parámetros devuelve un valor que se usa en la expresión que la llama.

Procedimientos

- Se invocan como una sentencia.
- Al regresar de la invocación se ejecuta la sentencia siguiente a la llamada.

Ejemplo

- **Problema: Escriba una primitiva para multiplicar dos fracciones.**
- **En Pascal podemos representar una fracción como su numerador y denominador en forma separada y construir una primitiva multiplicar fracciones multiplicando los numeradores y luego los denominadores.**
- **4 datos de entrada: los 2 numeradores y los 2 denominadores.**
- **2 datos de salida: el numerador y el denominador resultado.**

```
NumRes := Num1 * Num2
DenRes := Den1 * Den2
```


Procedimientos y parámetros

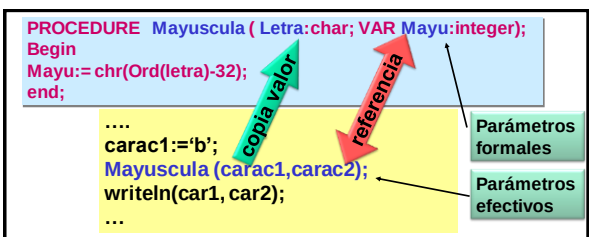
```
PROCEDURE MultiplicarFracciones
        (Num1, Den1, Num2, Den2 : INTEGER;
         VAR NumRes,DenRes: INTEGER);
BEGIN
   NumRes := Num1 * Num2;
   DenRes := Den1 * Den2;
END;
```

Parámetros por valor

Parámetros por referencia

Los parámetros formales pueden ser:

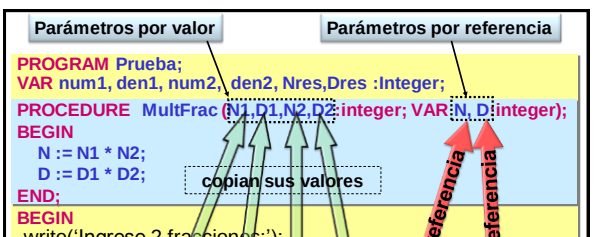
- **por valor: sólo permiten recibir valores y se los utiliza para entrada de datos.**
- **por referencia: cuando se anteponen la palabra VAR. En este caso, se crea una referencia con el parámetro efectivo, y por lo tanto, permite salida de datos.**



Diferencias entre los tipos de parámetros formales:

P. por valor: el valor del parámetro efectivo Carac1 se copia al formal correspondiente (Letra),las modificaciones a Letra no afectan a la variable Carac1.

P. por referencia: se crea un referencia entre Carac2 y Mayu. Todo cambio en Mayu afecta y cambia a Carac2.



Sugerencia: pase a la máquina y ejecute.