Tecnologías en la Educación Matemática

Procedimientos y funciones

Dpto. de Ciencias e Ingeniería de la Computación
UNIVERSIDAD NACIONAL DEL SUR

Los sub-programas (primitivas)

- Bloques de código que llevan a cabo una tarea concreta
  (resuelven un sub-problema concreto)
- Tienen un propósito bien definido.
- Permiten reutilizar código de manera sencilla y segura: pueden ser usados más de una vez en el programa principal sin necesidad de reescribir todo (o *copiar-pegar*).
- Pascal permite definir 2 tipos de sub-programas: Funciones y Procedimientos

2

Funciones y Procedimientos

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 siempre 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.
- Pueden devolver muchos valores o ninguno

3

Procedimientos sin datos de salida

Problema: Dibujar triángulo como la que se muestra a continuación, donde la cantidad de filas depende de un entero *filas*.

```
1
22
333
4444
…
```

```
procedure dibujarTriangulo(filas: integer);
 var i, j: integer;
 begin
    for i:=1 to filas do
        begin
          for j:=1 to i do
              write(i);
          writeln;
        end;
 end;
```

4

Procedimientos

Dato de entrada = parámetro por valor

Variables locales

Se calcula el valor de **n** (parámetro real) y ese valor es asignado a **filas** (parámetro formal).

```
program piramide;
var n: integer;
 procedure dibujaTriangulo(filas: integer);
 var i, j: integer;
 begin
    for i:=1 to filas do
        begin
          for j:=1 to i do
              write(i);
          writeln;
        end;
 end;
begin
writeln('Ingrese la cantidad de filas de la piramide: '); readln(n);
dibujaPiramide(n);
end.
```

5

Procedimientos y Funciones

Escribir un procedimiento que dado un número entero, retorne el número anterior y el número siguiente.

```
procedure sigant(n: integer; var na, ns: integer);
begin
   na := n – 1;
   ns := n + 1;
end;
```

6

Procedimientos y Funciones

```
procedure sigant(n: integer; var na, ns: integer);
begin
   na := n – 1;
   ns := n + 1;
end;
```

El procedimiento *sigant* tiene tres parámetros.

El parámetro n está pasado por valor, cuando empieza la ejecución del procedimiento, se reserva una locación de memoria para n, que se inicializa con el valor del parámetro real y se destruye cuando termina el procedimiento.

Los parámetros na y ns están pasados por variable (también llamados por referencia), son **referencias** a las locaciones de memoria que corresponden a los parámetros reales. También se reservan locaciones de memoria para estas referencias.

Procedimientos y Funciones

```
function doble(n: integer): integer;
begin
  doble := n * 2;
end;
```

La función doble tiene un único parámetro, pasado por valor.

La función computa un valor y lo asigna a su nombre.

Procedimientos y Funciones

```
Program pyf;
var a, b, c, d: integer;
…
Begin
   write ('Ingrese dos números ');
   readln(a, d);
   sigant(a, b, c);  {1}
   a := doble(b) + doble(c);  {2}
   sigant(a, c, b);  {3}
End.
```

Procedimientos y Funciones

```
procedure sigant (n: integer; var na, ns: integer);
begin
   na := n – 1;
   ns := n + 1;
end;
```

sigant(a, b, c); {1}

La primera invocación de sigant asigna el valor del parámetro real a al parámetro formal n.

El parámetro real b queda ligado al parámetro formal na, cualquier modificación en na, afecta al valor almacenado en b.

El parámetro real c queda ligado al parámetro formal ns, cualquier modificación en ns, afecta al valor almacenado en c.

Cuando termina la ejecución se destruyen las locaciones de memoria ligadas a los parámetros.

Procedimientos y Funciones

```
function doble(n : integer): integer;
begin
  doble := n * 2;
end;
```

a:= **doble(b)** + doble (c); {2}

En la primera invocación a doble, se reserva una locación de memoria para el parámetro formal n, que se inicializa con el valor del parámetro real b.

Cuando la función se termina la locación se destruye y se reanuda la evaluación de la expresión. El resultado de la primera invocación constituye el valor del primer operando de la suma.

Luego se realiza la segunda invocación que procede de igual manera con el parámetro real c y el resultado de la función constituye el segundo operando de la suma.

Procedimientos y Funciones

```
procedure sigant (n: integer; var na, ns: integer);
begin
   na := n – 1;
   ns := n + 1;
end;
```

sigant(a, c, b); {3}

La segunda invocación de sigant asigna el valor del parámetro real a al parámetro formal n.

El parámetro real c queda ligado al parámetro formal na, cualquier modificación en na, afecta al valor almacenado en c.

El parámetro real b queda ligado al parámetro formal ns, cualquier modificación en ns, afecta al valor almacenado en b.

Cuando termina la ejecución se destruyen las locaciones de memoria ligadas a los parámetros.

Manejo de memoria

Un parámetro por valor se inicializa en el momento de la invocación del procedimiento con el valor del parámetro real.

Al terminar la ejecución del procedimiento la variable asociada al parámetro formal se destruye y su valor se pierde.

Un parámetro por referencia introduce un nuevo nombre para referirse a la celda de memoria que corresponde al parámetro real. Cualquier modificación del parámetro formal dentro del procedimiento o función afecta al valor almacenado en parámetro real.

Pasaje de Parámetros

Cuando comienza la ejecución de un programa en Pascal se reservan una o más celdas en memoria para cada una de las variables globales (variables declaradas en el programa principal).

Cuando un procedimiento o función se invoca se reservan celdas para cada una de las variables locales al procedimiento o función, como así también para los parámetros.

Para cada parámetro pasado por referencia se reserva una celda que mantiene la referencia al parámetro real. Todas estas celdas son liberadas cuando el procedimiento o función termina.

Manejo de memoria

Pasos en la invocación a un procedimiento:
1) Guardar lugar para parámetros formales, variables locales y nombre de la función (si lo es la primitiva)
2) Pasaje de parámetros: Copiar parámetros por valor, poner referencias en parámetros por variable.
3) Ejecutar primitiva
4) Liberar espacio de la primitiva

Definición:
procedure p(pvalor: ---; var pvariable: ---); …

Invocaciones posibles:
p(100, a)
p(a, a)
p(sqr(a)+30, a)

Manejo de memoria

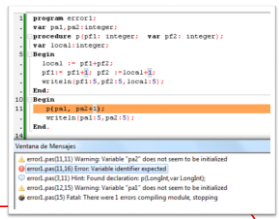
Variables GLOBALES

```
program paramProc;
Var pa1, pa2: integer;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
Begin
  local := pf1+pf2;
  pf1:= pf1+1; pf2 :=local+1;
  writeln(pf1, pf2, local);
End;
Begin
  p(pa1, -10);
  writeln(pa1:5,pa2:5);
End.
```

Error en compilación
El parámetro real **no puede ser una constante** cuando el pasaje de parámetros es **por referencia**

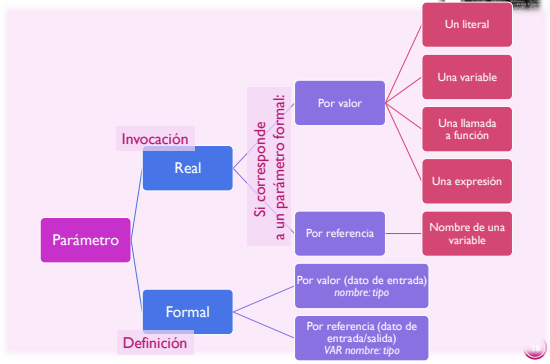
Manejo de memoria

```
program paramProc;
Var pa1, pa2: integer;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
Begin
  local := pf1+pf2;
  pf1:= pf1+1; pf2 :=local+1;
  writeln(pf1, pf2, local);
End;
Begin
  p(pa1, pa2+1);
  writeln(pa1:5,pa2:5);
End.
```



Error en compilación
El parámetro real **no puede ser una expresión** cuando el pasaje de parámetros es **por referencia**

Manejo de memoria



Pasaje de Parámetros

```
Program parametros;
var pa1, pa2: integer;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local:=pf1+pf2;
  writeln(pf1, pf2, local);
  pf1:= pf1+1; pf2:=local+1;
  writeln(pf1, pf2, local);
end;
begin
   pa1:=1;  pa2:=3;
   writeln(pa1, pa2);
   p(pa1, pa2);
   writeln(pa1, pa2);
end.
```

Variables globales

Parámetros formales

Variables locales

Parámetros reales (efectivos)

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
Begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1, pf2, local);
End;
Begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
End.
```

pa1	1
pa2	5

Se agrega la definición de una función

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
   pa1:=1; pa2:=5;
   p(pa1, pa2);
   writeln(pa1:5,pa2:5);
end.
```

pa1	1
pa2	5
pf1	1
pf2	
local	

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
   pa1:=1; pa2:=5;
   p(pa1, pa2);
   writeln(pa1:5,pa2:5);
end.
```

pa1	1
pa2	5
pf1	1
pf2	
local	
x	1
f	

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
   pa1:=1; pa2:=5;
   p(pa1, pa2);
   writeln(pa1:5,pa2:5);
end.
```

pa1	1
pa2	5
pf1	1
pf2	
local	
x	1
f	2

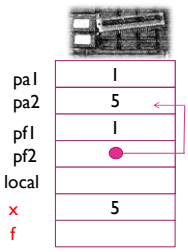
Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  2
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
   pa1:=1; pa2:=5;
   p(pa1, pa2);
   writeln(pa1:5,pa2:5);
end.
```

pa1	1
pa2	5
pf1	1
pf2	
local	

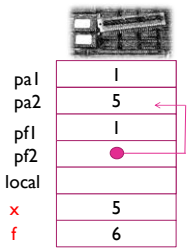
Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin      2
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```



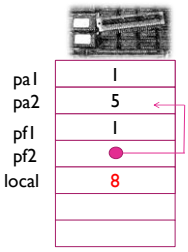
Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin      2
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```



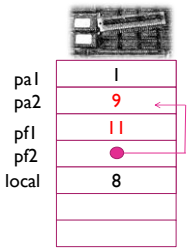
Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin      2      6
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```



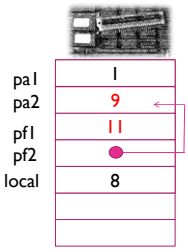
Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```



Manejo de memoria

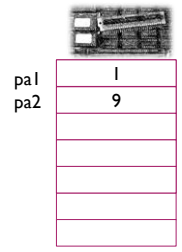
```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```




```
11 9 8
```


Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2 :integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```




```
11 9 8
1 9
```

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=f(pa1);
  p(f(pa1*2), f(pa2));
  writeln(pa1:5,pa2:5);
end.
```

pa1
pa2

Error en compilación
El parámetro real no puede ser una expresión cuando el pasaje es por referencia

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
    …
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
    …
end;
begin
  …
end.
```

El ambiente de referenciamiento del programa principal se denomina **entorno global**.

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
var local: …;
begin
    …
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
    …
end;
begin
  …
end.
```

El ambiente de referenciamiento de un sub-programa se compone del **entorno global** + el entorno local.

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
    …
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
    …
end;
begin
  …
end.
```

El ambiente de referenciamiento de un sub-programa se compone del **entorno global** + el entorno local.

Además siempre se pueden utilizar todos los identificadores predefinidos.

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```

- La variable x es el parámetro formal de la función f y solo es visible en el bloque de la función.
- Las variables pf1, pf2 y local solo son visibles en el bloque de p.

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
  f:=x+1;
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
  local := f(pf1) + f(pf2);
  pf1:= pf1+10; pf2 :=local+1;
  writeln(pf1:5,pf2:5,local:5);
end;
begin
  pa1:=1; pa2:=5;
  p(pa1, pa2);
  writeln(pa1:5,pa2:5);
end.
```

- Cuando p invoca a f las variables pf1, pf2 y local continúan almacenadas en memoria, pero no son accesibles desde el bloque ejecutable de f.
- Al terminar la ejecución de f, el control vuelve a p y sus parámetros y variables locales vuelven a ser accesibles.

Manejo de memoria

```
program parampyf;
var pa1, pa2: integer;
function f(x: integer): integer;
begin
    …
end;
procedure p(pf1: integer; var pf2: integer);
var local: integer;
begin
    …
end; …
```

Aunque Pascal permite que el procedimiento p y la función f accedan a las variables globales pa1 y pa2, la programación modular recomienda no hacerlo y adoptaremos esa recomendación.

La programación modular establece que el bloque ejecutable de cada función y procedimiento acceda únicamente a sus parámetros formales y variables locales.

Procedimientos y Funciones

Un programa en Pascal puede incluir procedimientos y funciones definidos por el programador.

En la declaración de un procedimiento o función se establece su nombre y la lista de parámetros.

Cuando un procedimiento o función ha sido definido puede ser usado mediante una instrucción de invocación.

38

Procedimientos y Funciones

ACTIVACIÓN DE UN PROCEDIMIENTO

- Un procedimiento se invoca desde una instrucción.
- Una vez invocado el control (la ejecución) pasa al procedimiento.
- El bloque de código que incluye la invocación, se suspende.
- Cuando comienza la ejecución se reserva espacio en memoria para las variables locales y para los parámetros por valor, que se inicializan con los valores de los parámetros reales. También para las referencias almacenadas en los parámetros por variable.

39

Procedimientos y Funciones

ACTIVACIÓN DE UN PROCEDIMIENTO

- El procedimiento se ejecuta y cuando termina, el control retorna, provocando algún efecto.
- El efecto de la ejecución de un procedimiento puede ser modificar uno o más parámetros por variable o producir una salida por consola o por archivo.
- El bloque de código que había quedado suspendido se reanuda, en la instrucción siguiente a la invocación.

40

Procedimientos y Funciones

ACTIVACIÓN DE UNA FUNCIÓN

- Una función se invoca desde una expresión.
- Una vez invocada el control (la ejecución) pasa a la función.
- El bloque de código que incluye la invocación, se suspende.
- Cuando comienza la ejecución se reserva espacio en memoria para las variables locales y para los parámetros por valor, que se inicializan con los valores de los parámetros reales.

41

Procedimientos y Funciones

ACTIVACIÓN DE UNA FUNCIÓN

- La función se ejecuta y cuando termina, el control retorna al lugar de la expresión desde donde se la llamó.
- El bloque de código que había quedado suspendido se reanuda, continuando con la evaluación de la expresión que contiene a la invocación.
- El efecto de la ejecución de una función es retornar un valor.

42