Resolución de Problemas y Algoritmos

clase 15: resolución de problemas utilizando recursión

Dr. Alejandro J. García
e-mail: agarcia@cs.uns.edu.ar
http:// cs.uns.edu.ar /~ajg

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

Temas vistos

Los siguientes temas, vistos en clases anteriores, están todos relacionados y son muy importantes:

- **Diseño de la solución dividiendo el problema**
- **Funciones y procedimientos en Pascal**
- **Parámetros (por valor y por referencia)**
- **Entorno de referencia de los identificadores**
- **Visibilidad – Identificadores locales, globales, etc.**

¿Alguna pregunta?

Como un repaso de los temas vistos, realice la traza de los siguientes programas: repaso1, repaso2 y repaso3.

```
program Repaso1;
var i, tope, suma:integer;
    Function F3 (N:integer):integer;
    var local:integer;
     begin
     local:= N; F3:= N + local;
    end;
begin
Writeln('Ingrese un tope '); readln(tope);
Suma:=0;
for i:=1 to tope do suma:=suma+F3(i);
Writeln('La suma es ', suma);
end.
```



```
program Repaso2; 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); 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); 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); end;
begin global:= 3 + F1(5) * 10; writeln(global); end.
```

```
program Repaso3; var V:integer;
    procedure P3 (N:integer; var R:integer);
    var local: integer;
    begin writeln('Entro a P3 con ', N);
     local:= N-1; writeln('Al fin dejo de llamar !');
     N:= N* 10; writeln('Salgo de P3 con ', N); end;
    procedure P2 (N:integer; var R:integer);
    var local: integer;
    begin writeln('Entro a P2 con ', N);
     local:= N-1; P3 (local);
     N:= N* 10; writeln('Salgo de P2 con ', N); end;
    procedure P1 (N:integer; var R:integer);
    var local: integer;
    begin writeln('Entro a P1 con ', N, R);
     N:= N-1; R:= N; P2 (N, local);
     N:= N* 10; writeln('Salgo de P1 con ', N); end;
begin writeln('Llamo a P1:'); V:=5; P1(3, V); write(V); end.
```

Algoritmos recursivos

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una se *sensado* y otra de *efecto*):

- **quedan-escalones: función que retorna verdadero si está frente a un escalón, o falso en caso contrario.**
- **subir-escalón: hace al robot subir un escalón.**

Puede escribirse el siguiente algoritmo que usa 3 primitivas:

Algoritmo: subir-una-escalera
Si quedan-escalones
entonces: - subir-escalón
- subir-una-escalera

Algoritmos recursivos

- **Recursión es la forma en la cual se especifica un proceso basado en su propia definición.**
- **La solución de un problema es recursiva, si el proceso que describe utiliza recursión.**
- **Un algoritmo es recursivo si se define en términos de sí mismo.**
- **Un algoritmo no debe entrar en un ciclo infinito, por ende, un algoritmo recursivo será válido, si:**
  (a) existe un caso base que no se define en términos de si mismo, y
  (b) la referencia a sí mismo es relativamente más sencilla (o reducida) que el caso considerado.

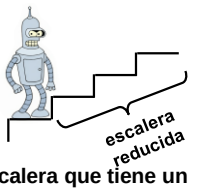
Algoritmos recursivos

- **Un algoritmo recursivo será válido, si:**
  (a) existe un caso base que no se define en términos de si mismo, y
  (b) la referencia a sí mismo es relativamente más sencilla o reducida que el caso considerado.

Ejemplo: subir una escalera

(a) Caso base: no hay escalones

(b) Caso general:
- sube un escalón
- y luego queda por subir una escalera que tiene un escalón menos, y por lo tanto es más reducida.



Algoritmos recursivos

Considere un escenario donde debemos programar un robot para subir una escalera, y se dispone de las primitivas (una se *sensado* y otra de *efecto*):

- **quedan-escalones: retorna verdadero o falso**
- **subir-escalón: hace al robot subir un escalón**

El siguiente algoritmo es incorrecto ¿Por qué?

Algoritmo 2 subir-escalera
-subir-escalón
-subir-escalera

MAL

Falla porque no hay indicado un caso base (a).

Algoritmos recursivos

El Algoritmo 3 también es incorrecto ¿Por qué?

Algoritmo 3 subir-la-escalera
Si quedan-escalones
entonces: - subir-la-escalera
- subir-escalón

MAL

En este caso falla (b) porque la referencia a sí mismo NO es relativamente más sencilla o reducida (es igual)

Otros algoritmos recursivos

Se quiere controlar un conjunto de solicitudes (de beca, o pase de comisión, etc) con las siguientes primitivas:

- **Procesar-solicitud: saca la solicitud del conjunto y realiza el control correspondiente**
- **quedan-solicitudes retorna verdadero o falso**

Algoritmo: controlar solicitudes
Si quedan-solicitudes
entonces: - procesar-solicitud
- controlar solicitudes

Soluciones recursivas

- **Dado un problema P, la solución puede realizarse en forma recursiva cuando:**
  - **Hay un caso base, esto es una solución para un caso trivial del problema que no se define en términos de si misma.**
  - **En el caso general, la solución de cualquier instancia del problema P (excepto la trivial) requiere resolver instancias más simples o pequeñas del mismo.**
- **De esta forma, al utilizar una solución recursiva, el problema queda divido en dos subproblemas: (1) caso base y (2) caso general.**

Otros algoritmos recursivos

- Generalmente los trabajos de impresión (jobs) son enviados a una cola de impresión (queue) antes de ser impresos. Se puede escribir un algoritmo que se encargue de imprimir los trabajos pendientes de la siguiente manera:

```
Algoritmo imprimir-trabajos
Si quedan-trabajos-en-cola
  entonces: | - sacar-un-trabajo(T)
            | - imprimir-un-trabajo(T)
            | - imprimir-trabajos
```


Otros algoritmos recursivos

Para comer un plato de ravioles:

```
Algoritmo comer-ravioles
Si quedan-ravioles
entonces: | - comer-un-raviol
          | - comer-ravioles
```

Para tomar un vaso de una bebida:

```
Algoritmo  tomar-bebida
Si queda-líquido
entonces: | - tomar-un-sorbo
          | - tomar-bebida
```


Problema propuesto: 2^N

- Para N no negativo, la función 2^N puede definirse recursivamente de la siguiente manera:

$$2^N \begin{cases} 2^0 = 1 \quad \text{(para N=0)} \\ 2^N = 2 * 2^{N-1} \quad \text{para N>0} \end{cases}$$

Planteo recursivo para 2^N

- Caso base o trivial: si N=0 entonces 2^N es 1
- Caso general o caso recursivo: Si N>0 entonces 2^N es $2 * 2^{N-1}$


```
program DosAlaNRecursivo;
var  num, resu: real;
     FUNCTION  dosAlaN (N:real):real;
     var aux:real;
      begin
       if (N = 0) then aux:=1
                  else  aux:= 2 * dosAlaN(N-1);
       dosAlaN:=aux;
      end;
begin
writeln(' Ingrese un número'); Readl(Num);
resu:=dosAlaN(Num);
writeln(' 2 a la ', Num,' es ' , resu );
end.
```

Realice la traza para num=0 y num=2. ¿Cuántas veces se llama a la dosAlaN con respecto al valor de num?

Problema propuesto: factorial

- Las función factorial (!) puede definirse recursivamente de la siguiente manera:

$$N! \begin{cases} 0! = 1 \\ N! = N * (N-1)! \end{cases}$$

- Observe que tiene un caso base y otro caso que se define con una instancia más simple de sí mismo.

Planteo Recursivo para Factorial de N

Caso Base: si N = 0 Factorial de N es 1

Caso General:

si N > 0 entonces Factorial de N es N * Factorial de N-1


```
program Factorial-Recursivo;
var  num, fact: integer;
     Function Factorial (N:integer):integer;
     begin
      IF N=0 THEN Factorial:=1
      ELSE Factorial:= N * Factorial(N-1);
     end;
begin
writeln(' Ingrese un número'); Readl(Num);
fact:=factorial(Num);
writeln(' Factorial de ', Num,' es ' , Fact );
end.
```