TECNOLOGÍAS EN EDUCACIÓN MATEMÁTICA

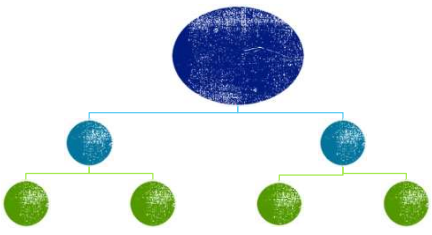
MODULO 15

Dpto. de Ciencias e Ingeniería de la Computación
UNIVERSIDAD NACIONAL DEL SUR
Año 2019

Descomposición de Problemas en Sub-problemas

Cuando la complejidad de los problemas aumenta, la tarea de hallar una solución se torna más difícil.

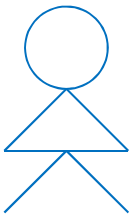
Una metodología para reducir la complejidad consiste en plantear la solución del problema a partir de la solución de una serie de sub-problemas más sencillos que forman parte del problema original.

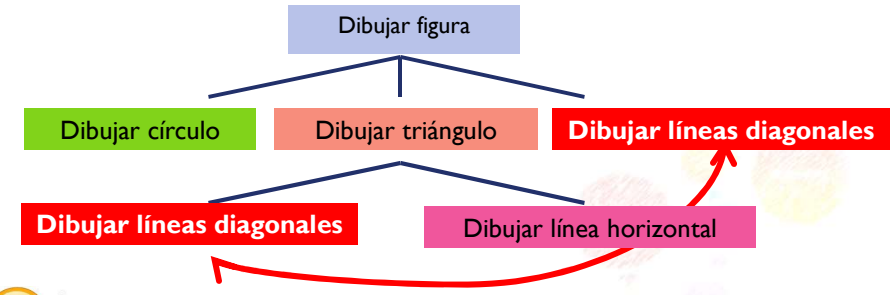


Descomposición de Problemas en Sub-problemas

Problema: Dibujar esta figura

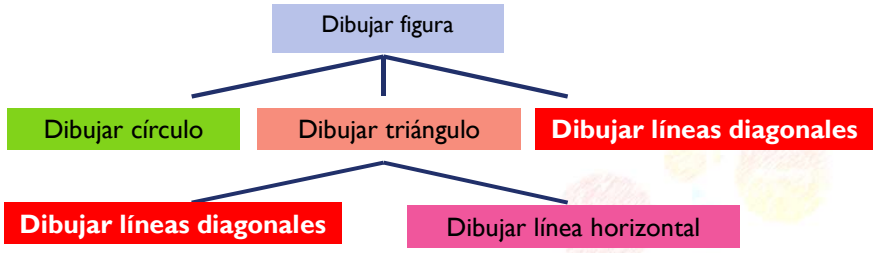
1. Dibujar el círculo
2. Dibujar el triángulo
    1. Dibujar las dos líneas diagonales
    2. Dibujar la línea horizontal
3. Dibujar otras dos líneas diagonales





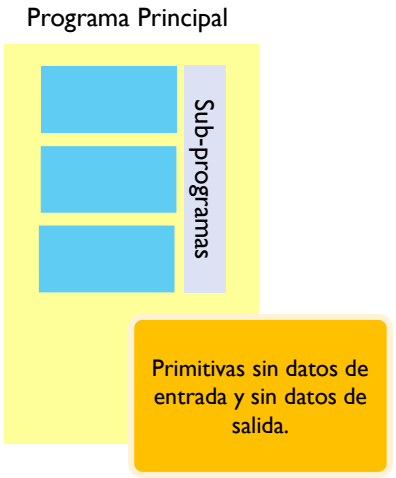
OJO!!!!! En realidad son el mismo "sub-problema"

Descomposición de Problemas en Sub-problemas



Dibujar círculo, dibujar líneas diagonales
y dibujar línea horizontal son acciones primitivas
que se implementan como sub-programas.

```
program dibujo;
 procedure dibujaCirculo;
  begin
          writeln('   *   ');
          writeln(' *   * ');
          writeln('   *   ');
  end;
 procedure dibujaDiagonales;
  begin
          writeln('  /\  ');
          writeln(' /  \ ');
          writeln('/    \');
  end;
 procedure dibujaLinea;
  begin
          writeln('_______');
  end;
begin
  dibujaCirculo;    dibujaDiagonales;    dibujaLinea;    dibujaDiagonales;
  readln;
end.
```



Descomposición de Problemas en Sub-problemas

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*).
- Ayudan a que el código del programa principal sea:
  - **Legible**: Resulta más sencillo leer sólo el nombre del subprograma que todo su código.
  - **Ordenado**: Cada subprograma ocupa un lugar concreto dentro de todo el código.

Descomposición de Problemas en Sub-problemas

Hay que distinguir entre la declaración y la invocación de sub-programas.

- ✓ La declaración sirve para definir el sub-programa
- ✓ La invocación se utiliza para usar el sub-programa dentro del programa principal (o de otro sub-programa)

```
program dibujo;
 procedure dibujaCirculo;
   begin
          writeln('   *   ');
          writeln(' *   * ');
          writeln('   *   ');
   end;
 procedure dibujaDiagonales;
   begin
          writeln('  /\  ');
          writeln(' /  \ ');
          writeln('/    \');
   end;
 procedure dibujaLinea;
   begin
          writeln('_______');
   end;
begin
  dibujaCirculo;   dibujaDiagonales;   dibujaLinea;   dibujaDiagonales;
  readln;
end.
```

Declaraciones

Invocaciones

```
program dibujo;
var i: integer;
 procedure dibujaCirculo;
   begin
          writeln('   *   ');
          writeln(' *   * ');
          writeln('   *   ');
   end;
 procedure dibujaDiagonales;
   begin
          writeln('  /\  ');
          writeln(' /  \ ');
          writeln('/    \');
   end;
 procedure dibujaLinea;
   begin
          writeln('_______');
   end;
begin
  …
  readln;
end.
```

Declaraciones

```
for i:=1 to 3 do
 begin
  dibujaCirculo;
  dibujaCirculo;
  dibujaDiagonales;
  dibujaLinea;
 end;
```

Invocaciones

Descomposición de Problemas en Sub-problemas

En Pascal, los sub-programas son implementados mediante procedimientos o funciones.

- Procedimientos predefinidos: *write*, *writeln*, *read*, *readln*
- Funciones predefinidas: *sqr*, *abs*, *chr*, *ord*

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

En la declaración se establece su nombre y la lista de parámetros (datos que se intercambian con el exterior).

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

Descomposición de Problemas en Sub-problemas

```
Program p;
var n: integer;
begin
writeln('Ingrese un número:');
readln(n);
if (abs(n)=n)
then
        writeln('El numero es positivo.')
else
        writeln('El numero es negativo.');
end.
```

PROCEDIMIENTO

La invocación constituye en sí misma una instrucción.

Descomposición de Problemas en Sub-problemas

FNCION

La invocación se realiza desde una expresión.

```
Program p;
var n: integer;
begin
writeln('Ingrese un número:');
readln(n);
if (abs(n)=n)
then
        writeln('El numero es positivo.')
else
        writeln('El numero es negativo.');
end.
```

Pascal -Funciones

Pascal permite que el programador agregue nuevos operadores asociados a identificadores a través de la definición de **funciones**.

Una función, predefinida por Pascal o definida por el programador, se caracteriza porque se **invoca** desde una expresión.

Definiremos funciones que pueden ser invocadas desde expresiones aritméticas, lógicas o de caracter.

Pascal - Funciones

Por ejemplo:

x := sqr(y) + potencia(y, n);

- **sqr** es una función predefinida de Pascal que recibe un parámetro de tipo **real** y computa un resultado de tipo **real**.
- **potencia** es una función definida por el programador que recibe dos parámetros, uno de tipo **real** y otro de tipo **integer** y computa un valor de tipo **real**.

Pascal -Funciones

Una **función** es una **instrucción compuesta** que tiene un **nombre**, *puede* tener **parámetros**, *puede* contener **declaraciones** y *debe* **retornar un único valor** de un tipo dado.

```
Algoritmo minimo
DE: n, m {enteros}
DS: minimo {entero}
Comienzo
Si (n<m)
        entonces
                minimo<- n
        sino
                minimo<-m
Fin
```

- El **nombre** la función es minimo.
- Los **parámetros** de la función son dos enteros n y m.
- El **resultado** es de tipo *integer* y **se liga a través de una asignación usando el nombre de la función.**

Pascal -Funciones

```
function minimo (n, m: integer): integer;
begin
  if (n < m)
  then minimo := n
  else minimo := m
end;
```

Dato de salida

Siempre tiene que haber al menos una asignación donde se establece el valor del nombre de la función

Pascal -Funciones

```
program pruebaMin;
var n1, n2, min: integer;
function minimo (n, m: integer): integer;
begin
  if (n < m)
  then minimo := n
  else minimo := m
end;
begin
  write ('Ingrese dos números ');
  readln (n1,n2);
  min := minimo(n1, n2);
  writeln ('El minimo es ',min);
end.
```

writeln('El minimo es ', minimo(n1, n2));

Pascal -Funciones

Cuando en la parte izquierda de una asignación aparece el nombre de la función, estamos indicando cuál va a ser el **resultado** de computar la función.

```
program pruebaMin;
var n1, n2, min: integer;
 function minimo (n, m: integer): integer;
  begin
   if (n < m)
   then minimo := n
   else minimo := m
  end;
begin
 …
end.
```

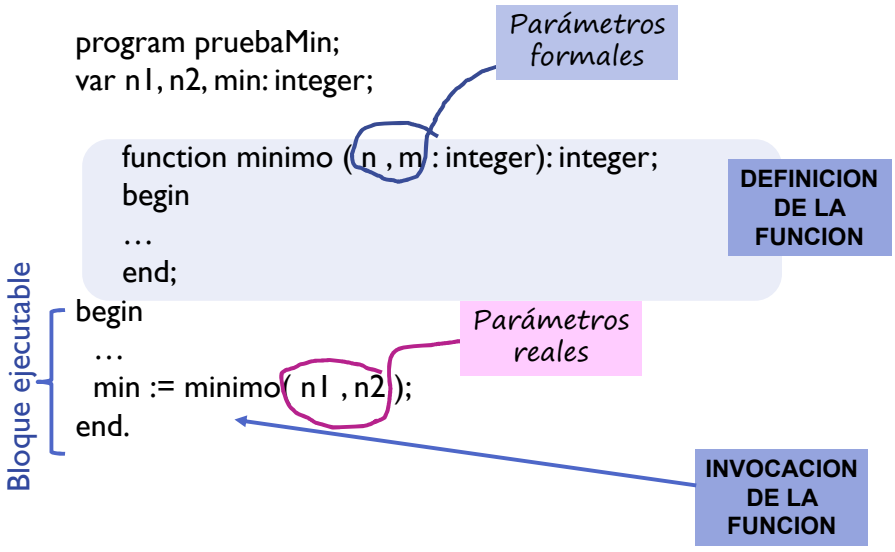
Pascal -Funciones

```
program pruebaMin;
var n1, n2, min: integer;
 function minimo (n, m: integer): integer;
  begin
   if (n < m)
   then minimo := n
   else minimo := m
  end;
begin
 …
end.
```

EN LA DEFINICION DE LA FUNCION:
El nombre de la función SOLAMENTE
puede aparecer
DEL LADO IZQUIERDO
de una asignación.

- n y m son los **parámetros formales** de la función, es decir que son los datos que la función va a utilizar para **computar** el resultado.

Pascal -Funciones



Pascal -Funciones

En el bloque ejecutable del programa principal **invocamos** a la función a través de su nombre.

Los parámetros n1 y n2 son los **parámetros reales** de la función.

Los parámetros formales y reales tienen que coincidir en **tipo, orden y número**.

```
program pruebaMin;
var n1, n2, min: integer;
 function minimo (n, m: integer): integer;
  begin
   if (n < m)
   then minimo := n
   else minimo := m
  end;
begin
 …
end.
```

Pascal -Funciones

Problema: Leer dos números enteros y determinar si ambos tienen igual cantidad de dígitos.

Primitiva: Contar la cantidad de dígitos de un número entero n.

```
Algoritmo cantDigitos
DE: n {entero}
DS: cantDigitos {entero}
Comienzo
cantDigitos<-0
mientras (n<>0) hacer
        n<-n div 10
        cantDigitos← cantDigitos+1
Fin
```

¿Puedo escribir esto en lenguaje Pascal? ¿Porque?

Pascal -Funciones

Problema: Leer dos números enteros y determinar si ambos tienen igual cantidad de dígitos.

Primitiva: Contar la cantidad de dígitos de un número entero n.

```
Algoritmo cantDigitos
DE: n {entero}
DS: cantDigitos {entero}
Comienzo
cantDigitos<-0
mientras (n<>0) hacer
        n<-n div 10
        cantDigitos←cantDigitos+1
Fin
```

EN LA DEFINICION DE LA FUNCION: El nombre de la función SOLAMENTE puede aparecer DEL LADO IZQUIERDO de una asignación.

Pascal -Funciones

```
function cantDigitos( num : integer ): integer;
var contador: integer;
begin
  contador := 0;
  while (num<>0) do
   begin
    contador := contador + 1;
    num := num div 10;
   end;
  cantDigitos := contador;
end;
```

Variable local: auxiliar para no usar el nombre de la función

Observemos que hemos usado una **variable auxiliar** para el contador y al terminar el procesamiento del número **asignamos la variable auxiliar al nombre la función**.

Pascal -Funciones

```
program igualCantDigitos;
var n1, n2: integer;
function cantDigitos(num : integer): integer;
begin
 …

end;
begin
  write (‘Ingrese dos números enteros‘);   readln(n1, n2);
  if (cantDigitos(n1)=cantDigitos(n2))
            then writeln (‘Ambos nros tienen = cant de digitos’)
            else writeln(‘Los nros no tiene = cant de digitos’);
end.
```

Cuando la función cantDigitos termina, tanto n1 como n2 conservan el valor que tenían antes de la invocación

Pascal -Funciones

```
function cantDigitos( num : integer): integer;
```

¿Donde hay **expresiones** en nuestros programas?

- ASIGNACION
  - var:=expresion
- CONDICIONES
  - if (expresion_logica) then …
  - while (expresion_logica) do …
- Procedimientos predefinidos write y writeln
  - write(expresion)
  - writeln(expresion)

Pascal -Funciones

function cantDigitos(num : integer): integer;

- ASIGNACION

  cantDigAyB:=cantDigitos(a)+cantDigitos(b);

- CONDICIONES

  if (candDigitos(n)>3) then …

  while (cantDigitos(num)<>0) do …

- Procedimientos predefinidos write y writeln

  writeln(n,'tiene ', cantDigitos(n),' dígitos')

  writeln(cantDigitos(numero)+20)

La invocación puede aparecer en todos los lugares donde puede haber un valor del tipo de salida de la función.

Pascal -Funciones

function cantDigitos(num : integer): integer;

Ejemplos de maneras de invocar a la función con distintos parámetros reales:

```
c:= cantDigitos(3) …
c:= cantDigitos(a) …
c:= cantDigitos(a+b)
c:= cantDigitos(sqr(n)) …
```

El parámetro real puede ser cualquier expresión que compute un valor del tipo del parámetro formal.
Al momento de la invocación, se evalúa la expresión y el valor obtenido se asigna al parámetro formal.

Pascal -Funciones

Sucesiones

Problema: calcular el i-ésimo término de la siguiente sucesión infinita.

S = 2/1, 4/2, 8/6, 16/24, 32/120, 64/720…

Término general: $2^N/N!$

- Descomponer el problema en sub-problemas:
  - Calcular Potencia
  - Calcular Factorial
  - Calcular el i-ésimo Término usando *Potencia* y *Factorial*.

Pascal -Funciones

ALGORITMO Término-i-ésimo
DATOS DE ENTRADA: i {natural}
DATOS DE SALIDA: Término-i-ésimo {real}
COMIENZO
  Término-i-ésimo ← Potencia(2,i)/Factorial(i)
FIN ALGORITMO

```
function iesimo(i: integer): real;
begin
  iesimo := potencia(2, i) / factorial(i);
end;
```

Pascal -Funciones

Si ahora quisiera escribir un algoritmo para calcular la sumatoria de los primeros K términos de la sucesión anterior...

```
ALGORITMO SumaKTérminos
DATOS DE ENTRADA: K {natural}
DATOS DE SALIDA: Sumatoria {real}
DATOS AUXILIARES: i {natural}
COMIENZO
    Sumatoria ← 0
    para i desde 1 hasta K hacer
        Sumatoria ← Sumatoria + iesimo(i)
   Fin Repetir
FIN ALGORITMO
```

El dato de entrada se lee de teclado

El dato de salida se muestra por pantalla

Este es el programa principal

Pascal -Funciones

```
program sumaKterminos;
var i, k: integer; suma: real;
function factorial(n : integer): integer;  ...
function potencia (m: real; n: integer): real;  ...
function iesimo (i: integer): real;
begin
iesimo := potencia(2, i) / factorial(i);
end;
begin
repeat
writeln('Cantidad de terminos a sumar: ');
readln(k)
until (k>=0);
suma:=0;
for i:=1 to k do
 suma:= suma + iesimo(i);
writeln('La suma de los primeros ', k, ' terminos es: ', suma:2:2);
readln;
end.
```

Validación de k

Mismo nombre de dato en distinto ambiente de referenciamiento, es DISTINTO DATO

Pascal -Funciones

Problema: Determinar si un caracter dado es una vocal minúscula.

```
function esVocalMin (car:char) : boolean;
begin
   if (car = 'a') or (car = 'e') or (car = 'i')
      or (car = 'o') or (car = 'u')
   then esVocalMin := true
   else esVocalMin := false
end;
```

car in ['a', 'e', 'i', 'o', 'u']

Pascal -Funciones

Definición de funciones:
El **nombre** de la función aparece por lo menos dos veces, en el **encabezamiento** y a la **izquierda de una asignación**.

```
function esVocalMin(car:char): boolean;
begin
   esVocalMin := car = 'a' or car = 'e' or car = 'i'
                 or car = 'o' or car = 'u'
end;
```

car in ['a', 'e', 'i', 'o', 'u'];

Pascal -Funciones

La variable car de tipo *char* es el **parámetro formal** de la función. Cuando la función se **invoque** debe hacerse con un **parámetro real** de tipo *char*. La función **computa** un **resultado** de tipo *boolean*.

La invocación debe aparecer en un contexto que requiera un valor de tipo *boolean*. Por ejemplo en la expresión lógica de un while o de un if.

```
function esVocalMin(car:char): boolean;
begin
  esVocalMin := car = 'a' or car = 'e' or car = 'i'
                or car = 'o' or car = 'u'
end;
```

Pascal -Funciones

Problema: Determinar si un caracter dado es una letra minúscula.

```
function esMinuscula (car:char) : boolean;
begin
  if (car >= 'a') and (car <= 'z')
  then esMinuscula := true
  else esMinuscula := false
end;
```

Pascal -Funciones

Problema: Determinar si un caracter dado es una consonante minúscula.

car: Parametro formal

```
function esConsonanteMin(car: char): boolean;
begin
  if (esMinuscula(car) and not (esVocalMin(car))
  then esConsonanteMin := true
  else esConsonanteMin:= false
end;
```

esConsonanteMin := esMinuscula(car) and not (esVocalMin(car))

Pascal -Funciones

Problemas similares son modelados por expresiones diferentes.

Determinar si un caracter es:

- Letra minúscula
- Vocal
- Consonante

```
function esVocalMin(car: char): boolean;
begin
   if (car = 'a') or (car = 'e') or (car = 'i') or (car = 'o') or (car = 'u')
   then esVocalMin := true
   else esVocalMin := false
end;
```

```
function esMinuscula(car: char): boolean;
begin
   if (car >= 'a') and (car <= 'z')
   then esMinuscula := true
   else esMinuscula := false
end;
```

```
function esConsonanteMin(car: char): boolean;
begin
   if (esMinuscula(car)) and not (esVocalMin(car))
   then esConsonanteMin := true
   else esConsonanteMin:= false
end;
```

Pascal -Funciones

Resumen Parámetros de Funciones:

Los datos de entrada:

- En el programa principal se leen con READ.
- En las primitivas se pasan como parámetro por valor (entre los paréntesis del encabezado).

Los datos de salida:

- En el programa principal se muestran con WRITE.
- En las primitivas:
  - Cuando es UNO SOLO se implementa una función y se devuelve en el nombre de la función.

Pascal -Funciones

Problema: Leer una secuencia de caracteres terminada en punto y contar la cantidad de vocales minúsculas, consonantes minúsculas y otros caracteres que contiene la secuencia.

Pascal -Funciones

Problema: Mostrar por consola el factorial de cada uno de los valores naturales comprendidos entre a y b. El programa debe validar los valores a y b ingresados por el usuario. Esto es, se asume que a y b son enteros, pero si a>b o a<0 se muestra un mensaje de error y se solicita que se vuelvan a ingresar los límites del intervalo.

Por ejemplo si a=3 y b= 5 el programa debe mostrar:

Factorial de 3 es 6

Factorial de 4 es 24

Factorial de 5 es 125

Pascal -Funciones

Problema: Leer por consola un número real r y un número natural m (validar) y mostrar el resultado de computar:

$$\sum_{k=0}^{m} k!\, r^k$$

TECNOLOGÍAS EN EDUCACIÓN MATEMÁTICA

FIN MODULO 15

Dpto. de Ciencias e Ingeniería de la Computación
UNIVERSIDAD NACIONAL DEL SUR
Año 2019