Resolución de Problemas y Algoritmos

Clase 14:

- **Estrategias de resolución de problemas basadas en el uso de primitivas.**
- **Archivos de texto para entrada y salida.**

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:

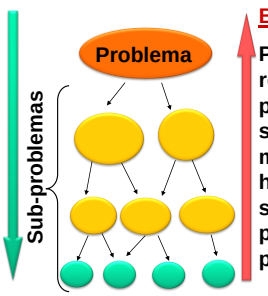
1. **Técnicas para resolver problemas complejos.**
2. **Cómo escribir algoritmos basados en primitivas y algoritmos que son primitivas.**
3. **Cómo implementar en Pascal primitivas y poder usar las dos técnicas anteriores.**

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

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

Top-down:

Por 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

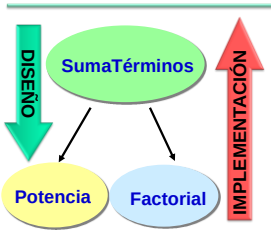
En Pascal, ¿tengo una primitiva para N! (factorial)?

¿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

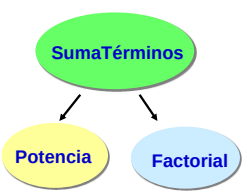



```
PRIMITIVA SumaTerminos
Dato Entrada : K {cant. términos }
Dato Salida : Suma
COMIENZO
  i ← 0     Suma ← 0
  Repetir K veces
    i ← i + 1
    Suma←Suma + Potencia(2,i)
                Factorial(i)
FIN ALGORITMO
```

```
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



PRIMITIVAS en Pascal

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

```
PROGRAM sumaK;
{resuelve el problema de sumar los primeros "K" términos de la serie 2^N / N! }
VAR cant: integer;  resultado:real;
  FUNCTION factorial (N:integer): integer;
  {retorna el factorial del parámetro N}        {TAREA: COMPLETAR el cuerpo}
  FUNCTION potencia (B,E:integer):integer;
  {retorna B elevado a la E}                    {TAREA: COMPLETAR el cuerpo}
  PROCEDURE SumaTerminos ( K:integer; VAR suma: real) ;
  VAR i: integer; {suma los primeros "K" términos de la serie 2^N / N! }
  BEGIN { el parámetro por referencia "suma" acumula y retorna el resultado}
    suma:= 0; {el parámetro por valor "K" indica cuantas iteraciones hacer}
   FOR i:=1 TO K DO   suma := suma+ potencia(2,i) / factorial(i) ;
  END;
BEGIN
writeln('ingrese cantidad de términos a sumar'); readln(cant);
sumaTerminos(cant, resultado);
writeln('la suma es: ', resultado);
END.
```


Tarea (muy importante) propuesta

- **Complete el programa sumaK 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);
- **"sumaTerminos" también podría haber sido una función. Como tarea adicional haga otra versión del programa con una función para "sumaTerminos".**

Reflexión

En clase estuvimos reflexionando sobre las diferencias, ventajas y desventajas entre:

a) ubicar a las funciones factorial y potencia en el entorno global,

b) ubicar a las funciones factorial y potencia en el entorno local al procedimiento "SumaTerminos".

Analice las diferencias y reflexiones sobre las ventajas de cada una de las opciones.

IMPORTANTE: prohibición en RPA

- Los identificadores (constantes, tipos, variables, procedimientos, o funciones) del entorno global son visibles en todo el programa.
- En esta materia **se PERMITE** el uso de **constantes**, **procedimientos**, **funciones** y **tipos** globales..

Pero…

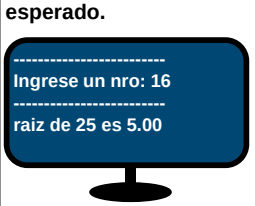
- En esta materia **se PROHIBE** el uso de **variables globales (y no locales)** en procedimientos y funciones.

- **Ayuda**: siempre que surja la necesidad de usar variables globales es porque tiene que usar o una constante, o una variable local o un parámetro.
- Vea los ejemplos que hay a continuación. Páselos a la máquina y reflexione sobre ellos.


```
program prohibido;
var  i: integer;
    Procedure Linea;
    begin
     For i:=1 to 25 DO  write('-');
     writeln;
    end; {linea}
Begin
linea;
write(' Ingrese un nro: '); Readln(i);
linea;
writeln('raiz de ',i,' es', SQRT(i):2:0);
end.
```

•La variable "i" es usada en "linea" como global.

•Esto afecta al resultado esperado.



- **Solución: crear una variable "i" local.**
- **Es evidente que el identificador "i" debería tener un nombre más significativo.**


```
program prohibido2;
var  i,max: integer;
    Procedure linea;
    var i: integer;
    begin
     For i:=1 to max DO  write('-');
    end; {linea}
    {.....}
    Procedure producto;
    var j: integer;
    begin
      for j:=1 to 5 do max:= max * j;
      write(max);
    end; {producto}
begin
max:= 15;
linea;
producto;
linea;
end.
```

En este programa también se ven los problemas de usar una variable global:

• La variable "max" es usada en "linea" como global para el tope del bucle.

• Luego "max" es usada también como global en "producto", afectando a así a las siguientes llamadas del procedimiento "linea".

Solución: que max sea una constante (o un parámetro en "linea") y usar una variable local o un parámetro por referencia en "producto".

```
program bien1;
const max= 15; var  i: integer;
    Procedure linea;
    var i: integer;
    begin
     For i:=1 to max DO  write('-');
    end; {linea}
    {.....}
    Procedure producto;
    var j, max: integer;
    begin
      max:=0;
      for j:=1 to 5 do max:= max * j;
      write(max);
    end; {producto}
begin
linea;
producto;
linea;
end.
```

```
program bien2;
var  i: integer;
   Procedure linea(max:integer);
   var i: integer;
   begin
    For i:=1 to max DO  write('-');
   end; {linea}
   {.....}
   Procedure producto(var max:integer);
   var j: integer;
   begin
     max:=0;
     for j:=1 to 5 do max:= max * j;
   end; {producto}
begin
linea(15);
producto(i);  write(i);
linea(15);
end.
```


Tipos de datos en Pascal

¿Qué elementos de Pascal tienen un tipo asociado?
- una variable,
- una expresión,
- un parámetro,
- una función.

Por lo tanto vamos a "actualizar" nuestra definición de tipo de dato.

Tipo de Dato: define el conjunto de valores posibles que puede tomar un elemento como una variable, expresión, parámetro o función; y también define las operaciones que pueden usarse sobre esos valores.

Archivos de texto en Pascal (TEXT)

- **En Pascal, existe un tipo predefinido "TEXT" que permite trabajar con archivos de texto.**

```
Program ejemplo;
VAR T1, T2,  documento: TEXT;
```

- **A primera vista parece que es lo mismo que tener un FILE OF char. Pero a veremos que no es así.**
- **Recordemos que un tipo de dato define los valores y las operaciones que pueden usarse sobre ellos.**
- **TEXT es un tipo estructurado, que permite tener una secuencia de caracteres,**
- **Tiene todas las operaciones de FILE, pero agrega algunas operaciones y facilidades.**

Problema: escriba un programa para abrir un archivo de texto ya existente llamado "texto.txt", y mostrar por pantalla su contenido.

Programa: a continuación hay una solución posible. *Observe que usa todas operaciones conocidas de FILE.*

```
Program leer; {Este programa permite leer todo el contenido de un
archivo de texto, (carácter por carácter) y mostrarlo en pantalla }
VAR T: TEXT;    elemento: char;
begin
   assign(T, 'texto.txt');
   reset(T);  {abre el archivo para leer de él}
   while not eof(T) do
   begin  read(T,elemento); write(elemento); end;
   close(T);
end.
```


Operaciones sobre archivos de texto en Pascal

Además de las operaciones vistas sobre archivos secuenciales (FILE) se agregan:

Procedimientos predefinidos:

- **readln(T): desplaza el "puntero" de lectura en T hasta el carácter siguiente a un fin de línea (enter).**
- **writeln(T): escribe un fin de línea (enter) en T.**

Función predefinida:

- **eoln(F) (end of line): retorna TRUE si se llegó al final de una línea y FALSE en caso contrario.**

Observaciones:

readln(T,e) es equivalente a read(T,e); readln(T)

writeln(T,e) es equivalente a write(T,e); writeln(T)

End Of Line (fin de línea) EOL

- **End-of-line** (**EOL**) fin de línea (o **line break** o **newline**) es un caracter especial, o secuencia de caracteres, que indica el final de una línea de texto y el paso a la siguiente. Se le llama así porque el caracter a la derecha de **EOL** aparecerá en la línea de abajo.
- Los sistemas operativos representan EOL con uno o dos códigos de control: **LF** (Salto de línea) o **CR** (Retorno de carro) o ambos. Por ejemplo:
- **LF**: Multics, Unix, GNU/Linux, AIX, Xenix, Mac OS X, FreeBSD , BeOS, Amiga, RISC OS,
- **CR+LF**: MS-DOS, OS/2, Microsoft Windows, Symbian
- **CR**:Commodore 8-bit, TRS-80, Apple II family, Mac OS

Problema: escriba un programa para abrir un archivo de texto ya existente llamado “texto.txt”, y contar cuantas líneas tiene.

```
Program lineas; {Este programa cuenta las líneas de un archivo de texto aprovechando el prodemiento predefinido readln que avanza hasta el final de una línea del archivo (ie, un enter) }
VAR T: TEXT;    cant:integer;
begin
    assign(T, ‘texto.txt’); reset(T);  cant:=0;
    while not eof(T) do
        begin  readln(T);  cant:=cant+1; end;
    writeln(‘Cantidad de líneas: ’, cant);
    close(T);
end.
```


Problema: escriba un programa para abrir un archivo de texto ya existente llamado “texto.txt”, y generar otro que cuando encuentre un punto baje de línea.

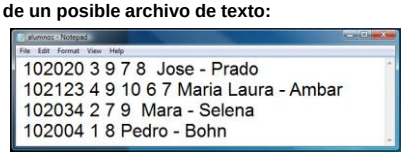
```
Program lineas;
VAR T1,T2: TEXT;    cant:integer;      ch: char;
begin
    assign(T1, ‘texto.txt’); reset(T1);
    assign(T2, ‘otro.txt’); rewrite(T2);
    while not eof(T1) do
    begin
        read(T1,ch);
        write(T2,ch);
        if ch=‘.’ then writeln(T2);
    end;
    close(T1); close(T2);
end.
```


Facilidades en archivos de texto (TEXT)

- **Una facilidad del tipo TEXT es que utilizando los procedimientos predefinidos write o writeln puedo escribir elementos de cualquier tipo simple y son transformados a texto automáticamente.**
- **Otra facilidad, es que si se conoce de antemano el formato del archivo (esto es que tipo de valores tiene) entonces utilizando los procedimientos read y readln puedo leer con variables de diferentes tipos simples y se realizará automáticamente la conversión adecuada.**
- **El “buffer” es un archivo text. Todo lo que puede hacer con el buffer lo puede hacer en un text.**

Problema propuesto

- **Considere un archivo de tipo TEXT donde cada línea tiene un formato fijo con los siguientes elementos: un número de LU, la cantidad (N) de notas de un alumno, seguido de una secuencia de N enteros que representan sus notas, y finalmente un texto con nombre y apellido (separados por un guión).**
- **Ejemplo de un posible archivo de texto:**



- **Escriba un procedimiento que dado un número de libreta, si lo encuentra muestre nombre, apellido y el promedio del alumno. Realice un programa de prueba.**

División del problema

Escriba un procedimiento que dado un número de libreta, si lo encuentra muestre nombre, apellido y el promedio del alumno.

Algoritmo busca_alumno:
Datos de entrada: LU y archivo de texto
Recorre el archivo de texto desde el comienzo.
Mientras no llegue al final del archivo o a encontrar el LU hacer: leer un número de libreta
  Si el número leído es el dato de entrada LU entonces: calcular promedio (otra primitiva)
    mostrar nombre (otra primitiva)
    Mostrar promedio
  de lo contrario: saltar al final de la línea del archivo

Descomposición del problema y primitivas


```
Program alumnos;
  procedure buscar_alumno
    function calcular_promedio
    procedure mostrar_nombres
begin
  ...
end.
```

Arriba a la izquierda se muestra una posible división del problema en primitivas. A la derecha una posible implementación de las primitivas en Pascal (donde “calcular_promedio” y “mostrar_nombres” son parte del entorno local de “buscar_alumno”).

Material adicional

El programa completo del problema anterior se encuentra en estos dos archivos:

clase14.pas.pdf

clase14.pas