RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

CLASE 17

Estrategias de resolución de problemas basadas en el uso de primitivas y división del problema

Luciano H. Tamargo
http://cs.uns.edu.ar/~lt
Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2016

CONCEPTOS: TIPOS DE DATOS EN PASCAL

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

Tipo de Dato: define el **conjunto de valores** posibles que puede tomar un elemento de un programa (variable, expresión, parámetro o función), define **las operaciones** que pueden usarse sobre esos valores, y define una **representación interna** para su almacenamiento en memoria.


```
PROGRAM Ejemplo;
TYPE TipoElemento = Integer;
     TipoArch = FILE OF TipoElemento;
VAR F1:TipoArch;
  PROCEDURE mostrarA(VAR archi:TipoArch; separador:char);
  VAR elemento: TipoElemento;
  BEGIN {muestra el contenido de un arch. de nros enteros
         usando un "separador" enviado por parámetro…}
    Reset(archi);
    WHILE not eof(archi) DO
      BEGIN
        read(archi, elemento);
        write(elemento,'  ', separador,'  ');
      END; {while}
    close(archi);
  END;
BEGIN  {programa}
assign(F1, 'mis-numeros.datos');
mostrarA(F1, ';');
END. {fin del programa}
```

Con archivos es obligatorio usar parámetros por referencia

ARCHIVOS COMO PARÁMETROS

- Las **funciones** y los **procedimientos** pueden recibir datos de tipo **FILE** (archivos) como parámetros.
- **Ejemplos**:
  - MostrarContenidoArchivo(archivo1);
  - **IF** Pertenece_elemento(E, archivo1) **THEN** …
  - Copiar_contenido(archivo1, archivo_nuevo);

- En Pascal, es **obligatorio** que un **parámetro** de **tipo archivo** sea un parámetro **por referencia**.
- Dado que los parámetros por referencia deben ser de tipos idénticos, debe crearse un identificador de tipo archivo.

REPASO: COMPATIBILIDAD ENTRE PARÁMETROS

- En un parámetro **POR REFERENCIA**, el tipo del parámetro formal **debe ser idéntico** al tipo del parámetro efectivo. Estos es, se cumple que:
  a) Están declarados con el mismo identificador de tipo.
  b) Los identificadores de tipo son diferentes (ej.: `T1` y `T2`) pero han sido definidos como equivalentes por una declaración de la forma `T1 = T2.`

- De esta forma es **incorrecto**:

```
VAR F1: FILE OF Integer;
…
PROCEDURE ejemplo( VAR A: FILE OF Integer);
…
ejemplo(F1);
```

MAL

A y F1 NO SON DE TIPOS IDÉNTICOS

REPASO: COMPATIBILIDAD ENTRE PARÁMETROS

- En un parámetro **POR REFERENCIA**, el tipo del parámetro formal **debe ser idéntico** al tipo del parámetro efectivo. Estos es, se cumple que:
  a) Están declarados con el mismo identificador de tipo.

- De esta forma es **correcto**:

```
TYPE TipoArchi = FILE OF Integer;
VAR F1: TipoArchi;
…
PROCEDURE ejemplo( VAR A: TipoArchi);
…
ejemplo(F1);
```

BIEN

Ahora A y F1 SI SON DE TIPOS IDÉNTICOS

REFLEXIÓN SOBRE 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?

Esta **importancia** va más allá de RPA, en su vida profesional es muy probable que trabaje **en un equipo.**

SOBRE EL TRABAJO PROFESIONAL FUTURO

- Sin importar la dimensión del problema, existe una gran **responsabilidad** en la correcta **resolución** del mismo.
- Considere por ejemplo las **consecuencias** negativas de una incorrecta resolución de un problema de pequeña escala como *la validación de la identidad del piloto del avión que usted está por abordar*, o la *validación de acceso a transferencias de su propia cuenta bancaria*.
- Un sistema de gran escala (*como reserva y venta de pasajes)* estará formado por un conjunto de soluciones a problemas de pequeña escala (como controlar que una fecha sea correcta).
- Permitir el ingreso y trabajar luego con fechas incorrectas puede tener malas consecuencias.

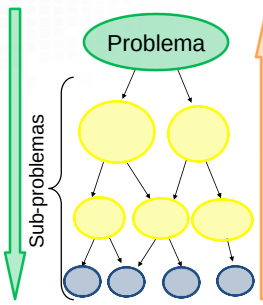
PRIMITIVAS EN EL DESARROLLO DE SOFTWARE

- Si **trabajo en grupo** y tengo a cargo una parte, debo tener una manera de **compartir** esa parte de forma que los demás puedan usarla como una primitiva sin necesidad de conocer los detalles de cómo está hecha.
- Si **trabajo solo** y tengo que abordar un problema que no es trivial (el cual puede ser divido en partes), y además, algunas de esas **partes** pueden "**re-utilizarse**", entonces también necesito una forma de implementar una primitiva.
- Al resolver un problema en el futuro, no solo dispondré de las primitivas predefinidas, si no también las que he construido antes o las de mis compañeros de trabajo.

CONCEPTOS: ESTRATEGIAS "TOP-DOWN" Y "BOTTOM-UP"

Estrategia Top-down:

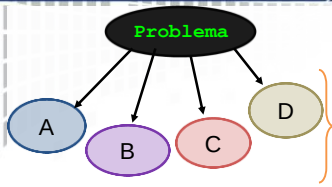
Por **división** del problema principal en subproblemas más simples hasta llegar a problemas que no necesitan dividirse.



Estrategia Bottom-up:

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

DIVISIÓN DE UN PROBLEMA EN SUB-PROBLEMAS




```
Program SOLUCIÓN;
  Function A
  Procedure B
  Function C
  Procedure D
Begin
  …
End.
```

Metología: Para resolver un problema complejo se propone:

1) dividir en **subproblemas**,
2) resolver cada parte y luego
3) para cada parte implementar primitivas en Pascal: **como funciones o procedimientos**

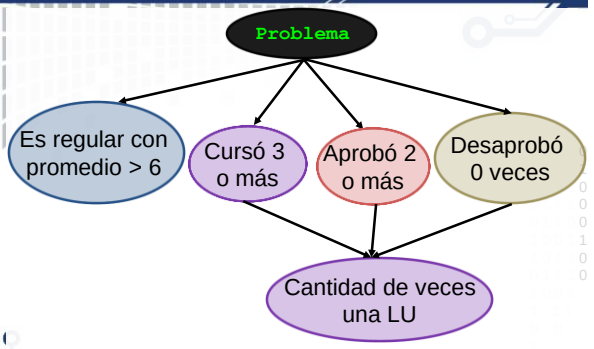
PROBLEMA PROPUESTO

- El Departamento de Ciencias e Ingeniería de la Computación (DCIC) quiere premiar a sus buenos alumnos que quieran ir al Congreso de Desarrolladores de Video Juegos.
- Para aquellos **alumnos** que lo soliciten y **cumplan los requisitos** se les pagará la inscripción y el viaje.
- **Los requisitos son**:
  - ser alumno regular con un promedio mayor a 6, y
  - en el año anterior:
    - haber cursado 3 o más materias,
    - haber aprobado por lo menos dos materias y
    - no tener ninguna materia desaprobada.

PROBLEMA PROPUESTO

- Se desea desarrollar una aplicación que **a partir del archivo** "inscriptos.alu" con los números de LU de los inscriptos, **genere otro archivo** "cumplen.alu" con los inscriptos que cumplen los requisitos.
- Para esta aplicación se dispone del archivo "regulares-más-de6.alu" con los LU de los que son regulares y tienen promedio mayor a 6.
- También se tienen los archivos "cursadas.alu", "aprobadas.alu" y "desaprobadas.alu" que contienen pares (LU - código materia), con las cursadas, aprobadas y desaprobadas del año anterior.
- Se requiere usar archivos de texto para todos los casos.
- Se asume que no hay errores de carga en los archivos.

DIVISIÓN DEL PROBLEMA EN SUB-PROBLEMAS



EJEMPLO / CASO DE PRUEBA

- Estos dos archivos son secuencias de LU
  - inscriptos.alu: 55055 89100 99099 88008 77077 95095 81118
  - regulares-más-de6.alu: 89100 88008 77077 95095 81118
- Estos tres archivos son secuencias de pares LU código_materia
  - cursadas.alu:89100 11 88008 11 77077 22 95095 22 81118 33 89100 33 88008 33 77077 44 89100 44 88008 44 77077 23
  - aprobadas.alu: 88008 11 95095 22 81118 33 89100 33 88008 33 77077 44 89100 44 88008 44 77077 23
  - desaprobadas.alu: 55055 11 99099 22 88008 44
- Con estos datos quedan elegidos las siguientes LU:
  - cumplen.alu: 89100 77077

ALGORITMO GENERAL

Abrir "inscriptos" para leer
Crear "cumplen" para escribir
Mientras hay elementos en "inscriptos"
hacer:
- leer una LU del archivo de inscriptos
- SI pertenece (LU, regulares_mas_de6)
  y cantidad(cursadas, LU) ≥ 3
  y cantidad(aprobadas, LU) ≥ 2
  y cantidad(desaprobadas,LU) = 0
  entonces: agregar esa LU al archivo "cumplen"

cerrar "inscriptos"
cerrar "cumplen"

FUNCTION PERTENCE

```
15 function pertenece (buscado: integer; var archivo: text):boolean;
   {retorna true si el elemento buscado está en el archivo de texto}
   var elemento: integer; encontre: boolean;
   begin
       reset(archivo); encontre:=false;
20     while not eof(archivo) and not encontre do
       begin
         read(archivo, elemento);
         encontre:=elemento=buscado;
       end;
25     pertenece:= encontre;
       close(archivo);
   end;
```


FUNCTION CANTIDAD

```
30 function cantidad (var archivo: text; lu: integer):integer;
   {retorna la cantidad de veces que está una LU en un archivo
    de pares LU materia}
   var elemento,materia, cant: integer;
   begin
35     reset(archivo);   cant:=0;
       while not eof(archivo) do
       begin
         read(archivo, elemento);
         if elemento = lu  then cant:=cant+1;
40       read(archivo, materia);
       end;
       cantidad:=cant;
       close(archivo);
   end;
```

```
program clase15_div_feria_libro;
{Recorre inscriptos y copia los que cumplen los requisitos}
var inscriptos, cumplen, reg_mas_d6, cursadas, aprobadas, desaprobadas: text;
    LU:integer;
function pertenece (buscado: integer; var archivo: text):boolean;
function cantidad (var archivo: text; lu: integer):integer;
begin
assign(inscriptos , 'inscriptos.alu');
assign( reg_mas_d6, 'regulares-mas-de6.alu');
assign(cursadas,'cursadas.alu');
assign(aprobadas , 'aprobadas.alu');
assign(desaprobadas , 'desaprobadas.alu');
assign(cumplen , 'cumplen.alu');
reset(inscriptos); rewrite(cumplen);
while not eof(inscriptos) do
begin
  read(inscriptos,LU);
  if  pertenece(LU, reg_mas_d6)
      and (cantidad(cursadas,LU) >= 3)
      and (cantidad(aprobadas,LU) >= 2)
      and (cantidad(desaprobadas,LU) = 0)
  then write(cumplen,LU,' ')
end;
close(inscriptos); close(cumplen);
writeln('El archivo cumplen.alu fue generado. Presione enter '); readln;
end.
```

PROCEDIMIENTOS Y PARÁMETROS EN PASCAL

- MultiplicarFracciones tiene 6 parámetros formales: 4 por valor (para recibir datos) y 2 por referencia (para devolver datos)

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

Los parámetros por referencia ¿siempre van al final?
Respuesta: no.

¿Puede un procedimiento tener sólo parámetros por valor?

```
PROCEDURE BAJAR_LINEAS(cant:INTEGER);
VAR v:integer;{Deja "cant" líneas en blanco}
BEGIN
FOR v:=1 TO cant DO writeln;
END;
```

PROCEDIMIENTOS Y PARÁMETROS EN PASCAL

¿Puede un procedimiento tener sólo parámetros por referencia?

```
PROCEDURE PasarAmayuscula(VAR L:char);
BEGIN {Si recibe una minúscula, cambia el valor por mayuscula}
IF (L >= 'a') and (L <= 'z') THEN
   L := chr(ord(L) - (ord('a') - ord ('A'))
END; {Si no recibe una minúscula, el valor recibido no se cambia}
```

¿Puede un procedimiento no tener parámetros?

```
PROCEDURE PAUSA;
BEGIN {muestra mensaje y espera por un ENTER del usuario}
writeln('pulse ENTER para continuar');
readln;
END;
```

PROCEDIMIENTOS VS. FUNCIONES

¿Puede un procedimiento tener un único dato de salida?

```
PROCEDURE EsVocal(letra:char; var Es: boolean);
BEGIN {primitiva para identificar letras vocales}
CASE letra OF {si letra es vocal retorna true en ES}
  'A','E','I','O','U','a','e','i','o','u':
   Es:=true;
   ELSE Es:=false;
   END; {o false en caso contrario}
END;
```

¿Qué diferencia hay con tener una función como esta?

```
FUNCTION EsVocal (letra :char): boolean;
BEGIN {primitiva para identificar letras vocales}
CASE letra OF {si letra es vocal la función retorna true}
  'A','E','I','O','U', 'a','e','i','o','u':
   EsVocal:=true;
ELSE EsVocal:=false; END {o false en caso contrario}
END;
```

FUNCIONES Y PARÁMETROS EN PASCAL

¿Puede una función no tener parámetros?

```
FUNCTION leer_letra:CHAR;
VAR aux: char; {Esta función sin parámetros lee del buffer}
BEGIN {hasta que el carácter leído sea una letra y la retorna}
   REPEAT
     read(aux)
   UNTIL (aux>='A') and (aux<='Z') or (aux>='a')
         and (aux<='z')
   leer_letra:= aux
END;
```

- Llamada a la función:

ch:=leer_letra;

FUNCIONES Y PARÁMETROS EN PASCAL

¿ Puede una función tener parámetros por referencia?

```
TYPE TipoElemento = integer;
     ARCHI: FILE OF TipoElemento;
…
FUNCTION cantidad_elementos(VAR A:ARCHI): integer;
VAR aux: TipoElemento; cant:integer;
BEGIN {retorna la cantidad de elementos de un archivo}
cant := 0;
reset(A);
WHILE not eof(A) DO {por cada elemento leído suma uno}
  BEGIN
    read(A,aux);
    cant:=cant+1;
  END;
cantidad_elementos := cant;
close(A);
END;
```

FUNCIONES Y PARÁMETROS EN PASCAL

¿ Puede una función tener parámetros por referencia?

```
{Función que retorna la cantidad de líneas de un archivo de texto}
FUNCTION cantidad_lineas(VAR A:text):integer;
VAR
  cant:integer;
BEGIN
  cant:= 0;
  reset(A);
  WHILE not eof(A) DO
    BEGIN
      readln(A);
      cant:=cant+1;
    END;
  cantidad_lineas := cant;
  close(A);
END;
```

Recuerde que "text" es un tipo predefinido y estructurado (archivo)

CONCEPTO: ENTORNO DE REFERENCIA PARA UN BLOQUE B

El entorno de referencia de un bloque B está formado por los siguientes cuatro entornos:

1. El **entorno local**: conjunto de identificadores (parámetros formales, constantes, tipos, variables, el nombre de los procedimientos y funciones) *declarados* dentro del **bloque B**.
2. El **entorno global**: conjunto de identificadores *declarados* en el bloque del programa principal.
3. El **entorno no-local**: conjunto de identificadores *declarados* en los bloques que contienen al **bloque B**, exceptuando al global.
4. El **entorno predefinido**: conjunto de identificadores ya *declarados* por el compilador de Pascal y disponible para todo programa (Ejemplos de identificadores predefinidos: maxint, char, write, eof).

EJEMPLO DE ENTORNO DE REFERENCIA

```
PROGRAM Ejemplo; {para ejercitar el concepto de entorno}
CONST min= 0; max=9; TYPE TRango = min..max;
VAR DelPrograma: integer; global: real;

PROCEDURE Uno (var digito:Trango);
VAR aux: Trango;
BEGIN aux:=digito; digito:= aux+max; END;

PROCEDURE Dos (aux:integer; var num:Trango; );
  FUNCTION en_rango (aux:Integer):boolean;
  BEGIN en_rango= (aux>=min) and (aux <=max) END;
BEGIN IF en_rango(aux) THEN num:=aus ELSE num:=min
END;

BEGIN
dos(8,global); uno(global); dos(global,delprograma);
END.
```

Calcular el entorno de cada uno de los cuatro bloques del programa.

IMPORTANTE: PROHIBICIÓN EN RPA

- En cualquier función o procedimiento definida por el programador:
  - está **permitido usar constantes, tipos, procedimientos, y funciones** que fueron declarados en su **entorno local, global o en el entorno no-local.**
  - también **puedo usar variables o parámetros del entorno local.**
- Sin embargo,…
  - En RPA, en los procedimientos y funciones, se PROHIBE y será considerado un error el uso de variables globales, o variables declaradas en un entorno no-local.
  - El uso de variables declaradas en otros entornos que no sea el local no es una buena pauta de programación y **lleva a cometer errores de programación que son muy difíciles de encontrar**.


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

MAL: usa una variable global

En el programa **incorrecto**:

- La variable "i" es usada en "linea" como global.
- Esto afecta al resultado esperado ya que "línea" modifica la variable i del programa.

```
--------------------
Ingrese un nro: 16
--------------------
raíz de 25 es 5.00
```

- **Solución**: crear una variable "i" local.
- Además, es evidente que la variable global "i" debería tener un nombre más significativo (vea el programa a continuación)

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

variable global, solo usada en código del programa

variable local, solo usada en "línea"

```
--------------------
Ingrese un nro: 16
--------------------
raíz de 16 es 4.00
```

- **Observe** que ahora no hay error, ya que el procedimiento línea no modifica la variable del programa.

IMPORTANTE: PROHIBICIÓN EN RPA

- En RPA, en los procedimientos y funciones, se PROHIBE y será considerado un error el uso de variables globales, o variables declaradas en un entorno no-local.

- No olvide esto: siempre que surja la necesidad de usar variables globales es porque tendría que usar una constante, una variable local o un parámetro.

PROBLEMA PROPUESTO COMO TAREA

- Realizar un programa que muestre el contenido de un archivo de enteros llamado 'mis-numeros.datos', luego solicite al usuario un elemento E, elimine todas las apariciones E, y vuelva a mostrar el contenido del archivo. Esta operación podría repetirse cuantas veces el usuario quiera.

Escriba casos de prueba


```
PROGRAM Eliminar; {una posible solución: completar lo que falta}
TYPE TipoEle = Integer;
     TipoArch = FILE OF TipoEle;
VAR F1:TipoArch; Elem: tipoEle;
 PROCEDURE mostrarArchivo(VAR archi:TipoArch; separador:char);
 PROCEDURE EliminarDeArchivo(E:TipoEle;VAR original:TipoArch);
 var auxiliar: TipoArch;
  PROCEDURE pasar(E:TipoEle; VAR original, aux: TipoArch);
  {...pasa todos los elementos que son distintos de E al archivo auxiliar...}
  PROCEDURE copiar(VAR origen, destino: TipoArch);
  {... hace una copia idéntica del archivo origen en destino... }
begin
 assign(F1, 'mis-numeros.datos');
 repeat
   mostrarArchivo(F1, ';');
   writeln(' Ingrese elemento a eliminar');  readln(Elem);
   EliminarDeArchivo(Elem,F1);
 until ….  // completar lo que falta
end.
```

PRIMITIVA PARA MOSTRAR UN ARCHIVO EN PANTALLA

```
PROCEDURE mostrarArchivo(VAR archi: TipoArch;
                         separador:char);
VAR elemento: TipoElemento;
BEGIN {...muestra el contenido de un archivo usando un
       "separador" enviado por parámetro...}
 Reset(archi);
 WHILE not eof(archi) DO
   BEGIN
     read(archi, elemento);
     write(elemento,'  ', separador ,'  ' );
   END; {while}
 close(archi);
END;
```


PRIMITIVA ELIMINAR ELEMENTO DE UN ARCHIVO

```
PROCEDURE EliminarDeArchivo(E: TipoElemento;
                            VAR original:TipoArch);
{...Elimina del archivo todas las apariciones de "E"...}
VAR aux:TipoArch;
 PROCEDURE pasar(E:TipoElemento;VAR original,
                                 aux: TipoArch);
 {pasa todos los elementos que son distintos de E al archivo auxiliar}
 PROCEDURE copiar(VAR origen, destino: TipoArch);
 {...hace una copia idéntica del archivo origen en destino...}
BEGIN
 assign(aux, 'auxiliar.tmp');
 pasar(E, original, aux);
 copiar(aux, original);
END;
```


PRIMITIVA PASAR

```
PROCEDURE pasar(E: TipoElemento;
                VAR original, aux: TipoArch);
{pasa todos los elementos que son distintos de E al archivo auxiliar}
VAR
elemento: TipoElemento;
BEGIN
 Reset(original);
 rewrite(aux);
 WHILE not eof(original) do
  BEGIN
   read(original,elemento);
   IF elemento <> E THEN write(aux, elemento);
   END;
 close(original);
 close(aux);
END;
```

PRIMITIVA PARA "DUPLICAR" UN ARCHIVO

```
PROCEDURE copiar(VAR origen,destino: TipoArch);
{...hace una copia idéntica del archivo origen en destino... }
VAR
elemento: TElemento;
BEGIN
  Reset(origen);
  rewrite(destino);
  WHILE not eof(origen) DO
    BEGIN
      read(origen,elemento);
      write(destino, elemento);
    END;
close(origen);
close(destino);
END;
```

PROBLEMA PROPUESTO

- Considere que un grupo de aseguradoras comparte su información, cada una tiene 2 archivos "sin_siniestro" y "morosos", con los DNI de sus clientes ya sean actuales o anteriores.
- Considere que dispone de los 6 archivos de 3 aseguradoras. Una cuarta aseguradora quiere consultar esos archivos para poder hacer un descuento a un nuevo cliente. La aseguradora hará el descuento si el cliente estuvo sin siniestro en al menos una de las otras 3 y nunca ha sido moroso en las otras 3 aseguradores.

Escriba casos de prueba

PROBLEMA PROPUESTO COMO TAREA