Resolución de Problemas y Algoritmos

Clase 16:
Estrategias de resolución de problemas basadas en el uso de primitivas y división del problema

Dr. Diego R. García

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

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.**

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

Ejemplo

```
program ejemplo;
 Var letra: char;
 procedure pasarAMayuscula …
  {Si recibe una minúscula, cambia el valor por mayúscula,
   de lo contrario, el valor recibido no se cambia} …
begin
…
Repeat {validación de ingreso de datos: Letras N, S, E, O}
    Write('ingrese una de estas letras: N, S, E, O ');
    read(Letra);
    pasarAMayuscula(Letra); {paso a mayúscula si es minúscula}
Until (letra = 'S') or (letra='N') or (letra='E') or (letra='O');
…
end.
```

Copiar en el pizarrón (y sus notas).

Indique casos de prueba.

Realice una traza para cada una de las opciones porpuestas para el procedimiento y mostradas a continuación, indique cuál es correcta y cuál incorrecta, explicando el error de programación cometido.

Parámetros en Pascal: ¿cuál es el correcto?

```
PROCEDURE Opcion1(L:char);
Var Mayu: char;
begin  { Si recibe una minúscula, cambia el valor por mayúscula}
if (L >= 'a') and (L <= 'z')  then Mayu := chr(ord(L) - (ord('a') - ord ('A'))
end; {Si no recibe una minúscula, el valor recibido no se cambia}
```

Error: "Mayu" es variable local a opción1 y no es accesible desde el programa.

MAL

```
PROCEDURE Opcion2( L:char);
Var Mayu: char;
begin {  Si recibe una minúscula, cambia el valor por mayúscula}
  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}
```

Error: "L" es parámetro por valor y no modifica el valor del parámetro efectivo.

MAL

```
PROCEDURE Opcion3(VAR L:char);
Var Mayu: char;
begin {Si recibe una minúscula, cambia el valor por mayúscula}
  if (L >= 'a') and (L <= 'z')  then Mayu := chr(ord(L) - (ord('a') - ord ('A'))
end; {Si no recibe una minúscula, el valor recibido no se cambia}
```

Error: "Mayu" es variable local a opción3 y no es accesible desde el programa.

MAL

Parámetros en Pascal: ¿cuál es el correcto?

```
PROCEDURE Opcion4(L:char);
Var Mayu: char;
begin {Si recibe una minúscula, cambia el valor por mayúscula}
  if (L >= 'a') and (L <= 'z')  then L := chr(ord(L) - (ord('a') - ord ('A'));
  write(L);
end; {Si no recibe una minúscula, el valor recibido no se cambia}
```

Error: "L" es parámetro por valor y no modifica el valor del parámetro efectivo.

MAL

Error: El valor de "L" es mostrado por pantalla pero no modifica ninguna variable del programa.

```
PROCEDURE Opcion5(VAR L:char);
begin {Si recibe una minúscula, cambia el valor por mayúscula}
  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}
```

BIEN

¡Correcto! ☺ "L" es parámetro por referencia y modifica el valor del parámetro efectivo.

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.

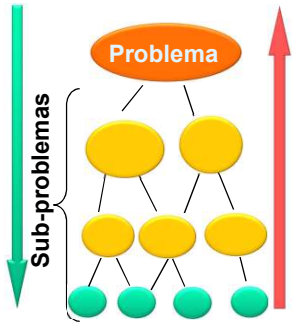
Primitivas en el desarrollo de software

- Si trabajo en grupo y tengo a cargo una parte, debo tener una manera 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.



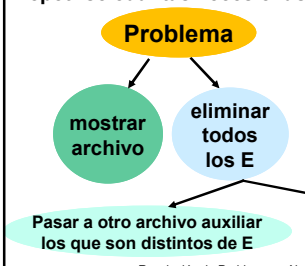
Estrategia Bottom-up:

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

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.

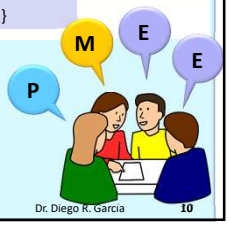
Escriba casos de prueba




```
Repetir
    mostrar archivo  (primitiva)
    solicitar elemento E
    eliminar todos los E  (primitiva)
    mostrar archivo
hasta que el usuario lo decida
```



```
Program Eliminar; {una posible solución: completar lo que falta}
TYPE TipoElemento = Integer;   TipoArch: FILE OF TipoElemento;
VAR F1:TipoArch; Elem: tipoElemento;

  PROCEDURE mostrarArchivo (  VAR archi: TipoArch; separador:char);

  PROCEDURE EliminarDeArchivo(E:TipoElemento; VAR original:TipoArch);
  var auxiliar: TipoArch;

    PROCEDURE pasar ( E:TipoElemento; VAR original, auxiliar: 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);
      mostrarArchivo(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 auxiliar:TipoArch;

  PROCEDURE pasar ( E:TipoElemento; VAR original, auxiliar: 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(auxiliar, 'auxiliar.tmp');
    pasar(E, original, auxiliar);
    copiar(auxiliar, original);
End;
```

Primitiva pasar

```
PROCEDURE pasar ( E:TipoElemento;
                  VAR original, auxiliar: TipoArch);
{pasa todos los elementos que son distintos de E al archivo auxiliar}
Var elemento: TipoElemento;
begin
   Reset(original); rewrite(auxiliar);
   while not eof(original) do begin
     read(original,elemento);
     if elemento <> E then write(auxiliar, elemento);
   end;
   close(original); close(auxiliar);
End;
```

Primitiva para "duplicar" un archivo

```
PROCEDURE copiar (VAR origen, destino: TipoArch);
{... hace una copia idéntica del archivo origen en el
archivo 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

La universidad quiere premiar a sus buenos alumnos que quieran ir a visitar la Feria del Libro. 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.
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. EJEMPLO
Se requiere usar archivos de texto para todos los casos. ESQUEMA
Se asume que no hay errores de carga en los archivos. ALGORITMO

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

Enunciado

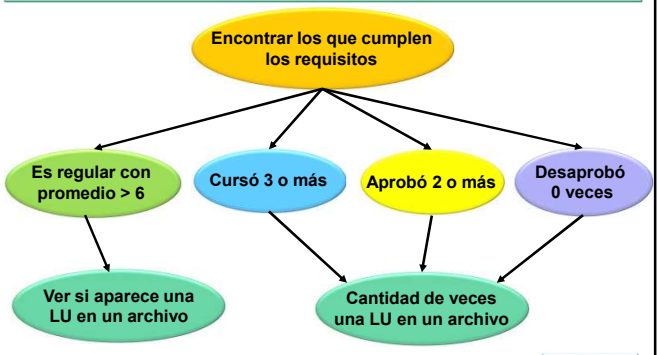
Algoritmo general

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"

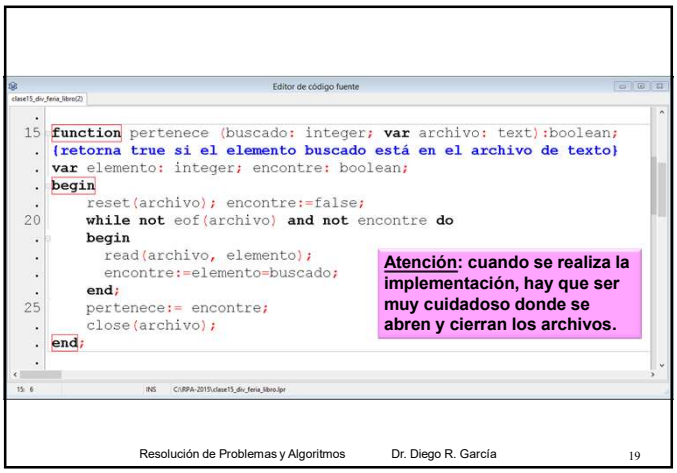
cerrar "inscriptos"
cerrar "cumplen"

Enunciado

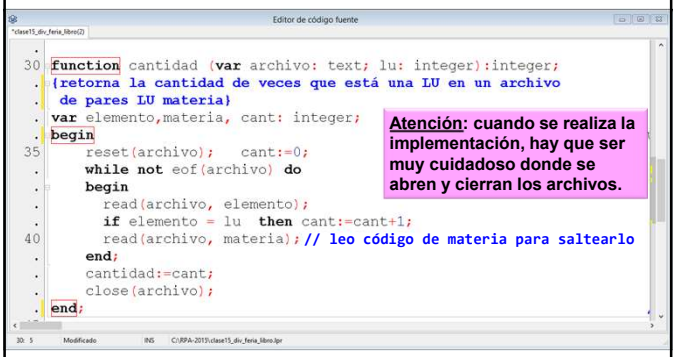
División del problema en sub-problemas



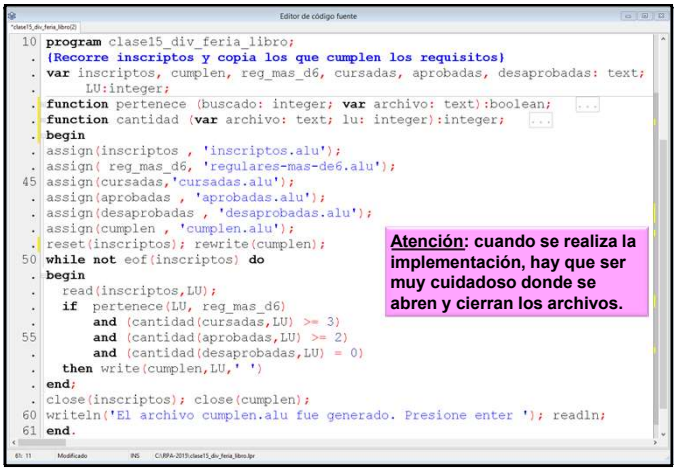
Enunciado



Resolución de Problemas y Algoritmos Dr. Diego R. García 19



Resolución de Problemas y Algoritmos Dr. Diego R. García 20



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

Resolución de Problemas y Algoritmos Dr. Diego R. García 22

Algoritmo general

Leer DNI del cliente

SI no pertenece (DNI, morosos_comp1)
  y no pertenece (DNI, morosos_comp2)
  y no pertenece (DNI, morosos_comp3)
  y (pertenece (DNI, sin_siniestro_comp1)
    o pertenece (DNI, sin_siniestro_comp2)
    o pertenece (DNI, sin_siniestro_comp3)
)
entonces: hacer el descuento a DNI

Atención: cuando se realiza la implementación, hay que ser muy cuidadoso donde se abren y cierran los archivos.

Resolución de Problemas y Algoritmos Dr. Diego R. García 23

Problema propuesto como tarea

Resolución de Problemas y Algoritmos Dr. Diego R. García 24