Resolución de Problemas y Algoritmos

Clase 10:
Definición y compatibilidad de tipos de datos.
Funciones definidas por el programador.

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

Tipos de datos en Pascal

- Pascal es un lenguaje **fuertemente tipado**.
- Esto implica que el *tipo de dato* de todas las variables debe ser declarado.
- Lo que se busca con esto es prevenir errores.

Además, hoy veremos que:
(1) el programador puede definir NUEVOS tipos de datos,
(2) no sólo las variables tienen un tipo de dato asociado,
(3) existe una forma de relacionar los diferentes tipos de datos de Pascal de acuerdo a su "compatibilidad".

Tipos definidos por el programador

- Poder definir y usar tipos de datos fue un muy importante avance en la evolución de los lenguajes de programación.
- Permiten dar claridad al código fuente.
- Dan información al compilador, que puede ser usada para prevenir errores, y además generar un mejor código.
- Hay compiladores que realizan un chequeo de tipos al compilar, otros al ejecutar, y otros en ambos momentos (en algunos casos se puede configurar).
- IMPORTANTE: en esta materia vamos a usar solamente algunas de las ventajas poder definir nuevos tipos.
- Más adelante, en otras materias de su carrera descubrirá muchas más.

Tipo de Dato: define el conjunto de valores posibles que puede tomar una variable, y también define las operaciones que pueden usarse sobre esos valores.

En Pascal

Tipos **Predefinidos**
- **Ejemplos vistos hasta ahora:**
- –**REAL**
- –**BOOLEAN** (ordinal)
- –**CHAR** (ordinal)
- –**INTEGER** (ordinal)

Tipos **definidos por el programador**
- –**FILE OF…** (estructurados)
- – **Subrangos** (de ordinales)
- – otros que no se ven en RPA

Definir nuevos tipos permite claridad y abstracción.
Dos conceptos fundamentales en el desarrollo de Software

Tipos subrangos

```
TYPE
  LetrasMayusculas = 'A' .. 'Z';
  Numeros_de_Mes = 1..12;
  Digitos = 0..9;
```

Palabra reservada que indica la sección de declaración de nuevos tipos.

Rango de valores posibles para el tipo: todos los comprendidos entre los indicados.

Nombre de un nuevo tipo (identificador)

En Pascal, se pueden definir nuevos **tipos subrangos**, indicando:en primer lugar un identificador como nombre para el nuevo tipo, luego el símbolo "=", y finalmente, separados por un par de puntos consecutivos ".." un valor inicial y un valor final de algún tipo **ordinal**. Estos valores definen el "RANGO" de todos los valores posibles para los elementos de este nuevo tipo de dato.

Nuevos tipos de datos definidos por el programador

```
PROGRAM Ejemplo;
CONST meses=12;
TYPE
  Tipo_Digito = 0..9;    TNumMes = 1..meses;
  TNumDia = 1..31;       TNumDeCarta = 1..12;
  LetrasMayusculas = 'A' .. 'Z';
  LetrasMinusculas = 'a' .. 'z';
VAR
  digito: tipo_digito;
  num: integer;
  carta: TnumDeCarta;
  Inicial: LetrasMayusculas;
BEGIN
  digito:= 3; carta:=12; Inicial:='A'; num:=digito;
```

Declaración de constantes

Declaración de tipos

Declaración de variables

Otra forma de definir nuevos tipos de datos

```
TYPE
  Archivos = FILE OF char;
  Numeros = integer;
  Productos = FILE OF numeros;
```

Rango de valores posibles para el nuevo tipo: los mismos del tipo indicado a la derecha del símbolo "=".

Nombres para nuevos tipos (identificadores)

También se pueden definir nuevos **tipos**, indicando: un identificador como nombre para el nuevo tipo, luego el símbolo "=", y luego algún tipo predefinido, tipo estructurado, o tipo definido por el usuario antes. Los valores posibles para los elementos de este nuevo tipo de dato, son los valores del tipo usado a la derecha del "=".

Esto será necesario para elementos que vamos a introducir en las clases que siguen.

Tipos definidos por el programador

```
PROGRAM Ejemplo;
TYPE  Entero = integer;  Logico = Boolean;
      NroReal = real;  Letra = char;
      Telemento = Entero;
      TipoArchivo = FILE OF Telemento;
VAR   Inicial: Letra;  Es_Par: Logico;
      Num: Telemento ;
      Archivo1, Archivo2: TipoArchivo;
BEGIN
Inicial := 'A'; Num:= 4;
Es_Par := (Num MOD 2 ) = 0
Assign(Archivo1,'num.dat');
rewrite(archivo1);
write(Archivo1,num);
```

Declaración de tipos

Declaración de variables

Relaciones entre tipos de datos

Considere las siguientes declaraciones:

```
TYPE TipoCaraDado = 1..6;
VAR Tirada: TipoCaraDado;
    Caida: 1..6;
    resultado: Integer;
```

1) ¿Tienen Tirada y Caida el mismo tipo?

2) ¿Es posible escribir lo siguiente?

```
resultado := resultado + Tirada + Caida;
```

Para responder estas preguntas es necesario introducir algunos conceptos teóricos.

Estos conceptos son necesarios para futuras clases.

Relaciones entre tipos de datos

En Pascal existen tres relaciones entre tipos de datos:

1. **Igualdad o Identidad.**
2. **Compatibilidad.**
3. **Compatibilidad de asignación.**

(1) Tipos idénticos en Pascal

Dos elementos tienen **tipos idénticos** si se cumple una de las siguientes opciones:

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 .**

Ejemplo: ¿Cuáles variables tienen tipos idénticos?

```
TYPE  T = INTEGER;
      T1 = T;
VAR   A, A1: T;   A2: REAL;
      B: INTEGER;
      C: T1;
      D: -32768 .. 32767;
```


(2) Tipos compatibles en Pascal

Dos tipos son **compatibles** si al menos una de las siguientes opciones es verdadera:

a) Ellos son idénticos.

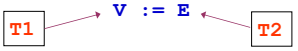
b) Uno es subrango del otro.

c) Ambos son subrangos del mismo tipo.

Ejemplo: ¿Cuáles son compatibles?

```
TYPE  T = Integer;   Sub = 1..1000;   Sub2 = Sub;
      Sub1 = 100..200;  Sub3 = 0..99;  TipoNum = Real;
VAR   A: Sub;   B: INTEGER;   C: Sub1;
      D: Sub2;   E: Sub3;   F: TipoNum;
```

(3) Compatibilidad de asignación



Una expresión `E` de tipo `T2` es asignación-compatible con el identificador `V` de tipo `T1` si al menos una de las siguientes declaraciones es verdadera:

1) `T1` y `T2` son idénticos.

2) `T1` es real y `T2` es entero o subrango de entero.

3) `T1` y `T2` son subrangos o enteros, y el valor de `E` es un valor permitido del tipo `T1`.

Palabras Reservadas y elementos predefinidos

Estas son todas las palabras reservadas de Pascal (están **resaltadas** las que veremos en RPA, y **subrayadas** las que aún tenemos que ver):

and	**end**	nil	set
array	**file**	**not**	**then**
begin	**for**	**of**	**to**
case	**function**	**or**	**type**
const	goto	packed	**until**
div	**if**	**procedure**	**var**
do	in	**program**	**while**
downto	label	record	with
else	**mod**	**repeat**	

Motivación

Indique a que elemento de Pascal corresponde cada uno de estos identificadores, y agregue un ejemplo más a cada columna de la tabla:

then	**integer**	**TRUNC**	**WRITE**
and	**real**	**EOF**	**READ**
program	**char**	**CHR**	**RESET**

Funciones y procedimientos predefinidos

Al programar en Pascal han usado primitivas predefinidas.

Por ejemplo:

```
…
assign(F1,'mi-archivo');
WRITELN ('ingrese un número'); READLN (num);
RESET(F1); REWRITE(F2);
while not EOF(F1) do begin
  READ (f1,elem);
  if TRUNC(elem) > SQR(num)
   then WRITE (F2, elem)
   else WRITELN ( TRUNC (elem), ' menor que' , SQR (num) );
end;
…
```

primitivas como sentencias

primitivas en expresiones

¿Quién creó el código de esas primitivas?

Sobre el trabajo profesional futuro

Como profesional, un informático debe tener la capacidad de resolver problemas utilizando computadoras.
Estos problemas pueden ser

- de muy gran escala*: como por ejemplo mantener en órbita a la estación espacial internacional*;
- de gran escala: por ejemplo desarrollar un sistema de reserva y venta de pasajes de una compañía aérea;
- o de pequeña escala *como validar la identidad de un usuario mediante su nombre y clave, o controlar que una fecha ingresada en un formulario de una página web sea correcta*.

Reflexione un momento…

- ¿Cómo será el trabajo profesional si hay que desarrollar software a gran escala?
- ¿ Trabajo en grupo?
- ¿ Tengo a cargo una parte?
- ¿ Me relaciono con otros?
- ¿ Tardo varios días en hacer parte ?

- Los lenguajes de programación evolucionaron para permitir que los programadores puedan hacer sus propias primitivas, compartir y "re-utilizar" el código lo más posible y de la mejor manera posible.

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 definidas por mi o mis compañeros de trabajo.

Funciones en Pascal

Reflexionemos sobre las Funciones en Pascal:

- **Se utilizan en una expresión.**
- **Siempre retornan un valor de un tipo de Pascal.**

Ejemplos de algunas funciones predefinidas :

- **EOF(F): recibe un manejador y retorna boolean**
- **TRUNC(R): recibe real y retorna integer**
- **SQRT(R): recibe real y retorna real**
- **CHR(I): recibe integer y retorna char**

¿Podré construirme mis propias funciones?

Procedimientos en Pascal

Reflexionemos ahora sobre procedimientos:

- **Se los usa en una sentencia.**
- **Pueden tener 0 o más parámetros.**

Ejemplos de algunos procedimientos predefinidos en Pascal:

- **Writeln**
- **Assign (F, 'nombre')**
- **reset (F)**
- **Read (F, A)**

¿Podré construirme mis propios procedimiento?

Problemas propuestos: lo más simple posible :)

- **Problema 1: Escribir un programa que solicite al usuario tres números entre 1 y 200: N1, N2, y N3. El programa deberá mostrar por pantalla la raiz cuadrada de cada número. (puedo usar SQRT …)**
- **Problema 2: Escribir un programa que solicite al usuario tres números entre 1 y 200: N1, N2, y N3. El programa deberá mostrar por pantalla todas las combinaciones de un número elevado con otro. Esto es, N1 elevado a la N2, N1 a la N3, N2 a la N3, N2 a al N1, N3 a la N1 y N3 a la N2.**
- **La solución al problema 2 requiere de poder programar la potenciación (que ya lo hemos hecho).**

"sub-problema" del propuesto

La potenciación es una operación matemática entre dos términos denominados: base (b) y exponente (e). Se escribe b^e y se lee usualmente como "b elevado a la e".

- **Problema: escriba una función "potencia" que reciba dos enteros positivos (base y exponente) y retorne $base^{exponente}$.**

Solución: Multiplicar el valor de base "exponente" veces

Algoritmo:

resultado:=1;

Repetir exponente veces:

resultado:=resultado*base.

Función Potencia

Algoritmo: potencia
Datos de entrada: base y exponente
Dato de salida: resultado
resultado:=1;
Repetir exponente veces:
resultado:=resultado*base.

Para implementar este algoritmo como una función en Pascal, hay una parte que ya hemos visto:

```
resultado := 1;
FOR i:= 1 TO exponente DO resultado:= resultado*Base;
```

Veremos ahora como indicar el resto de los elementos para obtener una función en Pascal.

En Pascal toda función tiene:

1. Un **nombre** (con el cual se la invocará desde una expresión).
2. **Parámetros** (entre los cuales estarán los datos de entrada).
3. **Tipo del resultado** (que será el tipo de la función y determinará en que expresión podrá ser usada)
4. **Variables locales** (que son propias de la función).
5. Sentencias (también llamado "**cuerpo**" de la función).
6. **Asignación** de una expresión al **nombre** de la función (al menos una vez). Es la forma de retornar un valor.

```
                1          2          3
FUNCTION Potencia (Base, Exponente:integer) : integer;
VAR aux,resultado: integer;   4
BEGIN
  resultado := 1;
5 FOR aux:= 1 TO Exponente DO resultado := resultado * Base;
  Potencia:= resultado;   6
END;
```



```
PROGRAM prueba; {prueba la función potencia}
VAR num1,num2: integer;
FUNCTION Potencia(Base,Exponente:integer) : integer;
{utilidad: calcula "base" a la "exponente"}
VAR aux,resultado: integer;
BEGIN
  resultado := 1;
  FOR aux:= 1 TO Exponente DO
        resultado := resultado * Base;
  Potencia:= resultado;
END;
BEGIN
writeln('ingrese dos enteros positivos);
readln(Num1,Num2);
write(Num1,' a la ',Num2,' es ',potencia(Num1,Num2) );
END.
```

Haga la traza (o copie del pizarrón)

Variables globales
Tipo de la función
Parámetros formales
Comentario
Variables Locales
Asignación del resultado
Llamada a la función

Volviendo sobre el problema propuesto

- **Problema 2: Escribir un programa que solicite al usuario tres números entre 1 y 200: N1, N2, y N3. El programa deberá mostrar por pantalla todas las combinaciones de un número elevado con otro. Esto es, N1 elevado a la N2, N1 a la N3, N2 a la N3, N2 a al N1, N3 a la N1 y N3 a la N2.**
- **Con la función potencia puedo resolver una parte.**
- **¿Puedo hacer algo similar con la entrada y validación de datos?**
- **Ejemplo: FUNCTION leer_y_validar(v1,v2:integer):integer;**
- **De manera tal que pueda hacer llamadas de este estilo:**

```
N1:=leer_y_validar(1,200);
N2:=leer_y_validar(1,200);
```



```
PROGRAM problema2;
CONST menor=1; mayor=200;
VAR n1,n2,n3: integer;
FUNCTION Potencia(Base,Exponente:integer) : integer;
…
FUNCTION leer_y_validar(menor,mayor:integer): integer;
…
BEGIN
N1:=leer_y_validar(menor, mayor);
N2:=leer_y_validar(menor, mayor);
N3:=leer_y_validar(menor, mayor);
write(N1,' a la ',N2,' es ',potencia(N1,N2) );
write(N1,' a la ',N3,' es ',potencia(N1,N3) );
…
END.
```

Funciones en Pascal

Para llamar (invocar o usar) a una función se debe:

1) **Utilizar en una expresión.**
2) **Coincidir la cantidad de parámetros y el tipo de dato de cada uno de los parámetros.**
3) **El tipo del resultado debe ser compatible con el tipo de la expresión en la que se lo llama.**

Observación: en una sentencia de asignación

identificador := expresión

El nombre de una función a la izquierda del := se usa para darle valor a la función.

El nombre de una función usado en la expresión a la derecha del := es usado para llamar a la función

Invocación a funciones (en expresiones)

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN  IF ('A' <= letra) and (letra <= 'Z' )
          THEN EsMayuscula:=true
          ELSE EsMayuscula:=false
END;
```

Algunos ejemplos de invocación (siempre en expresiones):

```
VAR ch: char;   es_mayu:boolean;
...
read(ch);
es_mayu := EsMayuscula(ch);
writeln(EsMayuscula(ch));
IF (EsMayuscula(ch) or (ch='@'))
   THEN …
WHILE not EsMayuscula(ch) and not  EOF(…)  DO …
```


En Pascal, para que una función retorne un valor, se debe ejecutar al menos 1 vez, una asignación que en su parte izquierda tenga el nombre de la función y a la derecha una expresión del mismo tipo que la función.

```
FUNCTION Cubo (N:integer):integer;
BEGIN
Cubo:= N*N*N;
END;
```

CORRECTO

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
   IF ('A' <= letra) and (letra <= 'Z' )
   THEN EsMayuscula:=true
   ELSE EsMayuscula:=false
END;
```

CORRECTO

Para que una función retorne un valor, se debe ejecutar al menos 1 vez, una asignación que en su parte izquierda tenga el nombre de la función.

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
   EsMayuscula:=false;
   IF ('A' <= letra) and (letra <= 'Z' )
   THEN EsMayuscula:=true
END;
```

CORRECTO

```
FUNCTION EsMayuscula (letra :char): boolean;
BEGIN
   IF ('A' <= letra) and (letra <= 'Z' )
   THEN EsMayuscula:=true
END;
```

INCORRECTO: cuando no es mayúscula no se ejecuta la asignación de valor a la función.

Si no hay una asignación de ese tipo, o ninguna se ejecuta, entonces es un error de programación.

```
FUNCTION Cubo (N:integer):integer;
VAR aux: integer;
BEGIN
aux:= N*N*N;
END;
```

INCORRECTO: falta la asignación de valor a la función.

```
FUNCTION Cubo (N:integer):integer;
VAR aux,P: integer;
BEGIN
  Cubo := 1;
  FOR aux:= 1 TO 3
     DO Cubo := Cubo * N;
END;
```

INCORRECTO: "Cubo" no es una variable, es el nombre de la función. Si usa el nombre de la función en una expresión (como aquí) entonces ¡está llamando a la función!

Continuará …