Pilas. Uso (I)

Transformación de expresiones aritméticas de notación infija a postfija.

Ejemplo:

$$6 + 3 * 2 = 6\ 3\ 2\ *\ +$$

Se realiza en una sola pasada, usando una pila donde se van apilando los operadores.

Antes de apilar un operador se desapilan los que tengan una prioridad mayor o igual a la del nuevo operador

Pilas. Uso (I)

```
PROCEDURE InfijaPostfija(cadInf: String; VAR cadPost: String);
VAR
   posicion: integer;
      p: TipoPila;
   c: char;
BEGIN
   cadPost  := '';
   posicion := 1;
   CrearPilaVacia(p);
   WHILE (posicion <= length(cadInf)) DO {iteramos por la cadena}
   BEGIN
      CASE cadInf[posicion] OF
        { Se copia el numero en la cadena de salida}
      '0'..'9':
            cadPost := cadPost + LeerNumero(cadInf,posicion);
```

Pilas. Uso (I)

```
'+','-','*','/':
BEGIN
   IF NOT EsPilaVacia(p) THEN  {si hay operadores}
      BEGIN
      { Desapilar operador y evaluar >= prioridad }
         Cima(c,p);
      WHILE (NOT EsPilaVacia(p)) AND (Prioridad(c) >=
                     Prioridad(cadInf[posicion])) DO
         BEGIN
            cadPost := cadPost + c +' '; Desapilar(p);
            IF NOT EsPilaVacia(p) THEN {sacar y evaluar}
                     Cima(c,p);
      END;
    END;
    { se apila nuevo operador }
    Apilar(cadInf[Posicion],p);
       Inc(posicion);
END;
```

Pilas. Uso (I)

```
' ':  Inc(posicion);
  END; {CASE}
END; {WHILE}
 { Desapilamos los operadores que queden en la pila }
   WHILE NOT EsPilaVacia(p) DO
   BEGIN
      Cima(c,p);
         cadPost := cadPost + c + ' ';
      Desapilar(p)
   END
END;
```

- Si consideramos el uso de paréntesis o un convenio de evaluación de derecha a izquierda en una potencia

Pilas. Uso (I)

Por ejemplo:

2 ^ 2 ^ 3 = 2 ^ (2 ^ 3) = 2 ^ 8 <> (2 ^ 2) ^ 3 = 4 ^ 3

- El paréntesis tiene la mayor prioridad en la evaluación hasta que llegue el cierre (le damos prioridad baja en la pila para que no salga hasta que llegue el cierre)

Operador	Prioridad dentro	Prioridad fuera
^	3	4
*, /	2	2
+, -	1	1
(	0	5

Pilas. Uso (I)

```
PROCEDURE InfijaPostfija2(cadInf:String; VAR cadPost:
  String);
VAR
    posicion: integer;
       p: TipoPila;
    c: char;
BEGIN
   cadPost  := '';
    posicion := 1;
    CrearPilaVacia(p);
    WHILE posicion<= length(cadInf) DO
    BEGIN
       CASE cadInf[posicion] OF
         { Se copia el numero en la cadena de salida}
       '0'..'9':
             cadPost := cadPost + LeerNumero(cadInf,posicion);
```

Pilas. Uso (I)

```
      '(','+','-','*','/','^':
BEGIN
   IF NOT EsPilaVacia(p) THEN
      BEGIN
      { Desapilamos los operadores con prioridad >= }
         Cima(c,p);
      WHILE (NOT EsPilaVacia(p)) AND (PrioridadDentro(c) >=
                     PrioridadFuera(cadInf[posicion])) DO
         BEGIN
            cadPost := cadPost + c +' '; Desapilar(p);
            IF NOT EsPilaVacia(p) THEN
                     Cima(c,p);
       END;
      (* Apilamos el nuevo operador*)
      Apilar(cadInf[Posicion],p);
        Inc(posicion);
    END;
END;
```

Pilas. Uso (I)

```
    ')':
      BEGIN
        { Se desapila hasta encontrar '('  }
            Cima(c,p);
            WHILE NOT EsPilaVacia(p) AND (c <>'(') DO
          BEGIN
                  cadPost := cadPost +c+' ';
                  Desapilar(p);
                IF NOT EsPilaVacia(p) THEN
                    Cima(c,p);
      END;
          Desapilar(p); Inc(posicion);
       END;
      ' ':  Inc(posicion);
  END; {CASE}
END; {WHILE}
```

Pilas. Uso (I)

```
    { Desapilamos los operadores que queden en la pila }
    WHILE NOT EsPilaVacia(p) DO
    BEGIN
       Cima(c,p);
          cadPost := cadPost + c + ' ';
       Desapilar(p)
    END
END;
```

Pilas. Uso (II)

Evaluación de Expresiones Aritméticas

- Mientras haya elementos en la expresión
  - 1. LEER ELEMENTO
  - 2. SI ES OPERANDO METERLO EN UNA PILA DE NUM. REALES
  - 3. SI ES OPERADOR ENTONCES
  - 3.1 SACAR LOS 2 ELEMENTOS SUPERIORES DE LA PILA
  - 3.2 EVALUAR LA OPERACIÓN
  - 3.3 METER EL RESULTADO EN LA PILA
- Resultado = Cima(pila)

Pilas. Uso (II)

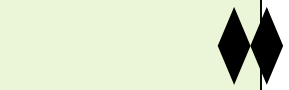
```
UNIT PilaReal;
INTERFACE
  TYPE
  TipoElemento = Real;
  TipoPila = ^TipoNodo;

  TipoNodo=RECORD
   info:TipoElemento;
   ant:TipoPila
  END;
```

```
UNIT PilaCar;
INTERFACE
   TYPE
   TipoElemento = Char;
   TipoPila = ^TipoNodo;

   TipoNodo = RECORD
    info:TipoElemento;
    ant:TipoPila
   END;
```

- Otra posibilidad es introducir el TipoElemento en una unidad diferente

Pilas. Uso (II)

```
PROGRAM Evaluar;

USES PilaCar, PilaReal;


FUNCTION PrioridadDentro(operador:char):integer;

BEGIN

 ....

END;


FUNCTION PrioridadFuera(operador:char):integer;

.....

FUNCTION LeerNumero(cadena:String; VAR pos:Integer):String;

......

PROCEDURE InfijaPostFija(cadInf:String; VAR cadPost:String);

....
```



Pilas. Uso (II)

```
FUNCTION LeerNumeroReal(cadena: String; VAR pos:integer): real;
VAR
  valorAux: real;
  divisor: integer;
BEGIN
  {PARTE ENTERA}
  valorAux := 0.0;
  WHILE (pos<=length(cadena)) AND ( cadena[pos] IN ['0'..'9'] ) DO
  BEGIN
     valorAux := valorAux *10 + ord(cadena[pos]) - ord('0');
     Inc(pos);
  END;
  {FIN PARTE ENTERA}
```

Pilas. Uso (II)

```
 IF pos <= length(cadena) THEN
   IF cadena[pos]='.' THEN
  BEGIN
      Inc(pos);
      divisor:=10;
      WHILE (pos<=length(cadena)) AND (cadena[pos] IN ['0'..'9'])DO
      BEGIN
        valorAux := valorAux + (ord(cadena[pos])-ord('0'))/divisor;
        divisor := divisor * 10;
        Inc(pos)
       END;
   END;
   LeerNumeroReal := valorAux
END;
```

Pilas. Uso (II)

```
FUNCTION EvaluarPostfija(sPost: String) : Real;
VAR
 pilaR: PilaReal.TipoPila;
 op1, op2, valorTmp: real;
 posicion: integer;
BEGIN
 PilaReal.CrearPilaVacia(pilaR);
 posicion := 1;
       WHILE (posicion <= length(sPost)) DO
        CASE sPost[posicion] OF
        '0'..'9':
         BEGIN
           op1:= LeerNumeroReal(sPost, posicion);
           PilaReal.Apilar(op1, pilaR)
         END;
```

Pilas. Uso (II)

```
'+','-','*','/','^':
        BEGIN
           IF PilaReal.EsPilaVacia(pilaR) THEN
            BEGIN
               EvaluarPostFija := 0.0;
               Writeln('EXPRESION INCORRECTA');
               posicion := length(sPost) + 1;
            END
           ELSE
            BEGIN
             PilaReal.Cima(op2, pilaR);
             PilaReal.Desapilar(pilaR);
             IF PilaReal.EsPilaVacia(pilaR) THEN
              BEGIN
```

Pilas. Uso (II)

```
        EvaluarPostFija := 0.0;
        writeln('EXPRESION INCORRECTA');
        posicion := length(sPost) + 1
      END
    ELSE
      BEGIN
        PilaReal.Cima(op1, pilaR);
        PilaReal.Desapilar(pilaR);
        CASE sPost[posicion] OF
          '^':valorTmp := Exp(op2* Ln(op1));
          '*':valorTmp := op1*op2;
          '/':valorTmp := op1/op2;
          '+':valorTmp := op1+op2;
          '-':valorTmp := op1-op2;
        END;
```

Pilas. Uso (II)

```
                PilaReal.Apilar(valorTmp, pilaR);
                Inc(posicion)
               END
              END
          END { '+','-','*','/','^' }
        ELSE
          Inc(posicion);
       END; { CASE }


  { Si pila sólo tiene un elemento ese es el resultado }
  PilaReal.Cima(valorTmp, pilaR);
  PilaReal.Destruir(pilaR);
  EvaluarPostFija := valorTmp
END;
```

Pilas. Uso (III)

```
FUNCTION Balanceada(expr: String): Boolean;
VAR
  c: char;
  p: TipoPila;
  posicion: Integer;
BEGIN
   CrearPilaVacia(p);
   posicion := 1;
   WHILE (posicion <= length(expr)) DO
   BEGIN
      IF expr[posicion]='(' THEN
  Apilar(expr[posicion], p)
      ELSE IF expr[posicion]=')' THEN
       Desapilar(p);
      Inc(posicion);
   END;
   Balanceada:= EsPilaVacia(p)
END;
```

Comprobación de paréntesis balanceados