Expressions arithmétiques

Évaluation d'une expression post-fixée (EPF)

In : EPF ; *Out :* valeur de l'expression.
Conventions : l'expression est terminée par `$`. On dispose de fonctions `chiffre` et `valchiffre`.
On suppose qu'il n'y a pas d'erreur de syntaxe.

```
var
  x : char ;   p : pile_entier ;   val, op1, op2 : integer ;   fin : boolean ;
begin
  init_pile(p) ;
  fin := FALSE ;
  repeat
    read(x) ;
    if chiffre(x) then
       empiler(p,valchiffre(x))
    else begin
      op2 := depiler(p) ;
      op1 := depiler(p) ;
      case x of
        '+' : val := op1 + op2 ;
        '-' : val := op1 - op2 ;
        '*' : val := op1 * op2 ;
        '/' : val := op1 / op2 ;
        '$' : fin := TRUE ;
      end ;
      empiler(p, val) ;
   until fin ;
   writeln('Résultat : ', dépiler(p)) ;
end.
```

Traduction d'une EAIP en une EPF

In : EAIP (expr. arith. incomplètement parenthésée) ; *Out :* EPF.
Exemples : *In :* `( 3 - 5 ) * ( 8 + 2 ) $` *Out :* `3 5 - 8 2 + *`.
In : `( 3 - 5 ) *  8 + 2  $` *Out :* `3 5 - 8 * 2 +`.

```
begin
repeat
  read(x) ;
  case x of
    '('      : empiler(P, '(') ;
    ')'      : while sommet(P) <> '(' do {y := depiler(P) ; write(y) ;}
               depiler(P) (* on dépile '(' *)
    opérande : write(x) ;
    opérateur: while (sommet(P) est un opérateur plus ou aussi
                      prioritaire que x) do
                  {y := depiler(P) ; write(y) ;}
               empiler(P, x) ;
until x = '$' ;
while not vide(P) do {y := depiler(P) ; write(y) ;}
end.
```