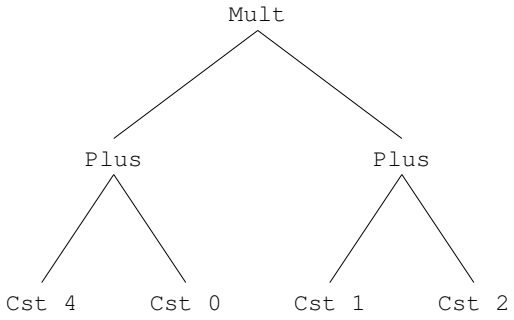
TP5 : *Arbres et expressions arithmétiques*

Un arbre arithmétique permet de représenter des expressions arithmétiques. Il est constitué soit de feuilles qui sont des entiers, soit d'un noeud qui contient un opérateur arithmétique ($+$, $-$, $*$ ou $/$), un sous arbre gauche et un sous arbre droite. L'expression $((4 + 0) * (1 + 2))$ peut ainsi se représenter de la manière suivante :



1 Arbres binaires

On définit le type OBJECTIVE CAML suivant pour représenter les expressions arithmétiques :

```
type op = Plus | Minus | Mult | Div
type expr =
  | Cst of int
  | Op of op * expr * expr
```

Questions préliminaires

1. Définir une valeur OBJECTIVE CAML de type `expr` s'appelant `e1` qui représente l'expression arithmétique $((4 + 0) * (1 + 2))$.
2. Écrire une fonction `left : expr -> int` qui retourne la valeur de la feuille qui est la plus à gauche dans l'arbre. L'exécution de `left e1` doit retourner la valeur `4`.

Évaluation

Pour calculer la valeur entière associée à une expression arithmétique, une méthode consiste à : (1) calculer la valeur associée à la sous-expression de gauche, (2) calculer la valeur associée à la sous-expression de droite et (3) appliquer la « bonne opération » entre ces deux valeurs.

3. Écrire une fonction `eval : expr -> int` qui calcule la valeur associée à une expression.

Affichage

Nous voulons être capable d'afficher des expressions arithmétiques dans le terminal. Pour cela, vous pourrez utiliser les fonctions suivantes :

- `print_string : string -> unit`
- `print_int : int -> unit`

4. Écrire la fonction `print_op : op -> unit` qui affiche un opérateur.
5. Écrire une fonction `print_expr : expr -> unit` qui affiche une expression en notation infixe entièrement parenthésé. Par exemple, `e1` sera affichée `((4 + 0) * (1 + 2))`. Cette fonction doit être définie par cas sur la structure des expressions :
   - si l'expression est une constante, il faut afficher la valeur de la constante :
   - si l'expression est un noeud, il faut afficher une parenthèse ouvrante, afficher la sous-expression gauche, afficher l'opérateur, afficher la sous-expression droite et enfin afficher la parenthèse fermante.

Construction

Nous allons maintenant définir une fonction `build_right : op -> int list -> expr` qui permet de construire une expression à partir d'un opérateur arithmétique et d'une liste d'entiers. Cette expression représentera l'application de droite à gauche de l'opérateur donné en paramètre sur les entiers de la liste. Par exemple, `build_right Minus [1; 2; 3; 4]` retournera une valeur OBJECTIVE CAML :

```
Op (Minus, Cst 1, Op (Minus, Cst 2, Op (Minus, Cst 3, Cst 4)))
```

correspondant à l'expression $1 - (2 - (3 - 4))$.

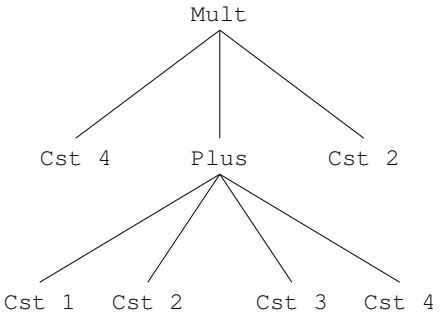
6. Écrire la fonction `build_right : op -> int list -> expr`. On supposera que la liste contient toujours au moins un élément.
7. En utilisant la fonction `List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a`, écrire une fonction `build_left : op -> int list -> expr` qui a un comportement similaire à la fonction `build_right` mise à par que les applications de l'opérateur doivent de faire de droite à gauche. Ainsi, `build_left Minus [ 1; 2; 3; 4 ]` retourne la valeur

   ```
   Op (Minus, Op (Minus, Op (Minus, Cst 1, Cst 2), Cst 3), Cst 4)
   ```

   correspondant à l'expression $((1 - 2) - 3) - 4$.

2 Arbres n-aires

Nous changeons maintenant la représentation des arbres. Nous ne voulons plus nous limiter à des arbres qui ont deux fils, mais nous voulons des arbres qui peuvent avoir un nombre quelconque de fils. Ainsi, l'expression $3 * (1 + 2 + 3 + 4) * 5$ pourra être représentée par l'arbre suivant :



On définit le type OBJECTIVE CAML suivant pour représenter ces nouveaux arbres :

```
type nexpr =
  | N_Cst of int
  | N_Op of op * nexpr list
```

Dans cette représentation, on supposera que
- la liste associée à un constructeur `N_Op` sera toujours non vide ;
- la soustraction est associative à gauche, c'est-à-dire que la valeur `N_Op (Minus, [N_Cst 4; N_Cst 3; N_Cst 2])` correspond à l'expression $(4 - 3) - 2$;
- la division est associative à droite, c'est-à-dire que la valeur `N_Op (Div, [N_Cst 4; N_Cst 3; N_Cst 2])` correspond à l'expression $4/(3/2)$.

8. Définir une valeur `ne1 : nexpr` correspondant à l'expression $3 * (1 + 2 + 3 + 4) * 5$.
9. Écrire la fonction `nleft : nexpr -> int` correspondant à la fonction `left`. Le résultat de `nleft ne1` doit être la valeur `3`.
10. Écrire la fonction `neval : nexpr -> int` correspondant à la fonction `eval`. Le résultat de `neval ne1` doit être la valeur `150`.
11. Écrire les fonctions `nexpr_of_expr: expr -> nexpr` et `expr_of_nexpr` qui permettent de passer d'une représentation à une autre.