U.E. S2IN423 - CCTP

Représentation d'expressions algébriques – Dérivation

Instructions :

Vous pouvez faire ce devoir individuellement ou à deux.

Le devoir est à rendre sous la forme d'un fichier de type texte qui contiendra votre programme (type, fonctions, expressions, exemples ...)

Ce fichier devra impérativement contenir sur les premières lignes et sous formes de commentaires les informations suivantes : nom, prénom, numéro étudiant du ou des deux personnes ayant fait le devoir.

Ce devoir est à rendre exclusivement par dépôt sur la page Moodle de l'UE Programmation fonctionnelle avant le vendredi 1er avril à 24h00

Avant de faire ce devoir je vous conseille vivement de revoir la dernière section du cours : « Application des types récursifs : Expressions arithmétiques - Représentation, évaluation, écriture. »

Représentation d'expressions algébriques

On définit le type suivant permettant de représenter des expressions algébriques formées à partir de constantes, de la variable x, des opérations binaires de base et de quelques fonctions usuelles.

```
(* Un type somme pour représenter les opérations  (i.e les opérateurs binaires) *)
type top = Plus| Moins | Fois | Div;;

(* Le type somme pour représenter les fonctions usuelles *)
type tfonc = Sin | Cos | Ln | Exp | Rac | Opp;;

(* Un type récursif pour représenter des expressions algébriques *)
type texpr =
  | Const of float
  | X
  | Op of texpr * top * texpr
  | F of tfonc * texpr
  | P of texpr * int;;
```

Significations des constructeurs :

`Const` : un expression constante, c'est à dire un nombre
`X` : la variable x
`Op` : pour construire une expression à partir d'un opérateur binaire
`F` : pour construire une expression à partir d'une fonction
`P` : pour construire une expression avec une puissance entière

Une expression algébrique est donc construite :
- soit avec le constructeur `Const` et un flottant pour représenter une constante
- soit avec le constructeur `X` pour représenter la variable x
- soit avec le constructeur `Op`, un opérateur binaire et de deux expressions algébriques
- soit avec le constructeur `F`, une fonction et une expression algébrique
- soit avec le constructeur `P`, une expression algébrique et un entier pour la puissance

Exemples d'expressions :

$3x + 25$ est représentée par : `Op(Op(Const(3.0), Fois, X), Plus, Const(25.0))`

$\sqrt{x^2+3}$ est représentée par : `F(Rac, Op(P(X, 2), Plus, Const(3.0)))`

Exercice 0
Recopiez dans votre fichier source les définitions des types donné plus haut.
Et n'oubliez pas d'indiquer en commentaire au début de votre fichier : numéros étudiants, noms et prénoms !

Exercice 1
Déclarez dans votre fichier les expressions de type `texpr` qui représentent les expressions algébriquers suivantes :

$$7x^3 - 4x + 8 \qquad \frac{1+\sin(3x)}{1-\cos(x)}$$

$$1 - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} \qquad \ln(1 + \sqrt{x^2+3})$$

Exercice 2 — Conversion en écriture infixée totalement parenthésée
Écrivez une fonction `stringinf_of_texpr` de type `texpr -> string` qui prend en paramètre une `texpr` et retourne une chaîne de caractères correspondant à l'écriture infixée totalement parenthésée de l'expression.

Exemple : en reprenant les expressions définie plus haut :

```
# let e1 = Op( Op(Const(3.0), Fois, X), Plus, Const(25.0)) ;;
val e1 : texpr = Op (Op (Const 3., Fois, X), Plus, Const 25.)

# let e2 = F(Rac, Op( P(X, 2), Plus, Const(3.0)));;
val e2 : texpr = F (Rac, Op (P (X, 2), Plus, Const 3.))

# stringinf_of_texpr e1;;
- : string = "((3. * x) + 25.)"

# stringinf_of_texpr e2;;
- : string = "√((x^2) + 3.))"
```

Rq : si votre environnement ne vous permet pas d'avoir le caractère unicode √ vous pouvez simplement représentez la racine carrée par la chaîne `sqrt`.

Exercice 3 — Conversion en écriture préfixée
Écrivez une fonction `stringpref_of_texpr` de type `texpr -> string` qui prend en paramètre une `texpr` et retourne une chaîne de caractères correspondant à l'écriture préfixée de l'expression.

Exemple : toujours pour les mêmes expressions les chaînes obtenues en écriture infixée sont :

```
# stringpref_of_texpr e1;;
- : string = "+ * 3. x 25."

# stringpref_of_texpr e2;;
- : string = "√ + ^ x 2 3."
```

Exercice 4 — Évaluation

Écrivez une fonction récursive `evaltexpr` de type `texpr -> float -> float` qui prend en paramètres une `texpr` et un flottant et qui retourne comme résultat la valeur de l'expression algébrique correspondante en donnant comme valeur à la variable x le flottant donné en paramètre.

Exemple : toujours pour les mêmes expressions :

```
# evaltexpr e1 7.2;;
- : float = 46.6

# evaltexpr e2 2.0;;
- : float = 2.64575131106459072

# evaltexpr e2 1.0;;
- : float = 2.
```

Exercice 5 — Dérivation formelle

On rappelle ci-dessous les règles de dérivation usuelles où c est une constante, n un entier, u et v des fonctions et x la variable de dérivation :

$$\begin{array}{rcl}
(c)' & = & 0 \\
(x)' & = & 1 \\
(u+v)' & = & u'+v' \\
(u-v)' & = & u'-v' \\
(u\times v)' & = & u'\times v+u\times v' \\
(\frac{u}{v})' & = & \frac{u'\times v-u\times v'}{v^2} \\
(\sin(u))' & = & u'\times\cos(u) \\
(\cos(u))' & = & -u'\times\sin(u) \\
(\ln(u))' & = & \frac{u'}{u} \\
(e^u)' & = & u'\times e^u \\
(u^n)' & = & n\times u'\times u^{n-1} \\
(\sqrt{u})' & = & \frac{u'}{2\times\sqrt{u}} \\
(-u)' & = & -u'
\end{array}$$

Écrivez une fonction récursive `derive` de type `texpr -> texpr` qui prend en paramètre une expression algébrique de type `texpr` et renvoie sa dérivée par rapport à la variable x sous forme d'une expression algébrique de type `texpr`.
Cette fonction doit implanter les règles de dérivations ci-dessus.
Elle doit construire la `texpr` de la dérivée en appliquant strictement les règles de dérivation. Elle ne doit en aucun cas effectuer de simplification.

Ces règles de dérivations sont par nature récursive : la dérivée d'une expression est construite à partir des dérivées des sous-expressions qui la composent. Par exemple, pour dériver une expression qui est construite comme somme de deux expressions u et v il faut récursivement dériver les deux expressions u et v et construire l'expression résultat comme somme des deux expressions dérivées.

Exemple : toujours pour les mêmes expressions vous devriez obtenir comme résultat de la dérivation les expressions suivantes :

```
# derive e1;;
- : texpr = Op (Op (Op (Const 3., Fois, Const 1.), Plus, Op (Const 0., Fois, X)), Plus, Const 0.)

# derive e2;;
- : texpr = Op (Op (Op (Const 2., Fois, Op (Const 1., Fois, P (X, 1))), Plus, Const 0.),
 Div, Op (Const 2., Fois, F (Rac, Op (P (X, 2), Plus, Const 3.))))
```

Exercice 6 — Simplification

L'application stricte et automatique des règles de dérivation conduit à une expression dérivée non simplifiée. La simplification des `texpr` pourrait donner lieu à un projet complet, cependant avec quelques règles de base on peut déjà faire des simplifications intéressantes.

On donne ci-dessous quelques règles de simplification (e désigne une `texpr`, $n1$, $n2$, $n3$ des constantes et x est la variable de dérivation. On peut en écrire bien d'autres.

Écrivez une fonction récursive `simplif` de type `texpr -> texpr` qui prend en paramètre une `texpr` et qui construit une expression simplifiée en appliquant, entre autre, les règles données ci-dessous.
Vous pouvez bien entendu compléter cette fonction avec d'autres règles.

La première règle se traduit pas : « une `texpr` de la forme `Op(Const(0.0), Plus, e)` se simplifie en `e` »
La troisième règle se traduit pas : « une `texpr` de la forme `Op(Const(n1), Plus, Const(n2))` se simplifie en `Const(n3)` avec $n_3 = n_1 + .n_2$

$0 + e$	$\longrightarrow$	e
$e + 0$	$\longrightarrow$	e
$n1 + n2$	$\longrightarrow$	$n3$ où $n3$ est le résultat de la somme des nombres $n1$ et $n2$
$e - 0$	$\longrightarrow$	e
$n1 - n2$	$\longrightarrow$	$n3$ où $n3$ est le résultat de la différence des nombres $n1$ et $n2$
$x - x$	$\longrightarrow$	0
$0 \times e$	$\longrightarrow$	0
$e \times 0$	$\longrightarrow$	0
$e \times 1$	$\longrightarrow$	e
$1 \times e$	$\longrightarrow$	e
$n_1 \times n_2$	$\longrightarrow$	$n3$ où $n3$ est le résultat du produit des nombres $n1$ et $n2$
$\frac{n1}{n2}$	$\longrightarrow$	$n3$ où $n3$ est le résultat du quotient des nombres $n1$ et $n2$
$\frac{e}{1}$	$\longrightarrow$	e
	etc ...	