Travaux Pratiques de C

Cours de langage et programmation

—L3 Informatique—

On définira en *variable globale* un "tableau de fonctions" prédéfinies, `TabFonc`. Ce tableau sera de taille N et contiendra des fonctions de bases.

`F_ID` : $x \longmapsto x$	`F_COS` : $x \longmapsto \cos(x)$
`F_SIN` : $x \longmapsto \sin(x)$	`F_EXP` : $x \longmapsto e^x$
`F_SQRT` : $x \longmapsto \sqrt{x}$	`F_LN` : $x \longmapsto \ln(x)$

On dispose en outre d'opérateurs binaires, agissant sur les fonctions :

$'+'$	$f+g : x$	$\longmapsto f(x)+g(x)$	addition
$'-'$	$f-g : x$	$\longmapsto f(x)-g(x)$	soustraction
$'*'$	$f*g : x$	$\longmapsto f(x)*g(x)$	multiplication
$'/'$	$f/g : x$	$\longmapsto f(x)/g(x)$	division
$'$`o`$'$	$f \circ g : x$	$\longmapsto f(g(x))$	composition

On dispose aussi des constantes réelles.

A partir des fonctions de base de `TabFonc` et des opérateurs ci-dessus, on peut construire des fonctions plus complexes par combinaison.
Par exemple, la fonction $F : x \longmapsto (2.5 + sin(x)) * e^{cos(x)+sin(x)}$, s'écrit sous la forme `(2.5+F_SIN)*(F_EXP o(F_COS+F_SIN))`.

▸ Exercice 1. Fonctions

i) Définir un type `p_fct` *représentant un pointeur sur fonction prenant en paramètre un réel (*`double`*) et renvoyant un réel.*

ii) Définir une énumération `usual_fct` *dont les valeurs sont* `F_ID`, `F_COS`, *etc.*

iii) Construire le tableau `TabFonc` *de* `p_fct` *de sorte que, par exemple,* `TabFonc[F_SIN]` *soit un pointeur sur une fonction qui calcule le sinus. On redéfinira les fonctions de base '*`_cos`*', *`_inv`*, etc., de manière à sécuriser le programme en cas d'erreur (par exemple, si on veut calculer* $\sqrt{x}$ *avec* $x = -1, 0$*, le programme devra afficher un message d'erreur du type* `ERREUR : La fonction <sqrt> n'est pas définie au point -1.0`*, et répercuter cette erreur aux fonctions de niveau supérieur, celles devant utiliser la valeur de* $\sqrt{x}$ *calculée).*

▸ Exercice 2. Arbres de fonctions

On souhaite représenter ces fonctions sous forme d'arbres binaires, les nœuds internes étant des opérateurs binaires et les feuilles étant soit des fonctions, soit des constantes.

i) Définir une énumération `node_type` *dont les valeurs sont* `BIN_OP`, `FCT` *et* `CST`.

ii) Définir le type `Function`*, de sorte que si* `f` *est une variable de type* `Function`*,* `*f` *dispose des champs suivants :*
– un champ `type` *qui selon sa valeur* `BIN_OP`, `FCT`, `CST`*, indique si le nœud représente un opérateur, une fonction ou une constante ;*
– un champ `op` *utilisé uniquement pour les nœuds de type* `BIN_OP` *et qui selon sa valeur*

('+','','-','o') indique la nature de l'opérateur ;*
– un champ `fct` *utilisé uniquement pour les nœuds de type* `FCT` *et qui caractérise la fonction ;*
– un champ `valCst` *utilisé uniquement pour les nœuds de type* `CST` *et qui indique la valeur (réelle) de la constante.*
– deux champs `f` *et* `g` *utilisé uniquement pour les nœuds de type* `BIN_OP` *et qui stocke les sous-fonctions auxquelles l'opérateur s'applique.*

Écrire ce type de sorte que les champs `op`, `fct`, `valCst` *utilisent le même emplacement en mémoire.*

i) Construire l'arbre binaire correspondant à la fonction F précédente.

ii) Écrire une fonction `Function newFunction(enum usual_fct f)` *qui crée un nouveau noeud et initialise ses champs* `type` *et* `fct`. *Écrire de même une fonction* `Function newBinOp(char op, Function f, Function g)` *et une fonction* `Function newCst(double c)`.

iii) Écrire une fonction `void libere(Function fct)` *qui libère l'espace alloué à la fonction* `fct`.

iv) Écrire une fonction `double evalue(Function fct, double x)` *qui calcule la valeur de* `fct` *au point* `x`. *A quel type de parcours d'arbre celà correspond-il (largeur, profondeur, préfixe, suffixe, infixe) ?*

▸ **Exercice 3. Dérivée de fonction**

i) Construire l'arbre binaire correspondant à la dérivée ΔF de la fonction F précédente.

ii) Ecrire une fonction `Function clone(Function fct)` *qui construit une copue de l'arbre représentant la fonction* `fct`.

iii) Ecrire une fonction `Function derive(Function fct)` *qui construit l'arbre de la fonction dérivée de* `fct`.

▸ **Exercice 4. Lecture et affichage en ligne** *On veut pouvoir créer une fonction sur la ligne de commande et faire en sorte que le programme construise l'arbre correspondant. La lecture se fera en notation préfixée et l'affichage en notation standard. Par exemple pour créer la fonction*

$$F : x \longmapsto e^{\sin(x)}(\cos(x) + \sin(e^x))$$

on écrira sur la ligne de commande :

```
exp sin o cos sin exp o + *
```

et le programme nous affichera la fonction construite sous la forme :

```
F(x) : exp(sin(x))*(cos(x)+sin(exp(x)))
```

On utilisera, sans l'écrire, une structure de pile générique, c'est-à-dire dont les éléments sont définis comme des pointeurs sur void.

i) Écrire une fonction `int GetFunction(char* ligne, Pile P)` *permettant de lire récursivement la ligne de commande et stockant la fonction extraite sur la pile.*

ii) Écrire une fonction `void Writefunction(Function F)` *permettant d'écrire la fonction comme défini ci-dessus. On utilisera un tableau de noms de fonctions parallèle au tableau des fonctions.*