Préparation aux oraux Centrale/Mines-Ponts n°2

Calculatrice

L'objectif de ce TP est de réaliser une calculatrice. On s'intéressera à plusieurs aspects de cette réalisation : la manipulation de polynômes, puis la gestion d'expressions mathématiques, et enfin la représentation graphique de fonctions.

Les questions sont annotées de différents pictogrammes pour signifier ce qui est attendu de l'élève :
- ✎ signifie une question à rédiger sur le compte-rendu (il sera rendu en même temps que le code) ;
- 🗣 signifie une question (à préparer et) à présenter à l'examinateur ;
- 💻 signifie une question où du code devra être écrit.

L'évaluation des compétences de l'étudiant se basera principalement sur les critères suivants :
- la qualité et la clarté du code ;
- la qualité et la clarté du compte-rendu ;
- la qualité des interactions avec l'examinateur (en particulier sur les connaissances du cours) ;
- l'avancement dans le sujet.

I. Polynômes en SQL.

Dans le fichier base de données `polynomes.db`, vous trouverez une seule table `monome` contenant trois colonnes : `facteur`, `exposant` et `poly`. L'utilisation de l'interpréteur SQLite3 est détaillé dans l'Annexe A. Afin de représenter un polynômes P, on le décompose comme somme de monômes $a \times X^k$. Chaque ligne de la table contient un monôme, où a est la valeur de `facteur`, et k est la valeur de `exposant`. La valeur de `poly`, un entier, permet de déterminer quel polynôme P est représenté, avec un identifiant commun.

✎ **Q1.** Quelle est la représentation des polynômes X et $X^2 - 1$ dans la table ?

💻 **Q2.** Lister les polynômes présents dans la table `monome`.

Remarquons que la représentation décrite ci-dessus n'est pas unique : nous pouvons représenter $2X$ comme $2 \times X$ (un unique monôme) ou comme $X + X$ (somme de deux monômes). On dira que la représentation d'un polynôme est *acceptable* s'il n'y a pas plusieurs monômes de même exposant. Cette représentation acceptable est unique.

💻 🗣 **Q3.** Écrire une requête SQL permettant de trouver les polynômes dont la représentation n'est pas acceptable. Proposer une modification de la structure de données imposant que la représentation d'un polynôme soit acceptable.

💻 **Q4.** Écrire une requête SQL permettant de calculer $P + Q$ où P et Q sont deux polynômes représenté dans la table `monome`. La requête retournera deux colonnes `facteur` et `exposant` représentant le polynôme $P + Q$ de manière acceptable.

💻 **Q5.** Comme pour Q4, écrire une requête SQL permettant de calculer $P \times Q$ où P et Q sont deux polynômes représentés dans la table `monome`.

II. Expressions mathématiques en OCaml.

Dans cette partie, on cherche à effectuer des opérations sur des expressions mathématiques (évaluation, simplification, dérivation), puis à convertir une chaîne de caractères en une expression (et inversement). On définit le type `expression` comme dans Code 1.

```
type expression =
| Var of char
| Number of int
| Plus of expression * expression
| Mult of expression * expression
| Neg of expression
| Inv of expression
```

Code 1. Représentation en OCaml d'une expression mathématique.

II.1. Représentation d'une expression.

Q6. Discuter de comment représenter une expression mathématique à l'aide d'une expression de type `expression`. On s'intéressera particulièrement à comment représenter une *différence* ou un *quotient*. Représenter les expressions ci-dessous en OCaml :

$$\frac{x^3+x}{x^2}, \qquad 2x-1, \qquad \text{et} \qquad x^2+y^2-1.$$

Q7. Étant donnée une variable, calculer la dérivée d'une expression par rapport à cette variable. On ne simplifiera pas les expressions retournées, c'est l'objet de Q9.

Q8. On souhaite évaluer une expression. On souhaite donc assigner à une variable un flottant (sa valeur). Quelle représentation utiliser pour assigner à chaque variable leur valeur ? Cette représentation n'a pas besoin d'être efficace : le nombre de variables est faible, et en général moins de 2. Écrire une fonction d'évaluation d'une expression.

Q9. On souhaite simplifier une expression, comme celles obtenues à la suite de Q7. Par exemple, on peut simplifier la multiplication par zéro, par un, l'addition d'un zéro, *etc.* Proposer des règles de simplification d'une expression et les implémenter.

II.2. Transformations entre chaînes de caractères et expressions.

Dans cette sous-section, on s'intéresse à la transformation entre chaîne de caractère et expression.

On rappelle qu'une expression *bien parenthésée* est une séquence de symboles mathématiques où chaque ouverture de parenthèse est correctement appariée avec une fermeture correspondante. En d'autres termes, chaque parenthèse ouvrante doit être suivie d'une parenthèse fermante dans un ordre correct, et chaque paire de parenthèses doit contenir une expression valide entre elles.

Q10. Donner une grammaire non contextuelle $\mathcal{G}$ reconnaissant les expressions mathématiques bien parenthésées.

Q11. Implémenter, en OCaml, une fonction `vers_chaine : expression -> string` transformant une expression en chaîne de caractère bien parenthésée la représentant.

Q12. Écrire une fonction `decoupe : string -> char list` qui transforme une chaîne de caractère en la liste des caractères qui la compose.

Q13. Écrire une fonction `parse_numbers : char list -> expression * (char list)` qui, sur une entrée s, retourne le couple (e, r) où $s = e \cdot r$ et e est l'expression représentant le plus long préfixe non-vide de s ne contenant que des entiers de 0 à 9 (avec le variant `Number`, *c.f.* Code 1).

Q14. Expliquer brièvement pourquoi une expression bien parenthésée a, au plus, un préfixe bien parenthésé.

Q15. Écrire une fonction `parse_expr : char list -> expression * (char list)` qui, sur une entrée s, retourne le couple (e, r) où $s = e \cdot r$ et e est l'unique préfixe de l'expression représentée dans la chaîne de caractères en entrée.

Q16. Écrire une fonction `vers_expression : string -> expression` transformant la chaîne de caractères en l'expression quelle représente.

III. Calculatrice graphique en OCaml.

L'objectif de cette partie est la représentation de courbes sur une fenêtre. Pour cela, nous utiliserons OCaml ainsi que le module `Graphics`. Les commandes permettant d'inclure le module `Graphics` lors de la compilation/exécution du code OCaml sont données en Annexe B. Pour afficher une fenêtre, on utilise le code ci-dessous.

```
open Graphics;

let _ = open_graph ""
```

Dans les fichiers associés à ce TP, vous trouverez la documentation du module `Graphics`. Elle pourra vous être utile dans la suite de ce TP.

Q17. Remplir la fenêtre en jaune en utilisant *uniquement* les fonctions `open_graph`, `size_x`, `size_y` et `plot`. Vous pourrez regarder la documentation de ces fonctions. On procédera, par exemple, en utilisant deux boucles `for`.

On suppose que l'on veuille « colorier » un point de la fenêtre ouverte précédemment aux coordonnées $(x, y) \in [x_1, x_2] \times [y_1, y_2]$. L'écran sera donc représenté par ses « limites » : x_1–x_2 pour l'axe x, et y_1–y_2 pour l'axe y. Pour avoir les coordonnées du pixel à modifier, on utilisera la formule :

$$\text{pix}(x, y) = \left(\texttt{size\_x()} \times \left\lfloor \frac{x - x_1}{x_2 - x_1} \right\rfloor, \texttt{size\_y()} \times \left\lfloor \frac{y - y_1}{y_2 - y_1} \right\rfloor \right)$$

Q18. On souhaite afficher le graphe d'une fonction représentée par son expression mathématique (de type `expression`). On supposera que la dérivée de cette fonction n'est pas trop importante sur la fenêtre considérée. Écrire, en OCaml, une fonction dont la signature est `dessine : float -> float -> float -> float -> expression -> unit` qui, lorsqu'elle est appelée de la forme `dessine x1 x2 y1 y2 e` trace la courbe de l'expression `e` sur la fenêtre avec les limites définies par les paramètres `x1`, `x2`, `y1`, `y2`.

Q19. On souhaite à présent afficher une courbe paramétrée. Écrire une fonction `dessine_param` dont les arguments sont, dans l'ordre `t1`, `t2`, `dt`, `x1`, `x2`, `y1`, `y2`, `ex`, et `ey`. Les arguments `x1`, `x2`, `y1` et `y2` définissent les limites de la fenêtre. Les arguments `t1` et `t2` définissent les limites de la courbe paramétrée, et `dt` sa résolution (vue comme le pas d'incrémentation). Les expressions `ex` et `ey` sont pour représenter les deux fonctions $x(t)$ et $y(t)$.

Par la suite, on ne donne pas les arguments des fonctions à coder. En effet, ils sont généralement les mêmes.

Q20. Écrire une fonction qui dessinera les axes x et y passant par l'origine $(0, 0)$. On vérifiera que chacun des axes soit dans les limites de l'écran.

Q21. Écrire une fonction `dessine_tang` qui dessine la droite tangente à une fonction en un point x. On pourra utiliser la fonction codée en Q7.

Annexe A. Utilisation de SQLite3.

Pour cet exercice, on vous fournit un fichier `polynomes.db` qui stocke une base de données SQLite3. Pour pouvoir exécuter des commandes sur ce fichier on peut lancer la commande `sqlite3 polynomes.db` depuis le dossier où se trouve ce fichier. Cette commande lance un interpréteur SQLite3. On peut par exemple taper la commande suivante pour avoir le nombre d'enregistrements dans la table `monome` :

```
SELECT COUNT(*) FROM monome ;
```

Pour avoir un affichage plus lisible avec SQLite3, on pourra utiliser les deux commandes suivantes (à copier directement dans le shell SQLite3) :

```
.header on
.mode column
```

Annexe B. Compilation OCaml avec `Graphics`.

Pour charger ce module, on pourra utiliser la commande

```
ocaml -I /home/candidat/.opam/default/lib/graphics graphics.cma
```

pour exécuter, et écrire la ligne `#load "graphics.cma";;` (ou `open Graphics;;`) dans un fichier d'extension `.ml`.

REMARQUE. La commande sera à adapter en fonction du nom d'utilisateur. On remplacera, dans la commande, le « `candidat` » par le nom d'utilisateur de la session Linux actuelle.