SdV-1

TD d'Informatique No. 1

1 Éléments constitutifs d'un ordinateur

Le cœur d'un ordinateur est ce que l'on nomme une Unité Arithmétique et Logique (ALU en bon anglais...). Une ALU est une sorte de « centrale de manipulations » qui effectue des transformations électriques rapides, correspondant à des manipulations des *symboles* codés par ces signaux électriques. Elle marche de concert avec une mémoire qui alimente l'ALU en « données » c'est-à-dire, encore, des suites de signaux électriques.

Le codage mentionné plus haut repose techniquement principalement sur trois représentations concrètes : les valeurs logiques (vrai/faux), les valeurs numériques (i.e. les nombres), et les commandes c'est-à-dire les manipulations à effectuer sur les codages des données précédentes. L'élément de base de ces codages est l'information « binaire », *le courant peut passer* ou *ne peut pas passer*, et est matériellement réalisé par un genre de transistor largement miniaturisé.

Les commandes successives à effectuer sont placées en des endroits déterminés de la mémoire et sont lues les unes après les autres, ce qui conduit à enchaîner des opérations et finalement à faire des calculs complexes. Le premier usage de la mémoire est donc de mémoriser le programme des opérations à effectuer. Pour ce faire l'ALU possède une zone dans laquelle elle code la place de la mémoire où se trouve la prochaine opération à effectuer, une zone dans laquelle elle a recopié l'opération en cours, et quelques zones qui servent à manipuler les données traitées. Ces zones sont appelées des registres.

Maintenant que l'on a vu comment transitent les commandes successives à effectuer dans l'ALU, parlons des données : il ne servirait à rien de faire tant de manipulations si elles ne sortaient jamais de l'ALU. Pour cela il y a des commandes de stockage des registres de l'ALU vers la mémoire. De même, pour alimenter l'ALU en données, il existe des commandes de chargement des données (copies de la mémoire vers les registres de l'ALU).

Ainsi, on voit que si l'on conçoit un programme de commandes judicieux, on peut modifier à volonté l'état de la mémoire pour lui faire stocker des résultats de calculs aussi complexes et sophistiqués que l'on veut. La question qui se pose maintenant est d'exploiter cette mémoire en la montrant à l'extérieur sous une forme compréhensible pour l'Homme, ou réexploitable ultérieurement. C'est le rôles des périphériques.

Avec ce que l'on a vu jusqu'à maintenant, s'il y a une coupure de courant alors tout est perdu car les mini-transistors ne conservent pas leur état sans courant. Donc certains périphériques sont des mémoires « de masse » comme disque dur, disquettes, CDrom, zip, bandes, etc. ou certaines mémoires « flash » ayant une durée raisonnable de conservation des données.

Il faut également pouvoir naturellement entrer les données et les ordres, d'où des périphériques comme clavier, instruments de mesure, etc. Enfin il faut montrer à l'utilisateur les résultats des manipulations symboliques par exemple via un écran, une imprimante...Il faut également partager et échanger des informations avec d'autres ordinateurs $\Longrightarrow$ les réseaux, ce qui conduit directement à la conception par exemple de votre salle de TD.

2 Mise en route

Exercice 1 : Entrainez-vous à :
- déplacer une fenêtre
- « iconifier » une fenêtre et la réouvrir
- changer la taille d'une fenêtre à volonté
- passer une fenêtre en plein écran puis lui redonner sa taille initiale.

Dans les grandes lignes, un programme s'exécute dans chaque fenêtre et un « window manager » (gestionnaire de fenêtres) comme Windows permet de lancer plusieurs programmes en même temps.

Exercice 2 : Faites une manipulation qui révèle la différence entre *quitter* un programme et *iconifier* sa fenêtre.

3 Premier contact avec un langage de programmation : CAML

Si vous avez un ordinateur personnel et si vous souhaitez vous procurer CAML, c'est gratuit :

OCAML est en standard sur la distribution Mandrake, et camllight sur
`http ://pauillac.inria.fr/caml/distrib-caml-light-fra.html`.

Vous pouvez maintenant lancer le programme *Camlwin* dans le menu démarrer. La fenêtre `Terminal` de Camlwin comprend deux sous-fenêtres. Celle du haut contient la session Caml. Celle du bas vous permet de taper les expressions à soumettre à Caml. Pour soumettre une expression à Caml, il vous suffit de taper sur la touche Entrée. Pour sauter une ligne, il faut utiliser la touche enter[1].

Le symbole `#` est appelé *prompt* et indique que le mode interactif de *Caml* attend que vous tapiez quelque chose. Le texte que vous allez taper que l'on appellera *expression* dans la suite de ce document commencera juste après le caractère `#` et se terminera lorsque vous taperez un double point virgule[2] `;;`.

Par exemple tapons `1 + 2 ;;` Entrée. Nous proposons ainsi à *Caml* l'expression `1 + 2`

```
#1 + 2;;
- : int = 3
```

Cet exemple très simple illustre le travail que réalise le mode interactif de *Caml* :

1. lire l'expression tapée : `1 + 2`
2. calculer sa valeur
3. afficher le résultat de cette évaluation `- : int = 3` puis un nouvelle prompt pour revenir en 1.

Le résultat affiché (`- : int = 3`) nous indique que :

« `-` » : il a juste calculé une valeur sans lui donner un nom particulier ;
« `: int` » : cette valeur est de type entier relatif ;
« `= 3` » : elle vaut 3.

Pour quitter le mode interactif de *Caml*, taper `quit() ;;` après le prompt ou sélectionner `Exit` dans le menu `File`.

Ce mode interactif convient bien pour des expressions courtes. Pour réaliser des programmes plus longs, il est plus pratique d'écrire directement un programme avec un éditeur de texte. Nous utiliserons donc l'éditeur associé à Camlwin : « *Cmdcaml* ». On peut lancer cet éditeur par le menu « `Démarrer` ». *Cmdcaml* lance à son tour une fenêtre Camlwin.

Lorsqu'on écrit un programme avec l'éditeur Cmdcaml, on peut tester de différentes façons les expressions tapées :
- dans camlwin : utiliser `include` du menu `File`,
- évaluer l'expression « `include "a :\\td1";;` » si votre fichier a été sauvé sous le nom `a :\td1.ml`,[3]
- sélectionner avec la souris l'expression à tester dans l'éditeur, copier l'expression (menu Edition/copier ou Ctrl-C), coller l'expression dans camlwin (menu Edition/coller ou Ctrl-V),
- utiliser le menu `Evaluer/Tout` ou le racourci clavier Ctrl-T, qui permet d'évaluer l'ensemble du fichier.

Votre découverte de l'environnement de travail est maintenant terminée. Pour vous familiariser avec cet environnement, relancez CmdCaml et soumettez l'exemple suivant à Caml en utilisant les différentes méthodes vues plus haut.

```
(* Premier exemple *)
1 + 2 + 3 ;;
(* deuxieme exemple *)
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 + 18
+ 19 + 20 ;;
( 20 + 1 ) * 20 ;;
(* autre exemple *)
(3*5+2) * (3*5+2) * (3*5+2)  + 4 * (3*5+2) * (3*5+2) + (3*5+2) + 18 ;;
```

Caml ignore tout ce qui est placé entre `(*` et `*)` ; on peut ainsi introduire des commentaires (qui peuvent occuper plusieurs lignes) pour faciliter la compréhension d'un programme.

4 Définition de la syntaxe des expressions booléennes

Un programme dans un langage donné est une suite de symboles, construite en respectant certaines règles. La *syntaxe* explique comment sont construits les mots et les phrases valides du langage (les expresions), la *sémantique* en donne le sens.

Par exemple on peut définir les formes possibles d'expressions booléennes. Ainsi, une expression booléenne est :

[1] en bas à droite du pavé numérique
[2] ces 2 points-virgules doivent être consécutifs. Il ne faut les séparer l'un de l'autre ni par un espace, ni par un saut de ligne
[3] Ainsi, dans la commande `include`, chaque caractère \ présent dans le nom doit être doublé et l'extension « `.ml` » peut être omise.

– une constante booléenne (`true` ou `false`)
– ou une expression unaire de la forme « `not` *exp* » où *exp* est une expression booléenne plus simple
– ou une expression binaire de la forme « *exp1* `&&` *exp2* »où *exp1* et *exp2* sont des expressions booléennes plus simples
– ou une expression binaire de la forme « *exp1* `||` *exp2* »où *exp1* et *exp2* sont des expressions booléennes plus simples

Il s'agit là d'une définition par récurence, très simple, dont la première ligne est le cas de base.
Exemples d'expressions correctes (vis à vis de ces règles) : « `true` », « `true && false` », « `true && false || true` ».
Expressions incorrectes : « `true &&` », « `true && || false` ». Elles comportent des *erreurs syntaxiques*.

Exercice 3 : Tapez les exemples d'expressions précédentes (correctes ou incorrectes).
Lorsque deux opérateurs se suivent, essayez d'entrer ces opérateurs avec, puis sans espace entre eux.

Un opérateur *unaire* est un opérateur qui prend un seul argument. Ici, `not` est unaire. Un opérateur *binaire*, comme `&&` et `||`, prend deux arguments. L'*arité* d'un opérateur est son nombre d'arguments.

Un opérateur binaire peut être placé, comme ici, en position *infixe* c'est-à-dire, entre ses arguments. Il peut aussi être placé en position *préfixe* (devant ses arguments : « `||` exp exp » ou *postfixe* (derrière ses arguments : « exp exp `||` »). Dans le langage CAML, `||` et `&&` sont infixes mais les fonctions que vous programmerez vous-mêmes seront préfixes.

5 Priorité et parenthésage à gauche

Nous allons maintenant travailler avec des expressions arithmétiques que nous pouvons définir de la façon suivante. Une expression arithmétique est :
– un nombre entier relatif
– ou une expression binaire de la forme « *exp1* `+` *exp2* » où *exp1* et *exp2* sont des expressions arithmétiques
– ou une expression binaire de la forme « *exp1* `-` *exp2* » où *exp1* et *exp2* sont des expressions arithmétiques
– ou une expression binaire de la forme « *exp1* `*` *exp2* » où *exp1* et *exp2* sont des expressions arithmétiques
– ou enfin une expression binaire de la forme « *exp1* `/` *exp2* » où *exp1* et *exp2* sont des expressions arithmétiques

Nous savons maintenant comment écrire des expressions arithmétiques syntaxiquement correctes. Notre problème est de savoir quel sens leur donner. Tout d'abord, il convient de dire que le sens attribué aux symboles d'opérateur est le sens usuel. Ainsi, `+` est interprété comme l'addition entière, `/` est interprété comme la division entière et le mot `56` est interprété comme l'entier 56. Mais il subsiste quelques problèmes. Par exemple, quelle va être la valeur de l'expression `2*3+4` ?

En effet, si l'on ne nous précise rien[4], nous ignorons dans quel ordre effectuer les opérations.

L'expression $2*3+4$ peut *a priori* être interprétée comme $A =$ « `(2 * 3) +4` » ou comme $B =$ « `2 * (3 +4)` ».

Exercice 4 : De combien de façon peut-on évaluer l' expression `2*8/4+5` ?

Une première solution pour départager ces différentes façons consisterait à utiliser la notation préfixe ou postfixe qui ne posent pas ces problèmes

Exercice 5 : Donner A et B en notation préfixe puis postfixe. Expliquez pourquoi elles ne sont pas ambigües.

Une autre solution consisterait à ajouter des parenthèses pour lever l'ambiguïté.

Pour limiter l'usage de parenthèses dans la notation infixe, on peut définir un ordre d'utilisation, appelé *priorité* sur les opérateurs. Un opérateur *op1* est prioritaire devant *op2* si *op1* « choisit » ses arguments avant *op2*. Par exemple, si "*" est prioritaire devant "+", $2*3+4$ se lit $(2*3)+4$. Si on souhaite un groupement différent des arguments, il faut alors utiliser des parenthèses : $2*2+4$ vaut 8 alors que $2*(2+4)$ vaut 12. Si deux opérateurs ont même priorité, leurs arguments leur sont fournis de gauche à droite. Par exemple, si "*" et "/" ont même priorité, $2*3/4$ signifie $(2*3)/4$. Les parenthèses servent donc à préciser le sens des expressions.

Un opérateur peut également être déclaré « associant à gauche » : dans une expression contenant plusieurs occurrences de cet opérateur, les arguments sont groupés de gauche à droite. Par exemple, si "+" associe à gauche, $2+3+4+5$ signifie $((2+3)+4)+5$. Il peut de même être déclaré « associant à droite » (si "+" associe à droite, $2+3+4+5$ signifie alors $2+(3+(4+5))$).

Notons que ce statut d'association à gauche ou à droite n'est qu'une facilité syntaxique et ne préjuge en rien des propriétés mathématiques des opérations interprétant le symbole. Dans le cas présent, l'addition est associative. Les expressions proposées ont donc la même valeur.

[4] et donc si l'on ne sait pas si les règles usuelles en mathématiques sont valables

6 Expressions arithmétiques et booléennes

Les opérateurs `=`, `<`, `>`, `>=` et `<=` permettent de comparer deux entiers de ML.

Ainsi, `a=b` est une expression à valeurs booléennes qui vaut `true` si `a` est égal à `b` et `false` sinon. Par exemple, `2=3` vaut `false` tandis que `2+3=5` vaut `true`.

Exercice 6 : Évaluez les expressions suivantes d'abord sur papier puis ensuite à l'aide de Camlwin et commentez les résultats obtenus.

- `3>4`
- `4*64 < 256`
- `3> 4 || 8=2*4`
- `2+3=5=4+1`
- `false=2+3`
- `(2=4) || (4>1) && (19 >= 17+2)`

Quelles sont les priorités ? proposez éventuellement des expressions complémentaires pour confirmer vos affirmations.

7 Les tables de vérité

On a vu en cours les tables de vérité des opérations suivantes :

not	true	false
	false	true

&&	true	false
true	true	false
false	false	false

||	true	false
true	true	true
false	true	false

Elles donnent respectivement les valeurs de vérité des formules « `not`(p) » en fonction de la valeur de vérité de p, puis « $(p$`&&`$q)$ » puis « $(p$`||`$q)$ » en fonction des valeurs de vérité de p et de q.

Exercice 7 : Donnez de même les tables de vérité des formules suivantes en fonction de la valeur de p :

1. $(p$ `&&` $p)$
2. $(p$ `||` `not`$(p))$
3. $(p$ `&&` `not`$(p))$
4. `not`(`not`(p))
5. `not(` p `&&` `not`(p) `)`

Exercice 8 : Donnez les tables de vérité des formules suivantes en fonction des valeurs de p et q :

1. `not(` `not`(p) `&&` `not`(q) `)`
2. `not(` `not`(p) `||` `not`(q) `)`
3. `(` q `||` `not`(p) `)`
4. $(p$ `&&` $q)$ `||` `(` p `&&` `not`(q) `)` `||` `(` `not`(p) `&&` q `)` `||` `(` `not`(p) `&&` `not`(q) `)`

Pouvez-vous écrire la première et la deuxième formule sous des formes plus simples ayant les mêmes tables de vérité ? lequelles ?

Que signifie la troisième formule ?

Exercice 9 : Deux formules sont dites *équivalentes* si elles ont la même table de vérité. Parmi les formules des deux exercices précédents, quelles sont les formules équivalentes deux à deux ?

Une *tautologie* est une formule qui est toujours vraie, quelles que soient les valeurs de vérité des variables qu'elle contient. Quelles sont les tautologies parmi les formules précédentes ?

8 Déclarations : associer un nom à une valeur

Évaluez la valeur de l'expression :

```
((2 * 4) * (2 * 6) + 1) * ((2 * 4) * (2 * 6) + 1) + 2 * ((2 * 4) * (2 * 6) + 1) + 7
```

Vous pouvez utiliser le copier/coller de windows pour éviter d'avoir à retaper la sous-expression $((2*4)*(2*6)+1)$. Néanmoins, l'écriture mathématique suivante serait plus confortable :

$$\textit{Soit } a = (2*4)*(2*6)+1\textit{, calculons } a^2+2a+7$$

En donnant un nom à une sous-expression, nous rendons ainsi l'ensemble de l'expression plus lisible. En CAML :

```
# let a = (2 * 4) * (2 * 6) +1 ;;
a : int = 97
# a * a + 2 * a + 7 ;;
- : int = 9610
```

On peut donc maintenant construire des expressions arithmétiques comportant des *identificateurs*. Par exemple le symbole `a` est un identificateur dans les lignes précédentes en ML.
Un identificateur est un nom formé à partir des lettres de l'alphabet (minuscules et majuscules), du caractère souligné `_` et des chiffres de 0 à 9 (ne confondez pas le caractère `_` qui est sur la même touche que le caractère `8` et le caractère `-` qui est sur la même touche que le caractère `6`). Un identificateur commence obligatoirement par une lettre, peut comporter jusqu'à 256 caractères mais surtout pas de blancs.

Exercice 10 : Dans la session suivante, devinez les réponses affichées par ML puis vérifiez attentivement vos réponses en soumettant les phrases à CAML les unes après les autres.

```
# let nbr_1 = 25 ;;
# let  nbr_2 = nb1 + 10 ;;
# let  nbr_2 = nbr_1 + 10 ;;
# let  nbr_1 = nbr_2 + 20 ;;
# nbr_1 ;;
# nbr_2 * nbr_1 + 10;;
```

9 Expressions et déclarations

Exercice 11 : (portée lexicale). Tapez la session ML suivante :

```
# let const = 3 ;;
# let f x = x * const ;;
# f 1 ;;
# let const = 6 ;;
# f 1 ;;
```

Que constatez-vous ? Expliquez pourquoi l'on obtient ce résultat.

Exercice 12 : Expliquez pourquoi certaines des expressions suivantes déclenchent des erreurs. Corrigez la première.

```
# let f (a,b,c) = if a = 0 then if b = 0 then if c = 0 then "tous nuls"
    else "pas tous nuls" ;;
# let x = 7.0 ;;
# if (x mod 2) = 0 then true  else "impair" ;;
# let nb_sol_deg1 (a,b) = if a =. 0.0  then (nb_sol_deg0 b)  else 1 ;;
```

Exercice 13 : Indiquez d'abord les réponses de ML et ensuite vérifiez-les en soumettant la session à CAML.

```
# let x = 25 in x + 2 * x ;;
# x + 1 ;;
# let x = 1 ;;
# let x = 4 in x + 2 * x ;;
# x + 2 ;;
# let t = 3 and y = let z = x + 3 in z + 2 ;;
# let y = let z = y + 3 in z + 2 ;;
```

10 Portée des déclarations

Exercice 14 : Indiquez d'abord les réponses de ML et ensuite vérifiez-les en soumettant la session à CAML.

```
# let pi = "pi" in "surface = " ^ pi ^ " r " ^ "au carré" ;;
# pi ^ " (disque)" ;;
# let pi = 3.14 ;;
# let pi = 10. /. 3. in 2. *. pi *. 3.0 ;;
# pi +. 2.0 ;;
# let pi = 4. and y = let z = pi *. 2. in z +. 2. ;;
# let y = let z = y + 3 in z + 2 ;;
```

TD d'Informatique No. 2

1 Fonctions simples

Exercice 1 : Que répond ML pour chaque ligne ci-dessous ?

```
# let mon_poly x = x *. x +. 2. *. x +. 1. ;;
# mon_poly 3. ;;
# let mon_poly x = x *. x -. 2. *. x +. 1. ;;
# mon_poly 3. ;;
```

Exercice 2 : Écrivez les fonctions suivantes :

1. `volume_sphere` ayant un nombre réel `r` (type `float`) comme argument et donnant le volume de la sphère de rayon `r`
2. `surface_carre`, `surface_disque`, `surface_rectangle` et `surface_triangle` qui donnent respectivement la surface d'un carré, d'un disque, d'un rectangle, d'un triangle (en fonction de la base et de la hauteur).

Exercice 3 : Écrivez une fonction qui teste si son argument est pair. (On rappelle que l'expression `a mod b` retourne le reste de la division euclidienne de `a` par `b`.)

Exercice 4 : Écrivez une fonction `encadre` qui prend une chaîne de caractères en argument et ajoute de chaque côté la chaîne `"!!"` (appelée cadre). Par exemple `(encadre "glop")` retourne la chaîne de caractères `"!!glop!!"`. Donnez le type de cette fonction.

On veut maintenant que la chaîne cadre soit un paramètre du problème. Écrire la nouvelle fonction et donner son type.

Exercice 5 : Écrire une fonction nommée `signe` qui prend en argument un entier relatif et en indique le signe par une chaîne de caractères (`"positif"`, `"negatif"` ou `"nul"`).

Exercice 6 : Être ou ne pas être un triangle. On admettra qu'un triplet (a, b, c) définit un triangle si et seulement si on a les trois inégalités $a + b \geq c$, $b + c \geq a$ et $c + a \geq b$.

1. Écrire une fonction de 3 arguments `r1`, `r2` et `r3` qui retourne `true` si ces 3 réels définissent un triangle, `false` sinon.
2. Écrire une fonction qui retourne
   - `"equilateral."` si les 3 réels forment un triangle équilatéral (les 3 côtés égaux)
   - `"isocele."` si les 3 réels forment un triangle isocèle (seulement 2 des 3 côtés égaux)
   - `"quelconque."` si les 3 réels forment un triangle ni équilatéral, ni isocèle
   - `"non triangle."` si les 3 réels ne définissent pas un triangle.
3. Écrire une fonction qui indique si les 3 réels définissent un triangle rectangle.

2 Décomposition de nombres

Exercice 7 : Définissez 3 fonctions nommées `unite`, `dizaine` et `centaine` telles que `unite n` retourne le chiffre des unités de `n`, `dizaine n` retourne le chiffre des dizaines de `n` et `centaine n` retourne le chiffre des centaines de `n`.

Par exemple pour l'entier 2311, le chiffre des unités est 1, celui des dizaines est 1 et celui des centaines est 3.

Exercice 8 : Sachant que la fonction `int_of_float : float -> int` calcule la partie entière d'un nombre flottant et que la fonction `float_of_int : int ->float` représente le plongement « canonique » de `int` dans `float` (c'est-à-dire l'inclusion de $\mathbb{Z}$ dans $\mathbb{R}$), trouvez ce que fait la fonction suivante. Tapez cette fonction au clavier.

```
# let puissance n = int_of_float ( exp ((log 10.0) *. (float_of_int n)) ) ;;
```

Exercice 9 : Pour généraliser les fonctions `unite`, `dizaine` et `centaine`, écrivez une fonction `chiffre` telle que `chiffre(k,n)` fournisse le `k`-ième chiffre (en partant de la droite) du nombre `n`. On utilisera la fonction `puissance` précédente.

Exercice 10 : Définissez la fonction `deuxDerniers` qui donne les deux derniers chiffres de son argument. Par exemple `deuxDernier 2311` donne `11`.

Définissez ensuite la fonction `tranche` telle que `tranche(d,k,n)` donne les `d` chiffres de `n` à gauche du `k`-ième chiffre inclus (`k`-ième en partant de la droite du nombre). Par exemple, `tranche(2,2,2311)` donne `31`.

Exercice 11 : une application pratique. Écrire les fonctions `sexe`, `annee` et `mois` qui prennent en argument un numero de sécurité sociale (à 7 chiffres) et trouvent respectivement le sexe (chaîne de caractères), l'année ou le mois de naissance correspondant.

Exercice 12 : vérifier « à la main » le bon typage. Pour chacune des fonctions déclarées dans cette partie « Décomposition des nombres », tracer l'arbre de leur expression et vérifier à la main qu'elles sont bien typées.

TD d'Informatique No. 3

1 Déclarations locales, failwith, typage

Exercice 1 : Indiquez d'abord les réponses de ML et ensuite vérifiez-les en soumettant la session à CAML.

```
# let x = 2 ;;
# let t = 3 and y = let z = x + 3 in z + 2 ;;
# let x = 5 in  let y = 3 + x in  let z = y + x in  x + y + z  ;;
# let u = (if t = 3 then failwith "Echec!" else 7 mod (t-3))  in  u + 1  ;;
```

Exercice 2 : Un étudiant a eu une note d'examen et une note de DS. Sa note finale sera la plus grande des deux notes suivantes : sa note d'examen ou la moyenne de sa note d'examen et de sa note de DS.

1. Écrivez une fonction `noteFinale` qui prend en arguments deux flottants représentant la note de DS et la note d'examen d'un étudiant et qui donne la note finale de l'étudiant.
2. Écrivez une fonction `estAdmis` qui prend deux flottants représentant la note de DS et la note d'examen d'un étudiant et détermine si l'étudiant est reçu ou ajourné. Elle pourra utiliser la fonction `noteFinale`.

Exercice 3 : Écrivez une fonction `resoudre(a,b,c)` qui trouve une solution de l'équation $(ax + b = c)$ si elle existe.

2 Géométrie dans le plan

Nous allons écrire des fonctions permettant de travailler avec quelques figures géométriques simples du plan. Les différents types de figure géométrique considérés sont :

- le `point`, représenté par sa coordonnée en x (son abscisse) et sa coordonnée en y (son ordonnée)
- le `rectangle` plein, dont l'un des cotés est horizontal, représenté par deux de ses sommets : le sommet en bas à gauche que l'on nommera `SO` (comme Sud-Ouest) et le sommet en haut à droite que l'on nommera `NE` (comme Nord-Est).
- le `segment` (non nécessairement horizontal ou vertical) représenté par ses deux points extrêmes `S1` et `S2`
- le `disque` représenté par son centre et son rayon (positif ou nul)

Exercice 4 : Déclarez les types `point`, `rectangle`, `segment` et `disque`. Donnez ensuite quelques exemples d'éléments de ces types. Pour les exercices ultérieurs, vous testerez systématiquement vos fonctions à l'aide de ces exemples.

Exercice 5 : Écrivez une fonction `distance` qui calcule la distance entre deux `point`s. On rappelle que :

- la fonction `sqrt` calcule la racine carrée de son argument.
- la distance entre un point p_1 et un point p_2 d'abscisses respectives x_1 et x_2 et d'ordonnées respectives y_1 et y_2 vaut $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Exercice 6 :

1. Écrivez les fonctions `nord_ouest` et `sud_est` qui prennent un rectangle en argument et donnent les points correspondants.
2. Écrivez une fonction `perimetre_rectangle` ayant comme argument un rectangle et qui calcule son périmètre.
3. Écrivez une fonction `perimetre_disque` qui calcule le périmètre d'un disque.

Exercice 7 : Écrivez une fonction `dans_disque` ayant un point `p` et un disque `d` comme arguments et qui indique si le point `p` est dans le disque `d`.

Exercice 8 : Écrivez une fonction `rectangle_dans_disque` ayant un rectangle `r` et une disque `c` comme arguments qui indique si `r` est dans `c`.
Écrivez de même une fonction `disque_dans_rectangle` ayant un disque `c` et un rectangle `r` comme arguments qui indique si `c` est dans `r`.

3 Restaurant du personnel

Les employés d'une entreprise accèdent à leur restaurant grâce à une carte à puce contenant les informations suivantes :

- Nom et Prénom du possesseur de la carte
- Année de validité
- Salaire mensuel
- Nombre de tickets restant en réserve

Un repas peut être composé au choix de zéro ou un plat chaud et d'autant de plats complémentaires (entrée, fromage, dessert) que l'on veut.

Un plat chaud coûte 2 tickets et un plat complémentaire coûte un ticket. Si une personne prend un plat chaud et trois plats complémentaires, on lui compte un repas complet qui lui coûte seulement trois tickets.

Exercice 9 : Définissez un type enregistrement nommé `carte` permettant de représenter une carte de restaurant. Déclarez ensuite une constante de type carte appelée `ma_carte` dont le nom est `"Moi"`, le prénom `"Bibi"`, l'année de validité est l'année en cours, le salaire 950 et le nombre de tickets en réserve 36.

Exercice 10 : Écrivez une fonction `ajout` ayant une carte `c` et un nombre de tickets `t` comme arguments qui renvoie une carte déduite de `c` en lui ajoutant `t` tickets en réserve.

Remarque : Dans un langage fonctionnel comme ML, une fonction ne modifie pas ses arguments. Il en est de même en mathématiques où dire que $f(x)$ modifie x n'aurait aucun sens. Par conséquent, si l'on veut mettre à jour `ma_carte` en achetant 50 tickets, l'expression « `ajout(ma_carte,50)` » ne suffira pas pour modifier `ma_carte` ; il faudra en fait *redéfinir* `ma_carte` c'est-à-dire écrire « `let ma_carte = ajout(ma_carte,50);;` »

Exercice 11 : Écrivez une fonction `valide` ayant comme arguments une carte et l'année en cours et qui teste si la carte est valide.

Exercice 12 : Le prix d'un ticket dépend du salaire.

- Strictement positif mais inférieur à 1000 € : ticket à 4 €
- Strictement supérieur à 1000 € : ticket à 5 €
- Un stagiaire sera considéré comme gagnant un salaire nul et paiera 3 € son ticket.

Écrivez une fonction `achat` ayant une carte `c` et une somme `s` (un nombre entier d'euros) comme arguments et qui renvoie une carte déduite de `c` en ajoutant le nombre de tickets correspondant à la somme `s` (on négligera la monnaie rendue).

Exercice 13 : Écrivez une fonction `manger` ayant comme arguments une carte `c`, un nombre de plats chauds `chaud` et un nombre de plats de complément `comp`, et qui retourne une carte issue de la carte `c` en lui enlevant le nombre adéquat de tickets. On fera échouer la fonction s'il n'y a pas assez de tickets ou trop de plats chauds.

TD d'Informatique No. 4

Exercice 1 : Écrivez une fonction `prix` en ML qui donne le prix en Francs d'un coup de téléphone en fonction de deux arguments : la durée de l'appel en secondes et l'heure d'appel. Le prix doit être calculé sachant que :
- l'heure est simplement représentée par un entier compris entre 0 et 23, qui est la partie entière de l'heure véritable, afin de simplifier les calculs.
- les 60 premières secondes coûtent 30 centimes (c'est-à-dire 0.3€) quelle que soit l'heure d'appel et tout appel de moins de 60 secondes coûte également 30 centimes.
- chaque seconde supplémentaire coûte 2 centimes (c'est-à-dire 0.02€) si l'heure d'appel est comprise entre 8 et 20 inclus, et 1 centime (c'est-à-dire 0.01€) si l'heure d'appel est comprise entre 21h et 7h inclus (c'est-à-dire en fait 7h59 puisqu'on ne représente que la partie entière de l'heure).

On pourra utiliser la fonction `float_of_int` de ML qui transforme un nombre entier (type `int`) en nombre réel (type `float`).

Nombres complexes

Exercice 2 : Un nombre complexe sera représenté par ses parties `reelle` et `imaginaire` qui sont des réels. Déclarez le type `complexe` correspondant puis déclarez les complexes `i` et `z0` valant respectivement : i et $1+i$.

Exercice 3 : Écrivez une fonction `module` qui calcule le module d'un nombre complexe.

Exercice 4 : Écrivez les fonctions `somme`, `soustr` et `mult` ayant deux nombres complexes comme arguments et calculant respectivement leur somme, soustraction et multiplication.

Fonctions récursives

Exercice 5 : Écrivez une fonction `puissance` qui prend un nombre réel `r` et un entier `n` en arguments et qui calcule r^n. On prendra soin de gérer proprement tous les cas (`n` négatif, etc.).

Exercice 6 : Écrivez une fonction `pgcd` qui calcule le pgcd de 2 nombres sur le principe bien connu suivant : s'ils sont égaux alors c'est facile, sinon c'est égal au pgcd du plus petit des deux nombres avec la différence des deux ($pgcd(m,n) = pgcd(m, n-m)$). On fera attention aux questions de signe des deux nombres.
Écrivez une fonction `pgcd2` qui utilise plutôt l'équation $pgcd(m,n) = pgcd(m, n \ mod \ m)$ et comparez leurs performances.

Exercice 7 : Écrivez une fonction `produitInterv` qui prend deux entiers `a` et `b` en entrée et fournit en sortie le produit des nombres compris entre `a` et `b`.

Exercice 8 : On considère la suite récurrente définie par : $\begin{cases} u_0 = 1 \\ u_n = \sqrt{u_{n-1}+1} & \text{si } n \geq 1 \end{cases}$
Écrivez la fonction nommée `u` qui prend en argument l'entier `n` et retourne la valeur de u_n.
Rappelons que `sqrt : float -> float` calcule la racine carrée.

Exercice 9 : Trois marques concurrentes « Super », « Maxi » et « Ultra » vendent ce mois-ci de la lessive au kilo respectivement aux prix de 1€50, 3€50 et 5€.
Chaque mois, chaque marque veut suivre ses concurrents et si elle est soit la plus chère, soit la moins chère des trois, alors elle décide d'aligner son prix sur la moyenne des prix de ses concurrents au mois précédent. Par exemple le mois prochain Super vendra sa lessive à $\frac{3,50+5}{2} = 4$€25 du kilo, Maxi restera au même prix, etc.
Programmez les fonctions `prix_Super`, `prix_Maxi` et `prix_Ultra` qui prennent en argument un entier `n` et calculent le prix du kilo de la lessive correspondante dans `n` mois.
Indications : pour simplifier on ne cherchera pas à arrondir les résultats, et pensez à programmer une fonction intermédiaire, par exemple `let aligne(moi,autre1,autre2) =...`

Exercice 10 : Écrivez une fonction `symetrique` qui prend un nombre entier `n` en argument et retourne un booléen qui dit si les chiffres qui le constituent forment un palindrome.
Indication : cette fonction ne doit être définie que sur les nombres positifs ; s'il n'y a qu'un chiffre c'est facile, sinon comparer les premiers et derniers chiffres et recommencer avec le milieu.

Exercice 11 : Écrivez une fonction `nbSym` qui prend deux entiers `a` et `b` en entrée et fournit en sortie le nombre d'entiers compris entre `a` et `b` qui sont des nombres `symetriques`.

SdV-1

TD d'Informatique No. 5

Exercice 1 : Écrire une fonction `troisieme` qui fournit le troisième élément d'une liste (s'il existe).

Exercice 2 : Écrire une fonction `complete` qui prend une liste de réels en argument et qui :
- si la liste contient plus de deux éléments, ajoute la moyenne des deux premiers éléments au début de la liste
- si la liste contient un seul élément, double l'occurrence de cet élément
- si la liste est vide, rend la liste contenant seulement 0.

Exercice 3 : Écrire une fonction `dernier` qui prend une liste en argument et rend le dernier élément de la liste.

Exercice 4 : Écrire une fonction `plus_grand` qui prend une liste d'entiers en argument et rend le plus grand élément de la liste. Indication : la fonction prédefinie `max` calcule le maximum entre deux entiers.

Exercice 5 : Écrire une fonction `concatene` qui prend une liste de chaînes de caractères en argument et rend la concaténation de tous les éléments de cette liste.

Exercice 6 : Écrire une fonction `begayer` qui prend une liste en argument et rend une liste où chaque élément apparaît deux fois *consécutives* (la longueur de la liste fournie en résultat est donc deux fois celle de départ).

Exercice 7 : Écrire une fonction `est_dans` qui prend un élément et une liste en arguments et teste si l'élément est dans la liste. Note : une « fonction qui teste ...blabla... » retourne un booléen (`true` ou `false`).

Exercice 8 : Écrire une fonction `indice` qui prend un élément et une liste en arguments et rend l'indice de l'élément dans la liste, s'il existe.

Exercice 9 : Écrire une fonction `dico` qui prend un mot et une liste de couples (mot,définition) en argument et rend la définition correspondante au mot si elle existe, ou le mot lui-même sinon.

```
#dico;;
- : 'a * ('a * 'a) list -> 'a = < fun >
#let d = [("bonjour","salutation matinale.");
("caml","langage de programmation fonctionnelle.");
("nul","(1)pas bien (2)element neutre de certaines operations.")];;
d : (string * string) list =...
#dico ("caml",d);;
- : string = "langage de programmation fonctionnelle"
#dico ("pluc",d);;
- : string = "pluc"
```

Exercice 10 : Écrire une fonction `insere` qui prend un élément, un indice et une liste en arguments et rend une liste dans laquelle l'élément est ajouté et apparaît à l'indice choisi. Si l'indice proposé dépasse la taille de la liste, alors l'élément est inséré en fin de liste.

```
#insere;;
- : 'a * int * 'a list -> 'a list = < fun >
#insere (0,2,[1;2;3;4]);;
- : int list = [1; 0; 2; 3; 4]
#insere (0,6,[1;2;3;4]);;
- : int list = [1; 2; 3; 4; 0]
#insere (0,-1,[1;2;3;4]);;
- : int list = [0;1; 2; 3; 4]
```

Exercice 11 : Écrire une fonction `supprime` qui prend un élément et une liste en arguments et supprime toutes les occurrences de cet élément dans la liste.

Exercice 12 : Écrire une fonction `supp_doublons` qui prend une liste en argument et rend une liste où chaque élément n'apparaît qu'une seule fois.

Exercice 13 : Écrire une fonction `decoupe` qui prend un indice et une liste en arguments et rend un couple de listes telles que :
- la première est le début de la liste initiale jusqu'à l'indice inclus
- la seconde est le reste de la liste initiale à partir de l'indice.

```
#decoupe;;
```

```
- : int * 'a list -> 'a list * 'a list = < fun >
#decoupe (3,[1;2;3;4;5]);;
- : int list * int list = [1; 2; 3], [4; 5]
#decoupe (8,[1;2;3;4;5]);;
- : int list * int list = [1; 2; 3; 4; 5], []
#decoupe (0,[1;2;3;4;5]);;
- : int list * int list = [], [1; 2; 3; 4; 5]
```

TD d'Informatique No. 6

1 Fonctions diverses sur les listes

Exercice 1 : Écrivez une fonction `resumer` qui prend en argument une liste de réels `l` et retourne la liste de réels dont le premier élément est la moyenne des deux premiers de `l`, le second est la moyenne des troisième et quatrième éléments de `l`, le troisième est la moyenne des cinquième et sixième éléments de `l`, etc. Si la liste `l` est de longueur impaire, on copie de dernier élément tel quel.

Exercice 2 : Même exercice en ne prenant pas en compte les valeurs nulles de la liste `l`.

Exercice 3 : Écrivez une fonction `moyenne` qui prend en argument une liste de nombres réels et rend la moyenne de ces nombres.

Exercice 4 : Écrivez une fonction qui prend en arguments un élément `e` et une liste `l` supposée triée et non redondante, et qui retourne une liste également triée et non redondante obtenue en insérant `e` au bon endroit dans `l`. Noter qu'on ne sait pas *a priori* si la liste est strictement croissante ou strictement décroissante. Si `l` contient déjà `e` alors on ne l'insère pas ; de plus si `l` est de longueur inférieure ou égale à 1, on place `e` à gauche de `l`.

Exercice 5 : Écrivez une fonction qui prend en arguments une liste `p` (appelée le « motif », « pattern » en anglais) et une seconde liste `l` de même type, et qui retourne un entier indiquant à quel rang de la liste `l` on peut trouver le motif. Par exemple `filtre ([T;A;T;A],[C;G;T;C;T;A;G;T;A;T;A;G;T;C])` donne `8` comme résultat car le motif `[T;A;T;A]` apparaît en huitième position de la liste `[C;G;T;C;T;A;G;T;A;T;A;G;T;C]`. La fonction doit fournir un message d'erreur si le motif n'est pas présent dans `l`.

2 Polynômes à coëfficients entiers

On représente un polynôme par une liste de monômes, triée par degrés strictement décroissants. Dans les exercices qui suivent, sauf le 11, on supposera que les polynômes donnés en argument des fonctions sont bien formés, c'est-à-dire que les degrés des monômes sont toujours positifs ou nuls, en ordre strictement décroissants, et que les coëfficients sont non nuls. Le polynôme nul sera représenté par la liste vide.

Exercice 6 : Chaque monôme est représenté par son coëfficient `coef` et son degré `deg`. Définissez ainsi le type `monome`. Déclarez ensuite deux ou trois exemples de polynômes.

Exercice 7 : Écrivez une fonction `degre` qui donne le degré d'un polynôme. Par convention, le degré du polynome nul ([]) sera -1.

Exercice 8 : Écrivez une fonction `valuation` qui prend un polynôme `p` en argument et rend le degré de son monôme de plus faible degré. Si le polynôme est nul ([]) rendre -1.

Exercice 9 : Écrivez une fonction `extrait` qui extrait d'un polynôme `p` le polynôme constitué de ses monômes de degré pair.

Exercice 10 : Écrivez une fonction `valeur` qui prend en arguments un polynôme `p` et une valeur entière `x`, et qui rend la valeur de `p(x)`.

Exercice 11 : Écrivez une fonction `correct` qui vérifie si une liste de monônes constitue un polynôme bien formé.

Exercice 12 : Écrivez une fonction `ajoute` qui effectue la somme de deux polynômes.

Exercice 13 : Écrivez une fonction qui effectue le produit d'un monôme avec un polynôme.

Exercice 14 : Écrivez une fonction qui effectue le produit de deux polynômes.

TD d'Informatique No. 7

Gestion d'une bibliothèque de prêts

L'objectif de ce TD est d'écrire un type de données muni de fonctions de gestion qui permettent la gestion du stock en livres et des abonnés d'une bibliothèque.

Exercice 1 : Un abonné est fiché à la bibliothèque avec les informations suivantes : son `numero` d'inscription qui sert de clef, son `nom`, son `adresse` et son numero de `tel`éphone. Écrivez le type `abonne` correspondant.

Exercice 2 : Un livre est répertorié à la bibliothèque par son numéro d'`ISBN` qui sert de clef, la liste de ses `auteurs`, son `titre`, le `nombre` d'exemplaires qui appartiennent à la bibliothèque et la liste des numéros des abonnés qui détiennent actuellement un exemplaire de ce livre en `emprunts`. Écrivez de type `livre` correspondant. Déclarez ensuite trois livres différents quelconques.

Exercice 3 : La gestion d'une bibliothèque impose de connaître à tout moment deux choses : la liste des abonnés `inscrits` à la bibliothèque et la liste des livres du `stock` de la bibliothèque, empruntés ou non. Ecrire le type `biblio` correspondant. Déclarez ensuite une bibliothèque `vide` qui n'a aucun abonné et aucun livre.

On gèrera les listes d'inscrits et le stock en les triant sur la clef de manière strictement *décroissante*. Par ailleurs les numéros des abonnés sont simplement incrémentés de 1 à chaque nouvel abonné.

Exercice 4 : Écrivez la fonction `inscrire` qui ajoute un nouvel abonné à une bibliothèque. On supposera sans le vérifier que le nouvel abonné n'est pas déjà inscrit.

Exercice 5 : Écrivez une fonction `desabonne` qui supprime un abonné (dont on connaît le numéro) de la liste des abonnés inscrits d'une bibliothèque. Ne rien changer si le numéro n'était déjà pas parmi les inscrits.

Exercice 6 : Écrivez une fonction `autorisé` qui vérifie si un numéro d'abonné fait réellement parti des abonnés inscrits d'une bibliothèque.

Exercice 7 : Écrivez la fonction `acheter` qui ajoute dans le stock de la bibliothèque `n` exemplaires d'un livre qui n'y est pas déjà. Laisser le stock inchangé si le numéro ISBN fourni était par erreur déjà en stock.

Exercice 8 : Écrivez la fonction `completer` qui ajoute dans le stock de la bibliothèque `n` exemplaires d'un livre qui y est déjà. Laisser le stock inchangé si par erreur le numéro ISBN fourni n'appartenait pas déjà au stock.

Exercice 9 : Écrivez la fonction `emprunter` qui permet à un abonné inscrit à la bibliothèque d'emprunter un exemplaire d'un livre du stock. Si le numéro de l'abonné n'est pas dans la liste des inscrits, ou si le numéro ISBN du livre n'appartient pas au stock, ou enfin si l'emprunteur a déjà un exemplaire de ce livre, ne pas préter le livre et donc laisser l'état de la bibliothèque inchangé.

Exercice 10 : Écrivez de même la fonction `rendre` qui permet à un abonné de rendre un exemplaire de livre qu'il a emprunté.

Exercice 11 : Écrivez une fonction qui fournit le nombre de livres empruntés par un abonné dont on connaît le numéro.

Exercice 12 : Pour chacune des fonctions récursives (éventuellement déclarées localement) que vous avez programmées dans les exercices précédents, démontrez qu'elle termine.