Université de Bordeaux
Année 2019-2020

Mastère 2
T.A.P

TD n°2 - Programmation fonctionnelle

Exercice 1: Liaisons et environnement

La construction "`let` ⟨symb⟩ = ⟨expr⟩" permet de définir des nouvelles valeurs dans l'environnement sous forme de *liaisons*. Il est aussi possible de définir des valeurs qui restent *locales* à une expression en utilisant la construction : "`let` ⟨symb⟩ = ⟨expr1⟩ `in` ⟨expr2⟩". La valeur définie par ⟨symb⟩ est alors accessible uniquement dans la seconde expression.

Remarque : En OCaml, les définitions par liaison sont toujours des définitions sur des constantes. Même s'il est possible de définir des références sur des objets, il n'y a pas à proprement parler de *variable* dans le langage.

1. Prédire puis tester le résultat du programme suivant :

```
let pi = 3.1415;;
let rot_gen pi (x,y,ang) =
  let newx = x*.cos(ang*.pi) +. y*.sin(ang*.pi) in
  let newy = x*.sin(ang*.pi) -. y*.cos(ang*.pi) in (newx,newy);;
let rot = (rot_gen pi);;
let pi = "troispointquatorze";;
rot (1.,0.,1.);;
```

2. Tester le programme C suivant et réaliser la suite des évaluations OCaml qui lui correspond :

```
#include <stdio.h>

int i = 0;
int f (int j) { i = i + j;  return i; }

int main(void) {
  f(1); printf("%d,␣", i);
  f(2); printf("%d,␣", i);
  i=421; f(2); printf("%d␣\n", i);
}
```

▷ C# est un langage de programmation développé par Microsoft, et qui possède un compilateur et une machine virtuelle open source à l'intérieur du projet Mono (https://www.mono-project.com). Pour programmer en C# sous Mono, il faut utiliser les outils suivants :

- Compilation : utiliser le compilateur `mcs` de la manière suivante pour obtenir un fichier `.exe` à partir d'un fichier de code `.cs` :

```
mcs file.cs
```

- Exécution : utiliser la machine virtuelle de la manière suivante :

```
mono file.exe
```

Voici un exemple de programme C# très simple :

```
using System;
class Program {
  public static void Main() {
    Console.WriteLine("Caramba !");
} }
```

Exercice 2: Délégués C#

Dans le langage C#, il existe une construction permettant d'écrire une lambda-expression sous la forme d'un objet particulier appelé *délégué* (*delegate* en anglais, cf. https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/delegate). Par exemple :

```
// Type declaration
public delegate int OneInt (int value);
// Anonymous function
public static OneInt square = new OneInt(i => i*i);
// One-time declaration using predefined type
public static Func<int, int> cube = i => i*i*i;
// Delegate using multiple arguments
public delegate void Concat (string s1, string s2);
public static Concat c = (a,b) =>
                         Console.WriteLine(a+b);
```

```
(* Equivalent in OCaml *)
int → int

let square i = i*i

let cube i = i*i*i

(string*string) → unit
let c (a,b) =
       print_string (a^b)
```

Le mot-clé `delegate` permet de définir un *type de fonction* à travers les types de ses paramètres et son type de retour. Il devient alors possible de créer de fonctions possédant ce type en utilisant la notation permettant de représenter des fonctions anonymes comme `(a,b) => a+b`. Il existe des types génériques de délégués, nommés `Func<T1,...,TN,TRES>` qui décrivent les fonctions `TRES f(T1,...,TN)`, pour N entre 1 et 16.

1. Construire un délégué C# qui prenne en paramètres deux entiers, une fonction, et qui applique cette fonction à ces deux entiers.
   Appliquer votre délégué à des fonctions simples comme la somme ou le produit.
   Remarque : il est possible d'utiliser un délégué intermédiaire.

La bibliothèque GtkSharp (https://www.mono-project.com/docs/gui/gtksharp et sa documentation http://docs.go-mono.com/?link=N%3aGtk) permet d'utiliser GTK à l'intérieur de Mono. Concrètement, il devient possible de créer une application graphique comme suit :

```
using System;
using Gtk;

public class GtkHelloWorld {
  static void onClick (object obj, EventArgs args) {
    Console.WriteLine("I have been clicked by a {0}", obj); }

  public static void Main() {
    Application.Init();

    //Create the Window
    Window myWin = new Window("Brave new world");
    myWin.Resize(200,200);
    HBox myBox = new HBox (false, 10);
    myWin.Add(myBox);

    // Set up a button object.
    Button hello = new Button ("Hello");
    hello.Clicked += new EventHandler(onClick);
    myBox.Add(hello);

    //Show Everything
    myWin.ShowAll();
    Application.Run(); }
}
```

Cet exemple se compile avec la commande suivante :

```
mcs -pkg:gtk-sharp-2.0 fichier.cs && mono fichier.exe
```

Ici, l'application contient un bouton `hello` qui, lorsque l'on clique dessus, exécute un `EventHandler` constitué ici à partir de la fonction `onClick`.

2. Remplacer la fonction `onClick` par une fonction anonyme.
3. Écrire une application qui contient deux boutons, nommés `ping` et `pong`. Seul l'un des deux boutons doit être actif à un instant donné (cf. la propriété `Sensitive`). Appuyer sur un bouton le désactive et active le second.

L'idée ici tient dans le fait que les handlers associés à chaque bouton doivent avoir connaissance de l'autre bouton.

4. Comment fonctionnent les fermetures en C# ? Vérifier ce fonctionnement en assignant le handler aux boutons, puis en modifiant les boutons après coup.

Exercice 3: Java lambda-expressions

La version 8 du langage Java a introduit la possibilité de créer des lambda-expressions, c'est-à-dire des valeurs fonctionnelles comme celles vues en OCaml. La syntaxe des lambda-expressions en Java 8 est documentée à l'adresse https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html. Typiquement, une lambda-expression s'écrit comme suit :

```
(u,v) → { return u + v; }
```

Cette écriture se décompose en :

— une liste de paramètres formels entre parenthèses, typés ou non ;

— le token '→' ;

— un bloc de code entouré d'accolades.

Comme exemple de méthode utilisant une lambda-expression, la méthode `forEach` possède le prototype suivant :

```
default void    forEach(Consumer<? super T> action)
```

1. Écrire le code permettant d'afficher une liste d'entiers `List<Integer>` à l'aide de la fonction `forEach` et d'une lambda-expression.
2. Où est définie la méthode `forEach` ?
   Pourquoi est-elle marquée comme une méthode par défaut ?
3. Écrire le code permettant de trier une `List<Integer>` à l'aide d'une lambda-expression et de la méthode `sort` de l'interface `List`.
4. Comment se comporte l'inférence de type pour les lambda-expressions ?
   Quels sont les types des valeurs fonctionnelles ainsi créées ?
   Comparer l'expressivité des lambdas-expressions Java à celles de C# et d'OCaml.
   Note : La documentation est accessible à l'adresse `https://docs.oracle.com/javase/10/docs/api/java/util/function/package-summary.html`.
5. Quel problème est posé par la capture de variable locale ?