Bistromathique (Reloaded !)

Projet 2/3 pour le module de programmation fonctionnelle

Corfa Uriel uriel@corfa.fr
Giron David thor@epitech.eu

Résumé: Le projet Bistromathique (Reloaded !) a pour but la création d'une calculatrice supportant les opérations de base sur les nombres entiers, sans limite de précision.

Il se décompose en trois étapes : la gestion des calculs sur nombres infinis, la gestion des arbres représentant ces calculs, et le front-end permettant de parser des expressions arithmétiques pour les résoudre.

Table des matières

I Introduction

Au cours de votre scolarité, vous avez déjà eu à réaliser le projet `Bistromathique`. Source de nombreuses déceptions, il a su en son temps vous montrer que la piscine de C n'était pas une fin en soi. Cependant il contient bien plus, et le projet `Bistromathique (Reloaded !)` vous montrera, lui, combien la programmation fonctionnelle est adaptée à ce genre de problématique.

II Le projet

Ce projet se décompose en 3 étapes :

- La création d'un type `bigint` gérant les nombres infinis ;
- La gestion des expressions arithmétiques sous forme d'arbre ;
- Le parseur générant ces arbres, et le front-end permettant d'utiliser votre `Bistromatique (Reloaded !)` de façon interactive.

II.1 Etape 1

II.1.1 Travail à réaliser

Vous devez créer un type `bigint` représentant un nombre entier, sans limite de précision. Un exemple d'un tel type existe déjà, il s'agit du type standard `big_int`, dont la documentation est disponible à l'adresse suivante : http://caml.inria.fr/pub/docs/manual-ocaml/libref/Big_int.html.

Vous n'avez pas à recréer l'intégralité des fonctions du module. Voici la liste des fonctions obligatoires. A vous de trouver le type de ces fonctions :

- `bigint_of_string` : convertit un `string` en `bigint`. Cette fonction doit être capable de reconnaître les préfixes suivants :
  - "0x" indique que le nombre qui suit est en hexadécimal (base 16). Les caractères alphabétiques du nombre sont indifféremment en majuscules ou en minuscules.
  - "0b" indique que le nombre qui suit est en binaire (base 2).
  - "0" indique que le nombre qui suit est en octal (base 8).
  - En l'absence d'un préfixe, le nombre est en décimal.
- `string_of_bigint` : convertit un `bigint` en `string`. La sortie est en base 10.
- `string_of_bigint_base` : convertit un `bigint` en `string`. La base est indiquée par le premier argument de la fonction, qui est un variant ainsi défini :

```
1 type base = Binary | Octal | Decimal | Hexadecimal
```

- `add` additionne deux `bigints` et retourne le résultat.
- `sub` soustrait deux `bigints` et retourne le résultat.
- `mul` multiplie deux `bigints` et retourne le résultat.
- `div` divise deux `bigints` et retourne le résultat entier.
- `modulo` divise deux `bigints` et retourne le reste entier.

`Vous devez gérer les erreurs d'une façon appropriée de chacune de ces fonctions !.`

II.1.2 Consignes de rendu

Vous devez rendre votre travail dans un module nomme `Bigint`.

Vous etes encourages a rendre un jeu de tests unitaires pour votre module `Bigint`, bien que cela ne soit pas obligatoire. Celui-ci devra se trouver dans un module **séparé**.

Une attention toute particuliere sera apportee a l'interface de votre module `Bigint` en soutenance. Assurez-vous que celle-ci soit suffisante et utilisable.

II.2 Etape 2

II.2.1 Travail à réaliser

Vous devez implanter un moyen de représenter une expression arithmétique sous forme d'arbre. Votre arbre sera modélisé sous la forme d'une valeur de type `arith_expr`. Vous êtes libre de définir `arith_expr` comme bon vous semble, tant qu'il respecte les consignes suivantes :

- `arith_expr` doit utiliser votre type `bigint` pour représenter les valeurs numériques, et uniquement celui-ci car il est cense etre suffisant.
- Vous devez rendre une fonction `string_of_arith_expr` convertissant une expression arithmétique en chaîne de caractères. La représentation est libre, mais doit être lisible par un humain.
- Vous devez rendre une fonction `solve_arith_expr` résolvant une expression arithmétique et retournant le résultat sous forme de `bigint`. L'appellant de `solve_arith_expr` pourra alors afficher le resultat sur la sortie adequate dans le format voulu.

II.2.2 Consignes de rendu

Vous devez rendre cette étape dans un module `ArithExpr`. Ce module dépendra du module `Bigint` réalisé à l'étape précédente. Il devra être possible d'y substituer un autre module `Bigint` respectant les consignes ci-dessus lors de la soutenance, sans modifications du module `ArithExpr`.

Vous etes encourages a rendre un jeu de tests unitaires pour votre module `ArithExpr`, bien que cela ne soit pas obligatoire. Celui-ci devra se trouver dans un module **séparé**.

Une attention toute particuliere sera apportee a l'interface de votre module `ArithExpr` en soutenance. Assurez-vous que celle-ci soit suffisante et utilisable.

II.3 Etape 3

II.3.1 Travail à réaliser

Vous devez implanter un front-end à votre type `arith_expr`. Ce front-end doit permettre le comportement suivant :

```
./bistro [-obase (2|8|10|16)] [inputfile]
```

- Lorsqu'il est précisé, `obase` indique la base de sortie de votre `Bistromathique (Reloaded !)`. S'il est absent, la base par défaut est le décimal.
- Lorsqu'il est précisé, inputfile indique un fichier depuis lequel lire les expressions à résoudre, une expression par ligne sans fioritures. S'il est absent, lisez sur l'entrée standard une expression qui sera calculee apres l'appui sur la touche `entree`. Une fois l'expression calculee, l'utilisateur pourra entrer une nouvelle expression. S'il ne le souhaite pas, l'utilisateur doit pouvoit quitter le programme en tapant "quit" a la place d'une expression, ou bien faire un `ctrl+d`.

Indices `Votre Bistromathique (Reloaded !) doit pouvoir résoudre plusieurs expressions arithmétiques à la suite les unes des autres.`

Pour implanter le lexer et le parser, vous avez deux possibilites :

- Ecrire un lexer et un parser a la main.
- Utiliser `ocamllex` pour generer votre lexer et `menhir` pour generer votre parser.

`Dans un contexte aussi simple que celui de ce projet, notez que le temps d'apprentissage des outils sera équivalent au temps passé à réaliser un lexer et un parseur manuellement. Si vous choisissez les outils de generation, vous serez interrogés sur le contenu de vos fichiers .mll et .mly ! N'allez pas les récupérer sur Internet sans les comprendre ou bien vous serez consideres comme des tricheurs !`

Les expressions seront en notation infixe. Voici quelques exemples d'expressions valides :

```
1+2+ 3+4
0x1 + 02 + 0b11+4
1* (2+3 -4 )
(1/2)/3/4
```

Pour vous assurer que votre programme fonctionne bien, votre front-end devra en outre être capable de parser une chaîne produite par votre fonction `string_of_arith_expr` réalisée à l'étape précédente.

II.3.2 Consignes de rendu

Vous devez rendre cette étape dans un module `BistroReloaded`. Ce module dépendra des modules `Bigint` et `ArithExpr` réalisés aux étapes précédentes. Il devra être possible d'y substituer d'autres modules `Bigint` et/ou `ArithExpr` respectant les consignes ci-dessus lors de la soutenance, sans modifications du module `BistroReloaded`.

Vous etes encouragés à rendre un jeu de tests unitaires pour votre module `BistroReloaded`, bien que cela ne soit pas obligatoire. Celui-ci devra se trouver dans un module **séparé**.

Une attention toute particulière sera apportée à l'interface de votre module `BistroReloaded` en soutenance. Assurez-vous que celle-ci soit suffisante et utilisable.

Vous devez fournir un makefile compilant un éxécutable nommé `bistro` respectant l'usage indiqué ci-dessus. Il devra bien sur être possible de compiler votre programme vers du bytecode OCaml ou bien vers du code natif avec ce makefile.

III Consignes

Vous **DEVEZ** impérativement respecter les consignes suivantes :

- Tous les traits impératifs d'`OCaml` sont interdits, sauf concernant les exceptions, les séquences d'instructions quand nécessaire et les tableaux. Ceci est bien évidement limité au strict necessaire et devra faire l'objet d'une justification en soutenance. Utiliser un simple tableau pour representer un bigint est moins une bonne idee qu'il n'y parait a la premiere réflexion.
- Il est interdit que votre programme se termine à cause d'une exception non gérée.
- Vous avez le droit d'utiliser les modules de la bibliotheque standard d'OCaml dont les documentations sont disponibles aux adresses suivantes :
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Arg.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Array.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Char.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Buffer.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Filename.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Lexing.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Map.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Parsing.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Printf.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Queue.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Str.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Sys.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/Unix.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/UnixLabels.html
- Bien que le point precedent permette de deduire celui-ci, Le module standard `big_int` est interdit.
- Les étapes seront corrigées dans l'ordre du sujet. Il n'est donc pas utile de présenter une étape 2 sans étape 1.
- Chaque étape possède ses consignes de rendu propres. Tout manquement à ces consignes entrainera le refus de corriger cette étape bien que le projet ne soit pas corrige par moulinette.

- La vitesse d'éxécution ne fait pas partie des objectifs principaux de ce projet. Des performances modestes seront acceptées. Un code extrêmement rapide sera apprécié tant que son optimisation ne nuit pas à la propreté du code rendu.
- Toutes les erreurs doivent être gérées, de façon appropriée, à tous les niveaux, sans exception a cette regle.
- Il n'existe pas de norme pour `OCaml` à `Epitech`. Toutefois, tout code illisible pourra etre sanctionné arbitrairement. Une indentation claire, de l'espace entre les fonctions et le respect des 80 colonnes vous éviteront très probablement des ennuis.
- La soutenance mettra un accent tout particulier sur le contrôle de vos connaissances personnelles. Cela pourra prendre la forme d'un re-code.
- Gardez un œil sur ce sujet car il est susceptible d'être modifié.
- Les évolutions du sujet seront notifiées sur le forum du module `B4 - Programmation Fonctionnelle` de l'intranet `Epitech`. Vous devez lire ce forum régulièrement.
- Si vous pensez qu'un module qui pourrait vous etre utile manque à la liste, merci d'en faire la requete sur le forum.
- Si vous constatez des fautes ou des incoherences dans ce sujet, merci d'en notifier les auteurs afin d'apporter les corrections.
- Vous devez rendre votre projet sur le depot `SVN` mis à votre disposition par le `Koalab`. Vos dépôts seront ouverts au maximum 48h après la date de fin d'inscription au projet, intranet `Epitech` faisant foi. Il ne sera plus possible de s'inscrire au projet et donc de passer en soutenance une fois cette date depassée.
- Vos dépôts seront fermés en écriture à l'heure exacte de la fin du projet, intranet `Epitech` faisant foi.
- Seul le code présent sur votre dépôt sera évalué lors de la soutenance. La documentation relative aux dépôts fournis par le `Koalab` est fournie avec ce sujet.

Bon courage !