Compte Rendu Projet 1 Compilation Avancée « ML2C ».
AFFES Mohamed Amin « 3262731 »

Les fichiers modifiés dans ce projet sont «prod.ml » ainsi que « trans.ml ».
Les fichiers crées sont « runtime.h » et « runtime.c ».

pour exécuter effectuer les commandes suivantes :
make
./ml2c ftest.ml
make
./ftest

Remarque concernant le projet avant de détailler la façon dont j'ai traité le problème et la solution proposée, mon projet effectue les affichages demandés sauf les 2 derniers, l'avant dernier est affiché mais n'est pas correct et le dernier affichage n'est pas affiché une erreur de segmentation intervient avant peut être à cause de la donnée affiché précédente qui n'est pas correcte et ceci car l'avant dernier affichage est censé être une liste de MLfun contenant un entier dans son environnement.

Ce qui devrait être affiché :
```
<fun> [1]::<fun> [2]::<fun> [3]::<fun> [4]::<fun> [5]::<fun> [6]::<fun> [7]::<fun> [8]::<fun> [9]::<fun> [10]::[]
```
Ce que mon programme affiche :
```
<fun> [1]::2::3::4::5::6::7::8::9::10::[]
```

Donc comme vous pouvez voir mon implémentation de la fonction map ne fonctionne que pour le premier élément de la liste d'entier mais pas pour les autres ce qui rend l'avant dernier affichage de test incorrect et ce qui est peut être aussi la cause de l'erreur de segmentation qui empêche le dernier affichage car le resultat est utilisé pour la suite.

J'ai passé beaucoup de temps a essayer de débuguer sans succés.
Donc pour résumer mon programme fonctionne compile donc et affiche correctement tous les affichages sauf les deux derniers.

Début du rapport :

Comme vous pouvez voir dans le fichier « runtime.h » j'ai choisi de définir une structure MLvalue qui contient un entier « type » et un union des différents MLvalue possibles.

L'entier type sert à indiquer le type du MLvalue pour savoir à quelle contenu de notre union l'accès est possible. Pour plus de détails voir les « define » au début du fichier « runtime.h ».

L'union contient les différents types de MLvalue c'est à dire :
MLunit, MLbool, MLint, MLdouble, MLstring, MLpair, MLlist, MLfun;
J'ai fait le choix de ne pas déclarer de type MLprimitive car MLfun peux être aussi une MLprimitive ici, si c'est une MLprimitive on n'utilise que les champs name et invoke et si c'est un MLfun on utilise les champs MLcounter, invoke et MLenv bien entendu la différence se fait aussi grâce à la variable type du MLvalue.
MLPair et MLList sont aussi des structures déclarés avant le MLvalue.

Ensuite je déclare les variables globales dans « runtime.h » afin qu'elles puissent être utilisés dans le fichier généré « test.c » par « pro.ml ».

Puis les fonctions MLPrimitive_invoke_X qui correspondent aux fonctions des primitives MLX correspondantes déclarés en global.

Puis viennent les fonctions de création de chaque type de notre MLvalue en prenant en paramètre les champs dont on a besoin.

Puis la fonction MLaddenv qui sert à ajouter un élément à l'environnement d'une MLfun.

Puis pour le reste j'ai définit des fonctions équivalentes aux méthodes déclarés dans le « runtime.java » . Pour les fonctions MLequal et MLprint j'ai fait un switch pour traiter chaque cas de façon adéquate.

Les variables globales sont initialisés dans la fonction void init_runtime() se trouvant dans le fichier « runtime.c ».

Concernant le fichier « prod.ml » :

C'est ce fichier qui produit le fichier « ftest.c ».

J'ai modifié les fonctions initial_special_env et initial_trans_env pour qu'elles correspondent à mon implémentation. Pareil Pour les autres fontctions.

Ce fichier commence par produire les variables globales du fichier « ftest.c » correspondant aux variables statiques de la classe ftest du fichier « ftest.java ».

Puis pour chaque classe dans « test.java » j'ai choisi de créer une nouvelle structure puis je déclare un pointeur de cette structure ayant pour nom le nom de la classe correspondante dans « ftest.java » chacune de ces structures contient un MLfun et une variable de type int MAX.
Les fonctions invoke et invoke real sont produits de la même façon que pour « test.java » en prenant en compte les modifications faites sur les procédure dans « prod.ml » pour que ça corresponde à la nouvelle implémentation en language c.
Ensuite une fonction est défini « new_classname » pour chaque structure définissant les champs du pointeur de la structure déclarée retournant un MLvalue contenant le MLfun de cette structure puis une fonction « f_typestructure » appelant cette fonction et retournant son résultat. Chaque fois que cette fonction est appelé la variable est donc réinisialisé.

Avant le main une fonction void init() est défini affectant les valeurs de certaines des pointeurs déclarés en global au début de ftest.c.

Au début du main on appel init_runtime() et init() puis le reste du main est produit.

Concernant le fichier « trans.ml » :

Seulement deux choses en été modifiés la première appeler MLprint au lieu de MLruntime.MLprint et la deuxième pour les variables globales de « ftest.c » pour qu'elles soient appelés directement par leurs noms et non test.nomVariable.

Avec tous ces fichiers le programme « ftest.ml » est donc traduit dans le fichier « ftest.c » et affiche la quasi totalité de l'affichage demandé avec une erreur comme cité au début du compte rendu.