Rapport de rendu de la troisième partie du projet de compilation

Binôme :

AMGHAR Nassim

WAN Dong

Groupe : 1

Sommaire :

I) Introduction :

Dans ce rapport nous commencerons par vous présenter nos choix en ce qui concerne l'implémentation et l'ensemble du code généré par le compilateur à la suite de cela nous discuterons des différentes parties fonctionnelles ou non. Nous exposerons les difficultés rencontrer et nous finirons par les conclusions tirer suite aux tests effectue. Nous avons préférés commencer directement à partir de la correction proposée étant donné que notre deuxième partie contient des erreurs.

II) Implémentation :

Pour la séparation du code Ocaml, nous avons choisi de le diviser en deux partie une pour le renomage (Renamer.ml) et l'autre la plus importante pour la génération de code (Compile.ml), Nous détaillerons ces parties ci-dessous.

a) Code générer :

- Le Renomer parcourt tous l'AST (génère lors du typage) pour donner un identifiant unique à chaque variable grâce à cette fonction :

```
let var_nb = ref 0
 let rename_var id =
  incr(var_nb);{loc = id.loc ;node =
id.node^"_"^string_of_int(!var_nb)}
```

- Le Renomer renome aussi la fonction main du programme en « prog_main » afin de s'assurer qu'elle s'exécute toujours en premier grâce au code suivant :

```
if ((String.compare id.node "main") = 0 ) then
  "prog_main"
else  id.node
```

- Le programme commence toujours par une étiqueté main qui jump vers la fonction main du programme « prog_main » et implémente un code de sortie correct. Elle assure par cela le bon fonctionnement linéaire du programme.

- Le code générer n'est pas du tout optimiser, toutes les variables locales sont passées sur la pile, les variables de type Char sont enregistrer sur 4 bits sur la pile pour éviter des problèmes d'alignement. Pour éviter qu'une fonction qui retourne Void ou la fonction « main » tourne dans une boucle infinie dans le cas de l'oublie du return on a ajouté un return implicite dans le code.

- L'instruction return, enregistre le résultat dans le bon emplacement réservé par l'appel à Ecall, à la suite de quoi elle libère l'espace du block de la fonction.

b) Quelques problèmes rencontrés :

Nous avons rencontré une large palette de problèmes, certain on put être résolu et d'autre non. On peut trouver plusieurs catégories parmi les plus récurent :

- Problèmes de décalages : concerne plus le décalage des variables locales et des arguments par rapport au FP aussi les décalages à l'intérieur des structures. Nous avons plusieurs fois bloqué à cause de l'alignement c'est pour cela que nous avons préféré aligner même les Char sur la pile.
- Omission de return : dans le cas du main ou de fonction qui retourne Void nous avons dû rajouter du code qui fait un retour par défaut à la fin de ces fonction qui ne s'exécute que s'il y a omission.

c) Partie non fonctionnel

Notre compilateur ne permet pas de générer du code qui fonctionne correctement pour les programmes suivant :

- Fonction qui retourne pointeur, char, structure ou union
- Les variables déclarés dans des blocks come dans les boucles ou instructions conditionnelles

III) Récapitulatif des tests réussis et échoués

Par rapport aux tests fournis par le prof, environ 50% fonctionne dont :

- Fonction putchar
- Fonctions récursives
- Boucles si pas de variables déclarés à l'intérieur
- Operateur unaire marche sauf Etoile
- Etc.

La majorité des tests qui ne fonctionnent pas sont liées aux problèmes énoncés en haut.

IV) Conclusion :

Le travail réalisé a été repartie équitablement au sein du binôme vu que la majeure partie du travail a été réalisé en présence des deux membres du binôme. Come extensions du compilateur on aurait voulu premièrement faire fonctionner tous les programmes qui passent le typage et deuxièmement faire un meilleur alignement.