WENDLING Quentin / URBAIN Nathan – CMI IRVIJ

Rapport du projet de compilation M1

Pendant ce projet nous avons pris la décision de travailler de manière itérative en passant par plusieurs sous langages successifs (sprint sur le GitHub) en incrémentant les différentes fonctionnalités ainsi que par le développement d'un AST.
Un sprint était validé quand on passait divers tests et que nous étions certains de repartir sur des bases toujours solides pour pouvoir toujours avancer et apporter des modifications minimes et maîtrisées sur les sprints précédents nous garantissant ainsi une non-régression du code lors des sprints suivants.
L'organisation du projet se fut sur GitHub en effectuant des commits propres référent une issue (tâche) préalablement écrite pour cibler le travail à faire et donner la répartition des tâches.

Les différentes décompositions de sprint furent divisées comme ceci:
- Sprint 0 : Mise en place des fonctions de l'AST et Table de symbole
- Sprint 1 : Printi, Printf , main() , return , commentaire
- Sprint 2 : Déclaration/Affectation variable Int, expressions arithmétiques demandées
- Sprint 3 : Structure de contrôle
- Sprint 4 : Boucle
- Sprint en parallèle : Define
- Sprint 5 : Tableau
- Sprint 6 : Stencil
- Sprint 7 : Fonctions simples et récursives

Notre compilateur passe par un arbre de syntaxe abstraite qui passera ensuite par une analyse sémantique afin de détecter certaines erreurs du type sémantique présent dans le fichier donné. Par exemple, il y aura une erreur si une variable est affectée sans déclaration, un tableau mal initialisé, une fonction appelée avec un mauvais nombre d'arguments.
L'AST sera ensuite traduite en Code intermédiaire sous forme de Quad.

Ce que notre compilateur sait compiler:

- Une fonction main sans argument avec return entier (ou variable, ou expression)
- Les déclaration des define et leurs utilisations
- Les déclarations de variable Int et leurs affectations par une expression ou constante
  id ayant une regex comme suit: [a-zA-Z_][a-zA-Z0-9_]*
- Les expressions arithmétiques avec constante ou variable :
  + - * / ++ -- et le - unaire
- Les structures de contrôle :
  et (&&), not (!), ou (||), inférieure/supérieur égale ou non (< ≤ > ≥)
  if(condition){instructions} else{Instructions}
  if(condition){instructions}
- Les boucles :
  while(condition){instructions},
  for(liste d'initalisation ; condition ; liste d'auto-incrementation) {instructions}
- Les déclarations de tableau : id[int][int]… de dimension n
- Affectation par liste lors de la déclaration de tableau : int tab[4] = {1,2,3,4}
- Utilisation des tableaux dans les expressions arithmétiques ou conditions :
  4 + tab[i][j] , tab[i] == 4
- L'affichage de string : printf(« text »)
- L'affichage d'entier avec variable,case de tableau ou constante :
  printi(id),printi(42), printi(tab[2])
- Commentaire sur 1 ou plusieurs lignes avec // ou /* */
- Déclaration et affectation de stencil
  stencil test{1,2} = {{4,6,3},{4,6,3},{4,6,3}};
- Opérateur de stencil :
  int testApply = test$testTab[1][1][3]; / int testApply2 = testTab[1][1][3]$test;
- Fonctions avec une liste d'arguments et un return obligatoire (pouvant être récursive)
  int fctTest(int id1, int id2) {instructions}

Un MakeFile est disponible est possède plusieurs commandes :
- make : Compile le projet
- make test : Compile le projet et lance les jeux de test automatiquement et annonce s'ils ont compilé ou non
- make clean : nettoie le répertoire
- make help : donne les commandes disponibles