|  |  |
| --- | --- |
|  |  |
| COMPILATEUR MINIC VERS DOT |  |
|  |  |
|  | 16/05/2021COMPILATION |
|  | Antoine COUSSONMargaux SCHMIED |

**Introduction:**

Il doit comporter une introduction qui décrit votre travail en général, ce que vous avez réussi à faire et vos échecs.

Lors de ce semestre nous avons travailler sur ce projet qui est la création d’un mini compilateur C vers un langage de graph,

Pour ce faire nous somme parti de l’archive contenant une base du fichier yacc nommé minic.y, ainsi que du fichier Lex ANSI-l.l, nous avons réussi à générer des dot de fichier minic valide et a soulevé des erreurs de déclaration de variables et de fonctions.

### Lex:

Nous avons tout d’abord pris connaissance de ce que le miniC permettait et interdisait et des token présent dans le yacc. Nous avons ensuite analysé les expressions régulières pour les reconnaître et renvoyer le token correspondant.

Nous avons dû aussi repérer les caractères qui n’étaient pas pris en charge par le lex d’origine et les renvoyer.

Puis absorber le contenu lexicalement juste mais inutile au code (ex : les commentaires)

Ainsi que garder un indice représentant le numéro de la ligne pour la détection d’erreurs.

Pour finir, s’il reste des caractères non reconnus nous soulevons une erreur de type lexical.

### Yacc:

En premier lieu nous avons créé un union comportant : “id” une chaîne de caractère que nous utilisons pour transmettre les valeurs au token, “tree” une structure de donnée qui représente des fonctions et “symbole” leur contenue et une seconde structure de donnée qui représente les déclarations.

Une fois l’union créée nous avons assigné le type “id” a tous nos token puis avons assigné aux production le type adéquat.

Une fonction soulevant les erreurs syntaxiques a été créé puis nous sommes passé à la création de nos arbres.

Tout d'abord nous avons modifié les grammaires contenant des erreurs, comme liste Expression et liste params, ces erreurs avaient été mise en en évidence par un étudiant sur le forum,

Nous avons ensuite découvert un problème dans les priorités des expressions que nous n’avons pas réussi à résoudre. Pour palier à ce problème nous avons réécrit les productions en la rendant certes redondante mais fonctionnelle.

Pour chaque dérivation nous créons le nœud de l’arbre en l’associant à ses fils ainsi que ses suivants et en lui assignant les attributs appropriés.

À la création d’un nœud de l’arbre nous lui assignons également un nœud de la table des symboles comportant les variable, tableau ou fonction déclaré à cet instant ainsi que ces déclaration fils et suivant.

Lors d’une déclaration nous créons un nœud spécial qui est le nœud de la table des symboles, il sera attribué au nœud où cela se produit et ses fils en hériteront,

Ce nœud sert à savoir ce qui a été défini et

### C:

Pour pouvoir réaliser les actions sémantiques nous avons utilisé des fonctions définies dans un fichier .c et un .h

Nos structures y sont définies ainsi que des fonctions pour les initialiser,

Par la suite nous avons implémenté des parcours de nos structurent qui nous permettent d’analyser le code traduit et de soulever les erreurs s’il y en a,

Une fois les vérifications faites nous écrivons un fichier dot en traduisant la structure principale, nous avons dans chaque nœud de notre arbre l’information de son nom, de son type, de sa nature (nous permet de changer la forme du nœud sur le dot si par exemple c’est un appel de fonction ou autre), une fois tous les nœuds défini dans le dot, nous relions chaque pères à ses fils

une fois fait nous fermons le fichier

Une des grandes difficultés de ce projet pour moi était le C je ne suis pas trop familier avec ce langage et il m’arrivait de chercher d’où provenait un segfault pendant un certain temps

### Organisation et partage du travail

Pour la répartition du travail nous avons eu des difficultés à séparer le travail pour réaliser des tâches distinctes, nous avons codé la majorité du temps en live share (connecté sur la même machine) en réfléchissant aux problèmes ensemble et en testant, à tour de rôle des solutions ou en imaginant des algorithmes, nous avons utilisé au cours du projet le logiciel de versionning Git, cela nous a permis de nous coordonner sur la dernière version fonctionnelle en date du projet.

### Aide extérieure et travail de recherche :

Pas d’aide direct juste des discussions entre camarade ou des topics déjà existant sur internet

Nous avons principalement cherché des informations sur le lex et le yacc dans les documents fournis par le prof ainsi que dans le livre *Compilers Principles Techniques and Tools,* nous nous sommes aussi aidé de forums où nous pouvions trouver des conseils sur le yacc ainsi que des explications sur son fonctionnement, stackOverflow était une bonne ressource, nous consultons des topic déjà existant, nos questions avaient quasiment déjà toutes été posé.

Pour Lex nous nous sommes aussi aidé des td ainsi que des ressources sur moodle.

Pour tester les expressions régulières faites pour les commentaires par exemple nous utilisions des sites nous permettant de les tester en temps réel tels que [regexr.com](https://regexr.com/) pour n’en citer qu’un.

Pour des interrogations sur le C nous cherchions principalement sur internet notamment des conseils sur l’allocation mémoire et les pointeurs, la définition de structures et autres…

Encore une fois StackOverflow nous as grandement aider, ainsi que des sites comme [www.developpez.com](http://www.developpez.com)

### Difficultés rencontrées :

N'étant pas familier avec le lex et la yacc il nous a fallu un certain temps pour bien appréhender le travail à fournir, ce temps aurait été très apprécié sur la fin de la réalisation du projet.

Trouvé une bonne structure pour la génération du dot et la table de symbole n’a pas était simple nous avons dû nous y prendre a plusieurs reprise.

L’un de nous travaillant sur mac nous avons eu beaucoup de problème de segmentation fault qui on était réglé par l’utilisation d’un machine virtuel linux.

Malgré nos efforts nous n’avons pas réussi, avec la grammaire fournie, à avoir une priorité des opérations correctes c'est pourquoi nous avons dû la modifier en la rendant plus redondante mais fonctionnelle.

### Solutions proposées :

Nous nous sommes appuyés sur l’exemple du dot pour avoir une idée de ce que l’arbre a généré devait être, nous avons donc imaginé une structure permettant de relier des nœuds les uns les autres, ces nœuds comportent un nom, une indication de leur nature (ex appel, if …)

Nous avons donc choisi de faire une structure se rapprochant d’une liste chaînée de listes chaînées chaque nœud possède 1 fils et 1 suivant, son suivant est son frère il représente un maillon de la liste chainé des fils du père,

son fils est le premier de ses fils, c’est une structure récursive qui contient également la table  des symboles se référant à son niveau dans le code, la structure de la table des symbole est elle aussi sous forme d’arbre, nous avons choisis cette représentation car nous la trouvions plus clair pour nous, une des différence entre la structure pour les symbole et la structure principale est que la table des symbole à connaissance de son père c’est une liste doublement chaînées de listes doublement chaînées cela nous permet de vérifier si par exemple une variable est défini dans son bloc et si elle ne l’est pas alors on regarde récursivement si elle est présente dans son père et ainsi de suite si nous ne passons pas par une définition alors cela veut dire que nous ne l’avons pas défini, cette façon de parcourir nous empêche de regarder des symbole défini dans des lieu plus précis du code par exemple si dans un bloc nous appelons un identificateur qui a été défini dans uniquement dans un if, ce dernier étant le frère de l’instruction nous ne passerons pas par lui pour vérifier sa définition empêchant les problèmes laissant passer de mauvaises définitions.

### Conclusion :

Il a été très dur de démarrer le projet, c’est une tâche conséquente et nous avons pendant longtemps fait du sur place au début, nous ne comprenions pas comment faire, au fur et à mesure des tests nous avons commencé à comprendre comment toutes les parties s'agençaient.

Une fois le fonctionnement du yacc pris en mains nous n'avions plus qu'à programmer en C, nous somme aller très vite sur le dernier mois du projet.

 En effet pour les vérifications nous n’avions plus qu'à parcourir et vérifier s’il n’y avait pas d’erreurs sémantique dans le code, ces erreurs nous ne les avons pas trouvés énoncés clairement dans le sujet donc nous avons dû faire des tests sur du C pour voir ce qui provoquait des erreurs, nous documenter sur ce qui était permis ou pas, nous avons cherché dans la documentation du C, poser des questions sur le forum Moodle.

En fin de compte même si notre compilateur n’est pas parfait nous avons le sentiment d’avoir beaucoup appris en “mettant les mains dans le cambouis” peut être plus qu’avec seulement des cours, la motivation à faire fonctionner le compilateur était notre moteur durant le semestre et nous avons souvent été frustré de ne rien faire marcher pendant plusieurs jours, mais quand tout marchait nous étions ravis.