Sommaine:

1° Présentation

L'objectif de ce projet est de générer un programme capable de convertir un programme Pascal en un programme C, on procédant tout d'abord par la vérification de la syntaxe et de la sémantique ainsi ensuite la traduction en C uniquement si l'étape précédente a réussi.
L'analyseur Lexical devra être codé pour Lex et l'analyseur grammatical pour yacc.

2° Contrainte

L'analyseur lexical devra être capable d'identifier les mots-clés utilisé en Pascal, les valeurs littérales ainsi que les identifiant. Ces identifiants devront être passés à un analyseur grammatical qui les stockera ensuite dans une table des symboles. L'analyseur grammatical devra décrire la grammaire du langages Pascal. Si le programme Pascal a été correctement reconnu et ne contient aucune erreur sémantique, il faudra le traduire en un programme C.
L'analyseur grammatical composera la plus grande contrainte de ce projet.

3° Organisation et environnement de développement

Concernant la répartition des taches, il a été difficile de bien partager le travail étant donné que nous avons avancé dans le projet à deux étant donné la complexité du projet.

Pour l'environnement de développement, l'ensemble du projet est codé dans un environnement Linux. Les machines utilisent un environnement Ubuntu. De plus, nous avons utilisé un systéme de subvertion pour le projet à l'aide de Git et de Github. Il nous a paru essentiel d'avoir ce type d'outils lorsqu'on travaille à plusieurs sur un projet.

4° Organisation de l'application:

a) L'analyseur lexical: synthawe.l

Le fichier synthaxe.l codé pour Lex a pour tache d'identifier les mots-clés utilisé en Pascal, les valeurs littérals ainsi que les identifiants. Voici une liste de tout les mots-clés reconnu par l'analyseur lexical:

PROCEDURE, PROGRAM, FUNCTION, STRING, INTEGER, TRUE, FALSE, WRITELN, WRITE, READLN, AND, ARRAY, CASE, CONST, DIV, DO, ELSE, END, FOR, IF, MOD, NOT, OF, OR, BEGIN, THEN, TO, DOWNTO, VAR, READLN, WHILE, WITH,RANDOMIZE, RANDM, SQRT, REPEATN, UNTIL

Notre programme est également capable de reconnaître les identifiants contenu sous la forme [a-zA-Z]([a-zA-Z0-9])*, les nombres entiers, les nombres réels et les opérateurs suivant { := , : , , , . , = ,>= , > , <s= , (, <, - , <> , + ,] ,) , ; , / , * , ** , -> , ^}

Une fois chaque mots-clés, identifiants, nombre entiers, réels et opérateur reconnu on les transmets à l'analyseur grammatical par l'intermédiaire des tokens.

b) L'analyseur grammatical: grammaire.y

c) La table dalide et ce symbole: table.h

5° Exécutions

Nous avons accompagné notre application avec plusieurs fichiers test écrit en Pascal, compilé au préalable pour vérifier la validité des programmes. Tout ces fichiers tests via l'application nous génères des programmes .C auquel nous avons testé la validité on les passant au compilateur. Ces fichiers test fonctionne car elle se limité aux fonctionnalités que nous avons mis en place pour la gestion d'une sous partie du lanage Pascal.

Donner un exemple d'un programme pascal et du programme c généré

7° Difficultés rencontrés

8° Conclusion