Manuel d’utilisation de l’Interpreter.

I – présentation du fichier Lex

II - L’ensemble des expressions régulières reconnue

III – Présentation du fichier Yacc

IV – Utilisation de mon interpreter

I : Le fichier Lex :

Le fichier Lex est utilisé pour définir un analyseur lexical, qui est l'un des composants d'un interpreter. L'analyseur lexical est responsable de lire le code source du programme et de le décomposer en "tokens", qui sont des unités de syntaxe du langage de programmation.

Dans mon fichier Lex, je définis plusieurs expressions régulières qui correspondent à différents types de tokens. Par exemple, l'expression régulière "set" pour correspondre au mot clé "Set", ou encore l'expression régulière "identifier" pour correspondre aux identifiants de variables.

Pour chaque expression régulière j’indique une action à effectuer lorsque cette expression est reconnue par l'analyseur lexical. Par exemple, pour l'expression régulière "set", l'analyseur lexical doit renvoyer le token "SET" au fichier Yacc.

II - L’ensemble des expressions régulières reconnue :

Déclaration des types :

boolT: [bB][oO][oO][lL]

intT: [iI][nN][tT]

float: [fF][lL][oO][aA][tT]

Les valeurs correspondantes au type :

false: [fF][aA][lL][sS][eE]

true: [tT][rR][uU][eE]

chiffre: [0-9]

entier: {chiffre}+

decimal: {entier}\.{entier}

Les opérateurs mathématiques :

"+" , "-" , "\*" , "/"

Les opérateurs logiques :

or: "OR"

and: "AND"

xor: "XOR"

not: "NOT"

Les mots clés :

set: "Set"

print: "print"

sin: "sin"

cos: "cos"

Nom de variable :

identifier: [a-zA-Z]([a-zA-Z0-9]|"?"|"$")\*

Autre :

blancs : [ \t]+

commentaire : #.\*

"\n", "(" , ")" , "=" , ";"

III Le fichier Yacc :

Le fichier Yacc (Yet Another Compiler Compiler) est utilisé pour définir un analyseur syntaxique, qui est le second élément d’un interpreter. L'analyseur syntaxique est responsable de vérifier la syntaxe du programme et de construire un arbre de syntaxe abstraite (AST) qui représente la structure du programme.

Dans mon fichier Yacc, j’ai défini les règles de grammaire du langage de programmation qui décrivent la structure des différentes parties du programme (déclaration de variable avec ou sans valeur, affectation d’une variable, afficher le contenu d’une variable). Pour chaque règle de grammaire, j’ai ensuite indiqué une action à effectuer. Par exemple pour afficher le contenu d’une variable il faudra reconnaitre l’ensemble de token PRINT suivi de IDENTIFIER cette action déclenchera alors l’appel à la fonction « print\_variable » :

PRINT IDENTIFIER {print\_variable($2);}

IV - – Utilisation de mon interpréter

Pour utiliser mon Interpréter il vous faudra tout d’abord compiler mon fichier Yacc et mon fichier Lex ensemble, la compilation par commande semble défectueuse de mon côté je me demande encore pourquoi mais une compilation par l’éditeur « Flex Windows » nous génère bien un .exe.

Mon programme est capable d’être lancé avec un fichier en paramètre de la façon suivante :

>$ interpreterTimotheeSauvageB3AL < monFichierTest.txt

Pour l’ensemble des règles de mon programme :

Le première règles obligatoire est tout d’abord est que chacune de votre expression doit finir par un « ; » peu importe ce qu’elle fait. La deuxième est si vous compiler avec un fichier en paramètre vous devrez rajouter le terme « FIN » tout à la fin de votre fichier.

Maintenant passons aux règles concernant les expressions, une déclaration se fera toujours en commençant par Set suivi du type de la variable ainsi que son nom, ensuite vous pouvez lui affecter une valeur ou juste la déclarer, (la valeur par défaut sera alors 0 par défaut). Voici quelques exemples :

Set int test ;

Set float test2 ;

Set BoOl test3 ;

Vous pouvez vérifier le résultat en faisant :

Print test ;

Pour déclarer et affecter il vous suffira de rajouter l’opérateur « = » ainsi que la valeur.

La valeur peut être une expression et doit correspondre au type déclarer.

Set int test = 1 ;

Set FloAt test2 = 1.2+1.2;

Set BoOl test3 = NOT True;

Vous pouvez aussi affecter des Valeur aux variables déjà déclarer, cependant il vous faut faire gaffe au type.

(Variable « test » déclarée au-dessus)

test = 2 ; (OK CAR INT DANS INT)

test = 2.3 (PAS OK CAR FLOAT DANS INT)

Quant a lui le Boolean est considérer comme un int (0 ou 1) mais ne peux pas être affecter ou additionner avec un entier.

Les opérateur Logiques comme mathématique répondent aux règles de priorité dans les calculs. Il existe aussi les fonctions sin et cos qui vous permettent de calculer la valeur d’un sinus ou d’un cosinus

Set InT test = 2 \* sin(3) ;

Set FLOAT test2 = 2.2 \* (2.2 + 1.1) ;

Les cas d’erreurs, malheureusement je sais que mon programme n’est pas parfait et le token FIN en sera la preuve, mais je peux en rajouter, la division par 0 est reconnue comme une erreur mais affecte quand même le premier terme ( a = 12/0 => a = 12 + erreur), j’ai aussi un soucis sur mes Float qui n’accepterais pas une expression du type : 1.1 + 1, qui donnerais un Boolean mais nous devront l’écrire : 1.1 + 1.0.