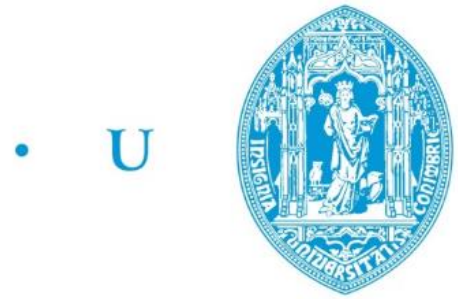

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Compiladores

Projeto Final

Carlos Miguel da Luz Lima – 2017266922
Gabriel Nunes Saraiva - 2017256436

1. Gramática Reescrita.

Para construirmos a gramática fomos seguindo a estrutura fornecida no enunciado, alterando quando fosse necessário.
Nos casos onde havia uma expressão que se podia repetir zero ou mais vezes, foi feita uma lista à parte que se ia repetindo, por exemplo, no caso da produção:
- **Statement -> LBRACE { Statement } RBRACE** foi transformada em duas produções:
- Statement -> LBRACE List RBRACE
- List -> List Statement | ε
Nos casos em que uma parte podia ser opcional, por exemplo:
Statement -> IF LPAR Expr RPAR Statement [ELSE Statement]
Foi transformada na seguinte produção:
Statement -> IF LPAR Expr RPAR Statement
| IF LPAR Expr RPAR Statement ELSE Statement
Para além destes casos foi necessário criar uma produção auxiliar para tratar de situações do assign:
Expr_extra -> Assignment | Expr em que Expr serve para expressões que não são Assignment, como, por exemplo, **Expr -> Expr + Expr.** Esta divisão serve para solucionar problemas de Shift/Reduce.
Para evitar vários conflitos Shift/Reduce foi necessário definir precedências e associatividades que o Yacc permite fazer com %left e %right.
Para o analisador recuperar localmente de erros de sintaxe foi necessário incluir regras de erro na gramática, tais como:
FieldDecl->PUBLIC STATIC Type ID CommaIDList
| error SEMICOLON

2. Algoritmos e Estruturas de Dados da AST e da Tabela de Símbolos.

1. AST

Para construir a árvore criamos uma estrutura que guarda um type e um token, sendo o type o tipo do nó em questão e o token o seu nome definido no código. Para além disso a estrutura guarda também dois ponteiros: um a apontar para os irmãos e o outro para os filhos.

Para podermos manipular a árvore construímos as seguintes funções:

- Create_node(char*, char*, int, int) recebe um type e um token e cria um novo nó. Os dois parâmetros int, correspondem à linha e coluna em que estaria o nó, contudo, para a tabela de símbolos não era necessário.
- add_child(node_type*, node_type*) recebe dois nós, um pai e um filho e cria uma ligação entre eles.
- add_sibling(node_type*, node_type*) recebe dois nós e cria uma relação de irmãos entre eles.
- is_null_node(node_type*) que recebe um nó e verifica se o type do nó é NULL.
- print_ast(node_type*, int) serve para imprimir a árvore, começando pelo nó que recebe por parâmetro, sendo este a root.
- clear_ast(node_type*) que liberta a memoria toda ocupada pela árvore.
- number_siblings(node_type*) que recebe um nó e devolve o seu número de irmãos.
- verifica_no(node_type*) e verifica_no2(node_type*) foram usadas para saber o type de um nó era Null1 ou Null2 respetivamente. Foram necessárias estas funções para lidar com o Semicolon, pois de outra maneira dava-nos erro.

2. Tabela de Símbolos

Para a Tabela de Símbolos criamos uma função que corre todos os nós da árvore e que chamava uma função específica consoante o type do nó.

Para facilitar a impressão da tabela de símbolos, criamos uma estrutura de lista ligadas. Utilizamos três listas globais desse tipo, onde na variável global *nome_funcoes* guardamos o nome de todas as funções e o tipo da função, no variável *var_locais* todas as variáveis locais de uma função e o seu tipo e na variável *lista* todos as variáveis que eram parâmetro de entrada de uma função e o seu tipo.