Rui Pedro Dias Barata

2015238609

rpbarata@student.dei.uc.pt

---

## I. Gramática re-escrita

Fonte: https://www.lysator.liu.se/c/ANSI-C-grammar-y.html

Program → FunctionsAndDeclarations
FunctionsAndDeclarations → FunctionsAndDeclarationsMandatory
FunctionsAndDeclarationsZeroOrMore
FunctionsAndDeclarationsMandatory → FunctionDefinition | FunctionDeclaration | Declaration
FunctionsAndDeclarationsZeroOrMore → FunctionsAndDeclarationsMandatory
FunctionsAndDeclarationsZeroOrMore | $
FunctionDefinition → TypeSpec FunctionDeclarator FunctionBody
FunctionBody → LBRACE DeclarationsAndStatements RBRACE | LBRACE RBRACE
DeclarationsAndStatements → DeclarationsAndStatements Statement | DeclarationsAndStatements
Declaration | Statement | Declaration
FunctionDeclaration → TypeSpec FunctionDeclarator SEMI
FunctionDeclarator → ID LPAR ParameterList RPAR
ParameterList → ParameterDeclaration | ParameterList COMMA ParameterDeclaration
ParameterDeclaration → TypeSpec ID | TypeSpec
Declaration → TypeSpec DeclaratorList SEMI | error SEMI
DeclaratorList → Declarator DeclaratorZeroOrMore
DeclaratorZeroOrMore → COMMA DeclaratorList | $
TypeSpec → CHAR | INT | VOID | SHORT | DOUBLE
Declarator → ID ASSIGN Expr | ID
Statement → SEMI | Expr SEMI | LBRACE RBRACE | LBRACE StatementZeroOrMore RBRACE |
LBRACE error RBRACE | IF LPAR Expr RPAR StatementOrError ELSE StatementOrError | IF LPAR
Expr RPAR StatementOrError NOELSE | WHILE LPAR Expr RPAR StatementOrError | RETURN Expr
SEMI | RETURN SEMI
StatementZeroOrMore → StatementZeroOrMore StatementOrError | StatementOrError
StatementOrError → Statement | error SEMI
Expr → ExprA | Expr COMMA ExprA
ExprA → ExprB | ExprB ASSIGN ExprA
ExprB → ExprC | ExprB OR ExprC
ExprC → ExprD | ExprC AND ExprD
ExprD → ExprE | ExprD BITWISEOR ExprE

ExprE → ExprF | ExprE BITWISEXOR ExprF
ExprF → ExprG | ExprF BITWISEAND ExprG
ExprG → ExprH | ExprG EQ ExprH | ExprG NE ExprH
ExprH → ExprI | ExprH LT ExprI | ExprH GT ExprI | ExprH LE ExprI | ExprH GE ExprI
ExprI → ExprJ | ExprI PLUS ExprJ | ExprI MINUS ExprJ
ExprJ → ExprK | ExprJ MUL ExprK | ExprJ DIV ExprK | ExprJ MOD ExprK
ExprK → ExprM | PLUS ExprK | MINUS ExprK | NOT ExprK
ExprL → ExprA | ExprL COMMA ExprA
ExprM → ExprN | ID LPAR RPAR | ID LPAR ExprL RPAR | ID LPAR error RPAR
ExprN → ID | INTLIT | CHRLIT | REALLIT | LPAR Expr RPAR | LPAR error RPAR

### II. Algoritmos e estruturas de dados da AST e da tabela de símbolos

Para representar a AST, criou-se uma "lista" de *Nodes*. Cada nó desta lista aponta para um dos seus filhos (*child*) e para uma lista de irmãos (*brother*). À medida que a análise sintática ocorre, esta estrutura é preenchida, adicionando *childs* ou *brothers* aos nós já criados anteriormente.

Foi definida a lista *globalTable* que guarda os símbolos da tabela global. Cada nó desta estrutura tem um ponteiro para uma lista de símbolos de cada função. Conseguimos assim construir a tabela de símbolos, iterando sobre estas estruturas.

No ficheiro *structs.h* temos então as seguintes estruturas para ajudar a construir estes algoritmos:

- *Node*: representa um nó na AST;
- *TableList*: lista de símbolos;
- *SymList*: lista de simbolos usados para construir a *TableList*;
- *ArgList*: lista de argumentos

## III. Geração de código

A geração de código não foi acabada.