Compiladores

Relatório do projecto prático

Trabalho realizado por:
Gonçalo Filipe Lucas Menino Rodrigues Pinto nº2014202176
Gabriel Monteiro Maia Pereira nº2014204763

1. Gramática

Na forma que nos foi disponibilizada a gramática era ambígua pelo que foi necessário proceder à realização de alterações para retirar a ambiguidade da mesma. Uma destas alterações foi especificar a associatividade dos operadores usando as keywords %left, %right e %nonassoc. No exemplo a seguir podemos observar uma produção ambígua:

S -> 1 + 2 + 3

Se não especificarmos a associação do operador '+' esta produção irá tornar-se ambígua visto que a gramática não consegue perceber se primeiro deve realizar a adição dos dois operandos à esquerda ou dos dos dois operandos à direita. Assim, se associarmos o operador '+' à esquerda (%left) a produção ficará da seguinte forma:

S -> (1 + 2) + 3

Se por outra forma especificarmos a associação do operador '+' à direita (%right), a produção ficaria então da seguinte forma:

S -> 1 + (2 + 3)

Se não pretendermos que o operador não seja associado nem à direita nem à esquerda então fazemos recurso da keyword %nonassoc.

Embora seja necessário especificar a associação dos diferentes operadores, isto por si só não é suficiente para desambiguar a gramática, é necessário também fazer a distinção da prioridade dos diferentes operadores. Assim sendo, a ordem pela qual a associação dos operadores é apresentada é a ordem de prioridade dos mesmos operadores mas na forma inversa, ou seja, o primeiro operador não é o operador que apresenta maior prioridade mas sim o operador que apresenta menor prioridade. A ordem de prioridade de associatividade de operadores presente no nosso projecto apresenta-se em baixo apresentada.

```
%left COMMA
%right ASSIGN
%left OR
%left AND
%left EQ NE LT LE GE GT
%left PLUS MINUS
%left STAR DIV MOD
%right NOT
%left LPAR RPAR LSQ RSQ
%nonassoc spec
```

2. Estruturas de dados utilizadas

%union

```
%union{
    struct Token* token;
    struct node_ast* node;
}
```

Estrutura utilizada para fazer a respectiva passagem de dados entre o lex e o yacc.

- **Token* token:** token no qual irá ser armazenado os dados recebidos pelo lex
- **node_ast*:** nó remetente para a AST

Token

Estrutura utilizada para guardar os dados recolhidos por parte do lex.

```
typedef struct Token {
  char* id;
  int line;
  int column;
} Token;
```

- **char* id:** id do token
- **int line:** linha na qual se encontra o token representado
- **int column:** coluna na qual se encontra o token representado

As variáveis int line e int column são utilizadas para guardar a posição inicial do respectivo token e utilizadas para posterior impressão de erros semânticos.

node_ast

Estrutura utilizada para a criação dos nós que se encontram na árvore AST.

```
typedef struct node_ast{
    struct node_ast* pai;
    struct node_ast* irmao;
    struct node_ast* filho;
    Token* id;
    char value[100];
    int num_params;
    char* s_type;
    int is_function;
    char** params;
}node;
```

- **node_ast* pai:** ponteiro para o nó node_ast pai
- **node_ast* irmao:** ponteiro para o primeiro nó node_ast da lista ligada de irmãos
- **node_ast* filho:** ponteiro para o primeiro nó node_ast da ligada de filhos
- **Token* id:** ponteiro para o Token correspondente do nó
- **char value[100]:** valor do node_ast caso seja um nó terminal em que possua valor
- **int num_params:** número de parâmetros que o nó tem caso seja uma função
- **char* s_type:** tipo de nó utilizado para a anotação da árvore AST anotada
- **int is function:** variável que indica se o nó representado é uma função
- **char** params:** variável que armazena parâmetros no caso de o nó representar um função

Symbol_Type

Estrutura de Enumeração que serve para indicar o tipo do símbolo que irá ser inserido na tabela de símbolos.

```
typedef enum{FuncDecl,FuncTable,Global,Variable,Return}symbol_type;
```

Symbol

```
typedef struct table_elem{
    char* id;
    char* type;
    int num_params;
    int num_func;
    symbol_type s_type;
    int is_param;
    struct table_elem** params;
    struct table_elem* next;
}Symbol;
```

- **char* id:** id do símbolo da tabela
- **char* type:** tipo do símbolo da tabela (undef em caso de erro cujo tipo de resultado seja desconhecido)
- **int num_params:** número de parâmetros que o símbolo apresenta, no caso de este representar uma função
- **int num_func:** número da função no qual o símbolo se encontra
- **symbol_tye s_type:** tipo de símbolo que irá ser inserido na tabela de símbolos
- **int is_param:** variável que indica se o símbolo representa um parâmetro de uma função
- **struct table_elem** params:** conjunto de parâmetros que o símbolo apresenta, no caso de este ser uma função
- **struct table_ele* next:** ponteiro para o próximo elemento da tabela de símbolos