Compilador para Linguagem Pascal Simplificado

Gabriel de Oliveira Almeida
Gustavo Lopes Santana
João Victor Silva Menezes
FCT - UNESP

O trabalho consiste na implementação de um compilador completo, desde a implementação do analisador léxico, sintático e semântico; gerador de código; e até o interpretador para linguagem Pascal Simplificado (LALG) durante a disciplina de Compiladores em 2019.

Analisador Léxico

O analisador léxico foi responsável por montar a tabela de Tokens, em que através de uma classe enum, o qual, é capaz de reconhecer todas as palavras reservadas, números inteiros e flutuante, operadores, delimitadores e todos os caracteres presentes na linguagem, se não reconhecido deve ocorrer erro Léxico no código. A formação dos tokens é dada pela leitura de caracter a caracter presente no código fonte, unindo-os quando há espaços, tabulações, pula linha e após há a verificação na tabela para que faça sua classificação, e para isso utiliza-se o expressões regulares.

Analisador Sintático

O analisador sintático está responsável pela verificação da ordem em que os tokens são introduzidos, de modo a garantir que não haja incompatibilidade na gramática. É executado após ocorrer toda a análise sintática, percorrendo a tabela de tokens gerada no passo anterior e verificando iterativamente a gramática, em que cada derivação da gramática há um método correspondente, o controle de repetição de zero ou mais vezes, dada na EBNF por {α}, sendo α a cadeia, é dada pela estrutura de repetição while, evitando a recursão na execução do programa. Os erros sintáticos deixa explícito qual token deve assumir o lugar do token inesperado, por isso utiliza-se os mesmo classificadores presente na análise léxica. Vale expor que não houve alteração de sintaxe da LALG, respeita toda a EBNF.

Quando erros sintáticos são encontrados, ocorre o descarte de tokens, até que se encontre um token “seguro” que possibilite retomar a análise sintática.

Analisador Semântico

O analisador semântico é apenas executado, se somente se não for verificado erros sintáticos no código fonte, uma vez que além da tabela de símbolos , também é responsável pela geração do código intermediário, assim erros sintáticos não são tolerados, já que pode ocorrer a expansão dos erros e afetar o desempenho do programa.

Este possui estrutura semelhante ao analisador sintático, possui uma função para cada derivação na gramática, porém nele cabe apenas verificar a semântica da gramática,

uma vez que a sintaxe já está correta, utilizando a tabela de símbolos para auxiliar a análise, o cabeçalho da tabela é dado por

Lexema		Categoria	Escopo	Tipo	Endereço	Utilizada
lexema	token					

Tabela 1. Cabeçalho da tabela.

A cada símbolo lido, procura-se na tabela de símbolos se já foi declarado ou até mesmo se já foi inicializado, ou seja, utilizado um vez anteriormente. Desse modo, evita-se conflito ao declarar variáveis iguais em mesmo escopo, utilizar variáveis não declaradas, ou até mesmo usar variáveis que não foi inicializada, isto é, não possua valor, tal como no código a seguir:

```
int x,y;
real x; //Erro semântico, variável já declarada.
begin
      y:= (x + y) + z;
      //Erro semântico, y não possui valor.
      //Erro semântico, z não foi declarada.
end.
```

Código 1. Exemplo de erros semânticos.

O mesmo tipo de erro é feito para procedimentos, não valendo apenas para as variáveis. Outro erro que pode ser detectado pelo o analisador semântico é a incompatibilidade entre os tipos de dados em uma atribuição, por exemplo, caso faça o seja atribuído um valor real a uma variável inteira, ou até mesmo se um booleano seja inserido em variáveis não booleanas, assim como os erros citados anteriormente e esse são feitos verificando a tabela, uma vez que na tabela como mostrado na Tabela 1, possui informações sobre determinado símbolo, bastando apenas realizar comparações de tipos, utilizado e até mesmo da coluna escopo entre os símbolos para verificar se há anomalias no código fonte lido.

Erros de divisão ocorre apenas se atribuído em uma variável inteira ou booleana, porém se for real, mesmo que seja divisão entre inteiros, não deve ter problemas, já que o resultado gerado é real, logo aceita-se um real receber determinado resultado.

Vale lembrar que a função “read()” e “write()” são os únicos procedimentos que possuem tratamento separado dos demais procedimentos, uma vez que são funções reservadas da linguagem, que servem para interagir com o usuário, entrada ou saída de valores na tela do usuário.

O código intermediário é gerado durante a análise semântica, enquanto realiza suas verificações, se correto, salva-se os comandos em um ArrayList, porém se encontrar erros na gramática, afetará também o código intermediário.

Para resolver expressões numéricas, tal como

(a * b)+(a * b)+c

O algoritmo transforma essa expressão posfixa em notação polonesa reversa, infixa, para que facilite a geração do código intermediário, logo tem-se

a b * a b * + c +

Como há a necessidade de retirar a base da pilha, foi implementado utilizando a estrutura deque, assim é possível retirar o elemento na base da estrutura (esquerda a direita), tendo o código gerado, a seguir supondo que a=10, b=20 e c=50.

Código Intermediário	Pilha	Deque
CRVL a*	10	b * a b * + c +
CRVL b*	10 20	* a b * + c +
MULT	10 * 20 = 200	a b * + c +
CRVL a*	200 10	b * + c +
CRVL b*	200 10 20	* + c +
MULT	200 200	+ c +
SOMA	200+200 = 400	c +
CRVL c*	400 50	+
SOMA	400+50 = 450	

Tabela 2. Exemplo de geração de código para resolver expressões.

Interpretador

O interpretador de código possui uma pilha e um vetor de variáveis para auxiliar na execução do código intermediário. Utiliza-se o vetor de variáveis para armazenar todos os valores de todas as variáveis alocadas e a pilha para realizar operações das instruções, utilizando apenas o topo dela, apenas as funções pop e push foram utilizadas.

Ambas as estruturas são do tipo double, já que aceitas ambos os tipos de variáveis (inteiras e ponto flutuante). Quando inicializa ou há a necessidade de desalocar memória, instrução DMEM, insere Double.NaN (not a number) em sua posição.

O índice do vetor refere-se ao “endereço” determinado na análise semântica, iniciando em 0.