Trabalho Prático
Projeto e Implementação de um Compilador

Disciplina: Compiladores

Data: 2018/01

1. Especificação da Linguagem

<PROGRAMA> ::= **inicio** <DECL_SEQUENCIA> **;** <COMANDOS> **; fim**
<DECL_SEQUENCIA> ::= <TIPO> **:** <DECL>
DECL> ::= <IDENTIFICADOR> **,** <DECL> | <IDENTIFICADOR>
<COMANDOS> ::= <COMANDO> | <COMANDOS> **;** <COMANDO>
<COMANDO> ::= <ATRIBUICAO | <LEITURA> | <ESCRITA>
<ATRIBUICAO> ::= <IDENTIFICADOR> **:=** <EXP>
<LEITURA> ::= **leia (** <IDENTIFICADOR> **)**
<ESCRITA> ::= **escreva (** <IDENTIFICADOR> **)**
<EXP> ::= <EXP> <OP> <EXP> | **(** <EXP> **)** | **num** | **id**
<OP> ::= + | - | ***
<TIPO> ::= **int**
<IDENTIFICADOR> ::= **id**

Observações:

a) Comentários: Uma vez que os comentários servem apenas como documentação do código fonte, ao realizar a compilação deste código faz-se necessário eliminar todo o conteúdo entre seus delimitadores: /* */

b) Tipo de dado: A linguagem admite apenas números naturais IN = {0, 1, 2, 3, 4, ...}.

2. Exemplos de Programas

1)

```
/* programa de exemplo /*
inicio /* início do programa */
    int : x ; /* Declaração de variáveis */
    leia ( x ) ; /* entrada de um número */
    escreva ( x ) ; /* saída de um número */
fim /* término do programa */
```

2)

```
/* programa de exemplo /*
inicio /* início do programa */
    int : x , y , z ; /* Declaração de variáveis */
    leia ( x ) ; /* entrada de um número */
    leia ( y ) ; /* entrada de um número */
    z := x + y ; /* soma x e y e armazena em z */
    escreva ( z ) ; /* saída de um número */
fim /* término do programa */
```

3)

```
inicio
    int : x , y , z ;
    leia ( x ) ;
    leia ( y ) ;
    z := x + y ;
    y := ( z * x ) + ( y * z);
    escreva ( y ) ;
fim
```

3 Objetivos

O objetivo deste trabalho é desenvolver um compilador para uma linguagem simples de expressões aritméticas.

O compilador deve conter as etapas de:

1. Análise léxica
2. Análise sintática
3. Análise semântica
4. Geração de Código Objeto

3.1. Análise Léxica

Nesta etapa o analisador deverá fazer a leitura do programa fonte e gerar como saída uma lista de tokens reconhecidos. Em caso de erro o analisador léxico deve informar a linha em que ocorreu o erro e o tipo de erro encontrado. É importante destacar que em caso de erros o analisador deve informar o erro e prosseguir no seu processo de análise. Lembre-se que ao encontrar um erro o compilador não irá gerar o código objeto mas irá realizar toda a etapa de análise (léxica, sintática e semântica).

3.1.1 Erros Léxicos

Serão considerados erros léxicos as 2 possibilidades:

1. Nome de identificador inválido: São considerados identificadores válidos somente os nomes de variáveis que iniciem por uma letra de a..z seguido por zero ou mais letras ou dígitos (0..9). Portanto são exemplos de identificadores válidos:

int : valor, x, y, aux1, aux100, flag, flag1, nota1, nota2;

São considerados identificadores inválidos:

int : v@lor, _x, #y, 100aux;

Neste caso o programa deve mostrar ao usuário uma mensagem de erro indicando a linha em que o erro ocorreu e a mensagem “identificador inválido”

2. Número inválido: Considerando-se que a linguagem admite apenas números naturais não são permitidos números inteiros negativos, nem números reais tais como: -5, 5.0, -6.34

Neste caso o programa deve mostrar ao usuário uma mensagem de erro indicando a linha em que o erro ocorreu e a mensagem “número inválido”

Esses portanto, são os únicos erros possíveis no processo de análise léxica. Um erro como, por exemplo, uma palavra reservada escrita de maneira errada (por exemplo, leiia ao invés do comando leia) é um erro sintático que será reconhecido pelo analisador léxico como um identificador válido. Caberá ao analisador sintático reportar este erro futuramente.

3.2. A Análise Sintática

A análise sintática irá receber como entrada a lista de tokens reconhecida pelo analisador léxico e tem como principal função indicar se a palavra pertence à linguagem, neste caso o programa está sintaticamente correto ou, se a palavra não pertence à linguagem caracterizando-se por um erro sintático.

Para a construção do analisador sintático fica à escolha do aluno qualquer um dos seis analisadores sintáticos possíveis.

3.3. Análise Semântica

É a etapa responsável por verificar erros semânticos, por exemplo, variáveis que estão sendo utilizadas mas não foram declaradas. Outra importante ação, que é de responsabilidade do analisador semântico, é o de dar significado aos comandos reconhecidos da linguagem.

Para esta etapa deverá ser feito o projeto de um analisador semântico de alto nível, ou seja, deverá ser entregue a tabela de ações semânticas e um exemplo de grafo de dependências que indique a ordem de avaliação dos atributos estabelecidos para os símbolos da gramática.

3.4. Geração de Código

É a etapa responsável por fazer o mapeamento do código fonte para o código objeto (mais especificamente para a linguagem assembly) Para a geração de código deverá ser feito o projeto de tradução para código intermediário utilizando-se da técnica de código de 3 endereços.

Referências

Compiladores Princípios e Práticas. Kenneth C. Louden.

Compiladores. Princípios, Técnicas e Ferramentas. Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman.

Implementação de Linguagens de Programação: Compiladores. Ana Maria de Alencar Price e Simão Sirineo Toscani