Disciplina: Compiladores - UFS
Turma: T01 2019.2
Prof. Galileu Santos de Jesus

Introdução

Com diversas linguagens de programação em inglês, iremos criar uma em nosso idioma. O nome iremos definir no decorrer do curso. Será uma linguagem experimental, imperativa e apresenta as características deste documento. Onde o mesmo pode sofrer modificações no decorrer do curso. Em caso de modificações na especificação, este documento será atualizado.

A seguir são discutidas as características de nossa linguagem. Atentos à especificação, uma vez que não seguida, poderá receber nota inferior.

1. Características e Léxico

Regras para identificadores:
1 - Pode-se utilizar: números, letras maiúsculas, letras minúsculas e underscore (sublinhado).
2 - O primeiro caractere deve ser sempre uma letra ou o underscore.
3 - Não são permitidos espaços em branco e caracteres especiais (ex.: @, $, +, -, %, ! etc.).
4 - Não podemos usar palavras reservadas (palavras que sejam um token da linguagem).
5 – Identificadores numéricos podem ser do tipo inteiro ou real. A diferença entre eles é a parte decimal dos números reais, separada por uma vírgula.

Regras para comentários:
1 - A linguagem aceita comentários de linha indicados por //
2 - A linguagem aceita comentários de bloco (possivelmente de múltiplas linhas) indicados por /* e */
3 - Os comentários de bloco devem ser aninhados e balanceados.

Regras para strings:
1 - As strings serão escritas entre aspas simples ' e '.
2 - Uma string não pode conter uma quebra de linha.
3 - Perceba que não existe um tipo string na linguagem, mas sim um tipo caractere. Logo, se quisermos declarar uma string de tamanho 3, fazemos:

```
caractere: var_str [3];
```

Regras para vetores:
1 - Os vetores são declarados da seguinte forma: tipo : nome_vetor [tam];
2 - Os índices dos vetores vão de 0 a tam-1 (isso também vale para strings).
3 - Não existem vetores com mais de uma dimensão (e se perguntar o que significa isso perde meio ponto).

Regras para expressões lógicas:
1 - Os operadores >, <, >= e <= não podem ser usados com operandos que sejam expressões lógicas.
2 – Os operadores = e <> não podem ser usados em cadeia. Ex.: a = b = c
3 – Uma cadeia com os operadores = e <> pode ser obtida através do uso de parênteses. Ex.: (a = b) = (c = d)

Operadores:
1 - Operadores aritméticos: +, -, *, /
2 - Operadores relacionais: >, <, >=, <=, =, <>
3 - Operadores booleanos: 'nao', 'e' e 'ou'.
4 - A prioridade dos operadores é igual à de C, e pode ser alterada com o uso de parênteses.
5 - Atribuição de valores é feita com o operador :=
6 - Comandos são terminados com ; (ponto e vírgula).

Palavras reservadas:
1 - caractere, real, inteiro, inicio, fim, se, senão, enquanto, para, avalie, caso, verdadeiro, falso, booleano.
2 - Todos os operadores e divisores da linguagem.

2. Sintático

A gramática abaixo foi escrita em uma versão de E-BNF seguindo as seguintes
convenções:
1 - variáveis da gramática são escritas em letras minúsculas sem aspas.
2 - Tokens são escritos entre aspas simples
3 - símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado.
4 - o símbolo '|' indica produções diferentes de uma mesma variável.
5 - o operador [ ] indica uma estrutura sintática opcional.
6 - o operador { } indica uma estrutura sintática que é repetida zero ou mais vezes.

```
programa : 'programa' ID 'inicio' {declaracao} {comando} 'fim.'
declaracao : tipo ':' {var ','} var ';'
                | 'const' ID valor ';'
tipo : 'real' | 'inteiro' | 'caractere'
var : ID | ID '[' N_INT ']'
valor : STRING | N_INT | N_REAL
comando : var ':=' exp ';'
          | 'leia' '(' {var ','} var ')' ';'
          | 'escreva' '(' {exp ','} exp ')' ';'
          | 'se' '(' exp-logica ')' 'entao' {comando} comando ['senao' {comando}
          comando] 'fim se' ';'
          | 'avalie' '(' exp ')' {'caso' valor ':' {comando} comando} ['senao' ':'
          {comando} comando] 'fim avalie' ';'
          | 'enquanto' '(' exp-logica ')' 'faca' {comando} comando 'fim enquanto'
          ';'
          | 'repita' {comando} comando 'ate' '(' exp-logica ')' ';'
          | 'para' var 'de' N_INT 'ate' N_INT 'faca' {comando} comando
          'fim para' ';'
          | 'para' var 'de' N_INT 'passo' N_INT 'ate' N_INT 'faca' {comando}
          comando 'fim para' ';'
exp : valor
          | var
          | '(' exp ')'
          | '-' exp
          | exp '+' exp
          | exp '-' exp
          | exp '*' exp
          | exp '/' exp
          | exp-logica
exp-logica:
          '(' exp-logica ')'
          | exp '=' exp
          | exp '<>' exp
          | exp '<=' exp
          | exp '>=' exp
          | exp '<' exp
          | exp '>' exp
          | 'nao' exp-logica
          | exp-logica 'e' exp-logica
          | exp-logica 'ou' exp-logica
          | exp-logica 'xor' exp-logica
```

3. Semântico

- Nos casos omissos neste documento, a semântica da linguagem segue a semântica de C
- As expressões em se e enquanto devem avaliar um tipo booleano
- Em operações entre os tipos inteiro e real, os valores inteiros devem ser convertidos para reais.

4. Desenvolvimento do Trabalho

Os trabalhos devem ser desenvolvidos em dupla ou individualmente. Será aberto um fórum no SIGAA para a discussão sobre cada uma das etapas. Em caso de dúvida, verifique inicialmente no fórum se ela já foi resolvida. Se ela persiste, consulte o professor.

4.1. Ferramentas

- Submissão das etapas do projeto via SIGAA. Será criada uma tarefa para cada etapa.
- Implementação com SableCC e Java.

4.2. Avaliação

- A avaliação será feita com base nas etapas entregues e em entrevistas feitas com os grupos.
- A aula que define o prazo de entrega de cada etapa está especificada no plano de curso da disciplina.

4.3. Etapas

Análise Léxica (valor: 2.5):

- Analisador léxico em SableCC
- Impressão dos lexemas e tokens reconhecidos
- Três códigos da linguagem que, unidos, usem todos os recursos da linguagem.

Análise Sintática (valor: 2.5):

- Analisador sintático em SableCC
- Impressão da árvore sintática em caso de sucesso

Sintaxe Abstrata (valor: 2.5):

- Analisador sintático abstrato em SableCC
- Impressão da árvore sintática em caso de sucesso

Análise Semântica (valor: 2.5):

- Validação de escopo e de existência de identificadores
- Verificação de tipos

Geração de código (extra: 2.0):

- Código em linguagem alvo: Pascal

A critério do docente, o valor das etapas pode ser modificado, desde que este novo cálculo produza uma nota não menor que a produzida pelo cálculo original. Entregas após o prazo sofrem penalidade de meio ponto por dia de atraso.

Bom projeto!