



Especificação da Linguagem Tesauro

Introdução

Para variar um pouco, vamos criar uma linguagem cujas expressões e palavras reservadas são levemente diferentes das que costumamos usar na maioria das linguagens de programação. A linguagem **Tesauro** é uma linguagem imperativa, e apresenta as características descritas neste documento.

Tesauro é uma linguagem experimental, então esta especificação é passível de adaptações. Em caso de modificações na especificação, estas serão notificadas via SIGAA.

1. Características e léxico

Regras para identificadores:

- Pode-se utilizar: letras maiúsculas, letras minúsculas e underline ('_').
- O primeiro caractere deve ser sempre uma letra.
- Não são permitidos números, espaços em branco e caracteres especiais (ex.: @, \$, +, -, ^, % etc.).
- Identificadores não podem ser iguais às palavras reservadas ou operadores da linguagem.
- Podem representar o nome do código, variáveis ou constantes (**unalterable**).

Tipos primitivos:

- A linguagem aceita os tipos symbol, real, integer.
- symbol: tipo que representa um elemento da tabela ASCII, sendo escrito com aspas simples. Exemplo: 'a', '\n'.
- real: números reais, com parte decimal separada por um ponto.
- integer: números inteiros. Podem ser expressos no sistema binário e decimal. Exemplo: b10100 (binário) ou d10 (decimal). Na base decimal não é necessário prefixo (mas ele é opcional).

Vetores:

- Um vetor é composto de uma ou mais variáveis com o mesmo tipo primitivo.
- O tamanho dos vetores é definido durante sua criação.
- Os índices dos vetores vão de 1 ao seu tamanho.
- Existem vetores multidimensionais. Exemplo: **integer vector [2][3] nome;**
- Vetores unidimensionais do tipo **symbol** podem ter seus valores definidos por cadeias entre aspas duplas (“ e ”).

Blocos

- Delimitados pelas palavras **start** e **finish**.

Comentários:

- A linguagem aceita comentários de uma ou mais linhas, delimitados por { e }.
- O funcionamento dos comentários de bloco em Tesauro são similares aos da linguagem C. Exemplo: o que acontece se você compilar `/**/` em C? Estudem como um compilador C reconhece o fim de um comentário de bloco.

Estruturas de controle (mais detalhes na Seção Semântico):

- **in case that**
- **as long as**
- **considering**

Operadores:

- Possui operadores aritméticos, relacionais e booleanos.
- Comandos são terminados com ';' (ponto-e-vírgula).

Funções primitivas:

- A linguagem possui dois procedimentos primitivos: **capture** e **show**

2. Sintático

A gramática da linguagem foi escrita em uma versão de E-BNF seguindo as seguintes convenções:

- Variáveis da gramática são escritas em letras minúsculas sem aspas;
- Tokens são escritos entre aspas simples;
- Símbolos escritos em letras maiúsculas representam o lexema de um token do tipo especificado;
- O símbolo | indica produções diferentes de uma mesma variável;
- O operador [] indica uma estrutura sintática opcional;
- O operador { } indica uma estrutura sintática que é repetida zero ou mais vezes.

```

programa : 'code' ID bloco

bloco : 'start' { declaracao } { comando } 'finish'

declaracao : tipo {ID ','} ID ';'
| 'unalterable' tipo ID '=' valor ';'

tipo-base : 'integer' | 'real' | 'symbol'

tipo : tipo-base | tipo-base 'vector' '[' exp ']' {'[' exp ']' }

var : ID | ID '[' N_INT ']' {'[' N_INT ']' }

valor : SYM | N_INT | N_REAL

comando :
    var ':=' exp ';'
| 'capture' '(' {var ','} var ')' ';'
| 'show' '(' {exp ','} exp ')' ';'
| 'in case that' '(' exp-logica ')' 'do' comando ['else' comando] ';'
| 'as long as' '(' exp-logica ')' 'do' comando ';'
| 'considering' var 'from' N_INT 'to' N_INT 'by' N_INT 'do' comando ';'
| bloco

exp :
| valor
| var
| '(' exp ')'
| '-' exp
| exp '+' exp
| exp '-' exp
| exp '*' exp
| exp '/' exp
| exp '%' exp
| exp '==' exp
| exp '!=' exp
| exp '<=' exp
| exp '>=' exp
| exp '<' exp
| exp '>' exp
| '!' exp-logica
| exp-logica 'and' exp-logica
| exp-logica 'or' exp-logica
| exp-logica 'xor' exp-logica

```

3. Semântico:

- Nos casos omissos neste documento, a semântica da linguagem segue a semântica de C.
- A execução de um programa consiste na execução do código iniciado com a palavra **code**, seguida por um bloco.

Blocos

- Escopo das variáveis e constantes: global e local.

Estruturas de controle:

- **in case that**: similar ao “if-else” do C.
- **as long as**: similar ao “while” do C.
- **considering**: similar ao “for” do C. O terceiro número inteiro representa o incremento (passo) que a variável sofrerá a cada iteração.

Operadores:

- Em operações entre os tipos inteiro e real, os valores inteiros devem ser convertidos para reais.
- A semântica das operações com resultado booleano é igual à de C: 0 vale como falso e qualquer outro inteiro é verdadeiro.
- A prioridade dos operadores é igual à de C, e pode ser alterada com o uso de parênteses.
- Em qualquer expressão, um caractere tem seu valor automaticamente promovido para inteiro.

Procedimentos primitivos:

- **capture**: procedimento para entrada de dados a partir do teclado. Salva os valores lidos nas variáveis que foram passadas como argumentos.
- **show**: procedimento para exibição de um ou mais valores resultantes de expressões passadas como argumentos.

O que deve ser verificado na análise semântica:

- Se entidades criadas pelo usuário (variáveis, vetores e constantes) são inseridas na tabela de símbolos - com os atributos necessários - quando são declaradas;
- Se uma entidade foi declarada e está em um escopo válido no momento em que ela é utilizada (regras de escopo são iguais às de C);
- Se entidades foram definidas (inicializadas) quando isso se fizer necessário;
- Checar a compatibilidade dos tipos de dados envolvidos nos **comandos, expressões e atribuições**.

4. Desenvolvimento do Trabalho

Trabalhos devem ser desenvolvidos em trio, dupla ou individualmente. Foi aberto um fórum no SIGAA para a discussão sobre as etapas. Em caso de dúvida, verifique inicialmente no fórum se ela já foi resolvida. Se ela persiste, consulte a professora.

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):

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

Análise Sintática (valor: 2.5):

- Analisador sintático em SableCC
- Impressão da árvore sintática em caso de sucesso ou impressão dos erros

Sintaxe Abstrata (valor: 2.5):

- Analisador sintático abstrato em SableCC
- Impressão da árvore sintática

Análise Semântica (valor: 2.5):

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

Geração de código (extra: 2.5):

- Código em linguagem alvo: Pascal

A critério da 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 trabalho!