Insper

Lógica da Computação - 2019/2

Roteiro 9 - Simple Calculator **v2.4**

Raul Ikeda - rauligs@insper.edu.br

Entrega: 2019/2

Nome:

Objetivos

1. Implementar Declarações de Funções
2. Escopo de variáveis

Exemplo de sintaxe:

```
Function Soma(x as Integer, y as Integer) as Integer
  Dim a as Integer
  a = x + y
  Print a
  Soma = a
End Function

Sub Main()
  Dim a as Integer
  Dim b as Integer
  a = 3
  b = Soma(a, 4)
  Print a
  Print b
End Sub
```

Parte I: Funções

Como exemplificado acima, o VBA permite declarar funções simples e com declaração de tipo. Todas as funções a princípio possuem visibilidade pública e podem ser usadas em outras Funções (ou Sub). Não é possível declarar uma função dentro de uma função.

Tarefas

- Criar uma *branch* a partir da versão v2.3.X.
- Atualizar o EBNF e o DS no GitHub.
- Modificar o *Tokenizer* para realizar a tokenização correta.
- Criar mais 2 nós da AST:
  - *FuncDec*: possui 2 filhos: *VarDec* e *Statements*. Os argumentos da declaração devem ser incorporados ao *VarDec*, incluindo o próprio nome da função e seu tipo correspondente. O *Evaluate()* apenas cria uma variável na *SymbolTable* atual, sendo o nome da variável o nome da função, o valor apontando para o próprio nó *FuncDec* e o tipo será *FUNCTION*.

  - *FuncCall*: possui n filhos do tipo identificador ou expressão - são os argumentos da chamada. O *Evaluate()* vai realizar o verdadeiro *Evaluate()* da *FuncDec*, recuperando o nó de declaração na *SymbolTable*, atribuindo os valores dos argumentos de entrada e executando o bloco (segundo filho).
- Modificar o analisador sintático para interpretar o bloco de declaração corretamente. Colocar a análise da chamada da função dentro da função que analisa atribuições (se você atualizou o DS sabe o porquê disso).
- Ao criar a raiz da sua AST (um nó do tipo *Statements*), adicione como último filho, uma chamada de função *FuncCall* para a função *Main()*. Lembrando que é responsabilidade do Sintático verificar se ela existe ou não.

Parte II: Escopo de variáveis

A ideia principal da implementação de escopo é criar uma nova *SymbolTable* para cada chamada de função. Contudo as variáveis pertencentes aos blocos ancestrais também devem estar visíveis.
Para solucionar o problema, vamos modificar a classe *SymbolTable* para agir como uma lista ligada de objetos *SymbolTable*, onde o novo objeto aponta para o seu ancestral e a tabela inicial aponta para nulo. Portanto a verificação de uma variável deve seguir de forma recursiva no ancestral imediato até acabar a lista na raiz.
No caso de bloco de comandos, as variáveis pertencentes ao ancestral são visíveis ao bloco interno. No exemplo acima, a variável *b* declarada na *Main()* é visível dentro da função *Soma()*.
Já a variável *a* declarada na *Main()* é inalcançável em *Soma()*, já que ela possui sua própria declaração.
No caso de conflito de nomes, vale a declaração mais recente. Veja as impressões da variável *a* no programa acima. O retorno da função é dada pela variável que possui o nome da função. Essa variável funciona como uma variável local declarada após a sua chamada. Não confundir com a variável que possui o tipo *FUNCTION* alocada na *SymbolTable* raiz.

Tarefas:

- Criar um atributo *ancestor* na classe *SymbolTable*
- Adicionar ao construtor o argumento que preenche o atributo. Toda vez que criar uma nova, passa a *SymbolTable* atual como ancenstral. No primeiro objeto, passar um valor nulo.
- Quando for pesquisar uma variável na tabela, deve primeiro olhar o próprio dicionário, depois olhar para ancestral imediato recursivamente até atingir a raiz.
- Cada vez que houver uma Chamada de Função, criar uma nova *SymbolTable* e realizar a troca das tabelas na etapa semântica.
- Tomar cuidado com as mensagens de erro.

Base de Testes:

Além do código exemplo acima que deve funcionar conforme o esperado, teste os seguintes elementos:

- Com erros:
  - Chamar a função com número incorreto de argumentos.
  - Passar argumentos de tipos diferentes.
  - Chamar uma função que não existe.
  - Usar uma variável fora de escopo.
- Sem erros:
  - Fazer uma função recursiva.

1. Sugira testes adicionais

Para ganhar A+

- Prazo: 03/Jun/2019.
- Você deve implementar a geração de código para funções.
  - O seu compilador deve realizar todas as etapas de compilação.
  - Ele deve ser hábil suficiente para fazer chamadas recursivas.
  - Gerar um relatório explicando como foi feita a etapa.