

## Trabalho Prático - Especificação da Etapa 4: Verificação Semântica

### Resumo:

Na quarta etapa do trabalho de implementação de um compilador para a linguagem **break192** é necessário fazer verificação semântica, ou seja, uma série de testes de coerência comportamental das construções sintáticas reconhecidas e representadas na AST.

O programa desta etapa deve manter as funcionalidades das etapas anteriores, e adicionalmente emitir as mensagens de erros semânticos, terminando com a chamada de **exit(4)** se tiver havido algum erro semântico. Assim, o programa tem as seguintes códigos de saída: 0- sucesso, sem erros sintáticos ou semânticos; 1- arquivo não informado; 2- arquivo inexistente; 3- erro de sintaxe; 4- existência de um ou mais erros semânticos.

### Visão Geral das Verificações:

- a. Declarações – Devem ser verificadas as declarações. Todos os identificadores devem ter sido declarados, seja como variável, vetor ou como função. Os símbolos definidos como identificadores na tabela de símbolos devem ter seu tipo trocado para um desses tipos conforme a declaração, verificando-se se não houve dupla declaração ou símbolo não declarado;
- b. Uso correto – o uso dos identificadores deve ser compatível com sua declaração. Variáveis somente podem ser usadas sem indexação, vetores somente podem ser usados com indexação, funções apenas devem ser usadas com chamada, isto é, seguidas da lista de argumentos entre parênteses;
- c. Tipos de dados – As declarações também devem registrar os tipos de dados, em um novo campo, *dataType*, na tabela de símbolos. Com o auxílio dessa informação, quando necessário, os tipos de dados corretos devem ser verificados onde forem usados, em expressões aritméticas, relacionais, lógicas, ou para índices de vetores;
- d. Argumentos e parâmetros – A lista de argumentos deve ser verificada contra a lista de parâmetros formais na declaração da função. Cada chamada de função deve prover um argumento para cada parâmetro, e ter o tipo **compatível**;

## Definições Semânticas:

- Há três tipos diferentes de identificadores:  
escalares, vetores, funções
- Há cinco tipos de dados para declarações:  
bool, byte, int, long, float
- Há dois tipos de dados incompatíveis entre si:  
aritméticos (números), booleanos
- Há quatro tipos de literais:  
inteiros, reais (em ponto flutuante), caracteres, strings
- Literais *string* só devem ser usados no comando *print*.  
Literais inteiros, reais em ponto flutuante e caracteres são intercambiáveis, **compatíveis**, e podem aparecer em quaisquer expressões aritméticas, e serem processados como dados numéricos inteiros (*byte*, *int*, *long*) ou reais (*float*). No acesso a vetor, **não** é necessário garantir que o índice seja um valor inteiro. Expressões que potencialmente resultem em valores flutuantes **podem** ser usadas como índice de vetor.
- Os tipos de dados inteiros e reais podem ser usados e convertidos livremente em expressões aritméticas, ou seja, são compatíveis, podendo ser também usados como argumentos para parâmetros definidos assim. O resultado de cada operação sempre terá o tipo de dado mais amplo e preciso.
- Nas expressões, é necessário verificar os tipos corretos distinguindo entre os tipos numéricos e os booleanos (variáveis booleanas ou resultado de operadores relacionais), que são incompatíveis entre si.
- Existe a possibilidade da identificação ser feita pelo tipo do nodo filho da árvore (tipo do operador) ou pelo tipo de dado (dataType) do identificador, se o nodo filho é um identificador (AST\_SYMBOL).

## Perguntas e Respostas

Alguma pergunta?

## Lista de Verificações

- variáveis redeclaradas
- anotar tipo (natureza) nas tabela hash
- anotar tipo de dado (dataType) na tabela hash
- variáveis não-declaradas
- verificar natureza, se:
  - escalares são usados como escalares
  - vetores são usados como vetores
  - funções são usadas como funções
  - não esqueça de verificar no lado direito (expressões) e no lado esquerdo (atribuições)

- verificar tipos de dados nas expressões
- verificar argumentos de chamada de função versus parâmetros:
  - não pode haver menos argumentos
  - não pode haver mais argumentos
  - os tipos devem ser compatíveis (não iguais, lembre-se)
- verificar tipo do valor de retorno da função
- verificar índices dos vetores (não pode ser booleano), tanto na expressão quanto na atribuição

## Controle e organização do seu código fonte

Você deve seguir as mesmas regras das etapas anteriores para organizar o código, permitir compilação com **make**, permitir que o código seja rodado com **./etapa4**, e esteja disponível como **etapa4.tgz**.

Porto Alegre, Outubro de 2019