Compiladores – 2017/1

Trabalho Prático T3

30 de maio de 2017

1 Objetivo

O objetivo deste trabalho é a criação de um analisador semântico para a linguagem C-Minus, como terceiro componente do *front-end* de um compilador.

2 Descrição do Problema

A implementação desse trabalho envolve duas atividades principais: escrever código que constrói a representação intermediária do programa de entrada como uma árvore de sintaxe abstrata (AST), e desenvolver alguns testes que analisam se o programa de entrada está correto do ponto de vista semântico.

O seu analisador semântico deve verificar:

1. se as variáveis/funções utilizadas foram declaradas;
2. se há variáveis/funções redeclaradas; e
3. se o número de argumentos (aridade) de uma chamada de função está correto.

É recomendável dividir o desenvolvimento do trabalho nas seguintes etapas:

- Implemente uma estrutura de dados árvore que seja adequada para uso como AST.
- Modifique a gramática do trabalho 2 para incorporar ações semânticas que constroem a AST do programa de entrada.
  Obs. 1: Use os arquivos `.dot` e `.pdf` disponibilizados pelo professor como referência para verificar a construção da sua AST.
  Obs. 2: A sua AST não precisa ficar idêntica ao do professor. Você é livre para construir a sua estrutura como preferir; os exemplos disponibilizados são apenas referências.
- Implemente uma tabela de símbolos e uma tabela de literais. Você é livre para utilizar a estrutura que preferir.
- Utilizando a AST e as tabelas, realize as verificações do analisador semântico e exiba mensagens de erro como abaixo.

2.1 Mensagens de erro

Utilização de variáveis e funções: Se uma variável for utilizada sem ser declarada, imprima no terminal:

```
SEMANTIC ERROR (XX): variable 'VV' was not declared.
```

Onde `XX` é o número da linha do programa onde o erro foi detectado e `VV` é o nome da variável.

Se uma função for utilizada sem ser declarada, imprima no terminal:

```
SEMANTIC ERROR (XX): function 'FF' was not declared.
```

Onde `XX` é o número da linha do programa onde o erro foi detectado e `FF` é o nome da função.

Redeclarações de variáveis e funções: Se uma variável for redeclarada no programa de entrada, imprima no terminal:

`SEMANTIC ERROR (XX): variable 'VV' already declared at line YY.`

Onde `XX` é o número da linha do programa onde o erro foi detectado, `VV` é o nome da variável e `YY` é o número da linha aonde a variável foi originalmente declarada.

Se uma função for redeclarada no programa de entrada, imprima no terminal:

`SEMANTIC ERROR (XX): function 'FF' already declared at line YY.`

Onde `XX` é o número da linha do programa onde o erro foi detectado, `FF` é o nome da função e `YY` é o número da linha aonde a variável foi originalmente declarada.

Aridade das funções: Se uma função for chamada com um número de argumentos incompatível com a sua aridade, imprima no terminal:

```
SEMANTIC ERROR (XX): function 'FF' was called with AA arguments but declared
with PP parameters.
```

Onde `XX` é o número da linha do programa onde o erro foi detectado, `FF` é o nome da função, `AA` é o número de argumentos passados para a função e `PP` é o número de parâmetros declarados.

Verificação de tipos: Neste trabalho não será realizada verificação explícita de tipos. Por conta disso, os casos de teste `err3_14` e `err3_15` que a princípio deveriam indicar erros semânticos são compilados com sucesso. Entretanto, é interessante destacar que o uso de tipos implícitos como `string`, `bool` em locais incorretos é detectado pelo *parser*. Isso é possível porque algumas informações de tipagem estão presentes diretamente nas regras da gramática (e.g., a regra que define o comando `write` indica que somente `strings` são aceitas). Assim, alguns casos de teste que efetivamente possuem erros semânticos (incompatibilidade de tipos) acabam marcados como contendo erros de sintaxe, pois é o *parser* que detecta o erro.

Demais mensagens: As mensagens de erros léxicos e sintáticos são as mesmas do Trabalho 2 e devem continuar sendo exibidas como antes.

Se o programa estiver correto, o seu compilador deve exibir uma mensagem indicando que o programa foi aceito, como abaixo:

```
$ ./trab3 < program.cm
PARSE SUCCESSFUL!
```

ATENÇÃO: os arquivos de saída disponibilizados pelo professor contém informações adicionais úteis (conteúdo das tabelas) para auxiliar o desenvolvimento do trabalho. No entanto, o seu analisador deve exibir SOMENTE as mensagens indicadas nessa seção, quando forem adequadas. O seu analisador deve construir a AST do programa de entrada mas NÃO DEVE exibi-la na tela.

3 Implementando e Testando o Trabalho

As convenções léxicas e sintáticas da linguagem C-Minus já foram apresentadas nas especificações dos Trabalhos 1 e 2, respectivamente. Você deve adaptar o *scanner* e *parser* desenvolvidos anteriormente para compor o analisador semântico deste trabalho.

Observações importantes:

- O seu analisador pode terminar a execução ao encontrar o primeiro erro no programa de entrada.
- Os arquivos de entrada de exemplo são os mesmos do Trabalho 1.
- Valide o seu programa usando os gabaritos disponibilizados na sala da disciplina no AVA.
- Ao submeter o seu trabalho para correção, além dos códigos-fonte envie um arquivo `Makefile` que gera como executável para o seu analisador semântico um arquivo de nome `trab3`.

4 Regras para Desenvolvimento e Entrega do Trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 27/06/2017 (Terça-feira). Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho é **individual**.
- **Linguagem de Programação e Ferramentas**: Para implementar o seu analisador sintático você deve obrigatoriamente usar o `bison`.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado com todo o seu trabalho.
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

5 Avaliação

- O trabalho vale 3.0 pontos na média parcial do semestre.
- Trabalhos com erros de compilação receberão nota zero. Isso inclui trabalhos com conflitos de *shift-reduce* ou *reduce-reduce*.
- Caso seja detectado plágio (entre alunos ou da internet), todos os envolvidos receberão nota zero.
- Serão levadas em conta, além da correção da saída do seu programa, a clareza e simplicidade de seu código.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)