Compiladores

Trabalho Prático T3

19 de maio de 2016

1 Objetivo

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

1.1 A LP de entrada

A linguagem de entrada padrão que o seu compilador deve aceitar é o G-Portugol (veja o manual da linguagem no AVA). Você é livre para escolher uma outra linguagem de entrada (e.g., Pascal, etc) mas nesse caso converse com o professor antes de começar para ver se a LP que você escolheu é razoável para este projeto.
Obs.: Toda a documentação do professor vai assumir que a linguagem de entrada é o G-Portugol.

2 Descrição do Problema

Você deverá implementar um analisador semântico para programas escritos em G-Portugol, partindo obrigatoriamente do seu *parser* e *scanner* desenvolvidos nos trabalhos T2 e T1, respectivamente.

O seu analisador semântico deve verificar:

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

Você deverá utilizar uma tabela hash para implementar sua Tabela de Variáveis/Funções.

O seu analisador semântico deve imprimir mensagens de erro na tela quando o programa de entrada estiver incorreto. Se o programa estiver correto, não é necessário imprimir nada. (Erros sintáticos continuam sendo impressos da mesma forma – conforme trabalho 2.)

IMPORTANTE 1: Você pode retirar o código que gera a árvore de derivação do seu* parser *antes de começar esse trabalho.

IMPORTANTE 2: O seu analisador pode terminar quando o primeiro erro for encontrado.

2.1 Informações Adicionais

Tabela de funções: as funções primitivas da linguagem são: `imprima` e `leia`. A função de leitura tem aridade 0. Já a função de impressão tem aridade variável (função do tipo *varargs* em C – como o `printf`), e portanto a verificação de aridade para `imprima` deve ser se o número de argumentos é > 0.

Utilização de variáveis: se uma variável for utilizada sem ser declarada, imprima no terminal:

```
Erro semantico na linha NNN: variavel 'xxx' nao foi declarada.
```

Onde `NNN` é o número da linha do programa onde o erro foi detectado e `xxx` é o nome da variável.
Obs. 1: O seu programa deve fazer esse teste também para as funções do programa.
Obs. 2: A mensagem de erro para uso de funções não declaradas é análogo.

Redeclarações de variáveis: se uma variável for redeclarada no programa de entrada, imprima no terminal:

```
Erro semantico na linha NNN: variavel 'xxx' ja foi declarada na linha MMM.
```

Onde `NNN` é o número da linha do programa onde o erro foi detectado, `xxx` é o nome da variável e `MMM` é o número da linha aonde a variável foi originalmente declarada.
Obs.: A mensagem de erro para funções redeclaradas é análogo.

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

```
Erro semantico na linha NNN: a funcao 'fff' foi chamada com XXX argumentos
mas declarada com YYY parâmetros.
```

Verificação de tipos: caso haja alguma expressão com tipos de dados incompatíveis, imprima no terminal:

```
Erro semantico na linha NNN: expressao com tipos de dados incompatíveis.
```

2.2 Testando o trabalho

O trabalho será testado da seguinte maneira para um trabalho gerado com o `flex/bison`:

```
./trab3 < entrada.gpt
```

onde `entrada.gpt` é o arquivo com o programa na linguagem G-Portugol que deve ser analisado.

Caso o seu trabalho tenha sido feito no ANTLR a execução deve ser:

```
java Trab3 entrada.gpt
```

Exemplo de um arquivo de entrada (`teste.gpt`):

```
algoritmo teste;
inicio
    x := 42 + 3;
    f := 0 <= 1;
fim
```

Executando `./trab3 < teste.gpt`, teremos como resposta no terminal:

```
Erro semantico na linha 3: variavel 'x' nao foi declarada.
```

Ao submeter o seu trabalho para correção, além dos códigos-fonte envie um arquivo LEIAME com as instruções para compilar o seu trabalho. Idealmente, a sua submissão deve incluir também um arquivo `Makefile` que gera como executável para o seu *parser* um arquivo de nome `trab3`. *(Se você não sabe usar o `make` mas quer aprender, converse com o professor.)*

3 Regras para Desenvolvimento e Entrega do Trabalho

- **Data da Entrega**: O trabalho deve ser entregue até às 23:55 h do dia 09/06/2016 (Quinta-feira). Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho deve ser feito em grupos de 2 alunos. Informe seu grupo para o professor para que ele possa fazer o cadastro no AVA. IMPORTANTE: alunos sem cadastro no AVA ou que não tenham sido incluídos em algum grupo não conseguiram submeter o trabalho.
- **Linguagem de Programação e Ferramentas**: Para implementar o seu analisador sintático você pode escolher entre usar o `flex/bison` ou o ANTLR.
  Obs.: Se você quiser você pode usar outras ferramentas à sua escolha, mas nesse caso você deverá ser capaz de resolver os eventuais problemas que surgirem por conta própria.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado com todo o seu trabalho. SOMENTE uma pessoa da dupla deve enviar o trabalho no AVA. **Coloque o nome das duas pessoas da dupla no nome do arquivo do 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.

4 Avaliação

- O trabalho vale 2.5 pontos na média parcial do semestre.
- Trabalhos com erros de compilação receberão nota zero.
- 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.)