UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Projeto de Implementação de um Compilador para a Linguagem **T++**

Análise Semântica (Trabalho – 3ª parte)

Prof. Rogério Aparecido Gonçalves[1]

[1] *Universidade Tecnológica Federal do Paraná (UTFPR)*

Departamento de Computação (DACOM)

rogerioag@utfpr.edu.br

29 de outubro de 2019

Resumo

Este documento apresenta a especificação da 3ª parte do trabalho de implementação da disciplina. O objetivo nessa etapa é projetar e implementar a a fase de *Análise Semântica* do compilador para a linguagem **T++**.

Sumário

1 Análise Semântica

1.1 Instruções Gerais

1. Faça download do arquivo do modelo de estrutura do trabalho e relatório disponível na página da disciplina no moodle. Descompacte e trabalhe nos arquivos e estrutura fornecida, pois será a mesma estrutura que deverá ser entregue ao final do projeto.

2. Siga a estrutura fornecida para desenvolver o trabalho.

3. O relatório deve ter a descrição do trabalho e dos programas, o código fonte dos programas, uma explicação sobre o funcionamento do programa, o processo de tradução com exemplos de instruções dos três formatos e um exemplo de execução do seu programa reproduzindo a saída gerada.

4. Deverão ser entregues:

   a) O código fonte dos programas.
   b) Relatório em **pdf** que pode ser feito no formato do `LibreOffice` ou no `Latex`.

5. O projeto deve seguir a estrutura de diretórios e arquivos, disponível no formato. A estrutura do projeto é apresentada na Figura 1.

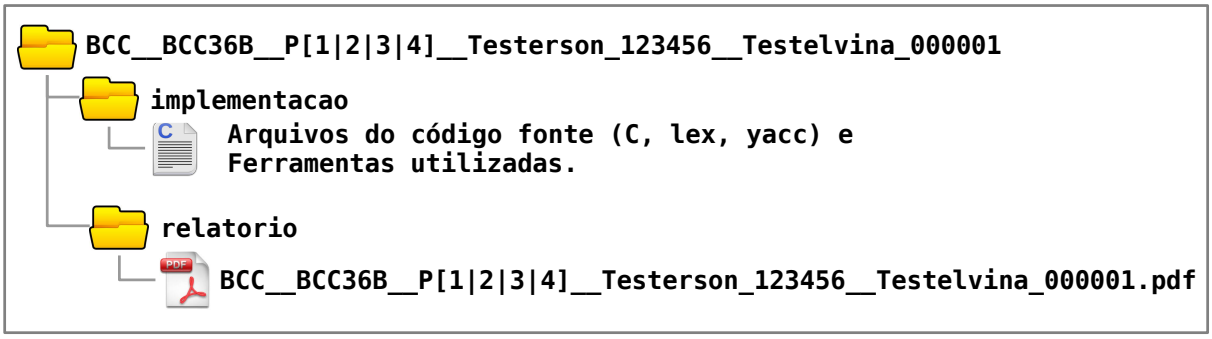

Figura 1: Formato de Entrega

1.2 Implementação

Com as Análises Léxica e Sintática já concluídas, espera-se ter como saída uma Árvore Sintática Abstrata (AST).

Para a terceira parte do trabalho, a **AST** deve ser percorrida para a realização da análise sensível ao contexto (conforme apresentado em sala de aula) e a geração da tabela de símbolos. A análise semântica consiste da "segunda passada" do compilador no código, pois o código fonte é novamente analisado desde o início na AST, porém agora para realizar a sua anotação

de atributos. A tabela de símbolos deve ser gerada por meio das estratégias apresentadas em sala de aula.

O principal motivo da **Análise Semântica** é verificar se existem erros de contexto para a linguagem **TPP** apresentada na primeira e segunda parte deste trabalho.

1.3 Regras Semânticas

As **Regras Semânticas** que precisam ser verificadas para a identicação de erros semânticos ou de contexto são:

1. **Funções e Procedimentos.** O **identificador de função** (nome e quantidade de parâmetros formais), além de que os parâmetros formais devem ter um apontamento para o identificador de variáveis.

2. **Função Principal.** Todo programa escrito em **TPP** deve ter uma *função principal* declarada. Verificar a existência de uma função principal que inicializa a execução do código. Caso contrário, deve apresentar a mensagem:

   Erro: *Função principal não declarada.*

   A função principal normalmente é do tipo `inteiro`, assim é esperado que seu retorno seja um valor `inteiro`, do contrário a mensagem deve ser emitida:

   Erro: Função principal deveria retornar inteiro, mas retorna vazio.

3. **Funções e Procedimentos.** A quantidade de parâmetros reais de uma chamada de função/procedimento `func` deve ser igual a quantidade de parâmetros formais da sua definição. Caso contrário, gerar a mensagem:

   Erro: Chamada à função 'func' com número de parâmetros menor que o declarado

4. **Funções e Procedimentos.** Uma função deve retornar um valor de tipo compatível com o tipo de retorno declarado. Se a função principal que é declarada com retorno inteiro, não apresenta um `retorna(0)`, a mensagem deve ser gerada:

   Erro: Função principal deveria retornar inteiro, mas retorna vazio.

   Funções precisam ser declaradas antes de serem chamadas. Caso contrário a mensagem de erro deve ser emitida:

   Erro: Chamada a função 'func' que não foi declarada.

   Uma função qualquer não pode fazer uma chamada à função principal. Devemos verificar ser existem alguma chamada para a função principal partindo de qualquer outra função do programa.

   Erro: Chamada para a função principal não permitida.

   Uma função pode ser declarada e não utilizada. Se isto acontecer uma aviso deverá ser emitido:

   Aviso: Função 'func' declarada, mas não utilizada.

Se a função principal fizer uma chamada para ela mesmo, a mensagem de aviso deve ser emitida:

Aviso: Chamada recursiva para principal.

5. **Variáveis.** O **identificador de variáveis locais e globais**: nome, tipo e escopo devem ser aramazenados na Tabela de Símbolos. Variáveis devem ser **declaradas**, **inicializadas** e antes de serem **utilizadas** (leitura). Lembrando que uma variável pode ser declarada:

   - no escopo do procedimento (como expressão ou como parâmetro formal);
   - no escopo global

   Warnings deverão ser mostrados quando uma variável for declarada mas nunca utilizada. Se uma variável 'a' for apenas declarada e não for inicializada (escrita) ou não for utilizada (não lida), o analisador deve gerar a mensagem:

   Aviso: Variável 'a' declarada e não utilizada.

   Se houver a tentativa de leitura ou escrita de qualquer variável não declarada a seguinte mensagem:

   Aviso: Variável 'a' não declarada.

   Se houver a tentativa de leitura de uma variável 'a' declarada, mas não inicializada:

   Aviso: Variável 'a' declarada e não inicializada.

   Warnings deverão ser mostrados quando uma variável for declarada mais de uma vez. Se uma variável 'a' for declarada duas vezes no mesmo escopo, o aviso deve ser emitido:

   Aviso: Variável 'a' já declarada anteriormente

6. **Atribuição.** Na atribuição devem ser verificados se os tipos são compatíveis. Por exemplo, uma variável `a` recebe uma expressão `b + c`. Os tipos declarados para `a` e o tipo resultado da inferência do tipo da expressão `b + c` deverão ser compatíveis. Se `b` for `inteiro` e `c` for `inteiro`, o tipo resultante da expressão será também `inteiro`. Se assumirmos, por exemplo, que `b` é do tipo `inteiro` e `c` do tipo `flutuante`, o resultado pode ser `flutuante` (se assumirmos isso para nossa linguagem). O que faria a atribuição `a := b + c` apresentar tipos diferentes e a mensagem deve ser apresentada:

   Aviso: Atribuição de tipos distintos 'a' inteiro e 'expressão' flutuante

   O mesmo pode acontecer com a atribuição de um retorno de uma função, se os tipos forem incompatíveis o usuário deve ser avisado:

   Aviso: Atribuição de tipos distintos 'a' flutuante e 'func' retorna inteiro

7. **Coerções implícitas.** *Warnings* deverão ser mostrados quando ocorrer uma coerção implícita de tipos (inteiro<->flutuante). Atribuição de variáveis ou resultados de expressões de tipos distintos devem gerar a mensagem:

   Aviso: Coerção implícita do valor de 'x'. **Aviso:** Coerção implícita do valor retornado por 'func'.

8. **Arranjos.** Na linguagem `tpp` é possível declarar arranjos, pela sintaxe da linguagem o índice de um arranjo é inteiro e isso deve ser verificado. Na tabela de símbolos devemos armazenar se uma variável declarada tem um tipo, se é uma variável escalar ou um vetor ou uma matriz. Podemos armazenar um campo 'dimensões', que '0': escalar, '1': arranjo unidimensional (vetor) e '2': arranjo bidimensional (matriz) e assim por diante. Encontrado a referência a um arranjo, seu o índice, seja um número, variável ou expressão deve ser um `inteiro`. Do contrário, a mensagem deve ser gerada:

   Erro: Índice de array 'X' não inteiro.

   Se o acesso ao elemento do arranjo estiver fora de sua definição, por exemplo um vetor `A` é declarado como tendo `10` elementos (0 a 9) e há um acesso ao `A[10]`, a mensagem de erro deve ser apresentada:

   Erro: índice de array 'A' fora do intervalo (out of range)

9. E outros situações descritas nos arquivos de testes.

Para a realização da análise dos erros apontados, a construção da **Tabela de Símbolos** deve ser realizada adequadamente. Uma Tabela de Símbolos pode começar a ser construída desde as análises anteriores. Por exemplo, na **Análise Léxica** temos os `TOKENS` e os lexemas que casaram com a expressão regular que gerou cada um desses tokens. Temos também o número da linha e da coluna que o lexema foi encontrado no código fonte de entrada. Desta forma, as entradas da Tabela de Símbolos podem ir sendo preenchidas com essas informações iniciais .

Na **Análise Sintática** quando o reconhecimento entrar na função que trata a regra, por exemplo, na regra de `declaração de variáveis`, todas as variáveis da lista de identificadores irão assumir o tipo declarado:

```
inteiro: a, b, c[10]
```

Que entraria na regra:

```
declaracao_variaveis ::= tipo ":"lista_variaveis
```

A declaração indica que os elementos da `lista_variaveis`, que são as variáveis escalares `a` e `b` e o arranjo unidimensional `c` terão por declaração o `tipo` que no exemplo é `inteiro`.

1.4 Exemplo

```
1 inteiro: a, b, c[10], d[10][10]
2
3 inteiro func
4
5 inteiro principal()
```

1.5 Árvore Sintática Abstrata

Após a análise semântica e geração de dados referentes a esta passagem, é esperado como saída uma árvore sintática abstrata anotada (ASTO).

Tomemos como exemplo uma atribuição `a := b + c`, de acordo com a sintaxe da linguagem **TPP** temos a subárvore representada na Figura 2.

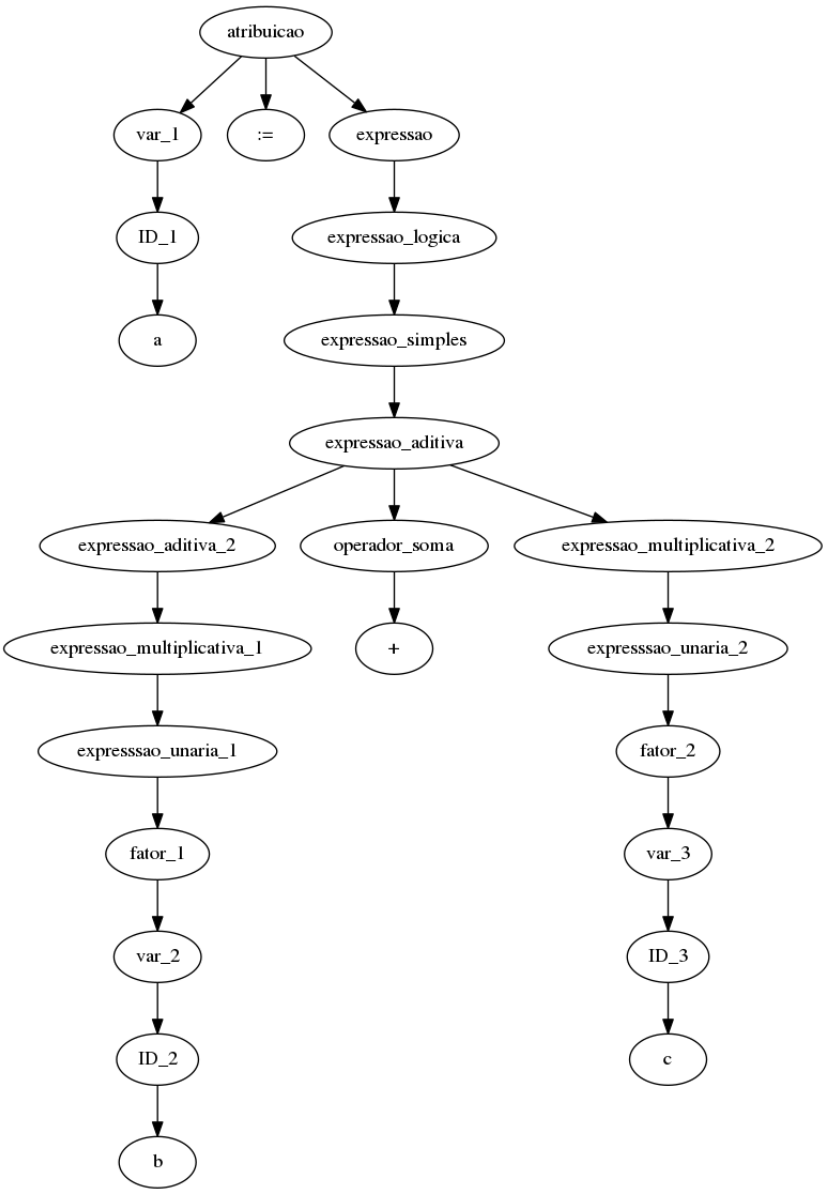

Figura 2: Atribuição expandida

Após a verificação das regras semânticas, pode-se fazer uma poda dos nós internos da árvore para facilitar a geração de código. A Figura 3 apresenta a Árvore Sintática Abstrata simplificada.

O Código que deve ser gerado para uma `atribuicao` é apresentado no Código 1.

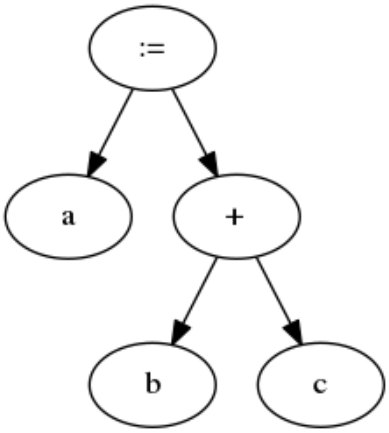

Figura 3: Atribuição depois da poda

```
1  %2 = load i32, i32* @b, align 4
2  %3 = load i32, i32* @c, align 4
3  %4 = add nsw i32 %2, %3
4  store i32 %4, i32* @a, align 4
```

Código 1: Código Gerado

1.6 Testes

Alguns casos de testes estão disponíveis no moodle institucional junto com essa especificação. Serão executados esses testes e outros testes que o professor julgar necessário durante a avaliação desta parte do trabalho.

1.7 Documentação

Durante toda a disciplina o aluno criará uma documentação formal da implementação do compilador para a linguagem. Sendo os relatório com conteúdo acumulativo, isto é, as fases subsequentes irão complementar o conteúdo existente das fases anteriores. Neste ponto do trabalho o aluno deverá incluir na documentação:

- Estratégias utilizadas para a realização da análise sensível ao contexto;
- Estratégia utilizada para a geração da tabela de símbolos;
- Qualquer decisão de projeto referente à semântica da linguagem T++;

Utilize o formato de artigo da SBC[1] para fazer o relatório.

[1] [1] Formato para publicação de artigos da SBC: http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros

1.8 Avaliação

Será avaliado se a análise sensível ao contexto executa a saída proposta (ASTO) e se todos os erros e *warnings* semânticos são mostrados corretamente para um exemplo simples escrito pelo professor/aluno. O essencial da linguagem deve ser mantido (conforme especificado no primeiro trabalho) para que o mesmo exemplo funcione para testar todos os trabalhos com o mínimo de alteração.

- Serão avaliados, dentre outros critérios:

  a) Da implementação:

  – O funcionamento do programa.
  – O capricho e a organização na elaboração do projeto.
  – A corretude da implementação em relação ao que foi pedido no trabalho.
  – A colocação em prática dos conceitos que foram discutidos em sala de aula de forma correta.
  – A qualidade do projeto e da implementação (descrição e elaboração do projeto e o passo a passo da implementação).

  b) Do relatório:

  – O conteúdo e a forma que foi apresentado, se o formato é o mesmo solicitado.
  – Organização das ideias e do processo de tradução.
  – O capricho na elaboração e na formatação do texto, bem como o conteúdo do texto.

- Não serão avaliados os trabalhos:

  a) Que chegarem fora do prazo.
  b) Que não forem feitos nas ferramentas solicitadas.
  c) Que não estão no formato especificado.
  d) Que não foram compactados em um só arquivo.
  e) Que não tiverem identificação (nome e matrícula).
  f) Que forem cópias de outros trabalhos ou materiais da internet.
  g) Que não seguirem todas estas instruções.

- Não se esqueça que o trabalho contribui com $1,5$ ponto da nota.

1.9 Entrega e apresentação

O trabalho será **individual** e deverá ser entregue até o dia **07/11/2019** no moodle da disciplina em um pacote compactado e apresentado no mesmo dia. A estrutura do projeto com os arquivos do projeto (fonte e relatório) deve ser compactada (zipados) e o arquivo compactado deve ser enviado pelo moodle utilizando a opção de submissão "Trabalho 3a. parte - Análise Semântica", o nome do arquivo compactado deve seguir o padrão de nomes do formato.

Deverá ser especificado na entrega o mecanismo de execução da varredura para a realização da correção.

Obs.: Favor utilizar ZIP como forma de compactação. O RELATÓRIO DEVE SER TAMBÉM ENTREGUE IMPRESSO, NO HORÁRIO DA AULA, PARA O PROFESSOR.

Referências

AHO, Alfred V., Monica S. LAM, Ravi SETHI, e Jeffrey D. ULLMAN. 2008. *Compiladores: princípios, técnicas e ferramentas*. 2 ed. São Paulo, SP: Pearson Addison-Wesley.

JARGAS, Aurélio Marinho. 2012. *Expressões regulares: uma abordagem divertida*. 4 ed. São Paulo, SP: Novatec.

LOUDEN, Kenneth C. 2004. *Compiladores: Princípios e Práticas*. 1st ed. São Paulo, SP: Thomson.