Compiladores
Relatório do trabalho prático – Analisador Semântico e tabela de símbolos

Alline Santos Ferreira
Arthur Miranda Moreira

1. Introdução

Quando utilizamos um software não pensamos no que acontece em um computador para que o resultado final seja o programa e suas funções. Na disciplina de Compiladores, estamos aprendendo que existe um processo por trás do software que a partir de um código fonte, construído por um programador, há um programa que faz a tradução desse código, em um formato que o computador seja capaz de executar e, como efeito final dessa execução é exibido o programa para que o usuário final possa utilizar.

Na disciplina de Compiladores, aprendemos mais detalhadamente como essa tradução ocorre. Sabemos que a estrutura deste software é dividida em análise e síntese. Na primeira parte, o programa fonte passa por uma fase de análise da estrutura gramatical. Se ela estiver incorreta, conforme é definido pela linguagem de programação utilizada para sua construção, o compilador exibe mensagem de erro para que o programador possa identificar e corrigir. Uma vez que essa fase de análise é feita com sucesso, então, na segunda etapa, denominada síntese, o compilador constrói o programa a partir do reconhecimento das informações analisadas e, em seguida, gera o executável que é o programa utilizado pelo usuário final.

Este trabalho visa, entendermos na prática, o funcionamento do compilador. Para isso, o trabalho proposto na disciplina é dividido em três etapas: Analisador Léxico, Analisador Sintático, Analisador Semântico e Tabela de Símbolos, o que compõe o frontend do compilador.

Neste relatório, focaremos na construção da **terceira etapa** do trabalho: o **Analisador Semântico e Tabela de Símbolos**. O Analisador Semântico é responsável por verificar a consistência do programa fonte com a definição da linguagem. Ele também reúne informações sobre tipos e as salva na Tabela de Símbolos. É através do Analisador Sintático (construído na etapa 2) que as validações realizadas pelo Analisador Semântico serão realizadas, na forma de ações semânticas. As validações são verificação de tipos e verificação de unicidade. Como todos os identificadores são variáveis, não é necessário fazer verificação de classe.

A seguir, detalharemos a construção do analisador semântico e da tabela de símbolos e as modificações necessárias no analisador léxico e sintático, desenvolvido na primeira e segunda etapa.

2. Desenvolvimento

2.2 Descrição da implementação

Para a implementação do Analisador Semântico, utilizamos a linguagem de programação Java. E para execução do programa é necessário informar o caminho onde se encontra o arquivo de teste. Ao digitar o caminho válido, o arquivo é aberto pelo analisador sintático. Ele instancia um analisador léxico e realiza chamadas para coletar o fluxo de tokens, verificando se estão coerentes com as regras gramaticais da linguagem de programação dada. Neste momento ocorre também a análise semântica para verificar se os operadores presentes na gramática possuem operandos compatíveis e se o identificador é definido uma única vez. Caso contrário será enviado mensagens de erro identificando a falha.

2.2.1 Classes para o Analisador

Para desenvolver o Analisador Semântico, adicionamos no programa desenvolvido nas etapas 1 e 2, as classes **SymbTable, SymbTableException**, **SemanticException** e **Node**.

A classe **SymbTable** tem como finalidade construir a tabela de Símbolos, é nela que o Analisador Semântico registra as informações sobre os tipos para uso futuro. Nesta classe é construída uma tabela de símbolos para que possa suportar diferentes tipos de escopos. A tabela de símbolos foi implementada como uma lista de Hashmaps que contém o nome de cada identificador e seu tipo. A classe contém métodos para iniciar um novo escopo; remover o último escopo; inserir uma variável no último escopo adicionado, e pesquisar um identificador na tabela (há dois métodos de pesquisa, um deles busca apenas no escopo mais recente, o outro pesquisa no mais recente, e não encontrando, busca nos escopos mais exteriores. O primeiro é usado para buscar as variáveis no momento da declaração, pois só pode haver uma variável com determinado lexema por escopo, mas em diferentes escopos o identificador pode se repetir. O segundo é usado ao utilizar uma variável, deve ser verificado se ela foi declarada em algum bloco, procurando do mais interno ao mais externo).

As classes **SymbTableException** e **SemanticException** indicam erros semânticos e na atualização na tabela de Símbolos. Como regra, este compilador deve permitir atribuições quando o tipo da expressão é compatível com a variável. E, para as operações aritméticas, devem ocorrer respeitando as seguintes regras: Caso um operador seja real, o resultado deve ser um real. Quando ambos forem inteiros, o resultado deve ser inteiro, quando a operação ocorrer entre um caractere e outro inteiro, o resultado também deve ser inteiro. Para esse tipo de operação, o tipo real e caractere são discordantes. Também não são permitidos dois operandos do tipo caractere. Já para operadores relacionais (>, <=, ==, etc.) não há restrições.

Foi implementada uma classe **Node** (nó), que possui um atributo para guardar informações de tipo. Os não terminais (produções) instanciam nodes para transmitir o atributo tipo. Quando um pai necessita da informação dos tipos dos filhos, os nós são retornados para o pai, que faz as verificações necessárias. Quando é necessário herdar essa informação (em específico, no momento em que uma variável é declarada e é necessário armazenar seu tipo na tabela de símbolos), o nó é enviado para o filho como argumento da função que implementa a produção.

Os códigos deste compilador (em todas as etapas) foram inspirados na obra dos autores Alfred Aho, Monica Lam, Ravi Sethi e Jeffrey Ullman.

No próximo capítulo serão indicados os resultados dos testes validando a construção do Compilador.

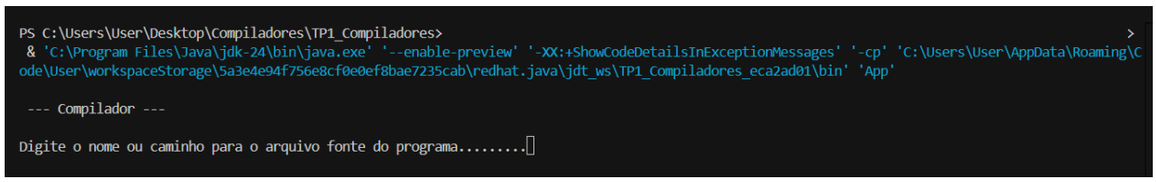
3. Resultados

Para executarmos o programa a fim de avaliarmos a criação do Analisador Semântico, foram disponibilizados pela professora 5 arquivos testes. O objetivo era verificar se o compilador construído, reconheceria os erros, léxicos, semânticos e sintáticos existentes, identificando através de uma mensagem em qual linha houve o erro e abortando a execução. Em seguida, os erros seriam corrigidos pelo usuário e ao repetir a execução, ele deveria avaliar novamente e com sucesso. Nessa etapa, o compilador deve ser capaz de identificar todos os tipos de erros.

A seguir mostraremos os resultados de execução de todos os arquivos testes. E outro teste feito pelos autores do trabalho.

3.1 Início da execução

Informar o caminho onde está salvo o arquivo teste.



3.2 Programas de entrada e saída do compilador

Para essa etapa do trabalho, foram coletados os códigos de teste com os erros léxicos e sintáticos já corrigidos, produzidos ao fim da segunda etapa do trabalho. Dessa forma, os testes são feitos com ênfase nos erros semânticos.

Teste do Programa 1:

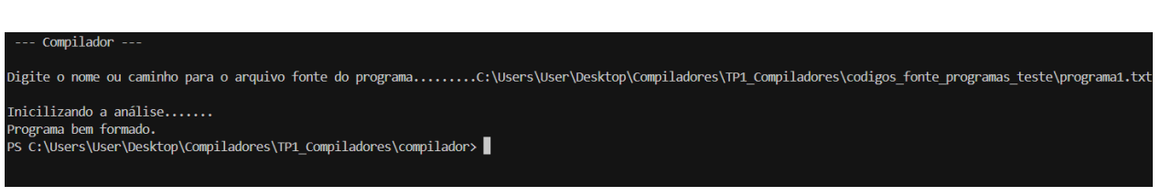
```
program
   int: a,b,c;
   float: result;
   char: ch;

begin
   out("Digite o valor de a:");
   in (a);
   out("Digite o valor de c:");
   in (ch);
   b = 10;
   result = (a * ch)/(b - 345.27); % atribuição permitida, a * ch é inteiro, b - 345.27 é real, inteiro /
                                   %real retorna um tipo real.
   out("O resultado e: ");
   out(result);
   result = result + ch;           % operação não permitida, inteiro e char são incompatíveis

end
```

Resultado do teste 1:

Ocorre erro na linha 15, ao tentar somar um inteiro com um caractere. Ao alterar o tipo da variável ch para inteiro, na linha 3

```
   int: ch;
```

e executar novamente:

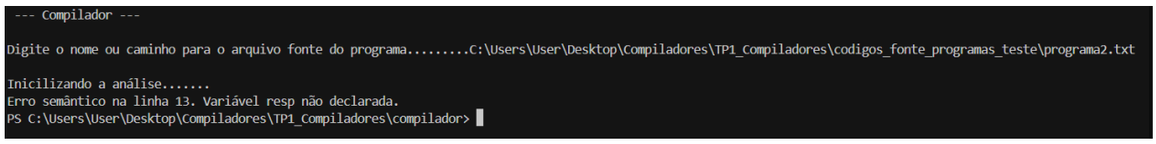


O programa é aceito.

Teste do Programa 2:

```
program
   float: raio, area;

begin
   repeat
   char: resposta;
   in(raio);
   if (raio > 0.0) then
      area = 3.0 * raio * raio;
      out (area);
   end;
   out ("Deseja continuar?");
   in (resp);                                % resp não declarada
until ((resp == 'N') || (resp == 'n'));

end
```

Resultado do teste 2:



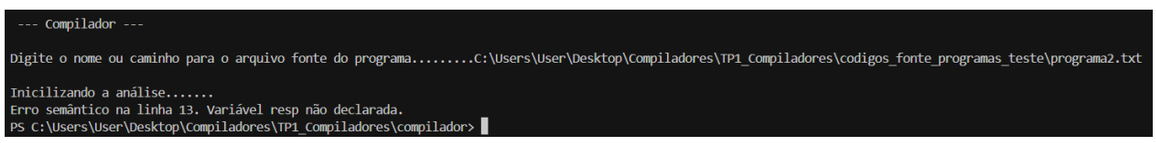
Ocorre um erro na linha 13, pois a variável resp não foi declarada. Substituindo na linha 5 a declaração

```
   char: resposta;
```

por

```
   char: resp;
```

e executando:



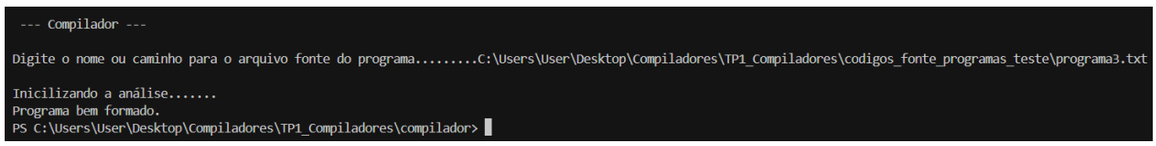
O programa é aceito.

Teste do Programa 3:

```
program

   int: a, b, aux;

 begin
   in (a);
   in(b);
   if (a>b) then
```

```
        int: aux;                   % essa atribuição é permitida pois está em outro bloco
        aux = b;
        b = a;
        a = aux;
    end;
    out(a);
    out(b);
 end
```

Resultado do teste 3:

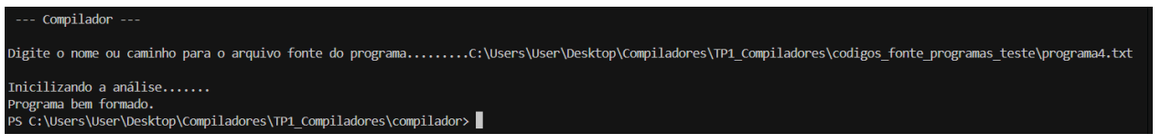


O programa é aceito.

Teste do Programa 4:

```
program
Int:  a, b, c, maior, outro;
 begin
 repeat
    out("A");
    in(a);
    out("B");
    in(b);
    out("C");
    in(c);
    %Verifica o maior
    if ( (a>b) && (a>c) ) then
        maior = a;

    else
        if (b>c) then
            maior = b;

        else
             maior = c;
        end;
    end;
    out("Maior valor: ");
    out (maior);
    out ("Outro? ");
    in(outro);
until (outro == 0);
 end
```

Resultado do teste 4:

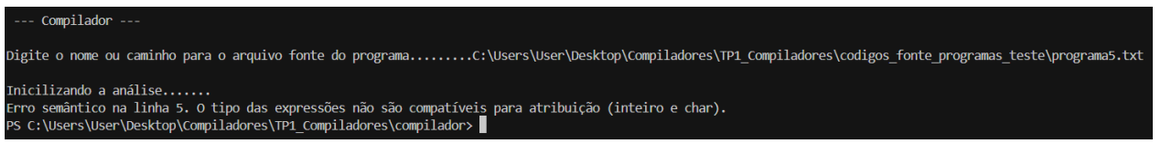


Programa aceito.

Teste do Programa 5:

```
program
   int:pontuacao, pontuacaoMaxina, disponibilidade;
   char: pontuacaoMinima;
begin
   disponibilidade = 'S';
   pontuacaoMinima = 50;
   pontuacaoMaxima = 100;
   out("Pontuacao Candidato: ");
   in(pontuacao);
   out("Disponibilidade Candidato: ");
   in(disponibilidade);
   { Comentario
   grande}

   while (pontuacao>0 && (pontuação<=pontuacaoMaxima)) do
      int: cont;
      cont = cont + 1;
      if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
         out("Candidato aprovado");
      else
         out("Candidato reprovado");
      end;

      out("Pontuacao Candidato: ");
      in(pontuacao);
      out("Disponibilidade Candidato: ");
      in(disponibilidade);

 end;
 out (cont);
end
```

Resultado do teste 5:



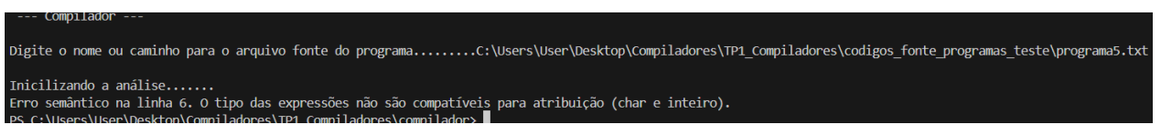
Retornado erro semântico na linha 5. Está sendo atribuído um char para a variável disponibilidade, que tem tipo inteiro. Trocando a linha

disponibilidade = 'S';

por

disponibilidade = 0;

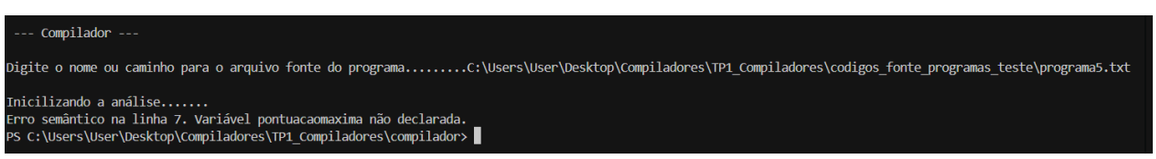
E executando novamente:



É retornado erro na linha 6. A variável pontuacaoMinima é do tipo char, e está sendo atribuído a ela um valor inteiro. Trocando o tipo da variável para inteiro:

int: pontuacaoMinima;

E executando novamente:



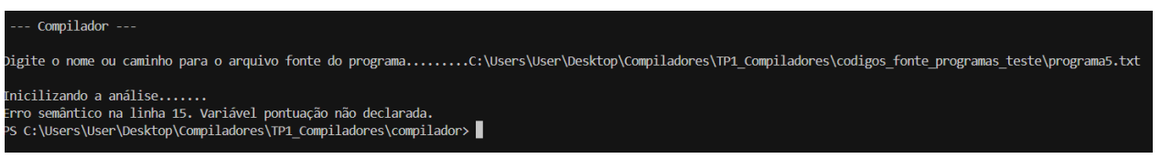
Retornando erro na linha 7. A variável pontuacaoMaxima não foi declarada, e sim a variável pontuacaoMaxina. Substituindo a declaração:

int:pontuacao, pontuacaoMaxina, disponibilidade;

por

int:pontuacao, pontuacaoMaxima, disponibilidade;

E executando:

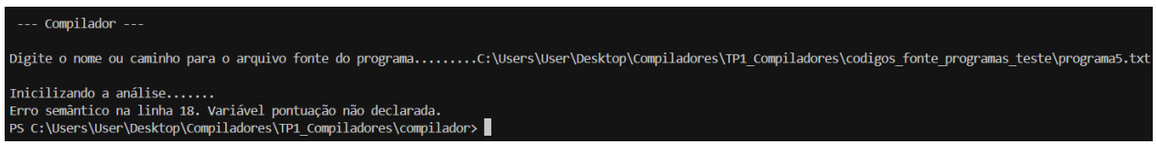


Ocorre erro na linha 15. A variável pontuação não foi declarada, apenas a variável pontuacao. Substituindo:

while (pontuacao>0 && (pontuação<=pontuacaoMaxima)) do

por

while (pontuacao>0 && (pontuacao<=pontuacaoMaxima)) do

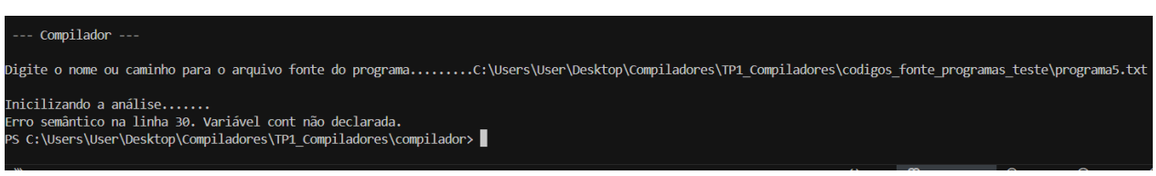


Ocorre erro na linha 18. Novamente, a variável pontuação não foi declarada, apenas a variável pontuacao. Substituindo:

if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then

por

if ((pontuacao > pontuacaoMinima) && (disponibilidade==1)) then



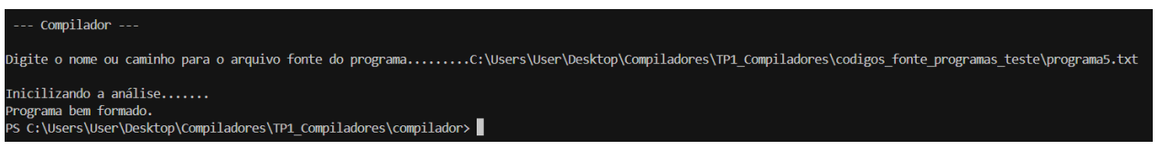
Ocorre erro na linha 30, a variável cont foi declarada, porém no bloco do while, que já foi fechado. Movendo a linha out(cont) para antes do end (onde o bloco é fechado): Trocando

```
end;
out (cont);
```

Por:

```
out (cont);
end;
```

E executando novamente.

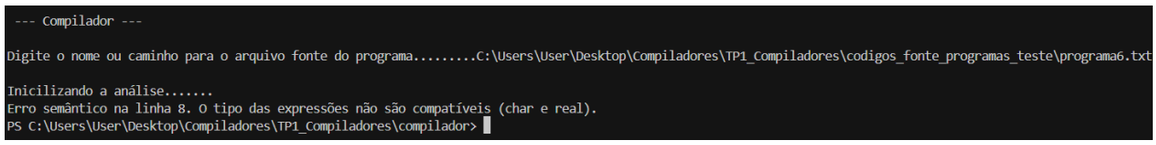


O programa é aceito.

Teste do Programa 6: O programa 6 foi alterado para um que teste erros semânticos. A motivação é avaliar os tipos de expressões. Devido a precedência com o parêntese, a expressão b + c é calculada primeiro, e o tipo resultante é inteiro.

```
program
        char: a;
        int: b;
        float: c;
        float: result;
begin
        out("Testando promoção de tipos :");
        result = a+(b+c);
end
```

Resultado do teste 6:



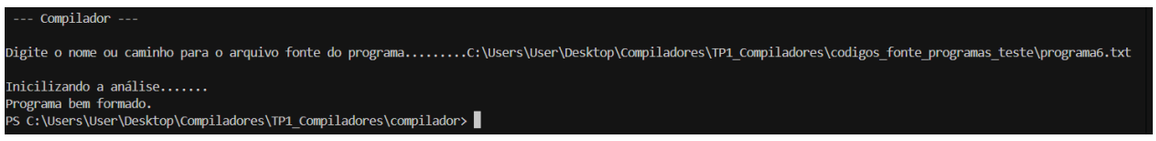
Ocorre erro de tipos, pois os tipos char e real são incompatíveis. Alterando a precedência dos parênteses, trocando

result = a+(b+c);

Por:

result = (a+b)+c;

E executando novamente:



O programa é aceito. Ao calcular a expressão a + b antes, a expressão resultante tem tipo inteiro, que é compatível com real. Ao fazer (a+b) + c, o tipo resultante é real, sendo permitida a atribuição de result.

4. Conclusão

Compreender o funcionamento de um compilador é de suma importância para estudantes e futuros profissionais da área de computação. É fundamental saber que cada linguagem de programação possui sua própria gramática. Durante o processo de compilação, o programa é avaliado para garantir que sua estrutura esteja coerente com as regras de análise léxica, sintática e semântica dessa gramática. Essa compreensão é crucial não apenas para escrever código correto e robusto, mas também para entender a origem dos erros reportados durante a fase de compilação de um programa em desenvolvimento.

A elaboração deste trabalho proporcionou uma experiência prática valiosa, permitindo vivenciar como são realizadas as etapas de Análise Léxica, Sintática e Semântica, que compõem a maior parte do frontend de um compilador.

5. Referência Bibliográfica

1. AHO, A. V. Jeffrey; Sethi, R.; Lam, MS (2008). **Compiladores. Princípios, técnicas e ferramentas.** São Paulo: Addison-Wesley, Pearson.