**INSTITUTO TECNOLÓGICO DE AERONÁUTICA - ITA**

DIVISÃO DE CIÊNCIA DA COMPUTAÇÃO

**CES-41 - Compiladores**



**4º Laboratório**

**Analisador Semântico para a**

**Linguagem COMP-ITA 2016**

**ALUNO**

Felipe Tuyama de Faria Barbosa ftuyama@gmail.com

Luiz Angel Rocha Rafael luizangel.100@gmail.com

**PROFESSOR**

Fábio Carneiro Mokarzel mokarzel@ita.br

São José dos Campos, 01 de Junho de 2016

**1. Organização das tarefas**

Na execução desta atividade foi criado um pequeno script Batch para automatizar a execução das ferramentas Yacc e Lex para o programa desejado e mais uma bancada de “Testes Unitários” para diversas situações semânticas da Linguagem COMP ITA 2016.

Esse script tem também a opção de trabalhar indefinidamente (até uma interrupção do usuário), atualizando o resultado a cada dez segundos, garantindo assim que a saída ficará sempre sincronizada com os arquivos .L e .Y programados. Abaixo mostro o funcionamento da rotina:

@**echo off**

**echo** ######################################

**echo** # Analisador Semântico #

**echo** # para Linguagem COMP-ITA 2016 #

**echo** ######################################

:loop

**cd** ..

**echo** Chamando Flex

flex Compilador/\_Lab04.l

**echo** Chamando Yacc

yacc -v Compilador/\_Lab04.y

**echo**.

**echo** Compilando arquivo

gcc y.tab.c main.c yyerror.c -o \_Lab04 -lfl

**echo**.

**echo** Compilando programa

**REM** \_Lab04 < Compilador/\_Lab04Teste.dat > Compilador/\_Lab04TesteOutput.dat

**REM** \_Lab04 < Compilador/\_Lab04Prog.dat > Compilador/\_Lab04ProgOutput.dat

**REM** goto Testes

**echo**.

**echo** ### Testes para Declaração de Variáveis

\_Lab04 < Compilador/Bench/BenchDeclaracao.dat > Compilador/Bench/oBenchDeclaracao.dat

**echo**.

**echo** ### Testes para função Principal ###

\_Lab04 < Compilador/Bench/BenchPrincipal.dat > Compilador/Bench/oBenchPrincipal.dat

**echo**.

**echo** ### Testes para Variáveis Indexadas

\_Lab04 < Compilador/Bench/BenchIndexada.dat > Compilador/Bench/oBenchIndexada.dat

**echo**.

**echo** ### Testes para Compatibilidade

\_Lab04 < Compilador/Bench/BenchCompatibilidade.dat > Compilador/Bench/oBenchCompatibilidade.dat

**echo**.

**echo** ### Testes para Para

\_Lab04 < Compilador/Bench/BenchPara.dat > Compilador/Bench/oBenchPara.dat

**echo**.

**echo** ### Testes para CallFunc

\_Lab04 < Compilador/Bench/BenchCallFunc.dat > Compilador/Bench/oBenchCallFunc.dat

**echo**.

**echo** ### Testes para RetornoFunc

\_Lab04 < Compilador/Bench/BenchRetornoFunc.dat > Compilador/Bench/oBenchRetornoFunc.dat

:Testes

**REM** Loop compilation with delay

**REM** timeout /t 10

**REM** goto loop

**echo**.

**echo** Completo

**pause**

Código 1: Script Batch de automatização de testes unitários semânticos.

Os arquivos Yacc e Lex são reaproveitados da última atividade de laboratório, mantendo as suas funcionalidades de analisador léxico e sintático (incluindo o Pretty Printer desenvolvido). Nesta atividade o arquivo Yacc será incrementado para realizar também análise semântica para a Linguagem COMP-ITA 2016, partindo dos moldes idealizados em sala de aula.

**2. Resultados**

A fim de verificar exaustivamente o cumprimento dos requisitos semânticos do Analisador Semântico desenvolvido, foram criadas 7 bancadas temáticas de testes unitários para o programa:

|  |  |
| --- | --- |
| **Testes** | **Arquivo** |
| Declaração de Variáveis | BenchDeclaracao.dat |
| Função Principal | BenchPrincipal.dat |
| Compatibilidade | BenchCompatibilidade.dat |
| Variáveis Indexadas | BenchIndexada.dat |
| Comandos Se, Enquanto e Para | BenchPara.dat |
| Chamada de Função | BenchCallFunc.dat |
| Retorno de Função | BenchRetornoFunc.dat |

Tabela 1 – Bancadas de testes unitários.

Os requisitos semânticos foram divididos nestes arquivos da seguinte forma:

|  |  |
| --- | --- |
| **Arquivo** | **Requisitos Semânticos** |
| BenchDeclaracao | • Qualquer identificador deve ser declarado antes de usado.  • Um identificador não pode estar declarado mais de uma vez dentro de uma função, ou como global, mas pode estar declarado ao mesmo tempo como global e numa função qualquer, ou em duas ou mais funções quaisquer.  • Identificadores podem ser do tipo nome de variável ou nome de função.  • Variáveis escalares, expressões e elementos de variáveis indexadas podem ser do tipo inteiro, real, caractere ou lógico.  • O tipo de uma variável comum ou de um parâmetro não pode ser o tipo vazio.  • Toda variável escalar e ao menos um elemento de cada variável indexada deve ser inicializado e referenciado pelo menos uma vez no programa. |
| BenchPrincipal | • O programa deve ter uma e uma só função de cabeçalho principal, devendo esta ser a última função desse programa.  • Uma função não pode ter o mesmo nome que o de uma variável global.  • A linguagem não admite recursividade. |
| BenchCompatibilidade | •Deve haver compatibilidade entre os tipos dos dois lados de um comando de atribuição, conforme a seguinte tabela:  Tipo do lado esquerdo Tipo do lado direito  Inteiro Inteiro ou Caractere  Real Inteiro, Real ou Caractere  Caractere Inteiro ou Caractere  Lógico Lógico  •Os tipos dos resultados das diversas classes de expressões só podem ser os seguintes: Classe da expressão Tipo Aritmética Inteiro, Real ou Caractere Relacional Lógico  Lógica Lógico  •Os tipos dos operandos admitidos pelos operadores de expressões são os seguintes: Operadores Tipos admitidos dos operandos  || && ! Lógico  < <= > >= Inteiro, Real ou Caractere  = != Todos (se um for lógico o outro também deve ser)  + - \* / ~ Inteiro, Real ou Caractere  % Inteiro ou Caractere |
| BenchIndexada | • A constante inteira usada no dimensionamento de uma variável indexada deve ser maior do que zero.  • Variáveis escalares não podem ter subscritos.  • O número de subscritos de uma variável indexada deve ser igual ao seu número de dimensões declarado.  • Os elementos de uma variável indexada só poderão ser atribuídos ou receber atribuição um de cada vez.  • Os elementos de uma variável indexada só poderão ser lidos, ou escritos um de cada vez.  • A expressão aritmética no subscrito de uma variável indexada deve ser do tipo inteiro ou caractere. |
| BenchPara | • As expressões nos cabeçalhos de comandos se e enquanto e no encerramento de comandos repetir devem ser relacionais ou lógicas.  • A variável da inicialização do cabeçalho de um comando para deve ser escalar do tipo inteiro ou caractere.  • A variável da atualização do cabeçalho de um comando para deve ser a mesma daquela de sua inicialização.  • A primeira e a terceira expressão de um comando para deve ser do tipo inteiro ou caractere e a segunda expressão deve ser do tipo lógico. |
| BenchCallFunc | • O identificador de uma chamada de procedimento e o identificador de uma chamada de função deve ser do tipo nome de função.  • Um identificador de variável e de parâmetro deve ser do tipo nome de variável.  • O tipo de variável correspondente ao identificador de uma chamada de procedimento deve ser o tipo vazio.  • O tipo de variável correspondente ao identificador de uma chamada de função não deve ser o tipo vazio.  • O número de argumentos na chamada de uma função ou procedimento deve ser igual ao número de parâmetros do mesmo.  • Deve haver compatibilidade entre um argumento de chamada de uma função ou procedimento e seu parâmetro correspondente, conforme a seguinte tabela:  Tipo do parâmetro Tipo do argumento  Inteiro Expressão inteira ou caractere  Real Expressão inteira, real ou caractere  Caractere Expressão inteira ou caractere  Lógico Expressão de valor lógico |
| BenchRetornoFunc | • Todo comando retornar dentro de uma função de tipo vazio não deve ser seguido de expressão e dentro de funções de outros tipos deve ser seguido por uma expressão.  • Deve haver compatibilidade entre o tipo de uma função e o tipo da expressão de qualquer comando retornar em seu escopo, conforme a seguinte tabela:  Tipo da função Tipo da expressão retornada  Inteiro Inteiro ou Caractere  Real Inteiro, Real ou Caractere  Caractere Inteiro ou Caractere  Lógico Lógico  • Funções não são usados como parâmetros ou argumentos de chamada de outras funções. |

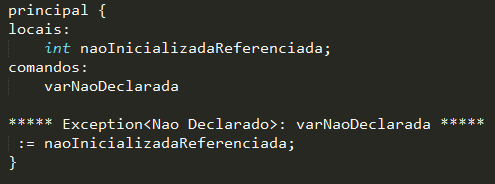
Tabela 2 – Divisão de requisitos semânticos por bancada de teste.

A seguir, serão exibidos por tópicos alguns exemplos de trechos de código com erros semânticos detectados pelo analisador semântico (dentro do formato padronizado adotado por nós: Exceptions<Tipo do Erro>). Para maiores detalhes do contexto destes trechos de código, indicamos a leitura dos arquivos de teste “Bench(...)” e dos arquivos de saída destes testes “oBench(...)”.

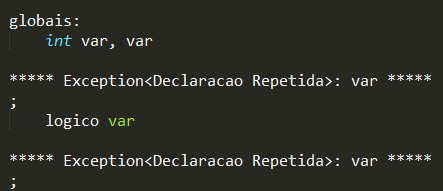
Os testes completos podem ser verificados nos arquivos .dat. Foram exibidos neste relatório somente alguns exemplos significativos de Exceptions.

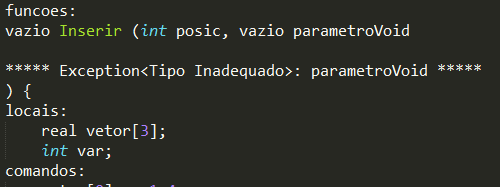
**2.1 BenchDeclaracao**

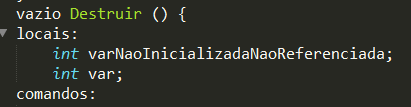
* **Qualquer identificador deve ser declarado antes de usado.**



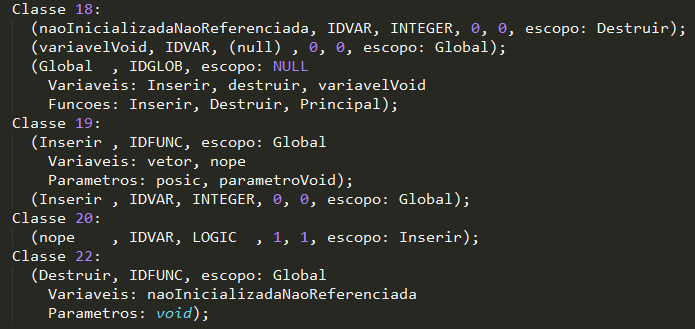
* **Um identificador não pode estar declarado mais de uma vez dentro de uma função, ou como global, mas pode estar declarado ao mesmo tempo como global e numa função qualquer, ou em duas ou mais funções quaisquer.**



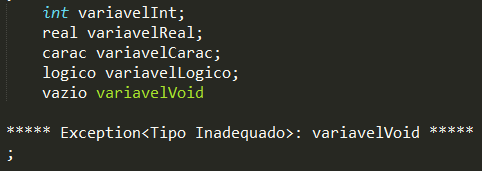




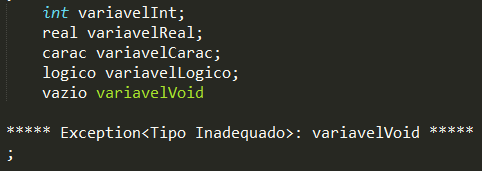
* **Identificadores podem ser do tipo nome de variável ou nome de função.**

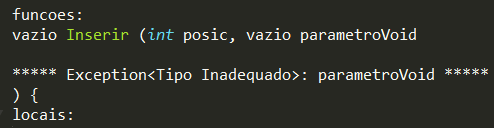


* **Variáveis escalares, expressões e elementos de variáveis indexadas podem ser do tipo inteiro, real, caractere ou lógico.**

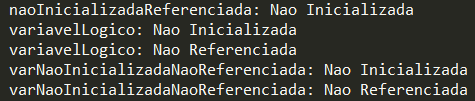


* **O tipo de uma variável comum ou de um parâmetro não pode ser o tipo vazio.**



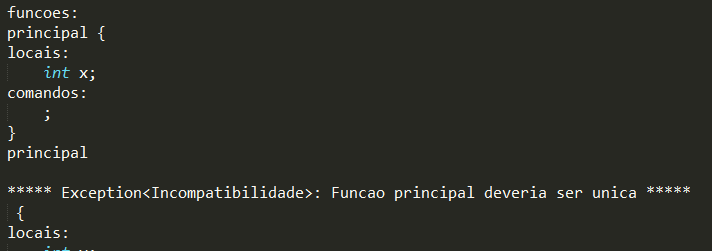


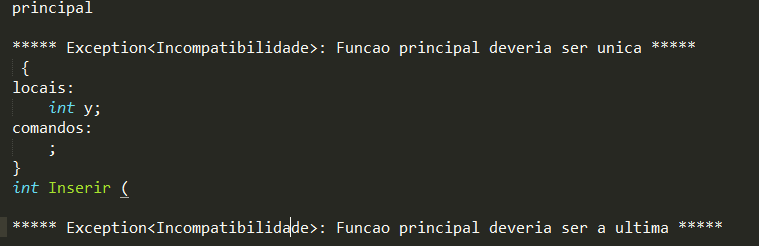
* **Toda variável escalar e ao menos um elemento de cada variável indexada deve ser inicializado e referenciado pelo menos uma vez no programa.**



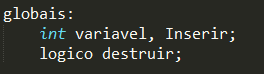
**2.2 BenchPrincipal**

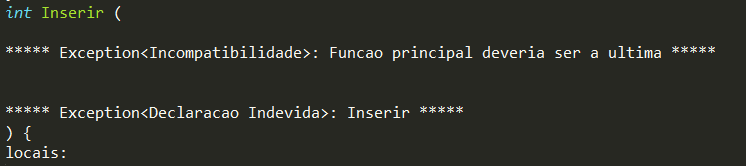
* **O programa deve ter uma e uma só função de cabeçalho principal, devendo esta ser a última função desse programa.**

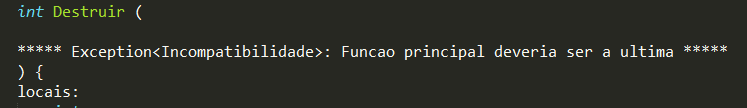




* **Uma função não pode ter o mesmo nome que o de uma variável global.**

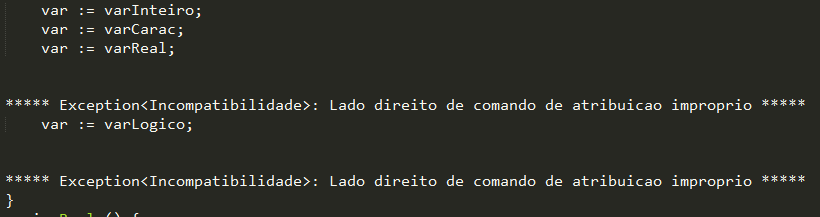




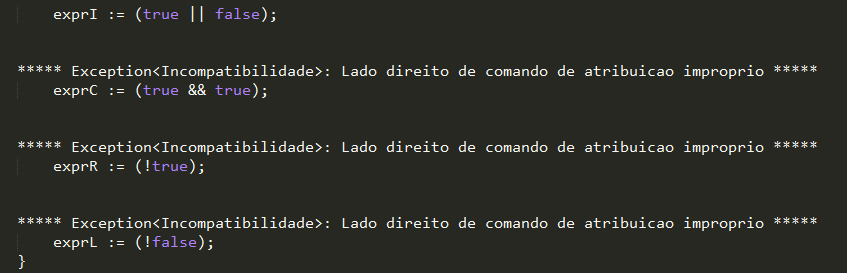


**2.3 BenchCompatibilidade**

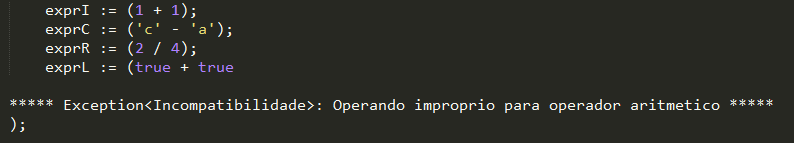
* **Deve haver compatibilidade entre os tipos dos dois lados de um comando de atribuição, conforme a seguinte tabela:**



* **Os tipos dos resultados das diversas classes de expressões só podem ser os seguintes:**

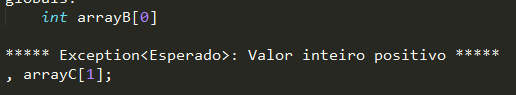


* **Os tipos dos operandos admitidos pelos operadores de expressões são os seguintes:**

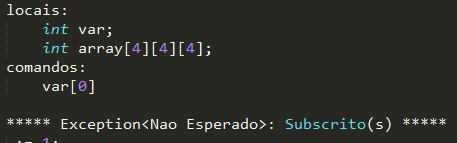


**2.4 BenchIndexada**

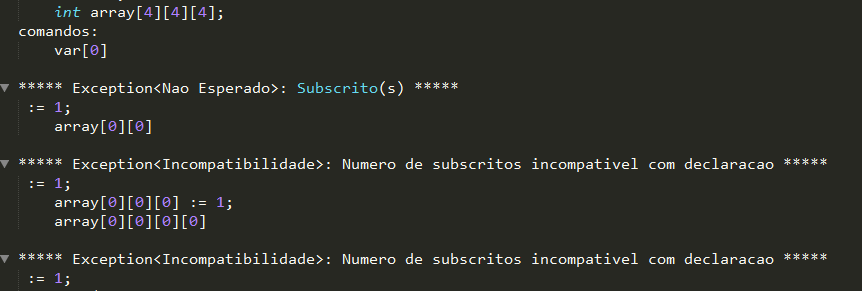
* **A constante inteira usada no dimensionamento de uma variável indexada deve ser maior do que zero.**



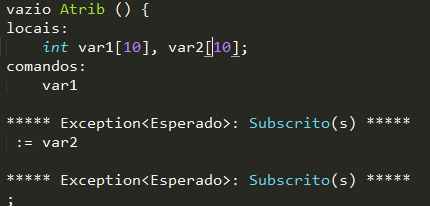
* **Variáveis escalares não podem ter subscritos.**



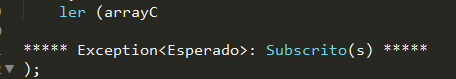
* **O número de subscritos de uma variável indexada deve ser igual ao seu número de dimensões declarado.**

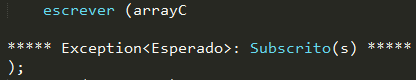


* **Os elementos de uma variável indexada só poderão ser atribuídos ou receber atribuição um de cada vez.**

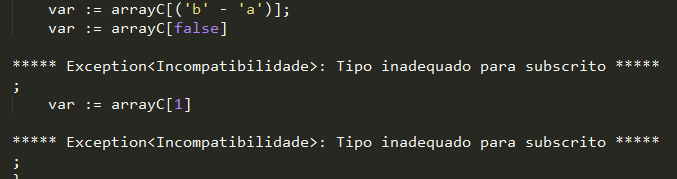


* **Os elementos de uma variável indexada só poderão ser lidos, ou escritos um de cada vez.**



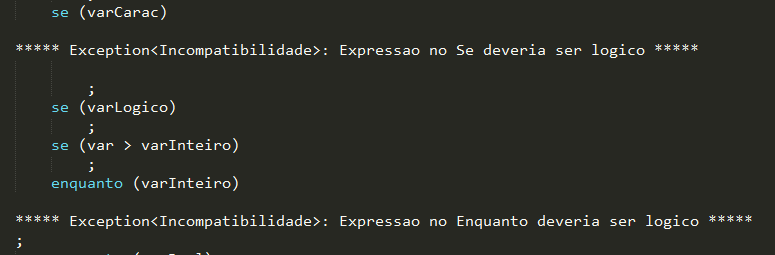


* **A expressão aritmética no subscrito de uma variável indexada deve ser do tipo inteiro ou caractere.**

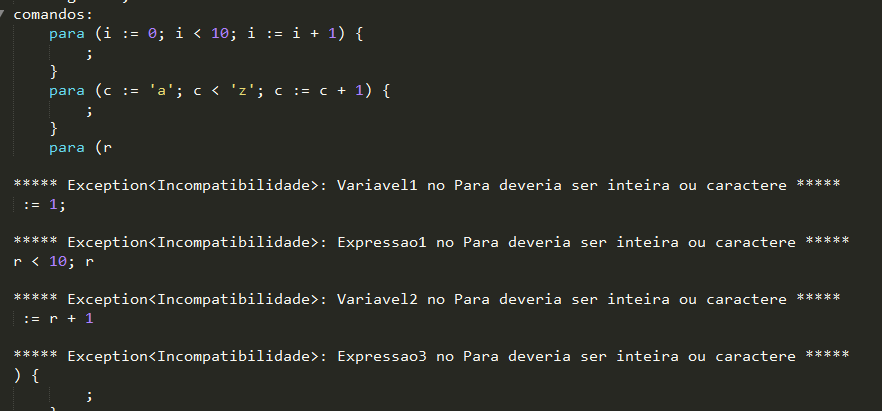


**2.5 BenchPara**

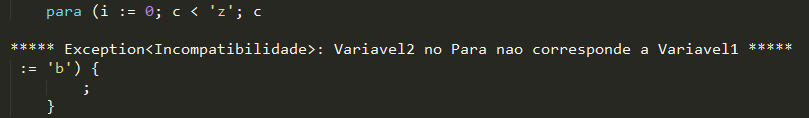
* **As expressões nos cabeçalhos de comandos se e enquanto e no encerramento de comandos repetir devem ser relacionais ou lógicas.**



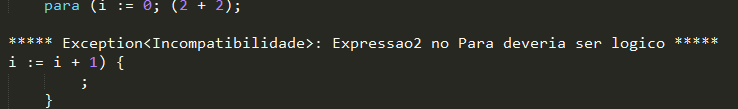
* **A variável da inicialização do cabeçalho de um comando para deve ser escalar do tipo inteiro ou caractere.**

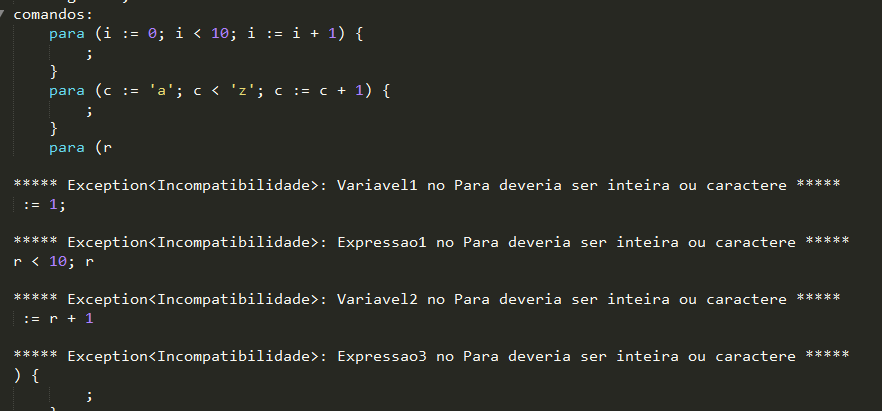


* **A variável da atualização do cabeçalho de um comando para deve ser a mesma daquela de sua inicialização.**



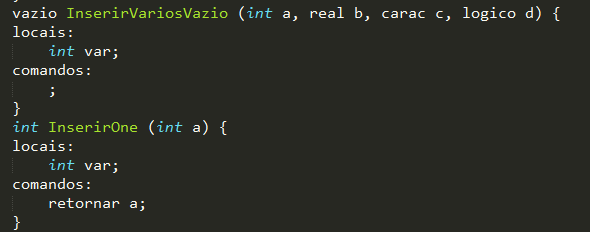
* **A primeira e a terceira expressão de um comando para deve ser do tipo inteiro ou caractere e a segunda expressão deve ser do tipo lógico.**

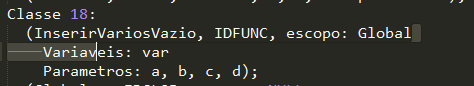


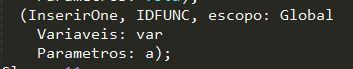


**2.6 BenchCallFunc**

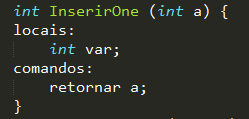
* **O identificador de uma chamada de procedimento e o identificador de uma chamada de função deve ser do tipo nome de função.**







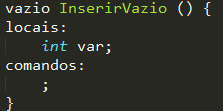
* **Um identificador de variável e de parâmetro deve ser do tipo nome de variável.**

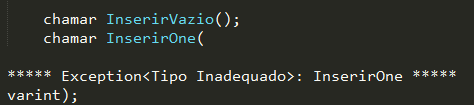




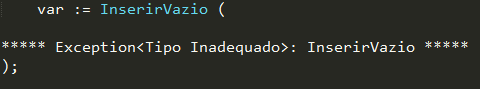


* **O tipo de variável correspondente ao identificador de uma chamada de procedimento deve ser o tipo vazio.**

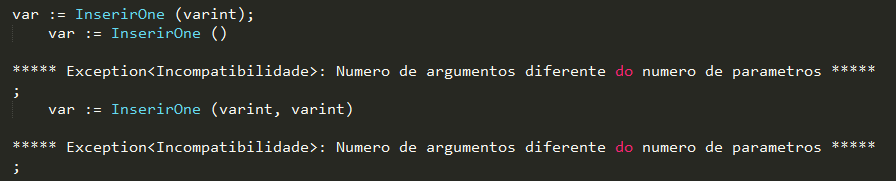


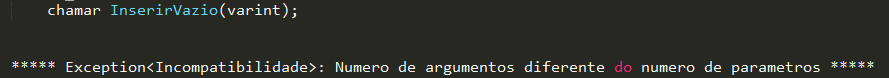


* **O tipo de variável correspondente ao identificador de uma chamada de função não deve ser o tipo vazio.**

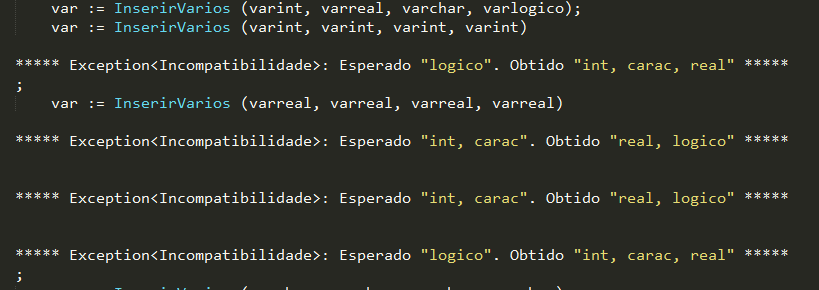


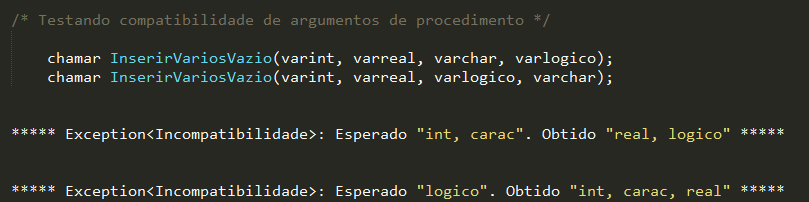
* **O número de argumentos na chamada de uma função ou procedimento deve ser igual ao número de parâmetros do mesmo.**





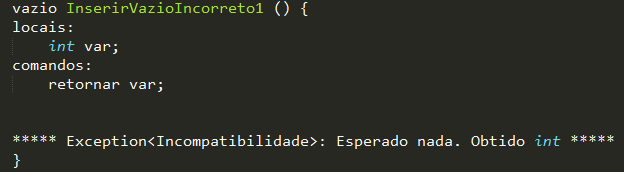
* **Deve haver compatibilidade entre um argumento de chamada de uma função ou procedimento e seu parâmetro correspondente, conforme a seguinte tabela:**

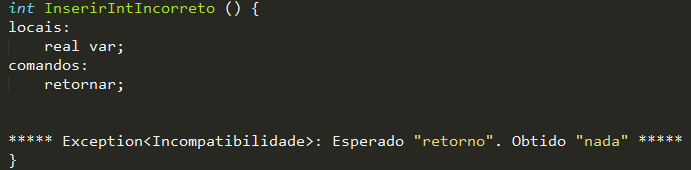




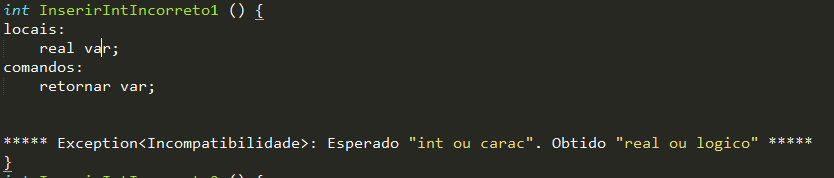
**2.7 BenchRetornoFunc**

* **Todo comando retornar dentro de uma função de tipo vazio não deve ser seguido de expressão e dentro de funções de outros tipos deve ser seguido por uma expressão.**

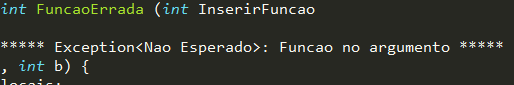


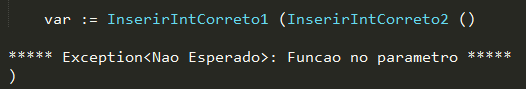


* **Deve haver compatibilidade entre o tipo de uma função e o tipo da expressão de qualquer comando retornar em seu escopo, conforme a seguinte tabela:**

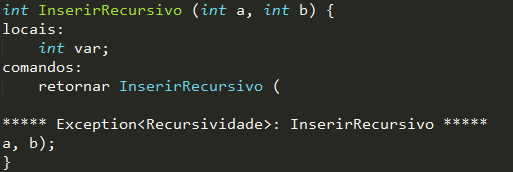


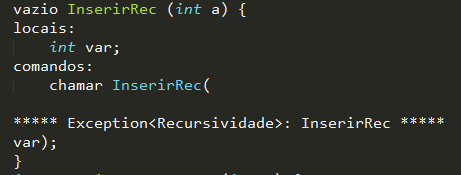
* **Funções não são usados como parâmetros ou argumentos de chamada de outras funções.**





* **A linguagem não admite recursividade.**





**3. Conclusão**

A análise semântica foi bem-sucedida para os casos de teste idealizados, conforme o esperado para os requisitos estabelecidos da Linguagem COMP ITA 2016. Assim, a ferramenta está pronta para ser reaproveitada nas próximas experiências de laboratório.