Rascal Compiler

William D. Costa RA 89239

Mateus Soares RA 90253

Descrição do Projeto:

Para o desenvolvimento das várias etapas que compõem um compilador, foram utilizadas as seguintes ferramentas:

- JFlex - Ferramenta geradora de analisador léxico para linguagem Java (equivalente ao Flex, visto em aula);
- CUP - Ferramenta geradora de analisador sintático para linguagem Java (equivalente ao Bison, visto em aula);

Foi optado pelo uso da liguagem Java, por questões de familiaridade da equipe. Sendo assim, varias outras ferramentas de organização de código como Maven (gerenciador de dependências) e JUnit (para testes) foram empregados.

Visão Geral do Projeto

Existem duas classes executáveis dentro do projeto, uma Service e uma Main. A Service é usada para fazer a chamada das ferramentas JFlex e CUP, que recebem seus respectivos arquivos *lexico.flex* e *sintatico.cup*. A execução desse módulo gera duas classes contendo os analisadores, além de uma tabela de simbolos. Os arquivos de interesse estão no seguinte diretório:

```
rascalJavaCompiler/src/main/resources/jflex/lexico.flex
rascalJavaCompiler/src/main/resources/sintatico.cup
```

Por entender que traria maior organização ao desenvolvimento, as fases de análise semântica e geração de código foram implementadas de maneira separada. Sendo assim, as validações necessárias são feitas, e caso nenhum erro seja detectado, uma tabela de simbolos globais, e uma tabela de simbolos referente aos subprogramas, são geradas.

O gerador de código Mepa recebe essas tabelas juntamente com a AST devidamente validada, e a partir daí gera as instruções Mepa.

Validações realizadas pelo analisador semântico:

Seguem abaixo as mensagens de erro lançadas

Declaração de tipo inválido

- "Tipo [tokenTipo] não suportado pela linguagem!"

Chamada de sub programa não definido

- "Função ou procedimento [nomeSub] não declarado"

Chamada de subprograma com número errado de parâmetros

- "Chamada de [nomeSub] com número inválido de parâmetros [numParamChamada] (esperado(s): [numParamDeclarado] parâmetro(s))"

Chamada da função *read* com função composta no parâmetro

- "A função 'read' não aceita expressões compostas"

Tentativa de atribuição de expressão para variáveis que não sejam do tipo integer

- A variavel [nomeVariável] não aceita operações aritméticas

Tentativa de uso de variável que não foi previamente declarada

- "Variável [nomeVariável] não foi declarada no escopo"

Declaração de variáveis com nome repetido

- "Declaração de variáveis com o mesmo nome!"

Declaração de variável em escopo local com mesmo nome de variável em escopo global

- "A variável [nomeVariável] já existe no escopo como um parâmetro"

Instruções de Execução

Geração do arquivo executável do compilador

Esta compilação é feita pelo Maven, logo, é necessário que o **Apache Maven v3.6.1** (ou versão compatível) esteja instalado.

Considerando que a aplicação foi desenvolvida em Java, é necessária a versão 1.8 do mesmo instalada no ambiente.

Cumpridas essas exigências, basta que seja executado o comando *mvn clean install* na raiz do projeto. a pasta *target* será criada, contendo o executável jar para a aplicação.

Lembrando que este executável está disponível nos documentos entregues.

Execução do compilador processando arquivo Rascal

Para a compilação de um arquivos Rascal, deve ser executado via terminal o arquivo jar do compilador, passando como parâmetro o arquivo a ser compilado, conforme exemplo abaixo:

```
java -jar compilador-1.0-SNAPSHOT.jar [op] [nomeDoArquivo.ras]
```

Onde:

- **[op]**: Parâmetro opcional, sendo:

    - **-o** Gera o arquivo json contendo a AST
- **[nomeDoArquivo.ras]** Nome do arquivo contendo o código fonte Rascal a ser compilado;

Caso as análises e geração de código ocorram sem problemas, um arquivo com o nome do programa (declarado dentro do código fonte) será gerado, com a extensão *.mepa*. Caso tenha sido passado o parâmetro -o, um arquivo com mesmo nome e sufixo *-AST.json* contendo a árvore sintática abstrata também será criado.

Etapas não atingidas nesta implementação

- Exigência de *begin* e *end* para delimitação dos blocos de *if* e *while* . Mesmo quando o bloco só possui uma linha de instruções, é necessário que seja alocado dentro de um bloco com begin e end.
- Chamada de função com outra função como parâmetro. Ex: *proc1(x, func(y))*
- Atribuição de variável booleana. Ex: *bool := x > y*
- Operadores lógicos AND e OR
- Expressão booleada com operação composta em ambos os lados: Ex. *if (x + 1 < y - 2)*. Porém, é aceitável operação composta do lado esquerdo da validação. Ex: *if (x + 1 < y)*

A falta de algumas destas construções impediram que o teste *funSimples2.ras* não fosse executado com sucesso, todos os outros foram compilados com a lógica original, apenas ajustando os blocos de if e while, e adicionando eventualmente alguma atribuição auxiliar.