UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Disciplina: 9797 - Compiladores
Professor Dr. Alisson Renan Svaigen

Trabalho Prático

Implementação de um compilador para a linguagem Rascal

Especificação Geral

O presente trabalho propõe a construção de um compilador para a linguagem experimental **Rascal** (Reduced Pascal). Essa linguagem consiste num subconjunto bastante reduzido da linguagem Pascal, com modificações em sua sintaxe original. O compilador deve realizar as três etapas de análise: léxica, sintática e semântica. Após a realização das análises, o compilador deverá gerar código intermediário para ser interpretado por uma máquina denominada MEPA (Kowaltowski, 1983). Uma implementação desta máquina será disponibilizada junto com os arquivos de apoio ao trabalho.

Para o desenvolvimento do trabalho, poderão ser utilizadas as seguintes linguagens: C, C++ ou Python. Para auxiliar nas análises a serem realizadas, podem ser utilizadas ferramentas auxiliares, como Flex[1], Bison[2] (para linguagem C) e PLY[3] (para linguagem Python). É requisito deste trabalho que uma Árvore Sintática Abstrata (AST) seja produzida a partir da análise sintática. O objetivo de se construir uma AST explícita em vez de utilizar as ações semânticas do Bison para a Tradução Dirigida por Sintaxe (SDT) é que, dessa forma, o código ficará mais organizado (modularizado), facilitando sua manutenção e depuração.

O compilador deve permitir ser executado a partir da linha de comando, passando como argumentos o arquivo de código fonte em Rascal e o arquivo de saída, onde será gravado o código objeto da MEPA.

Características da linguagem Rascal

Por se tratar de um subconjunto de Pascal[4], Rascal compartilha a maioria das características dessa linguagem, mas com pequenas alterações que serão descritas no

---

[1] https://westes.github.io/flex/manual/
[2] https://www.gnu.org/software/bison/manual/
[3] https://ply.readthedocs.io/en/latest/index.html
[4] Este livro apresenta uma visão geral da linguagem Pascal: https://www.ime.usp.br/~slago/slago-pascal.pdf

decorrer desta especificação. Uma delas é que **Rascal é case-sensitive**, diferentemente da linguagem Pascal original.

Palavras reservadas

`program`	`begin`	`if`	`do`	`or`
`var`	`end`	`then`	`read`	`not`
`procedure`	`false`	`else`	`write`	`div`
`function`	`true`	`while`	`and`	`integer`
				`boolean`

Símbolos

`(`	`,`	`-`	`<>`	`>=`
`)`	`;`	`*`	`>`	`<=`
`.`	`+`	`=`	`<`	`:=`
				`:`

Vamos considerar que Rascal **considera apenas a existência de números inteiros.** Desse modo, assim como em Pascal, o operador de divisão inteira é representado pela palavra `div`. Como não trabalharemos com números reais, este será o único operador de divisão disponível.

Tipagem

A linguagem Rascal possui apenas dois tipos primitivos:

- **`integer`**: representa valores numéricos inteiros;
- **`boolean`**: representa valores lógicos (`false` e `true`).

Diferente de Pascal, Rascal **não permite** que novos tipos sejam definidos pelo usuário.

Interfaces de Entrada e Saída

Dois comandos de entrada e saída estão disponíveis: `read` e `write`. O primeiro deles solicita a leitura de dados pela interface de entrada padrão, já o segundo exibe valores na interface de saída padrão.

Eles são procedimentos pré-definidos pela linguagem e são capazes de receber um número indeterminado de argumentos do tipo `integer` ou `boolean`. No entanto, não é possível "escrever uma mensagem" associada à etapa de leitura, visto que Rascal possui apenas tipos inteiros e booleanos.

Exemplo:

```
read(x, y);
write(x, y, 2*x+y, 3);
```

Especificação Léxica

Identificadores

Os identificadores são nomes associados a entidades do programa, como variáveis, tipos, sub-rotinas (funções e procedimentos), etc. Estes nomes podem ser criados pelo usuário, mas também podem ser pré-definidos pela linguagem.

Em Rascal, os nomes dos dois tipos primitivos (**`integer`** e **`boolean`**) são pré-definidos pela linguagem. Dessa forma, eles não são palavras reservadas e deverão ser instalados na tabela de símbolos logo no início da fase de análise semântica / tradução. Caso estes identificadores sejam utilizados durante o programa, deve-se verificar a correção de seu uso. Por exemplo, o identificador **`integer`**, que foi instalado na tabela de símbolos como um símbolo da categoria "tipo", não pode ser empregado como nome de uma nova variável declarada pelo usuário.

Os identificadores devem ser formados apenas por letras (minúsculas ou maiúsculas), sublinhas '_' ou dígitos (0 a 9), sendo que devem iniciar com uma letra. Nas regras gramaticais da linguagem, o símbolo `id` será utilizado para expressar um identificador qualquer.

Números

As constantes numéricas devem ser representadas na base decimal e podem conter qualquer combinação de dígitos entre 0 e 9. Os números negativos não serão processados na fase léxica, mas sim na fase sintática. Dessa forma, o número −42, por exemplo, consiste em dois símbolos: '−' e '42', representando uma expressão de negação numérica (inversão de sinal) sobre a constante 42.

Lógicos

As constantes lógicas `falso` e `verdadeiro` são representadas respectivamente pelas palavras reservadas **`false`** e **`true`**.

Comentários

A linguagem Rascal não aceita a inserção de comentários no código. Além disso, espaços em branco, tabulações e quebras de linha não possuem nenhum significado específico e devem ser ignorados.

Especificação Sintática

A gramática de Rascal, disponível no Anexo I, foi adaptada a partir da gramática apresentada por (Kowaltowski, 1983), a qual já é um subconjunto da linguagem Pascal. No entanto, caso sua equipe for utilizar alguma ferramenta auxiliar para implementação do compilador, certifique-se de que a notação utilizada para representação da gramática está de acordo com a ferramenta a ser utilizada!

Por exemplo, a gramática apresentada no Anexo I não está em um formato aceito pelo Bison, especialmente pelas construções:

- { α } para indicar repetição;
- [α] para indicar opção;

Essas construções podem ser facilmente transformadas em construções equivalentes da seguinte forma:

- Regras de repetição na forma $A \rightarrow \beta \{ \alpha \}$, se transformam em $A \rightarrow \beta\alpha \mid \beta$
- Regras opcionais na forma $A \rightarrow \beta [\alpha]$, se transformam em $A \rightarrow \beta \mid \beta\alpha$

Você vai precisar destas transformações principalmente para as regras de formação de lista e de expressões!

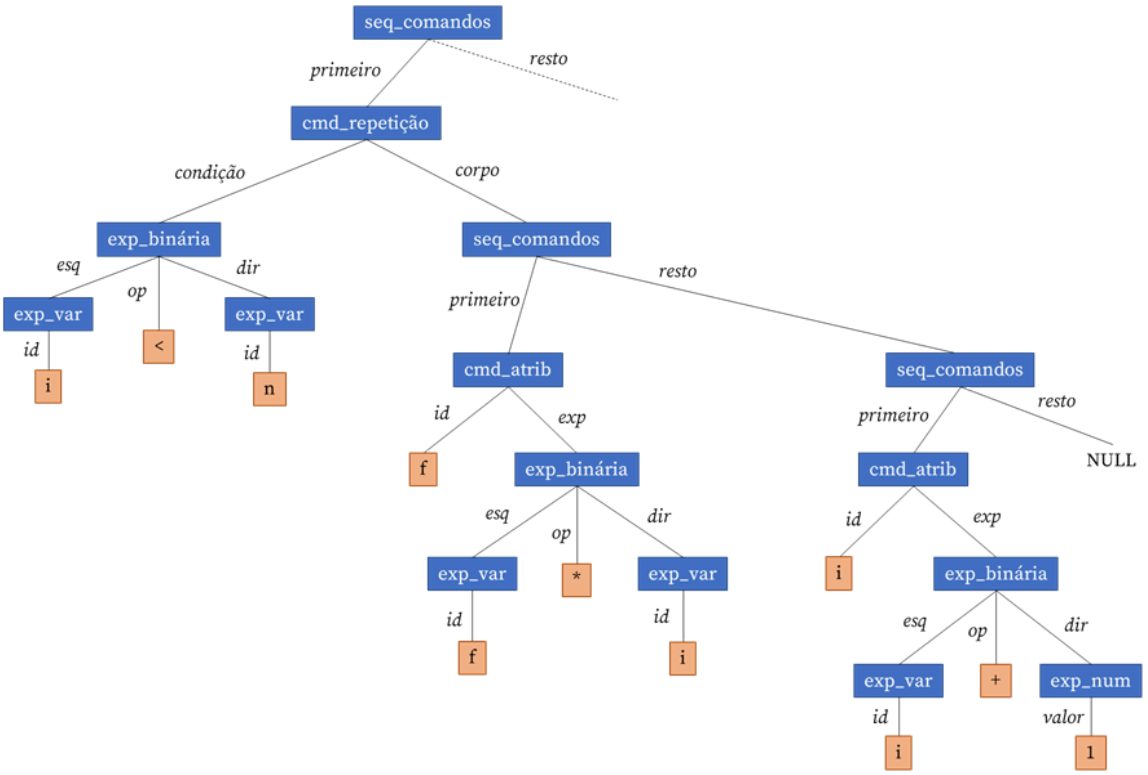
Construção da Árvore Sintática Abstrata

Como comentado anteriormente, a análise sintática terá como responsabilidade, além de verificar a correção da sintaxe do programa e emitir erros apropriados, construir uma Árvore Sintática Abstrata (AST).

A AST é uma representação simplificada da estrutura do programa, abstraindo as derivações sintáticas em nós que representam as construções da linguagem, em vez dos símbolos propriamente ditos. Por exemplo, considere o seguinte fragmento de código:

```
while i < n do
begin
  f := f * i;
  i := i + 1
end;
```

A AST desse fragmento poderia ser construída da seguinte forma:



Para uma inspiração de como definir as estruturas que irão compor a AST, investigue os códigos disponibilizados [5] por (Appel & Ginsburg, 1998), principalmente os que se referem a “Abstract Syntax”.

Especificação Semântica

Após a construção da AST, a análise semântica e a geração de código podem ser realizadas por meio de uma travessia na árvore. De maneira geral, cada nó da AST terá suas próprias regras de verificação semântica e geração de código. A seguir, são apresentados os tratamentos semânticos das principais construções da linguagem.

Programa

- Um programa consiste em uma sequência de declarações de variáveis (opcional), seguida de uma sequência de declarações de sub-rotinas (opcional) e, por fim, uma sequência de comandos (mandatória).

[5] https://www.cs.princeton.edu/~appel/modern/c/project.html

- Por ser a estrutura principal do programa (nó raiz da AST), a análise começa e termina com ela. Portanto, é aqui que a tabela de símbolos será inicializada e o escopo global será definido.
- Todas as declarações realizadas no programa estão dentro do escopo global, que permanece ativo até o fim da execução.
- Cada sub-rotina define seu próprio escopo local e, como não são permitidas declarações em blocos de comandos, blocos `begin ... end` não criam novo escopo.
- O identificador do programa deve ser instalado na tabela de símbolos como sendo da categoria “programa”.

Declaração

- Declarações de variáveis, funções, procedimentos e parâmetros instalam os símbolos correspondentes e suas vinculações (tipo, escopo, posição no escopo etc.) na tabela de símbolos.
- Antes de instalar um novo identificador, deve-se verificar se ele já existe no mesmo escopo:
    - Caso não exista, ele é instalado normalmente com suas vinculações.
    - Caso exista, a nova declaração é desconsiderada e deve ser emitido um alerta.

Comandos

Os comandos que merecem verificação semântica são:

- **Atribuição:**
    - A variável à esquerda da atribuição deve estar declarada e acessível no escopo atual.
    - A expressão à direita deve ser semanticamente correta e possuir o mesmo tipo da variável à esquerda.
- **Condicional e repetição:**
    - A expressão condicional do **`if`** e do **`while`** deve ser válida e resultar em um valor do tipo lógico.
- **Chamada de procedimento:**
    - O procedimento deve estar declarado e visível no escopo atual.
    - O número de argumentos deve coincidir com o número de parâmetros.
    - Os tipos dos argumentos devem corresponder, na mesma ordem, aos tipos dos parâmetros.

    - Todos os parâmetros são passados por valor, i.e., os argumentos reais são avaliados e copiados para os parâmetros formais.
- **Retorno de função:**
    - Toda função deve retornar exatamente um valor.
    - O tipo do valor retornado deve coincidir com o tipo declarado na função.
    - O retorno é realizado por meio de atribuição ao próprio identificador da função dentro de seu corpo.
    - A ausência de retorno, ou retorno de tipo incompatível, deve gerar erro semântico.
- **Leitura (read):**
    - Os argumentos devem ser variáveis declaradas e visíveis no escopo atual.
- **Escrita (write):**
    - Os argumentos devem ser expressões válidas e bem tipadas.
- **Expressões:**
    - **Aritmética:** O(s) operando(s) devem ser do tipo inteiro. O tipo resultante é inteiro.
    - **Relacional**: Os operandos devem ser do tipo inteiro. O tipo resultante é lógico.
    - **Igualdade/Diferença**: Os operandos devem ser do mesmo tipo primitivo. O tipo resultante é lógico.
    - **Lógica**: O(s) operando(s) devem ser do tipo lógico. O tipo resultante é lógico
    - **Uso de variável**: A variável deve estar declarada e visível no escopo atual. O tipo resultante do uso é o tipo declarado para a variável.
    - **Chamada de Função**: Análise análoga à chamada de procedimento. O tipo resultante da chamada é o tipo declarado para a função.

Geração de linguagem intermediária para a máquina MEPA

A especificação da linguagem intermediária, bem como da máquina MEPA, será apresentada em sala de aula, na qual as estratégias constarão no material de aula a ser disponibilizado.

Casos omissos

Quaisquer casos omissos relacionados ao desenvolvimento deste trabalho que não foram esclarecidos nesta descrição deverão ser arguidas diretamente com o professor da disciplina.

Normas para desenvolvimento e entrega do Trabalho Prático

O trabalho prático deve ser desenvolvido **por uma equipe de três alunos matriculados na disciplina**. Caso você não consiga formar um trio, converse com o professor da disciplina antecipadamente!

A submissão do trabalho prático será feita exclusivamente via Classroom, na página correspondente à disciplina. **O prazo final para a entrega será até às 23h59min do dia 07/12/2025**. **Não serão aceitos trabalhos entregues após o prazo final e também não serão aceitos trabalhos submetidos por outros locais (email, whatsapp, etc.)**

A entrega do trabalho deve consistir de:

- **Código-fonte** do compilador;
- **Relatório técnico** escrito em língua portuguesa, de maneira sucinta e objetiva, contendo:
  - Decisões de projeto e de implementação, com justificativas;
  - Visão geral dos módulos e organização do analisador léxico, sintático e semântico;
  - Passo a passo para compilar/executar o compilador;
  - Descrição e justificativas de possíveis etapas não cumpridas no desenvolvimento.

Datas Importantes:

- 02/10/2025 às 23h59: Entrega parcial - analisador léxico (entrega opcional)
- 07/11/2025 às 23h59: Entrega parcial - analisador léxico + sintático (entrega opcional)
- 07/12/2025 às 23h59: Entrega final - trabalho completo (entrega obrigatória)
- 08/12/2025 e 10/12/2025 - Apresentação do trabalho

Referências

Appel, A. W., & Ginsburg, M. (1998). **Modern Compiler Implementation in C**. Cambridge: Cambridge University Press.

Kowaltowski, T. (1983). **Implementação de Linguagens de Programação**. Rio de Janeiro: Guanabara Dois.

Anexo I - Gramática da linguagem Rascal

Regras léxicas para identificadores e números

```
<número> ::= <dígito> { <dígito> }

<dígito> ::= 0-9

<identificador> ::= <letra> { <letra> | <dígito> | '_'}

<letra> ::= a-zA-Z
```

Regras de Produção da Gramática

```
<programa> ::=
    'program' <identificador> ';' <bloco> '.'

<bloco> ::=
    [ <seção_declaração_variáveis> ]
    [ <seção_declaração_subrotinas> ]
    <comando_composto>

<seção_declaração_variáveis> ::=
     'var' <declaração_variáveis> ';' { <declaração_variáveis> ';'
}

<declaração_variáveis> ::=
    <lista_identificadores> ':' <tipo>

<lista_identificadores> ::=
    <identificador> { ',' <identificador> }

<tipo> ::=
    'boolean' | 'integer'

<seção_declaração_subrotinas> ::=
    { ( <declaração_procedimento> | <declaração_função> ) ';' }

<declaração_procedimento> ::=
```

```
        'procedure'  <identificador>  [  <parâmetros_formais>  ]  ';'
<bloco_subrot>

<declaração_função> ::=
     'function' <identificador> [ <parâmetros_formais> ] ‘:’ <tipo>
';' <bloco_subrot>

<bloco_subrot> ::=
    [ <seção_declaração_variáveis> ]
    <comando_composto>

<parâmetros_formais> ::=
      '('  <declaração_parâmetros>  {  ';'  <declaração_parâmetros>  }
')'

<declaração_parâmetros> ::=
    <lista_identificadores> ':' <tipo>

<comando_composto> ::=
    'begin' <comando> { ';' <comando> } 'end'

<comando> ::=
      <atribuição>
    | <chamada_procedimento>
    | <condicional>
    | <repetição>
    | <leitura>
    | <escrita>
    | <comando_composto>

<atribuição> ::=
    <identificador> ':=' <expressão>

<chamada_procedimento> ::=
    <identificador> [ '(' <lista_expressões> ')' ] regra com erro

<chamada_procedimento> ::=
    <identificador> '(' [ <lista_expressões> ] ')' regra corrigida

<condicional> ::=
    'if' <expressão> 'then' <comando> [ 'else' <comando> ]

<repetição> ::=
    'while' <expressão> 'do' <comando>
```

```
<leitura> ::=
    'read' '(' <lista_identificadores> ')'

<escrita> ::=
    'write' '(' <lista_expressões> ')'

<lista_expressões> ::=
    <expressão> { ',' <expressão> }

<expressão> ::=
    <expressão_simples> [ <relação> <expressão_simples> ]

<relação> ::=
    '=' | '<>' | '<' | '<=' | '>' | '>='

<expressão_simples> ::=
    <termo> { ( '+' | '-' | 'or' ) <termo> }

<termo> ::=
    <fator> { ( '*' | 'div' | 'and' ) <fator> }

<fator> ::=
      <variável>
    | <número>
    | <lógico>
    | <chamada_função>
    | '(' <expressão> ')'
    | 'not' <fator>
    | '-' <fator>

<variável> ::=
    <identificador>

<lógico> ::=
      'false'
    | 'true'

<chamada_função> ::=
    <identificador> [ '(' <lista_expressões> ')' ] regra com erro

<chamada_função> ::=
    <identificador> '(' [ <lista_expressões> ] ')' regra corrigida
```

Anexo II - Critérios de Avaliação do Trabalho Prático

Quesito	**Pontuação**
Implementação do Analisador Léxico	2,5
Implementação do Analisador Sintático	2,5
Implementação do Analisador Semântico	2,5
Implementação do Geração para código-objeto MEPA	1,5
Escrita do relatório técnico	1,0
Total	**10,0**

Observação: embora a apresentação do trabalho não seja um quesito na qual é atribuído pontuação, é **mandatório** que **toda a equipe** compareça à apresentação do trabalho, visto que é por meio da apresentação que o professor da disciplina se **certificará** que todos os membros da equipe contribuíram para a realização do trabalho prático.