Disciplina: 11927 – Introdução à Compilação
Profa. Valéria

Trabalho Prático

Implementação de um compilador para a linguagem Tascal

Objetivo

O trabalho prático consiste na implementação de um compilador simples para a linguagem Tascal (*Tiny Pascal*), contendo analisador léxico, analisador sintático e execução de ações semânticas básicas (verificação de declarações e tipos).

O trabalho poderá ser desenvolvido nas seguintes linguagens: C, C++ ou Python. Será permitido o uso de ferramentas de geração automática de *scanners* e *parsers*, como Flex[1], Bison[2] (para linguagem C/C++) e PLY[3](para linguagem Python).

O compilador deverá ser executado a partir da linha de comando, passando como argumento o arquivo de código-fonte em Tascal a ser analisado.

Normas para desenvolvimento e entrega do trabalho

O trabalho poderá ser desenvolvido **por uma equipe de duas pessoas**.

A submissão do trabalho deverá ser feita exclusivamente pelo *Classroom*. **O prazo final para a entrega será até às 21 horas do dia 20/10/2025**.

Deverão ser entregues os seguintes artefatos:

- **Código-fonte** do compilador (compactado no caso de vários arquivos separados);
- **Relatório técnico** curto (máx. 3 páginas), explicando:
  - Ferramenta usada (Flex/Bison ou PLY), se for o caso;
  - Organização básica do código;
  - Ações semânticas implementadas;
  - Descrição e justificativas de possíveis etapas não cumpridas.

Avaliação do trabalho

O código-fonte do compilador será avaliado de acordo com sua organização, correção e requisitos atendidos. Para isso, um conjunto de programas escritos em Tascal será utilizado para verificar as etapas que foram cumpridas no desenvolvimento do compilador. Seu compilador deve aceitar corretamente programas válidos e rejeitar programas inválidos com mensagens de erro adequadas. As mensagens de erro devem indicar pelo menos a linha e a natureza do erro (léxico, sintático ou semântico).

Também fará parte da avaliação do trabalho, uma apresentação sobre sua implementação, a ser realizada pela equipe nos dias 20/10/2025 e 22/10/2025, conforme agendamento prévio.

[1] https://westes.github.io/flex/manual/

[2] https://www.gnu.org/software/bison/manual/

[3] https://ply.readthedocs.io/en/latest/index.html

Características da linguagem

Por se tratar de um subconjunto de Pascal[4], Tascal compartilha muitas das características dessa linguagem, mas com diversas alterações que serão descritas no decorrer desta especificação. Uma delas é que **Tascal é *case-sensitive***, diferentemente da linguagem Pascal original.

Palavras reservadas

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

Símbolos especiais

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

Tipagem

A linguagem Tascal possui apenas dois tipos primitivos:

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

Diferentemente de Pascal, Tascal **não permite** que novos tipos sejam definidos pelo usuário.

Interfaces de entrada e saída

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

Eles são comandos de I/O da 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 Tascal não suporta o tipo *string*. Seguem alguns exemplos de uso desses comandos:

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

[4]Este livro apresenta uma visão geral da linguagem Pascal: https://www.ime.usp.br/~slago/slago-pascal.pdf

Especificação léxica

Identificadores

Em Tascal, identificadores nomeiam o programa e as variáveis globais. Os identificadores devem ser formados apenas por letras (minúsculas ou maiúsculas), sublinhas '_' ou dígitos (0 a 9), devendo 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 e símbolos a serem ignorados

A linguagem Tascal não aceita a inserção de comentários no código-fonte. Símbolos não reconhecidos pela linguagem devem ser reportados como erro léxico. Além disso, espaços em branco, tabulações e quebras de linha não possuem significado específico e devem ser ignorados.

Especificação sintática

A gramática de Tascal, disponível no **Anexo I** desta especificação, 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 sintaxe aceita pela ferramenta.

Por exemplo, a gramática do Anexo I utiliza a notação EBNF, a qual não corresponde ao formato aceito pelo Bison ou pelo PLY, especialmente pelas construções:

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

No entanto, essas construções podem ser facilmente transformadas em construções equivalentes da seguinte forma:

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

Você vai precisar dessas transformações principalmente para as regras de formação de lista e de expressões.

Especificação semântica

Após a análise sintática, o compilador deverá realizar verificações semânticas diretamente durante o reconhecimento das construções da linguagem. Essas verificações consistem em ações associadas às regras gramaticais, responsáveis por garantir que o programa segue as restrições da linguagem além da sintaxe.

Entre as responsabilidades desta etapa estão: controlar a tabela de símbolos, assegurar que variáveis sejam declaradas antes do uso, checar compatibilidade de tipos em expressões e comandos, e validar as condições em estruturas de controle.

A seguir, são descritas as principais regras semânticas que devem ser aplicadas às construções da linguagem Tascal.

Programa

- Um programa é composto por uma seção opcional de declaração de variáveis e uma sequência obrigatória de comandos.
- Como ele é a estrutura principal, a análise inicia e termina no programa. Nesse ponto, a tabela de símbolos é criada e o escopo global é estabelecido.
- Todas as declarações realizadas no programa estão dentro do escopo global, que permanece ativo até o fim da execução.
- O identificador do programa deve ser instalado na tabela de símbolos como sendo da categoria “programa”.

Declaração

- As declarações de variáveis instalam os símbolos correspondentes e suas vinculações (tipo, escopo etc.) na tabela de símbolos.
- Antes de instalar um novo identificador, deve-se verificar se ele já existe no escopo. ○ Caso não exista, ele é instalado normalmente com suas vinculações.
  - ○ Caso exista, deve ser gerado um erro semântico.

Comandos

Os comandos que terão verificação semântica são os seguintes:

- **Atribuição:**
  - ○ A variável à esquerda da atribuição deve estar declarada e visível no escopo atual. ○ A

expressão à direita deve ser semanticamente válida e possuir o mesmo tipo da variável à esquerda.

- **Condicional e repetição:**
  - A expressão condicional dos comandos **`if`** e **`while`** devem ser válidas e resultar em um valor do tipo lógico.
- **Leitura (`read`):**
  - Os argumentos devem ser variáveis declaradas e visíveis no escopo.
- **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.
  - **Literais:** inteiros resultam em `integer`, valores lógicos resultam em `boolean`.

Casos omissos

Quaisquer casos relacionados ao desenvolvimento deste trabalho que não foram esclarecidos nesta especificação deverão ser arguidos diretamente com a professora.

Referências

APPEL, A. W.; GINSBURG, M. **Modern Compiler Implementation in C**. Cambridge: Cambridge University Press, 1998.

KOWALTOWSKI, T. **Implementação de Linguagens de Programação**. Rio de Janeiro: Guanabara Dois, 1983.

Anexo I - Gramática da linguagem Tascal

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> ]
```

```
 <comando_composto>

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

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

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

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

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

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

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



```
<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>
 | '(' <expressão> ')'
 | 'not' <fator>
 | '-' <fator>

<variável> ::=
 <identificador>

<lógico> ::=
 'false'
 | 'true'
```