Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Trabalho Prático I

1. Objetivo

O objetivo desse trabalho é desenvolver um interpretador para um subconjunto de uma linguagem de programação conhecida. Para isso foi criada *miniPascal*, uma linguagem de programação de brinquedo baseada em *Pascal* (https://pt.wikipedia.org/wiki/Pascal_(linguagem_de_programa%C3%A7%C3%A3o)). Embora a linguagem *Pascal* defina os tipos de suas variáveis estaticamente, ou seja, em tempo de compilação, no *miniPascal* os tipos são definidos dinamicamente, ou seja, durante a interpretação. Contudo, os nomes das variáveis ainda precisam ser declarados previamente antes do corpo do programa, inclusive se forem constantes.

2. Contextualização

A seguir é dado um exemplo de utilização da linguagem *miniPascal* para verificar o tipo de triângulo formado por três dimensões dadas: se equilátero, isósceles ou escaleno. O programa termina caso o triângulo seja inválido.

```
program triangle;

const
  MSG = 'Enter the three sizes of a triangle: ';

var
  a, b, c;
  valid = 1;

begin
  repeat
    write(MSG);
    readln(a, b, c);

    write('(', a, ', ', b, ', ', c, ') ');

    if (a + b <= c) or (a + c <= b) or (b + c <= a) then
        begin
          writeln('is an invalid triangle');
          valid := 0
        end
    else
      if (a = b) and (b = c) then
        writeln('is an equilateral triangle')
      else if (a = b) or (a = c) or (b = c) then
        writeln('is a isosceles triangle')
      else
        writeln('is a scalene triangle');

    writeln();
  until valid = 0
end.
```

triangle.mpas

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

A linguagem deve definir as variáveis que serão usadas antes do corpo do programa, sejam elas constantes (na declaração de **const**) ou não (na declaração de **var**). A linguagem suporta três tipos primitivos: inteiros, reais (ponto-flutuante) strings (sequência de caracteres entre aspas simples). Operações aritméticas devem converter strings para números inteiros ou reais (usar 0 se não for possível). Não é possível fazer operações com strings (concatenação por exemplo), a não ser comparações (igualdade e diferença somente). A linguagem possui somente comentários de múltiplas linhas onde são ignoradas quaisquer sequências de caracteres entre `(*` e `*)`. A linguagem possui as seguintes características:

1) **Comandos:**
   a. **atribuição**: atribuir uma expressão a uma variável.
   b. `if`: executar comandos baseado em expressões condicionais.
   c. `case`: executar comandos baseados no valor de uma expressão.
   d. `while`: repetir comandos enquanto a expressão for verdadeira.
   e. `repeat`: repetir comandos enquanto a expressão for falsa.
   f. `for`: repetir comandos para uma sequência de valores.
   g. `write/writeln`: imprimir na tela.
   h. `readln`: ler do teclado.
2) **Constantes:**
   a. **Inteiro**: número formado por dígitos.
   b. **Real**: número com precisão em ponto-flutuante.
   c. **String:** uma sequência de caracteres entre aspas simples.
   d. **Lógico:** operações de comparações que obtém um valor lógico (não podem ser armazenados em variáveis).
3) **Valores:**
   a. Variáveis (começam com _ ou letras, seguidos de _, letras ou dígitos).
   b. Constantes (variáveis cujo valor não pode ser modificado)
   c. Literais (inteiros, reais e strings).
4) **Operadores:**
   a. **Inteiro:** `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `%` (resto, somente para inteiros).
   b. **Lógico:** `=` (igual), `<>` (diferença), `<` (menor), `>` (maior), `<=` (menor igual), `>=` (maior igual), `not` (negação).
   c. **Conector**: `and` (E lógico), `or` (OU lógico).
   d. **Atribuição:** `=` (atribuir), `+=` (incrementar e atribuir), `-=` (decrementar e atribuir), `*=` (multiplicar e atribuir), `/=` (dividir e atribuir), `%=` (resto da divisão para inteiros e atribuir)

3. Gramática

A gramática da linguagem *miniPascal* é dada a seguir no formato de Backus-Naur estendida (EBNF):

```
<program>  ::= program <id> ';'
               [ const <const> { <const> } ]
               [ var <var> { <var> } ]
               <block> '.'
```

```
<const>    ::= <id> = <value> ';'
<var>      ::= <id> { ',' <id> } [ = <value> ] ';'

<block>    ::= begin [ <cmd> { ';' <cmd> } ] end
<body>     ::= <block> | <cmd>

<cmd>      ::= <assign> | <if> | <case> | <while> | <for> | <repeat> | <write> | <read>
<assign>   ::= <id> := <expr>
<if>       ::= if <boolexpr> then <body> [else <body>]
<case>     ::= case <expr> of { <value> : <body> ';' } [ else <body> ';' ] end
<while>    ::= while <boolexpr> do <body>
<repeat>   ::= repeat [ <cmd> { ';' <cmd> } ] until <boolexpr>
<for>      ::= for <id> := <expr> to <expr> do <body>
<write>    ::= (write | writeln) '(' [ <expr> { ',' <expr> } ] ')'
<read>     ::= readln '(' <id> { ',' <id> } ')'

<boolexpr> ::= [ not ] <cmpexpr> { (and | or) <boolexpr> }
<cmpexpr>  ::= <expr> ('=' | '<>' | '<' | '>' | '<=' | '>=') <expr> | '(' <boolexpr> ')'

<expr>     ::= <term> { ('+' | '-') <term> }
<term>     ::= <factor> { ('*' | '/' | '%') <factor> }
<factor>   ::= <value> | <id> | '(' <expr> ')'
<value>    ::= <integer> | <real> | <string>
```

4. Instruções

Deve ser desenvolvido um interpretador em linha de comando que recebe um programa-fonte na linguagem *miniPascal* como argumento e executa os comandos especificados pelo programa. Por exemplo, para o programa *triangle.mpas* deve-se produzir uma saída semelhante a:

```
$ ./mpasi triangle.mpas
Usage: ./mpasi [miniPascal file]
$ ./mpasi triangle.mpas
Enter the three sizes of the triangle: 3 4 5
(3, 4, 5) is a scalene triangle

Enter the three sizes of the triangle: 0 0 0
(0, 0, 0) is an invalid triangle
```

O programa deverá abortar sua execução, em caso de qualquer erro léxico, sintático ou semântico, indicando uma mensagem de erro. As mensagens são padronizadas indicando o número da linha (2 dígitos) onde ocorreram:

Tipo de Erro	**Mensagem**
Léxico	Lexema inválido [*lexema*]
	Fim de arquivo inesperado
Sintático	Lexema não esperado [*lexema*]
	Fim de arquivo inesperado
Semântico	Operação inválida

Exemplo de mensagem de erro:

```
$ ./mpasi erro.mpas
03: Lexema não esperado [;]
```

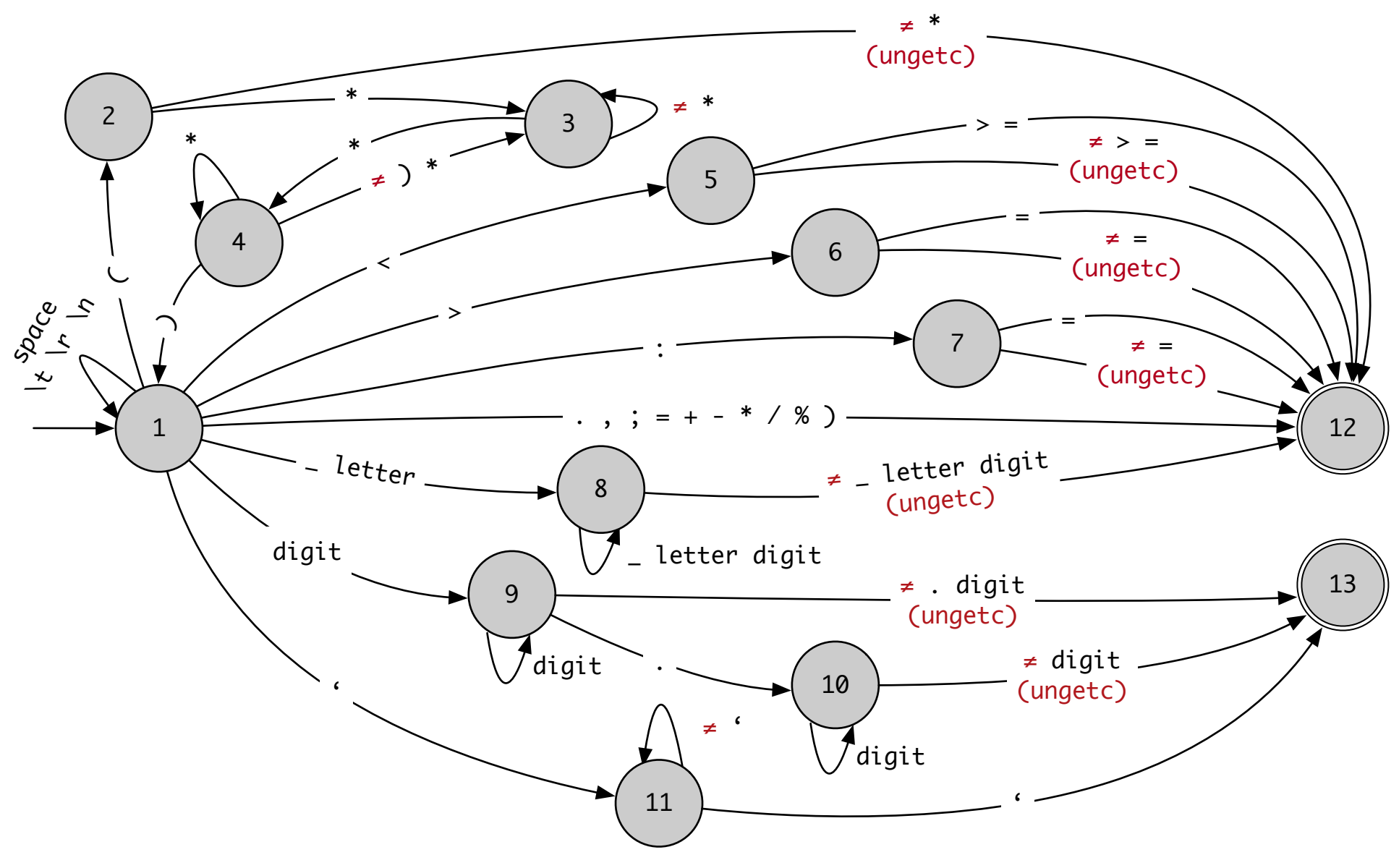
5. Avaliação

O trabalho deve ser feito em grupo de até dois alunos, sendo esse limite superior estrito. O trabalho será avaliado em 15 pontos, onde essa nota será multiplicada por um fator entre 0.0 e 1.0 para compor a nota de cada aluno individualmente. Esse fator poderá estar condicionado a apresentações presenciais a critério do professor.

Trabalhos copiados, parcialmente ou integralmente, serão avaliados com nota **ZERO**, sem direito a contestação. Você é responsável pela segurança de seu código, não podendo alegar que outro grupo o utilizou sem o seu consentimento.

6. Submissão

O trabalho deverá ser submetido até as 23:59 do dia 22/02/2021 (segunda-feira) via sistema acadêmico em pasta específica. Não serão aceitos, em hipótese alguma, trabalhos enviados por e-mail ou por quaisquer outras fontes.


1
2
3
4
5
6
7
8
9
10
11
12
13
space
\t \r \n
(
)
*
≠ *
(ungetc)
≠) *
<
>
:
. , ; = + - * / %)
> =
≠ > =
(ungetc)
=
≠ =
(ungetc)
≠ =
(ungetc)
- letter
_ letter digit
≠ _ letter digit
(ungetc)
digit
≠ . digit
(ungetc)
.
≠ digit
(ungetc)
‘
≠ ‘