Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Trabalho Prático I

1. Objetivo

O objetivo desse trabalho é desenvolver um interpretador para uma linguagem de programação fictícia chamada *MiniShell*, baseada em um subconjunto de funcionalidades de *shell script*. Essa linguagem é capaz de executar comandos em sistemas operacionais Linux para realizar determinadas tarefas de forma mais fácil e prática.

2. Contextualização

Linguagens de script são muito úteis para desenvolver programas simples para executar tarefas rapidamente. A seguir é dado um exemplo para imitar a saída do comando `/usr/bin/id` nessa linguagem.

```
# get the user
set user=$("whoami");

# get and print the effective uid
set uid=$("cat /etc/passwd | grep '^" . ${user} . ":' | cut -f 3 -d':'");
print "uid=" . ${uid} . "(" . ${user} . ") ";

# get the gid/group and print it
set gid=$("cat /etc/passwd | grep '^" . ${user} . ":' | cut -f 4 -d':'");
set group=$("cat /etc/group | grep '^.*:.*:" . ${gid} . ":.*$' | cut -f 1 -d':'");
print "gid=" . ${gid} . "(" . ${group} . ") ";

# print the groups
print "groups=" . ${gid} . "(" . ${group} . ")";
for line in $("cat /etc/group | sed 's/#.*//g' | sed '/^[[:space:]]*$/ d'"); do
    # find each group
    set gname=$("echo -n '" . ${line} . "' | cut -f 1 -d':'");

    # ignore the default group
    if [ ${gname} -ne ${group} ]; then
        # find the number
        set gnumber=$("echo -n '" . ${line} . "' | cut -f 3 -d':'");

        # check each group users for possible match
        for grp in $("echo -n '" . ${line} . "' | cut -f 4 -d':' | tr ',' '\n'"); do
            if [ ${grp} -eq ${user} ]; then
                print "," . ${gnumber} . "(" . ${gname} . ")";
            fi
        done
    fi
done
println "";
```

id.msh

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

Um programa em *MiniShell* possui uma memória indexada através de nomes de variáveis que armazenam apenas conteúdo no formato de texto (string). Todas as variáveis do sistema possuem visibilidade global. Para realizar operações aritméticas, essas strings devem ser convertidas previamente para sua contrapartida inteira. A linguagem possui comentários de uma linha onde são ignorados qualquer sequência de caracteres após o símbolo # (hashtag). A linguagem possui as seguintes características:

- **Comandos:**
  - `exit`: interromper a execução do programa.
  - `read`: ler uma string do teclado e armazenar numa variável.
  - `print/println`: imprimir uma sequência de texto.
  - `set`: armazenar uma sequência de texto em uma variável.
  - `exec`: executar um comando.
  - `if/elif/else`: executar comandos baseado em expressões condicionais.
  - `while`: repetir comandos enquanto a expressão condicional for verdadeira.
  - `for`: atribuir na variável o valor de cada linha dos dados e executar os comandos especificados.
- **Valores:**
  String:
  - **Variável**: começa com letra seguido de letras e dígitos
  - **Literal:** constante string entre aspas duplas
  - **Expansões:** começam com **$**
    - **Artimética:** expressões aritméticas entre **$((** e **))**.
    - **Parâmetro:** valor de uma variável entre **${** e **}** .
    - **Comando:** saída de um comando entre **$(** e **)**.

  Inteiro: constante inteiras formadas por dígitos
  Lógico: operações de comparações que obtém um valor lógico
- **Operadores:**
  - **String: .** (concatenação)
  - **Inteiro: +** (adição), - (subtração), ***** (multiplicação), **/** (divisão), **%** (resto inteiro), ****** (exponenciação)
  - **Lógico:**
    Comparadores:
    - **Unárias**: `--z` (vazia), `--n` (não vazia),
    - **Binárias**: `--eq` (iguais), `--ne` (diferentes), `--lt` (menor que), `--le` (menor igual), `--ge` (maior igual), `--gt` (maior que)

    Conectores:
    - && (E lógico) e || (OU lógico)

3. Gramática

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

```
<commands>    ::= <command> { <command> }
<command>     ::= <exit> | <read> | <print> | <set> | <exec> | <if> | <while> | <for>
<exit>        ::= exit ';'
<read>        ::= read <var> ';'
<print>       ::= (print | println) <expr> ';'
<set>         ::= set <var> '=' <expr> ';'
<exec>        ::= exec <expr> ';'
<if>          ::= if <bool-expr> ';' then <commands>
                      { elif <bool-expr> ';' then <commands> } [ else <commands> ] fi
<while>       ::= while <bool-expr> ';' do <commands> done
<for>         ::= for <var> in <expr> ';' do <commands> done

<expr>        ::= <string-expr> { '.' <string-expr> }
<string-expr> ::= <string> | <expansion>
<expansion>   ::= <arith-exp> | <param-exp> | <cmd-exp>
<arith-exp>   ::= '$((' <int-expr> '))'
<param-exp>   ::= '${' <var> '}'
<cmd-exp>     ::= '$(' <expr> ')'

<int-expr>    ::= <term> [ ('+' | '-') <term> ]
<term>        ::= <power> [ ('*' | '/' | '%') <power> ]
<power>       ::= <factor> [ '**' <power> ]
<factor>      ::= <var> | <number> | '(' <int-expr> ')'

<bool-expr>   ::= <clause> { ('&&' | '||') <clause> }
<clause>      ::= '[' [ '!' ] <cmp> ']'
<cmp>         ::= <unaryop> <expr> | <expr> <relop> <expr>
<unaryop>     ::= --z | --n
<relop>       ::= --eq | --ne | --lt | --le | --ge | --gt
```

4. Instruções

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

```
$ msi
Usage: ./msi [Mini Shell Script File]
$ msi id.msh
uid=1000(andrei) gid=1000(andrei) groups=1000(andrei),
4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lpad
min),124(sambashare),999(vboxsf)
```

O programa deverá abortar sua execução, em caso de qualquer erro léxico, sintático ou semântico, indicando uma mensagem de erro. As mensagens de erro são padronizadas indicando o número da linha (com no mínimo 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:

```
$ msi erro.msh
03: Lexema não esperado [;]
```

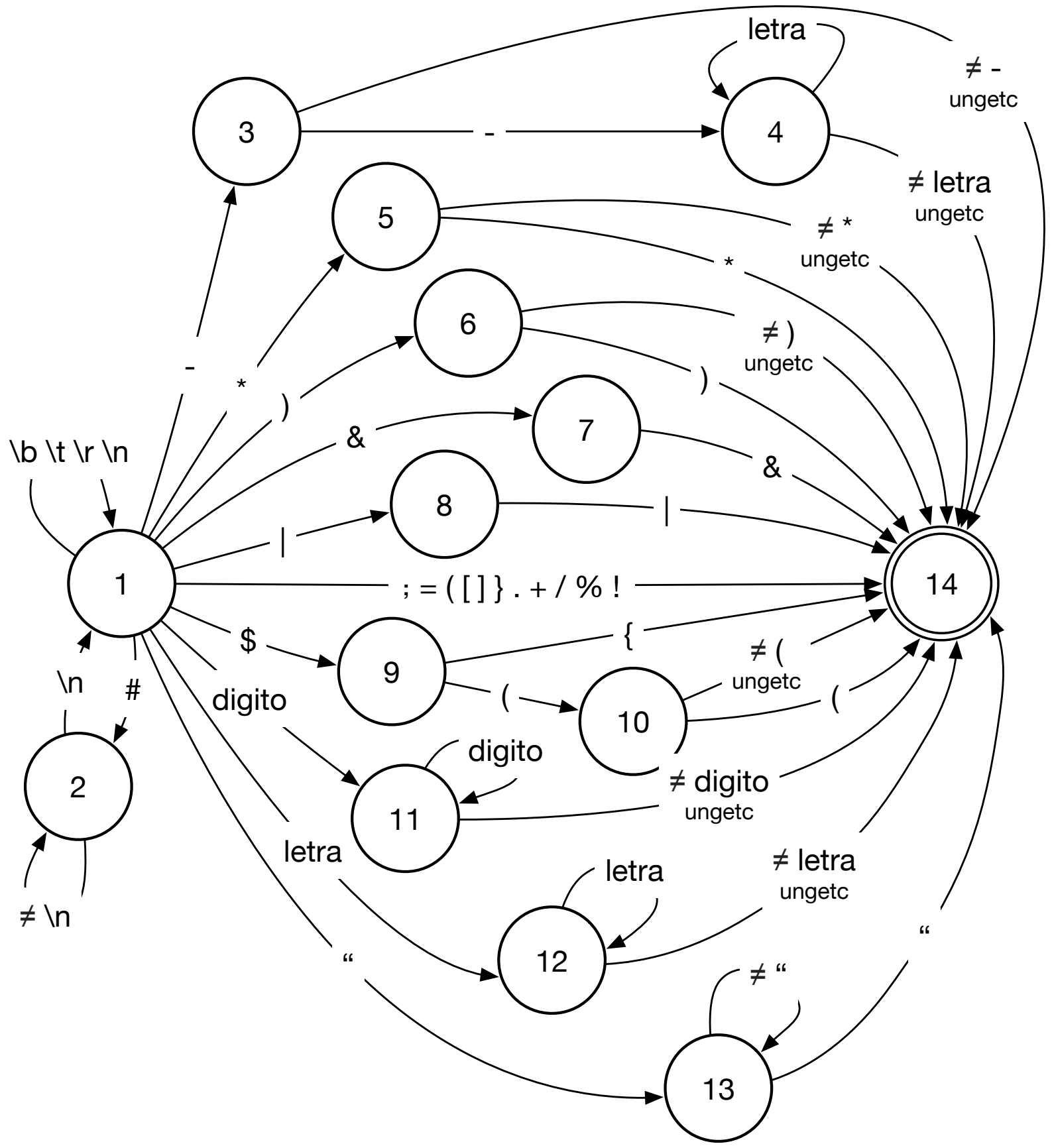
5. Avaliação

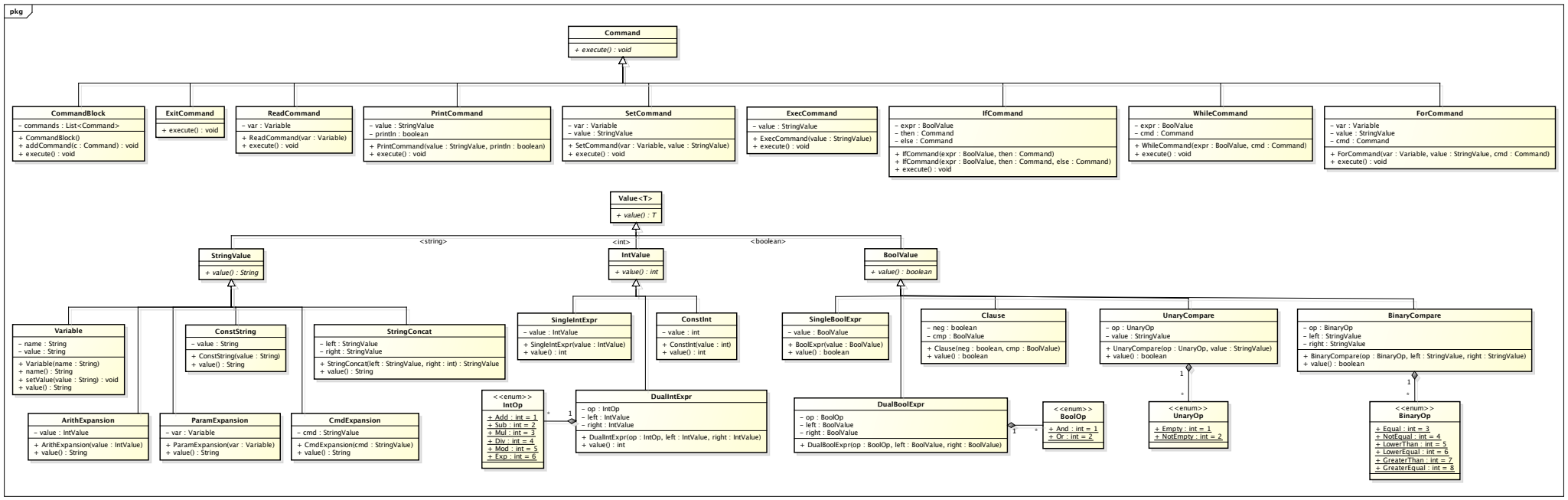
O trabalho deve ser feito em grupo de até dois alunos, sendo esse limite superior estrito. O trabalho será avaliado em 20 pontos, onde essa nota será multiplicada por um fator entre 0.0 e 1.0 para compor a nota de cada aluno individualmente. Esse fator poderá estar condicionado a apresentações presenciais 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 23:55 do dia 02/05/2016 (segunda-feira) via sistema acadêmico (Moodle) 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
14
\b \t \r \n
\n
#
≠ \n
-
letra
≠ -
ungetc
≠ letra
ungetc
*
≠ *
ungetc
)
≠)
ungetc
&
|
; = ([ ] } . + / % !
$
digito
{
≠ (
ungetc
(
≠ digito
ungetc
“
≠ “


pkg
Command
+ execute() : void
CommandBlock
- commands : List<Command>
+ CommandBlock()
+ addCommand(c : Command) : void
+ execute() : void
ExitCommand
+ execute() : void
ReadCommand
- var : Variable
+ ReadCommand(var : Variable)
+ execute() : void
PrintCommand
- value : StringValue
- printIn : boolean
+ PrintCommand(value : StringValue, printIn : boolean)
+ execute() : void
SetCommand
- var : Variable
- value : StringValue
+ SetCommand(var : Variable, value : StringValue)
+ execute() : void
ExecCommand
- value : StringValue
+ ExecCommand(value : StringValue)
+ execute() : void
IfCommand
- expr : BoolValue
- then : Command
- else : Command
+ IfCommand(expr : BoolValue, then : Command)
+ IfCommand(expr : BoolValue, then : Command, else : Command)
+ execute() : void
WhileCommand
- expr : BoolValue
- cmd : Command
+ WhileCommand(expr : BoolValue, cmd : Command)
+ execute() : void
ForCommand
- var : Variable
- value : StringValue
- cmd : Command
+ ForCommand(var : Variable, value : StringValue, cmd : Command)
+ execute() : void
Value<T>
+ value() : T
<string>
<int>
<boolean>
StringValue
+ value() : String
IntValue
+ value() : int
BoolValue
+ value() : boolean
Variable
- name : String
- value : String
+ Variable(name : String)
+ name() : String
+ setValue(value : String) : void
+ value() : String
ConstString
- value : String
+ ConstString(value : String)
+ value() : String
StringConcat
- left : StringValue
- right : StringValue
+ StringConcat(left : StringValue, right : int) : StringValue
+ value() : String
SingleIntExpr
- value : IntValue
+ SingleIntExpr(value : IntValue)
+ value() : int
ConstInt
- value : int
+ ConstInt(value : int)
+ value() : int
SingleBoolExpr
- value : BoolValue
+ BoolExpr(value : BoolValue)
+ value() : boolean
Clause
- neg : boolean
- cmp : BoolValue
+ Clause(neg : boolean, cmp : BoolValue)
+ value() : boolean
UnaryCompare
- op : UnaryOp
- value : StringValue
+ UnaryCompare(op : UnaryOp, value : StringValue)
+ value() : boolean
BinaryCompare
- op : BinaryOp
- left : StringValue
- right : StringValue
+ BinaryCompare(op : BinaryOp, left : StringValue, right : StringValue)
+ value() : boolean
ArithExpansion
- value : IntValue
+ ArithExpansion(value : IntValue)
+ value() : String
ParamExpansion
- var : Variable
+ ParamExpansion(var : Variable)
+ value() : String
CmdExpansion
- cmd : StringValue
+ CmdExpansion(cmd : StringValue)
+ value() : String
<<enum>>
IntOp
+ Add : int = 1
+ Sub : int = 2
+ Mul : int = 3
+ Div : int = 4
+ Mod : int = 5
+ Exp : int = 6
DualIntExpr
- op : IntOp
- left : IntValue
- right : IntValue
+ DualIntExpr(op : IntOp, left : IntValue, right : IntValue)
+ value() : int
DualBoolExpr
- op : BoolOp
- left : BoolValue
- right : BoolValue
+ DualBoolExpr(op : BoolOp, left : BoolValue, right : BoolValue)
<<enum>>
BoolOp
+ And : int = 1
+ Or : int = 2
<<enum>>
UnaryOp
+ Empty : int = 1
+ NotEmpty : int = 2
<<enum>>
BinaryOp
+ Equal : int = 3
+ NotEqual : int = 4
+ LowerThan : int = 5
+ LowerEqual : int = 6
+ GreaterThan : int = 7
+ GreaterEqual : int = 8