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 *MiniLambda*. Essa linguagem é capaz de executar operações sobre vetores de forma mais fácil e prática.

2. Contextualização

A seguir são dados dois exemplos de utilização da linguagem com vetores:

1) Contar quantos valores pares e ímpares existem em um vetor de inteiros de tamanho variado.

```
load("Entre com o tamanho do vetor: ") : s;
new zero [s] : array;
0 : i, par, impar;
array.apply(n -> i + 1 : i; load("Valor ", i, ": ") : n;);
array.each(n -> if n % 2 == 0 {
  par + 1 : par;
} else {
  impar + 1 : impar;
});
println("Voce entrou com ", par, " par(es) e ", impar, " impar(es)");
```

evenodds.ml

2) Contar quantos valores menores e maiores que a mediana existem em um vetor de inteiros aleatórios.

```
load("Entre com o tamanho do vetor: ") : s;
new rand [s] : array;
array.sort().at(s / 2) : medium;
array.filter(n -> n < medium).size() : left;
array.filter(n -> n > medium).size() : right;
println("Existe(m) ", left, " valor(es) menor(es) e ", right,
        " valor(es) maior(es) que a mediana ", medium);
```

medium.ml

Um programa em *MiniLambda* possui uma memória indexada através de nomes de variáveis que armazenam inteiros ou vetores de inteiros. Todas as variáveis do sistema possuem visibilidade global. Operações aritméticas só podem ser feitas em inteiros. A linguagem possui comentários de uma linha onde são ignorados qualquer sequência de caracteres após o símbolo # (hashtag). A linguagem possui as seguintes características:

- **Declarações:**
  - `atribuição`: guardar um inteiro/vetor em uma ou mais variáveis.
  - `print`: imprimir uma sequência de texto, inteiro e/ou vetor.
  - `if`: executar comandos baseado em expressões condicionais.
  - `while`: repetir comandos enquanto a expressão for verdadeira.

- **Valores:**
  String: uma sequência de caracteres entre aspas duplas.
  Variável: começa com letra seguido de letras e dígitos.
  Inteiro: constante inteiras formadas por dígitos.
  Lógico: operações de comparações que obtém um valor lógico.
  Vetor: um vetor de inteiros de determinado tamanho.
- **Operadores:**
  - **Inteiro:** `+` (adição), `-` (subtração), `*` (multiplicação), `/` (divisão), `%` (resto inteiro)
  - **Lógico:** `==` (igual), `!=` (diferença), `<` (menor), `>` (maior), `<=` (menor igual), `>=` (maior igual)
  - **Conectores**: and (E lógico), or (OU lógico)
- **Funções**:
  - **Obtém novos vetores**:
    - `new zero` [*n*]: cria um vetor de tamanho *n* com zeros.
    - `new rand` [*n*]: cria um vetor de tamanho *n* com valores inteiros aleatórios (entre 0 e 100).
    - `new fill` [*n*, *v*]: cria um vetor de tamanho *n* preenchido com o valor definido por *v*.
  - **Obtém um valor inteiro :**
    - *x*`.at(`*i*`)`: obtém o valor na posição *i* do vetor *x*.
    - *x*`.size()`: obtém o tamanho do vetor *x*.
  - **Obtém o próprio vetor:**
    - *x*`.show()`: imprime o vetor no formato [v_1, v_2, ..., v_n].
    - *x*`.set(`*i*`,` *v*`)`: define o valor *v* na posição *i* do vetor *x*.
    - *x*`.each(`*y* `->` `cmds)`: para cada elemento do vetor x, associar a variável *y* e executar os comandos `cmds`.
    - *x*`.apply(`*y* `->` `cmds)`: para cada elemento do vetor x, associar a variável *y*, executar os comandos `cmds` e depois adicionar o valor de y no próprio vetor.
  - **Obtém um NOVO vetor:**
    - *x*`.sort()`: obtém um novo vetor ordenado a partir de x.
    - *x*`.add(v)`:
      - **se `v` for inteiro:** obtém um novo vetor adicionando o novo valor v ao final do vetor x.
      - **se `v` for vetor**: obtém um novo vetor da concatenação do vetor x com o vetor v.
    - *x*`.filter(`*y* `->` `cond)`: associar a *y* cada elemento do vetor *x* e adicionar ao novo vetor se a condição for **verdadeira**.
    - *x*`.remove(`*y* `->` `cond)`: associar a *y* cada elemento do vetor *x* e adicionar ao novo vetor se a condição for **falsa**.

3. Gramática

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

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

```
<statements> ::= <cmd> { <cmd> }
<cmd>        ::= <assign> | <print> | <if> | <while>

<assign>     ::= <expr> [ ':' <var> { ',' <var> } ] ';'
<print>      ::= (print | println) '(' <text> ')' ';'
<if>         ::= if <boolexpr> '{' <statements> '}' [ else '{' <statements> '}' ]
<while>      ::= while <boolexpr> '{' <statements> '}'

<text>       ::= (<string> | <expr>) { ',' (<string> | <expr>) }

<boolexpr>   ::= <expr> <boolop> <expr> { (and | or) <boolexpr> }
<boolop>     ::= '==' | '!=' | '<' | '>' | '<=' | '>='

<expr>       ::= <term> [ ('+' | '-') <term> ]
<term>       ::= <factor> [ ('*' | '/' | '%') <factor> ]
<factor>     ::= ['+' | '-'] <number> | <load> | <value> | '(' <expr> ')'
<load>       ::= load '(' <text> ')'
<value>      ::= (<new> | <var>) { '.' <array> } [ '.' <int> ]

<new>        ::= new (<nzero> | <nrand> | <nfill>)
<nzero>      ::= zero '[' <expr> ']'
<nrand>      ::= rand '[' <expr> ']'
<nfill>      ::= fill '[' <expr> ',' <expr> ']'

<array>      ::= <show> | <sort> | <add> | <set> | <filter> | <remove> | <each> | <apply>
<show>       ::= show '(' ')'
<sort>       ::= sort '(' ')'
<add>        ::= add '(' <expr> ')'
<set>        ::= set '(' <expr> ',' <expr> ')'
<filter>     ::= filter '(' <var> '->' <boolexpr> ')'
<remove>     ::= remove '(' <var> '->' <boolexpr> ')'
<each>       ::= each '(' <var> '->' <statements> ')'
<apply>      ::= apply '(' <var> '->' <statements> ')'

<int>        ::= <at> | <size>
<at>         ::= at '(' <expr> ')'
<size>       ::= size '(' ')'
```

Laboratório de Linguagens de Programação
Prof. Andrei Rimsa Álvares

4. Instruções

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

```
$ mlambda
Usage: ./mlab [MiniLab File]
$ mlambda evenods.lab
Entre com o tamanho do vetor: 3
Valor 1: 5
Valor 2: 8
Valor 3: 2
Voce entrou com 2 par(es) 1 impar(es)
```

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
	Tipos inválidos

Exemplo de mensagem de erro:

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

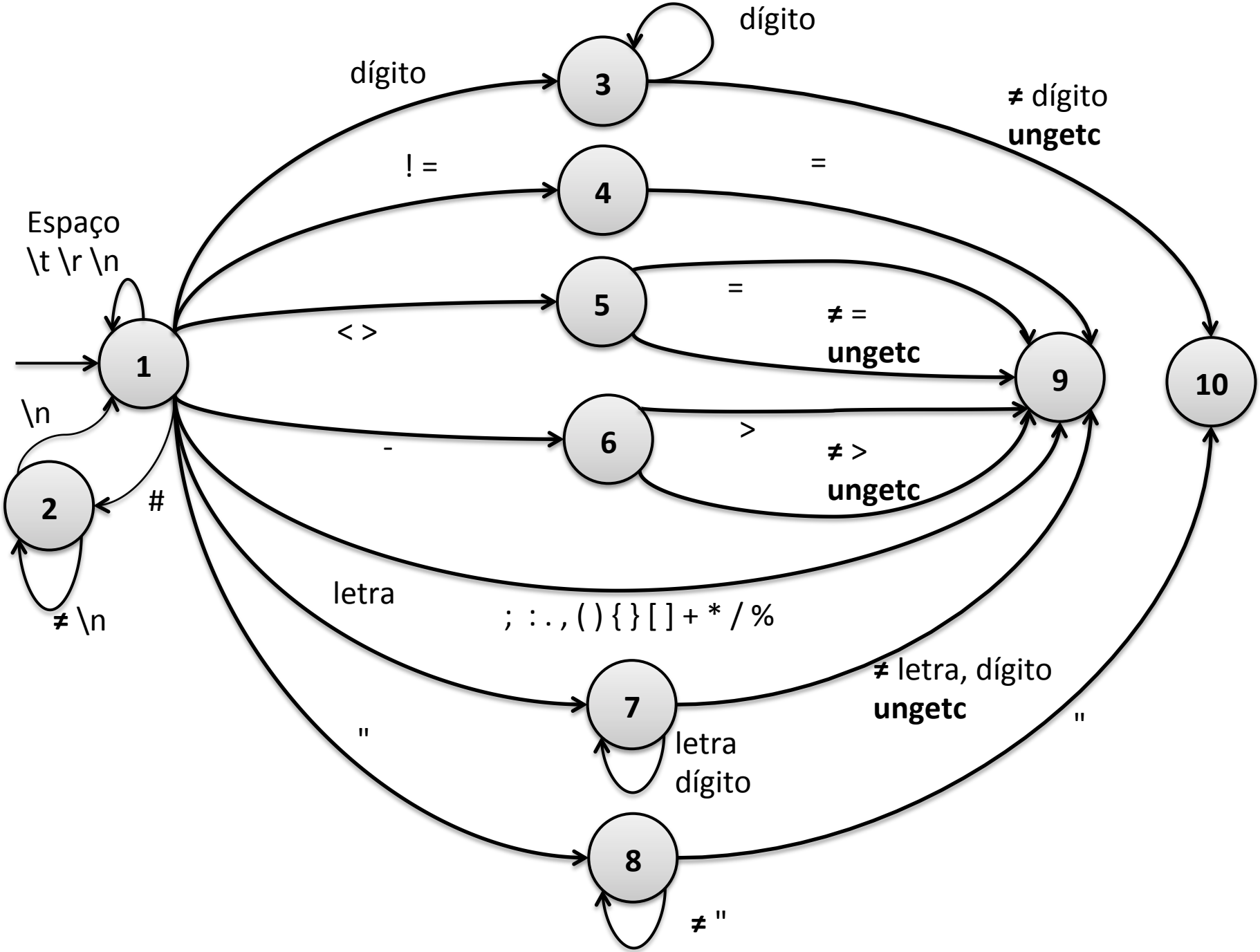
5. Avaliação

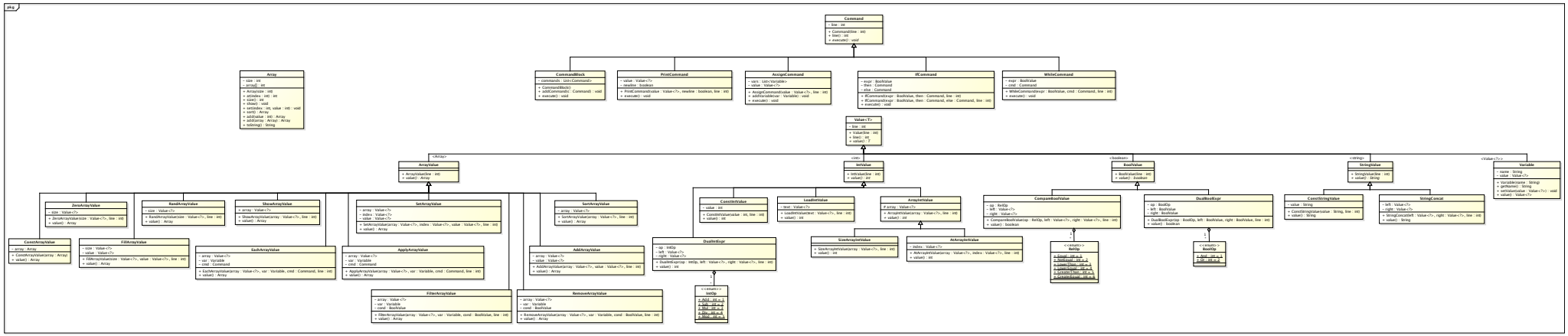
O trabalho deve ser feito em grupo de até dois alunos, sendo esse limite superior estrito. O trabalho será avaliado em 15 pontos, onde essa nota será multiplicada por um fator entre 0.0 e 1.0 para compor a nota de cada aluno individualmente. Esse fator poderá estar condicionado a apresentações presenciais 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 23:55 do dia 01/05/2017 (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.


dígito
dígito
3
≠ dígito
ungetc
! =
4
=
Espaço
\t \r \n
5
=
≠ =
ungetc
< >
1
9
10
\n
6
>
≠ >
ungetc
-
2
#
letra
; : . , ( ) { } [ ] + * / %
≠ letra, dígito
ungetc
7
"
"
letra
dígito
8
≠ \n
≠ "


pkg
Command
- line : int
+ CommandLine : int
+ line() : int
+ execute() : void
Array
- size : int
- array[] : int
+ Array(size : int)
+ at(index : int) : int
+ size() : int
+ show() : void
+ set(index : int, value : int) : void
+ sort() : void
+ add(value : int) : Array
+ add(array : Array) : Array
+ toString() : String
CommandBlock
- commands : List<Command>
+ CommandBlock()
+ addCommand(c : Command) : void
+ execute() : void
PrintCommand
- value : Value<?>
- newline : boolean
+ PrintCommand(value : Value<?>, newline : boolean, line : int)
+ execute() : void
AssignCommand
- vars : List<Variable>
- value : Value<?>
+ AssignCommand(value : Value<?>, line : int)
+ addVariable(var : Variable) : void
+ execute() : void
IfCommand
- expr : BoolValue
- then : Command
- else : Command
+ IfCommand(expr : BoolValue, then : Command, line : int)
+ IfCommand(expr : BoolValue, then : Command, else : Command, line : int)
+ execute() : void
WhileCommand
- expr : BoolValue
- cmd : Command
+ WhileCommand(expr : BoolValue, cmd : Command, line : int)
+ execute() : void
Value<T>
- line : int
+ Value(line : int)
+ line() : int
+ value() : T
<Array>
ArrayValue
+ ArrayValue(line : int)
+ value() : Array
<int>
IntValue
+ IntValue(line : int)
+ value() : int
<boolean>
BoolValue
+ BoolValue(line : int)
+ value() : boolean
<string>
StringValue
+ StringValue(line : int)
+ value() : String
<Value<?>>
Variable
- name : String
+ value() : Value<?>
+ Variable(name : String)
+ value(v : Value<?>) : void
+ set(v : Value<?>) : void
+ value() : Value<?>
ZeroArrayValue
- size : Value<?>
+ ZeroArrayValue(size : Value<?>, line : int)
+ value() : Array
RandArrayValue
- size : Value<?>
+ RandArrayValue(size : Value<?>, line : int)
+ value() : Array
ShowArrayValue
+ array : Value<?>
+ ShowArrayValue(array : Value<?>, line : int)
+ value() : Array
SetArrayValue
- array : Value<?>
- index : Value<?>
- value : Value<?>
+ SetArrayValue(array : Value<?>, index : Value<?>, value : Value<?>, line : int)
+ value() : Array
SortArrayValue
- array : Value<?>
+ SortArrayValue(array : Value<?>, line : int)
+ value() : Array
ConstIntValue
- value : int
+ ConstIntValue(value : int, line : int)
+ value() : int
LoadIntValue
- text : Value<?>
+ LoadIntValue(text : Value<?>, line : int)
+ value() : int
ArrayIntValue
array : Value<?>
+ ArrayIntValue(array : Value<?>, line : int)
+ value() : int
CompareBoolValue
- op : RelOp
- left : Value<?>
- right : Value<?>
+ CompareBoolValue(op : RelOp, left : Value<?>, right : Value<?>, line : int)
+ value() : boolean
DualBoolExpr
- op : BoolOp
- left : BoolValue
- right : BoolValue
+ DualBoolExpr(op : BoolOp, left : BoolValue, right : BoolValue, line : int)
+ value() : boolean
ConstStringValue
- value : String
+ ConstStringValue(value : String, line : int)
+ value() : String
StringConcat
- left : Value<?>
- right : Value<?>
+ StringConcat(left : Value<?>, right : Value<?>, line : int)
+ value() : String
ConstArrayValue
- array : Array
+ ConstArrayValue(array : Array)
+ value() : Array
FillArrayValue
- size : Value<?>
- value : Value<?>
+ FillArrayValue(size : Value<?>, value : Value<?>, line : int)
+ value() : Array
EachArrayValue
- array : Value<?>
- var : Variable
- cmd : Command
+ EachArrayValue(array : Value<?>, var : Variable, cmd : Command, line : int)
+ value() : Array
ApplyArrayValue
- array : Value<?>
- var : Variable
- cmd : Command
+ ApplyArrayValue(array : Value<?>, var : Variable, cmd : Command, line : int)
+ value() : Array
AddArrayValue
- array : Value<?>
- value : Value<?>
+ AddArrayValue(array : Value<?>, value : Value<?>, line : int)
+ value() : Array
FilterArrayValue
- array : Value<?>
- var : Variable
- cond : BoolValue
+ FilterArrayValue(array : Value<?>, var : Variable, cond : BoolValue, line : int)
+ value() : Array
RemoveArrayValue
- array : Value<?>
- var : Variable
- cond : BoolValue
+ RemoveArrayValue(array : Value<?>, var : Variable, cond : BoolValue, line : int)
+ value() : Array
DualIntExpr
- op : IntOp
- left : Value<?>
- right : Value<?>
+ DualIntExpr(op : IntOp, left : Value<?>, right : Value<?>, line : int)
+ value() : int
1
1
<< enum >>
IntOp
+ ADD : int = 1
+ SUB : int = 2
+ MUL : int = 3
+ DIV : int = 4
+ MOD : int = 5
SizeArrayIntValue
+ SizeArrayIntValue(array : Value<?>, line : int)
+ value() : int
AtArrayIntValue
- index : Value<?>
+ AtArrayIntValue(array : Value<?>, index : Value<?>, line : int)
+ value() : int
1
1
<< enum >>
RelOp
+ EQUAL : int = 1
+ NOTEQUAL : int = 2
+ LESSTHAN : int = 3
+ GREATERTHAN : int = 4
+ LESSTHANEQUAL : int = 5
+ GREATERTHANEQUAL : int = 6
1
1
<< enum >>
BoolOp
+ AND : int = 1
+ OR : int = 2