Criando um interpretador em Go: The hard way

Alex Sandro Garzão

THE DEVELOPER'S CONFERENCE

TDC Innovation - Março 2021

Agenda (ou alinhando expectativas)

Como surgiu o projeto

O que é o G-Portugol

O que é o gogpt, sua arquitetura, implementação, ...

Considerações finais

Não dá tempo para detalhar tudo....

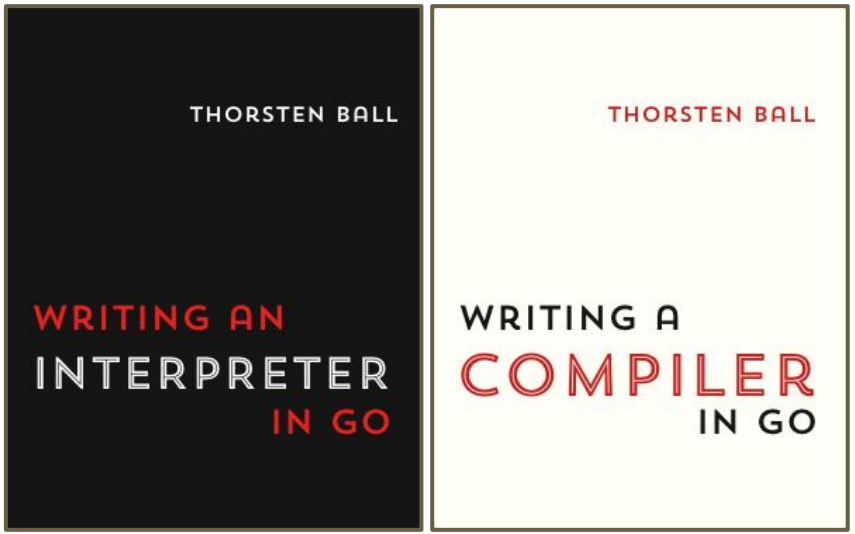
Quem sou

- Apaixonado por tecnologia :-)
  - Linguagens de programação, Compiladores, Máquinas virtuais, Geradores de código
  - Sistemas de alto desempenho
  - Projetos desafiadores

- Já atuei com “Toy compilers”
  - HoloC, UbiC, G-Portugol, Pascal para bytecode JVM
  - ST (IEC 61131-3) para ASM (80C51), ...
  - Implementações no gc “para aprendizado”

Como surgiu o projeto

Busca no Google: Golang compiler



- Ambos usam Go, TDD, apenas stdlib...
- TDD? Interpretador? Bora!!!
- Qual linguagem? G-Portugol!!!!

O que é o G-Portugol?

- É uma linguagem de programação (foco em aprendizado)
- Criada pelo Thiago Silva (1.1 é de 2006)
- Dialeto do portugol (ou português estruturado), com acentuação

Exemplo em G-Portugol :-)

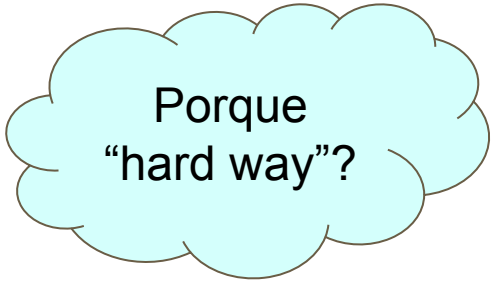
```
algoritmo olá_mundo;

início
    imprima("Olá mundo!");
fim
```

Também possui

- se/então/senão
- para, repita
- função, ...

O que é o gogpt?

- Interpretador, escrito em Golang :-)
- Entende e executa (interpreta) a linguagem G-Portugol
- TDD desde o primeiro “respiro”



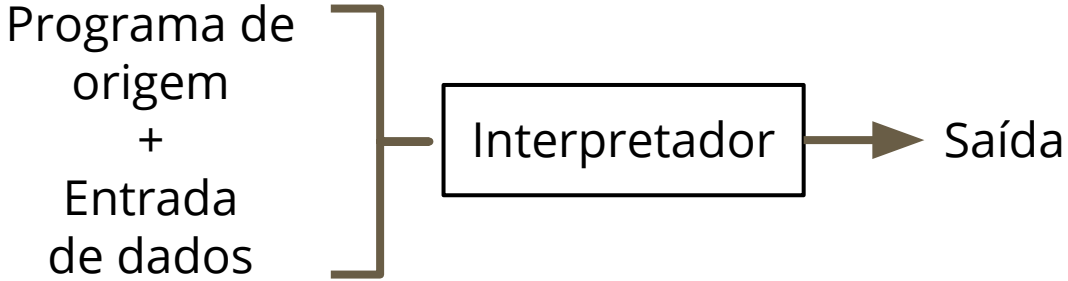
O que o gogpt faz?

```
algoritmo olá_mundo;

início
    imprima("Olá mundo!");
fim
```

```
~/personal/projects/gogpt-interpreter master ./cmd/gogpt/gogpt samples/hello_world.gpt
Olá mundo!

~/personal/projects/gogpt-interpreter master
```

O que é um interpretador?

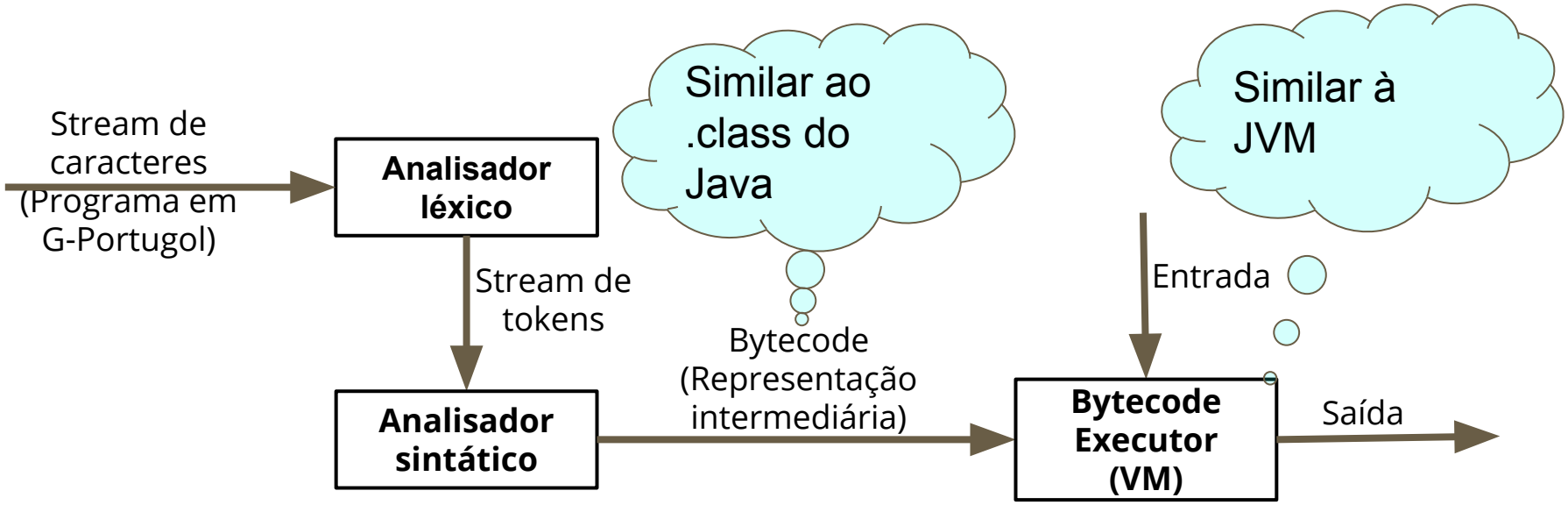
“Ao invés de produzir um programa (como um compilador faz), o interpretador diretamente executa as operações especificadas no programa de origem utilizando uma entrada de dados” [Aho, 2a edição]



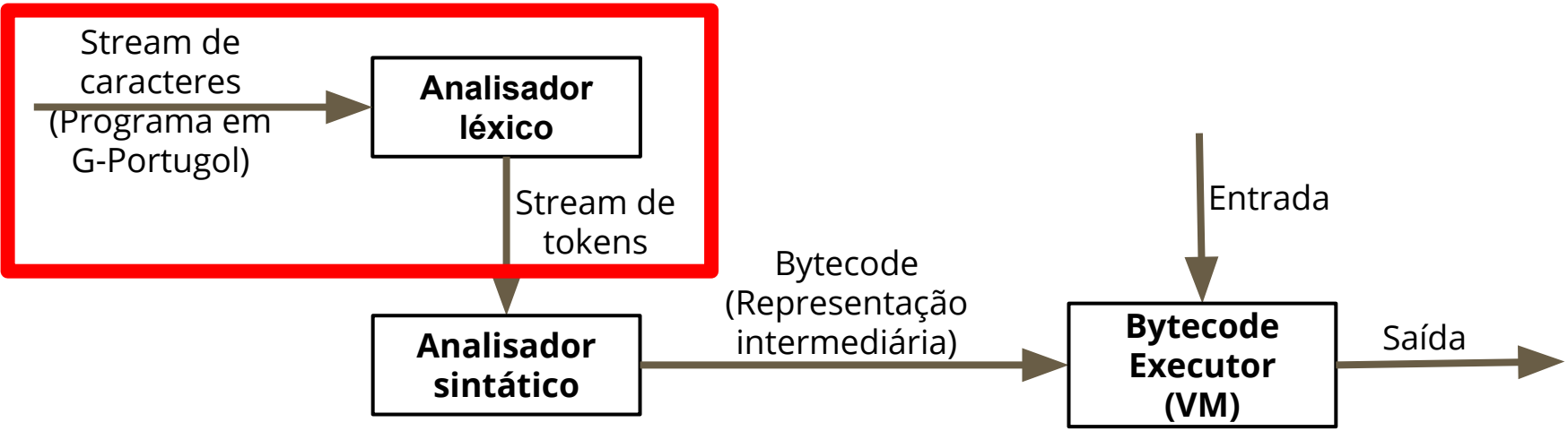
Meta inicial do gogpt: “Olá mundo!”

```
algoritmo olá_mundo;

início
    imprima("Olá mundo!");
fim
```

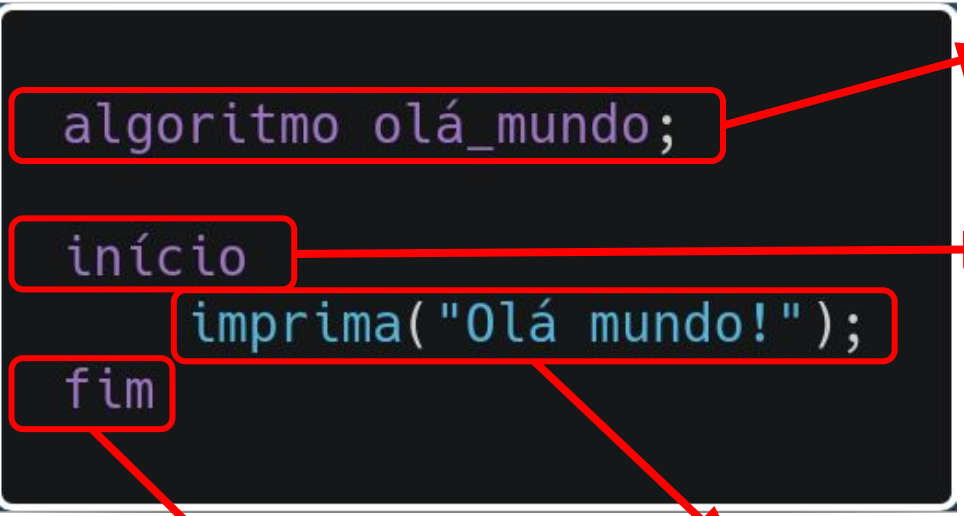
Arquitetura do gogpt e as etapas de interpretação



Analisador léxico



Analisador léxico



ALGORITHM
“**algorithm**”

IDENT
“**olá_mundo**”

SEMICOLON
“**;**”

BLOCKBEGIN
“**início**”

IDENT
“**imprima**”

LPAREN
“**(**”

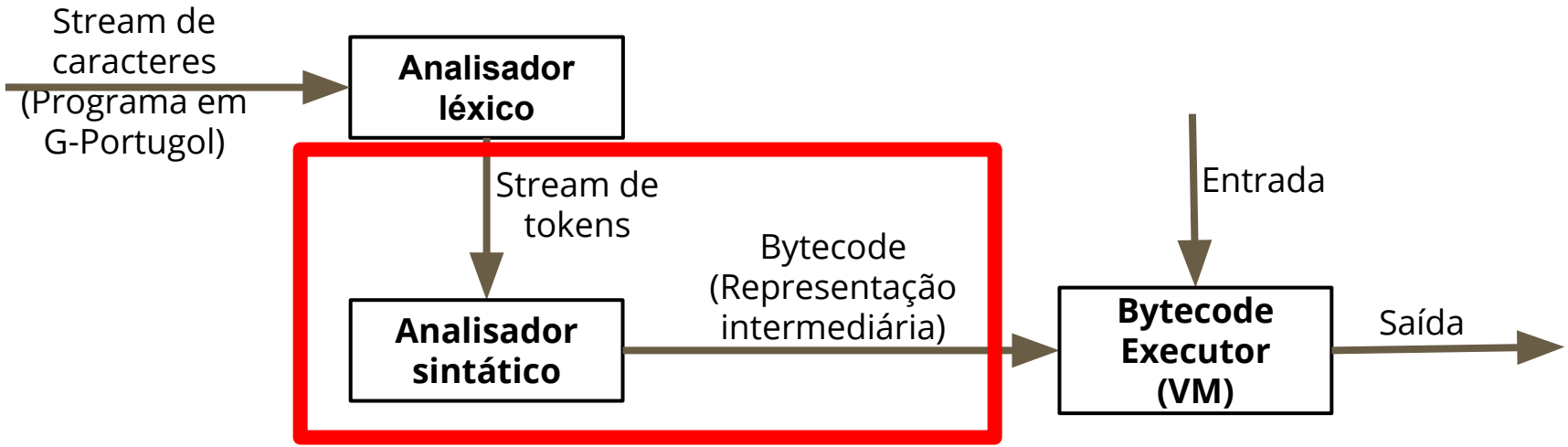
STRING
“**Olá mundo!**”

RPAREN
“**)**”

SEMICOLON
“**;**”

BLOCKEND
“**fim**”

Analisador sintático



Analisador sintático

Bytecode???

ALGORITHM
“**algorithm**”

IDENT
“**olá_mundo**”

SEMICOLON
“**;**”

BLOCKBEGIN
“**início**”

IDENT
“**imprima**”

LPAREN
“**(**”

STRING
“**Olá mundo!**”

RPAREN
“**)**”

SEMICOLON
“**;**”

BLOCKEND
“**fim**”

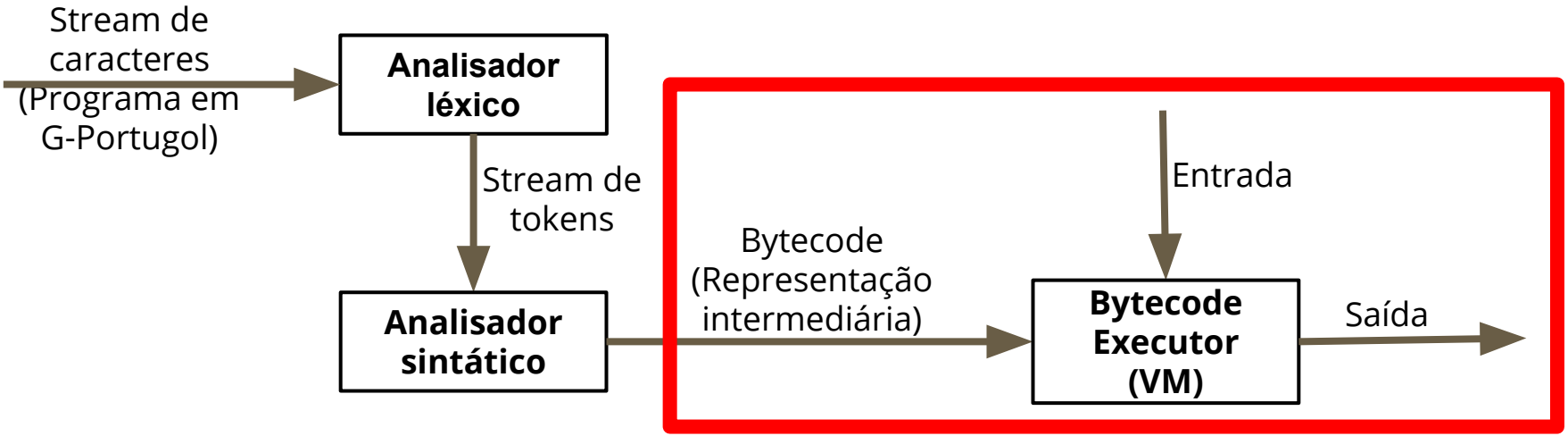
Constant Pool?

CONSTANT POOL		
IDX	TYPE	CONSTANT
0	STR	io.println
1	STR	Olá mundo!

Instructions?

INSTRUCTIONS	
OPCODE	CONSTANT
LDC 1	Olá mundo!
CALL 0	io.println

VM


Stream de caracteres (Programa em G-Portugol)
Analisador léxico
Stream de tokens
Analisador sintático
Bytecode (Representação intermediária)
Entrada
Bytecode Executor (VM)
Saída

VM

Startup?

CONSTANT POOL		
IDX	TYPE	CONSTANT
0	STR	io.println
1	STR	Olá mundo!

LDC?
CALL?

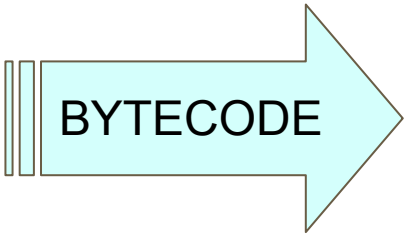
INSTRUCTIONS	
OPCODE	CONSTANT
LDC 1	Olá mundo!
CALL 0	io.println

Retira constante da pilha e imprime

Carrega a constante “Olá mundo!” na pilha...

Sumarizando a meta do gogpt: “Olá mundo!”

```
algoritmo olá_mundo;

início
    imprima("Olá mundo!");
fim
```



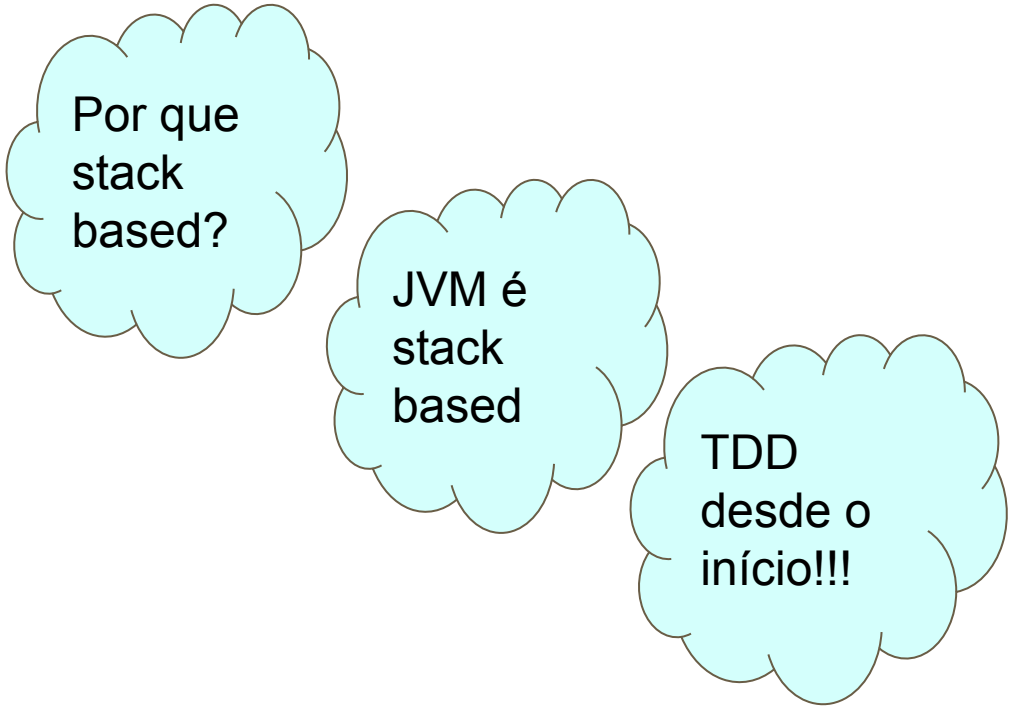
CONSTANT POOL		
IDX	TYPE	CONSTANT
0	STR	io.println
1	STR	Olá mundo!

INSTRUCTIONS	
OPCODE	CONSTANT
LDC 1	Olá mundo!
CALL 0	io.println

```
~/personal/projects/gogpt-interpreter master ./cmd/gogpt/gogpt samples/hello_world.gpt
Olá mundo!

~/personal/projects/gogpt-interpreter master
```

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - Constant pool (CP)
  - Stack interna
  - Instruções
    - LDC (LoaD Constant)
    - CALL
  - BytecodeExecutor
- **Parser**
- **gogpt = Parser + VM**



Falar é fácil! Show me the code :-)

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - **Constant pool (CP)**
  - Stack interna
  - Instruções
    - LDC
    - CALL
  - BytecodeExecutor
- **Parser**
- **gogpt = Parser + VM**

Teste: Adicionando dados na Constant Pool

```
1  func TestCpAddIntConstants(t *testing.T) {
2      cp := New()
3      assert.Equal(t, 0, cp.Add(123))
4      assert.Equal(t, 1, cp.Add(456))
5  }
```

Implementação inicial da Constant Pool

```
1  // CP has the items in a constant pool.
2  type CP struct {
3      constants []interface{}
4  }

6  // New creates a new constant pool.
7  func New() *CP {
8      cp := &CP{}
9      cp.constants = make([]interface{}, 0)
10     return cp
11 }

13 // Add adds a new item to the constant pool.
14 func (cp *CP) Add(item interface{}) int {
15     cp.constants = append(cp.constants, item)
16     return len(cp.constants) - 1
17 }
```

Teste: Obtendo dados da Constant Pool

```
1  func TestCpGetIntConstants(t *testing.T) {
2      cp := New()
3      assert.Equal(t, 0, cp.Add(123))
4      assert.Equal(t, 1, cp.Add(456))
5      v,   := cp.Get(0)
6      assert.Equal(t, 123, v)
7      v,   = cp.Get(1)
8      assert.Equal(t, 456, v)
9  }
```

Implementação do Get na Constant Pool

```
1  // Get gets an item from the constant pool.
2  func (cp *CP) Get(index int) (interface{}, error) {
3      res := cp.constants[index]
4
5      return res, nil
6  }
```

Teste: E se o índice não existe?

```
1  func TestCpGetIntConstantsError(t *testing.T) {
2      cp := New()
3      assert.Equal(t, 0, cp.Add(123))
4      assert.Equal(t, 1, cp.Add(456))
5      v, err := cp.Get(0)
6      assert.Equal(t, 123, v)
7      assert.NoError(t, err)
8      v, err = cp.Get(1)
9      assert.Equal(t, 456, v)
10     assert.NoError(t, err)
11     v, err = cp.Get(2)
12     assert.Equal(t, err, domain.ErrIndexNotFound)
13 }
```

Alterações no Get da Constant Pool

```
1  // Get gets an item from the constant pool.
2  func (cp *CP) Get(index int) (interface{}, error) {
3      if cp == nil || index > len(cp.constants)-1 {
4          return 0, domain.ErrIndexNotFound
5      }
6
7      res := cp.constants[index]
8
9      return res, nil
10 }
```

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - ~~Constant pool (CP)~~
  - **Stack interna**
  - Instruções
    - LDC
    - CALL
  - BytecodeExecutor
- **Parser**
- **gogpt = Parser + VM**

Stack: testes iniciais

- Adiciona 1 valor (Push) e valida o topo da pilha (Top)
- Adiciona 2 valores e valida
- Adiciona 1 valor, remove (Pop) e valida
- Adiciona e remove 2 valores
- Valida Stack Underflow
- ...

Implementação da Stack (New e Push)

```go
// Stack has the items of a stack.
type Stack struct {
    items []interface{}
}

// New creates a new stack.
func New() *Stack {
    s := &Stack{}
    s.items = make([]interface{}, 0)
    return s
}

// Push pushs a new item onto the stack.
func (s *Stack) Push(item interface{}) {
    s.items = append(s.items, item)
}
```

Implementação da Stack (Pop)

```go
// Pop pops an item from the stack.
func (s *Stack) Pop() (interface{}, error) {
    l := len(s.items)
    if l == 0 {
        return 0, domain.ErrStackUnderflow
    }

    res := s.items[l-1]
    s.items = s.items[:l-1]

    return res, nil
}
```

Implementação da Stack (Top)

```
1  // Top gets the top item on the stack.
2  func (s *Stack) Top() (interface{}, error) {
3      l := len(s.items)
4      if l == 0 {
5          return 0, domain.ErrStackUnderflow
6      }
7
8      res := s.items[l-1]
9
10     return res, nil
11 }
```

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - **Instruções**
    - **LDC**
    - CALL
  - BytecodeExecutor
- **Parser**
- **gogpt = Parser + VM**

Teste: Instrução LDC carregando ABC na stack

```
1  func TestValidLdcABC(t *testing.T) {
2      // CP 0: STR: "ABC"
3      cp := cp.New()
4      cpIndex := cp.Add("ABC")
5
6      // LDC 0
7      vars := vars.New()
8      st := stack.New()
9      stdin := infrastructure.NewFakeStdin()
10     stdout := infrastructure.NewFakeStdout()
11     ldc := New()
12     ldc.CpIndex = cpIndex
13     ldc.Execute(cp, vars, st, stdin, stdout)
14     stv, _ := st.Top()
15     assert.Equal(t, "ABC", stv)
16 }
```

Implementação de LDC (FetchOperands e Execute)

```
1  // FetchOperands gets the opcode operands.
2  func (i *LDCInst) FetchOperands(fetch ...) error {
3      var err error
4      i.CpIndex, err = fetch.Next()
5      return err
6  }
7
8  // Execute receives the context and runs the opcode.
9  func (i *LDCInst) Execute(cp, vars, st, stdin, stdout) error {
10     cpv, err := cp.Get(i.CpIndex)
11     if err != nil ...
12
13     st.Push(cpv)
14     return nil
15 }
```

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - **Instruções**
    - ~~LDC~~
    - **CALL**
  - BytecodeExecutor
- **Parser**
- **gogpt = Parser + VM**

Teste: Instrução CALL io.println

```
1  func TestCallStringHelloWorld(t *testing.T) {
2
3      cp := cp.New()                                // CP
4      printlnIndex := cp.Add("io.println")          // 0: STR "io.println"
5      messageIndex := cp.Add("Hello World!")        // 1: STR "Hello World!"
6      argsCountIndex := cp.Add(1)                   // 2: INT 1
7
8      ldc := ldci.New()                             // OPCODE: LDC 1 (Hello World!)
9      ldc.CpIndex = messageIndex
10     ldc.Execute(cp, vars, st, fakeStdin, fakeStdout)
11     ldc = ldci.New()                              // OPCODE: LDC 2 (1)
12     ldc.CpIndex = argsCountIndex
13     ldc.Execute(cp, vars, st, fakeStdin, fakeStdout)
14     call := New()                                 // OPCODE: CALL 0 (io.println)
15     call.CpIndex = printlnIndex
16     call.Execute(cp, vars, st, fakeStdin, fakeStdout)
17     assert.Equal(t, "Hello World!", fakeStdout.LastLine)
18 }
```

Implementação de CALL (FetchOperands e Execute)

```
1  // FetchOperands gets the opcode operands.
2  func (i *CALLInst) FetchOperands(fetch) error {
3      var err error
4      i.CpIndex, err = fetch.Next()
5      return err
6  }
7
8  // Execute receives the context and runs the opcode.
9  func (i *CALLInst) Execute(cp, vars, st, stdin, stdout) error {
10     cpv, err := cp.Get(i.CpIndex)
11     if err != nil ...
12
13     if cpv == "io.println" {
14         if err := i.printlnImpl(st, stdout); err != nil ...
15     } else if cpv == "io.readln" {
16         if err := i.readlnImpl(st, stdin); err != nil ...
17     }
18     return nil
19 }
```

Requisitos do gogpt (0.1)

- **VM (stack based)**
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - ~~Instruções~~
    - ~~LDC~~
    - ~~CALL~~
  - **BytecodeExecutor**
- **Parser**
- **gogpt = Parser + VM**

Teste: BCE executando um hello world

```
1  func TestBCECompleteHelloWorld(t *testing.T) {
2      ...
3      cp := cp.New()                          // CP
4      printlnIndex := cp.Add("io.println")    // 0: STR "io.println"
5      messageIndex := cp.Add("Hello World!") // 1: STR "Hello World!"
6      argsCountIndex := cp.Add(1)             // 2: INT 1
7
8      bc := bytecode.New()                        // OPCODES
9      bc.Add(instructions.LDC, messageIndex)   // LDC 1 (Hello World!)
10     bc.Add(instructions.LDC, argsCountIndex) // LDC 2 (1)
11     bc.Add(instructions.CALL, printlnIndex)  // CALL 0 (io.println)
12
13     bce := New(bc)
14     err := bce.Run(cp, vars, st, stdin, stdout)
15     assert.Nil(t, err)
16     assert.Equal(t, "Hello World!", stdout.LastLine)
17 }
```

Implementação de BCE (New)

```go
// BytecodeExecutor is responsible for execute the bytecode.
type BytecodeExecutor struct {
    bc           *bytecode.Bytecode
    instructions map[int]instructions.InstructionImplementation
    ip           int
}

// New creates a new BytecodeExecutor.
func New(bc *bytecode.Bytecode) *BytecodeExecutor {
    bce := &BytecodeExecutor{
        ip:           0,
        bc:           bc,
    }

    bce.instructions = make(map[int]instructions.InstructionImplementation)
    bce.instructions[instructions.LDC] = ldci.New()
    bce.instructions[instructions.CALL] = calli.New()
    return bce
}
```

Implementação de BCE (Run e Next)

```go
// Run receives the context (CP, vars, stack, stdin and stdout)
// and runs the bytecode.
func (bce *BytecodeExecutor) Run(cp, vars, st, stdin, stdout) error {
    for {
        opcode, err := bce.Next()
        if err != nil {
            return nil
        }

        instruction, _ := bce.instructions[opcode]
        instruction.FetchOperands(bce)
        instruction.Execute(cp, vars, st, stdin, stdout)
    }
}

// Next returns the next bytecode.
func (bce *BytecodeExecutor) Next() (code int, err error) {
    code, err = bce.bc.Get(bce.ip)
    bce.ip++
    return
}
```

Requisitos do gogpt (0.1)

- ~~VM (stack based)~~
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - ~~Instruções~~
    - ~~LDC~~
    - ~~CALL~~
  - ~~BytecodeExecutor~~
- **Parser**
- **gogpt = Parser + VM**

Teste: Parser do “Olá mundo!”

```
1  func TestValidHelloWorldAlgorithm(t *testing.T) {
2      alg := `algoritmo olá_mundo;
3              início
4                  imprima("Olá mundo!");
5              fim`
6      l := lexer.New(alg)
7      p := New(l)
8      err := p.Parser()
9      assert.Nil(t, err)
10 }
```

Teste: Parser gera bytecode esperado do “Olá mundo!”

Definindo o algoritmo

Definindo a CP esperada

Definindo o BC esperado

Executando o parser e validando a CP e o BC esperados

```
1  func TestBytecodeHelloWorldAlgorithm(t *testing.T) {
2      alg := `algoritmo olá_mundo;
3              início
4                  imprima("Olá mundo!");
5              fim`
6      expectedCp := cp.New()
7      printlnIndex := expectedCp.Add("io.println")
8      messageIndex := expectedCp.Add("Olá mundo!")
9      argsCountIndex := expectedCp.Add(1)
10
11     expectedBc := bytecode.New()
12     expectedBc.Add(instructions.LDC, messageIndex)
13     expectedBc.Add(instructions.LDC, argsCountIndex)
14     expectedBc.Add(instructions.CALL, printlnIndex)
15
16     p := New(lexer.New(alg))
17     err := p.Parser()
18     assert.Nil(t, err)
19     assert.Equal(t, expectedCp, p.GetCP())
20     assert.Equal(t, expectedBc, p.GetBC())
21 }
```

Requisitos do gogpt (0.1)

- ~~VM (stack based)~~
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - ~~Instruções~~
    - ~~LDC~~
    - ~~CALL~~
  - ~~BytecodeExecutor~~
- ~~Parser~~
- **gogpt = Parser + VM**

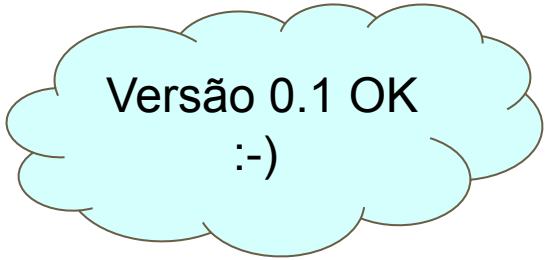
gogpt: função principal

```
1  func main() {
2      filename, _ := getFilenameFromArgs()
3      algorithm, _ := ioutil.ReadFile(filename)
4      l := lexer.New(string(algorithm))
5      p := parser.New(l)
6      p.Parser()
7      bce := bce.New(p.GetBC())
8      stdin := infrastructure.NewStdin()
9      stdout := infrastructure.NewStdout()
10     st := stack.New()
11     vars := vars.New()
12     bce.Run(p.GetCP(), vars, st, stdin, stdout)
13 }
```

$ cat && gogpt hello_world.gpt

Requisitos do gogpt (0.1)

- ~~VM (stack based)~~
  - ~~Constant pool (CP)~~
  - ~~Stack interna~~
  - ~~Instruções~~
    - ~~LDC~~
    - ~~CALL~~
  - ~~BytecodeExecutor~~
- ~~Parser~~
- ~~gogpt = Parser + VM~~



Depois disso, novas funcionalidades (0.2)

Declaração de variáveis

Função leia

Vários argumentos

```
algoritmo qual_o_seu_nome;

variáveis
    nome: literal;
    idade: literal;
fim-variáveis

início
    imprima("Qual o seu nome?");
    nome := leia();
    imprima("Qual a sua idade?");
    idade := leia();
    imprima("Olá ", nome, ". Você tem ", idade, " anos!!!");
fim
```

$ cat && gogpt what_is_your_name_and_how_old.gpt

O que falta?

- Expressões relacionais
- Expressões aritméticas
- Estruturas de repetição
- Tipos numéricos, lógicos, matrizes

Considerações finais

- Já fiz projetos similares em C/C++
- Mas este em Go ficou MUITO mais simples :-)
- Go tem uma boa documentação, bom tooling, ...

Referências

- Repositório do gogpt
- Golang
- TDD em Golang
- Aho, 2nd editon
- G-Portugol
- Manual do G-Portugol

Dúvidas????

Alex Sandro Garzão

alexgarzao@gmail.com

https://www.linkedin.com/in/alexgarzao/

https://github.com/alexgarzao

https://twitter.com/alexgarzao