Como TDD facilitou a implementação de um interpretador para uma linguagem em Português

Alex Sandro Garzão

THE DEVELOPER'S CONFERENCE

TDC POA - Novembro 2019

Agenda (ou alinhando expectativas)

Como surgiu o projeto

Quem é o G-Portugol

Quem é o gogpt, sua arquitetura do gogpt

Testes, testes e mais testes...

Considerações finais

Não dá tempo para tudo....

Quem sou

- Engenheiro de Software na Zenvia
- Minhas paixões (algumas)
  - Meus filhos, livros, cinema, andar de bike
  - Linguagens de programação, compiladores, máquinas virtuais
  - Domínios complexos, algoritmos, processamento em tempo real
  - Go, Python, C, C++, ...
  - Falar sobre tecnologia
  - Código bem feito :-)
- Já atuei com “Toy compilers”
  - HoloC, UbiC, G-Portugol, Pascal para bytecode JVM
  - ST (IEC 61131-3) para ASM (80C51), ...
  - gc

Vamos nos conhecer um pouco...

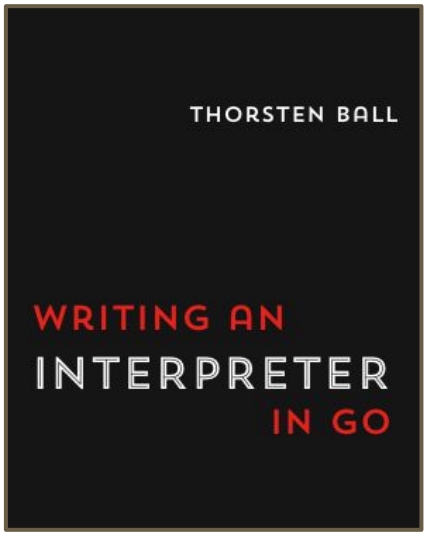
Quem tem conhecimento sobre como um interpretador e/ou compilador funciona?

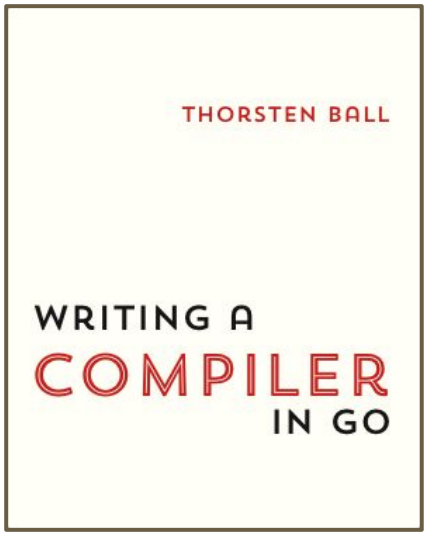
Quem já atuou na implementação de um?

Quem já olhou e/ou analisou o código de um?

Como surgiu o projeto

Busca no Google: Golang compiler



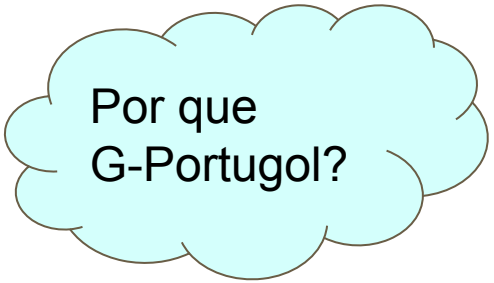


- Ambos usam Go, TDD, apenas stdlib...
- TDD? Interpretador? Hum... Vou tentar...
- Olhei o capítulo de exemplo
- Qual linguagem? G-Portugol!!!!

Quem é o G-Portugol?

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

Exemplo em G-Portugol :-)




```
algoritmo olá_mundo;

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

Também possui

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

Quem 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?

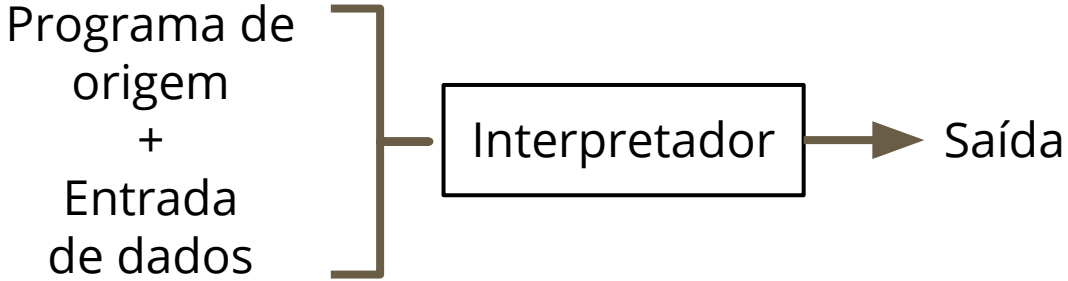
```
algoritmo olá_mundo;

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

```
➜  gogpt-interpreter git:(master) ./gogpt/gogpt samples/hello_world.gpt
Olá mundo!
➜  gogpt-interpreter git:(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]

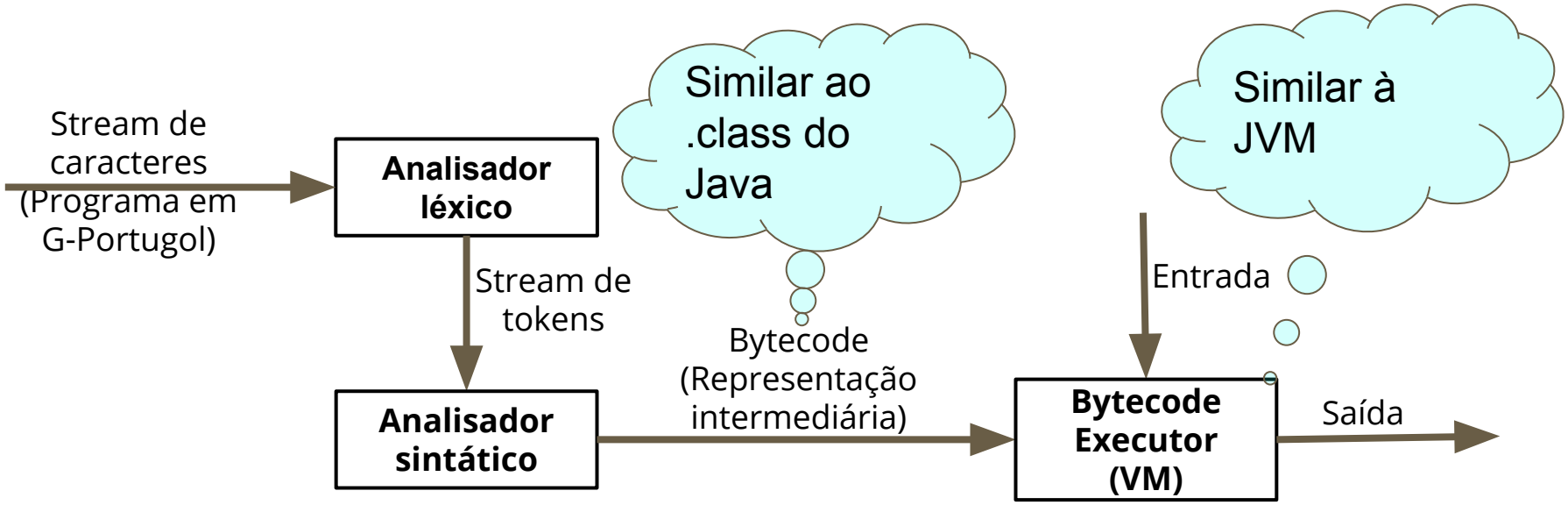


Meta inicial do gogpt: “Olá mundo!”

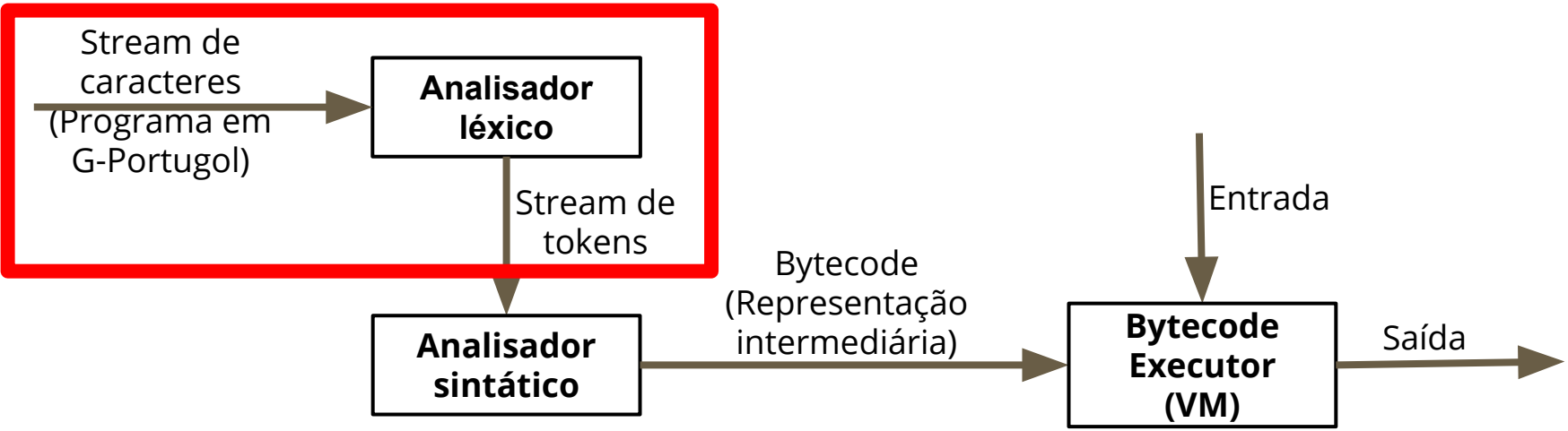
```
algoritmo olá_mundo;

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

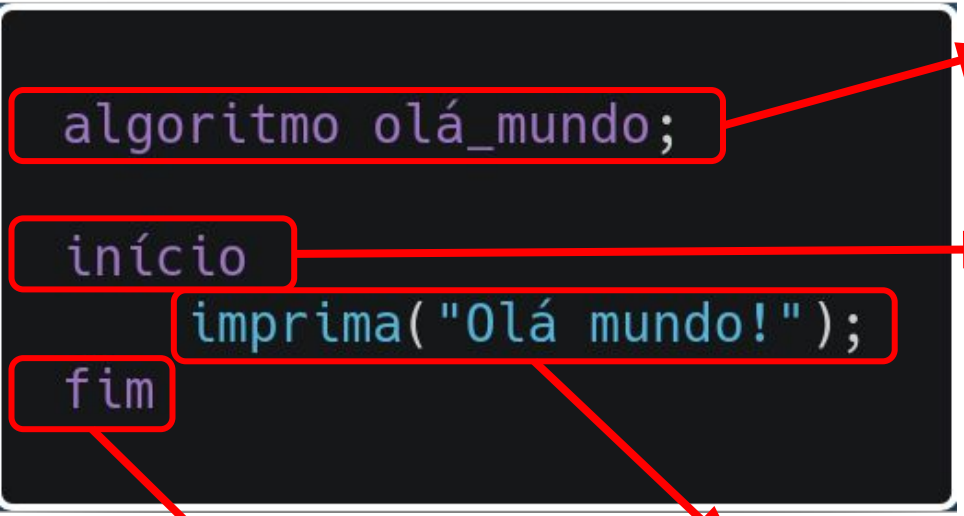
Arquitetura do gogpt e/ou etapas de interpretação



Analisador léxico



Analisador léxico



ALGORITHM
“**algorithm**”

IDENT
“**olá_mundo**”

SEMICOLON
“**;**”

BLOCKBEGIN
“**início**”

IDENT
“**imprima**”

LPAREN
“**(**”

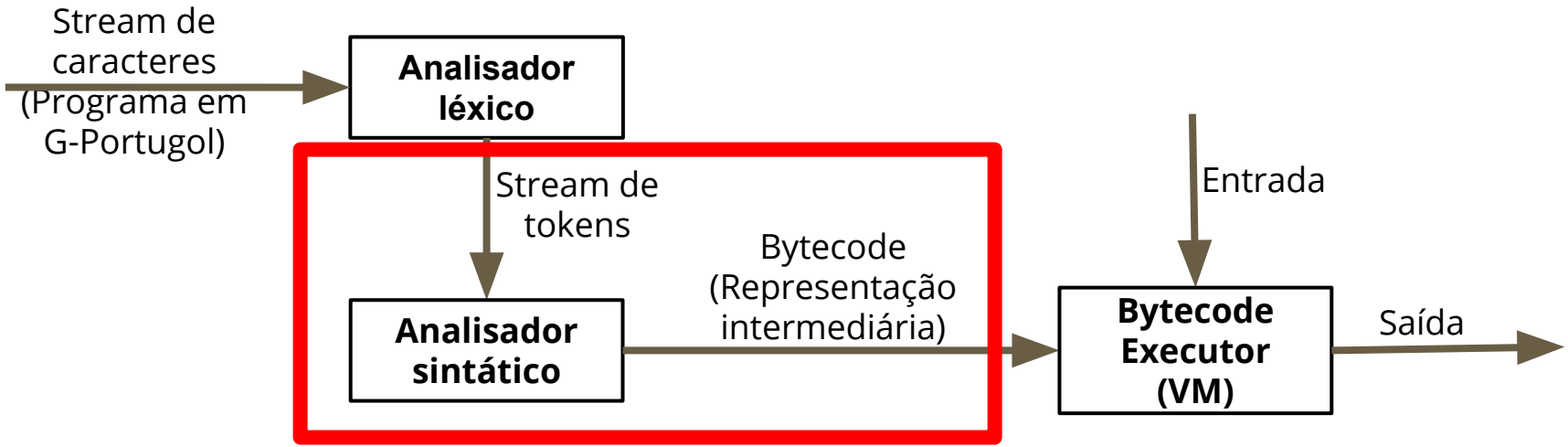
STRING
“**Olá mundo!**”

RPAREN
“**)**”

SEMICOLON
“**;**”

BLOCKEND
“**fim**”

Analisador sintático



Analisador sintático

ALGORITHM
“**algorithm**”

IDENT
“**olá_mundo**”

SEMICOLON
“**;**”

BLOCKBEGIN
“**início**”

IDENT
“**imprima**”

LPAREN
“**(**”

STRING
“**Olá mundo!**”

RPAREN
“**)**”

SEMICOLON
“**;**”

BLOCKEND
“**fim**”

Constant Pool?

Bytecode???

Instructions?

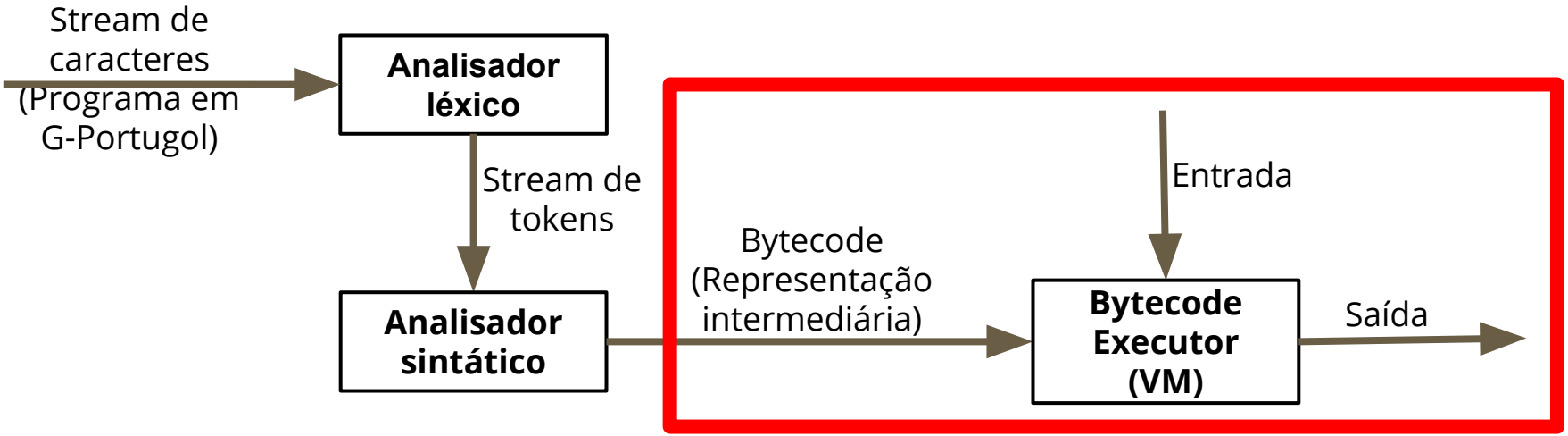
CONSTANT POOL

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

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

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

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

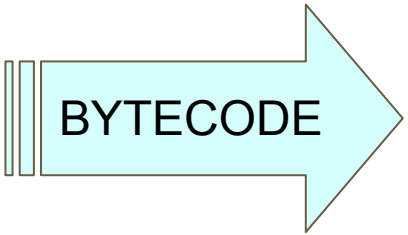
Carrega a constante “Olá mundo!” na pilha...

Retira constante da pilha e imprime

Sumarizando a meta do gogpt: “Olá mundo!”

```
algoritmo olá_mundo;

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

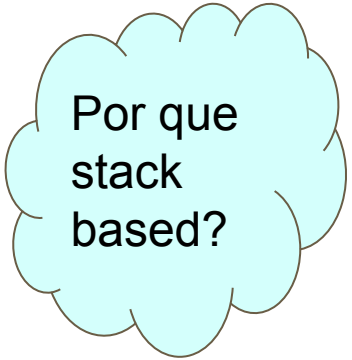


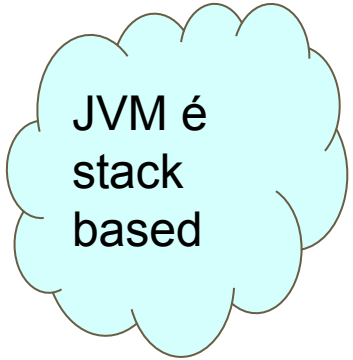
CONSTANT POOL		
IDX	TYPE	CONSTANT
0	STR	io.println
1	STR	Olá mundo!

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

Requisitos do gogpt

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





Falar é fácil.... show me the ~~code~~ tests :-)

Requisitos do gogpt

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

Teste: Adicionando dados na Constant Pool

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

Implementação inicial da Constant Pool

```go
// CP has the items in a constant pool.
type CP struct {
    constants []interface{}
}
```

```go
// New creates a new constant pool.
func New() *CP {
    cp := &CP{}
    cp.constants = make([]interface{}, 0)
    return cp
}
```

```go
// Add adds a new item to the constant pool.
func (cp *CP) Add(item interface{}) int {
    cp.constants = append(cp.constants, item)

    return len(cp.constants) - 1
}
```

Teste: Obtendo dados da Constant Pool

```
func TestCpGetIntConstants(t *testing.T) {
    cp := New()
    assert.Equal(t, 0, cp.Add(123))
    assert.Equal(t, 1, cp.Add(456))
    v, _ := cp.Get(0)
    assert.Equal(t, 123, v)
    v, _ = cp.Get(1)
    assert.Equal(t, 456, v)
}
```

Implementação do Get na Constant Pool

```
// Get gets an item from the constant pool.
func (cp *CP) Get(index int) (interface{}, error) {
    res := cp.constants[index]

    return res, nil
}
```

Teste: E se o índice não existe?

```
func TestCpGetIntConstantsError(t *testing.T) {
    cp := New()
    assert.Equal(t, 0, cp.Add(123))
    assert.Equal(t, 1, cp.Add(456))
    v, err := cp.Get(0)
    assert.Equal(t, 123, v)
    assert.NoError(t, err)
    v, err = cp.Get(1)
    assert.Equal(t, 456, v)
    assert.NoError(t, err)
    v, err = cp.Get(2)
    assert.EqualError(t, err, "Index not found")
}
```

Alterações no Get da Constant Pool

```
// Get gets an item from the constant pool.
func (cp *CP) Get(index int) (interface{}, error) {
    if index > len(cp.constants)-1 {
        return 0, errors.New("Index not found")
    }

    res := cp.constants[index]

    return res, nil
}
```

Requisitos do gogpt

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

Teste: Adicionando dados na Stack

```go
func TestStackPushOneValue(t *testing.T) {
    s := New()
    s.Push(111)
    v, _ := s.Top()
    assert.Equal(t, 111, v)
}
```

Implementação inicial da Stack (New e Push)

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

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

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

Implementação inicial da Stack (Top)

```go
// Top gets the top item on the stack.
func (s *Stack) Top() (interface{}, error) {
    l := len(s.items)
    res := s.items[l-1]

    return res, nil
}
```

Teste: Adicionando 2 valores na Stack

```
func TestStackPushTwoValues(t *testing.T) {
    s := New()
    s.Push(111)
    s.Push(222)
    v, _ := s.Top()
    assert.Equal(t, 222, v)
}
```

Teste: Adicionando e removendo um valor

```
func TestStackPushAndPopOneValue(t *testing.T) {
    s := New()
    s.Push(111)
    v, _ := s.Pop()
    assert.Equal(t, 111, v)
}
```

Implementação de Stack.Pop

```go
// Pop pops a item from the stack.
func (s *Stack) Pop() (interface{}, error) {
    l := len(s.items)
    res := s.items[l-1]
    s.items = s.items[:l-1]

    return res, nil
}
```

Teste: Adicionando e removendo dois valores

```
func TestStackPushAndPopTwoValues(t *testing.T) {
    s := New()
    s.Push(111)
    s.Push(222)
    v, _ := s.Pop()
    assert.Equal(t, 222, v)
    v, _ = s.Pop()
    assert.Equal(t, 111, v)
}
```

Teste: Stack underflow

```
func TestStackPopError(t *testing.T) {
    s := New()
    _, err := s.Pop()
    assert.EqualError(t, err, "Stack underflow")
    s.Push(111)
    _, err = s.Pop()
    assert.NoError(t, err)
    _, err = s.Pop()
    assert.EqualError(t, err, "Stack underflow")
}
```

Ajustes em Stack.Pop

```
// Pop pops a item from the stack.
func (s *Stack) Pop() (interface{}, error) {
    l := len(s.items)
    if l == 0 {
        return 0, errors.New("Stack underflow")
    }

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

    return res, nil
}
```

Requisitos do gogpt

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

Teste: Instrução LDC carregando 123 na stack

```
func TestValidLdcInt123(t *testing.T) {
    st := stack.New()
    stdout := infrastructure.NewStdout()
    // CP 0: (INT) 123
    cp := cp.New()
    cpIndex := cp.Add(123)
    // LDC 0
    ldc := New()
    ldc.CpIndex = cpIndex
    ldc.Execute(cp, st, stdout)
    stv, _ = st.Top()
    assert.Equal(t, 123, stv)
}
```

Implementação de LDC.New e .Execute

```
func New() *LDCInst {
    return &LDCInst{instructions.Instruction{
        Opcode: instructions.LDC,
    }, 0}
}

func (i *LDCInst) Execute(cp *cp.CP, st *stack.Stack,
        stdout instructions.StdoutInterface) error {

    cpv, _ := cp.Get(i.CpIndex)
    st.Push(cpv)

    return nil
}
```

Requisitos do gogpt

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

Teste: Instrução CALL io.println

```
func TestCallStringHello(t *testing.T) {
    // Cria CP, Stack, stdout...
    // CP 0: STR "io.println"
    printlnIndex := cp.Add("io.println")
    // CP 1: STR "Hello World!"
    messageIndex := cp.Add("Hello World!")
    // LDC 1 (Hello World!)
    ldc := ldci.New()
    ldc.CpIndex = messageIndex
    ldc.Execute(cp, st, stdout)
    // CALL 0 (io.println)
    call := New()
    call.CpIndex = printlnIndex
    call.Execute(cp, st, stdout)
    assert.Equal(t, "Hello World!", stdout.LastLine)
}
```

Implementação de CALL.New e .Execute

```
func New() *CALLInst {
    return &CALLInst{instructions.Instruction{
        Opcode: instructions.CALL,
    }, 0}
}

func (i *CALLInst) Execute(cp *cp.CP, st *stack.Stack,
        stdout instructions.StdoutInterface) error {

    cpv, _ := cp.Get(i.CpIndex)
    if cpv == "io.println" {
        text, _ := st.Pop()
        stdout.Println(text)
    }

    return nil
}
```

Requisitos do gogpt

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

Teste: BCE executando instrução LDC 222

```
func TestBCERunningLDC222(t *testing.T) {
    // CP 0: 222
    cp := cp.New()
    cpIndex := cp.Add(222)
    st := stack.New()
    stdout := infrastructure.NewFakeStdout()
    bc := bytecode.New()
    bc.Add(instructions.LDC, cpIndex)
    bce := New(bc)
    err := bce.Run(cp, st, stdout)
    assert.Nil(t, err)
    stv, _ := st.Top()
    assert.Equal(t, 222, stv)
}
```

Implementação de BCE.New e .Run

```
func New(bc *bytecode.Bytecode) *BytecodeExecutor {
    bce := ...
    bce.instructions[instructions.LDC] = ldci.New()
    bce.instructions[instructions.CALL] = calli.New()
    return bce
}

func (bce *BytecodeExecutor) Run(cp *cp.CP, st *stack.Stack,
        stdout instructions.StdoutInterface) error {
    for {
        opcode, _ := bce.Next()
        instruction, _ := bce.instructions[opcode]
        instruction.FetchOperands(bce)
        instruction.Execute(cp, st, stdout)
    }
}
```

Teste: BCE executando um hello world completo

```
func TestBCECompleteHelloWorld(t *testing.T) {
    ...
    // CP 0: STR "io.println"
    printlnIndex := cp.Add("io.println")
    // CP 1: STR "Hello World!"
    messageIndex := cp.Add("Hello World!")
    // LDC 1 (Hello World!)
    bc.Add(instructions.LDC, messageIndex)
    // CALL 0 (io.println)
    bc.Add(instructions.CALL, printlnIndex)
    bce := New(bc)
    err := bce.Run(cp, st, stdout)
    assert.Nil(t, err)
    assert.Equal(t, "Hello World!", stdout.LastLine)
}
```

Requisitos do gogpt

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

Teste: Lexer ignora caracteres inúteis

```
func TestIfUselessCharsHasBeenRemoved(t *testing.T) {
    l := New("   algoritmo\t\t  \t\t\n\t\r meuid;")
    assert.Equal(t, &Token{ALGORITHM, ALGORITHM}, l.NextToken())
    assert.Equal(t, &Token{IDENT, "meuid"}, l.NextToken())
    assert.Equal(t, &Token{SEMICOLON, SEMICOLON}, l.NextToken())
    assert.Equal(t, &Token{EOF, EOF}, l.NextToken())
}
```

Lexer: Outros testes

- Keywords válidos, inválidos
- Delimitadores
- IDs
- Strings válidos, inválidos
- Inteiros válidos, inválidos
- ...

Requisitos do gogpt

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

Teste: Parser de um algoritmo vazio

```
func TestValidEmptyAlgorithm(t *testing.T) {
    alg := `algoritmo olá_mundo;
            início
            fim`
    l := lexer.New(alg)
    p := New(l)
    pr := p.Parser()
    assert.Equal(t, true, pr.Parsed)
}
```

Teste: Parser do “Olá mundo!”

```
func TestValidHelloWorldAlgorithm(t *testing.T) {
    alg := `algoritmo olá_mundo;
            início
                imprima("Olá mundo!");
            fim`
    l := lexer.New(alg)
    p := New(l)
    pr := p.Parser()
    assert.Equal(t, true, pr.Parsed)
}
```

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

```
func TestBytecodeHelloWorldAlgorithm(t *testing.T) {
    alg := `algoritmo olá_mundo;
            início
                imprima("Olá mundo!");
            fim`
    expectedCp := cp.New()
    printlnIndex := expectedCp.Add("io.println")
    messageIndex := expectedCp.Add("Olá mundo!")
    l := lexer.New(alg)
    p := New(l)
    expectedBc := bytecode.New()
    expectedBc.Add(instructions.LDC, messageIndex)
    expectedBc.Add(instructions.CALL, printlnIndex)
    pr := p.Parser()
    assert.Equal(t, true, pr.Parsed)
    assert.Equal(t, expectedCp, p.GetCP())
    assert.Equal(t, expectedBc, p.GetBC())
}
```

$ go test ./...

Requisitos do gogpt

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

gogpt: função principal

Compilei, ajustei os imports… Funcionou :-)

```
func main() {
    filename := getFilenameFromArgs()
    algorithm := ioutil.ReadFile(filename)
    l := lexer.New(string(algorithm))
    p := parser.New(l)
    pr := p.Parser()
    bce := bce.New(p.GetBC())
    stdout := infrastructure.NewStdout()
    st := stack.New()
    bce.Run(p.GetCP(), st, stdout)
}
```

$ cat && gogpt hello_world.gpt

Depois disso, 2.5 KG de refactors :-)

- Instruções
- Lexer
- Parser
- Reorganização dos módulos internos

E várias novas funcionalidades

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

- Primeiro projeto “não convencional” que usei TDD desde o início
- Testes foram fundamentais :-)
  - Fiz vários experimentos em termos de design de código, com go, parser, ...
  - Com mock de E/S, conseguia testar rapidamente
- Em outro projeto, fiz compilador + VM (+- 2002)
  - Tinham muitos testes funcionais
  - Mas nada de testes unitários
  - Tempo de desenvolvimento da 0.1 - 5 dias, 0.2 - 15 dias, ...
  - Qualidade do código BEM duvidosa :-)
  - Refactor? Só se muito necessário
- Go é bem legal (sou fã), boa documentação, bom tooling
  - Excelente suporte para testes unitários (faz parte do 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