Tutorial em português 🇧🇷

3BC é uma linguagem de programação em baixo nível e de fácil aprendizado, que trabalha na forma de uma máquina virtual semelhante a um computador da década de 60, mas com uma arquitetura muito inusitada que possui processadores de apenas 3 bits.

Apesar de ter um aspecto para ser uma linguagem esotérica, tem uma boa capacidade de uso genérico para resolver problemas computacionais, e certas vantagens com a implementação em embarcados e microcontroladores.

Foi introduzida por um brasileiro no natal de 2020 após completar um mês de desenvolvimento, sendo um projeto para estudos e também como uma prova conceitual sobre cartões perfurados ser legível e prático tanto para humanos quanto máquinas.

Como fazer um “Ola mundo!”

Estas etapas irão guiá-lo para executar o exemplo hello world e para ter seu primeiro programa 3BC em execução em sua máquina, vamos começar.

Depois de baixar o binário na página *downalod*, basta descompactar o arquivo e utilizar a linha de comando para excutar o código-fonte usando o interpretador de acordo com seu sistema operacional: windows, mac os, unix ou linux.

Exemplo: `olamundo.3bc`

```
MODE NILL 0d2
STRC NILL 'o'
STRC NILL 'l'
STRC NILL 'a'
STRC NILL ' '
STRC NILL 'm'
STRC NILL 'u'
STRC NILL 'n'
STRC NILL 'd'
STRC NILL 'o'
STRC NILL '!'
```

windows

utilizando ‘terminal’ ou ‘cmd’ digite:

```
C:\Users\nicod\Downloads\3bc-windows-32> 3bc.exe helloworld.3bc
hello world!
C:\Users\nicod\Downloads\3bc-windows-32>
```

mac / unix / linux

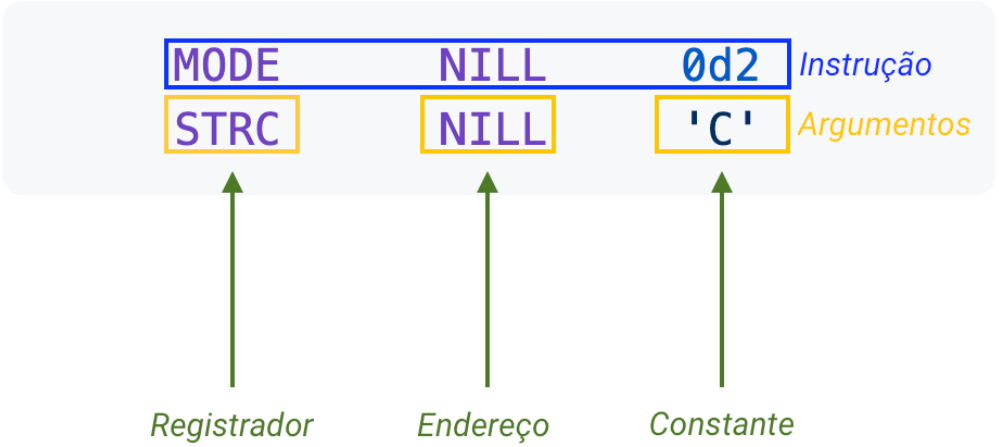
utilizando o shell de sua preferencia digite:

```
$ ./3bc helloworld.3bc
hello world!
$
```

Estrutura

Um grande ponto para facilitar o aprendizado, é a sua sintaxe muito bem organizada da linguagem. Com uma rápida visualização separada em linhas e colunas, onde destas possui um tamanho fixo de parâmetros, mesmo quando não utilizados.

Uma linha do programa é consequentemente uma instrução, que possui três parâmetros de maneira obrigatória. São separados pela respectivamente ordem de registrador, endereço, constante.



Registrador

O registrador define o comportamento da instrução seja imprimir na tela, capturar entradas, somar, etc.

Deve ser utilizado mnemónicos como `mode`, `math` e `nill` ou algumas expressões literais que representam seu opcode de 0 até 7. Se oriente pelo guia *cheatsheet*, onde explica detalhadamente sobre cada instrução.

Conforme o registrador é necessário passar ou não valores de endereço e constante. Pode tanto ser obrigatórios quanto restritivos, ou não aceitar ambos na mesma instrução, para evitar ambiguidade.

Endereço

Endereço é um espaço na memória utilizado para armazenar determinado dado, alguns conjuntos de instruções dependem disso.

Deve ser utilizar uma expressão literal. É importante destacar que utilizar *hash* tal como :minha_variavel em pequenos programas não há problema, mas estruturas monoliticas podém haver conflitos entre endereços.

Constante

Constante seria um valor fixo e imutável.

Sintaxe

Expressões Literais

Não existe nenhuma tipagem na linguagem, apenas dados brutos; Para facilitar a utilização, o interpretador aceita diversas representações destes valores, tais como:

Descrição	**Representação**
Caracter ASCII	`'Z'`
Decimal positivo	`9` `0d9` `0i9`
Decimal negativo	`-9` `-0d9` `-0i9`
Octal positivo/negativo	`0o7` `-0o7`
Binário positivo/negativo	`0b1` `-0b1`
HexaDecimal positivo/negativo	`0xF` `-0xF`
Gerador de Hash	`:minha_funcao`

Descrição	Representação
Vazio ou Zero	`0` `0d0` `0i0` `0o0` `0b0` `0x0` `'\0'` `NILL`

Etiquetas

Conhecida como *labels* são marcações no programa onde podem acontecer saltos condicionais ou chamadas de procedimentos, nenhuma etiqueta pode ser remarcada. É possível fazer pulos em qualquer direção do algorítimo.

Efetuando marcações

Quando ambas colunas registrador e endereço existirem valores nulos, qualquer valor constante será considerado demarcação de etiqueta.

```
NILL NILL 0x01
```

Dica de marcação

A Expressão literal gerador de *hash* tem o intuito de facilitar a organização das *labels* em seu código, assim não depender de apenas marcações numéricas.

```
NILL NILL :inicio_do_loop
```

Exemplo

```
MODE NILL 0d8
GOTO NILL :entrada # Salte para o ponto de 'entrada'

MODE NILL 0d2
STRC NILL 'o' # Não executar
STRC NILL 'l' # Não executar
STRC NILL 'a' # Não executar
STRC NILL ' ' # Não executar
NILL NILL :entrada
STRC NILL 'a' # Executar
STRC NILL 'm' # Executar
STRC NILL 'i' # Executar
STRC NILL 'g' # Executar
STRC NILL 'o' # Executar
```

Resultado esperado:

```
amigo
```

Paradigmas

Não estruturado

O paradigma não estruturado pode ser utilizado para diversas ocasiões como elaborar desvios de seleção e condição, fazer laços de repetição. e até mesmo um programa completo rápido e eficiente. Porém seu uso em excesso pode trazer grandes dificuldades de manuntenção e comportamentos inesperados.

MODO: 0d9

nome	octal	bit	description
goto	1	001	saltar para a etiqueta incondicionalmente
fgto	2	010	saltar para a etiqueta se a memória auxiliar estiver preenchida
zgto	3	011	saltar para a etiqueta se a memória auxiliar for vazia

nome	octal	bit	description
pgto	4	100	saltar para a etiqueta se a memória auxiliar for positiva
ngto	5	101	saltar para a etiqueta se a memória auxiliar for negativa

Laços de repetição

Infinito

Imprimir ‘cafe’ ininterruptamente.

```
MODE NILL 0d2
NILL NILL :loop
STRX NILL 0xCAFE
MODE NILL 0d9
GOTO NILL :loop
```

Faça enquanto

Imprimr ‘...’ utilizando *do while*.

```
# Valor de :var é 3
MODE NILL 0d6
ALOC :var 0d3

# Imprimir '.'
MODE NILL 0d02
NILL NILL :do_while
STRC NILL '.'

# Adicionar -1 em :var
MODE NILL 0d8
PUSH :var NILL
MODE NILL 0d11
MATH NILL -1
MODE 0 0d08
PULL :var 0

# Repetir enquanto var: for diferente de 0
MODE NILL 0d9
```

```
FGTO NILL :do_while
```

Enquanto

Imprimr '...' utilizando *while*.

```
# Valor de :var é 3
MODE NILL 0d6
ALOC :var 3

# Enquanto :var for diferente de 0
MODE NILL 0d8
NILL NILL :where_entry
PUSH :var NILL
MODE NILL 0d9
ZGTO NILL :where_exit

# Imprimir '.'
MODE NILL 0d2
STRC NILL '.'

# Subtrair 1 de :var
MODE NILL 0d8
PUSH :var NILL
MODE NILL 0d12
MATH NILL 1
MODE NILL 0d8
PULL :var NILL

# Final do enquanto
MODE NILL 0d9
GOTO NILL :where_entry
NILL NILL :where_exit
```

Desvio condicional

Se Diferente

Imprimir '!=' se a entrada for diferente de 5.

```
# Capturar teclado e armazenar em :cmp
MODE NILL 0d4
```

```
STRI :cmp NILL

# Comparar a negação (:cmp != 5)
MODE NILL 0d8
PUSH :cmp NILL
MODE NILL 0d12
MATH NILL 5
MODE NILL 0d9
ZGTO NILL :if_equal

# Imprimir '!='
MODE NILL 0d2
STRC NILL '!'
STRC NILL '='

# Fim programa.
NILL NILL :if_equal
```

Se Igual, Maior ou Menor

Imprimir ‘n<5’ se menor número for menor, imprimir ‘n=5’ se o número for exatamente 5 ou imprimir ‘n>5’ se for maior. Onde n sera a entrada de teclado.

```
# Capturar teclado e armazenar em :cmp
MODE NILL 0d3
STRI :cmp NILL

# Comparar com 5
MODE NILL 0d8
PUSH :cmp NILL
MODE NILL 0d12
MATH NILL 5
MODE NILL 0d9
NGTO NILL :if_less
ZGTO NILL :if_equal
PGTO NILL :if_greater

# Imprimir '<'
MODE NILL 0d2
NILL NILL :if_less
STRC NILL '<'
MODE NILL 0d9
```

```
GOTO NILL :if_exit

# Imprimir '='
MODE NILL 0d2
NILL NILL :if_equal
STRC NILL '='
MODE NILL 0d9
GOTO NILL :if_exit

# Imprimir '>'
MODE NILL 0d2
NILL NILL :if_greater
STRC NILL '>'
MODE NILL 0d9
GOTO NILL :if_exit

# Imprimir '5'
MODE NILL 0d2
NILL NILL :if_exit
STRI NILL 5
```

Procedimental

O Paradigma procedimental (popularmente chamado de procedural), traz a vantagem de fazer uma pilha de processos, onde existe um ponto de retorno no mesmo local de sua chamada. Isso pode facilitar em escrever grandes programas.

```
# Saltar para ponto de entrada
MODE NILL 0d9
GOTO NILL :entry

# Procedimento :print_bar
MODE NILL 0d2
NILL NILL :print_bar # Etiqueta
STRC NILL 'B'
STRC NILL 'a'
STRC NILL 'r'
MODE NILL 0d41 # Modo de retorno
BACK NILL NILL # Retornar incondicionalmente

# Procedimento :print_foo
```

```
MODE NILL 0d2
NILL NILL :print_foo # Inicio do processo
STRC NILL 'F'
STRC NILL 'o'
STRC NILL 'o'
MODE NILL 0d41
BACK NILL NILL # Fim do processo

# Ponto de entrada
MODE NILL 0d42   # Chamar procedimentos e retornar
NILL NILL :entry # Ponto de entrada
CALL NILL :print_foo # Chamar :print_foo
CALL NILL :print_bar # Chamar :print_bar
```