

# Documentação: Projeto do Jogo da Velha em MASM

João Pedro Gava Ribeiro e Rodrigo Belniok Czelusniak

22 de junho de 2022

## Fluxograma do Código em C

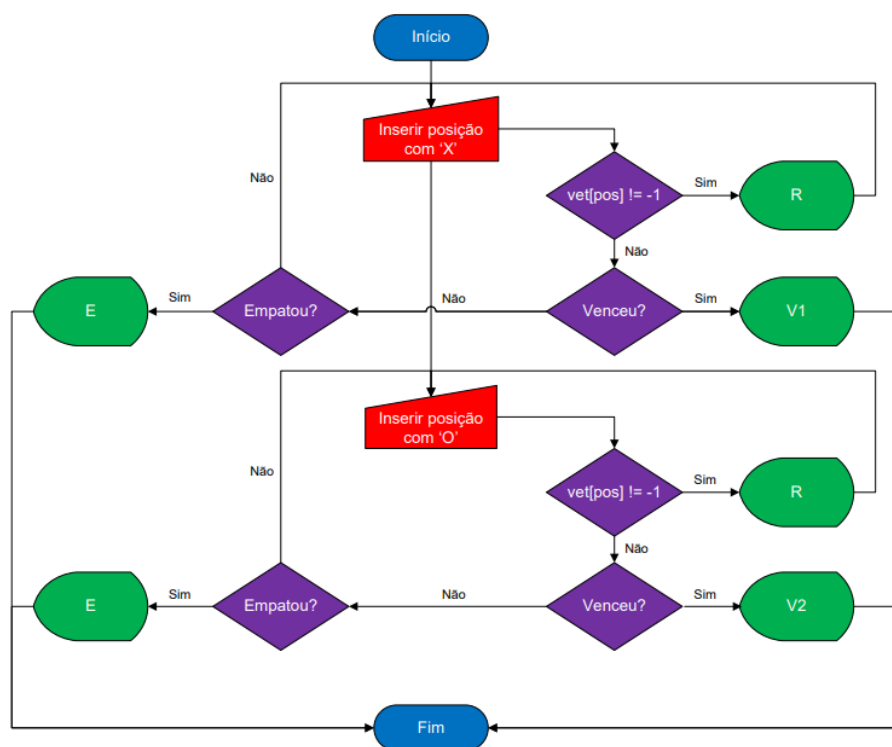


Figura 1: Fluxograma do Código na Linguagem C

## Legenda

Aqui foram dispostas as legendas das possíveis saídas conforme o fluxograma mostrado acima.

**R** Solicitação para nova entrada do usuário, já que a última foi considerada inválida.

*No programa em Assembly:* <valor de entrada invalido>, <posicao ja jogada> ou <invalid integer>.

**E** O jogo resultou em empate ("deu velha").

*No programa em Assembly:* <jogo empatado>.

**V1** O jogo teve como vitorioso o jogador 1 (ou o jogador 'X').

*No programa em Assembly:* <o jogador X ganhou>.

**V2** O jogo teve como vitorioso o jogador 2 (ou o jogador 'O').

*No programa em Assembly:* <o jogador O ganhou>.

## Código em C

Para um primeiro teste com o escopo de facilitar uma implementação posterior em Assembly, em especial no Microsoft Macro Assembler (MASM) pela IDE do Visual Studio 2019, foi produzido o seguinte código em linguagem C. Assim, promove-se um esclarecimento acerca do uso do comando goto: apesar dele não ser indicado em uma perspectiva de programação estruturada, ele representa muito bem os desvios condicionais existentes na linguagem de montador (do Assembly).

[CÓDIGO EM C DO PROJETO NO REPLIT](#)

[CÓDIGO EM C DO PROJETO PARA DOWNLOAD](#)

[TEMPLATE DA CONFIGURAÇÃO INICIAL PARA O MASM x86](#)

## Código em MASM

Em primeiro lugar, na implementação em MASM foram considerados os seguintes tipos de entradas inválidas pelo usuário:

- Posição menor que 1;
- Posição maior que 9;
- Posição já preenchida anteriormente;
- Valor informado não é um número.

As duas primeiras têm como saída que o valor de entrada é inválido, enquanto a terceira informará que a posição já foi jogada. Todavia, a quarta não precisou ser implementada, porque ela já é tratada pela biblioteca de E/S utilizada, com a saída de inteiro inválido.

Nesse sentido, foi necessário instalar e configurar a biblioteca [Irvine32](#) para a utilização plena de entrada e saída (E/S) por terminal, com a consequente interação com o usuário. Assim, para a impressão, ao EDI é atribuído o OFFSET da string desejada (Ex: `mov edi, offset posicao`) para ocorrer então a chamada (`call`) da função de biblioteca `writestring`, sendo a string em questão impressa na tela. O ENTER ou quebra de linha possui uma estrutura especial declarada no `.data` juntos às demais, tal que: `quebralinha BYTE ' ', 13, 10, 0`.

Por fim, por se tratar do MASM é indicado que o programa seja rodado em sistema operacional Windows compatível.

[CÓDIGO EM MASM DO PROJETO](#)

[PROGRAMA EXECUTÁVEL DO PROJETO](#)

**IMPORTANTE!** Caso você venha a baixar o executável no seu computador com o Windows Defender ou antivírus similar ativo, indicamos que o desative, senão ele poderá alertar acerca da

existência de falso positivo de vírus, a exemplo dos processos:

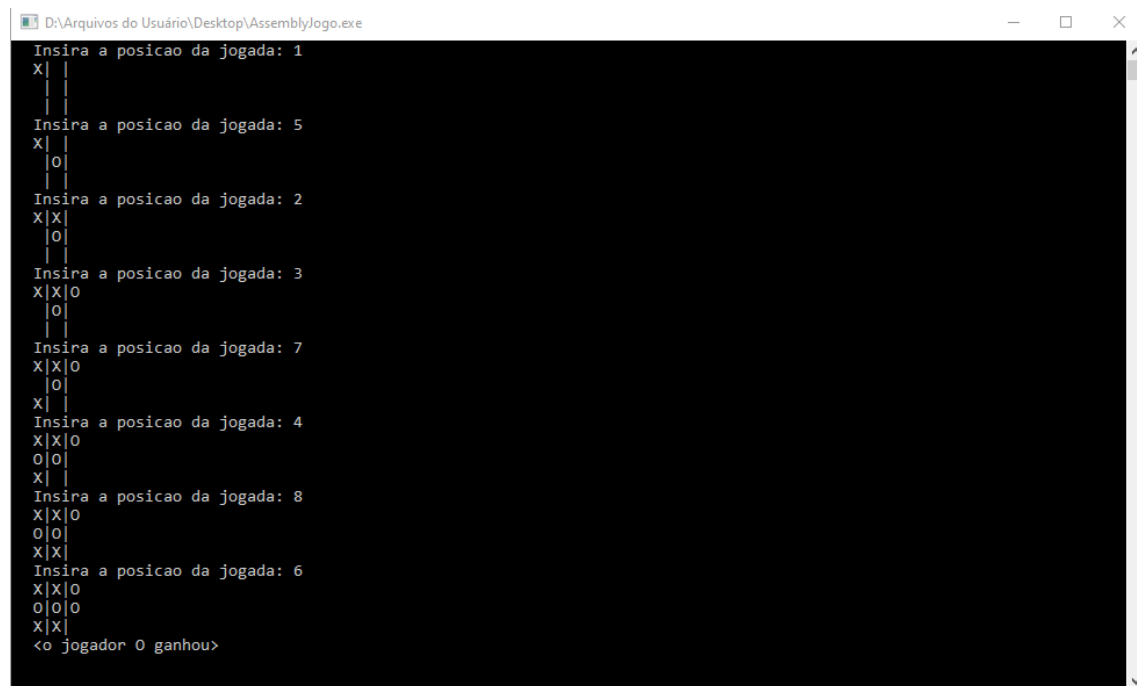
Trojan:Win32/Sabsik.TE.A!ml

Win32/Contebrew.A!ml

## Funcionamento

Em se tratando do funcionamento do jogo, ele inicia perguntando qual a posição desejada para o jogador 1 ('X') com a impressão de "Insira a posicao da jogada: ". Feita a escolha de uma posição inválida, sairá uma daquelas opções de saída predefinidas na tela, solicitando uma nova posição para aquele jogador. Feita a inserção de posição válida, ele irá mostrar o tabuleiro. Então, ele trocará para o jogador 2 ('O') e repetirá o processo já descrito até que o jogo venha a dar empate com a impressão de "<jogo empatado>" ou no momento que algum jogador vença, seja o jogador 1 com "<o jogador X ganhou>" ou o jogador 2 com "<o jogador O ganhou>".

Para um melhor entendimento do código-fonte do programa em Assembly, esse foi disposto com os respectivos comentários na sequência.



```
D:\Arquivos do Usuário\Desktop\Assembly\logo.exe
Insira a posicao da jogada: 1
X|
|
|
|
Insira a posicao da jogada: 5
X|
|O|
|
|
Insira a posicao da jogada: 2
X|X|
|O|
|
|
Insira a posicao da jogada: 3
X|X|O
|O|
|
|
Insira a posicao da jogada: 7
X|X|O
|O|
|X|
|
Insira a posicao da jogada: 4
X|X|O
O|O|
|X|
|
Insira a posicao da jogada: 8
X|X|O
O|O|
X|X|
|
Insira a posicao da jogada: 6
X|X|O
O|O|O
X|X|
<o jogador O ganhou>
```

Figura 2: Exemplo de terminal durante uma rodada do Jogo da Velha em MASM

## Código-fonte Comentado

```
.386
.model flat, stdcall
.stack 4096

ExitProcess PROTO, dwExitCode:DWORD

include Irvine32.inc
```

```

.data

;Conjunto de strings para serem impressas
jogada byte "Insira a posicao da jogada: ",0
vertical byte "|",0
horizontal byte "---", 0
x byte "X", 0
o byte "O", 0
space byte " ", 0
quebralinha BYTE ' ', 13, 10, 0
printavitoriax BYTE "<o jogador X ganhou>", 0
printavitoriao BYTE "<o jogador O ganhou>", 0
printaempate BYTE "<jogo empatado>", 0
invalido BYTE "<valor de entrada invalido>", 0
posicao BYTE "<posicao ja jogada>", 0


;Variável de input do usuário
myvar byte 20 DUP(?)


;Vetor de 9 numeros
dwArray dword 9 dup (?)


.code
main PROC


;Move vetor de 9 unidades para registrador esi
mov esi, offset dwArray


;Inicialização dos registradores
xor     ecx, ecx
xor     edx, edx
xor     ebx, ebx
xor     eax, eax


@@:
;Inicia todas as posições do vetor com "ESPAÇO"
mov [esi + 4 * eax], offset space


;Loop para percorrer o vetor com as posições do tabuleiro
inc eax
cmp eax, 9
jne @B


jogo:

```

```

;Chamada para o jogador entrar a posição
mov edx, offset jogada
call writestring
call readint

;Validação para valor maior que 9
cmp eax, 9
jg invalid

;Validação para valor menor que 1
cmp eax, 1
jl invalid

;Inicializa a posição com índice 0
dec eax

;Validação para posição já preenchida
mov edx, offset space
cmp [esi + 4 * eax], edx
jne utilizado

xor edx, edx
inc ecx
cmp ecx, 1
jne d
mov [esi + 4 * eax], offset x
cmp ecx, 2
jne u
d:

;Se ecx = 1, seta a posição escolhida pelo jogador em 0
mov [esi + 4 * eax], offset o

;Zera ecx, ou seja, jogador 0 jogou e muda a vez para X!
xor ecx, ecx
u:

;Inicialização de edi e eax
xor edi, edi
xor eax, eax
@@:

;Métodos abaixo serão usados para realizar a print do jogo em andamento

mov edx, [esi + 4 * eax]
call writestring
inc eax

```

```

mov edx, offset vertical
call writestring
mov edx, [esi + 4 * eax]
call writestring
inc eax
mov edx, offset vertical
call writestring
mov edx, [esi + 4 * eax]
call writestring
inc eax

inc edi

mov edx, offset quebralinha
call writestring
cmp edi, 3
jne @B

;Inicialização de eax, edi e edx
xor eax, eax
xor edi, edi
xor edx, edx

;Na sequência serão testadas todas as 16 situações de vitória de alguém,
;sendo feitos os desvios necessários para as linhas respectivas

condicao1:
xor edi, edi
xor edx, edx
condicao1a:

mov edx, [esi + 4 * 0]
mov edi, offset x
cmp edx, edi
jne condicao1b
mov edi, [esi + 4 * 1]
cmp edx, edi
jne condicao1b
mov edx, [esi + 4 * 2]
cmp edx, edi
je vitoriax

condicao1b:
xor edi, edi
xor edx, edx
condicao1c:

```

```

mov edx, [esi + 4 * 3]
mov edi, offset x
cmp edx, edi
jne condicao1d
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao1d
mov edx, [esi + 4 * 5]
cmp edx, edi
je vitoriax

```

```

condicao1d:
xor edi, edi
xor edx, edx
condicao1e:

```

```

mov edx, [esi + 4 * 6]
mov edi, offset x
cmp edx, edi
jne condicao2
mov edi, [esi + 4 * 7]
cmp edx, edi
jne condicao2
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriax

```

```

condicao2:
xor edi, edi
xor edx, edx
condicao2a:

```

```

mov edx, [esi + 4 * 0]
mov edi, offset o
cmp edx, edi
jne condicao2b
mov edi, [esi + 4 * 1]
cmp edx, edi
jne condicao2b
mov edx, [esi + 4 * 2]
cmp edx, edi
je vitoriao

```

```

condicao2b:
xor edi, edi
xor edx, edx
condicao2c:

```

```

mov edx, [esi + 4 * 3]
mov edi, offset o
cmp edx, edi
jne condicao2d
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao2d
mov edx, [esi + 4 * 5]
cmp edx, edi
je vitoriao

```

```

condicao2d:
xor edi, edi
xor edx, edx
condicao2e:

```

```

mov edx, [esi + 4 * 6]
mov edi, offset o
cmp edx, edi
jne condicao3
mov edi, [esi + 4 * 7]
cmp edx, edi
jne condicao3
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriao

```

```

condicao3:
xor edi, edi
xor edx, edx
condicao3a:

```

```

mov edx, [esi + 4 * 0]
mov edi, offset x
cmp edx, edi
jne condicao3b
mov edi, [esi + 4 * 3]
cmp edx, edi
jne condicao3b
mov edx, [esi + 4 * 6]
cmp edx, edi
je vitoriax

```

```

condicao3b:
xor edi, edi
xor edx, edx

```



```
condicao3c:

mov edx, [esi + 4 * 1]
mov edi, offset x
cmp edx, edi
jne condicao3d
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao3d
mov edx, [esi + 4 * 7]
cmp edx, edi
je vitoriax
```

```
condicao3d:
xor edi, edi
xor edx, edx
condicao3e:
```

```
mov edx, [esi + 4 * 2]
mov edi, offset x
cmp edx, edi
jne condicao4
mov edi, [esi + 4 * 5]
cmp edx, edi
jne condicao4
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriax
```

```
condicao4:
xor edi, edi
xor edx, edx
condicao4a:
```

```
mov edx, [esi + 4 * 0]
mov edi, offset o
cmp edx, edi
jne condicao4b
mov edi, [esi + 4 * 3]
cmp edx, edi
jne condicao4b
mov edx, [esi + 4 * 6]
cmp edx, edi
je vitoriao
```

```
condicao4b:
xor edi, edi
```

```

xor edx, edx
condicao4c:

mov edx, [esi + 4 * 1]
mov edi, offset o
cmp edx, edi
jne condicao4d
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao4d
mov edx, [esi + 4 * 7]
cmp edx, edi
je vitoriao

```

```

condicao4d:
xor edi, edi
xor edx, edx
condicao4e:

```

```

mov edx, [esi + 4 * 2]
mov edi, offset o
cmp edx, edi
jne condicao5
mov edi, [esi + 4 * 5]
cmp edx, edi
jne condicao5
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriao

```

```

condicao5:
xor edi, edi
xor edx, edx
condicao5a:

```

```

mov edx, [esi + 4 * 0]
mov edi, offset x
cmp edx, edi
jne condicao5b
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao5b
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriax

```

```

condicao5b:

```

```

xor edi, edi
xor edx, edx
condicao5c:

mov edx, [esi + 4 * 2]
mov edi, offset x
cmp edx, edi
jne condicao6
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao6
mov edx, [esi + 4 * 6]
cmp edx, edi
je vitoriax

condicao6:
xor edi, edi
xor edx, edx
condicao6a:

mov edx, [esi + 4 * 0]
mov edi, offset o
cmp edx, edi
jne condicao6b
mov edi, [esi + 4 * 4]
cmp edx, edi
jne condicao6b
mov edx, [esi + 4 * 8]
cmp edx, edi
je vitoriao

condicao6b:
xor edi, edi
xor edx, edx
condicao6c:

mov edx, [esi + 4 * 2]
mov edi, offset o
cmp edx, edi
jne continuar
mov edi, [esi + 4 * 4]
cmp edx, edi
jne continuar
mov edx, [esi + 4 * 6]
cmp edx, edi
je vitoriao

```

;Retorna ao jogo se não ocorreram 9 jogadas e, senão, dá o empate  
continuar:

```
inc ebx
cmp ebx, 9
jne jogo
cmp ebx, 9
jmp empate
```

;Printa na tela o empate e encerra o jogo  
empate:  
mov edx, offset printaempate  
call writestring  
jmp saida

;Printa que o jogador 1 saiu vitorioso  
vitoriax:  
mov edx, offset printavitoriax  
call writestring  
jmp saida

;Printa que o jogador 2 saiu vitorioso  
vitoriao:  
mov edx, offset printavitoriao  
call writestring  
jmp saida

;Printa que um valor de entrada inválido foi inserido  
invalid:  
mov edx, offset invalido  
call writestring  
mov edx, offset quebralinha  
call writestring  
jmp jogo

;Printa que a posição já foi preenchida antes  
utilizado:  
mov edx, offset posicao  
call writestring  
mov edx, offset quebralinha  
call writestring  
jmp jogo

;Encerra o jogo  
saida:  
call readint  
INVOKE ExitProcess, 0

main ENDP

END main