Analise do código

1. Código em C

```c
#include <stdio.h>
#include <string.h>

typedef struct tabuleiro {
        char tab[100][100];
        int lins;
        int cols;
} tabuleiro;
typedef char bool;

bool e_seg(char c) {
        return c != '~' && c != '.';
}

int contar_segs(tabuleiro t, bool lin, int num) {
        int i;
        int x = 0, y = 0, dx = 0, dy = 0, tam = 0;
        int count = 0;

        if(lin) {
                y = num - 1;
                dx = 1;
                tam = t.lins;
        } else {
                x = num - 1;
                dy = 1;
                tam = t.cols;
        }

        for(i = 0; i < tam; i++) {
                if(e_seg(t.tab[y][x]))
                        count++;
                x += dx;
                y += dy;
        }

        return count;
}

int main() {
        tabuleiro tab;
        tab.lins = 10;
        tab.cols = 10;
        strcpy(tab.tab[0], "~~~~~~~~~~");
        strcpy(tab.tab[1], "~~~~^~~~~~");
        strcpy(tab.tab[2], "~~0~#~~<#>");
        strcpy(tab.tab[3], "~~~~#~~~~~");
        strcpy(tab.tab[4], "~~~~v~~~~^");
        strcpy(tab.tab[5], "~~~~~~~~~v");
        strcpy(tab.tab[6], "~~~<>~~~~~");
        strcpy(tab.tab[7], "<>~~~~~0~0");
        strcpy(tab.tab[8], "~~~~~0~~~~");
        strcpy(tab.tab[9], "~<#>~~~~~~");

        int i;
        for(i = 1; i <= 10; i++)
                printf(" %d", contar_segs(tab, 1, i));
        putchar('\n');
        for(i = 1; i <= 10; i++)
                printf(" %d", contar_segs(tab, 0, i));
        putchar('\n');
        return 0;
}
```

2. Código em Assembly

```
0x0804841f <contar_segs+0>: push   %ebp
0x08048420 <contar_segs+1>: mov    %esp,%ebp
0x08048422 <contar_segs+3>: sub    $0x28,%esp
0x08048425 <contar_segs+6>: mov    0x2720(%ebp),%eax
0x0804842b <contar_segs+12>:    mov    %al,-0x1(%ebp)
0x0804842e <contar_segs+15>:    movl   $0x0,-0xc(%ebp)
0x08048435 <contar_segs+22>:    movl   $0x0,-0x10(%ebp)
0x0804843c <contar_segs+29>:    movl   $0x0,-0x14(%ebp)
0x08048443 <contar_segs+36>:    movl   $0x0,-0x18(%ebp)
0x0804844a <contar_segs+43>:    movl   $0x0,-0x1c(%ebp)
0x08048451 <contar_segs+50>:    movl   $0x0,-0x20(%ebp)
0x08048458 <contar_segs+57>:    cmpb   $0x0,-0x1(%ebp)
0x0804845c <contar_segs+61>:    je     0x804847a <contar_segs+91>
0x0804845e <contar_segs+63>:    mov    0x2724(%ebp),%eax
0x08048464 <contar_segs+69>:    dec    %eax
0x08048465 <contar_segs+70>:    mov    %eax,-0x10(%ebp)
0x08048468 <contar_segs+73>:    movl   $0x1,-0x14(%ebp)
0x0804846f <contar_segs+80>:    mov    0x2718(%ebp),%eax
0x08048475 <contar_segs+86>:    mov    %eax,-0x1c(%ebp)
0x08048478 <contar_segs+89>:    jmp    0x8048494 <contar_segs+117>
0x0804847a <contar_segs+91>:    mov    0x2724(%ebp),%eax
0x08048480 <contar_segs+97>:    dec    %eax
0x08048481 <contar_segs+98>:    mov    %eax,-0xc(%ebp)
0x08048484 <contar_segs+101>:   movl   $0x1,-0x18(%ebp)
0x0804848b <contar_segs+108>:   mov    0x271c(%ebp),%eax
0x08048491 <contar_segs+114>:   mov    %eax,-0x1c(%ebp)
0x08048494 <contar_segs+117>:   movl   $0x0,-0x8(%ebp)
0x0804849b <contar_segs+124>:   mov    -0x8(%ebp),%eax
0x0804849e <contar_segs+127>:   cmp    -0x1c(%ebp),%eax
0x080484a1 <contar_segs+130>:   jl     0x80484a5 <contar_segs+134>
0x080484a3 <contar_segs+132>:   jmp    0x80484f2 <contar_segs+211>
0x080484a5 <contar_segs+134>:   sub    $0xc,%esp
0x080484a8 <contar_segs+137>:   mov    -0x10(%ebp),%edx
0x080484ab <contar_segs+140>:   mov    %edx,%eax
0x080484ad <contar_segs+142>:   shl    $0x2,%eax
0x080484b0 <contar_segs+145>:   add    %edx,%eax
0x080484b2 <contar_segs+147>:   lea    0x0(,%eax,4),%edx
0x080484b9 <contar_segs+154>:   add    %edx,%eax
0x080484bb <contar_segs+156>:   shl    $0x2,%eax
0x080484be <contar_segs+159>:   lea    0x8(%ebp),%edx
0x080484c1 <contar_segs+162>:   add    %edx,%eax
0x080484c3 <contar_segs+164>:   add    -0xc(%ebp),%eax
```

3. Tabela de alocação inicial de registos

Variaveis	Constantes	Argumentos	Registos	Memoria
i	$0x0	t		
x	$0x0	lin		-0x(%ebp)
y	$0x0	num		-0x10(%ebp)
dx	$0x0			-0x14(%ebp)
dy	$0x0			-0x18(%ebp)
tam	$0x0			-0x1c(%ebp)
count	$0x0			-0x20(%ebp)

- É criada uma nova frame
- Faz-se o push de %ebp
- 8 bytes em baixo guarda-se o endereço de retorno
- 0x2720(%ebp),%eax (ou seja a frame vai ocupar 1016 bytes e nesse ponto sera o regresso para main
- 12 bytes para baixo do base pointer sao guardados os argumentos e incializadas as variáveis
- Primeira instrução do programa em c dessa funçãoo é comparar se a lin dá valor lógico 0 ou diferente de 0 ($0x0,-0x1(%ebp))
- De seguida se for maior salta para <conta_segs+91>
- Se nao
- Guarda em %eax o que está na memoria 2724(0x2724(%ebp),%eax) que vai ser o argumento num)
- Decrementa %eax (num-1)
- Guarda o que está em %eax na memoria -0x10(%ebp)
- Guarda constante $0x1 em memoria -0x14(%ebp) (dx=1)
- Guardo o que está em memoria 0x2718(%ebp) em %eax
- Guarda o que está em %eax na memoria 0x1c(%ebp) (tam = t.cols)
- Salta para contar_segs+117>
- Guarda o que está em %eax para memoria -0x1c(%ebp)
- Guarda constant $0x0 em -0x8(%ebp) (I = 0)
- Guarda o que está em memoria -0x8(%ebp) no registo %eax
- Compara o que está na memoria -0x1c(%ebp) com %eax (i < tam)
- Se for menor salta para <contar_segs+134>
- Se não salta para <contar_segs+211>
- Subtrai constant $0xc à %esp e guarda o resultado em %esp
- Guarda o que está em memoria -0x10(%ebp) em %edx
- Troca 2 bits para esquerda o conteudo de %edx
- Soma o que está em %edx com %eax e guarda em %eax
- Guard o que está em memoria 0x0(,%eax,4) em registo %edx
- Soma o que está em %edx com %eax e guarda em %eax
- Troca dois bits para esquerda em %eax
- Guarda o que está em memoria 0x8(%ebp) em %edx
- Soma o que está em registo %edx com %eax e guarda em %eax

- Soma o que está em memoria -0xc(%ebp) com %eax e guarda em %eax
- Guarda o que está em memoria (%eax) para registo %eax
- Faz push de %eax
- Chama a função <e_seg>
- Soma constante $0x10 com %esp e guarda em %esp
- Testa valor logico
- Se é maior salta para <contar_segs+188>
- Se nao guarda o que está em -0x20(%ebp) em registo %eax
- Incremento o que está em memoria (%eax) (+1)
- Guarda o que está em -0x14(%ebp) para registo %edx
- Guarda o que está em em memoria -0xc(%ebp) para resgisto %eax)
- Soma o que está em %edx com o que está na memoria (%eax) e guarda na memoria (%eax)
- Guarda o que está em memoria -0x18(%ebp) em %edx
- Guarda o que está em memoria -0x10(%ebp) em %eax
- Incrementa o valor de (%eax)
- Salta para <contar_segs+124>
- Guarda o que está em memoria -0x20(%ebp) para registo %eax
- Sai da função

A variável tab ocupa 10000 bytes de memoria porque cada char ocupa um byte e uma matriz 100x100 = 10000