Relatório - Laboratórios de Informática II - 2014/2015

Luís Fernandes A74748, Mário Silva A75654, Ricardo Certo A75315

1 de Junho de 2015

1 Implementação da função R

1.1 Função R

A função *comandoR* recebe um tabuleiro e resolve-o, usando para isso as estratégias E1, E2 e E3. Recebe como argumentos um tabuleiro e um apontador para a stack e usa também as variáveis i, j para fazer referência às linhas e às colunas respetivamente, usa a variável flag apenas para saber quando tem de sair fora do while e usa a variável count para contar o número de casas da matriz onde não aparece nem *'o'* nem *'.'* no tabuleiro passado como argumento. Enquanto a variável flag é diferente de -1, vai resolvendo o tabuleiro aplicando as estratégias definidas na etapa anterior, sequencialmente. Com o uso de strchr pretendemos ver em que parte do tabuleiro não existem *'o'* ou *'.'*, e a função devolve-nos um apontador para essa posição, que caso seja *NULL* (ou seja, estes caracteres não se encontrem nessa posição) incrementa uma unidade à variável count. Se a nossa variável count for diferente de e.linhas * e.colunas significa que ainda existem os caracteres em cima mencionados no tabuleiro e por isso a flag permanece igual a 0 e não sai do ciclo while, caso contrário é porque já não existem os caracteres no tabuleiro e por isso o nosso tabuleiro está resolvido e a função devolve-nos então o tabuleiro resolvido.

2 Análise do código em Assembly

```
0x00000020 <+0>: push   %edi                    = salvaguarda o registo
0x00000021 <+1>: push   %esi                    = salvaguarda o registo
0x00000022 <+2>: push   %ebx                    = salvaguarda o registo
```

```
0x00000023 <+3>: mov    0x2720(%esp),%esi       = reserva espaço para os
                                                  argumentos recebidos
                                                  pela função
0x0000002a <+10>: cmpb   $0x0,0x2728(%esp)     = compara o que está na
                                                  posição 0x2728(%esp) com 0
0x00000032 <+18>: mov    0x2724(%esp),%edx     = atribui um valor a dy
0x00000039 <+25>: mov    0x272c(%esp),%eax     = atribui o valor 0 a num
0x00000040 <+32>: je     0x90 <contar_segs+112>= se o for igual a 0 salta
                                                  para 0x90 , ou seja vai
                                                  para o else
0x00000042 <+34>: lea    -0x1(%eax),%ecx       = faz a operação y=num-1;
                                                  atualiza o valor de y
0x00000045 <+37>: xor    %edi,%edi             = inicializa %edi com
                                                  o valor 0
0x00000047 <+39>: mov    $0x1,%edx             = atribui o valor 1 a dy
0x0000004c <+44>: xor    %eax,%eax             = atribui o valor 0 a num
0x0000004e <+46>: test   %esi,%esi             = faz um teste à variável
0x00000050 <+48>: jle    0xa0 <contar_segs+128>= se o resultado do teste
                                                  for menor ou igual a 0
                                                  salta para 0xa0
0x00000052 <+50>: imul   $0x64,%ecx,%ecx       = multiplica y=y*y*64 ,
                                                  ou seja atribui um novo
                                                  valor a y
0x00000055 <+53>: imul   $0x64,%edi,%edi       = multiplica x=*x*64 ,
                                                  ou seja atribui um novo
                                                  valor a x
0x00000058 <+56>: add    %eax,%ecx             = atualiza o valor de y ,
                                                  fazendo a operação y+=x
0x0000005a <+58>: lea    0x10(%esp),%eax       = atualiza a variável num ,
                                                  com o que está guardado
                                                  em 0x10(%ebp)
0x0000005e <+62>: add    %edx,%edi             = atualiza o valor de x ,
                                                  fazendo a operação x+=dy
0x00000060 <+64>: add    %eax,%ecx             = atualiza o valor de y ,
                                                  fazendo a operação y+=num
0x00000062 <+66>: xor    %edx,%edx             = atualiza o valor de
                                                  dy para 0
0x00000064 <+68>: xor    %eax,%eax             = atualiza o valor
                                                  de num para 0
0x00000066 <+70>: xchg   %ax,%ax               = troca o valor dos
                                                  operadores
```

```
0x00000068 <+72>: movzbl (%ecx),%ebx           = move o valor que está na
                                                  pos de memória de
                                                  %ecx (y) para %ebx
0x0000006b <+75>: cmp    $0x2e,%bl             = compara o valor $0x2e
                                                  com o valor e bl
0x0000006e <+78>: je     0x7c <contar_segs+92> = se for igual a 0 salta
                                                  para 0x7c
0x00000070 <+80>: cmp    $0x7e,%bl             = compara o valor $0x7e com
                                                  o valor e bl
0x00000073 <+83>: setne  %bl                   = ve se bl é diferente de 0
0x00000076 <+86>: cmp    $0x1,%bl              = compara se 1 é igual a
0x00000079 <+89>: sbb    $0xffffffff,%eax      =
0x0000007c <+92>: add    $0x1,%edx             = adiciona 1 a dy
0x0000007f <+95>: add    %edi,%ecx             = soma y+=x , ou seja
                                                  atualiza o valor de y
0x00000081 <+97>: cmp    %esi,%edx             = compara o valor de dx com
0x00000083 <+99>: jne    0x68 <contar_segs+72> = salta para 0x68 se não
                                                  for 0
0x00000085 <+101>: pop    %ebx                  = recupera o registo %ebx
0x00000086 <+102>: pop    %esi                  = recupera o registo %esi
0x00000087 <+103>: pop    %edi                  = recupera o registo %edi
0x00000088 <+104>: ret                          = recupera o SP, o FP e
                                                  o IP e regressa à main
0x00000089 <+105>: lea    0x0(%esi,%eiz,1),%esi = resultado de chamar a
                                                  função e_segs
0x00000090 <+112>: mov    %edx,%esi             = move o que esta em dy para
0x00000092 <+114>: sub    $0x1,%eax             = faz a operação num-=1
0x00000095 <+117>: mov    $0x1,%edi             = atualiza o valor de x
                                                  para 1
0x0000009a <+122>: xor    %edx,%edx             = atualiza o valor de dy
                                                  para 0
0x0000009c <+124>: xor    %ecx,%ecx             = atualiza o valor de y
                                                  para 0
0x0000009e <+126>: jmp    0x4e <contar_segs+46> = salta incondicionalmente
                                                  para 0x4e
0x000000a0 <+128>: xor    %eax,%eax             = atualiza o valor de num
                                                  para 0
0x000000a2 <+130>: jmp    0x85 <contar_segs+101>= salta incondicionalmente
                                                  para 0x85
```