Programação em IA32
(Semana-12)

Exercícios adaptados do livro CSPP
Randal E. Bryant e David R. O'Hallaron

Os exercícios deverão ser resolvidos em grupos de 2/3 pessoas e a resolução condensada numa única folha, fornecida para o efeito, a entregar no final da aula.

Exercício 1. *(Transbordo do tampão de dados)*: A função `leLinha` lê (`gets`) uma sequência de caracteres da entrada de dados que copia (`strcpy`) para uma nova área de dados na memória (`malloc`) e retorna um apontador para o respectivo endereço (*resultado*). Segue-se um fragmento de código executável IA32.

```
1              /* Código de muito baixa qualidade usado para ilustrar
2                 práticas incorrectas de programação . */
3 char *leLinha()
4 {
5     char buf[8];
6     char *result;
7     gets(buf);
8     result = malloc(strlen(buf));
9     strcpy(result, buf);
10    return(result);
11 }
```

```
1   08048524     <leLinha>:
2   8048524: 55              push %ebp
3   8048525: 89 e5           mov %esp,%ebp
4   8048527: 83 ec 10        sub $0x10,%esp
5   804852a: 56              push %esi
6   804852b: 53              push %ebx

        ;Faça o diagrama do registo de activação da pilha aqui

7   804852c: 83 c4 f4        add $0xfffffff4,%esp
8   804852f: 8d 5d f8        lea 0xfffffff8(%ebp),%ebx
9   8048532: 53              push %ebx

10  8048533: e8 74 fe ff ff  call 80483ac <_init+0x50> ; chama gets
11
```

Considere o seguinte cenário: a função `leLinha` é chamada com o endereço de retorno `0x8048643`, o registo `%ebp` contém o valor `0xbffffc94`, o registo `%esi` tem o valor `0x1`, o registo `%ebx` vale `0x2`, a leitura produz a sequência "**`012345678901`**" e o programa termina abruptamente com o erro *segmentation fault*.
Utilizando o *depurador* (GDB) para detectar a anomalia chegámos à conclusão que a mesma ocorre durante a execução da instrução `ret` na função `leLinha`.

a) Considerando que a pilha "cresce para baixo", preencha a área de activação da pilha (*stack frame*) com toda informação relevante, disponível após a execução da instrução da linha 6 (no código desmontado), não esquecendo de indicar as posições apontadas pelos registos %ebp e %esp. Cada posição correspondente a 4 octetos deve conter um valor em hexadecimal (se conhecido) e à direita uma etiqueta que ajude a esclarecer o tipo de informação (e.g., "Endereço de Retorno")

```
+-------------+
| 08 04 86 43 | Endereço de Retorno
+-------------+
|             |
+-------------+
|             |
+-------------+
```

b) Modifique o diagrama para mostrar os valores após a chamada da função `gets` na linha 10.
c) Relacione o erro *segmentation fault*., com o retorno, após a conclusão, da chamada à função acima?
d) Houve registos registo(s) corrompido(s) no regresso da função `leLinha` e em caso afirmativo, qual a razão?
e) Além do transbordo de tampão de dados (*buffer overflow)* que outros problemas existem na função `leLinha`?

Números de alunos: 1ª _____ 2ª _____ 3ª _____ Turno ___

Exercício 2. *(Chamada de rotinas):* O seguinte fragmento de código aparece frequentemente em bibliotecas de rotinas:

```
1          call         seguinte
2 seguinte:
3          popl         %eax
```

a) Qual a finalidade deste fragmento de código?

b) Que valor é colocado no registo %eax?

Exercício 3. (*Matrizes*): A partir do fragmento de código abaixo, resultante da compilação da função `sum_element`, descubra os valores iniciais das constantes X e Y.

```
  #define X ??
  #define Y ??
```

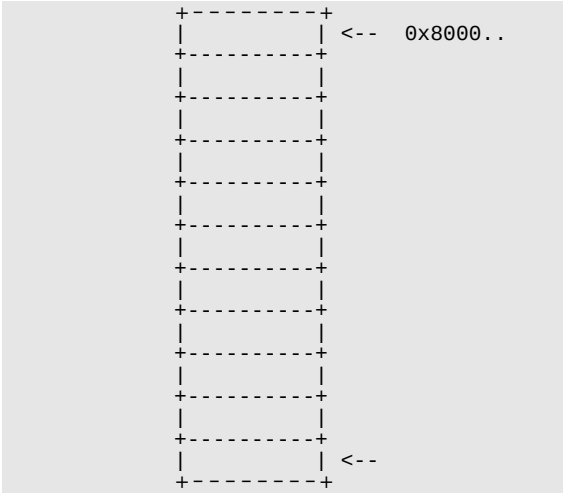
```
1 int mat1[X][Y];
2 int mat2[Y][X];
3
4 int sum_element(int i, int j)
5 {
6     return mat1[i][j] + mat2[j][i];
7 }
```

```
1     movl      8(%ebp),%ecx
2     movl      12(%ebp),%eax
3     leal      0(,%eax,4),%ebx
4     leal      0(,%ecx,8),%edx
5     subl      %ecx,%edx
6     addl      %ebx,%eax
7     sall      $2,%eax
8     movl      mat2(%eax,%ecx,4),%eax
9     addl      mat1(%ebx,%edx,4),%eax
```

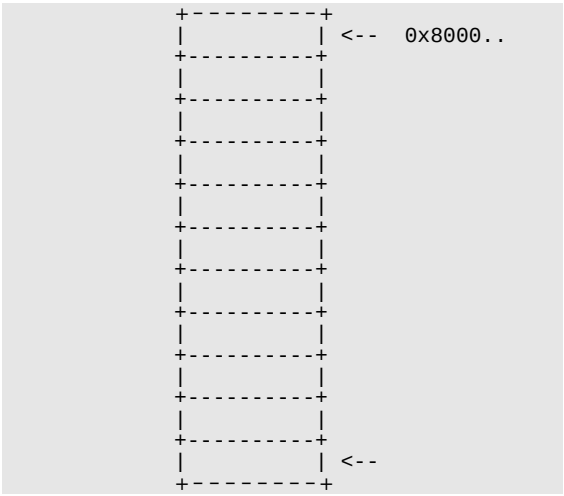
Números de alunos: 1ª _____ 2ª _____ 3ª _____ Turno ___

Exercício 1.

a)



b)



d)

e)

f)

Números de alunos: 1ª _____ 2ª _____ 3ª _____ Turno ___

Exercício 2.

a)

b)

Exercício 3.

```
int mat1[X][Y];
int mat2[Y][X];

int sum_element(int i, int j)
{
    return mat1[i][j] + mat2[j][i];
}
```

```
    movl    8(%ebp),%ecx
    movl    12(%ebp),%eax
    leal    0(,%eax,4),%ebx
    leal    0(,%ecx,8),%edx
    subl    %ecx,%edx
    addl    %ebx,%eax
    sall    $2,%eax
    movl    mat2(%eax,%ecx,4),%eax
    addl    mat1(%ebx,%edx,4),%eax
```