MAC0216 – Técnicas de Programação I

Instituto de Matemática e Estatística

Professor Daniel Macêdo Batista

Primeira Prova – 19 de outubro de 2022

Nome: _____

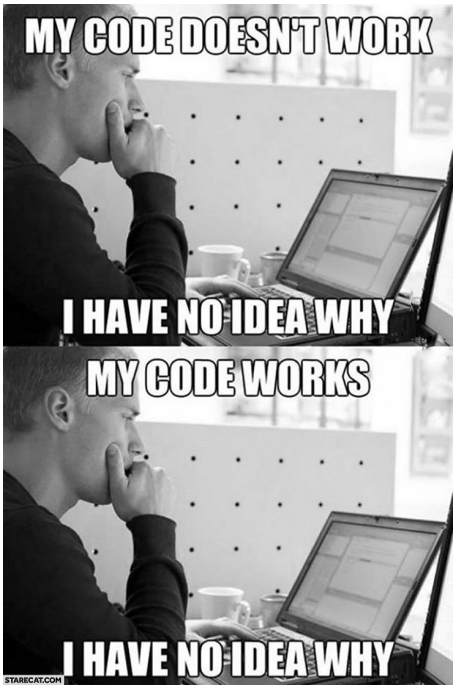
Assinatura: _____

Nº USP: _____

Instruções:

1. Não destaque as folhas deste caderno. A prova pode ser feita a lápis.
2. A prova consta de 6 questões e 18 páginas. Verifique antes de começar a prova se o seu caderno está completo.
3. No final deste caderno há um apêndice que pode ser útil na resolução das questões.
4. Não é permitido o uso de folhas avulsas para rascunho, a consulta a livros, apontamentos, colegas ou equipamentos eletrônicos. Desligue o seu celular e qualquer equipamento que possa perturbar o andamento da prova.

DURAÇÃO DA PROVA: 2 horas



Questão	Valor	Nota
1	1,0	
2	2,0	
3	2,0	
4	1,0	
5	2,0	
6	2,0	
Total	10,0	

QUESTÃO 1 (vale 1,0 ponto)

Escreva, usando `MOV`, `CMP`, `ADD`, `NEG` e saltos condicionais, instruções em Assembly equivalentes a cada uma das quatro sequências de instruções em Assembly abaixo. Considere a arquitetura x86-64 em um sistema operacional Linux.

a)
```
XOR RCX, RCX
```

b)
```
OR RDX, RDX
JNZ SALTOZ
```

c)
```
NOT RAX
```

d)
```
AND RBX, 0x0
```

QUESTÃO 2 (vale 2,0 pontos)

Considere o programa abaixo, escrito em Assembly para a arquitetura x86-64 em um sistema operacional Linux.

```
global _start

section .text

_start:
       mov rdi, 0
       mov rsi, buffer
       mov rdx, 256
       mov rax, 0
       syscall

       mov rcx, 0
loop:  mov bl, [buffer+rcx]
       cmp bl, 65
       jl  prox
       cmp bl, 90
       jle muda1
       cmp bl, 97
       jl  prox1
       cmp bl, 122
       jle muda2
       jmp prox1
muda1: add bl, 32
       mov [buffer+rcx], bl
       jmp prox2
muda2: sub bl, 32
       mov [buffer+rcx], bl
       jmp prox2

prox:  cmp bl, 10
       je  prox2
       cmp bl, 32
       je  prox2
prox1: mov byte [buffer+rcx], 46
prox2: inc rcx
       cmp rcx, rax
       jl  loop

       mov rdi, 1
       mov rsi, buffer
       mov rdx, rax
       mov rax, 1
       syscall

       mov rdi, 0
       mov rax, 60
       syscall

section .bss
buffer: resb 256
```

a) O que o programa faz?

b) Supondo que a string passada como entrada seja:

`Vou Tirar 10,0!`

Qual é a string de saída?

QUESTÃO 3 (vale 2,0 pontos)

a) Quais são as 3 responsabilidades de uma função chamadora e as 9 responsabilidades de uma função chamada em linguagem de montagem?

b) A sub-rotina abaixo, escrita em Assembly para a arquitetura x86-64 em um sistema operacional Linux, verifica a paridade de um número inteiro de 16 bits que está em AX. Modifique a sub-rotina para que ela vire uma função e identifique, com comentários, cada uma das 9 responsabilidades que você informou na letra anterior. Mesmo que alguma(s) das responsabilidades seja(m) opcional(is) para este problema, identifique com comentários onde ela(s) deveria(m) ser definida(s) no fluxo da função. Caso seja necessário modificar o local do parâmetro e/ou o local do retorno, para que a sub-rotina respeite as responsabilidades de funções, você está livre para fazer isso.

```
; PARÂMETRO: o valor a ser verificado precisa estar em ax
; RETORNO:   o retorno da função vai estar em ah:
;              . vai ser 0 se o número for par
;              . vai ser 1 se o número for ímpar
paridade: mov   bl, 2
          div   bl
          ret
```

QUESTÃO 4 (vale 1,0 ponto)

Informe, para cada afirmação a seguir, se ela é verdadeira ou falsa. No caso de afirmações falsas, informe o(s) comando(s) em bash necessário(s) para torná-la verdadeira.

a) Este comando concatena o conteúdo do arquivo `/tmp/saida-1.txt` no arquivo `saida-total.txt` localizado no home do usuário (Considere que os dois arquivos existem e o usuário que executa o comando tem permissão de lê-los e escrevê-los):

```
cat /tmp/saida-1.txt > ~/saida-total.txt
```

b) Este comando dá permissão de leitura, escrita e execução ao arquivo `/tmp/script.sh` para o dono, para o grupo e para os outros usuários (Considere que o arquivo existe e pertence ao usuário que executa o comando):

```
chmod 000 /tmp/script.sh
```

c) Esta sequência de comandos faz o conteúdo do arquivo `/tmp/saida-1.txt` ser impresso na tela e colocado no arquivo `/tmp/saida-2.txt`, sobrescrevendo este último (Considere que os arquivos existem e o usuário que executa os comandos tem permissão de lê-los e escrevê-los):

```
cat /tmp/saida-1.txt | tee /tmp/saida-2.txt
```

d) Esta sequência de comandos executa o programa `programa` localizado no diretório atual e, caso ocorra algum erro na execução do mesmo, a mensagem `Xi, deu erro` é impressa na tela (Considere que o programa de fato existe no diretório atual, respeita a padronização de códigos de saída de programas no Linux e o usuário que executa os comandos tem permissão de executá-lo):

```
./programa && echo "Xi, deu erro"
```

QUESTÃO 5 (vale 2,0 pontos)

Considere o bash script `~/script.sh` abaixo que recebe como parâmetro o nome de um arquivo contendo valores de temperaturas medidos em um computador (o arquivo passado como parâmetro sempre terá ao menos 1 valor medido e todos os valores sempre serão números reais maiores que zero. Cada linha do arquivo será uma string desta forma: `temp=TEMPERATURA'C`, onde TEMPERATURA é o valor real da temperatura medida.)

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo Uso: $0 [arquivo de temperaturas]
    exit 1
fi

T=$(wc -l $1 | cut -f 1 -d ' ')
S=0
O=(0 0)

for L in $(cat $1); do
    V=$(echo $L | cut -f 2 -d "=" | cut -f 1 -d "'")
    if [ $( echo "$V>${O[1]}" | bc -l ) -eq 1 ]; then
        O[1]=$V
    fi
    S=$(echo "$S+$V" | bc -l)
done

O[0]=$(echo "scale=1;$S/$T" | bc -l) # Obs.: scale=1 quando passado
                                     # para o bc serve para informar
                                     # que os resultados devem ser
                                     # exibidos com 1 casa decimal

echo "${O[*]}"

exit 0
```

a) O que o script faz?

b) Supondo que o programa seja executado desta forma:

`~/script.sh /tmp/saida-3.txt`

O que é impresso na tela? (Considere que o script de fato está no home do usuário, que o usuário tem permissão para executá-lo e que o conteúdo do arquivo `/tmp/saida-3.txt` é este abaixo):

```
temp=53.0'C
temp=53.0'C
temp=53.5'C
temp=55.0'C
temp=60.0'C
temp=61.5'C
temp=55.0'C
temp=53.0'C
```

QUESTÃO 6 (vale 2,0 pontos)

Escreva um bash script que recebe como parâmetro na linha de comando o nome de um arquivo em texto puro que contém uma linha para cada pessoa matriculada em uma disciplina e cada linha tem o seguinte formato:

```
NUSP;nota na P1;nota na P2
```

Por exemplo:

```
112345;10.0;9.0
123451;0.0;
134512;10.0;10.0
145123;0.0;
151234;5.5;10.0
112346;10.0;10.0
```

Caso alguém não tenha feito uma das provas, o campo específico não terá nenhum caractere. Os números USP são representados por números inteiros de 6 dígitos. As notas são números reais com 3 ou 4 caracteres e o separador de casa decimal é o ponto. Considere que cada prova teve pelo menos uma pessoa que a fez.

O bash script deve imprimir na tela as seguintes informações:

- Média da turma para cada prova;
- Porcentagem de participação para cada prova.

Segue abaixo a saída esperada do script para o arquivo de exemplo exibido acima.

```
Média da P1: 5.91666666666666666666
Média da P2: 9.75000000000000000000
Porcentagem de presença na P1: 100.00000000000000000000%
Porcentagem de presença na P2: 66.66666666666666666600%
```

APÊNDICE

```
$ head -n 64 /usr/include/x86_64-linux-gnu/asm/unistd_64.h

#ifndef _ASM_X86_UNISTD_64_H
#define _ASM_X86_UNISTD_64_H 1

#define __NR_read 0
#define __NR_write 1
#define __NR_open 2
#define __NR_close 3
#define __NR_stat 4
#define __NR_fstat 5
#define __NR_lstat 6
#define __NR_poll 7
#define __NR_lseek 8
#define __NR_mmap 9
#define __NR_mprotect 10
#define __NR_munmap 11
#define __NR_brk 12
#define __NR_rt_sigaction 13
#define __NR_rt_sigprocmask 14
#define __NR_rt_sigreturn 15
#define __NR_ioctl 16
#define __NR_pread64 17
#define __NR_pwrite64 18
#define __NR_readv 19
#define __NR_writev 20
#define __NR_access 21
#define __NR_pipe 22
#define __NR_select 23
#define __NR_sched_yield 24
#define __NR_mremap 25
#define __NR_msync 26
#define __NR_mincore 27
#define __NR_madvise 28
#define __NR_shmget 29
#define __NR_shmat 30
#define __NR_shmctl 31
#define __NR_dup 32
#define __NR_dup2 33
#define __NR_pause 34
#define __NR_nanosleep 35
#define __NR_getitimer 36
#define __NR_alarm 37
#define __NR_setitimer 38
#define __NR_getpid 39
#define __NR_sendfile 40
#define __NR_socket 41
#define __NR_connect 42
#define __NR_accept 43
```

```
#define __NR_sendto 44
#define __NR_recvfrom 45
#define __NR_sendmsg 46
#define __NR_recvmsg 47
#define __NR_shutdown 48
#define __NR_bind 49
#define __NR_listen 50
#define __NR_getsockname 51
#define __NR_getpeername 52
#define __NR_socketpair 53
#define __NR_setsockopt 54
#define __NR_getsockopt 55
#define __NR_clone 56
#define __NR_fork 57
#define __NR_vfork 58
#define __NR_execve 59
#define __NR_exit 60

$ man syscall # Apenas as tabelas que interessam

Arch/ABI    Instruction           System  Ret  Ret  Error    Notes
                                  call #  val  val2
-----------------------------------------------------------------
alpha       callsys               v0      v0   a4   a3       1, 6
arc         trap0                 r8      r0   -    -
arm/OABI    swi NR                -       r0   -    -        2
arm/EABI    swi 0x0               r7      r0   r1   -
arm64       svc #0                w8      x0   x1   -
blackfin    excpt 0x0             P0      R0   -    -
i386        int $0x80             eax     eax  edx  -
ia64        break 0x100000        r15     r8   r9   r10      1, 6
m68k        trap #0               d0      d0   -    -
microblaze  brki r14,8            r12     r3   -    -
mips        syscall               v0      v0   v1   a3       1, 6
nios2       trap                  r2      r2   -    r7
parisc      ble 0x100(%sr2, %r0)  r20     r28  -    -
powerpc     sc                    r0      r3   -    r0       1
powerpc64   sc                    r0      r3   -    cr0.SO   1
riscv       ecall                 a7      a0   a1   -
s390        svc 0                 r1      r2   r3   -        3
s390x       svc 0                 r1      r2   r3   -        3
superh      trap #0x17            r3      r0   r1   -        4, 6
sparc/32    t 0x10                g1      o0   o1   psr/csr  1, 6
sparc/64    t 0x6d                g1      o0   o1   psr/csr  1, 6
tile        swint1                R10     R00  -    R01      1
x86-64      syscall               rax     rax  rdx  -        5
x32         syscall               rax     rax  rdx  -        5
xtensa      syscall               a2      a2   -    -
```

```
Arch/ABI      arg1  arg2  arg3  arg4  arg5  arg6  arg7  Notes
----------------------------------------------------------------
alpha         a0    a1    a2    a3    a4    a5    -
arc           r0    r1    r2    r3    r4    r5    -
arm/OABI      r0    r1    r2    r3    r4    r5    r6
arm/EABI      r0    r1    r2    r3    r4    r5    r6
arm64         x0    x1    x2    x3    x4    x5    -
blackfin      R0    R1    R2    R3    R4    R5    -
i386          ebx   ecx   edx   esi   edi   ebp   -
ia64          out0  out1  out2  out3  out4  out5  -
m68k          d1    d2    d3    d4    d5    a0    -
microblaze    r5    r6    r7    r8    r9    r10   -
mips/o32      a0    a1    a2    a3    -     -     -     1
mips/n32,64   a0    a1    a2    a3    a4    a5    -
nios2         r4    r5    r6    r7    r8    r9    -
parisc        r26   r25   r24   r23   r22   r21   -
powerpc       r3    r4    r5    r6    r7    r8    r9
powerpc64     r3    r4    r5    r6    r7    r8    -
riscv         a0    a1    a2    a3    a4    a5    -
s390          r2    r3    r4    r5    r6    r7    -
s390x         r2    r3    r4    r5    r6    r7    -
superh        r4    r5    r6    r7    r0    r1    r2
sparc/32      o0    o1    o2    o3    o4    o5    -
sparc/64      o0    o1    o2    o3    o4    o5    -
tile          R00   R01   R02   R03   R04   R05   -
x86-64        rdi   rsi   rdx   r10   r8    r9    -
x32           rdi   rsi   rdx   r10   r8    r9    -
xtensa        a6    a3    a4    a5    a8    a9    -

$ man 2 exit # Apenas o protótipo

#include <unistd.h>

void _exit(int status);

$ man 2 read # Apenas o protótipo

#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);

$ man 2 write # Apenas o protótipo

#include <unistd.h>

ssize_t write(int fd, const void *buf, size_t count);
```

```
$ man 2 open # Apenas o protótipo

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);

$ man ascii # Apenas a tabela que interessa

The decimal set:

  0 nul    1 soh    2 stx    3 etx    4 eot    5 enq    6 ack    7 bel
  8 bs     9 ht    10 nl    11 vt    12 np    13 cr    14 so    15 si
 16 dle   17 dc1   18 dc2   19 dc3   20 dc4   21 nak   22 syn   23 etb
 24 can   25 em    26 sub   27 esc   28 fs    29 gs    30 rs    31 us
 32 sp    33  !    34  "    35  #    36  $    37  %    38  &    39  '
 40  (    41  )    42  *    43  +    44  ,    45  -    46  .    47  /
 48  0    49  1    50  2    51  3    52  4    53  5    54  6    55  7
 56  8    57  9    58  :    59  ;    60  <    61  =    62  >    63  ?
 64  @    65  A    66  B    67  C    68  D    69  E    70  F    71  G
 72  H    73  I    74  J    75  K    76  L    77  M    78  N    79  O
 80  P    81  Q    82  R    83  S    84  T    85  U    86  V    87  W
 88  X    90  Y    90  Z    91  [    92  \    93  ]    94  ^    95  _
 96  `    97  a    98  b    99  c   100  d   101  e   102  f   103  g
104  h   105  i   106  j   107  k   108  l   109  m   110  n   111  o
112  p   113  q   114  r   115  s   116  t   117  u   118  v   119  w
120  x   121  y   122  z   123  {   124  |   125  }   126  ~   127 del

# Alguns comandos para expressões lógicas em bash:

Expressão      | Verdadeira se
---------------------------------------------------------------------
-e ARQ         | o arquivo ARQ existe
-d DIR         | o diretório DIR existe
-z STR         | o comprimento da string STR é zero
-n STR         | o comprimento da string STR não é zero
STR1 == STR2   | as strings STR1 e STR2 são iguais
STR1 != STR2   | as strings STR1 e STR2 são diferentes
STR1 < STR2    | a string STR1 é lexicograficamente menor que STR2
NUM1 -eq NUM2  | os inteiros NUM1 e NUM2 são iguais
NUM1 -ne NUM2  | os inteiros NUM1 e NUM2 são diferentes
NUM1 -lt NUM2  | o inteiro NUM1 é menor que o inteiro NUM2
NUM1 -le NUM2  | o inteiro NUM1 é menor ou igual ao inteiro NUM2
NUM1 -gt NUM2  | o inteiro NUM1 é maior que o inteiro NUM2
NUM1 -ge NUM2  | o inteiro NUM1 é maior ou igual ao inteiro NUM2
! EXPR         | EXPR é uma expressão falsa
EXPR1 -a EXPR2 | ambas as expressões EXPR1 e EXPR2 são verdadeiras
EXPR1 -o EXPR2 | ao menos uma das expressões EXPR1 ou EXPR2 é verdadeira
```

```
# Sequência de alguns comandos em bash com um arquivo de exemplo /tmp/saida.txt

$ cat /tmp/saida.txt
aluno1;4;5;6,0
aluno2;1;2;4,5
professor1;1;2;5,6
professor2;2;3;5.0
professor3;;4;5.1
professor4;5;6;1.2
$ grep aluno /tmp/saida.txt
aluno1;4;5;6,0
aluno2;1;2;4,5
$ grep professor2 /tmp/saida.txt
professor2;2;3;5.0
$ cut -f 2 -d ';' /tmp/saida.txt
4
1
1
2

5
$ wc -l /tmp/saida.txt
6 /tmp/saida.txt
$ grep professor2 /tmp/saida.txt | tr ';' '|'
professor2|2|3|5.0
```