## Arquitectura de Computadores



Licenciatura em Eng.ª Informática

Frequência –

15 de Abril de 2015 Duração: 75 min.

Nome:	Número:

## **Notas Importantes:**

A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior. Não serão admitidas quaisquer tentativas de fraude, levando qualquer tentativa detectada à reprovação imediata, tanto do facilitador como do prevaricador.

Durante a prova pode consultar a bibliografia da disciplina (slides, livros, enunciados e material de apoio aos trabalhos práticos). No entanto, não é permitido o uso de computadores/máquinas de calcular ou a consulta de exercícios previamente resolvidos.

## Respostas: indicar resposta V (Verdadeiro) ou F (Falso), para cada alínea da questão Nota: cada pergunta pode ter mais do que uma resposta Verdadeira; perguntas correctas valem 2 pontos; perguntas erradas descontam 1 ponto; notas inferiores a zero valem zero.

	1	2	3	4	5	6	7	8	9	10
a.	F	V	F	F	F	F	V	V	F	V
b.	F	F	V	V	V	V	F	V	V	F
c.	F	V	F	V	F	F	F	F	V	V
d.	V	V	V	F	V	F	V	F	F	V
e.	F	F	F	V	F	V	F	F	F	F

- 1. Nas aulas práticas utilizou a estrutura vector\_t declarada no código anexo, em que a função main chama a função transforma. O que é impresso no ecrã?
  - a. O ecrã imprime Value: 0
  - **b.** O ecrã imprime Value: 1
  - c. Não é possível prever o que vai ser impresso no ecrã porque vec2x1 está declarada na memória dinâmica (heap)
  - d. Não é possível prever o que vai ser impresso no ecrã porque vec2x1 está declarada na pilha (stack) \*
  - e. A função transforma () causa uma fuga de memória porque não liberta adequadamente a memória.

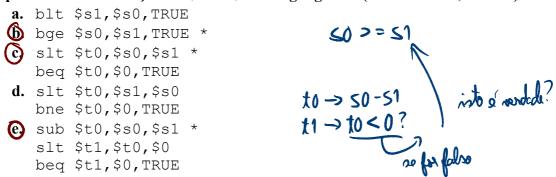
```
#include <stdio.h>
#include <stdlib.h>
struct vector t{
      size t size;
      int *data;
};
void transforma(struct vector t *ptr){
int vec2x1[2] = {0,0};
free(ptr->data);
ptr->data=vec2x1;
int main(){
struct vector_t *vec;
vec=(struct
                vector t
                            *) malloc(sizeof(struct
vector_t));
vec->size=2;
 vec->data=(int *)malloc(vec->size*sizeof(int));
\text{vec->data}[0]=1:
 vec->data[1]=2;
transforma(vec);
printf("Value: %d \n", *((vec->data)+1) );
```

- 2. Relativamente à seguinte função, com protótipo void func(int v[], int k), é correcto afirmar que (assuma que a chamada à função é feita com k>0):
  - a. Troca dois elementos vizinhos do array v [] \*
  - **b.** Incrementa o elemento k do array v []
  - **c.** Não é possível efectuar a troca de duas variáveis sem recorrer a uma variável temporária em linguagem C \*
  - d. Não é possível efectuar a troca de duas variáveis numa única operação atómica \*
  - e. Coloca mais um elemento (a seguir ao último) no array v[]

3. Considere um datapath com cinco etapas: 1 – instruction fetch; 2 – instruction decode; 3 – ALU; 4 – memory access; 5 – register write. Indique para cada uma das seguintes alíneas se a afirmação "a intrução está activa em todas as etapas do datapath" é verdadeira ou falsa.

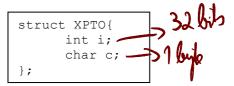
```
X beq $t0,$t1,-6 mooks $\forall $\forall $\b$. lw $t1,12($t0) *
X slti $t2,$t1,1 mooks $\forall $\for
```

4. Indique para cada uma das alíneas se é verdadeiro ou falso que as instruções envolvidas implementam a instrução if(a>=b) em linguagem C (Nota: \$s0=a, \$s1=b).



- 5. Qual o propósito do typecast realizado sobre a função malloc?
  - a. Verificar se os blocos de memória a alocar pela função malloc estão livres.
  - **b.** Indicar ao compilador o tipo de dados que irão ser armazenados nos blocos de memória alocados pela função malloc.\*
  - c. Indicar ao compilador o tamanho da memória alocada pela função malloc.
  - d. Transformar um ponteiro do tipo void genérico num ponteiro do tipo desejado. \*
  - e. Libertar a memória alocada por malloc após esta não mais ser necessária.

6. Considere a seguinte struct em linguagem C:



O valor ocupado por x em memória, após a atribuição x=malloc(sizeof(struct XPTO)), corresponde a:

- **a.** 64 bits
- **b.** 5 bytes \*
- c. 5 bits
- **d.** 8 bytes
- **e.** 40 bits \*
- 7. Considere a instrução addi \$4,\$5,1 e um datapath com cinco etapas: 1 instruction fetch; 2 instruction decode; 3 ALU; 4 memory access; 5 register write. Indique se cada uma das afirmações seguintes é verdadeira ou falsa.
  - A instrução está activa nas 3 primeiras etapas do datapath. \*
  - **b.** A instrução está inactiva apenas na última etapa do datapah de escrita nos registos.
  - c. A instrução está activa em todas as etapas do datapath.
  - d A instrução está inactiva apenas na etapa do datapah de acesso à memória. \*
  - e. A instrução está activa apenas na etapa da ALU.
- 8. Relativamente a uma variável declarada como "global" estática, indique quais as afirmações verdadeiras e falsas.
  - **a.** A variável é armazenada num espaço de memória que permanece alocado durante todo o "runtime" do programa. \*
  - **b.** A variável é armazenada num espaço de memória acessível a partir de qualquer zona do programa. \*
  - c. Se usar uma variável local com um nome idêntico, a variável global tem precedência.
  - **d.** A variável não é guardada na zona "estática" de memória.
  - e. Os dados são válidos até ao instante em que o programador faz a sua desalocação manual.
- 9. Relativamente ao espaço de armazenamento designado por pilha ("stack"), diga quais afirmações são verdadeiras ou falsas.
  - a. Quando uma determinada rotina termina, a "stack frame" relativa a essa rotina é removida e todos os dados lá armazenados são literalmente apagados. 🛆
  - **b.** Quando uma determinada rotina termina, a "stack frame" relativa a essa rotina é simplesmente descartada (não explicitamente apagada). \* V
  - **c.** Quando uma determinada rotina termina, o "stack pointer" é incrementado de modo a voltar para o ponto na pilha onde se encontrava antes da chamada da rotina. \* **V**
  - d. Quando uma determinada rotina termina, os dados contidos na "stack frame" criada são copiados para a zona de memória dinâmica ("heap") e só depois a "stack frame" é descartada.
  - **e.** Quando uma determinada rotina num programa termina, a sua "stack frame" permanece na pilha até que o programa chegue ao fim.

10. Relativamente ao seguinte código em linguagem C em que é A [n] é um array de inteiros de tamanho n, diga se cada uma das seguintes opções é verdadeira ou falsa.

```
int i=0;
int g=0;

do {
    g = g + A[i];
    i = i + 1;
}while (i != n);
```

- a. Este código permite somar/acumular todos os elementos do array. \*
- **b.** Este código permite somar/acumular os elementos do array, com excepção do último elemento.
- c. O ciclo do-while é equivalente ao seguinte ciclo while: \*

d. O ciclo do-while é equivalente ao seguinte ciclo for: \*

for(
$$i=n-1; i>=0; i--$$
) {  
 g = g + A[i];};

e. O ciclo do-while é equivalente ao seguinte ciclo while:

```
while (i \leq n) \{

g = g + A[i];

i = i + 1; \};
```