Primeira Prova de EDA1 - Engenharia de Software - 29 de novembro de 2022

Nome do aluno:: _____

1 (60 pontos) Escolha V para verdadeiro e F para falso em cada afirmação:

- (V) A quantidade de chamadas recursivas de uma função está limitada ao tamanho da *stack* (pilha) que armazena todas as variáveis locais do programa
- (V) `int x=10; x++` soma 1 no valor atribuído a `x`, ou seja, `x` passa a armazenar 11
- (F) É impossível armazenar um vetor dentro de uma `struct`
- (V) A definição de um tipo abstrato de dados, em `C`, é feito com a palavra reservada `struct`
- (V) Variáveis do tipo ponteiro, em `C`, são variáveis que armazenam um endereço de memória
- (F) Recursão e *structs* são definições antagônicas na criação de um programa
- (V) Uma função receber um valor por referência significa que a função recebe por cópia o endereço de uma posição na memória
- (F) O tipo `char` é um tipo especial para armazenar caracteres e nada mais
- (V) `int x=10;int *y=&x` faz com que `y` receba o endereço de `x`
- (F) Em `C`, `return` é uma função
- (F) A linguagem `C` não possui manipulação de ponteiros
- (V) Dadas duas variáveis do tipo `struct ALGO`, é possível atribuir uma na outra, por exemplo: `struct ALGO x,y; x=preenche_struct();y=x;`
- (F) A linguagem utilizada na disciplina EDA1 é `python`
- (F) O `scanf` é uma simples função para ler somente inteiros
- (F) Uma função *recursiva* nada mais é do que um absurdo e inexistente na computação
- (V) Send `*y` um ponteiro para um inteiro `(*p)++` soma 1 no valor armazenado no endereço que `*y` aponta
- (F) Uma função jamais poderá devolver um tipo abstrato de dados, definido como `struct`
- (V) Sendo `*y` um ponteiro para um inteiro armazenado no endereço $0x1$, `y++` faz com que `*y` aponte para o endereço $0x5$
- (F) `int x=10;int *y=&x; y++` soma 1 no valor atribuído a `x`, ou seja, `x` passa a armazenar 11
- (V) A chamada `malloc()` solicita ao Sistema Operacional um conjunto de bytes contíguos

2 (5 pontos) Seja `vet` um vetor de 4 elementos: `TIPO vet[4].`

Supor que depois da declaração, `vet` esteja armazenado no endereço de memória `4092` (ou seja, o endereço de `vet[0]`). Supor também que na máquina usada uma variável do tipo char ocupa 1 byte, do tipo int ocupa 2 bytes, do tipo float ocupa 4 bytes e do tipo double ocupa 8 bytes.

Qual o valor de `vet+1` se:

1. `vet` for declarado como `char`?
2. `vet` for declarado como `int`?
3. `vet` for declarado como `float`?
4. `vet` for declarado como `double`?

Estão **certas** as seguintes afirmativas (marque com um X):

☐ 4094, 4093, 4100, 4096
☒ 4093, 4094, 4096, 4100
☐ 4092, 4092, 4092, 4092
☐ 4093, 4094, 4096, 5000

3 (5 pontos) Considere as declarações:

```
int vetor[10];
int *ponteiro;
```

1. `vetor = vetor +2;`
2. `vetor++;`
3. `vetor = ponteiro;`
4. `ponteiro = vetor;`
5. `ponteiro = vetor+2;`

Estão **certas** as seguintes afirmativas (marque com um X):

☐ 2,4
☒ 3,5
☐ 1,2
☐ 4,5

4 (30 pontos) Balanceado

Uma sequência de N elementos V_i ORDENADOS, com N par, é dita balanceada quando a soma do maior elemento com o menor elemento é igual a soma do segundo maior elemento com o segundo menor elemento, que por sua vez é igual a soma do terceiro maior elemento com o terceiro menor elemento, e assim por diante. Escreva uma função **RECURSIVA** em `C` que decida se o vetor está balanceado ou não. Esta função deverá retornar 1 quanto estiver balanceado e 0 quando não estiver. Implemente quaisquer funções auxiliares que achar necessário.

A função tem o seguinte protótipo (e não pode ser modificado):

```
int balanceada(int *vetor, int inicio, int fim)
```

Restrições:

- $2 \leq N \leq 1000$
- $-1000 \leq V_i \leq 1000$

Exemplo de vetor balanceado:

```
1 3 7 11 15 17
```

Exemplo de vetor não balanceado:

```
3 3 5 8 9 10 11 27 32 41 45 51
```

4.1 Escreva a sua resposta no espaço abaixo: