Oficina de Programação em C (ICP037)

Prof. Ronald Souza – IC/UFRJ

Segunda Prova – 11/07/2024

GABARITO

Questão 1) (1.0 ponto)

Que saída o código abaixo imprimirá? *Na sua resposta, basta informar o que será impresso. Não é preciso descrever o passo a passo.*

```
#include <stdio.h>
void sequencia(int x) {
    if (x % 10) {
        sequencia(x / 10);
        printf("%d", x % 10);
    }
}
int main() {
      sequencia(40274);
      return 0;
}
```

RESPOSTA: 274

Questão 2) (2.5 pontos)

A função abaixo deve realizar uma busca binária de um inteiro (**num**) sobre um vetor de inteiros (**vet**) de tamanho **tam**, já ordenado em ordem **decrescente,** e deve retornar 1 caso o elemento buscado seja encontrado no vetor, e 0 caso contrário. No entanto, a função possui 7 erros, cada um em uma linha distinta. **Cuidado:** nem toda linha está errada!

→ Reescreva a função abaixo **preservando a formatação** e **corrigindo os seus 7 erros.** ***Não acrescente nem remova linhas em relação ao total de linhas do código abaixo.***

```
int busca(int num, int vet[], int tam) {
    int ini = 0, fim = tam, meio;
    while (meio < fim) {
       meio = ini+fim / 2;
       if (num == vet[meio])
             return 0;
       if (vet[meio] < num)
             ini = meio / 2;
       else
             fim = fim * 2;
    }
    return 1;
}
```

RESPOSTA:

```
int busca(int num, int vet[], int tam) {
    int ini = 0, fim = tam - 1, meio;
    while (ini <= fim) {
       meio = (ini+fim) / 2;   //Sem os parênteses, a precedência da divisão computa "fim / 2" primeiro.
       if (num == vet[meio])
             return 1;         //Elemento encontrado!
       if (vet[meio] < num)
             fim = meio - 1;   //Pois o vetor está em ordem DECRESCENTE.
       else
             ini = meio + 1;   //Note que trocaríamos essa condição com a de cima se o vetor fosse crescente.
    }
    return 0;                  //Elemento não foi encontrado.
}
```

Questão 3 (2.5 pontos) Chama-se **Quadrado Mágico** uma matriz quadrada de números em que a soma de cada coluna, de cada linha e das duas diagonais são iguais. Veja o exemplo para uma matriz 3x3:

2 7 6
9 5 1
4 3 8

*Repare acima que a soma dos elementos de **cada linha**, de **cada coluna** e das **duas diagonais** é sempre igual a 15!*

Deseja-se implementar uma **função** que receba de entrada uma matriz quadrada e sua dimensão, e retorne 1 caso a matriz seja um quadrado mágico ou retorne 0, caso contrário. A escrita da função foi iniciada no código abaixo, onde já são verificadas as duas diagonais (principal e secundária). Porém, ainda resta verificar se a propriedade também vale para **todas as linhas** e **todas as colunas**. **Complete o código abaixo, incluindo tal verificação.** *Na sua resposta, não é preciso reescrever toda a função, bastando-se implementar a parte que falta.*

```
#define N 100
int ehMagico(int m[N][N], int tam) {
	int ref = 0, prox;
	for (int i = 0; i < tam; i++)
		ref += m[0][i];

	//Diagonal Principal:
	prox = 0;
	for (int i = 0; i < tam; i++)
		prox += m[i][i];
	if (prox != ref)
		return 0; //Não é mágico :(

	//Diagonal Secundária:
	prox = 0;
	for (int i = 0; i < tam; i++)
		prox += m[tam - i - 1][i];
	if (prox != ref)
		return 0; //Não é mágico :(

	//Linhas e colunas:
	//IMPLEMENTAR!

	return 1; //Passou nos testes!  É mágico! :)
}
```

RESPOSTA NA PRÓXIMA PÁGINA

RESPOSTA DA QUESTÃO 3:

```
for (int i = 0; i < tam; ++i) {
      int l = 0, c = 0;
      for (int j = 0; j < tam; ++j) {
            l += m[i][j];
            c += m[j][i]; //Note que apenas trocamos i com j.
      }
      if (l != ref || c != ref) return 0;
}
```

UMA RESPOSTA ALTERNATIVA (linhas e colunas tratadas separadamente):

```
      //Linhas:
      for (int i = 0; i < tam; i++) {
            prox = 0; //Zeramos a condição para cada linha
            for (int j = 0; j < tam; j++)
                  prox += m[i][j];
            if (prox != ref) return 0;
      }

      //Colunas:
      for (int i = 0; i < tam; i++) {
            prox = 0; //Zeramos a condição para cada coluna
            for (int j = 0; j < tam; j++)
                  prox += m[j][i]; //Note que apenas trocamos i com j.
            if (prox != ref) return 0;
      }
```

Questão 4) Trabalharemos com o Tipo Abstrato de Dado (TAD) **`Ponto Cartesiano`**. Para isso, siga os seguintes passos:

a) **(1.0 ponto)** Crie o **tipo estruturado `tPonto`**. Sua estrutura conterá 2 **racionais**, **`x e y`**, referentes às coordenadas do ponto no plano cartesiano.

RESPOSTA:

```
typedef struct {
	float x, y;
} tPonto;
```

b) **(1.0 ponto)**

Agora, crie uma função que, dados **2 racionais quaisquer** de entrada, ***a*** e ***b,*** retorne um **`Ponto Cartesiano`** de coordenadas (a, b).

RESPOSTA:

```
tPonto ponto(float a, float b) {
	tPonto p = { a, b };
	return p;
}
```

c) **(1.5 ponto)** Crie uma **função** que receba um **vetor de pontos cartesianos** e também **o seu tamanho N**, e **retorne o seu centróide**, ou seja, o **ponto cartesiano** correspondente à média deles, cujas coordenadas são *(média dos valores de x, média dos valores de y)*.

RESPOSTA:

```
tPonto centroide(tPonto vet[], int tam) {
	tPonto c = { 0,0 };
	for (int i = 0; i < tam; i++){
		c.x += vet[i].x;
		c.y += vet[i].y;
	}
	c.x /= tam;
	c.y /= tam;
	return c;
}
```

d) **(2.0 pontos)** Crie uma **função** que receba um vetor de pontos cartesianos e seu tamanho N, e então (i) ordena esse vetor em ordem **crescente** da coordenada x de cada ponto pelo método **Bubble Sort** e (ii) imprime todos os pontos no formato "(x,y)", **seguindo a ordenação crescente estabelecida no item (i)**. Sua função **não** deve retornar valores.

RESPOSTA:

```
void imprimeOrdenado(tPonto vet[], int tam) {
	const int V = 1, F = 0;
	int ordenado = F;
	int fim = tam - 1;
	while (!ordenado) {
		ordenado = V;
		for (int i = 0; i < fim; i++) {
		if (vet[i].x > vet[i+1].x) {
			ordenado = F;
			tPonto aux = vet[i];
			vet[i] = vet[i + 1];
			vet[i + 1] = aux;
		}
		}
		--fim;
	}

	//Impressão:
	for (int i = 0; i < tam; i++)
		printf("(%.1f, %.1f) ", vet[i].x, vet[i].y);
}
```

Questão 5) (1.5 pontos)
Deseja-se implementar uma função **recursiva** que verifica se um **vetor de inteiros** é um palíndromo i.e. se gera a mesma sequência se lido "de trás pra frente" por ex.: {7,4,4,7}), retornando 1 se verdadeiro e 0 caso contrário.
Em seu caderno de respostas (**e não aqui!**), indique **claramente, em letra de forma,** a alternativa correta para ser o corpo da função **palindromo()** abaixo:

```
int palindromo(int v[], int ini, int fim) {
    // O que retornar??
}
```

```
A) return (ini == fim) ? 1 : (v[ini] == v[fim]) && palindromo(v, ini + 1, fim - 1);
B) return (ini <= fim) ? 1 : (v[ini] == v[fim]) || palindromo(v, ini + 1, fim - 1);
C) return (ini > fim)  ? 0 : (v[ini] == v[fim]) && palindromo(v, ini + 1, fim - 1);
D) return (ini != fim) ? 0 : (v[ini] == v[fim]) || palindromo(v, ini + 1, fim - 1);
E) return (ini > fim)  ? 1 : (v[ini] == v[fim]) && palindromo(v, ini + 1, fim - 1);
```

//Exemplo de chamada:

```
#include <stdio.h>
#define TAM_v1 5
#define TAM_v2 6
int main() {
    int v1[TAM_v1] = { 5,4,1,4,5 };    //Tamanho ímpar
    int v2[TAM_v2] = { 3,5,4,4,5,3 };  //Tamanho par
    puts("v1 eh palindromo?");
    puts((palindromo(v1, 0, TAM_v1 - 1)) ? "sim" : "nao"); //Imprimirá "sim".
    puts("v2 eh palindromo?");
    puts((palindromo(v2, 0, TAM_v2 - 1)) ? "sim" : "nao"); //Imprimirá "sim".
    return 0;
}
```

RESPOSTA:
Letra **E:** `return (ini > fim)  ? 1 : (v[ini] == v[fim]) && palindromo(v, ini + 1, fim - 1);`