Segunda Atividade de Avaliação Escrita

CET641 - LPII - Turma 2023.2 - 09/10/2023

Aluno:_____

Primeira Etapa - Prova Objetiva

(Valendo 3 de 10)

Questão 1 (1,5 de 3)- Um grupo de alunos realizou uma avaliação que consiste em responder 200 questões, cada uma valendo um ponto. O professor deseja saber como ficou a distribuição da pontuação da turma, sabendo que nenhum aluno zerou. Para isso ele fez o seguinte programa :

```c
int* contNotas(int notas[], int* tam);

int main(int argc, char const *argv[])
{
    int pontuacao[300];
    //Apenas para testar
    for(int i = 0; i < 300; i++)
        pontuacao[i] = 45 + rand() % 150;
    int tam = 300;
    int* qNotas = contNotas(pontuacao, &tam);
    for(int i = 0; i < tam; i++)
        printf("%d \n", qNotas[i]);
    free(qNotas);
    return 0;
}
```

A função `contNotas`, cujo cabeçalho está no código, recebe a array com a pontuação de cada aluno e a quantidade de alunos, por referência. Na função se deve determinar a pontuação mínima e máxima da turma e construir o histograma com a distribuição de pontos entre o valor mínimo e o máximo, e o tamanho deste histograma. Os códigos a seguir tentam fazer a implementação desta função. Marque quais implementações estão corretas (**V**) e quais não (**F**). Justifique sua resposta para aqueles casos que considerar incorretos.

a)

```c
int* contNotas_V1(int notas[], int* tam){
    int min, max;
    min = max = notas[0];
    for(int i = 1; i < *tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    int* qNotas = (int*)malloc((max - min + 1) * sizeof(int));
    for(int i = 0; i < *tam; i++)
            qNotas[i]++;
    *tam = max - min + 1;
    return qNotas;
}
```

b)

```c
int* contNotas_V2(int notas[], int* tam){
    int min, max;
    min = max = notas[0];
    for(int i = 1; i < *tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    int* qNotas = (int*)calloc((max - min + 1), sizeof(int));
    for(int i = 0; i < *tam; i++)
            qNotas[notas[i] - min]++;
    *tam = max - min + 1;
    return qNotas;
}
```

c)

```c
int* contNotas_V3(int notas[], int* tam){
    int min, max;
    for(int i = 0; i < tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    int* qNotas = (int*)calloc((max - min + 1), sizeof(int));
    for(int i = 0; i < tam; i++)
            qNotas[notas[i] - min]++;
    tam = max - min + 1;
    return qNotas;
}
```

d)

```c
int* contNotas_V4(int notas[], int* tam){
    int min, max;
    for(int i = 0; i < tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    *tam = max - min + 1;
    int* qNotas = (int*)malloc(*tam * sizeof(int));
    for(int i = 0; i < *tam; i++)
            qNotas[i] = 0;
    for(int i = 0; i < *tam; i++)
            qNotas[notas[i] - min]++;
    return qNotas;
}
```

e)

```c
int* contNotas_V5(int notas[], int *tam){
    int min, max;
    min = max = notas[0];
    for(int i = 1; i < *tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    int intervalo = max - min + 1;
    int* qNotas = (int*)malloc(intervalo * sizeof(int));
    for(int i = 0; i < intervalo; i++)
            qNotas[i] = 0;
    for(int i = 0; i < *tam; i++)
            qNotas[notas[i] - min]++;
    *tam = intervalo;
    return qNotas;
}
```

f)

```c
int* contNotas_V6(int notas[], int* tam){
    int min, max;
    min = max = notas[0];
    for(int i = 1; i < tam; i++)
    {
        if(min > notas[i])
            min = notas[i];
        else if(max < notas[i])
            max = notas[i];
    }
    int* qNotas = (int*)calloc((max - min + 1), sizeof(int));
    for(int i = 0; i < tam; i++)
            qNotas[i]++;
    tam = max - min + 1;
    return qNotas;
}
```

Questão 2 (1,5 de 3) - Para melhorar a implementação do exercício anterior, o professor decidiu armazenar as pontuações numa lista encadeada. Veja o código que ele implementou para testar:

```c
struct notas{
    int pont;
    struct notas* prox;
};

typedef struct notas Notas;

Notas* array2list(int notas[], int tam);
int contNotas(Notas* lista, int* max, int* min);

int main(int argc, char const *argv[])
{
    int pontuacao[300];
    //Apenas para testar
    for(int i = 0; i < 300; i++)
        pontuacao[i] = 45 + rand() % 150;
    int tam = 300;
    Notas* lista = array2list(pontuacao, tam);
    int max, min;
    int cont = contNotas(lista, &max, &min);
    return 0;
}
```

No exemplo, os elementos do array são armazenados numa lista com ajuda da função `array2list`. Posteriormente, a função `contNotas` retorna quantos elementos tem na lista e qual a maior e a menor pontuação. Dos seguintes exemplos, determine quais podem ser utilizados (V) e quais não (F) para estas finalidades.:

a)

```c
int contNotasV1(Notas* lista, int* max, int* min)
{
    int cont = 0;
    if (lista != NULL)
    {
        *max = *min = lista->pont;
        cont = 1;
        Notas* aux = lista->prox;
        while(aux != NULL)
        {
            if(*min > aux->pont)
                *min = aux->pont;
            else if(*max < aux->pont)
                *max = aux->pont;
            aux = aux->prox;
            cont++;
        }
    }
    return cont;
}
```

b)

```c
int contNotasV2(Notas* lista, int* max, int* min)
{
    if (lista == NULL)
        return 0;
    if (lista->prox == NULL)
    {
        *max = *min = lista->pont;
        return 1;
    }
    int cont = contNotasV2(lista->prox, max, min);
    if(*min > lista->pont)
        *min = lista->pont;
    else if(*max < lista->pont)
        *max = lista->pont;
    return cont + 1;
}
```

c)

```c
Notas* array2list_V1(int notas[], int tam)
{
    Notas* lista = NULL;
    for(int i = 0; i < tam; i++)
    {
        Notas* novo = (Notas*)malloc(sizeof(Notas));
        novo->pont = notas[i];
        novo->prox = lista;
        lista = novo;
    }
    return lista;
}
```

d)

```c
Notas* array2list_V2(int notas[], int tam)
{
    Notas* lista = NULL;
    if (tam != 0)
    {
        lista = (Notas*)malloc(sizeof(Notas));
        lista->pont = notas[0];
        lista->prox = array2list_V2(notas + 1, tam - 1);
    }
    return lista;
}
```

Segunda Etapa - Prova Discursiva

(Valendo 5 de 10)

Questão 3(1,0 de 5) - Implementar a seguinte função: `char* geraPalavra(int tam)` que gera uma string com uma "palavra", gerada de forma aleatória misturando letras e alternando consoantes e vogais. As palavras devem ter no mínimo dois caracteres e no máximo `tam` caracteres, todos minúsculos.

Questão 4(2,0 de 5) - Crie uma função, para gerar uma imagem, que recebe dois inteiros contendo o número de linhas e colunas da mesma. A função retorna um ponteiro de ponteiros para inteiros (`int** geraImagem(int lin, int col)`). A matriz gerada

representa uma imagem, contendo `lin x col` pixels, e em cada posição da mesma temos a intensidade da cor do pixel correspondente, preenchidos de forma aleatória, com valores entre 0 e 255.

Questão 5(2,0 de 5) - Crie uma função que recebe um ponteiro de ponteiros para inteiros, contendo valores entre 0 e 255, dois inteiros contendo o número de linhas e colunas do array, e retorna um ponteiro de ponteiros para inteiros (`int** filtroMaximo(int **img, int lin, int col)`). O array de entrada representa uma imagem, contendo `lin x col` pixels, e em cada posição da mesma temos a intensidade da cor do pixel correspondente (valores entre 0 e 255). A função retorna o endereço de uma nova imagem, alocado dinamicamente, gerada a partir da aplicação de um filtro de máximo na imagem original. Isto é, cada pixel, nas coordenadas `(i, j)` da nova imagem, se constrói como o valor máximo da intensidade das cores nos pontos `(i, j)`, `(i+1, j)`, `(i-1, j)`, `(i, j+1)`, `(i, j-1)`, `(i+1, j-1)`, `(i+1, j+1)`, `(i-1, j-1)`, `(i-1, j-1)` da imagem original. Para os pixels nas bordas mantemos a cor original.

Terceira Etapa - Prova de Prática

(Valendo 2 de 10)

Implemente e teste os códigos dos exercícios da etapa anterior. Para cada exercício prepare um exemplo para testar suas funções. Envie pelo Classroom no prazo estabelecido os códigos desenvolvidos. O código deverá compilar e rodar, sem modificações, e funcionar de acordo com o esperado em cada caso. Indique, quando possível, alguns casos para testar seu código e qual deve ser o resultado em cada caso.