ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

PCS 3111 - Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

2020

Aula 02 – Ponteiros, Testes e Depuração

Atenção

1. Código inicial para resolução dos exercícios encontra-se **disponível no e-Disciplinas**.
2. Os tipos, os nomes e os parâmetros das funções **devem seguir o especificado** em cada exercício para fins de correção automática.
3. A função `main` **não deve ser submetida**. Caso contrário, a correção automática retornará um *Compilation Error.*

Exercício 1

Considere um software que tem como função gerenciar as informações de uma competição esportiva com diversas **equipes**. Suponha que essas equipes tem duas informações importantes: o `nome e o número de membros`, sendo elas armazenadas em dois vetores diferentes.
Por exemplo, se tivéssemos os seguintes vetores:

```
nomes = {“Poli”, “FEA”, “ESALQ”}, membros = {10, 9, 10}
```

teríamos três equipes: Poli, com 10 membros, FEA, com 9 membros, e ESALQ, com 10 membros.

Deseja-se implementar a seguinte função:

```
string* encontrarOponente(string nomes[], int membros[], int quantidade,
                          string nomeEquipe, int membrosEquipe);
```

Essa função deve procurar um oponente para uma equipe cujo nome e número de membros são passados nos parâmetros `nomeEquipe` e `membrosEquipe`, respectivamente. O parâmetro `quantidade` se refere ao tamanho dos dois vetores. Para que um oponente seja escolhido, temos duas regras:

1. O número de membros das equipes deve ser **igual**.
2. Para que a própria equipe não seja escolhida como oponente de si mesma, seu oponente deve ter um nome **diferente** do dela. Considere que equipes apresentam sempre nomes diferentes.

A função deve retornar um ponteiro para o **índice do vetor `nomes`** onde está guardado o nome do oponente escolhido. Caso haja mais de um candidato a oponente, o escolhido deve ser o **último** da lista. Caso não haja candidatos a oponente, retorne o valor NULL.

Usando como exemplo os vetores apresentados anteriormente, se `nomeEquipe == “Poli”` e `membrosEquipe == 10`, a função deve retornar o **ponteiro** equivalente ao índice 2 do vetor **`nomes`**, referente ao nome “`ESALQ`” (ou seja, será uma posição de memória como 0x6efea8).

Obs: não se esqueça fazer o include de **<string>** no arquivo e adicionar o comando **using namespace std;**

Exercício 2
Parâmetros podem ser usados como entrada e/ou saída de funções. Para se ter um parâmetro de saída pode-se usar ponteiros ou passagem por referência. Trabalharemos agora com três tipos de saída: através do retorno da função, através de ponteiros e através de referência. Implemente a função a seguir:

```
int calcularEstatisticas (int membros[], int quantidade, int* minimo,
                          int& maximo);
```

Essa função recebe o vetor `membros` cujo tamanho é definido em `quantidade`. Os valores do número de membros da menor equipe e da maior equipe são retornados nos ponteiros `minimo` e `maximo`, respectivamente. O valor da quantidade total de membros (representada pela soma dos membros de todas as equipes) é retornada por meio da função (comando **return**).

Note que o valor do mínimo é passado como ponteiro e o máximo é passado por referência.

Assuma que os valores passados pelos vetores sejam sempre positivos. Para o caso de nenhuma equipe estar presente na competição, ou seja, `quantidade == 0`, todos os valores retornados devem valer 0.

Por exemplo, considere os vetores do exercício 1. Teríamos que o mínimo valeria 9, o máximo valeria 10, e o retorno da função daria 10 + 9 + 10 = 29.

Cuidado: não assuma valores iniciais para os parâmetros passados pelo main.

Testes do Judge

Exercício 1
- Testa com oponente no início do vetor;
- Testa com oponente no meio do vetor;
- Testa com oponente no fim do vetor;
- Testa sem oponente;
- Testa com dois candidatos a oponentes;

Exercício 2
- Testa com quantidade 0;
- Testa com quantidade 1;
- Testa com quantidade maior do que 1;