Ponteiros

1. Para cada um dos itens seguintes, escreva uma única instrução que realiza a tarefa indicada. Suponha que as variáveis do tipo inteiro `long value1` e `value2` tenham sido declaradas e que value1 tenha sido inicializado como 200000.

   (a) Declare a variável `longPtr` como um ponteiro para um objeto do tipo `long`.

   (b) Atribua o endereço da variável `value1` à variável ponteiro `longPtr`.

   (c) Imprima o valor do objeto apontado por `longPtr`.

   (d) Atribua o valor do objeto apontado por `longPtr` à variável `value2`.

   (e) Imprima o valor de `value2`.

   (f) Imprima o endereço de `value1`.

   (g) Imprima o endereço armazenado em `longPtr`. O valor impresso é o mesmo que o endereço de `value1`? Escreva sua observação como um comentário no seu código submetido.

   O seu código ("`pratica1_exercicio1.cpp`") deve apresentar a seguinte estrutura:

```
int main() {
    long int value1 = 200000;
    long int value2 = 0;
    // insira seu codigo aqui...
    // insira aqui a resposta para o item (g)...
    return 0;
}
```

2. Implemente duas funções com nome `leiaInteiros` (`leiaInteiros1` e `leiaInteiros2`). Quando chamada, cada função deverá pedir para o usuário digitar dois números inteiros e, então, ler tais números a partir do teclado. Faça com que cada função receba 2 parâmetros e grave os dois inteiros lidos nesses parâmetros. Na primeira função, os parâmetros deverão ser passados por referência, enquanto na segunda função eles deverão ser passados utilizando ponteiros.

   Implemente também um método `main` e faça com que ele leia dois números utilizando a primeira versão da função `leiaInteiros` (`leiaInteiros1`) e, então, imprima esses números na tela. Após isso, seu método `main` deverá fazer a mesma coisa utilizando a segunda versão da função (`leiaInteiros2`).

   O seu código ("`pratica1_exercicio2.cpp`") deve apresentar a seguinte estrutura:

```
void leiaInteiros1(int ..., int ...){
    // insira seu codigo aqui...
}
void leiaInteiros2(int ..., int ...){
    // insira seu codigo aqui...
}
int main() {
    int a, b;
    // passagem por referencia
    leiaInteiros1(...);
    // insira seu codigo aqui...
    // utilize ponteiros
    leiaInteiros2(...);
    // insira seu codigo aqui...
    return 0;
}
```

3. Implemente três funções com nome `calculaTamanhoStringX` (para diferenciá-las, troque `X` por 1, 2 e 3). Suas funções deverão receber como parâmetro um apontador para char e, então, contar quantos caracteres a *string* apontada possui. Lembre-se que as *strings* "de C" (representadas por arranjos de caracteres) possuem um caractere de terminação (o caractere '\0').

   (a) A primeira função deverá utilizar uma estrutura for para varrer o arranjo de caracteres de forma "tradicional" (utilizando o [ ]).

   (b) A segunda função deverá utilizar uma estrutura do tipo for que deverá, então, utilizar aritmética de ponteiros (para acessar a posição i do arranjo, basta somar i ao ponteiro) para acessar os caracteres apontados pelo ponteiro.

   (c) A terceira função deverá utilizar uma estrutura do tipo for que deverá incrementar o ponteiro de modo a se deslocar pelo arranjo de caracteres. Nessa última função, você poderá declarar APENAS variáveis do tipo apontador para char! Além disso, não utilize a notação de arranjo ([ ]) para acessar os dados. Dica: utilize o operador `sizeof()` para saber o número de bytes ocupados por cada caractere.

   O seu código ("`pratica1_exercicio3.cpp`") deve apresentar a seguinte estrutura. Além disso, utilize a função `main` abaixo para testar suas funções:

```
int calculaTamanhoString1(char ...){
    // insira seu codigo aqui...
}

int calculaTamanhoString2(char ...){
    // insira seu codigo aqui...
}

int calculaTamanhoString3(char ...){
    // insira seu codigo aqui...
}

int main() {
    char str[51];
    cout << "Digite alguma string... (com ate 50 caracteres):";
    cin.getline(str, 50);
    cout << "Tamanhos calculados:" << endl;
    cout << calculaTamanhoString1(str) << " " << calculaTamanhoString2(str
        ) << " " << calculaTamanhoString3(str);
    cout << endl;
    return 0;
}
```

4. Considerando o registro abaixo, faça um programa com uma função `main` e uma função `leDadosJogador`. Na função `main`, declare um *array* com 5 jogadores e, em um laço do tipo `for`, chame a função `leDadosJogador` para cada jogador no arranjo e, após isso, varrer o arranjo novamente imprimindo os dados dos jogadores. A função `leDadosJogador` deverá receber como parâmetro um ponteiro para `jogador`, ler a partir do usuário os dados de um `jogador` (pontos, coordenadas x e y) e, finalmente, gravar tais dados no jogador recebido como parâmetro.

   O seu código ("`pratica1_exercicio4.cpp`") deve apresentar a seguinte estrutura:

```
struct Jogador {
    int pontos;
    int x,y;
};

```

```
void leDadosJogador(Jogador ...){
    // insira seu codigo aqui...
}

int main(){
    Jogador j[5];
    // insira seu codigo aqui...
    return 0;
}
```

5. Complete o trecho de código abaixo ("`pratica1_exercicio5.cpp`") de modo que ele imprima a *string* em ordem reversa. *ATENÇÃO!*: Não utilize a notação [ ] no trecho que você completar. Utilize apenas operações sobre apontadores!

```
int main() {
        char str[] = "abc teste";

        //Insira o codigo aqui...
        //Exemplo de saida para o exemplo acima: etset cba

        return 0;
}
```

Considerações Gerais!

- Exercício individual.
- Entrega: conforme agendado no PVANET Moodle;
- Conforme especificado crie um projeto para resolução de cada exercício (ex.: pratica1_exercicio1.cpp, pratica1_exercicio2.cpp, etc). Envie, através do PVANet Moodle, uma pasta compactada (.rar ou .zip) contendo todos os projetos. A pasta compactada deve conter informações do aluno (ex.: julio_reis-pratica1.zip).