EXERCÍCIOS DE FIXAÇÃO Nº 06.1
Básico sobre ponteiros

1) Qual a função dos operadores **&** e ***** quando associados a ponteiros? Exemplifique com código em C.

O operador & significa "endereço de uma variável".
O operador * (operador de indireção) representa o valor armazenado em um endereço apontado por um ponteiro.

```
// Exemplo
int x=1;      // declaração de variável
int *px;      // declaração de um ponteiro
px=&x;        // ponteiro aponta para o endereço da variável x
*px=2;        // altera o valor da variável x indiretamente (através do ponteiro)
```

2) Por que é importante inicializar um ponteiro antes do seu uso?

odo ponteiro não inicializado aponta para um endereço desconhecido, o que pode gerar um risco alto do programa travar. Por este motive, é fundamental que todo ponteiro seja inicializado!

3) As variáveis são sempre armazenadas nos mesmos endereços?

Não. A cada nova execução do programa, as variáveis podem ser armazenadas em endereços diferentes.

4) O que é indireção?

É o nome dado à operação de acesso ao conteúdo de uma variável através de um ponteiro.
O operador de indireção é o asterisco (*).

5) Como o compilador distingue o * usado para a multiplicação do * usado para "desreferenciamento" (acesso às informações existentes no endereço contido em um ponteiro) e do * usado para declarar um ponteiro?

O asterisco usado para a multiplicação aparece entre dois operandos, que pode ser um valor absoluto ou uma variável.
O asterisco usado para desreferenciamento (ou indireção) é colocado exclusivamente à frente de ponteiros e corresponde ao acesso do conteúdo do endereço acessado pelo ponteiro.
O asterisco usando na declaração de um ponteiro aparece exclusivamente no momento que o ponteiro é declarado, ou seja, quando há indicação do tipo de variável e, na mesma linha de código, há a indicação de um ponteiro.

6) Como os elementos de uma matriz são armazenados na memória?

Os valores de uma matriz são armazenados de forma contígua, ou seja, um valor imediatamente após o outro na memória. O número de bytes entre um elemento da matriz e o próximo varia conforme o tipo de dados declarado para a matriz.

7) Mostre duas maneiras de obter o endereço do primeiro elemento da matriz data[].

```
p1=&data[0];  // pode-se escrever explicitamente usando o operador &
p1=data;      // o nome da matriz também serve como indicativo do seu endereço inicial
```

8) Quando uma matriz é passada para uma função, quais são as duas maneiras de determinar onde a matriz termina?

1) É possível enviar, junto com o endereço da matriz, o número de elementos (linhas e colunas, se for o caso) que a matriz possui
2) É possível que a própria função encontre o "fim" da matriz com o uso da função "sizeof"

9) Cite seis operações que podem ser efetuadas com ponteiros e duas que não podem.

Operações que podem ser realizadas com ponteiros:

1) Inicialização (atribuição de um endereço ao ponteiro)
2) Indireção (acesso indireto ao conteúdo de uma variável)
3) Soma com valor escalar (para encontrar o próximo endereço de uma matriz/vetor)
4) Subtração de valor escalar (para encontrar o endereço anterior de uma matriz/vetor)
5) Subtração de um ponteiro por outro (para encontrar a "distância" entre os endereços)
6) Comparação entre ponteiros que apontam para uma matriz (uso de >,<,==,!==,>=,<=)

```
Operações que NÃO podem ser realizadas com ponteiros:
1) Multiplicação de ponteiros (produto de um ponteiro por outro)
2) Divisão de ponteiros (quociente de um ponteiro dividido por outro)
```

10) Suponha que você tenha dois ponteiros. Se o primeiro estiver apontando para o terceiro elemento de uma matriz do tipo *int* e o segundo para o quarto elemento da mesma matriz, que valor será obtido quando você subtrair o primeiro ponteiro do segundo?

```
Ao subtrair um ponteiro de outro, obtém-se a "distância" entre eles, ou seja, o número
de elementos que os separam. Neste caso, a distância entre o quarto elemento e o
terceiro é igual a UM.
// Exemplo
int matriz[5]={10,20,30,40,50};
int *p1,*p2,d;
p1=&matriz[3];  // p1 aponta para o terceiro elemento da matriz
p2=&matriz[4];  // p2 aponta para o quarto elemento da matriz
d=p2-p1;        // d irá receber o valor 1
```

11) Suponha que a matriz da questão anterior contenha valores do tipo *float*, que valor seria obtido com a subtração dos dois ponteiros?

```
Ao subtrair um ponteiro de outro, obtém-se a "distância" entre eles, ou seja, o número
de elementos que os separam. Esta operação não se altera, mesmo que o tipo de dados
seja diferente (int, float, char, long int, etc.). Sendo assim, a distância entre o
quarto elemento e o terceiro CONTINUARÁ sendo igual a UM.
// Exemplo
float matriz[5]={1.1,2.2,3.3,4.4,5.5},*p1,*p2;
int d;
p1=&matriz[3];  // p1 aponta para o terceiro elemento da matriz
p2=&matriz[4];  // p2 aponta para o quarto elemento da matriz
d=p2-p1;        // d irá receber o valor 1
```

12) Escreva uma declaração de um ponteiro chamado **char_ptr** para uma variável do tipo *char*.

```
char *char_ptr;  // a declaração de ponteiro é sempre igual, só altera o tipo de dados
```

13) Se um programa contivesse uma variável *int* chamada **coast**, como você declararia e utilizaria um ponteiro chamado **p_coast** para apontar para esta variável?

```
int coast, *p_coast;  // a declaração de ponteiro é igual, só altera o tipo
p_coast=&coast;
```

14) Continuando com o exercício 13, como você atribuiria o valor 100 à variável **coast** usando acesso direto e indireto?

```
coast=100;     // acesso direto (usando a própria variável)
*p_coast=100;  // acesso indireto (usando o ponteiro)
```

15) Continuando com o exercício 14, como você imprimiria o valor do ponteiro juntamente com o valor para o qual ele está apontando?

```
printf("Endereco: %p, Conteudo: %i.",p_coast,*p_coast);
```

16) Mostre como atribuir o endereço de um valor do tipo *float*, chamado **radius**, a um ponteiro.

```
float radius, *p_radius;  // a declaração de ponteiro é igual, só altera o tipo
p_radius=&radius;
```

17) Mostre duas maneiras de atribuir o valor 100 ao terceiro elemento da matriz **data[]**.

```
int data[5];
int *p1;
p1=data;     // p1 aponta para o primeiro elemento da matriz 'data'
*(p1+3)=100; // 1a. opção: de modo indireto, usando soma com escalar
p1[3]=100;   // 2a. opção: de modo indireto, usando o ponteiro com colchetes
```

18) Explique o que faz cada linha do trecho do programa abaixo:

```
int x=1, y=2, z[10];// declara duas variáveis inteiras x e y e uma matriz z, do tipo int
int *ip;            // declara um ponteiro para o tipo int
ip = &x;            // atribui o endereço de x para o ponteiro (inicialização)
y = *ip;            // atribui o valor 1 (conteúdo de x) para a variável y, usando indireção
*ip = 0;            // atribui o valor 0 para a variável x, usando indireção
ip=&z[0];           // faz com que o ponteiro aponte para o 1° elemento da matriz
```

19) Supondo o mesmo trecho de código do exercício anterior, explique cada uma das operações aritméticas abaixo, que utilizam ponteiros:

```
a)   y = *ip+1;    // atribui a y o valor do 1° elemento da matriz acrescido de 1
b)   *ip += 1;     // incrementa +1 ao valor do 1° elemento da matriz
c)   ++*ip;        // incrementa +1 ao valor do 1° elemento da matriz
d)   (*ip)++;      // incrementa +1 ao valor do 1° elemento da matriz
```

20) Considerando o fragmento de programa abaixo

```
{
      int a[10];
      int *pa;
      int aux;
      ...
      pa = a;
}
```

Complete as equivalências abaixo usando os conceitos de aritmética de ponteiros:

aux = a[2]	aux = pa[2] ou *(pa+2), usando "pa"
aux = a[i]	aux = pa[i] ou *(pa+i), usando "pa"
aux = a[2]	aux = *(a+2), usando "a"
aux = a[i]	aux = *(a+i), usando "a"
(a+2)	&a[2], usando "a"
(pa+1)	&a[1] ou (a+1), usando "a"