Exercícios de estruturas e ponteiros.

1) Complete a tabela abaixo com os respectivos valores após a execução de todas as instruções do código a seguir.

```
int main()
{
  int x=4;
  int y=7;
  int *p1;
  int *p2;
  p1 = &x;
  p2 = &y;
  x=y;
  *p2=12;
}
```

	Valor
x	
y	
*p1	
*p2	
&x	5000
&y	
p1	
p2	5004

2) Crie a função ordena que receba como parâmetro três variáveis (passagem por referência) e ordene os valores destas variáveis em ordem crescente, respeitando a ordem da passagem dos parâmetros. Por exemplo, considerando as seguintes variáveis com os valores: a=7, b=3 e c=5, se as mesmas forem passadas por referência, nesta ordem, para a função pedida, o valor das mesmas após o retorno da função deve ser a=3, b=5 e c=7.
3) Considere a seguinte sequência de instruções:

```
int x=10;
int *pt = &x;
```

Assinale cada uma das afirmações abaixo como verdadeiras (V) ou falsas (F), considerando que as mesmas são feitas após a execução das instruções acima.

( ) O valor de pt é 10

( ) o valor de *pt é igual ao valor de x

( ) pt armazena o endereço de x

( ) o * na segunda instrução diz respeito ao operador de ponteiro utilizado para acessar o que está no endereço armazenado em pt.

( ) o * na segunda instrução não é o operador de ponteiro utilizado para acessar o que está no endereço armazenado em pt, mas apenas o tipo utilizado na declaração da variável pt, indicando que a mesma é um ponteiro para inteiro, que em seguida é corretamente inicializada com um endereço de memória.

( ) a atribuição realizada na segunda instrução é válida, pois o * é utilizado na declaração de uma variável ponteiro, que é inicializada com um endereço de memória. O * nesta linha não diz respeito ao operador * para ponteiro.

4) Complete a tabela abaixo com os respectivos valores após a execução de todas as instruções do código a seguir (escreva “erro” no local do valor, caso a expressão na primeira coluna da tabela seja inválida)

```
int main()
{
   float a=2.5;
   float *pf;
   float **ppf;
   pf=&a;
   ppf=&pf;
}
```

	Valor
a	2.5
sizeof(a)	4
&a	3008
&pf	3004
&ppf	3000
*pf	
**ppf	
&pf	
**&pf	
pf	
*&*pf	
*&a	
pf	
ppf	
*ppf	
&**pf	

5) Faça um programa que leia números inteiros digitados pelo usuário e os armazene em um vetor. O programa deve ler os números digitados até que o usuário digite um número negativo (e este não deve ser armazenado no vetor). A memória utilizada no programa deve ser alocada dinamicamente. Ao iniciar, o tamanho do vetor deve ser 5. Antes de armazenar um novo valor ao vetor, o programa deve verificar se o tamanho do vetor não será excedido e, caso isso vá ocorrer, o tamanho do vetor deve ser aumentado para que este permita o armazenamento de duas vezes mais elementos do que o permitido no momento (ou seja, na primeira vez que o tamanho do vetor é aumentado, este passa a possuir tamanho 10, na segunda vez, tamanho 20, e assim por diante). Utilize neste programa as funções malloc e free (não esqueça de liberar memória não utilizada, caso seja necessário).