LISTA 7 - 13/05

Questão 1

Um ponteiro é uma variável com um determinado endereço de memória, assim é possível armazenar a memória de uma variável em outra utilizando o * entre o nome e o tipo da variável.

Questão 2

O ‘*’ no ponteiro serve para declara uma variável com ponteiro, assim ele acessa a memória da variável em questão, assim no exemplo do enunciado mostra de p é um ponteiro e terá seu valor baseado em outra variável.

Questão 3

a) p=&i indica que p terá o valor do endereço da variável i.
b) *p=i indica que p é um ponteiro em relação a i.

Questão 4

A função malloc() separa um espaço na memória RAM e depois retorna o endereço dessa memória e o número de bytes deve ser especificado entre os () para a ocupação da variável. A função free() serve para liberar espaço dentro da memória do código para que seja possível continuar a programação do mesmo, como uma forma de reciclar o espaço utilizado pelo malloc anteriormente, além disso o free deve ser utilizado no bloco todo, não somente em uma parte da programação.

Questão 5

A principal diferença entre malloc e calloc é a declaração entre os parênteses, o malloc tem como declaração um único número, o do byte, já o calloc precisa de 2 argumentos, o número de variáveis e o tamanho de bytes em cada uma das variáveis declaradas dentro do ().

Questão 6

O valor de x e y vai ser 12, já que x e y passa a ter o valor de *px que é igual a 12.

Questão 7

O valor de px vai ser igual a 3, já que *px tem o valor de x, é que 3, e py é igual a 4 já que *py obtém o valor de y que foi definido antes do printf como 4.

Questão 8

O valor de x e y será 3, já que x e y assumem o valor de A e B, respectivamente por causa da chamada do Troca(&x,y), e nesse parâmetro A e B passam ater o mesmo valor.

Questão 9

Os valores de ptr e x serão diferentes, tanto do espaço de memória quanto o valor final. O ptr inicia indicado como um ponteiro, e sua memória definida como o tamanho padrão do int, logo após seu valor é definido como 10, sem ter relação com nenhuma outra variável. O x é definido no início do código como 6 e não é alterado durante todo o restante, por isso segue com o valor de 6 até o fim, além disso seu a memória não é alterada, sendo em um local diferente o ptr, demostrando que as duas variáveis têm valores diferentes em todas as partes do código.

Questão 10

No 1º printf o valor impresso será o da memória que o ptr está localizado, por causa do malloc colocado para definição da varável, já no segundo printf o valor impresso será 0, já que antes do printf possui o free(ptr) que zera o valor definido anteriormente.

Questão 11

No código 1 o ptr está sendo salvo dentro da estrutura de repetição, tendo assim um valor final de 15. Já o código 2 a mudança dos números é feita na memória da variável, assim a repetição ocorre e depois será adicionado no valor final da memória, por causa da ausência do *.

Questão 12

Ele não possui nenhum erro de código, mas ao excuta-lo, por exemplo, o ptr acumula lixo de memória, por isso nunca será um valor correto e fixo, depois de rodar o código pela 1º vez o número que supostamente seria 0, passa a ser o valor final do último resultado.

Questão 13

```c
#include <stdio.h>
#include<stdlib.h>
#include <math.h>

int raizes(float a , float b, float c, float *x1, float *x2);
int main(void) {
  double a, b, c, delta=0, x1=0, x2=0;
  printf ("Digite o coeficiente da equação: ");
  scanf ("%lf", &a);
  if (a == 0) {printf ("Não é uma equação de 2º grau");}
  else {
  printf ("Digite os coeficientes de b e c: ");
  scanf ("%lf" "%lf", &b, &c);
  raizes(a, b, c, &x1, &x2);
  return 0;
  }
}
```

```c
int raizes(float a , float b, float c, float *x1, float *x2){
  float delta;
  delta = pow (b, 2) - (4*a*c);
  if (delta < 0) {printf ("Não existem raízes");}
  else if (delta == 0) {
    *x1 = (-b / (2*a));
    printf ("Existem duas raízes iguais: %.2lf", *x1); }
  else{
    *x1 = (-b + (sqrt (delta))/2*a);
    *x2 = (-b - (sqrt (delta))/2*a);
    printf ("As raízes são: %.2lf %.2lf", *x1, *x2);
  }
  return 0;
}
```