1. Note que, quando misturamos valores int e float na conta, o resultado é float:

```
11
16
1
1.250000
2
1
-5 -5
5.100000
4
```

---

2. Note que depois do #include <math.h>, M_PI é simplesmente um valor que existe para o nosso programa!

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

int main ()
{
    float r;
    scanf ("%f", &r);

    printf ("%f", (4.0/3.0)*M_PI*r*r*r);

    return (0);
}
```

---

3. O programa imprime:

```
3.539000
4
3.5
3.54
3.539
3.5390
3.53900
```

Pode-se observar que a sequência %.xf, onde x é um número inteiro, apresenta na saída um número real arredondado para x casas decimais.

4.

20
20
20
10
11
21
12

5.

```
#include <stdio.h>

int main () {
    float x1, x2, x3, x4, x5;
    float media;

    scanf ("%f %f %f %f %f", &x1, &x2, &x3, &x4, &x5);

    media = (x1 + x2)/2.0;
    printf ("%f\n", media);
    media = (x1 + x2 + x3)/3.0;
    printf ("%f\n", media);
    media = (x1 + x2 + x3 + x4)/4.0;
    printf ("%f\n", media);
    media = (x1 + x2 + x3 + x4 + x5)/5.0;
    printf ("%f\n", media);

    return 0;
}
```

---

6. Note que as variáveis aqui têm nomes mnemônicos. É fácil entender o que cada variável representa. Se usássemos coisas como a, b e c, o significado de cada variável para o programa seria bem menos claro.

```
#include <stdio.h>

int main () {
    int segundos_total;
    int dias, horas, minutos, segundos;

    scanf ("%d", &segundos_total);

    /* 1 dia tem 24 horas, 1 hora tem 60 minutos, 1 minuto tem 60 segundos. */
    /* Desconta os dias do total de segundos. */
    dias = segundos_total / (24 * 60 * 60);
    segundos_total = segundos_total - (dias * 24 * 60 * 60);

    /* Desconta as horas dos segundos restantes. */
    horas = segundos_total / (60 * 60);
    segundos_total = segundos_total - (horas * 60 * 60);

    /* Desconta os minutos dos segundos restantes. */
    minutos = segundos_total / 60;
    segundos = segundos_total - (minutos * 60);

    printf ("%d dias, %d horas, %d minutos, %d segundos\n",
        dias, horas, minutos, segundos);

    return (0);
}
```

7.

printf ("%f\n", x1 % x2); -> Tentando usar o operador % com variáveis float.

printf ("%f\n", &x1); -> Com o & antes de x1, não mostramos o valor de x1.

x1 = x2 -> Faltou o ;

printf ("%d\n", x3); -> x3 não foi inicializada.

X2 = 10; -> X2 não existe. X2 é diferente de x2.

x1 + 10.0; -> Esta linha até pode ser compilada e executada, mas não tem efeito!

x3 = 039; -> NÃO ADICIONE ZEROS à esquerda dos números, a não ser que você queria usar números octais (e se você não sabe o que é um número octal, não quer usar um!). Além disso, o dígito 9 não poderia ser usado em um número octal.

---

8. O código pode ou não apresentar um erro, dependendo do compilador que você está usando. Mais especificamente, ele não produz erro se o seu compilador tiver sido compilado para arquiteturas e sistemas operacionais de 64 bits (mas produzirá erro se o segundo valor for substituído por 9999999999999999999). O problema é que não é possível representar certos valores com uma quantidade de bits. O que EXATAMENTE significa isso - e o que causa o erro será estudado em algumas aulas!

---

9. Se o compilador estiver gerando programas que sequem à risca o que está indicado nos códigos, os programas serão diferentes. O primeiro programa simplesmente mostra o texto "1, 2, 3", enquanto o segundo converte os números 1, 2 e 3 para o texto "1, 2, 3" antes de mostrar. O segundo programa teoricamente seria então mais lento, mas na verdade a diferença é irrisória e humanamente imperceptível!

---

10. A resposta é mais complicada do que pode parecer à primeira vista, mas tem vários motivos. Discutiremos eles em aulas futuras!

---

11. x*x*x é mais simples E mais eficiente do que pow (x,3). A chamada a pow envolve trocas de contexto, e possivelmente conversões implícitas de tipos. Você provavelmente não vai notar diferença nenhuma se usar pow uma, dez, ou mil vezes. Mas se isso ocorrer milhões de vezes, a diferença pode se tornar considerável. Isso deve ficar mais claro no decorrer da disciplina!