1.

```
/* Para arredondar um número positivo, basta somar 0.5 ao mesmo e truncar. Se a
  parte decimal for maior ou igual a 0.5, a parte inteira aumentará em 1. Para
  números negativos vale a mesma regra, mas subtraindo 0.5. */

int arredonda (double x)
{
    if (x >= 0)
        return ((int) (x + 0.5));
    return ((int) (x - 0.5));
}
```

2.

```
/* Basta subtrair de x a parte inteira de x. Para obter a parte inteira de x,
   basta converter o valor para int. */

double casasDecimais (double x)
{
    return (x - (int) x);
}
```

3.

```
/* Repare que o "miolo" da solução é um algoritmo que já apareceu anteriormente,
   com uma "janela deslizante". */

int proxFibonacci (int n)
{
    int anterior2 = 0;
    int anterior = 1;
    int atual = anterior2 + anterior;

    if (n == 0) // Só para o caso especial do zero.
        return 0;

    while (atual < n)
    {
        anterior2 = anterior;
        anterior = atual;
        atual = anterior + anterior2;
    }

    return (atual);
}
```

4.

```
/* Exercício clássico. Começamos com um total igual a 1 e vamos multiplicando
este total pela base várias vezes. */

unsigned long long potencia (unsigned int base, unsigned int expoente)
{
    unsigned long long total = 1;
    int i;

    for (i = 0; i < expoente; i++)
        total *= base;

    return (total);
}
```

5.

```
/* A ideia aqui é ir iterativamente extraindo do número original o dígito menos
 significativo. Cada dígito extraído é colocado em uma variável, que é
 multiplicada por 10 a cada iteração. Repare que as manipulações em si são
 extremamente simples, elas só precisam ser usadas de forma "esperta". */

unsigned int inverteNum (unsigned int n)
{
    int invertido = 0;

    while (n > 0)
    {
        invertido = invertido*10 + n%10;
        n /= 10;
    }

    return (invertido);
}
```

6.

```
/* O código da função fica bem simples se você lembrar que podemos fazer
  comparações do tipo >= e == com os caracteres. Isso é possível porque um char
  na verdade é um número inteiro, que aponta para uma posição na tabela ASCII, e
  os caracteres e dígitos estão convenientemente localizados em blocos
  contíguos. */

int testaTipoChar (char c)
{
    if (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U')
        return (1);

    if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
        return (2);

    if (c > 'A' && c <= 'Z')
        return (3);

    if (c > 'a' && c <= 'z')
        return (4);

    if (c >= '0' && c <= '9')
        return (5);

    return  (0);
}
```