1.

```
/* Este exercício é um pouco mais complexo. Além do contador de iterações,
precisamos de uma variável que armazena a soma de todos os valores. Este tipo de
variável é muito comum, e tem até um nome: "acumulador". Além disso, precisamos
de um segundo contador, para saber quantos valores foram acumulados. */

#include <stdio.h>

#define N_ENTRADAS 10
#define MAX_VAL 20

int main ()
{
    int n, i, acum, n_acum;

    // Com while.
    acum = 0;
    n_acum = 0;
    i = 0;
    while (i < N_ENTRADAS)
    {
        scanf ("%d", &n);
        if (n < MAX_VAL) // Acumula.
        {
            acum += n;
            n_acum++;
        }
        i++;
    }

    printf ("%.2f\n", (float) acum / n_acum);

    // Com for.
    acum = 0;
    n_acum = 0;
    for (i = 0; i < N_ENTRADAS; i++)
    {
        scanf ("%d", &n);
        if (n < MAX_VAL) // Acumula.
        {
            acum += n;
            n_acum++;
        }
    }

    printf ("%.2f\n", (float) acum / n_acum);

    return (0);
}
```

2.

```
#include <stdio.h>

int main ()
{
    int i;
    int d1, d2, d3, d4; // Os 4 dígitos do número.
    int d12; // O número com os 2 primeiros dígitos.
    int d34; // O número com os 2 últimos dígitos.

    for (i = 1000; i <= 9999; i++)
    {
        // Usamos divisão e resto da divisão para isolar os dígitos.
        d1 = i / 1000; // Milhar.
        d2 = (i % 1000) / 100; // Centena.
        d3 = (i % 100) / 10; // Dezena.
        d4 = i % 10; // Unidade.

        // Juntamos de novo em 2 números.
        d12 = d1*10 + d2;
        d34 = d3*10 + d4;

        /* Note que daria para guardar os dígitos em variáveis, mas ficaria bem
           mais confuso:
        d12 = (i/1000)*10 + (i%1000)/100;
        d34 = (i%100)/10*10 + (i%10);

        O que é mais estranho é que, com inteiros, x/10*10 NÃO É necessariamente
        igual a x. Nem x/1000*10 é igual a x/100. Você consegue ver o motivo? */

        // Faz o teste.
        if ((d12+d34) * (d12+d34) == i)
            printf ("%d\n", i);

        /* Se você sabe o que é a função pow: NÃO É BOM USÁ-LA AQUI. Ela
           converte o número para double, faz uma troca de contexto, e realiza a
           multiplicação com pontos flutuantes e em um loop. */
    }

    return (0);
}

/* Solução 2: dá para simplificar! */
int main()
{
    int i, aux;
    for (i = 1000; i <= 9999; i++)
    {
        /* (i / 100) separa os 2 primeiros e
           (i % 100) separa os 2 últimos algarismos */
        aux = i / 100 + i % 100;
        if(aux * aux == i)
            printf("%d \n", i);
    }

    return 0;
}
```

3.

```
/* Este exercício também envolve o uso de um acumulador. Como estamos
   trabalhando com conceitos concretos (nota, juízes), usamos nomes
   informativos para as variáveis e a macro, mas note como a estrutura é
   parecida com a do exercício anterior. A "pegadinha" aqui está na parte de
   excluir a menor e a maior nota. Como não sabemos de antemão quais notas serão
   estas, não podemos deixar de contá-las no acumulador, a não ser que
   guardássemos todas as notas primeiro e deixássemos para fazer a soma no
   final, já excluindo a maior e a menor. Em vez disso, podemos simplesmente
   incluir estas notas no acumulador e descartá-las no final. É mais simples
   e mais elegante.

   Mais um detalhe: não estamos testando aqui se as entradas estão na faixa
   válida (0 a 10). Em programas reais, isto pode ser importante, mas aqui
   estamos simplesmente supondo que as entradas estão dentro da faixa. */

#include <stdio.h>

#define N_JUIZES 6

int main ()
{
    float menor_nota = 10;
    float maior_nota = 0;
    float soma_notas = 0;
    float nota;
    int i;

    for (i = 0; i < N_JUIZES; i++)
    {
        scanf ("%f", &nota);

        // Soma todas as notas.
        soma_notas += nota;

        if (nota < menor_nota)
            menor_nota = nota;
        if (nota > maior_nota)
            maior_nota = nota;
    }

    // Desconta as notas que não valem.
    soma_notas -= menor_nota;
    soma_notas -= maior_nota;

    // A nota final é a média das notas restantes.
    printf ("Nota final: %.1f\n", soma_notas/(N_JUIZES-2));

    return (0);
}
```

4.

```
/* Neste exercício, precisamos novamente de um acumulador, mas ele só é
atualizado quando encontramos um divisor de n. */

#include <stdio.h>

int main ()
{
    int i, n, soma_div;

    scanf ("%d", &n);

    soma_div = 0;
    // Verifica se cada número entre 1 e n é um divisor de n.
    // Por simplicidade, eu fui até n, mas na verdade dava para otimizar e ir só
    // até n/2. Você consegue ver o motivo?
    for (i = 1; i < n; i++)
        if (n % i == 0)
            soma_div += i; // Acumula os divisores de n.

    if (soma_div == n)
        printf ("Eh perfeito!\n");
    else
        printf ("Nao eh perfeito!\n");

    return (0);
}
```

5.

```
/* Colocarei aqui 2 respostas. Elas diferem apenas na parte entre o teste das
  entradas e o printf, (indicada por "A ou B" no código abaixo). */

#include <stdio.h>

int main ()
{
    int h1, m1, s1, h2, m2, s2;

    scanf ("%d:%d:%d", &h1, &m1, &s1);
    scanf ("%d:%d:%d", &h2, &m2, &s2);

    // Minutos e segundos entre 0 e 60, horas positivas.
    if (s1 >= 60 || s2 >= 60 || m1 >= 60 || m2 >= 60 ||
        s1 < 0 || s2 < 0 || m1 < 0 || m2 < 0 || h1 < 0 || h2 < 0)
    {
        printf ("Entrada inválida!\n");
        return (1);
    }

    **** A ou B ****

    printf ("%d:%d:%d\n", h1, m1, s1);

    return (0);
}

**** A ****

    /* Transforma tudo em segundos, soma, e decompõe novamente, reaproveitando
      h1, m1 e s1 para o resultado. */
    s1 += (m1*60) + (h1*3600);
    s2 += (m2*60) + (h2*3600);
    s1 += s2;
    h1 = s1/3600;
    s1 -= h1*3600;
    m1 = s1/60;
    s1 -= m1*60;

**** B ****

   /* O princípio aqui é o mesmo do algoritmo da soma que aprendemos na escola:
      quando a soma em um nível é maior do que o limite, "vai um". A diferença é
      que na soma normal, temos unidades, dezenas, centenas, etc. (módulo 10),
      mas na soma de horas temos minutos e segundos (módulo 60). Como h1, m1 e
      s1 não são mais usados, vou reaproveitar estas variáveis para colocar o
      resultado. */

    s1 += s2;
    if (s1 >= 60)
    {
        s1 -= 60;
        m1++;
    }

    m1 += m2;
    if (m1 >= 60)
    {
        m1 -= 60;
        h1++;
    }

    h1 += h2;
```

6.

```
#include <stdio.h>

#define N_TERMOS 10000

int main ()
{
    int i;
    double pi;
    double denom; // O denominador que sobe de 2 em 2.
    double sinal; // O sinal fica variando entre +1 e -1.

    // Inicializa tudo!
    pi = 0;
    denom = 1;
    sinal = 1;

    // Vamos usar um loop. Cada iteração coloca um termo a mais na solução.
    for (i = 0; i < N_TERMOS; i++)
    {
        /* Por clareza, usamos uma variável para o denominador, mas poderíamos
           obter o denominador diretamente de i. Você consegue pensar como? */
        pi += (sinal * 1.0/denom);

        sinal = -sinal; // Inverte o sinal.
        denom += 2; // Denominador sobe de 2 em 2.
    }

    // Precisa lembrar de multiplicar por 4 no final!
    pi *= 4;
    printf ("%0.10f", pi);
    return (0);
}
```

7.

a)

```
#include <stdio.h>

int main ()
{
    int i;
    float sinal = 1;
    float somatorio1 = 0;
    float somatorio2 = 0;

    for (i=1; i <= 1000000; i++)
    {
        somatorio1 += sinal * 1/i;
        sinal = -sinal;
    }

    sinal = -1;
    for (i=1000000; i >= 1; i--)
    {
        somatorio2 += sinal * 1/i;
        sinal = -sinal;
    }

    printf ("%.20f\n", somatorio1);
    printf ("%.20f\n", somatorio2);

    return (0);
}
```

b) A precisão do ponto flutuante. O importante aqui é ressaltar que a precisão não é infinita, e até mesmo a ordem das operações pode influir no resultado final.

c) Teremos uma precisão maior (i.e. os resultados ficarão mais próximos).