1.

```
#include <stdio.h>

int main ()
{
    int n, n_gerados = 1;

    /* Lê o primeiro valor. */
    scanf ("%d", &n);

    while (n != 1)
    {
        if (n % 2)
            n = 3*n+1;
        else
            n /= 2;

        printf ("%d\n", n);
        n_gerados++;
    }

    printf ("Sequencia com %d elementos.\n", n_gerados);

    return (0);
}
```

2.

```
#include <stdio.h>

#define META 1000000 // A meta a ser atingida.

int main ()
{
    int i;
    int n, // Número de dias.
        acessos, // Acessos em um dia.
        total = 0, // Número total de acessos. Começa em 0.
        dias_meta = -1, // Dia no qual a meta foi batida. Já iniciei com o erro.
        atingiu_meta; // Flag que diz se a meta foi batida.

    /* Inicializamos a flag. Note que não precisamos REALMENTE dela, dava para
       usar, por exemplo, dias_meta = 0 para indicar a mesma coisa. Deixei
       separado para ficar mais claro. */
    atingiu_meta = 0;

    // Lê o número de dias.
    scanf ("%d", &n);

    // Lê o número de acessos dia a dia.
    for (i = 0; i < n; i++)
    {
        scanf ("%d", &acessos);
        total += acessos;

        // Se bateu a meta AGORA (e não antes!)...
        if (!atingiu_meta && total >= META)
        {
            atingiu_meta = 1;
            // Como contamos a partir de 0, i é 1 a menos que o número de dias.
            // Daria para ter feito começando o loop em 1 e seguindo com <= n.
            dias_meta = i+1;
        }
    }

    printf ("%d\n", dias_meta);

    return (0);
}
```

3.

```
/* A chave para resolver este problema é guardar sempre 3 valores: os 2
 anteriores e o atual, que é igual à soma dos outros 2. A cada iteração, os
 valores são “deslizados”, como em uma “janela deslizante”. */

#include <stdio.h>

int main ()
{
    int i, n_termos;
    int anterior2 = 0;
    int anterior = 1;
    int atual = anterior2 + anterior;

    scanf ("%d", &n_termos);

    printf ("%d\n%d\n%d\n", anterior2, anterior, atual);

    for (i = 3; i < n_termos; i++)
    {
        anterior2 = anterior;
        anterior = atual;
        atual = anterior + anterior2;
        printf ("%d\n", atual);
    }

    return (0);
}
```

4.

```
#include <stdio.h>

int main () {
    int i;
    int n, // Número de valores.
        tem_picos = 0, // Flag que diz se achamos 2 ou mais picos.
        ant, meio, prox; // 3 alturas.

    scanf ("%d", &n);

    // Como precisamos sempre de 3 valores, lemos os 2 primeiros aqui fora!
    scanf ("%d", &ant);
    scanf ("%d", &meio);

    // Lê o número de acessos dia a dia. Note que já lemos 2 valores.
    for (i = 2; i < n; i++)
    {
        scanf ("%d", &prox);

        /* O que importa aqui é a definição da especificação: "se houver três
           números consecutivos na sequência, tal que o número do meio é menor
         do que os outros dois números, a montanha tem mais de um pico." */
        if (meio < ant && meio < prox)
            tem_picos = 1;

        // "Desliza" os valores.
        ant = meio;
        meio = prox;
    }

    if (tem_picos)
        printf ("N\n");
    else
        printf ("S\n");

    return (0);
}
```

5.

```
#include <stdio.h>

#define MIN_TERMO 0.2

int main ()
{
    double total = 1, termo = 1, denom = 2;

    printf ("%f %f\n", total, termo);
    termo = 1.0/denom;
    while (termo >= MIN_TERMO)
    {
        total += termo;
        printf ("%f %f\n", termo, total);
        denom++;
        termo = 1.0/denom;
    }

    return (0);
}
```

6.

```
/* O truque aqui é reconstruir o número de trás para a frente. Para isso,
   podemos ir isolando o último dígito do número usando o resto da divisão, e ir
   "recompondo" o número aos poucos. */

#include <stdio.h>

int main()
{
    unsigned int n, n_original, reconst;

    scanf ("%u", &n);

    // Guarda o n original para comparar no final.
    n_original = n;

    // Reconstroi o número de trás para a frente.
    // Vamos tirando o dígito final de n até n chegar em zero.
    reconst = 0;

    while (n)
    {
        // Desloca reconst 1 dígito à esquerda e adiciona o último dígito de n.
        reconst = reconst * 10 + n%10;

        // Descarta o último dígito de n.
        n /= 10;
    }

    // Compara o original com o reconstruído.
    if (n_original == reconst)
        printf ("Eh palindromo!\n");
    else
        printf ("Nao eh palindromo!\n");

    return 0;
}
```

7.

```
/* Este é um problema típico de repetição no qual, em um dado momento, você já
   leu somente alguns dos valores. Você precisa rastrear algumas coisas:

   1. O último valor informado (para saber se não mudou).
   2. Quantas vezes o último valor apareceu até o momento.
   3. Qual foi a maior sequência de números iguais vista até o momento. */

#include <stdio.h>

int main ()
{
    int i;
    int n, // Tamanho da sequência completa.
        x, // Um valor.
        x_anterior, // O valor anterior.
        tam_atual, // Tamanho da sequência atual.
        tam_maior; // Tamanho da maior sequência.

    /* Inicia a maior sequência com 0 para garantir que qualquer outra vai
    passar esta. */
    tam_maior = 0;

    // Pega o tamanho da sequência completa.
    scanf ("%d", &n);

    // Lemos o primeiro valor fora do loop, para entrar com x_anterior válido.
    scanf ("%d", &x_anterior);
    tam_atual = 1; // Já foi um valor!

    // Começamos a contagem no SEGUNDO número.
    for (i = 1; i < n; i++)
    {
        scanf ("%d", &x);

        if (x == x_anterior)
            tam_atual++; // Continua crescendo.
        else // Acabou a sequência!
        {
            x_anterior = x; // Este é o valor que vai repetir agora.

            // É a maior que já vimos?
            if (tam_atual > tam_maior)
                tam_maior = tam_atual;

            tam_atual = 1; // Começa a contar de novo.
        }
    }

    /* Uma coisa chata de lembrar é que talvez a sequência completa tenha
       terminado com repetições. Precisamos então verificar uma última vez se a
       sequência final foi a maior! */
    if (tam_atual > tam_maior)
        tam_maior = tam_atual;

    printf ("%d\n", tam_maior);

    return (0);
}
```