1.

```
void segundosParaHMS (int total_segundos, int *h, int *m, int *s)
{
    *h = total_segundos / 3600;
    total_segundos -= (*h * 3600);
    *m = total_segundos / 60;
    *s = total_segundos % 60;
}


int main ()
{
    int h, m, s;
    segundosParaHMS (4000, &h, &m, &s);

    printf ("%02d:%02d:%02d\n", h, m, s);

    return (0);
}
```

---

2.

```
int main ()
{
    int n, i, eh_palindromo, head, tail;

    scanf ("%d", &n);

    // Supõe que é palíndromo, e vai removendo as pontas até o número "sumir".
    eh_palindromo = 1;
    while (n && eh_palindromo)
    {
        removeExtremos (&n, &head, &tail);
        if (head != tail)
            eh_palindromo = 0;
    }

    if (eh_palindromo)
        printf ("Eh palindromo!\n");
    else
        printf ("Nao eh palindromo!\n");

    return (0);
}
```

3.

```
int leSequenciaKAlternante (int n, int* k)
{
    int i, num;
    int tam_seq; // Tamanho da sequência atual.
    int eh_par; // Flag, indica se o segmento atual é par.

    // Lê o primeiro valor "por fora".
    // Supondo aqui que n >= 1.
    scanf ("%d", &num);
    i = 1;
    tam_seq = 1;
    *k = -1; // k < 0 -> Estamos ainda na primeira sequência.
    if (num%2)
        eh_par = 0;
    else
        eh_par = 1;

    // Lê os valores restantes.
    while (i < n)
    {
        scanf ("%d", &num);

        // Manteve a paridade?
        if ((num%2 && !eh_par) || (num%2 == 0 && eh_par))
            tam_seq++;
        else
        {
            // Acabou o segmento!
            if (*k < 0) // Era o primeiro segmento?
            {
                *k = tam_seq;

                // Para ser k-alternante, n precisa ser múltiplo de k.
                if (n%*k != 0)
                    return (0);
            }
            else if (*k != tam_seq) // O tamanho do segmento mudou?
                return (0); // Não era k-alternante!

            tam_seq = 1; // Reseta o tamanho do segmento.
            eh_par = !eh_par; // Inverte a paridade.
        }

        i++;
    }

    // Se não fechou nenhum segmento, a sequência inteira era ou par ou ímpar.
    if (*k < 0)
        *k = n;

    return (1);
}
```

4.

```
#define LED_BASE_ADDR 0x1000
#define N_COLUNAS 128

int main
{
    int deslocamento; /* Deslocamento a partir do endereço base. */

    /* Para cada coluna... */
    for (deslocamento = 0; deslocamento < N_COLUNAS; deslocamento++)
    {
        unsigned char* ptr = LED_BASE_ADDR + deslocamento;

        /* O deslocamento serve tanto para saber o endereço da coluna de LEDs
           atual quanto para saber qual LED fica aceso na coluna atual. Para
           saber qual LED fica aceso, notamos que todos os endereços com
           deslocamento múltiplo de 8 terão o 1o LED aceso, os endereços
           imediatamente à direita terão o 2o LED aceso, e assim por diante.
           Basta então deslocar 0x80 (1o LED aceso) de um número entre 0 e 7,
           obtido como o deslocamento módulo 8. */
        *ptr = 0x80 >> (deslocamento % 8);

    }
    return (0);
}
```