Introdução a funções

Aula 17

Fábio Henrique Viduani Martinez

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação I, 2012

Conteúdo da aula

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- uma grande tarefa pode ser solucionada através de sua divisão em sub-tarefas e da combinação de suas soluções parciais
- em programação essas soluções menores são chamadas de módulos de programação e fazem parte de todo processo de construção de programas para solução de problemas reais
- módulos são uma ferramenta da programação estruturada que fornecem ao(à) programador(a) um mecanismo para construir programas que são fáceis de escrever, ler, compreender, corrigir, modificar e manter

Motivação

- na linguagem C, essa característica em um código é obtida através do uso de funções
- cada função na linguagem C pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas

Motivação

- na linguagem C, essa característica em um código é obtida através do uso de funções
- cada função na linguagem C pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas

Motivação

- na linguagem C, essa característica em um código é obtida através do uso de funções
- cada função na linguagem C pode ser vista como um pequeno programa com suas próprias declarações de variáveis e sentenças de programação
- além de facilitar a compreensão e a modificação de um programa e evitar a duplicação de código usado mais que uma vez, as funções podem ser usadas não só no programa para o qual foram projetadas, mas também em outros programas

Noções iniciais

```
/* Recebe dois números reais e devolve a média aritmética deles */
double media(double a, double b)
{
   return (a + b) / 2;
}
```

```
printf("Média: %g\n", media(x, y));
m = media(x, y);
```

Noções iniciais

```
/* Recebe dois números reais e devolve a média aritmética deles */
double media(double a, double b)
{
   return (a + b) / 2;
}
```

```
printf("Média: %g\n", media(x, y));
m = media(x, y);
```

Noções iniciais

```
/* Recebe dois números reais e devolve a média aritmética deles */
double media(double a, double b)
{
   return (a + b) / 2;
}
```

```
printf("Média: %g\n", media(x, y));
m = media(x, y);
```

Noções iniciais

```c
#include <stdio.h>

/* Recebe dois números reais e devolve a média aritmética deles */
double media(double a, double b)
{
   return (a + b) / 2;
}

/* Recebe três números reais e calcula
   a média aritmética para cada par */
int main(void)
{
   double x, y, z;

   printf("Informe três valores: ");
   scanf("%lf%lf%lf", &x, &y, &z);
   printf("Média de %g e %g é %g\n", x, y, media(x, y));
   printf("Média de %g e %g é %g\n", x, z, media(x, z));
   printf("Média de %g e %g é %g\n", y, z, media(y, z));

   return 0;
}
```

Noções iniciais

```
/* Recebe um número inteiro e o imprime na saída com uma mensagem */
void imprime_contador(int n)
{
   printf("%d e contando...\n", n);

   return;
}
```

Noções iniciais

```
/* Recebe um número inteiro e o imprime na saída com uma mensagem */
void imprime_contador(int n)
{
   printf("%d e contando...\n", n);
}
```

```
imprime_contador(i);
```

Noções iniciais

```
/* Recebe um número inteiro e o imprime na saída com uma mensagem */
void imprime_contador(int n)
{
   printf("%d e contando...\n", n);
}
```

```
imprime_contador(i);
```

Noções iniciais

```c
#include <stdio.h>

/* Recebe um número inteiro e o imprime na saída com uma mensagem */
void imprime_contador(int n)
{
   printf("%d e contando...\n", n);
}


/* Imprime um contador inteiro descrescente de 10 a 1 */
int main(void)
{
   int i;

   for (i = 10; i > 0; i--)
      imprime_contador(i);

   return 0;
}
```

Noções iniciais

```
/* Imprime uma mensagem na saída */
void imprime_msg(void)
{
   printf("Programar é bacana!\n");
}
```

```
imprime_msg();
```

Noções iniciais

```
/* Imprime uma mensagem na saída */
void imprime_msg(void)
{
   printf("Programar é bacana!\n");
}
```

```
imprime_msg();
```

Noções iniciais

```
#include <stdio.h>

/* Imprime uma mensagem na saída */
void imprime_msg(void)
{
   printf("Programar é bacana!\n");
}

/* Imprime 10 vezes uma mensagem na saída */
int main(void)
{
   int i;

   for (i = 1; i <= 10; i++)
      imprime_msg();

   return 0;
}
```

Definição e chamada de funções

- uma **função** da linguagem C é um trecho de código que pode receber um ou mais valores de entrada armazenados em variáveis, chamados de **parâmetros (de entrada)**, que realiza algum processamento específico e que pode devolver um único valor de saída
- construímos funções para resolver pequenos problemas bem específicos e, em conjunto com outras funções, resolver problemas maiores e mais complexos

Definição e chamada de funções

- uma **função** da linguagem C é um trecho de código que pode receber um ou mais valores de entrada armazenados em variáveis, chamados de **parâmetros (de entrada)**, que realiza algum processamento específico e que pode devolver um único valor de saída
- construímos funções para resolver pequenos problemas bem específicos e, em conjunto com outras funções, resolver problemas maiores e mais complexos

Definição e chamada de funções

A definição de uma função na linguagem C fornece quatro informações importantes ao compilador:

- quem pode chamá-la
- o tipo do valor que a função devolve
- seu identificador
- seus parâmetros de entrada

Definição e chamada de funções

Toda função deve necessariamente possuir:

- uma **interface**, que faz a comunicação entre a função e o meio exterior; a interface é definida pela primeira linha da função, onde especificamos o tipo do valor devolvido da função, seu identificador e a lista de parâmetros de entrada separados por vírgulas e envolvidos por um par de parênteses; e
- um **corpo**, que realiza o processamento sobre os valores armazenados nos parâmetros de entrada, e também nas variáveis declaradas internamente à função, e devolve um valor na saída; o corpo de uma função é composto pela declaração de variáveis locais da função e sua lista de comandos.

Definição e chamada de funções

Forma geral de uma **definição de uma função**:

```
tipo identificador(parâmetros)
{
    declaração de variáveis

    sentença_1;
    .
    .
    .
    sentença_n;
}
```

Definição e chamada de funções

- a declaração de variáveis de uma função deve vir primeiro, antes de qualquer sentença do corpo da função
- as variáveis declaradas no corpo de uma função pertencem exclusivamente àquela função e não podem ser examinadas ou modificadas por outras funções

Definição e chamada de funções

- a **chamada de função** consiste de um identificador da função seguido por uma lista de argumentos entre parênteses:

```
media(x, y)
imprime_contador(i)
imprime_msg()
```

Definição e chamada de funções

- uma chamada de uma função com valor devolvido `void` é sempre seguida por um `;` para torná-la uma sentença:

```
imprime_contador(i);
imprime_msg();
```

- uma chamada de uma função com valor devolvido diferente de `void` produz um valor que pode ser armazenado em uma variável ou usado em uma expressão aritmética, relacional ou lógica:

```
if (media(x, y) > 0)
   printf("Média é positiva\n");
printf("A média é %g\n", media(x, y));
```

Definição e chamada de funções

- uma chamada de uma função com valor devolvido `void` é sempre seguida por um `;` para torná-la uma sentença:

```
imprime_contador(i);
imprime_msg();
```

- uma chamada de uma função com valor devolvido diferente de `void` produz um valor que pode ser armazenado em uma variável ou usado em uma expressão aritmética, relacional ou lógica:

```
if (media(x, y) > 0)
   printf("Média é positiva\n");
printf("A média é %g\n", media(x, y));
```

Definição e chamada de funções

- uma função que devolve um valor diferente de **`void`** deve usar a sentença **`return`** para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão;
```

- a *`expressão`* em geral é uma constante ou uma variável:

```
return 0;

ou

return estado;
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2;

ou

return a * a * a + 2 * a * a - 3 * a - 1;
```

Definição e chamada de funções

- uma função que devolve um valor diferente de **`void`** deve usar a sentença **`return`** para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão;
```

- a ***`expressão`*** em geral é uma constante ou uma variável:

```
return 0;

ou

return estado;
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2;

ou

return a * a * a + 2 * a * a - 3 * a - 1;
```

Definição e chamada de funções

- uma função que devolve um valor diferente de **`void`** deve usar a sentença **`return`** para especificar qual valor será devolvido, com a seguinte forma geral:

```
return expressão;
```

- a ***`expressão`*** em geral é uma constante ou uma variável:

```
return 0;

ou

return estado;
```

- expressões mais complexas são possíveis:

```
return (a + b) / 2;

ou

return a * a * a + 2 * a * a - 3 * a - 1;
```

Exemplo

```
/* Recebe dois números inteiros e devolve
   o máximo divisor comum entre eles */
int mdc(int a, int b)
{
   int aux;

   while (b != 0) {
      aux = a % b;
      a = b;
      b = aux;
   }

   return a;
}
```

Exemplo

```
#include <stdio.h>
int mdc(int a, int b)
{
   int aux;

   while (b != 0) {
      aux = a % b;
      a = b;
      b = aux;
   }
   return a;
}

int main(void)
{
   int x, y;

   printf("Informe dois valores: ");
   scanf("%d%d", &x, &y);
   printf("O mdc entre %d e %d é %d\n", x, y, mdc(x, y));
   return 0;
}
```

Declaração de funções

```c
#include <stdio.h>

double media(double a, double b);       /* declaração */

int main(void)
{
   double x, y, z;

   printf("Informe três valores: ");
   scanf("%lf%lf%lf", &x, &y, &z);

   printf("Média de %g e %g é %g\n", x, y, media(x, y));
   printf("Média de %g e %g é %g\n", x, z, media(x, z));
   printf("Média de %g e %g é %g\n", y, z, media(y, z));

   return 0;
}

double media(double a, double b)        /* definição */
{
   return (a + b) / 2;
}
```

Exercícios

1. (a) Escreva uma função com a seguinte interface:

```
double area_triangulo(double base, double altura)
```

que receba dois números de ponto flutuante que representam a base e a altura de um triângulo e compute e devolva a área desse triângulo.

(b) Escreva um programa que receba uma sequência de n pares de números de ponto flutuante, onde cada par representa a base e a altura de um triângulo, e calcule e escreva, para cada par, a área do triângulo correspondente. Use a função descrita no item (a).

Exercícios

```c
#include <stdio.h>

double area_triangulo(double base, double altura)
{
   return (base * altura) / 2;
}

int main(void)
{
   int i, n;
   double b, a;

   printf("Informe n: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
      printf("Informe a base e a altura do triângulo: ");
      scanf("%lf%lf", &b, &a);
      printf("Área do triângulo: %g\n", area_triangulo(b, a));
   }

   return 0;
}
```

Exercícios

2. (a) Escreva uma função com a seguinte interface:

```
int mult(int a, int b)
```

que receba dois números inteiros positivos a e b e determine e devolva um valor que representa o produto desses números, usando o seguinte método de multiplicação:

- dividir, sucessivamente, o primeiro número por 2, até que se obtenha 1 como quociente;
- em paralelo, dobrar, sucessivamente, o segundo número;
- somar os números da segunda coluna que tenham um número ímpar na primeira coluna; o total obtido é o produto procurado.

Por exemplo, para os números 9 e 6, temos que 9×6 é

9	6	$\rightarrow$	6
4	12		
2	24		
1	48	$\rightarrow$	48
			54

(b) Escreva um programa que leia $n \geqslant 1$ pares de números e calcule os respectivos produtos desses pares, usando a função do item (a).

Exercícios

3. Para determinar o número de lâmpadas necessárias para cada aposento de uma residência, existem normas que fornecem o mínimo de potência de iluminação exigida por metro quadrado (m^2) conforme o uso desse ambiente. Suponha que só temos lâmpadas de 60 watts para uso.
   Seja a seguinte tabela de informações sobre possíveis aposentos de uma residência:

Utilização	Classe	Potência/m^2 (W)
quarto	1	15
sala de TV	1	15
salas	2	18
cozinha	2	18
varandas	2	18
escritório	3	20
banheiro	3	20

Exercícios

3. (continuação)
   (a) Escreva uma função com a seguinte interface:

```
int num_lampadas(int classe, double a, double b)
```

que receba um número inteiro representando a classe de iluminação de um aposento e dois números com ponto flutuante representando suas duas dimensões e devolva um número inteiro representando o número de lâmpadas necessárias para iluminar adequadamente o aposento.

   (b) Escreva um programa que receba uma sequência de informações contendo o nome do aposento da residência, sua classe de iluminação e as suas dimensões e, usando a função `num_lampadas`, imprima a área de cada aposento e o número total de lâmpadas necessárias para o aposento. Além disso, seu programa deve calcular o total de lâmpadas necessárias e a potência total necessária para a residência toda.
Suponha que o término se dá quando o nome do aposento informado é `fim`.

Exercícios

4. (a) Escreva uma função com a seguinte interface:

```
int mdc(int a, int b)
```

que receba dois números inteiros positivos a e b e calcule e devolva o máximo divisor comum entre eles.

(b) Usando a função do item anterior, escreva um programa que receba $n \geqslant 1$ números inteiros positivos e calcule o máximo divisor comum entre todos eles.

Exercícios

5. Dois números inteiros são **primos entre si** quando não existe um divisor maior do que 1 que divida ambos. Isso significa que o máximo divisor comum de dois números que são primos entre si é igual a 1. Por exemplo, 4 e 9 são primos entre si já que o maior divisor comum entre eles é 1. Por outro lado, 5 e 20 não são primos entre si.

   (a) Escreva uma função com a seguinte interface:

   ```
   int mdc(int a, int b)
   ```

   que receba dois números inteiros a e b e devolva o máximo divisor comum entre eles, usando o algoritmo de Euclides.

   (b) Escreva um programa que receba diversas sequências de números inteiros e conte, para cada sequência, a quantidade de pares de números que são primos entre si. Cada sequência contém um primeiro número inteiro n, tal que $0 \leqslant n \leqslant 100$, e uma lista de n números inteiros. A última lista contém $n = 0$.

Exercícios

6. (a) Escreva uma função com a seguinte interface:

```
int verifica_primo(int p)
```

que receba um número inteiro positivo p e verifique se p é primo, devolvendo 1 em caso positivo e 0 em caso negativo.

(b) Usando a função do item anterior, escreva um programa que receba $n \geqslant 1$ números inteiros positivos e calcule a soma dos que são primos.

Exercícios

7. Um número inteiro a é dito ser **permutação** de um número inteiro b se os dígitos de a formam uma permutação dos dígitos de b. Exemplo: 5412434 é uma permutação de 4321445, mas não é uma permutação de 4312455.
Observação: considere que o dígito 0 (zero) não ocorre nos números.

   (a) Escreva uma função com a seguinte interface:

   ```
   int conta_digitos(int n, int d)
   ```

   que receba dois números inteiros n e d, com $0 < d \leqslant 9$, devolva um valor que representa o número de vezes que o dígito d ocorre no número n.

   (b) Usando a função do item anterior, escreva um programa que leia dois números inteiros positivos a e b e responda se a é permutação de b.

Exercícios

8. (a) Escreva uma função com a seguinte interface:

```
int sufixo(int a, int b)
```

que receba dois números inteiros a e b e verifique se b é um sufixo de a. Em caso positivo, a função deve devolver 1; caso contrário, a função deve devolver 0.
Exemplo:

a	b		
567890	890	$\rightarrow$	sufixo
1234	1234	$\rightarrow$	sufixo
2457	245	$\rightarrow$	não é sufixo
457	2457	$\rightarrow$	não é sufixo

Exercícios

8. (continuação)

   (b) Usando a função do item anterior, escreva um programa que leia dois números inteiros a e b e verifique se o menor deles é subsequência do outro.
   Exemplo:

a	b		
567890	678	$\rightarrow$	b é subsequência de a
1234	2212345	$\rightarrow$	a é subsequência de b
235	236	$\rightarrow$	um não é subsequência do outro

Exercícios

9. Considere o seguinte processo para gerar uma sequência de números. Comece com um número inteiro positivo n. Se n é par, divida por 2. Se n é ímpar, multiplique por 3 e some 1. Repita esse processo com o novo valor de n, terminando quando $n = 1$. Por exemplo, a sequência de números a seguir é gerada para $n = 22$:

$$22 \;\; 11 \;\; 34 \;\; 17 \;\; 52 \;\; 26 \;\; 13 \;\; 40 \;\; 20 \;\; 10 \;\; 5 \;\; 16 \;\; 8 \;\; 4 \;\; 2 \;\; 1$$

É conjecturado que este processo termina com $n = 1$ para todo inteiro $n > 0$. Os números gerados nessa sequência são chamados de **ciclo de n**. Ademais, para um número n, o **comprimento do ciclo de n** é o número de elementos gerados na sequência. No exemplo acima, o comprimento do ciclo de 22 é 16.

Exercícios

9. (continuação)
   (a) Escreva uma função com a seguinte interface:

```
int comprimento(int n)
```

   que receba um número inteiro $n \geqslant 1$, imprima o ciclo de n e devolva o comprimento do ciclo de n.
   (b) Escreva um programa que leia um número inteiro $k > 0$ e uma sequência de k números inteiros positivos e, para cada um deles, mostre seu ciclo e seu comprimento do ciclo. Use a função do item (a).

Exercícios

10. O **piso** de um número x é o único inteiro i tal que $i \leqslant x < i + 1$. O piso de x é denotado por $\lfloor x \rfloor$. Aplicando o piso na função $\log_2$, obtemos a seguinte amostra de valores:

n	15	16	31	32	63	64	127	128	255	256
$\lfloor \log_2 n \rfloor$	3	4	4	5	5	6	6	7	7	8

   (a) Escreva uma função com a seguinte interface:

```
int piso_log2(int n)
```

   que receba um número inteiro $n \geqslant 1$ e devolva $\lfloor \log_2 n \rfloor$.

   (b) Escreva um programa que receba um número inteiro $k > 0$ e uma sequência de k números inteiros positivos e, para cada número n fornecido, calcule e imprima $\lfloor \log_2 n \rfloor$.

Exercícios

11. Podemos calcular a o valor da integral da função cosseno no intervalo (a, b) usando o método dos trapézios, descrito a seguir:

$$\int_a^b \cos(x) = \frac{h}{2}\Big(\cos(a) + 2\cos(a+h) + 2\cos(a+2h) + \\ \ldots + 2\cos(a+(n-1)h) + \cos(b) \Big),$$

onde

- a e b definem o intervalo de integração, com $-\pi/2 \leqslant a < b \leqslant \pi/2$;
- n é o número de sub-intervalos, com $n \geqslant 1$;
- h é uma constante, determinada por $h = (b-a)/n$.

Exercícios

11. (continuação)

(a) Escreva uma função com a seguinte interface:

```
double cosseno(double x, double epsilon)
```

que receba um número real x que representa um arco em radianos e um número real ε que representa a precisão desejada, onde $0 < \varepsilon < 1$, e use a seguinte série para calcular o cosseno de x:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} - \ldots + (-1)^k \frac{x^{2k}}{(2k)!} + \ldots$$

A função deve devolver a aproximação do valor do cosseno de x até que o termo $x^{2k}/(2k)!$ seja menor que ε.

Exercícios

11. (continuação)

    (b) Escreva um programa que que receba um número inteiro $k > 0$ que indica o número de casos de teste. Para cada caso de teste, receba dois números reais a e b que definem o intervalo de integração (a, b), onde $-\pi/2 \leqslant a < b \leqslant \pi/2$, um número inteiro n que representa o número de sub-intervalos e um número real ε que determina a precisão do cálculo da função cosseno, onde $0 < \varepsilon < 1$, integre numericamente a função cosseno sobre o intervalo (a, b) usando n trapézios, e imprima o valor da integral da função cosseno no intervalo (a, b) com precisão de 8 casas decimais. Use a função do item (a).

Exercícios

12. Um número inteiro positivo é chamado de **palíndromo** se lido da esquerda para a direita e da direita para a esquerda representa o mesmo número.
A função *inverta e adicione* é curiosa e divertida. Iniciamos com um número inteiro positivo, invertemos seus dígitos e adicionamos o número invertido ao original. Se o resultado da adição não é um palíndromo, repetimos esse processo até obtermos um número palíndromo.
Por exemplo, se começamos com o número 195, obtemos o número 9339 como o palíndromo resultante desse processo após 4 adições:

$$\begin{array}{r} 195 \\ +\ 591 \\ \hline 786 \end{array} \qquad \begin{array}{r} 786 \\ +\ 687 \\ \hline 1473 \end{array} \qquad \begin{array}{r} 1473 \\ +\ 3741 \\ \hline 5214 \end{array} \qquad \begin{array}{r} 5214 \\ +\ 4125 \\ \hline 9339 \end{array}$$

Exercícios

12. (continuação)
Este processo leva a palíndromos em poucos passos para quase todos os números inteiros positivos. Mas existem exceções interessantes. O número 196 é o primeiro número para o qual nenhum palíndromo foi encontrado por este processo. Entretanto, nunca foi provado que tal palíndromo não existe.

    (a) Escreva uma função com a seguinte interface:

    ```
    unsigned int inverte(unsigned int n)
    ```

    que receba um número inteiro positivo n e devolva esse número invertido.

Exercícios

12. (continuação)

    (b) Escreva um programa que receba um número inteiro $k > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número inteiro $n > 0$ e compute um número inteiro p através da função *inverta a adicione* descrita acima, mostrando na saída o número mínimo de adições para encontrar o palíndromo e também o próprio palíndromo. Caso um número inteiro de entrada use mais que 1.000 iterações/adições ou que o número gerado pelo processo ultrapasse o valor 4.294.967.295, escreva `? ?` na saída.

Exercícios

13. Uma sequência de n números inteiros é dita **k-piramidal** se pode ser subdividida em k blocos de comprimentos crescentes de números iguais: o primeiro bloco tem um número inteiro, o segundo bloco tem dois números inteiros iguais, o terceiro bloco tem três números inteiros iguais, e assim por diante, até o último bloco, que é o k-ésimo, que deve conter k números inteiros iguais. Exemplos:
    - A sequência $\underline{-7}\ \underline{2\ 2}\ \underline{0\ 0\ 0}\ \underline{1\ 1\ 1\ 1}$ é 4-piramidal;
    - A sequência $\underline{1}\ \underline{5\ 5}$ é 2-piramidal;
    - A sequência $\underline{8}\ \underline{8\ 8}\ \underline{8\ 8\ 8}$ é 3-piramidal;
    - A sequência $2\ 9\ 9\ 3\ 3\ 3\ 6\ 6$ não é piramidal.

Exercícios

13. (continuação)

(a) Escreva uma função com a seguinte interface:

```
int bloco(int m)
```

que receba um número inteiro $m > 0$, leia uma sequência de m números inteiros e devolva 1 se todos os números lidos são iguais; caso contrário, a função deve devolver 0.

(b) Escreva um programa que receba um número inteiro $t > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número inteiro $n > 0$ e uma sequência n números inteiros, e verifique se a sequência é k-piramidal para algum k. Em caso positivo, imprima o valor de k. Caso contrário, imprima `-1`.

Exercícios

14. Uma sequência de n números inteiros não nulos é dita **m-alternante** se é constituída por m segmentos: o primeiro com um elemento, o segundo com dois elementos e assim por diante até o m-ésimo, com m elementos. Além disso, os elementos de um mesmo segmento devem ser todos pares ou todos ímpares e para cada segmento, se seus elementos forem todos pares (ímpares), os elementos do segmento seguinte devem ser todos ímpares (pares).
Por exemplo:
    - A sequência com $n = 10$ elementos: $\underline{8}\ \underline{3\ 7}\ \underline{2\ 10\ 4}\ \underline{5\ 13\ 9\ 11}$ é 4-alternante.
    - A sequência com $n = 3$ elementos: $\underline{7}\ \underline{2\ 8}$ é 2-alternante.
    - A sequência com $n = 8$ elementos: $\underline{1}\ \underline{12\ 4}\ \underline{3\ 13\ 5}\ \underline{8\ 6}$ não é alternante, pois o último segmento não tem tamanho 4.

Exercícios

14. (continuação)

(a) Escreva uma função com a seguinte interface:

```
int bloco(int m)
```

que receba como parâmetro um inteiro m e leia m números inteiros, devolvendo um dos seguintes valores:

0, se os m números lidos forem pares;
1, se os m números lidos forem ímpares;
-1, se entre os m números lidos há números com paridades diferentes

(b) Usando a função do item anterior, escreva um programa que, dados um inteiro n, com $n \geqslant 1$, e uma sequência de n números inteiros, verifica se a sequência é m-alternante. O programa deve imprimir o valor de m ou exibir uma mensagem informando que a sequência não é alternante.