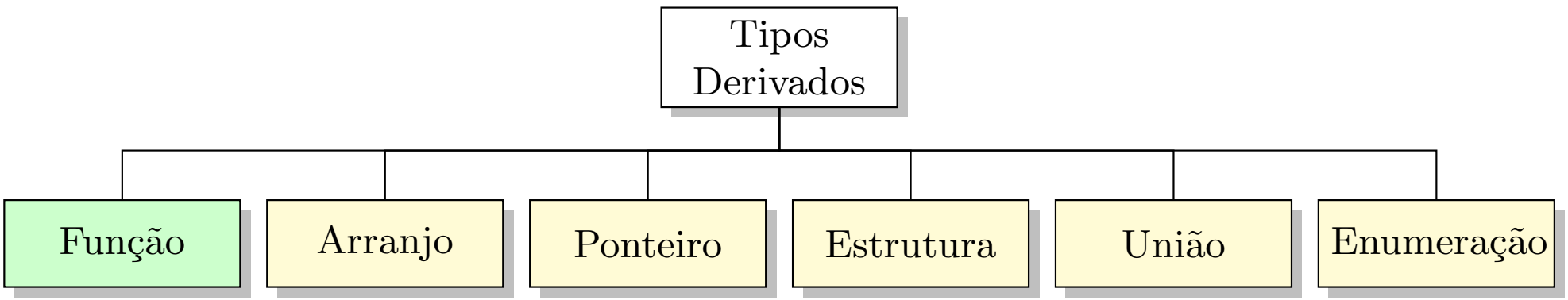
Arranjos

Alexsandro Santos Soares
prof.asoares@gmail.com

Universidade Federal de Uberlândia
Faculdade de Computação

Introdução

- Hoje estudaremos o segundo tipo derivado em C, o arranjo.
- A figura abaixo recapitula os seis tipos derivados.
- O tipo função já foi estudado.



Conceitos

- Imagine um problema no qual se tenha que ler, processar e imprimir 10 inteiros.
- Deve-se também manter todos os inteiros na memória até o final do programa.
- Como proceder?

Conceitos

- Imagine um problema no qual se tenha que ler, processar e imprimir 10 inteiros.
- Deve-se também manter todos os inteiros na memória até o final do programa.
- Como proceder?
  1. Podemos declarar 10 variáveis com nomes diferentes.

Conceitos

- Imagine um problema no qual se tenha que ler, processar e imprimir 10 inteiros.
- Deve-se também manter todos os inteiros na memória até o final do programa.
- Como proceder?
  1. Podemos declarar 10 variáveis com nomes diferentes.
     - Teremos 10 instruções de leitura, uma para cada variável, mais de 10 de impressão, etc.
     - E se tivermos 100, 1000 ou 10 000 elementos?

Conceitos

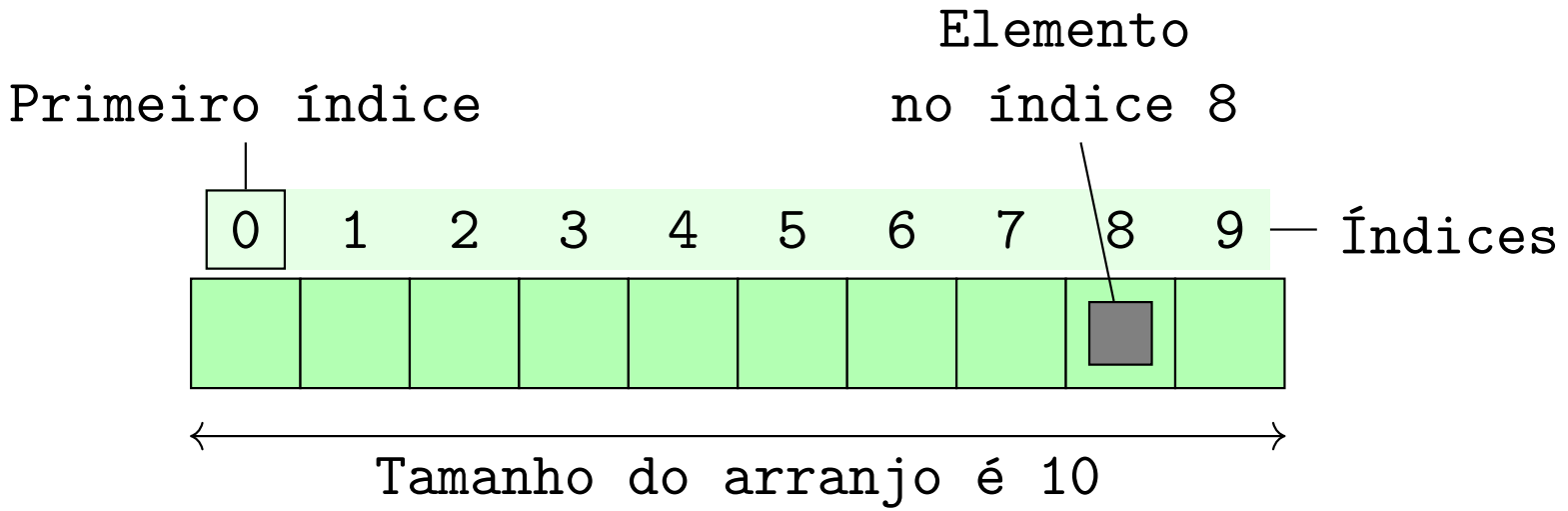
- Imagine um problema no qual se tenha que ler, processar e imprimir 10 inteiros.
- Deve-se também manter todos os inteiros na memória até o final do programa.
- Como proceder?
  1. Podemos declarar 10 variáveis com nomes diferentes.
     - Teremos 10 instruções de leitura, uma para cada variável, mais de 10 de impressão, etc.
     - E se tivermos 100, 1000 ou 10 000 elementos?
  2. Para processar grandes quantidades de dados precisamos de uma estrutura de dados chamada *arranjo*.

Conceitos

- Imagine um problema no qual se tenha que ler, processar e imprimir 10 inteiros.
- Deve-se também manter todos os inteiros na memória até o final do programa.
- Como proceder?
  1. Podemos declarar 10 variáveis com nomes diferentes.
     - Teremos 10 instruções de leitura, uma para cada variável, mais de 10 de impressão, etc.
     - E se tivermos 100, 1000 ou 10 000 elementos?
  2. Para processar grandes quantidades de dados precisamos de uma estrutura de dados chamada *arranjo*.
- Um arranjo é uma coleção de elementos de dados do mesmo tipo.

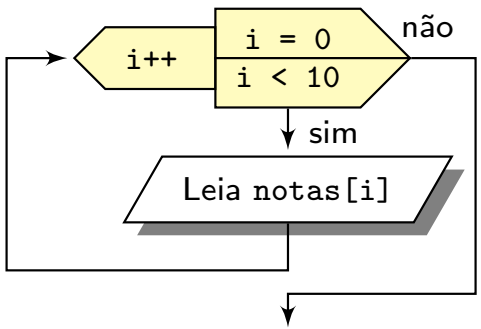
Arranjo

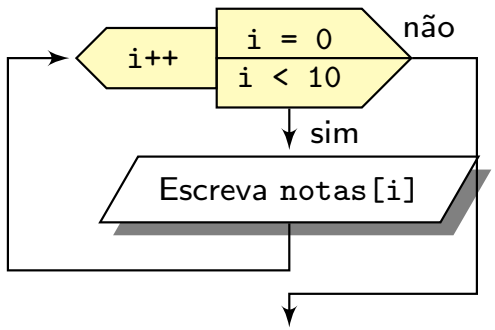
- Como um arranjo é uma coleção sequenciada podemos nos referir aos elementos dele como o primeiro elemento, o segundo elemento, e assim por diante até chegarmos ao último elemento.
- Em C, o primeiro elemento será colocado no endereço 0 do arranjo. O segundo está no endereço 1 e, continuando a sequência para o exemplo, o último elemento estará no endereço 9.
- Se chamarmos o arranjo de `notas`, então o primeiro elemento será referenciado por `notas[0]`, o segundo por `notas[1]` e o décimo por `notas[9]`.



Indexação

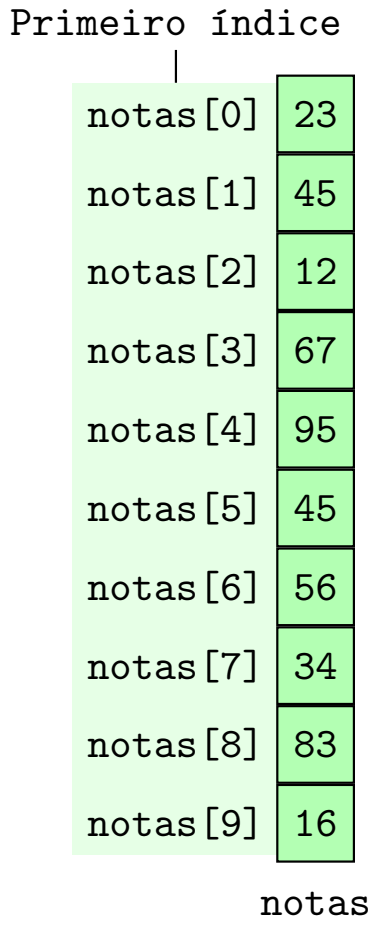
- O número entre colchetes é chamado de índice do arranjo e com ele podemos realizar quaisquer operações com um determinado elemento do arranjo.
- Por exemplo, para ler os 10 valores podemos usar o fluxograma da esquerda e para escrever, o da direita.





O arranjo notas

Abaixo está um arranjo típico em C, chamado **notas** de tamanho 10, juntamente com seus valores.



Usando arranjos em C

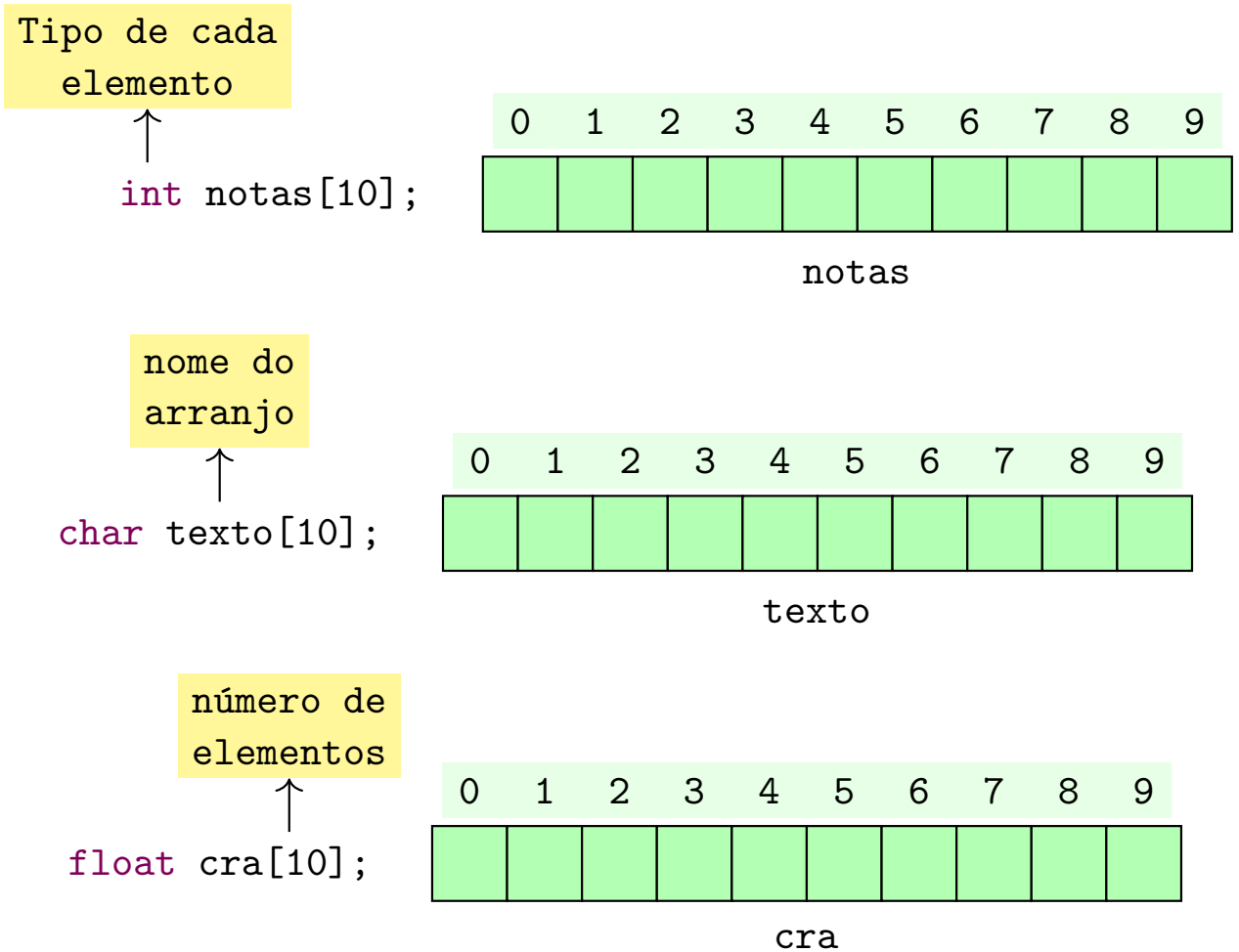
- C11 oferece dois tipos diferentes de arranjos: tamanho fixo e de tamanho variável.
- No arranjo de tamanho fixo o tamanho do arranjo é já conhecido durante a compilação do programa.
- No arranjo de tamanho variável o tamanho do arranjo somente será conhecido quando o programa for executado.

Declaração e definição

Antes de usar um arranjo deve-se declará-lo com o formato abaixo

```
tipo nomeDoArranjo[tamanho]
```

Em um vetor de tamanho fixo o tamanho é uma constante:



Declaração e definição

- O formato de declaração de um arranjo de tamanho variável é o mesmo, exceto que o tamanho é uma variável.
- Quando o programa for executado o tamanho será determinado e o arranjo estará definido.
- Uma vez definido, o tamanho não poderá ser alterado.
- No exemplo a seguir o tamanho do arranjo deve ser declarado e inicializado *antes* que ele seja usado na definição do arranjo de tamanho variável.

```
float vendas[tamanho];
```

Acessando elementos no arranjo

- C usa um índice para acessar um elemento individual do arranjo.
- O índice deve ser um valor inteiro ou uma expressão que produza um inteiro.
- A forma mais simples é usar uma constante numérica:

```
notas[0]
```

- No exemplo a seguir o índice é o valor de uma variável e, para processar os elementos em **notas**, um laço similar ao seguinte será usado:

```
for (i=0; i < 10; i++)
   processa(notas[i]);
```

Calculando endereços no arranjo

- Como C11 sabe onde os elementos individuais do arranjo estão localizados na memória?
  - No arranjo `notas`,por exemplo, existem dez elementos. Como encontrar um deles?
- Em C, o nome do arranjo é atalho para o endereço do primeiro byte do arranjo.
- O índice representa um deslocamento do início do arranjo para o elemento sendo referenciado.
- Com estas duas partes de dados, C pode calcular o endereço de qualquer elemento no arranjo usando a fórmula

```
endereço do elemento = endereço do arranjo +
                       (sizeof (elemento) * índice)
```

- Como exemplo, suponha que `notas` tenha sido armazenado no endereço 10 000. Se cada elemento inteiro de `notas` gastar 4 bytes para ser armazenado, então o endereço do elemento no índice 3 é

```
endereço do elemento = 10 000 + 4 * 3 = 10 012
```

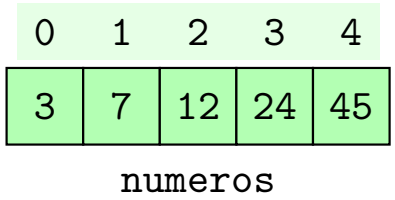
Armazenando valores nos arranjos

- A declaração e definição somente servem para reservar espaço de memória para os elementos do arranjo.
- Nenhum valor é armazenado!
- Se quisermos armazenar valores no arranjo devemos usar uma das seguintes opções:
  - inicializar os elementos no momento da definição;
  - Ler os valores do teclado;
  - Atribuir valores individualmente aos elementos.
- Como veremos, somente arranjos de tamanho fixo podem ser inicializados quando são definidos.
- Arranjos de tamanho variável devem ser inicializados usando os dois métodos restantes.

Inicializando

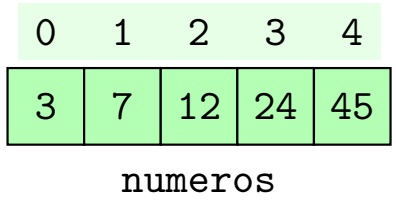
1. Inicialização básica:

```
int numeros[5] = {3,7,12,24,45};
```



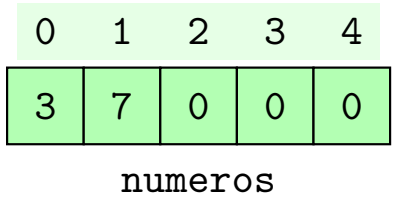
2. Inicialização sem o tamanho:

```
int numeros[] = {3,7,12,24,45};
```



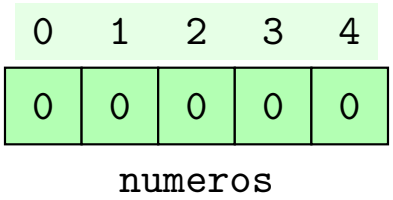
3. Inicialização parcial:

```
int numeros[5] = {3,7};
```



4. Inicialização com todos elementos zerados:

```
int numeros[5] = {0};
```



Entrando com valores

- Uma outra forma de preencher o arranjo é ler os valores do teclado ou de um arquivo.
- Quando o arranjo for totalmente preenchido, o laço mais apropriado é o `for` pois o número de elementos é fixo e conhecido:

```
for (i = 0; i < 10; i++)
   scanf("%d", &notas[i]);
```

- Note que o operador de endereços (&) é necessário na chamada de `scanf`.
- Se nem todos os elementos forem preenchidos, considere usar o laço `while`.

Atribuindo valores

- Podemos atribuir um valor a um elemento em particular usando a atribuição

```
notas[4] = 23;
```

- Em C não podemos atribuir um arranjo a outro arranjo. Se quisermos copiar o conteúdo de um arranjo para outro usamos:

```
for (i = 0; i < 25; i++)
   segundo[i] = primeiro[i];
```

- Se os valores do arranjo seguirem um padrão, podemos usar um laço para atribuir os valores. No exemplo a seguir, o laço atribui um valor que é duas vezes o índice do elemento:

```
for (i = 0; i < 10; i++)
   notas[i] = 2 * i;
```

- Neste outro exemplo, atribuímos números ímpares entre 1 e 19 ao arranjo:

```
for (i = 0; i < 10; i++)
   notas[i] = 2 * i + 1;
```

Permutando valores

- Uma atividade comum em arranjos é permutar o conteúdo de dois elementos.
- Considere que desejemos permutar o conteúdo de `numeros[3]` com o de `numeros[1]`.
- Observe a proposta abaixo. Ela faz o serviço?

```
numeros[3] = numeros[1];
numeros[1] = numeros[3];
```

Permutando valores

- Uma atividade comum em arranjos é permutar o conteúdo de dois elementos.
- Considere que desejemos permutar o conteúdo de `numeros[3]` com o de `numeros[1]`.
- Observe a proposta abaixo. Ela faz o serviço?

```
numeros[3] = numeros[1];
numeros[1] = numeros[3];
```

- Não! Ela só faz metade do serviço pois apenas o quarto elemento teve seu valor alterado. O valor do segundo continua igual.

Permutando valores

- Uma atividade comum em arranjos é permutar o conteúdo de dois elementos.
- Considere que desejemos permutar o conteúdo de `numeros[3]` com o de `numeros[1]`.
- Observe a proposta abaixo. Ela faz o serviço?

```
numeros[3] = numeros[1];
numeros[1] = numeros[3];
```

- Não! Ela só faz metade do serviço pois apenas o quarto elemento teve seu valor alterado. O valor do segundo continua igual.
- A solução correta é

```
temp = numeros[3];
numeros[3] = numeros[1];
numeros[1] = temp;
```

Imprimindo valores

- Outra operação comum é imprimir o conteúdo de um arranjo.
- Isto pode ser feito com a laço abaixo

```
for (i = 0; i < 10; i++)
    printf("%d ", notas[i]);
printf("\n");
```

- Se o arranjo for grande ficará díficil visualizar todos os elementos em uma única linha.
- Nesse caso é melhor imprimir uma quantidade menor por linha e continuar a impressão nas próximas linhas.

Imprimindo muitos valores de um só vez

O trecho de código a seguir permite que no máximo dez valores sejam escritos por linha.

```
#define TAM_MAX 25

int vet[TAM_MAX];

int numerosImpressos = 1;

for (int i = 0; i < TAM_MAX; i++){
  printf("%3d", vet[i]);
  if (numerosImpressos > 9){
     printf("\n");
     numerosImpressos = 1;
  } else
   numerosImpressos++;
} // for
```

Note que a variável `numerosImpressos` conta quantos números já foram impressos na linha atual.

Verificação do intervalo dos índices

- Não há verificação do intervalo de validade dos índices em C.
- É o programador que deve verificar se os índices estão dentro do intervalo do arranjo.
- Se usarmos um índice inválido obtermos resultados imprevisíveis, ou seja, não há como saber o valor que a elemento indexado terá.
- Erros deste tipo são bastante comuns em arranjos.
- Tomemos como exemplo o arranjo `notas` com 10 elementos. Imagine que se queira preenchê-lo com o código abaixo

```
for(i = 1; i <= 10; i++) // Código Errado
   scanf("%d", &notas[i]);
```

o código correto para isso é

```
for(i = 0; i < 10; i++)
   scanf("%d", &notas[i]);
```

Exemplo 1

No exemplo a seguir leremos uma sequência de números do teclado e depois a imprimiremos em ordem reversa.

```
10 #include <stdio.h>
11
12 #define TAM_MAX 50
13
14 int main(void){
15   int numMax=0; // Número máximo de inteiros ler na sequência
16   int numeros[TAM_MAX]={0};
17
18   printf("Você pode entrar com até 50 inteiros.\n");
19   printf("Quantos inteiros você quer entrar? ");
20   scanf("%d", &numMax);
```

```
22    if (numMax > TAM_MAX)
23      numMax = TAM_MAX;  // limita ao tamanho do arranjo
24
25    // Preenche o arranjo
26    printf("\nDigite seus números: \n");
27    for(int i = 0; i < numMax; i++)
28      scanf("%d", &numeros[i]);
29
30    // Imprime o arranjo
31    printf("\nSeus números revertidos são: \n");
32    for(int i = numMax - 1, numImpressos = 0;
33        i >= 0;
34        i--){
35      printf("%3d", numeros[i]);
36      if (numImpressos < 9)
37        numImpressos++;
38      else {
39        printf("\n");
40        numImpressos = 0;
41      } // else
42    } // for
43    printf("\n");
44
45    return 0;
46  } // main
```

Saída do exemplo

```
> ./exemplo1.exe
Você pode entrar com até 50 inteiros.
Quantos inteiros você quer entrar? 12

Digite seus números:
1 2 3 4 5 6 7 8 9 10 11 12

Seus números revertidos são:
 12 11 10  9  8  7  6  5  4  3
  2  1
```

Comunicação entre funções

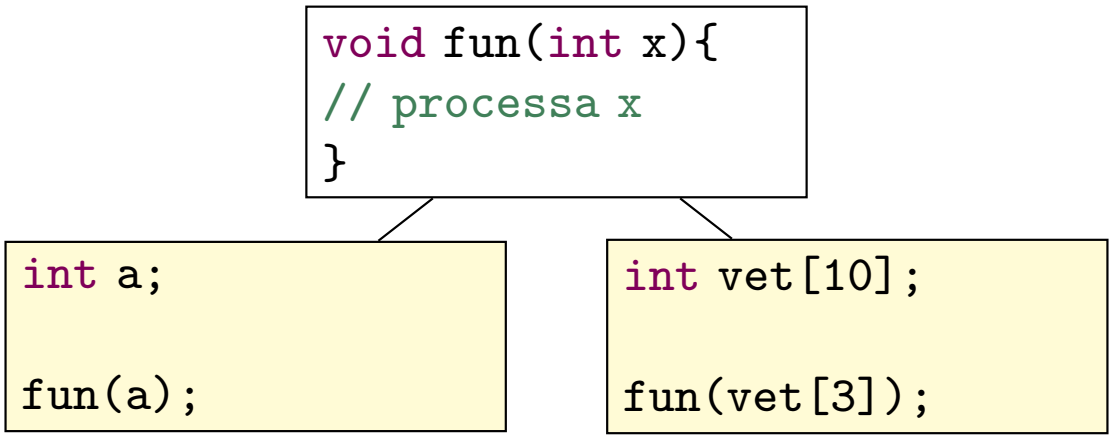
- Para processar arranjos em um programa grande, temos que estar aptos a passá-los para funções.
- Podemos passar arranjos de duas formas: passar elementos individualmente ou passar o arranjo todo.
- Começaremos discutindo como passar elementos individuais e depois veremos como passar o arranjo todo.

Passagem individual de elementos

- Podemos passar um determinado elemento do arranjo:
  - Por valor e nesse caso uma cópia é passada.
  - Por endereço.

Passagem de valores de dados

- Podemas passar o valor de um elemento do arranjo da mesma forma que passamos qualquer outro valor de dados.
- Desde que o tipo do elemento do arranjo case com o tipo do parâmetro da função, ele pode ser passado.
- Na figura abaixo é mostrado como um valor de um arranjo e de uma variável podem ser passados para a mesma função.



- Note como apenas um elemento do arranjo é passado usando a expressão indexada `vet[3]`.

Passagem de endereços

- Já vimos que podemos usar comunicação bidirecional para passar um endereço.
- Podemos passar um elemento individual do arranjo da mesma forma que passamos o endereço de uma variável qualquer: prefixos o elemento com o operador de endereço &.
- Lembre-se que ao passarmos um endereço há a comunicação bidirecional.

```
void fun(int* x){
// processa x
}
```

```
int a;

fun( &a );
```

```
int vet[10];

fun( &vet[3] );
```

Passagem do arranjo todo

- Aqui vemos a primeira situação na qual C não passa valores para uma função.
- A razão para isso está no fato de que muita memória e tempo poderiam ser usados ao passar cópias de arranjos grandes cada vez que quiséssemos usar uma função.
- Por exemplo, se o arranjo contendo 20 000 elementos fosse passado por valor para uma função, um outro arranjo, também com 20 000 elementos, precisaria ser alocado na região de memória da função chamada e cada elemento teria que ser copiado do arranjo original para o outro.
- Para evitar isso, C passa apenas o endereço do arranjo.
- Em C, o nome de um arranjo é uma expressão primária cujo valor é o endereço do primeiro elemento do arranjo.

Passagem do arranjo todo

- Ao passar o nome do arranjo, ao invés de um elemento indexado, é permitido à função chamada ter total acesso aos elementos do arranjo como se ela fosse a função chamadora, onde o arranjo foi definido.
- Na figura da esquerda é mostrado como passar um arranjo de *tamanho fixo*, enquanto na direita é mostrado como passar um arranjo de *tamanho variável*.

```
void fun(int vetf[]){
// processa vetf
}
```

```
int vet[10];

fun(vet);
```

```
void fun(int vetf[*]){
// processa vetf
}
```

```
int vet[tamanho];

fun(vet);
```

Arranjos de tamanho fixo

- Note que ao passar o arranjo, usamos simplesmente o nome do arranjo como parâmetro.
- Não precisamos especificar o número de elementos do arranjo de tamanho fixo.
  - Como o arranjo é de fato definido em algum outro ponto do programa, tudo o que importa é que o compilador saiba que é um arranjo.
- A declaração da função que recebe o arranjo pode ser feita de duas formas

```
void fun( int vet[], ...)
void fun( int* vet, ...)
```

- A primeira versão é melhor pois indica que a função recebe um arranjo.

Arranjos de tamanho variável

- Quando a função chamada recebe um arranjo de tamanho variável, devemos declará-lo e definí-lo como de tamanho variável.
- Na declaração da função usamos um asterisco no lugar do tamanho ou usamos uma variável para passar o tamanho.
- Na definição da função a variável contendo o tamanho do arranjo deve ser definida antes do arranjo.

```
// Declaração da função
float calcMediaVet( int tam, float vet[*]);

// Definição da função
float calcMediaVet( int tam, float vet[tam]){
...
}
```

- Como uma observação, o arranjo passado para uma função declarando um arranjo de tamanho variável pode ser tanto um arranjo de tamanho fixo quanto um de tamanho variável.
  - Desde que os tipos dos arranjos estejam corretos e o tamanho correto for especificado, o arranjo poderá ser processado.

Resumo das regras para passar todo o arranjo

1. A função deve ser chamada passado apenas o nome do arranjo.
2. Na definição da função, o parâmetro formal deve ser do tipo arranjo de tamanho fixo ou variável.
3. O tamanho de um arranjo de tamanho fixo não precisa ser especificado.
4. O tamanho de um arranjo de tamanho variável no protótipo da função deve ser um asterisco ou uma variável.
5. O tamanho do arranjo de tamanho variável na definição da função deve ser uma variável previamente definida como um parâmetro.

Passando um arranjo como uma constante

- Quando uma função recebe um arranjo e não o modifica, o arranjo deveria ser recebido como uma constante.
- Isso evita que a função acidentalmente modifique o arranjo.
- Declaramos um arranjo constante como no código a seguir

```
double media(const int vet[], int tam);
```

- Quando um arranjo é declarado como constante, qualquer tentativa de alterar um elemento será sinalizado pelo compilador como um erro.

Exemplo 2 (Média aritmética dos elementos do arranjo)

Vamos usar uma função para calcular a média aritmética dos inteiros presentes em um arranjo. Nesse caso passamos o nome do arranjo para a função e ela retorna a média como um número real.

```
 9 #include <stdio.h>
10
11 #define TAM_MAX 5
12
13 // Protótipo da função
14 double media_aritmetica(int vet[]);
15
16 int main(void){
17   double media=0.0;
18   int numeros[TAM_MAX]={3, 7, 2, 4, 5};
19
20   media = media_aritmetica(numeros);
21   printf("A média é%lf\n", media);
22
23   return 0;
24 } // main
```

```
26 /**
27  * @brief Calcula a média aritmética dos elementos de um arranjo
28  *
29  * @param vet um vetor contendo números inteiros
30  * @return a média aritmética
31  */
32 double media_aritmetica(int vet[]){
33   int soma = 0;
34
35   for(int i = 0; i < TAM_MAX; i++)
36     soma += vet[i];
37
38   return (soma / (double) TAM_MAX);
39 } // media_aritmetica
```

A execução do programa acima produz como saída

```
> ./exemplo2.exe
A média é 4.200000
```

Exemplo 3 (Média aritmética dos elementos de um arranjo de tamanho variável)

Vamos repetir o exemplo anterior, desta vez usando um arranjo com tamanho variável. Como este tipo de arranjo não pode ser inicializado na declaração, teremos que os dados do teclado. Note também que para que o programa fique o mais simples possível, criamos o arranjo em um bloco. Isso se fez necessário pois o tamanho do arranjo deve ser conhecido antes dele ser declarado.

```
10 #include <stdio.h>
11
12 // Protótipo da função
13 double media_aritmetica(int tamanho, int vet[*]);
14
15 int main(void){
16   int tamanho=0;
17   double media=0.0;
18
19   printf("Quantos números devo ler? ");
20   scanf("%d", &tamanho);
```

```
22   // Cria e preenche um arranjo de tamanho variável
23   {
24     int numeros[tamanho];
25
26     // Preenche o arranjo
27     for(int i = 0; i < tamanho; i++){
28       printf("Digite o número %2d: ", i + 1);
29       scanf("%d", &numeros[i]);
30     } // for
31     media = media_aritmetica(tamanho, numeros);
32   } // Bloco para preecher o arranjo
33
34   printf("A média é%lf\n", media);
35
36   return 0;
37 } // main
```

```
39 /**
40  * @brief Calcula a média aritmética dos elementos de um arranjo
41  *
42  * @param vet um vetor contendo números inteiros
43  * @return a média aritmética
44  */
45 double media_aritmetica(int tamanho, int vet[tamanho]){
46   int soma = 0;
47
48   for(int i = 0; i < tamanho; i++)
49     soma += vet[i];
50
51   return ((double) soma / tamanho);
52 } // media_aritmetica
```

A execução do programa acima produz como saída

```
> ./exemplo3.exe
Quantos números devo ler? 5
Digite o número  1: 3
Digite o número  2: 6
Digite o número  3: 9
Digite o número  4: 12
Digite o número  5: 15
A média é 9.000000
```

Exemplo 4 (Alterando valores no arranjo)

O programa a seguir usa a comunicação bidirecional para alterar valores no arranjo. Note que nenhum código especial é necessário para alterar os valores do arranjo. Pois, como o endereço dele foi passado, podemos simplesmente usar a notação de índices para alterá-lo.

```
 9 #include <stdio.h>
10
11 #define TAM_MAX 5
12
13 // Protótipo da função
14 void multiplica2(int vet[]);
15
16 int main(void){
17   int numeros[TAM_MAX]={3, 7, 2, 4, 5};
18
19   multiplica2(numeros);
20
21   printf("O arranjo agora contém: ");
22   for(int i = 0; i < 5; i++)
23     printf("%3d", numeros[i]);
24   printf("\n");
25
26   return 0;
27 } // main
```

```
29 /**
30  * @brief Multiplica cada elemento de um arranjo por 2
31  *
32  * @param vet um vetor contendo números inteiros
33  */
34 void multiplica2(int vet[]){
35
36   for(int i = 0; i < TAM_MAX; i++)
37      vet[i] *= 2;
38
39   return;
40 } // multiplica2
```

A execução do programa acima produz como saída

```
> ./exemplo4.exe
O arranjo agora contém:    6 14 4 8 10
```

Aplicações de arranjos

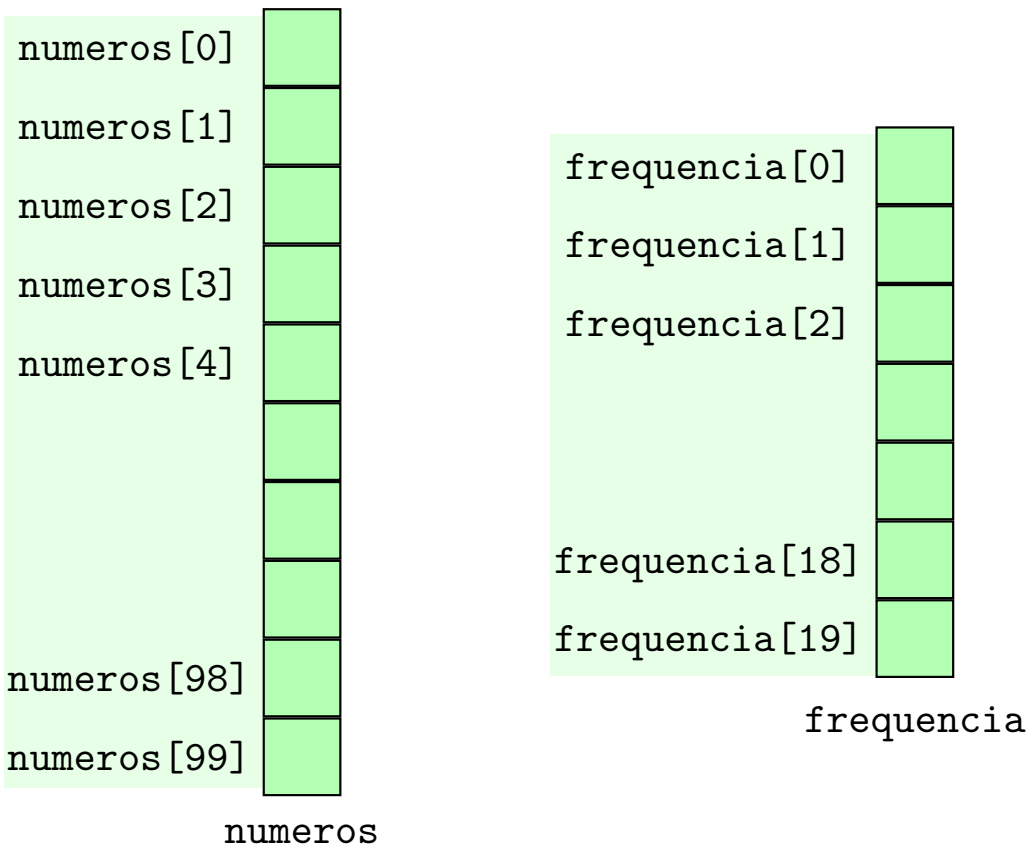
- Veremos a partir de agora um exemplo mais longo envolvendo arranjos.
- O exemplo lida com a contagem da ocorrência (frequência) de elementos em um arranjo e a exibição de uma representação gráfica para esta frequência.

Arranjos de frequência

- Em estatística é comum usar arranjos em duas aplicações comuns: distribuições de frequência e histogramas.
- Um arranjo de frequência mostra o número de elementos com valor idêntico encontrado em uma sequência de números.
  - Como exemplo, assuma que tenhamos uma amostra com 100 valores entre 0 e 19.
  - Queremos saber quantas valores são 0, quantos são 1, quantos são 2 e assim por diante até 19.

Arranjos de frequência

- Podemos colocar os valores da amostra em um arranjo chamado `numeros`.
- Depois criamos um arranjo com 20 elementos que conterá a frequência de cada número na sequência.



Arranjos de frequência

- Sabemos que a amostra contém exatamente 100 elementos, assim podemos usar um laço `for` para examinar cada valor do arranjo.
- Mas, como podemos relacionar o valor em `numeros` com uma posição na frequência?

Arranjos de frequência

- Sabemos que a amostra contém exatamente 100 elementos, assim podemos usar um laço `for` para examinar cada valor do arranjo.
- Mas, como podemos relacionar o valor em `numeros` com uma posição na frequência?
- Uma forma é atribuir o valor no arranjo de dados a um índice e depois usar esse índice para acessar o arranjo de frequência:

```
f = numeros[i];
frequencia[f]++;
```

Arranjos de frequência

- Sabemos que a amostra contém exatamente 100 elementos, assim podemos usar um laço `for` para examinar cada valor do arranjo.
- Mas, como podemos relacionar o valor em `numeros` com uma posição na frequência?
- Uma forma é atribuir o valor no arranjo de dados a um índice e depois usar esse índice para acessar o arranjo de frequência:

```
f = numeros[i];
frequencia[f]++;
```

- Entretanto, dado que um índice é uma expressão, podemos simplesmente usar o valor do arranjo de dados diretamente como um índice:

```
frequencia[ numeros[i] ]++;
```

- A função completa, chamada de `conteFrequencia`, é mostrada no próximo slide.

A função conteFrequencia

```
16 /**
17  * @brief Analisa os dados em nums e constrói a distribuição de
18  *        frequência
19  *
20  * @param nums      arranjo com dados validados a ser analisado
21  * @param tamanho   número de elementos no arranjo
22  * @param frequencia arranjo que acumulará a frequência
23  * @param intervalo valor/índice máximo para a frequência
24  */
25 void conteFrequencia(int nums[],   int tamanho,
26                      int frequencia[], int intervalo){
27
28   // Primeiro inicializa o arranjo de frequência
29   for(int i = 0; i < intervalo; i++)
30     frequencia[i] = 0;
31
32   // Varre nums e constrói o arranjo de frequência
33   for(int i = 0; i < tamanho; i++)
34     frequencia[ nums[i] ]++;
35
36   return;
37 } // conteFrequencia
```

Histogramas

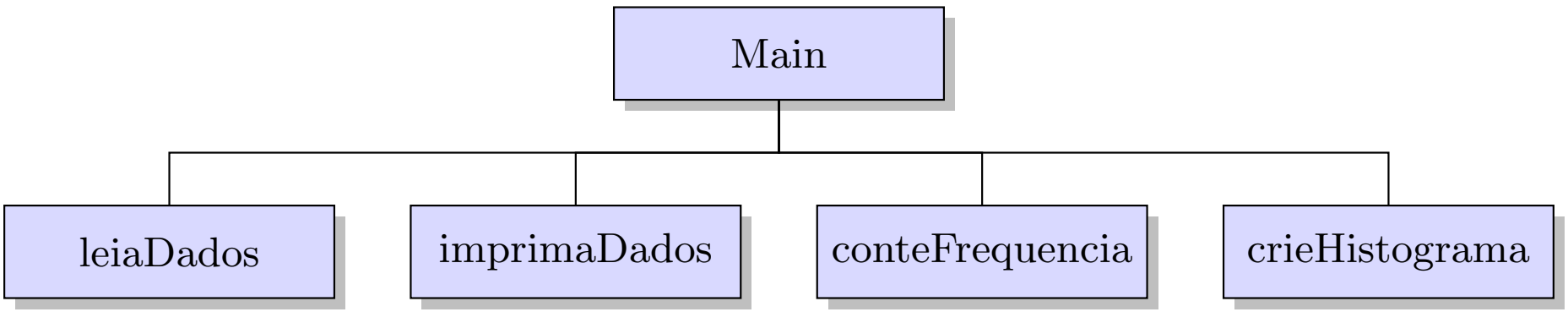
- Um histograma é uma representação gráfica de um arranjo de frequência.
- Ao invés de imprimir os valores dos elementos juntamente com suas frequências, imprime-se um histograma na forma de um gráfico de barras.
- Como um exemplo, a figura a seguir mostra um histograma para uma amostra com números entre 0 e 19.
  - Usamos asteriscos para construir a barra.
  - Cada asterisco representa uma ocorrência.

```
 0  0
 1  4 ****
 2  7 *******
 3  7 *******
 .
 .
 .
18  2 **
19  0
```

Histogramas

Exemplo 5 (Frequência e Histograma)

Vamos escrever um programa que constrói um arranjo de frequência para os dados no intervalo $[0, 19]$ e depois imprime o histograma. Os dados deveriam ser lidos de um arquivo, mas como ainda não vimos isso, leremos do teclado. Para tornar o programa flexível, a função `leiaDados` pode apenas prreencher parcialmente o arranjo. O projeto do programa é mostrado na figura abaixo.



A função crieHistograma

A função para imprimir o histograma é mostrada a seguir.

```
40 /**
41  * @brief Imprime um histograma representando os dados analisados.
42  *
43  * @param freq  um arranjo contendo o número de ocorrências de cada valor
44  * @param tamanho número de elementos no arranjo de frequência
45  */
46 void crieHistograma(int freq[], int tamanho){
47     for(int i = 0; i < tamanho; i++){
48         printf("%2d %2d ", i, freq[i]);
49         for(int j = 1; j <= freq[i]; j++)
50             printf("*");
51         printf("\n");
52     } // for i
53     return;
54 } // crieHistograma
```

A função leiaDados

```
57 /**
58  * @brief Lê os dados do teclado e coloca no arranjo. O arranjo
59  *        não precisa ser completamente preenchido.
60  *
61  * @param dados   um arranjo vazio
62  * @param tamanho número máximo de elementos do arranjo
63  * @param intervalo maior valor permitido
64  * @return número de elementos lidos
65  */
66 int leiaDados(int dados[], int tamanho, int intervalo){
67     int i = 0;
68     int numero = 0;
69
70     while (i < tamanho && scanf("%d", &numero) != EOF){
71         if (0 <= numero && numero < intervalo)
72             dados[i++] = numero;
73         else
74             printf("O número %d éinvalido e será ignorado.\n",
75                    numero);
76     } // while i
77
78     return i;
79 } // leiaDados
```

A função leiaDados

Note que o laço `while` na função `leiaDados` é controlado tanto pela contador `i` quanto pelo final de arquivo (`EOF`).

- `EOF` é lido por `scanf` sempre que o final de um arquivo é encontrado, ou, como nesse caso, que o usuário digite no terminal Ctrl + d, que simula o fim de arquivo.
- Assim pode-se ler menos que o número máximo de elementos solicitado (guardado em `tamanho`).
- Além disso, números que não estejam no intervalo permitido serão ignorados.

A função imprimaDados

```
82 /**
83  * @brief Imprime os dados como uma tabela
84  *
85  * @param dados   um arranjo com os dados
86  * @param tamanho número de elementos no arranjo
87  * @param maxLinha número máximo de elementos por linha
88  */
89 void imprimaDados(int dados[], int tamanho, int maxLinha){
90     int numImpressos = 0;
91 
92     printf("\n\n");
93     for(int i = 0; i < tamanho; i++){
94         printf("%2d ", dados[i]);
95         numImpressos++;
96         if (numImpressos >= maxLinha){
97             printf("\n");
98             numImpressos = 0;
99         } // if
100    } // for
101    printf("\n\n");
102    return;
103 } // imprimaDados
```

A função principal

```
106 int main(void){
107     int tamanho = 0;
108     int numeros[TAM_MAX]={0};
109     int frequencia[INTERVALO_FREQ]={0};
110 
111     tamanho = leiaDados(numeros, TAM_MAX, INTERVALO_FREQ);
112     imprimaDados(numeros, tamanho, 10);
113 
114     conteFrequencia(numeros, tamanho, frequencia, INTERVALO_FREQ);
115     crieHistograma(frequencia, INTERVALO_FREQ);
116     return 0;
117 } // main
```

Uso do exemplo 5

Podemos usar o programa entrando com os dados de duas formas:

1. Digitando os números um por um até que a quantidade seja 100 ou até que pressionemos Ctrl + d.
2. Colocando os números em um arquivo de nome `dados.dat` e redirecionando a entrada padrão no terminal para que o programa leia desse arquivo.

No primeiro caso teremos o seguinte

```
> ./exemplo5.exe
1 2 3 4 5 6 7 8 7 10
2 12 13 13 15 16 17 18 17 7
3 4 6 8 10 2 4 6 8 10 20
4 3 5 7 1 3 7 7 11 13
5 10 11 12 13 16 18 11 12 7
6 1 2 2 3 3 3 4 4 4 25
7 7 8 7 6 5 4 1 2 2
8 11 13 13 13 17 17 7 7
Ctrl + d
```

Uso do exemplo 5

Cuja saída será

```
O número 20 é inválido e será ignorado.
O número 25 é inválido e será ignorado.

 1  2  3  4  5  6  7  8  7 10
 2 12 13 13 15 16 17 18 17  7
 3  4  6  8 10  2  4  6  8 10
 4  3  5  7  1  3  7  7 11 13
 5 10 11 12 13 16 18 11 12  7
 6  1  2  2  3  3  3  4  4  4
 7  7  8  7  6  5  4  1  2  2
 8 11 13 13 13 17 17  7  7
```

Continua no próximo slide...

Uso do exemplo 5

```
 0  0
 1  4 ****
 2  7 *******
 3  7 *******
 4  8 ********
 5  4 ****
 6  5 *****
 7 12 ************
 8  5 *****
 9  0
10  4 ****
11  4 ****
12  3 ***
13  7 *******
14  0
15  1 *
16  2 **
17  4 ****
18  2 **
19  0
```

Uso do exemplo 5 – segunda forma

Podemos colocar os números em um arquivo de nome `dados.dat` e redirecionando a entrada padrão no terminal para que o programa leia desse arquivo.

```
> ./exemplo5.exe < dados.dat
```

A saída será idêntica à da entrada manual de dados.

Abaixo está o conteúdo do arquivo `dados.dat`

```
1 2 3 4 5 6 7 8 7 10
2 12 13 13 15 16 17 18 17 7
3 4 6 8 10 2 4 6 8 10 20
4 3 5 7 1 3 7 7 11 13
5 10 11 12 13 16 18 11 12 7
6 1 2 2 3 3 3 4 4 4 25
7 7 8 7 6 5 4 1 2 2
8 11 13 13 13 17 17 7 7
```

Para saber mais

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.

Fontes

- Forouzan, B. A and Gilbert, R. F. *Computer Science: a structured programming approach using C*. 3rd edition. Cengage Learning, 2007.