

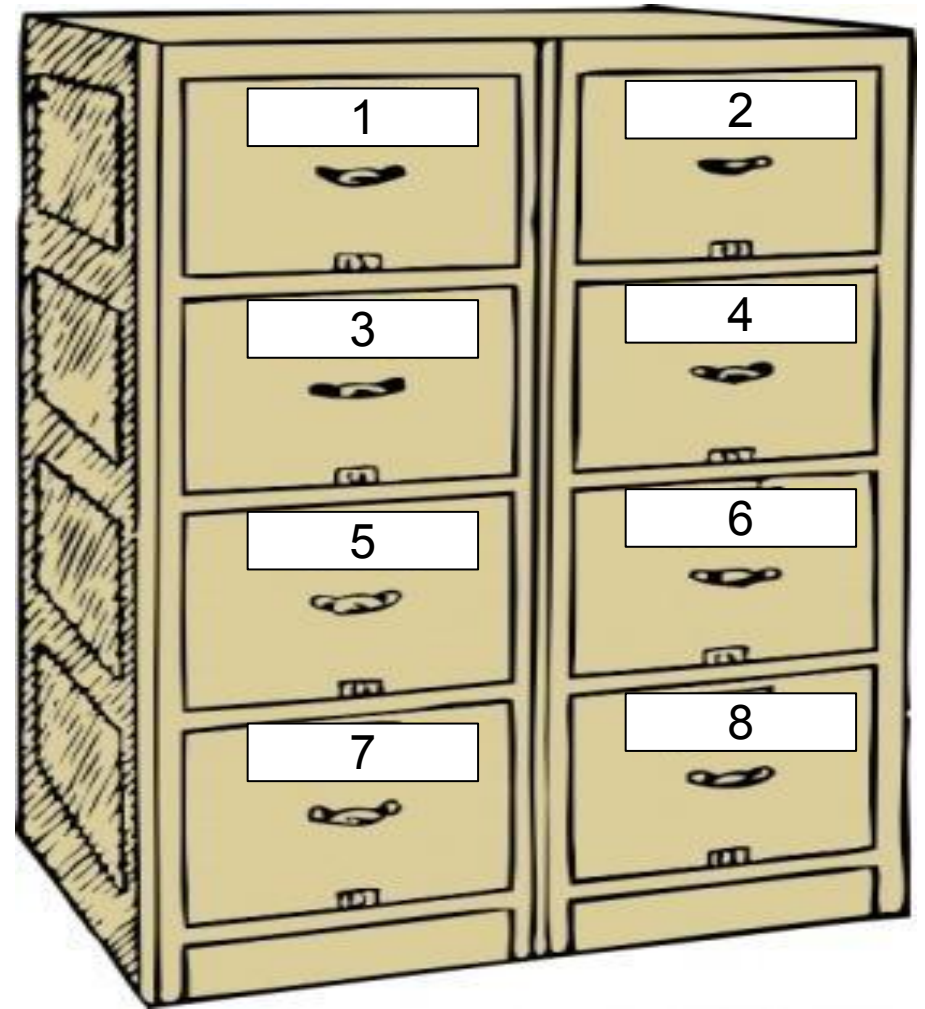
Algoritmos e Estruturas de Dados I

Variáveis Indexadas

Pedro O.S. Vaz de Melo

Por que índices são importantes?

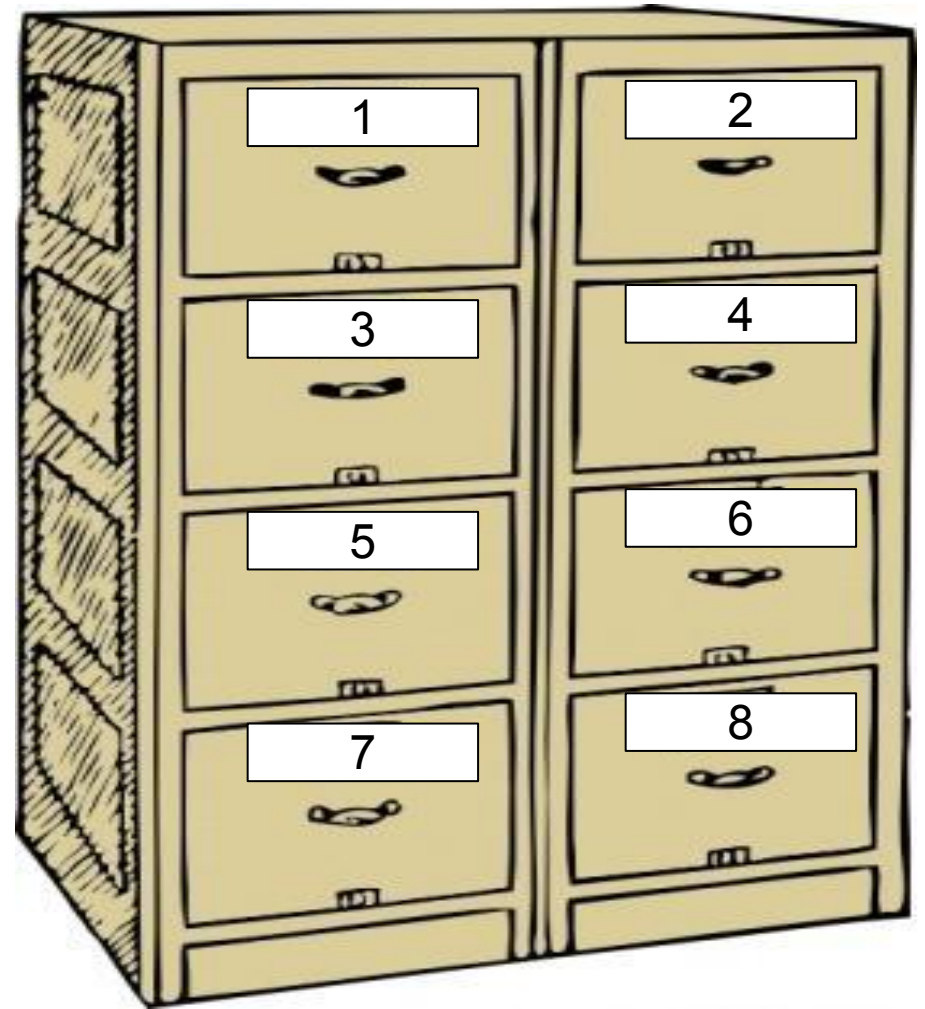
- Como uma loja de sapatos artesanais deve guardar os seus produtos?
- Tamanhos entre 35 e 42
- Produz um par por tamanho, em ordem aleatória
 - Ex: 39, 35, 42, 41, 37, 36, 38, 40



Armário da loja

Por que índices são importantes?

- Como uma loja de sapatos artesanais deve guardar os seus produtos?
- Tamanhos entre 35 e 42
- Produz um par por tamanho, em ordem aleatória
 - Ex: 39, 35, 42, 41, 37, 36, 38, 40
- **Gaveta = tamanho - 34**



Armário da loja

Problema 1

- Uma loja pretende **simular um dia de suas vendas**. Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que cada cliente compra apenas um produto.
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser gerados aleatoriamente.
- No máximo 200 produtos.
- No máximo 50 clientes.

Entendendo o problema

- Preciso de uma lista de produtos (max: 200) e de clientes (max: 50)

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 5.99 |
| 1 | 7.04 |
| 2 | 99.99 |
| 3 | 1.99 |
| 4 | 42.50 |
| ... | ... |

Clientes

| Id Cliente |
|------------|
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| ... |

Problema 1

- Uma loja pretende **simular um dia de suas vendas**. Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que **cada cliente compra apenas um produto.**
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser gerados aleatoriamente.
- No máximo 200 produtos.
- No máximo 50 clientes.

Entendendo o problema

- Preciso de uma lista de produtos (max: 200) e de clientes (max: 50)

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 5.99 |
| 1 | 7.04 |
| 2 | 99.99 |
| 3 | 1.99 |
| 4 | 42.50 |
| ... | ... |

Clientes

| Id Cliente |
|------------|
| 0 |
| 1 |
| 2 |
| 3 |
| 4 |
| ... |

Entendendo o problema

- Preciso de uma lista de produtos (max: 200) e de clientes (max: 50)

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 5.99 |
| 1 | 7.04 |
| 2 | 99.99 |
| 3 | 1.99 |
| 4 | 42.50 |
| ... | ... |

Compras dos clientes

| Id Cliente | Id Produto |
|------------|------------|
| 0 | 3 |
| 1 | 0 |
| 2 | 45 |
| 3 | 12 |
| 4 | 30 |
| ... | ... |

Codificando o Problema

```
4  #define MAX_PRODUTOS 200
5  #define MAX_CLIENTES 50
6
7  void main() {
8
9  float precos[MAX_PRODUTOS];
10 int compras[MAX_CLIENTES];
```

Entendendo o Problema

- Como popular os vetores **precos** e **compras**?

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 5.99 |
| 1 | 7.04 |
| 2 | 99.99 |
| 3 | 1.99 |
| 4 | 42.50 |
| ... | ... |

Compras dos clientes

| Id Cliente | Id Produto |
|------------|------------|
| 0 | 3 |
| 1 | 0 |
| 2 | 45 |
| 3 | 12 |
| 4 | 30 |
| ... | ... |

Codificando o Problema

```
//define o numero de produtos da loja
num_produtos = 50;
//define o preco de cada produto
for(i=0; i<num_produtos; i++) {
    preco = 9.99;
    precos[i] = preco;
}
```

Codificando o Problema

```
//define o numero de clientes que farao compras em um dia
num_clientes = 5;
//simula o que cada cliente comprou
for(i=0; i<num_clientes; i++) {
    id_produto = 32;
    compras[i] = id_produto;
}
```

Entendendo o Problema

- Como calcular o valor total das compras feitas em um dia?

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 9.99 |
| 1 | 9.99 |
| 2 | 9.99 |
| 3 | 9.99 |
| 4 | 9.99 |
| ... | ... |
| 32 | 9.99 |
| ... | ... |

Compras dos clientes

| Id Cliente | Id Produto |
|------------|------------|
| 0 | 32 |
| 1 | 32 |
| 2 | 32 |
| 3 | 32 |
| 4 | 32 |
| ... | ... |

Codificando o Problema

```
total_vendas = 0;
for(i=0; i<num_clientes; i++) {
    int cliente = i;
    id_produto = compras[cliente];
    float valor = precos[id_produto];
    printf("%3d\t%3d\t%7.2f\n", cliente, id_produto, valor);
    total_vendas = total_vendas + valor;
}
```

Codificando o Problema

| ----- | | |
|------------------|---------|------------|
| Cliente | Produto | Valor(R\$) |
| ----- | | |
| 0 | 32 | 9.99 |
| 1 | 32 | 9.99 |
| 2 | 32 | 9.99 |
| 3 | 32 | 9.99 |
| 4 | 32 | 9.99 |
| ----- | | |
| TOTAL DE VENDAS: | | 49.95 |

Problema 1

- Uma loja pretende **simular um dia de suas vendas**. Sabe-se que os preços dos produtos vendidos nesta loja variam de R\$5,00 a R\$100,00 e que cada cliente compra apenas um produto.
- O número e o preço de cada produto, o número de clientes e os produtos comprados pelos clientes devem ser gerados aleatoriamente.
- No máximo 200 produtos.
- No máximo 50 clientes.

Entendendo o problema

- Como gerar números aleatórios???

Preços dos produtos

| Id Produto | Valor |
|------------|-------|
| 0 | 5.99 |
| 1 | 7.04 |
| 2 | 99.99 |
| 3 | 1.99 |
| 4 | 42.50 |
| ... | ... |

Compras dos clientes

| Id Cliente | Id Produto |
|------------|------------|
| 0 | 3 |
| 1 | 0 |
| 2 | 45 |
| 3 | 12 |
| 4 | 30 |
| ... | ... |

Geração de números aleatórios

- A linguagem C dispõe da função `rand` para a geração de um inteiro aleatório no intervalo de 0 até `RAND_MAX` (constante definida em `stdlib.h`).
- Especificamente: `RAND_MAX` = 0x7FFF = 32767.
- Para gerar números aleatórios em um intervalo específico podemos escrever uma função:

```
int random(int n)
{
    return rand() % n;
}
```

Que valor é este?

Um inteiro pertencente
ao intervalo: `[0,n-1]`

Geração de números aleatórios

- Sendo assim, para a geração do número de produtos, podemos utilizar:

```
num_produtos = 1 + random(MAX_PRODUTOS) ;
```

- Se **MAX_PRODUTOS** é uma constante que vale 200, tem-se: **num_produtos** $\in [1, 200]$
- Para a geração dos preços dos produtos, podemos escrever:

```
preco = 5 + random(96) ;  
preco[i] = preco ;
```

- ou seja, **p** $\in [5, 100]$.

Geração de números aleatórios

- Ao trabalhar com números aleatórios, busca-se em cada execução do programa, gerar novos números.
- Para isso, ao se usar a função `rand`, é necessário fornecer uma `semente` à mesma... Como assim?
- Para uma dada `semente`, a função `rand` fará `brotar` uma determinada sequência de números.
- Para garantir que a sequência gerada seja sempre diferente, deve-se mudar a semente a cada execução.

Geração de números aleatórios

- Mas como mudar a semente?
- Uma boa ideia é usar o valor retornado pela função `time` (definida em `time.h`).
- A função `time` retorna como valor o número de segundos transcorridos desde 01/01/1970 e pode armazenar este valor num parâmetro.

```
t = time(p);
```

ou

```
t = time(NULL);
```

Caso não seja necessário o parâmetro `p`.

Geração de números aleatórios

- A função `srand` é responsável pela inicialização da semente a ser usada pela função `rand`.
- A função `srand` exige como parâmetro um valor inteiro do tipo `unsigned`.
- Podemos escrever:

```
srand( (unsigned) time (NULL) ) ;
```

Análise do programa

```
4 #define MAX_PRODUTOS 200
5 #define MAX_CLIENTES 50
6
7 int random(int n) {
8     return rand()%n;
9 }
10
11 void main() {
12
13     float precos[MAX_PRODUTOS];
14     int compras[MAX_CLIENTES];
15     int num_produtos, num_clientes, i, preco, id_produto;
16     float total_vendas = 0;
```

Análise do programa

```
19 //inicializar gerador de numeros aleatorios
20 srand((unsigned)time(NULL));
21
22 //define o numero de produtos da loja
23 num_produtos = random(MAX_PRODUTOS)+1;
24 //define o preco de cada produto
25 for(i=0; i<num_produtos; i++) {
26     preco = 5 + random(96);
27     precos[i] = preco;
28 }
29
30 //define o numero de clientes que farao compras em um dia
31 num_clientes = random(MAX_CLIENTES)+1;
32 printf("Simulacao para:\n %d produtos\n %d clientes\n", num_produtos, num_clientes);
33 //simula o que cada cliente comprou
34 for(i=0; i<num_clientes; i++) {
35     id_produto = random(num_produtos);
36     compras[i] = id_produto;
37 }
38
```


Análise do programa

```
39 //exibe resultado da simulacao
40 printf("-----\n");
41 printf("Cliente\tProduto\tValor(R$) \n");
42 printf("-----\n");
43
44 total_vendas = 0;
45 for(i=0; i<num_clientes; i++) {
46     int cliente = i;
47     id_produto = compras[cliente];
48     float valor = precos[id_produto];
49     printf("%3d\t%3d\t%7.2f\n", cliente, id_produto, valor);
50     total_vendas = total_vendas + valor;
51 }
52
53 printf("-----\n");
54 printf("TOTAL DE VENDAS:\t%7.2f\n", total_vendas);
55
56 getch();
57 }
```

Variáveis indexadas

- Considere a seguinte execução do programa:

```
Simulacao para:
 162 produtos
 5 clientes
-----
Cliente Produto Valor(R$)
-----
 0      124      46.00
 1       23      75.00
 2       21      40.00
 3       96      57.00
 4      116      43.00
-----
TOTAL DE VENDAS:      261.00
```

- Para esta simulação, observa-se 5 clientes (0 a 4)
- Os produtos comprados por estes clientes estão armazenados na variável indexada **compras**.

Variáveis indexadas

- Ou seja:

| | | | | | | | | |
|---------|----|----|----|----|----|---|-----|----|
| compras | 68 | 45 | 90 | 45 | 44 | | ... | |
| | 0 | 1 | 2 | 3 | 4 | 5 | ... | 49 |

- Exemplo: o cliente 0 comprou o produto 68.

- Os preços estão armazenados na variável **precos**:

| | | | | | | | | | | | | | | |
|--------|---|-----|----|----|-----|----|-----|----|-----|-----|-----|-----|-----|-----|
| precos | ? | ... | 5 | 57 | ... | 73 | ... | 82 | ... | ? | ... | ? | ... | |
| | 0 | | 44 | 45 | ... | 68 | ... | 90 | ... | 146 | ... | 158 | ... | 199 |

- Exemplo: o produto 68 tem preço igual a 73 reais e foi comprado pelo cliente 0.

Variáveis indexadas

- Observe que o **valor de uma variável indexada** pode ser o **índice** de uma outra variável indexada.
- Veja, por exemplo, o caso do cliente **0**:

```
compras[0] = 68  
precos[68] = precos[compra[0]] = 73
```

- Ou seja, o **índice** de uma variável indexada precisa **ser um inteiro** com valor limitado ao número de posições de memória declaradas para a variável.

Variáveis indexadas

- O **valor deste inteiro** pode ser o valor de uma **constante**, o valor de uma **expressão**, o valor de uma **variável** ou o valor retornado por uma **função**.
- Assim, são válidas as expressões:

```
int x[10], i = 5, a;
```

```
a = x[3];
```

```
a = x[i];
```

```
a = x[x[a]];
```

```
a = x[random(10)];
```

Variáveis indexadas

- Atenção!

- Ao utilizar variáveis indexadas, temos que controlar o valor do índice no intervalo 0 a $n-1$, onde n é o número de posições de memória.
- Caso contrário, podemos ter invasão de memória.

Variáveis indexadas

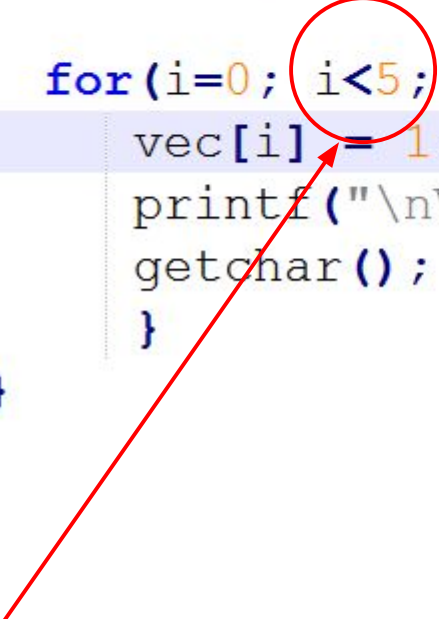
```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```

Variáveis indexadas

```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```



invasão de memória!

Variáveis indexadas

```
Valor de vec[0]: 1
```

```
Valor de vec[1]: 1
```

```
Valor de vec[2]: 1
```

```
Valor de vec[3]: 1
```

```
Valor de vec[1]: 1
```

```
Valor de vec[2]: 1
```

```
Valor de vec[3]: 1
```

```
Valor de vec[1]: 1
```

```
Valor de vec[2]: 1
```

Variáveis indexadas

```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```

| <u>End.</u> | <u>Variável</u> | <u>Valor</u> |
|-------------|-----------------|--------------|
| #1 | vec[0] | 10 |
| #2 | vec[1] | 20 |
| #3 | vec[2] | 30 |
| #4 | vec[3] | 40 |
| #5 | i | 0 |

Variáveis indexadas

```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```

| <u>End.</u> | <u>Variável</u> | <u>Valor</u> |
|-------------|-----------------|--------------|
| #1 | vec[0] | 1 |
| #2 | vec[1] | 1 |
| #3 | vec[2] | 1 |
| #4 | vec[3] | 1 |
| #5 | i | 3 |

Variáveis indexadas

```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```

| <u>End.</u> | <u>Variável</u> | <u>Valor</u> |
|-------------|-----------------|--------------|
| #1 | vec[0] | 1 |
| #2 | vec[1] | 1 |
| #3 | vec[2] | 1 |
| #4 | vec[3] | 1 |
| #5 | i | 4 |



vec[4]

Variáveis indexadas

```
#include <stdio.h>
#include <ctype.h>
void main ()
{
    int vec[]={10, 20, 30, 40};
    int i=0;

    for(i=0; i<5; i++) {
        vec[i] = 1;
        printf("\nValor de vec[%d]: %d", i, vec[i]);
        getchar();
    }
}
```

| <u>End.</u> | <u>Variável</u> | <u>Valor</u> |
|-------------|-----------------|--------------|
| #1 | vec[0] | 1 |
| #2 | vec[1] | 1 |
| #3 | vec[2] | 1 |
| #4 | vec[3] | 1 |
| #5 | i | 1 |



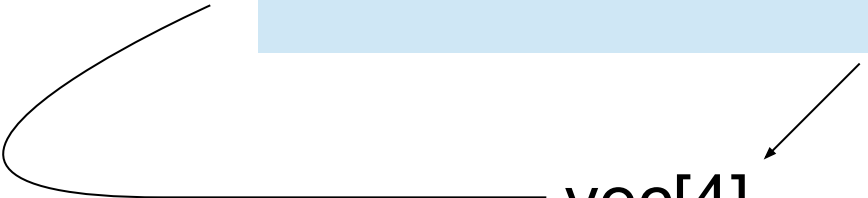
vec[4]

Variáveis indexadas

```
Valor de vec[0]: 1
Valor de vec[1]: 1
Valor de vec[2]: 1
Valor de vec[3]: 1
Valor de vec[1]: 1
Valor de vec[2]: 1
Valor de vec[3]: 1
Valor de vec[1]: 1
Valor de vec[2]: 1
```

| <u>End.</u> | <u>Variável</u> | <u>Valor</u> |
|-------------|-----------------|--------------|
| #1 | vec[0] | 1 |
| #2 | vec[1] | 1 |
| #3 | vec[2] | 1 |
| #4 | vec[3] | 1 |
| #5 | i | 1 |

vec[4]



Problema 2

- Uma grande empresa armazena em uma variável indexada os números dos cheques emitidos num dia pelo setor financeiro.
- Considere que os números dos cheques são valores inteiros de 1 a 100 e que os cheques são emitidos em uma ordem aleatória.
- Ao final do dia, para facilitar o controle, a empresa precisa ordenar estes dados em ordem crescente.

Ordenação por contagem

- A ordenação de dados é uma aplicação importante em computação, existindo vários algoritmos eficientes de ordenação.
- O algoritmo que veremos para resolver este problema é um dos mais simples e menos eficientes: **ordenação por contagem**.
- Exemplo: imagine uma variável indexada **vet** contendo os seguintes valores:

| | | | | | | | | |
|------------|----|----|----|-----|----|----|----|---|
| vet | 38 | 97 | 19 | 100 | 23 | 47 | 41 | 8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Ordenação por contagem

- Considere ainda uma outra variável **pos**, de mesmo tamanho, com valores inicialmente iguais a zero:

| | | | | | | | | |
|------------|---|---|---|---|---|---|---|---|
| pos | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- O propósito da variável **pos[i]** é armazenar a posição que o elemento **vet[i]** deve ter na ordenação.
- Para determinar esta posição, basta **contar** para cada elemento **vet[i]**, quantos elementos menores do que ele existem na variável **vet** e armazenar em **pos[i]**.

Ordenação por contagem

- Exemplo:

- | | | | | | | | | |
|------------|----|----|----|-----|----|----|----|---|
| vet | 38 | 97 | 19 | 102 | 23 | 47 | 41 | 8 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| i | vet[i] | Elementos menores do que vet[i] | pos[i] |
|----------|---------------|--|---------------|
| 0 | 38 | 19, 23, 8 | 3 |
| 1 | 97 | 38, 19, 23, 47, 41, 8 | 6 |
| 2 | 19 | 8 | 1 |
| 3 | 102 | 38, 97, 19, 23, 47, 41, 8 | 7 |
| 4 | 23 | 19, 8 | 2 |
| 5 | 47 | 38, 19, 23, 41, 8 | 5 |
| 6 | 41 | 38, 19, 23, 8 | 4 |
| 7 | 8 | - | 0 |

Análise do programa

```
// Programa p18.c
#include <stdio.h>
#include <stdlib.h>

#define MAX_TAM 50

int random(int n)
{
    return rand() % n;
}

int existe(int n, int v[], int k)
{
    int i;

    for (i = 0; i < n; i++)
        if (v[i] == k)
            return 1;
    return 0;
}

int main(int args, char * arg[])
{
    int i, j, n, r;
    int vet[MAX_TAM], pos[MAX_TAM], v_ord[MAX_TAM];

    // Inicializar gerador de números aleatórios
    srand( (unsigned) time(NULL) );
```

Veja o parâmetro **v[]** !!!



Esta função verifica se existe no vetor **v** (que tem **n** elementos) um elemento igual a **k**. Se existir, retorna **1**. Do contrário, retorna **0**.

Análise do programa

```
// Gerar o vetor inicial
n = random(MAX_TAM)+1;           // tamanho do vetor
i = 0;
do
{
    r = random(100)+1;
    if (existe(i,vet,r) == 0) ←
    {
        vet[i] = r; ←
        i++;
    }
}
while (i < n);
printf("Vetor inicial:\n");
for (i = 0; i < n; i++)
    printf("%d ",vet[i]);
printf("\n\n");
```

.Observe que um valor **r** gerado aleatoriamente, será incluído no vetor **v** somente se ele ainda não estiver no vetor.

Análise do programa

```
// Determinar posicao no vetor ordenado
for (i = 0; i < n; i++)
{
    pos[i] = 0;
    for (j = 0; j < n; j++)
        if (vet[i] > vet[j])
            pos[i]++;
    v_ord[pos[i]] = vet[i];
}

// Exibir o vetor ordenado
printf("Vetor ordenado:\n");
for (i = 0; i < n; i++)
    printf("%d ", v_ord[i]);
printf("\n\n");
system("PAUSE");
return 0;
}
```

Análise do programa

- Função **existe**:
Usada para evitar a geração de números de cheques repetidos.
- Observe que a variável indexada parâmetros como **int v[]**, ou seja, não se especificou o número de posições.
- Isto é possível somente em **listas de parâmetros de funções**.
- Neste caso, o **número de posições** da variável indexada declarada como parâmetro na função corresponde ao valor de um **outro parâmetro** (n).

```
int existe(int n, int v[],int k)
{
    int i;

    for (i = 0; i < n; i++)
        if (v[i] == k)
            return 1;
    return 0;
}
```

Vetores e matrizes

- Uma variável indexada pode ter uma ou mais **dimensões**. Considere, por exemplo, as declarações:

```
int a[10];  
char b[3][5];  
double c[2][2][3];
```

- Dizemos que **a** tem uma dimensão, **b** tem duas dimensões e **c** tem três dimensões.
- Para as declarações anteriores, temos:

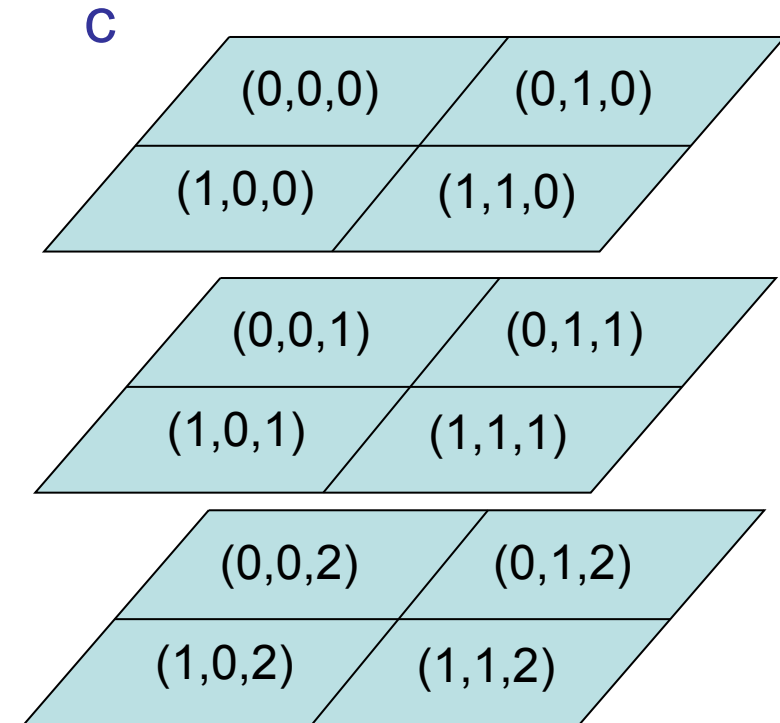
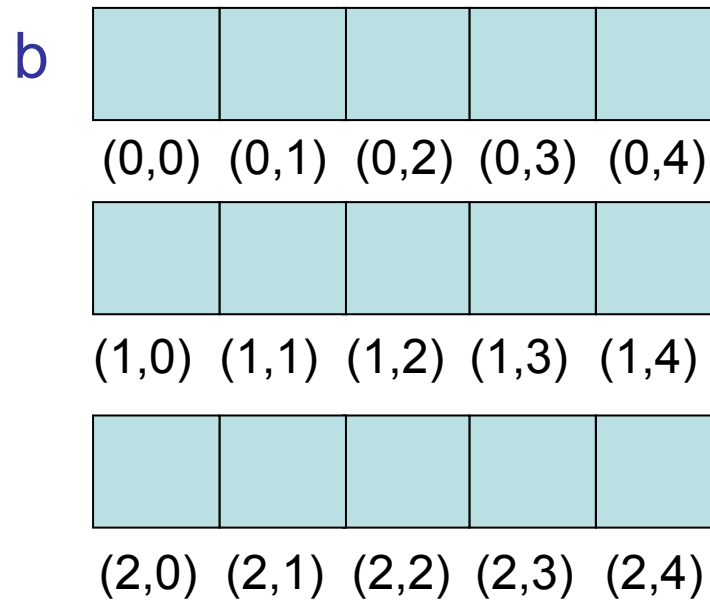
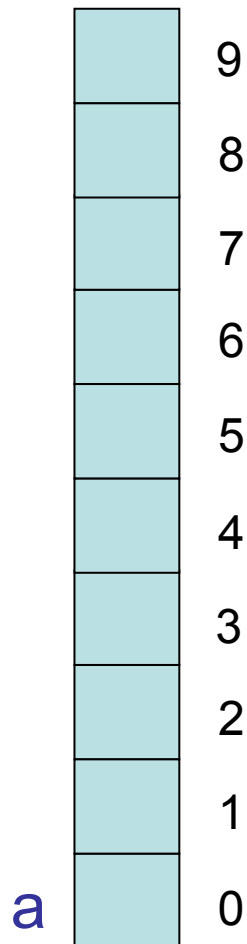
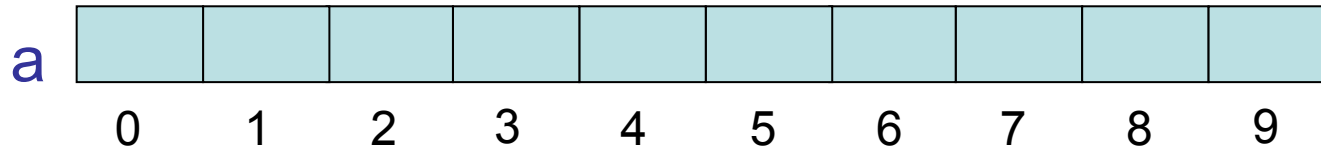
| Variável | Nº de posições | Memória alocada (bytes) | Exemplo de referência |
|----------|------------------|----------------------------|--------------------------|
| a | 10 | $10 * 4 = 40$ | <code>a[i]</code> |
| b | $3 * 5 = 15$ | $15 * 1 = 15$ | <code>b[i][j]</code> |
| c | $2 * 2 * 3 = 12$ | $12 * 8 = 96$ | <code>c[i][j][k]</code> |

Vetores e matrizes

- Em programação, os termos **vetor** e **matriz** são usados como sinônimos de variável indexada.
- O termo **vetor** é usado para uma variável indexada de uma única dimensão e o termo **matriz** para variáveis indexadas de duas ou mais dimensões.
- Para uma melhor compreensão das variáveis indexadas, utiliza-se **abstrações** a respeito da **disposição espacial** de seus elementos.

Vetores e matrizes

Considere as variáveis $a[10]$, $b[3][5]$ e $c[2][2][3]$



Problema 3

- Construa uma matriz para armazenar os resultados da simulação da rolagem de dois dados. O elemento $[i][j]$ da matriz armazena o número de vezes que o valor do primeiro dado é i e o valor do segundo dado é j .
- Considere que um vetor armazena a frequência de cada soma possível dos valores dos dados. Determinar qual é a soma mais frequente após rolar os dados 36.000 vezes. Imprima a matriz que armazena os resultados das rolagens dos dados.

Análise preliminar do problema

- Pergunta:
 - Quais são as somas possíveis para os valores dos dados?
- Resposta: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ou seja, existem 11 possibilidades.
- Ou seja, precisamos de um **vetor de 11 posições** para armazenar as frequências das somas acima.

Exemplo:

| | | | | | | | | | | |
|---|-----|----|----|---|---|---|----|-----|----|----|
| 3 | 125 | 23 | 41 | 5 | 9 | 2 | 86 | 231 | 45 | 13 |
|---|-----|----|----|---|---|---|----|-----|----|----|

0 1 2 3 4 5 6 7 8 9 10



Somas:

2 3 4 5 6 7 8 9 10 11 12

Análise do programa

```
int main(int args, char * arg[])
{
    int i,j,d1,d2;
    int rmf,val;
    int mat[6][6],res[11];

    // Inicializar gerador de números aleatórios
    srand( (unsigned) time(NULL) );

    // Zerar a matriz de resultados e o vetor de somas
    for (i = 0; i < 6; i++)
        for (j = 0; j < 6; j++)
            mat[i][j] = 0;
    for (i = 0; i < 11; i++)
        res[i] = 0;

    // Simular os jogos de dados
    for (i = 0; i < 36000; i++)
    {
        d1 = random(6); // dado 1
        d2 = random(6); // dado 2
        mat[d1][d2]++;
        res[d1+d2]++;
    }
}
```

.Observe que:

- .
- .O elemento `mat[0][0]` armazena quantas vezes o resultados foi:
 - . dado1 = 1
 - . dado2 = 1.
- .
- .O elemento `mat[3][5]` armazena quantas vezes o resultados foi:
 - . dado1 = 4
 - . dado2 = 6.
- .
- .E assim por diante...

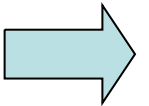
Análise do programa

```
// Exibir a matriz de resultados
printf("Matriz de resultados:\n");
for (i = 0; i < 6; i++)
{
    for (j = 0; j < 6; j++)
        printf("%5d ", mat[i][j]);
    printf("\n");
}

// Exibir o vetor de somas
printf("Vetor de somas:\n");
for (i = 0; i < 11; i++)
    printf("%5d ", res[i]);
printf("\n");

// Determinar a soma mais frequente
val = 0;
for (i = 0; i < 11; i++)
    if (res[i] > val)
    {
        val = res[i];
        rmf = i+2;
    }
printf("\nResultado mais frequente: %d\n", rmf);
system("PAUSE");
return 0;
}
```


← Entendeu?



Análise do programa

- Lembre-se que o valor da soma corresponde ao **valor do índice do vetor + 2**.

| | | | | | | | | | | | |
|---------------|---|-----|----|----|---|---|---|----|-----|----|----|
| res | 3 | 125 | 23 | 41 | 5 | 9 | 2 | 86 | 231 | 45 | 13 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Somas: | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

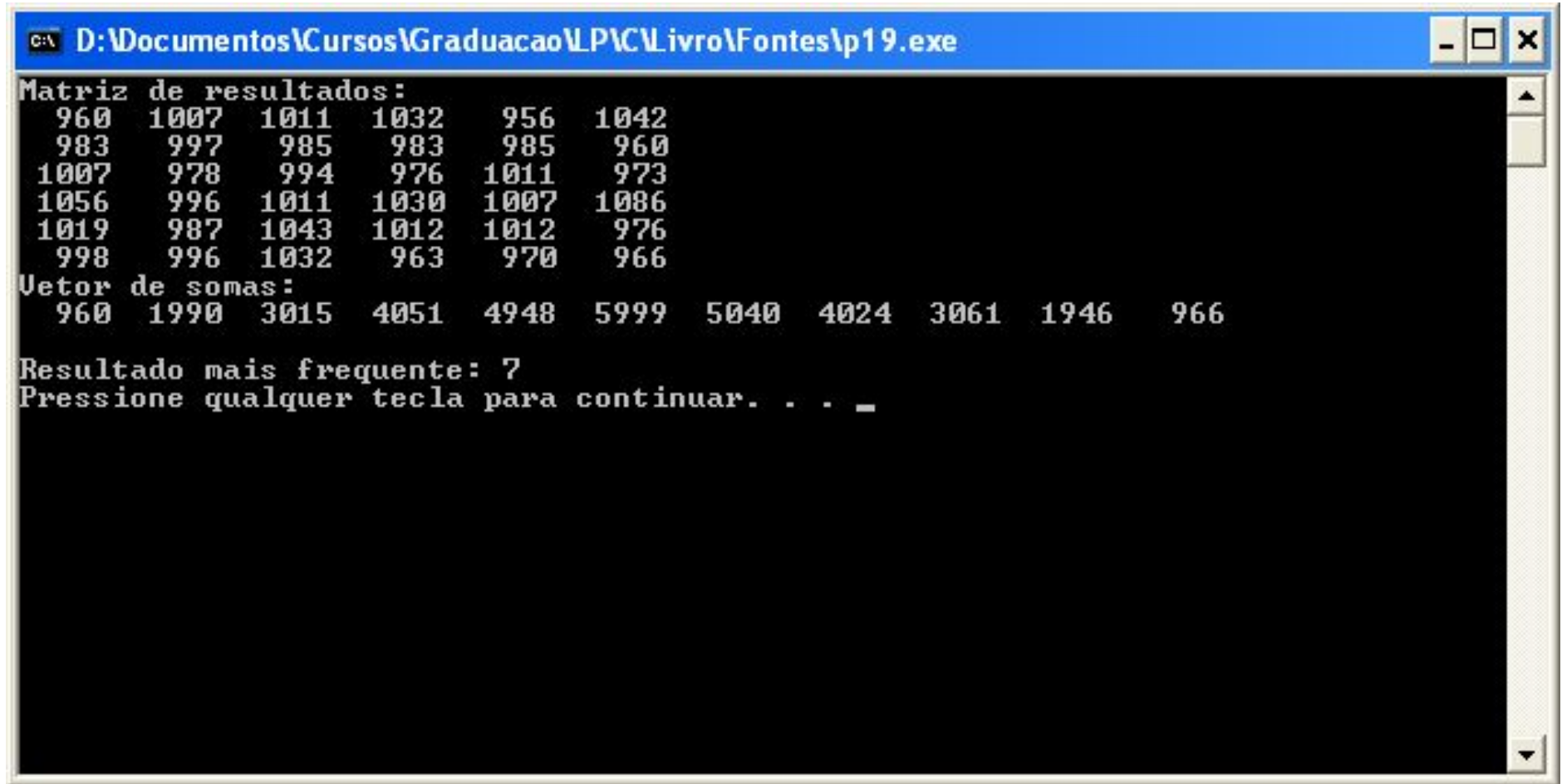


- Por isso, se a soma mais freqüente está em **res[i]**, o resultado (soma) mais freqüente será **i+2**:

```
if (res[i] > val)
{
    val = res[i];
    rmf = i + 2;
}
```

Análise do programa

Resultado da execução:



```
C:\ D:\Documentos\Cursos\Graduacao\LP\CLivro\Fontes\p19.exe
Matriz de resultados:
 960 1007 1011 1032  956 1042
 983  997  985  983  985  960
1007  978  994  976 1011  973
1056  996 1011 1030 1007 1086
1019  987 1043 1012 1012  976
 998  996 1032  963  970  966
Vetor de somas:
 960 1990 3015 4051 4948 5999 5040 4024 3061 1946  966

Resultado mais frequente: 7
Pressione qualquer tecla para continuar. . . _
```