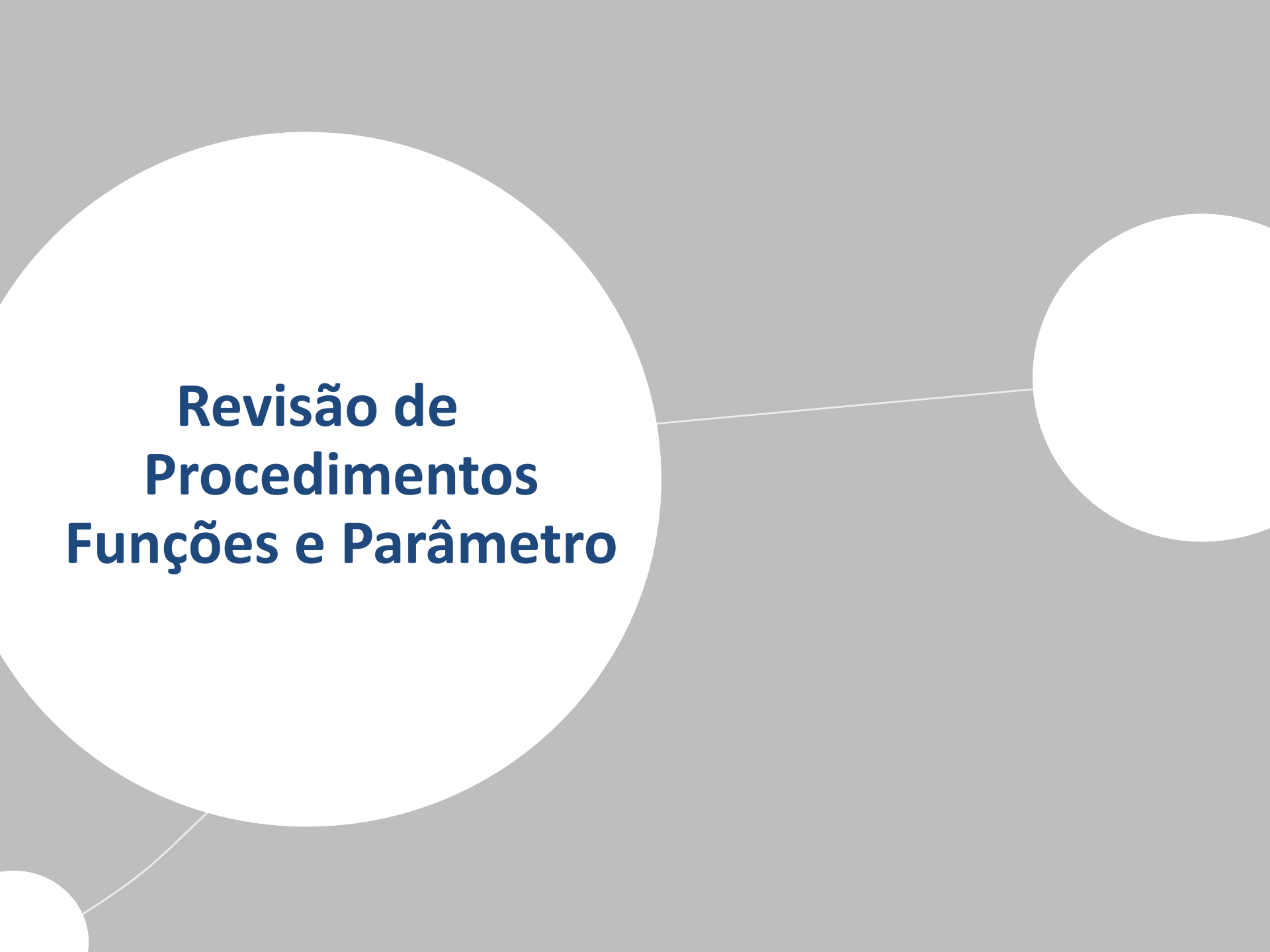




**Conteúdo
programático**

Conteúdo programático

- Revisão de Procedimentos Funções e Parâmetro;
- Algoritmos de ordenação:
 - ⇒ Trocas;
 - ⇒ Seleção
 - ⇒ Inserção;
- Algoritmos de pesquisa
 - ⇒ Sequencial;
 - ⇒ Binária;
- Fila;
- Pilha;
- Lista ligada;
- Outros;



Revisão de Procedimentos Funções e Parâmetro

Sub-Rotinas

- “Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado”;
- “Estes dizeres servem também para a construção de programas”;
- “Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema”;

■ Vantagens:

- ⇒ Subdivisão de algoritmos complexos, facilitando o seu entendimento;
- ⇒ Estruturação de algoritmos, facilitando, principalmente, a detecção de erros e a documentação de sistemas;
- ⇒ Modularização de sistemas que facilita a manutenção de software e a reutilização de subalgoritmos já implementados;
- ⇒ Garante maior legibilidade;

Procedimento (void)

- Executa uma sequência de passos, mas não retorna nenhum valor.

```
#include <stdio.h>
#include <stdlib.h>
int n1,n2;
void mostra()
{
    if (n1==n2)
        printf("\nNumero iguais....\n");
    else if(n1>n2)
        printf("\nPrimeiro numero e o maior....\n\n");
    else
        printf("\nSegundo numero e o maior....\n\n");
}
void recebe()
{
    printf("Entre com o primeiro numero:... ");
    scanf("%d",&n1);
    printf("Entre com o segundo numero:... ");
    scanf("%d",&n2);
}
main()
{
    system("cls");
    printf("*****\n");
    printf("*           E X E M P L O   D E   P R O C E\n");
    printf("*****\n");
    recebe();
    mostra();
    system("pause");
}
```

Função

- Executa uma sequência de passos, e sempre retorna um valor;

```
# include <stdio.h>
# include <stdlib.h>

float a,b,c;

float prod ( ) {
    return a * b ;
}

int main ( ) {
    a = 2;
    b = 110;
    c = prod() ;
    printf ( "%f\n\n",c ) ;
    system ( "pause" ) ;
    return 0 ;
}
```

Função com parâmetro

- Para que a função seja executada, primeiro ela precisa receber alguma(s) informação(ções), essa(s) informação(ões) é (são) chamada(s) de parâmetro;

```
# include <stdio.h>
# include <stdlib.h>

float prod ( float x, float y ) {
    return x * y ;
}

int main ( ) {
    float a, b, c;
    a = 2;
    b = 110;
    c = prod ( a, b ) ;
    printf ( "%f\n\n",c ) ;
    system ( "pause" ) ;
    return 0 ;
}
```


Parâmetro por valor e por referência

• PARÂMETRO POR VALOR

```
# include <stdio.h>
# include <stdlib.h>
```

```
int x;
void porvalor ( int a ) ;
```

```
int main ( ) {
    x = 10;
    porvalor ( x ) ;
    printf ( "x = %d\n\n", x ) ;
    system ( "pause" ) ;
    return 0 ;
}
```

```
void porvalor ( int a ) {
    a = 5 ;
    printf ( "x = %d      a = %d\n\n", x, a ) ;
}
```

• PARÂMETRO POR REFERÊNCIA

```
# include <stdio.h>
# include <stdlib.h>
```

```
int x;
void por_ref ( int *a ) ;
```

```
int main ( ) {
    x = 10;
    por_ref ( &x ) ;
    printf ( "x = %d\n\n", x ) ;
    system ( "pause" ) ;
    return 0 ;
}
```

```
void por_ref ( int *a ) {
    *a = 5 ;
    printf ( "x = %d      a = %d\n\n", x, *a ) ;
}
```



Ordenação

- Por que ordenar?

⇒ “Ordenar os dados também pode ser uma etapa preliminar para procurá-los, que é muito mais rápido do que uma procura linear”;

Ordenação para dois valores

```
#include <stdlib.h>
#include <stdio.h>

main()
{
    int a, b;
    printf("Entre com o valor de a: ");
    scanf("%d", &a);
    printf("Entre com o valor de b: ");
    scanf("%d", &b);

    if (a==b)
        printf("Valores iguais\n");
    else
        if (a < b)
            printf("%d, %d\n", a,b);
        else
            printf("%d, %d\n", b,a);

    system("pause");
}
```

- Desenvolva um programa que seja capaz de ordenar três valores inteiros quaisquer digitados.

Ordenação para três valores

```
#include <stdlib.h>
#include <stdio.h>

main()
{
    int a, b, c;
    printf("Entre com o valor de a:");
    scanf("%d",&a);
    printf("Entre com o valor de b:");
    scanf("%d",&b);
    printf("Entre com o valor de c:");
    scanf("%d",&c);

    if((a==b) && (a==c))
        printf("Valores iguais\n");
    else
        if ((a <= b) && (a <= c))
        {
            printf("%d, ", a);
            if (b < c)
                printf("%d, %d\n ",b,c);
            else
                printf("%d, %d\n ",c,b);
        }
        else
        {
            if ((b <= a) && (b <= c))
            {
                printf("%d, ", b);
                if (a < c)
                    printf("%d, %d\n ",a,c);
                else
                    printf("%d, %d\n ",c,a);
            }
            else
            {
                printf("%d, ", c);
                if (a < b)
                    printf("%d, %d\n ",a,b);
                else
                    printf("%d, %d\n ",b,a);
            }
        }
    system("pause");
}
```



Ordenação de Vetor Bubble Sort

Ordenação por trocas (Bubble Sort)

- Também conhecido como "método da bolha" ("bubble sort") é um dos métodos mais simples de ordenação.
- Compara dois elementos consecutivos de um vetor e se o da esquerda é maior que o da direita trocam de posição. Quando existem trocas, os elementos maiores tendem a deslocar-se para a direita e os menores para a esquerda.
- Consiste em percorrer o vetor, comparando-se cada elemento do vetor com o elemento imediatamente seguinte ($V[\text{índice}]$ com $V[\text{índice} + 1]$).

6	2	1	3	<u>4</u>	5	8	7	0
---	---	---	---	----------	---	---	---	---

- <https://www.youtube.com/watch?v=llX2SpDkQDc>
- <https://www.youtube.com/watch?v=lyZQPjUT5B4>

Ordenação por trocas (Bubble Sort)

```
void bsort(int vet[], int t)
{
    int i, j, k=0;

    for (i=0; i<t-1; i++)
    {
        for (j=0; j<t-(i+1); j++)
        {
            if (vet[j] > vet[j+1])
            {
                k=vet[j];
                vet[j]=vet[j+1];
                vet[j+1]=k;
            }
        }
    }
}
```

vet – recebe o vetor que será ordenado;

t – recebe a quantidade de elementos do vetor;

i – determina o número de etapas para ordenação;

j – determina o número de comparações em cada etapa e os índices a serem pesquisados para a comparação;

k – variável auxiliar para ajudar na troca de posição dos valores no vetor;

Ordenação por trocas (Bubble Sort)

```
void bsort(int vet[], int t)
{
    int i,j,k=0;

    for (i=0;i<t-1;i++)
    {
        for (j=0;j<t-(i+1);j++)
        {
            if (vet[j] > vet[j+1])
            {
                k=vet[j];
                vet[j]=vet[j+1];
                vet[j+1]=k;
            }
        }
    }
}
```

[illegible]

- 1 – Desenvolva um programa que ordene um vetor de 10 elementos de números inteiros, mostre o vetor ordenado na tela, e também o maior e o menor elemento desse vetor;
- 2 – Desenvolva um programa que receba o nome de 5 alunos, e mostre os nomes em ordem crescente na tela;
- Para ambos os exercícios utilizar o método de ordenação por trocas;



Ordenação de Vetor

Selection Sort

Ordenação por seleção (Selection Sort)

Este método consiste em selecionar o menor valor do vetor e movê-lo para o primeiro índice, a seguir, seleciona-se o próximo menor valor, desconsiderando o já colocado na primeira posição.

6	2	1	3	<u>4</u>	5	8	7	0
---	---	---	---	----------	---	---	---	---

- <http://www.youtube.com/watch?v=BSXlolKg5F8>

Ordenação por seleção (Selection Sort)

```
void ssort ( int v[ ], int t )
{
    int i, j, min, k ;
    for ( i = 0; i < ( t - 1 ) ; i++ )
    {
        min = i ;
        for ( j = ( i + 1 ) ; j < t ; j++ )
        {
            if ( v[ j ] < v[ min ] )
                min = j;
        }
        if ( i != min )
        {
            k = v[ i ];
            v[ i ] = v[ min ];
            v[ min ] = k;
        }
    }
}
```

v – recebe o vetor que será ordenado;

t – recebe a quantidade de elementos do vetor;

i – determina o número de etapas para ordenação;

j – determina o número de comparações em cada etapa e os índices a serem pesquisados para a comparação;

k – variável auxiliar para ajudar na troca de posição dos valores no vetor;

min – armazena o índice que contém o menor valor do vetor e o índice a ser pesquisado para a comparação;

Ordenação por seleção (Selection Sort)

```
void ssort ( int v[ ], int t )
{
    int i, j, min, k ;
    for ( i = 0; i < ( t - 1 ) ; i++ )
    {
        min = i ;
        for ( j = ( i + 1 ) ; j < t ; j++ )
        {
            if ( v[ j ] < v[ min ] )
                min = j;
        }
        if ( i != min )
        {
            k = v[ i ];
            v[ i ] = v[ min ];
            v[ min ] = k;
        }
    }
}
```

[illegible]



Ordenação de Vetor

Insertion Sort

Ordenação por inserção (Insertion Sort)

Este método consiste em inserir cada um dos elementos em sua posição correta relativa à sequência ordenada.

6	2	1	3	<u>4</u>	5	8	7	0
---	---	---	---	----------	---	---	---	---

- <https://www.youtube.com/watch?v=-Z00it6Nkz8>

Ordenação por inserção (Insertion Sort)

```
void isort(int v[], int t)
{
    int i, j, k;
    for (i = 1; i < t; i++)
    {
        k = v[i];
        j = i-1;
        while (j >= 0 && v[j] > k)
        {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = k;
    }
}
```

[illegible]

Ordenação por inserção (Insertion Sort)

```
void isort(int v[], int t)
{
    int i, j, k;
    for (i = 1; i < t; i++)
    {
        k = v[i];
        j = i-1;
        while (j >= 0 && v[j] > k)
        {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = k;
    }
}
```

v – recebe o vetor que será ordenado;

t – recebe a quantidade de elementos do vetor;

i – determina o número de etapas para ordenação;

j – determina o número de comparações em cada etapa e os índices a serem pesquisados para a comparação;

k – variável auxiliar para ajudar na troca de posição dos valores no vetor;

- 1 – Desenvolva um programa que contenha três vetores que devem ser preenchidos com 10 números cada um. Crie um procedimento que ordene o primeiro vetor utilizando o método de trocas, um procedimento que ordene o segundo vetor utilizando o método de seleção e um procedimento que ordene o terceiro vetor utilizando o método de inserção. Ao final mostre o conteúdo dos três vetores na tela;
- 2 – Refaça o exercício anterior mudando os vetores para que contenha 5 nomes cada um;



Pesquisa Sequencial

Lembram-se dessa pergunta?

- Por que ordenar?

⇒ “Ordenar os dados também pode ser uma etapa preliminar para procurá-los, que é muito mais rápido do que um procura linear”;

Pesquisa Sequencial

- Uma forma simples de procurar consiste em:
 - Começar no início do vetor; comparar sucessivamente o dado procurado com o dado do vetor; repetir o processo até encontrar o dado ou atingir o fim do vetor.
- Como percorremos sequencialmente o vetor, a este algoritmo chamamos pesquisa seqüencial.
- Exemplo: Pretende-se saber qual a posição que o valor 75 (se existir) ocupa num vetor de números.
 - Comparo 75 com 10 => São diferentes => Incremento a posição.
 - Comparo 75 com 20 => São diferentes => Incremento a posição.
 - Comparo 75 com 75 => São iguais => 75 ocupa a 3ª posição no vetor.

V [0]	V [1]	V [2]	V [3]	V [4]	V [5]
10	20	75	9	4	3

- A busca será mais rápida se o vetor estiver com seus elementos ordenados.
- O tempo de busca pode ser reduzido significativamente devido a relação entre os elementos da lista.
- Será feita a busca pelo valor V em um conjunto de N elementos ordenados.
- O procedimento retorna o valor da posição no vetor se encontrar V e -1 (um negativo) caso não encontre.

- Buscar o valor 5 no vetor;
- Vetor desordenado:

V [0]	V [1]	V [2]	V [3]	V [4]	V [5]
10	20	75	9	4	3

- Vetor Ordenado:

V [0]	V [1]	V [2]	V [3]	V [4]	V [5]
3	4	9	10	20	75

Pesquisa Sequencial

Vetor Desordenado

```
int pesqseq(int v[], int busca)
{
    int i;
    for (i = 0; i <= MAX; i++)
        if (v[i] == busca) return i;
    return -1;
}
```

Vetor Ordenado

```
int pesqseq(int v[], int busca)
{
    int i;
    for (i = 0; i <= MAX; i++)
    {
        if (v[i] == busca)
            return i;
        else
            if (v[i] > busca)
                return -1;
    }
    return -1;
}
```

Pesquisa Sequencial

// função para ordenar o vetor

```
void bsort(char vet[ ], int t)
{
    int i, j;
    char k;
    for (i = 0; i < t - 1; i++)
    {
        for (j = 0; j < t - (i + 1); j++)
        {
            if (vet[ j ] > vet[ j + 1 ])
            {
                k = vet[ j ];
                vet[ j ] = vet[ j + 1 ];
                vet[ j + 1 ] = k;
            }
        }
    }
}
```

// função de busca no vetor

```
int pesqseq(char v[ ], char busca)
{
    int i;
    for (i = 0; i <= MAX; i++)
    {
        if (v[ i ] == busca)
            return i;
        else
            if (v[ i ] > busca)
                return -1;
    }
    return -1;
}
```

Pesquisa Sequencial

- Simular a busca sequencial do vetor **ORDENADO** Nomes que contém os seguintes dados: Maria, Sandra, André, Mario, Dirce, Sandro, para os seguintes dados:

a) André

Pesqseq = _____

b) Marcos

Pesqseq = _____

c) Ana

Pesqseq = _____

d) Sandra

Pesqseq = _____

e) Tânia

Pesqseq = _____

i	V [0]	v [1]	V [2]	V [3]	V [4]	V [5]

- 1 – Desenvolva um programa que receba 20 números inteiros quaisquer digitados pelo usuário. Ao final peça para ele digitar outro número e usando a função “Pesquisa Sequencial Ordenada” informe em que posição do vetor o número se encontra e ao final mostre o vetor ordenado;



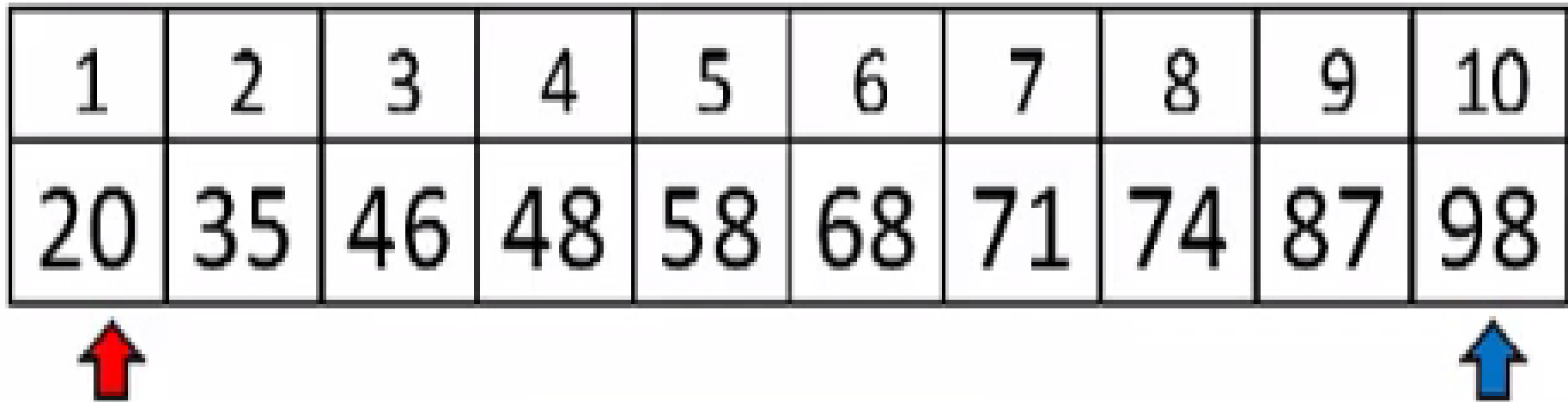
Pesquisa Binária

Pesquisa Binária

- ❑ A pesquisa binária só é aplicável a vetor ordenados!
- ❑ Deverá ser comparado o dado que se encontra a meio do vetor com o dado procurado, podendo acontecer uma de três coisas:
 - ✓ é igual ao dado procurado \Rightarrow está encontrado
 - ✓ é maior do que o dado procurado \Rightarrow continuar a procurar (do mesmo modo) no sub-vetor à esquerda da posição inspecionada
 - ✓ é menor do que o dado procurado \Rightarrow continuar a procurar (do mesmo modo) no sub-vetor à direita da posição inspecionada.
- ❑ Em outras palavras: Consideramos primeiro o elemento do meio da tabela.
- ❑ Se o dado deste elemento é maior do que o dado procurado, podemos garantir que o procurado não se encontra na 2ª metade da tabela.
- ❑ Se o vetor a ser inspecionado se reduzir a um vetor vazio, conclui-se que o dado procurado não existe no vetor inicial.

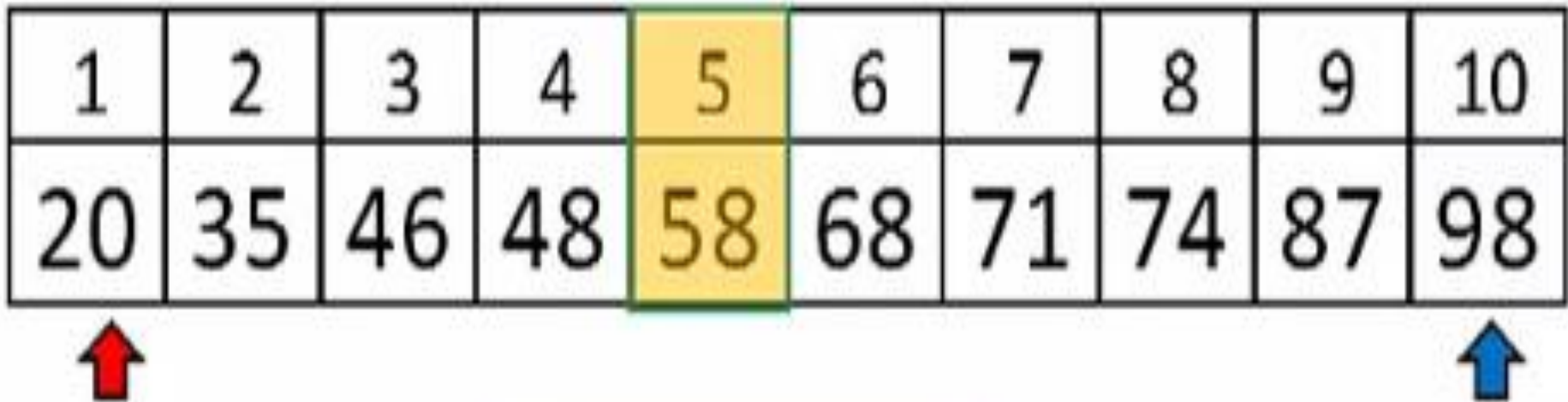
- Valor procurado 71

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



- Valor procurado 71

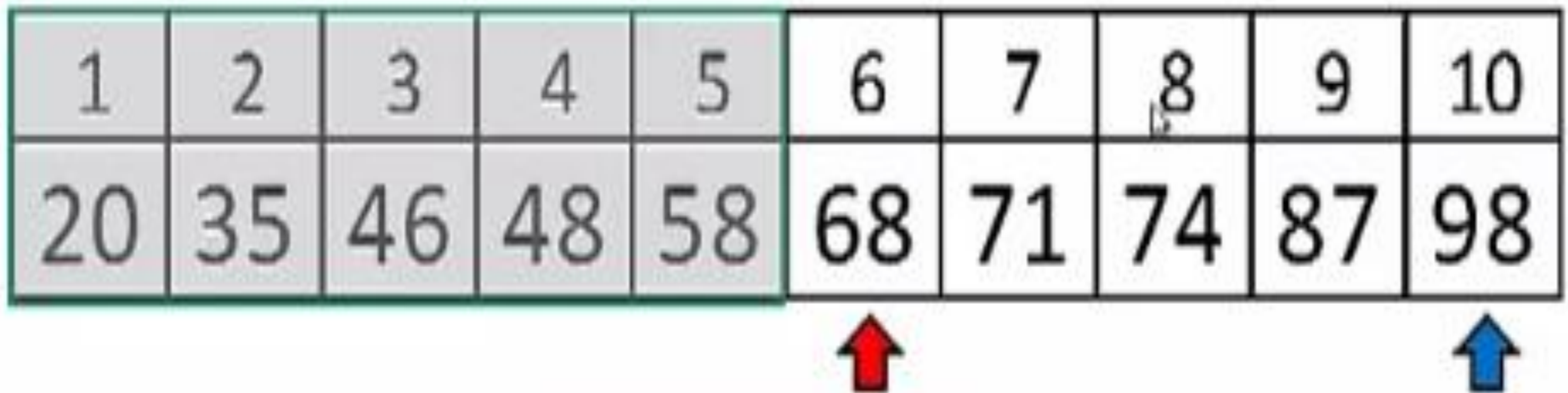
1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



A diagram illustrating a binary search step. A table of 10 sorted numbers is shown. The 5th element (58) is highlighted in yellow, indicating it is the current pivot. A red arrow points to the first element (20) and a blue arrow points to the last element (98), representing the search range boundaries.

- Valor procurado 71

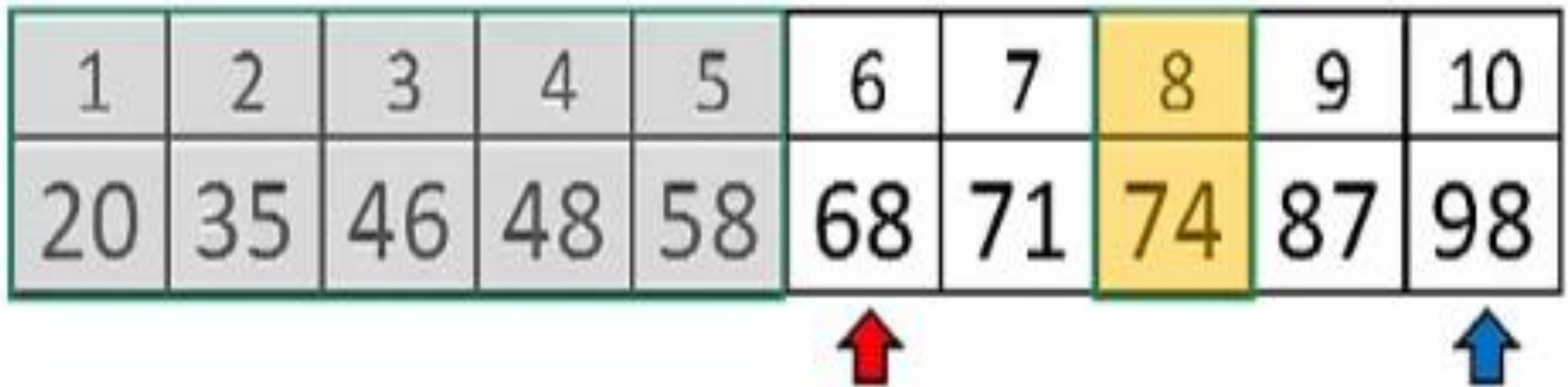
1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



The diagram illustrates a binary search step. A table of 10 sorted numbers is shown. The first five elements (20, 35, 46, 48, 58) are highlighted with a green border. A red arrow points to the middle element (68) at index 6, and a blue arrow points to the last element (98) at index 10.

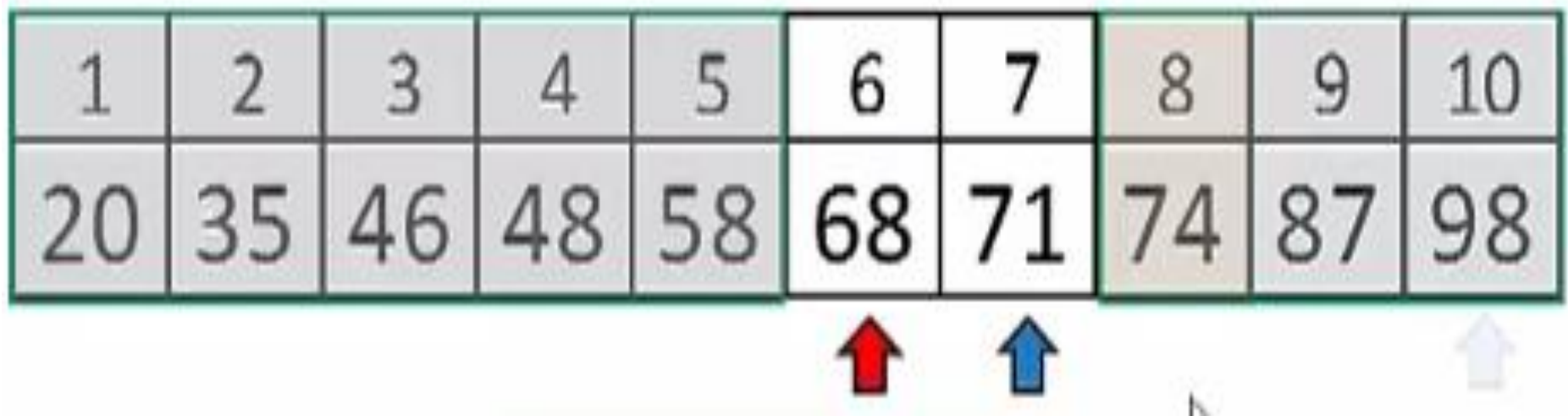
- Valor procurado 71

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



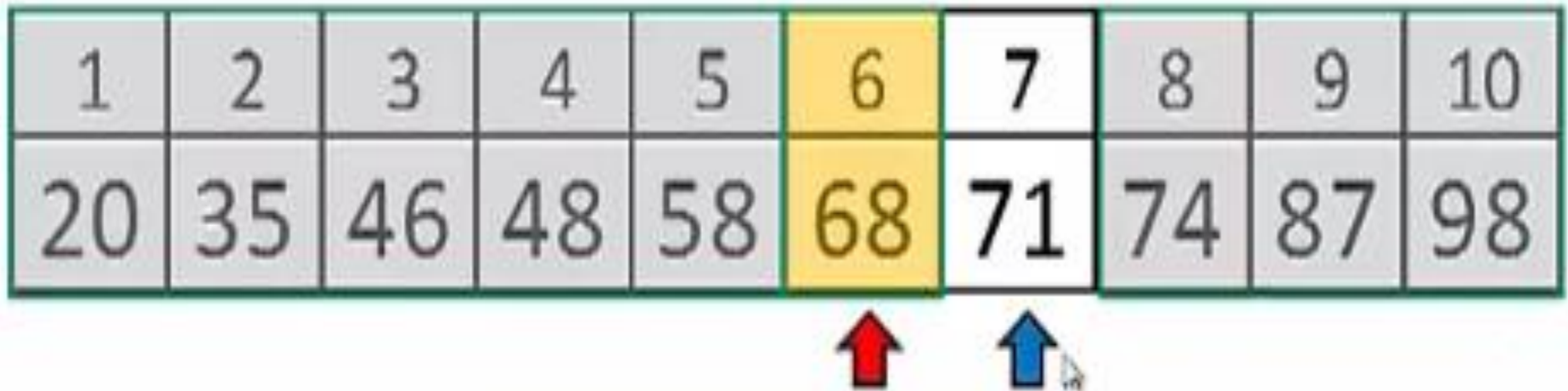
- Valor procurado 71

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



- Valor procurado 71

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



The diagram illustrates a binary search step. A table of 10 sorted numbers is shown. The 6th element (68) is highlighted in yellow, and a red arrow points to it. The 7th element (71) is the target value, and a blue arrow points to it.

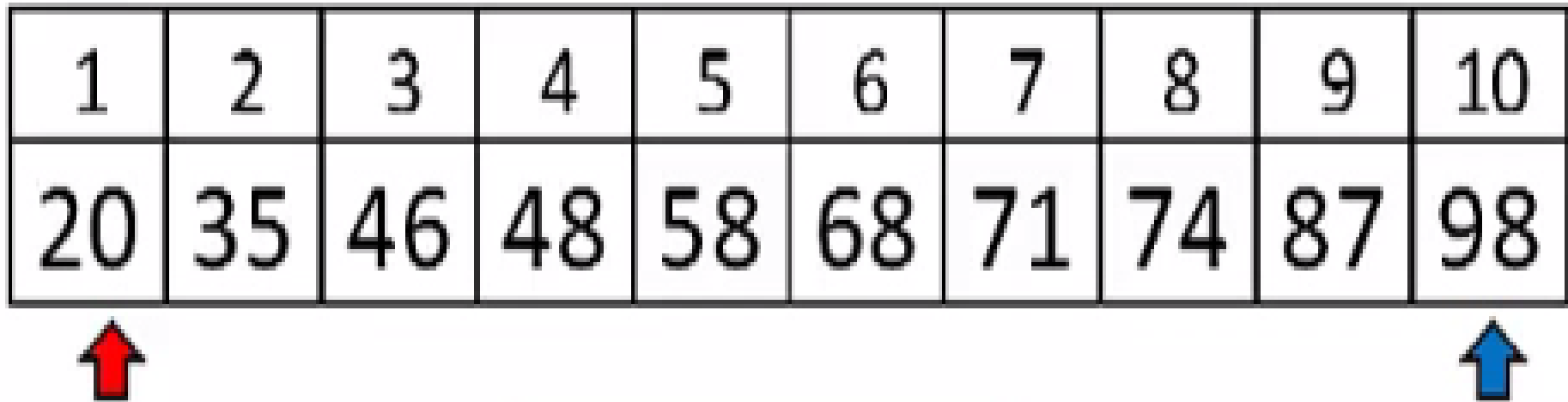
- Valor procurado 71
- 4 passos para achar o valor

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



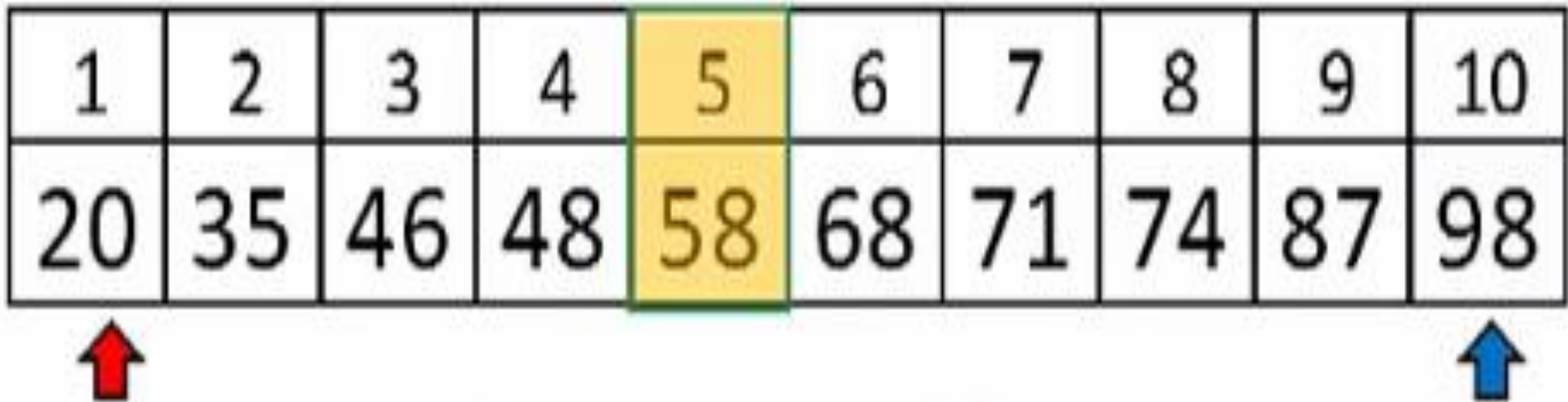
- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98

↑ ↑ 37

- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98

↑ ↑


- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98

↑ ↑

- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



- Valor procurado 37

1	2	3	4	5	6	7	8	9	10
20	35	46	48	58	68	71	74	87	98



```
int pesqbin(int v[], int busca)
{
    int inicio, fim, meio;
    inicio = 0;
    fim = MAX;
    while (inicio <= fim)
    {
        meio = ( inicio + fim ) / 2 ;
        if (v[meio]==busca)
            return meio;
        if (busca<v[meio])
            fim = meio-1;
        else
            inicio = meio + 1;
    }
    return -1;
}
```

Pesquisa Binária – Exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#define MAX 15

//protótipos das funções
void bsort ( char v [ ], int qtd );
int pesqbin ( char v [ ], char busca );

int main ( ) {
    setlocale (LC_ALL, "");
    int i, resp;
    char pesq;
    char vet [ MAX ] = { 'o', 'c', 'v', 's', 'n', 'r', 'p', 'e', 'm', 'j' };
    bsort ( vet, MAX );
    for(i=0;i<MAX;i++){
        printf("%c\n", vet[i]);
    }
    printf ( " Dado a ser pesquisado: " );
    scanf ( "%c", &pesq );
    resp = pesqbin ( vet, pesq );
```

Pesquisa Binária – Exemplo

```
if ( resp >= 0 )
    printf( "%c está na posição %d.\n\n", pesq, resp);
else
    printf ( "%c não está no vetor.\n\n ", pesq );
system ( "pause" );
return 0;
}
```

```
void bsort ( char v [ ], int qtd ) {
    int i, j;
    char k;
    for ( i = 0; i < qtd - 1; i++ ) {
        for ( j = 0; j < qtd - ( i + 1 ); j++ ) {
            if ( v [ j ] > v [ j + 1 ] ) {
                k = v [ j ];
                v [ j ] = v [ j + 1 ];
                v [ j + 1 ] = k;
            }
        }
    }
}
```

Pesquisa Binária – Exemplo

```
int pesqbin (char v [ ], char busca) {  
    int inicio, fim, meio;  
    inicio = 0;  
    fim = MAX - 1;  
    while ( inicio <= fim ) {  
        meio = ( inicio + fim ) / 2;  
        if (v [ meio ] == busca )  
            return meio;  
        if ( busca < v [ meio ] )  
            fim = meio - 1;  
        else  
            inicio = meio + 1;  
    }  
    return -1 ;  
}
```




Conclusão

- Para pesquisar dados em um vetor com 200.000 elementos a pesquisa binária faz no máximo +/- 18 comparações;
- Para o mesmo vetor, a pesquisa sequencial pode necessitar de 200.000 comparações;
- Dobrando o vetor de tamanho, a pesquisa binária fará no máximo 19 comparações (uma a mais), enquanto que a pesquisa sequencial poderá fazer 400.000 comparações;

- 1 – Desenvolva um programa que receba 20 números inteiros quaisquer digitados pelo usuário. Ao final peça para ele digitar outro número e usando a função “Pesquisa Binária” informe em que posição do vetor o número se encontra e ao final mostre o vetor ordenado;

- 2 – Desenvolva um programa que receba 20 números inteiros quaisquer digitados pelo usuário. Ao final peça para ele digitar outro número para realizar a busca e qual algoritmo de busca ele deseja usar: 1 – Pesquisa Sequencial, 2 – Pesquisa Binária. O programa deverá usar o algoritmo selecionado para realizar a busca. Ao final informe em que posição do vetor o número se encontra e mostre o vetor ordenado;



FILA

Fila - Teoria das Filas

- Uma Fila é um tipo especial de lista linear em que as inserções são realizadas num extremo, ficando as remoções restritas ao outro.
- O extremo onde os elementos são inseridos é denominado final da fila, e aquele de onde são removidos é denominado começo da fila.
- Cada vez que uma operação de inserção é executada, um novo elemento é colocado no final da fila.
- Na remoção, é sempre retornado o elemento que aguarda há mais tempo na fila, ou seja aquele posicionado no começo.
- A ordem de saída corresponde diretamente à ordem de entrada dos elementos na fila de modo que só primeiros elementos que entram são os primeiros a sair.
- Em vista disto, as filas são denominadas listas FIFO (First-In/First-Out).

Filas e Pilhas

conceito

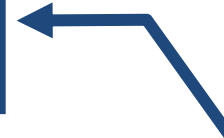
Início da FILA



Primeiro que sai
(FIFO)

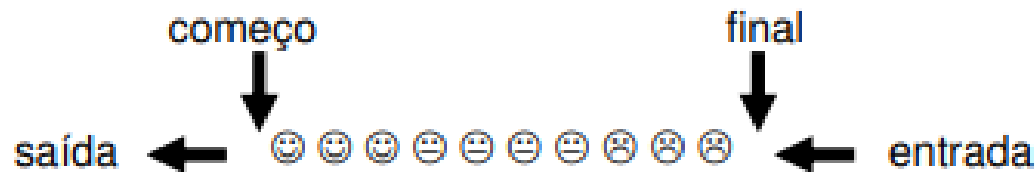


Primeiro
que entra



Fila - Teoria das Filas

- Uma fila funciona como a fila do cinema:
- a primeira pessoa a entrar na fila é a primeira pessoa a comprar o bilhete.
- A última pessoa a entrar na fila é a última pessoa a comprar o bilhete (ou - se já estiver esgotado - a não comprar o bilhete).



- Há diversas filas fazendo seu trabalho silenciosamente no sistema operacional de seu computador.
- Há uma fila de impressão onde seus trabalhos para impressão esperam até a impressora estar disponível.
- Uma fila também armazena dados do pressionamento de tecla na medida em que você digita no teclado.

Provê modelos para prever o comportamento de sistemas que oferecem serviços para demandas com taxas de chegadas e saídas aleatórias.

Utilizada para modelar sistemas de controle de:

- **Atendimentos**
- **Espera**
- **Saídas**

Exemplos:

- **Sistema telefônico**
- **Sistemas de comunicação de dados**
- **Sistema de impressão**
- **Sistemas de atendimentos em geral**

Tempo de espera de um cliente:

Quanto tempo um cliente espera no banco

Quanto tempo um pacote passa em um roteador

Acúmulo de clientes na fila:

Qual o tamanho médio da fila do banco

Como a fila do roteador se comporta

Tempo ocioso/ocupado dos servidores:

Quanto tempo o caixa fica livre

Qual a utilização do roteador

Taxa de saída (vazão):

Quantos clientes são atendidos por hora

Quantos pacotes são encaminhados por segundo

Funções de manipulação de Filas

- **FUNÇÕES BÁSICAS**

- Seja F uma variável do tipo fila e X um elemento qualquer
enqueue (f, x) - Função que insere X no fim da fila F .

dequeue (f) - Função que remove o elemento do começo da fila F devolvendo o valor do para a rotina que a chamou.

- **FUNÇÕES AUXILIARES**

qinit(f) – Função que esvazia a fila F .

qisfull (f) – Função que retorna um valor lógico informando se a pilha está cheia. Verdadeiro se estiver cheia ou Falso caso contrário.

qisempty (f) – Função que retorna um valor lógico informando se a fila está vazia. Verdadeiro se estiver vazia ou Falso caso contrário.

- Para eliminar o erro lógico, que sinaliza fila vazia e cheia ao mesmo tempo, basta utilizar a implementações de fila circular, onde acrescentamos uma variável contadora para indicar quantos elementos estão armazenados na fila.

Implementação do arquivo FILAS.H

```
#define Max 50
```

```
typedef int tpElem;
```

```
//Estrutura de dados da fila
```

```
typedef struct
```

```
{
```

```
    int total,comeco,final;
```

```
    tpElem valor[Max];
```

```
}tpFila;
```

```
//protótipo das funções
```

```
void qinit(tpFila *f);      // iniciar a fila
```

```
int qisFull(tpFila *f);    //Verificar se a fila está cheia
```

```
int qisEmpty(tpFila *f);   // verificar se a fila está vazia
```

```
void enqueue(tpFila *f, tpElem x); //colocar um dado no fim da fila
```

```
tpElem dequeue (tpFila *f);      //retirar um dado no começo da fila
```

Implementação do arquivo FILAS.C

```
#include "filas.h"
```

```
int qisFull(tpFila *f)
{
    return (f->total==Max-1);
}
```

```
int qisEmpty(tpFila *f)
{
    return (f->total==0);
}
```

```
void adc (int *i)
{
    (*i)++;
    if(*i==Max)
        *i=0;
}
```

```
void enqueue(tpFila *f, tpElem x)
{
    if (qisFull(f)==0)
    {
        f->valor[f->final]=x;
        adc(&f->final);
        f->total++;
    }
}
```

```
    else
    {
        printf("\nFila Cheia \n\n");
        system("pause");
    }
}
```

```
tpElem dequeue (tpFila *f)
{
    tpElem x;
    if (qisEmpty(f)==0)
    {
        x=(f->valor[f->comeco]);
        adc(&f->comeco);
        f->total--;
        return x;
    }
    else
    {
        printf("\nFila Vazia \n\n");
        system("pause");
    }
}
```

Exemplo de Utilização de Filas (inteiro)

```
#include <stdlib.h>
#include <stdio.h>
#include "filas.h"

int main()
{
    int i;
    tpFila f;
    int n;
    int op;

    qinit(&f);

    while(op!=3)
    {
        system("cls");
        printf("\n1-Inserir na fila\n2-Mostrar fila\n3-Sair\n\n");
        scanf("%d",&op);

        if(op==1)
        {
            printf("Digite um numero qualquer: ");
            scanf("%d",&n);
            enqueue(&f,n);
        }
        else if(op==2)
```

```
{
    printf("\n\nValor desolvido pela fila =>");
    while(qisEmpty(&f)==0)
        printf("\n%d",dequeue(&f));

    printf("\n\n\n");
    system("pause");
}
}
return 0;
}
```

Exemplo de Utilização de Filas (caracter)

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "filas.h"

int main()
{
    int i;
    tpFila f;
    char palavra[30];
    printf("Digite uma String qualquer: ");
    gets(palavra);
    qinit(&f);

    for(i=0; i<strlen(palavra); i++)
        enqueue(&f, palavra[i]);
    printf("\n\nValor devolvido pela Fila => ");

    while(qisEmpty (&f)==0)
        printf("%c", dequeue (&f));
    printf("\n\n\n");
    system("pause");
    return 0;
}
```

EXERCÍCIOS: FILAS

```
# include <stdio.h>
# include <stdlib.h>
# include "filas.h"
```

```
int main ( ) {
    tpFila f ;
    char x, y, w, z ;
    qinit ( &f ) ;
    enqueue ( &f, 'a' ) ;
    enqueue ( &f, 'b' ) ;
    printf ( "%c\n", dequeue ( &f ) ) ;
    enqueue ( &f, 'c' ) ;
    enqueue ( &f, 'd' ) ;
    x = dequeue ( &f ) ;
    y = dequeue ( &f ) ;
    enqueue ( &f, 'e' ) ;
    w = dequeue ( &f ) ;
    enqueue ( &f, dequeue ( &f ) ) ;
    printf ( "%c\n", dequeue ( &f ) ) ;
    enqueue ( &f, 'f' ) ;
    z = dequeue ( &f ) ;
    enqueue ( &f, 'g' ) ;
    printf ( "%c\n", dequeue ( &f ) ) ;
}
```

1. Faça a simulação de uma fila F, inicialmente vazia, após a execução de cada um dos comandos do programa abaixo e mostre os valores impressos ao final.

```
printf ( "%c\n", x );
printf ( "%c\n", y );
printf ( "%c\n", w );
printf ( "%c\n", z );
printf ( "\n\n\n" );
system ( "pause" );
return 0;
```

3

[illegible]

Exercício parte 2

1. Implemente a seguinte função dentro do arquivo “fila.h”, chamando-a posteriormente na função principal (insira no menu de opções essa chamada)

```
void printqueue (tpFila *f)
{
    for(int i=f->comeco; i<f->final; i++)
    {
        printf("%d",f->valor[i]);
        if ((i+1) < f->final)
            printf(" - ");
    }
}
```

2. Crie um sistema de senha de atendimento. O sistema deve emitir uma nova senha e quando uma senha for atendida, a mesma deverá ser removida da fila.



PILHA

Filas e Pilhas

conceito

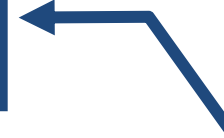
Início da FILA



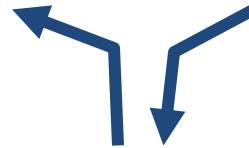
Primeiro que sai
(FIFO)



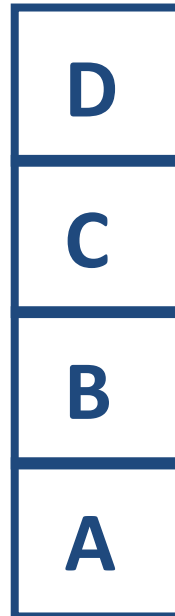
Primeiro
que entra



Último que entra
Primeiro que sai
(LIFO)



Topo da PILHA



- Uma pilha é uma lista linear em que apenas as operações de acesso, inserção e remoção são possíveis.
- Todas estas operações devem ser realizadas num mesmo extremo denominado topo.
- Devido às características das operações da pilha, o último elemento a ser inserido será o primeiro a ser retirado.
- Estruturas desse tipo são conhecidas como "LIFO" (last in, first out).
- Exemplo:
 - Em uma rua sem saída, tão estreita que apenas um carro passa por vez, o primeiro carro a sair será o último a ter entrado. Observe ainda que não podemos retirar qualquer carro e não podemos inserir um carro de tal forma que ele não seja o último.

Teoria das Pilhas

Ideal para processamento de estruturas aninhadas de profundidade (tamanho) imprevisível.

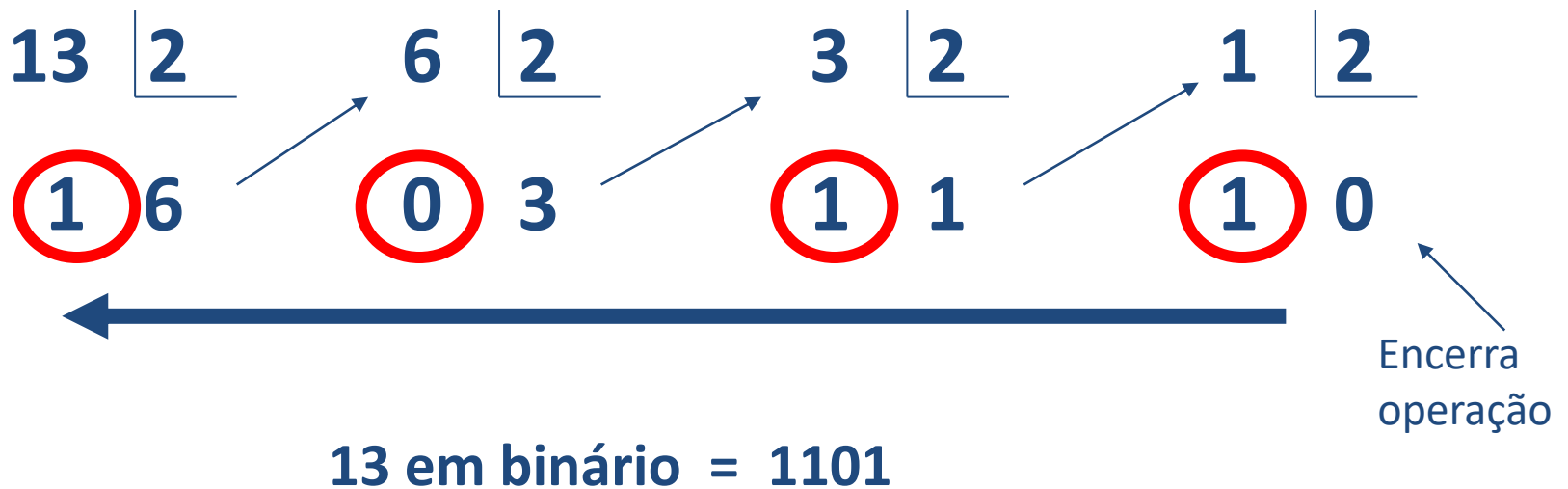
Uma pilha contém uma seqüência de obrigações adiadas: a ordem de remoção garante que as estruturas mais internas serão processadas antes das mais externas.

Aplicações em estruturas aninhadas:

- Controle de seqüências de chamadas de subprogramas;**

Exemplo do uso de PILHAS

Conversão da base decimal para binária



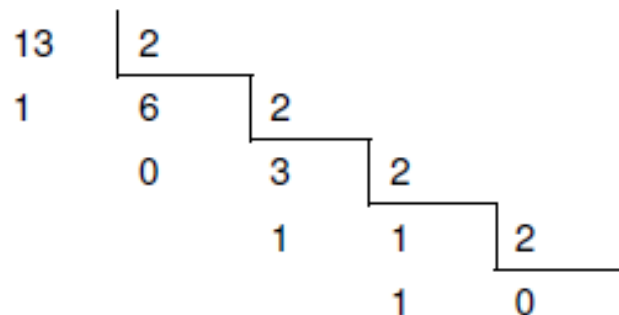
• FUNÇÕES BÁSICAS

- Seja P uma variável do tipo pilha e X um elemento qualquer
 - $\text{push}(p, x)$ – Função que insere X no topo de P. (Empilha)
- $\text{pop}(p)$ - Função que remove o elemento do topo da pilha P devolvendo o valor do topo para a rotina que a chamou. (Desempilha)
- $\text{top}(p)$ - Função que retorna uma cópia do elemento do topo de P, devolvendo o valor do topo da pilha P para a rotina que a chamou. (Copia)

• FUNÇÕES AUXILIARES

- $\text{init}(p)$ – Função que esvazia a Pilha P. (Inicia / Esvazia)
- $\text{isfull}(p)$ – Função que retorna um valor lógico informando se a pilha está cheia. Verdadeiro se estiver cheia ou Falso caso contrário. (Pilha cheia)
- $\text{isempty}(p)$ – Função que retorna um valor lógico informando se a pilha está vazia. Verdadeiro se estiver vazia ou Falso caso contrário. (Pilha vazia)

Dado um número inteiro,
positivo em base decimal,
convertê-lo para binário.



13 (decimal) = 1101 (binário)

```
# include <stdio.h>
# include <stdlib.h>
# include "pilhas.h"
```

```
int main ( ) {
    int n, r ;
    tpPilha p ;
    printf ( "Digite um inteiro positivo: " ) ;
    scanf ( "%d", &n ) ;
    init ( &p ) ;
    do {
        r = n % 2 ;
        push ( &p, r ) ;
        n = n / 2 ;
    } while ( n != 0 ) ;
    printf ( "\n\nCorrespondente ao Binario => " ) ;
    while ( isEmpty ( &p ) == 0 ) {
        r = pop ( &p ) ;
        printf ( "%d", r ) ;
    }
    printf ( "\n\n\n" ) ;
    system ( "pause" ) ;
    return 0 ;
}
```


- Até o momento, em nossos programas, na maioria das vezes, utilizamos bibliotecas padrão por meio da diretiva `#include <arquivo cabeçalho>`, como por exemplo, os arquivos `stdio.h`, `stdlib.h`, `string.h`, etc..
- Biblioteca é um conjunto de funções que podem ser utilizadas em outros programas sem necessidade se serem reescritas. É possível criar bibliotecas com funções de grande utilidade e reutilizá-las sempre que for conveniente.
- Toda biblioteca é composta por duas partes.
 - A primeira parte é o arquivo cabeçalho que possui a extensão `.h` (h de header – cabeçalho em inglês). Este arquivo deve conter os protótipos das funções e, caso necessário, as declarações de constantes, tipos, variáveis que os programas que farão uso da biblioteca precisam saber.
 - A segunda parte é o arquivo de código que possui a extensão `.c`. Este arquivo deve ter o mesmo identificador do arquivo cabeçalho e nele devem estar implementadas as funções declaradas como protótipo no arquivo `.h`.

- A diretiva `#include` indica ao compilador quais são as bibliotecas que devem ser incluídas no momento da compilação e linkedição. Sua sintaxe pode ser de duas formas:

`#include <nome_do_arquivo.h>` ou `#include "nome_do_arquivo.h"`

- Quando utilizamos a sintaxe `#include <nome_do_arquivo.h>` a procura pelo arquivo segue pelos diretórios pré-especificados do compilador .
- Quando utilizamos a sintaxe `#include "nome_do_arquivo.h"` a procura pelo arquivo é feita onde o programa fonte se encontra ou no caminho indicado.

6. A função `main()` abaixo promete ler uma palavra qualquer e imprimir a mesma na ordem inversa (exemplo: você digita a palavra "pasta de dente" e será impresso na tela "etned ed atsap"). Altere a função de tal forma que inverta palavra por palavra e não a frase inteira (exemplo: você digita a palavra "pasta de dente" e será impresso na tela "atsap ed etned").

```
int main ( ) {  
    int i;  
    tpPilha p;  
    char palavra [ 30 ] ;  
    printf ( "Digite uma palavra qualquer: " ) ;  
    gets ( palavra ) ;  
    init ( &p ) ;  
    for ( i = 0; i <= strlen ( palavra ) ; i++ )  
        push ( &p, palavra [ i ] ) ;  
    printf( "\n\nValor devolvido pela pilha => " ) ;  
    while ( isEmpty ( &p ) == 0 )  
        printf ( "%c", pop ( &p ) ) ;  
    printf ( "\n\n\n" ) ;  
    system ( "pause" ) ;  
    return 0 ;  
}
```



Lista Encadeada

Alocação de Memória

- Da área de memória que é reservada ao programa, uma parte é usada para armazenar as instruções a serem executadas e a outra é destinada ao armazenamento de dados.
- Quem determina quanto de memória será usado para as instruções é o compilador.
- Alocar área para armazenamento de dados, entretanto, é responsabilidade do programador.
- Quando a quantidade de memória utilizada pelos dados é previamente conhecida e definida no próprio código-fonte do programa, trata-se de alocação estática.
- Quando o programa é capaz de criar novas variáveis durante sua execução, dizemos que a alocação é dinâmica.

Lista Encadeada

Ponteiro

Ponteiro é uma variável especial que contém o endereço de memória de outra variável.

Ponteiro – Comandos básicos

```
< tipo de dado > * < identificador > ;
```

- Declara uma variável do tipo ponteiro.
- Pode haver um ponteiro (ou apontador) para qualquer tipo de variável.

```
(void*) malloc(tamanho em bytes);
```

- Aloca dinamicamente, durante a execução do programa, células de memória.
- Esta função devolve um ponteiro void que deve ser convertido para o tipo desejado.
- O tamanho em bytes pode ser determinado utilizando a função sizeof() que retorna o tamanho em bytes que um tipo de dados ocupa. Exemplo:

```
int *ptr;
```

```
ptr = ( int * ) malloc ( sizeof ( int ) ) ;
```

```
free (ponteiro)
```

- Libera as células de memória alocadas dinamicamente.
- Exemplo

```
free ( ptr );
```
- Para manipulação de valores utilizando ponteiros temos dois operadores:
 - O operador `*` devolve o valor contido no endereço apontado pela variável ponteiro;
 - O operador `&` devolve o endereço de memória alocado de uma variável.

Listas Encadeadas

- Por meio de ponteiros podemos alocar espaço para uma informação na memória e liberar este mesmo espaço quando não for mais necessário.
- Quando precisamos guardar várias informações na memória, independente da quantidade, podemos utilizar uma estrutura conhecida como Lista Encadeada.
- Uma lista encadeada é uma sequência de informação armazenadas em algum lugar da memória, sendo que as mesmas estão ligadas entre si por um endereço (pointer).
- Um lista encadeada é criada a partir da estrutura de um registro (estrutura STRUCT).

Outra definição para lista encadeada é um conjunto de elementos individualizados em que cada um referencia outro elemento distinto como sucessor.

Estrutura de uma lista encadeada

- A estrutura mais básica deste registro conterà dois tipos de campos:
 - O primeiro tipo serão os campos utilizados para armazenar as informações propriamente ditas e
 - O segundo tipo será um campo do tipo ponteiro que irá armazenar o endereço da próxima informação existente na memória.

```
struct lista
{
    char Dado;
    struct lista *Prox;
};
```


Lista Encadeada

```
struct lista
{
    char Dado;
    struct lista *Prox;
};
```

Inicio

143

345	
B	789
Dado	Prox

678	
E	NULL
Dado	Prox

789	
A	213
Dado	Prox

213	
C	446
Dado	Prox

765	
T	678
Dado	Prox

446	
A	765
Dado	Prox

143	
A	345
Dado	Prox

Lista Encadeada

```
struct lista  
{  
    char Dado;  
    struct lista *Prox;  
};
```

Conteúdo	Endereço de Memória
->Dado = B	345
->Prox = 789	
->Dado = E	678
->Prox = NULL	
->Dado = A	789
->Prox = 213	
->Dado = C	213
->Prox = 446	
->Dado = T	765
->Prox = 678	
->Dado = A	446
->Prox = 765	
->Dado = A	143
->Prox = 345	

Lista Encadeada

Exercício:

Desenvolva um programa para agendar o nome e o telefone de seus amigos. Considere o nome e telefone como sendo string. Utilize uma lista encadeada para resolver o exercício.