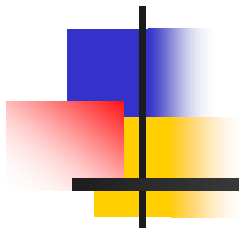


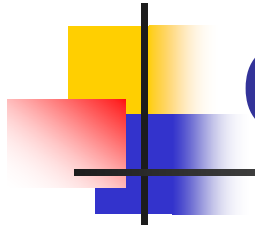
# Programação de Computadores I



Professor: Geraldo Pereira de Souza

---

Obs: Slides complementares às aulas de Programação de Computadores I em C. Em caso de erros ou sugestões favor entrar em contato no email [geraldopereira@decom.cefetmg.br](mailto:geraldopereira@decom.cefetmg.br)



# O que é C?

---

- Uma linguagem de programação
  - ⇒ Compilada
  - ⇒ Robusta
  - ⇒ Portável

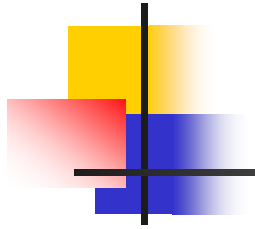


# Características

---

- Case Sensitive
- Compilada

# Exemplo de fonte em C



```
// Comentário de uma linha
```

```
/*
```

```
Comentário em  
múltiplas linhas
```

```
*/
```

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

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

```
/* Um Primeiro Programa */
```

```
int main ()
```

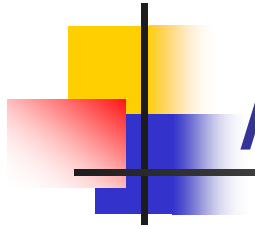
```
{
```

```
    printf ("Ola! Eu estou vivo!\n");
```

```
    system("pause");
```

```
    return(0);
```

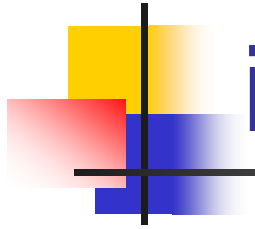
```
}
```



# // Comentário de uma linha

---

- ⇒ Comentários seguem a mesma sintaxe do C++:
  - ⇒ `“//”` Uma única linha
  - ⇒ `/* */` Múltiplas linhas



## int main() {

---

- **main** é a palavra chave que marca o início da declaração de um programa.
- **int** é uma palavra chave que indica que dentro do main deve aparecer um return com um valor inteiro.
- O nome do programa é definido no nome do arquivo. É recomendado não usar acentuação ou espaços no nome do arquivo.



# O trecho do programa

---

```
printf ("Ola! Eu estou vivo!\n");  
system("pause");  
return(0);
```

Obs: Para que você consiga ver a mensagem, a instrução `system("pause")` foi executada.

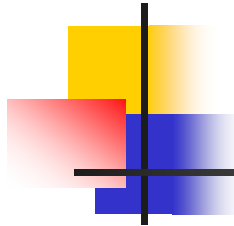


# return 0

---

- Indica que a função main terminou sem problemas.
- Será abordado mais adiante.





# void

---

- A função pode executar e não retornar nenhum valor. Não será necessário um return explícito no código.

# Exemplo em C++



```
// Comentário de uma linha
```

```
/*
```

```
    Comentário em  
    múltiplas linhas
```

```
*/
```

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

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

```
/* Um Primeiro Programa */
```

```
void main ()
```

```
{
```

```
    printf ("Ola! Eu estou vivo!\n");  
    system("pause");
```

```
}
```

# Exemplo em C



```
// Comentário de uma linha
```

```
/*
```

```
Comentário em  
múltiplas linhas
```

```
*/
```

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

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

```
/* Um Primeiro Programa */
```

```
main ()
```

```
{
```

```
    printf ("Ola! Eu estou vivo!\n");
```

```
    system("pause");
```

```
}
```



# Main

---

- Outras declarações possíveis:

*void main()*

*void main(void)*

*int main()*

*int main(void)*

*int main(int \*argv, int \*argc)*



```
printf ("\nOla! Eu estou vivo!");
```

---

- Imprime o texto na saída padrão com quebra de linha ("\\n").
- Para imprimir o conteúdo de uma variável do tipo int x:

```
printf ("\nValor de x = %d", x);
```

- Para imprimir o conteúdo de uma variável do tipo float y:

```
printf ("\nValor de y = %f", y);
```



```
printf ("Ola! Eu estou vivo!\n");
```

---

- Para imprimir formatando a saída:

```
printf ("Valor de y = %6.2f ", y);  
// 6 casas sendo 2 decimais
```

- Para imprimir o conteúdo de mais de uma variável:  

```
printf ("Valor de x=%d y= %f z=%d ", x, y, z);
```

**Obs: A ordem e o tipo das variáveis devem ser respeitadas.**



# Declaração de Variáveis

---

*<Tipo> <nome> = \* <valor inicial>*

- *Ex: int numeroCandidatos = 1000;*

*ou*

*int numeroCandidatos;*

*numeroCandidatos = 1000;*

- Devem ser declaradas no início da função main.



# Declaração de Variáveis

---

$\langle \text{Tipo} \rangle \langle \text{nome} \rangle = * \langle \text{valor inicial} \rangle$

• *Ex:*

*int x, y, z, w, media, soma;*

*ou*

*int x;*

*int y;*

*int z, w, media, soma;*





## scanf: Leitura de valores

---

A leitura de valores através do teclado é feita pela função scanf

- Exemplo para ler valores inteiros:

```
int ano;
```

```
printf("\nDigite o ano: ");
```

```
scanf("%d", &ano); // Na leitura sempre é  
// necessário usar o &
```



## scanf: Leitura de valores

---

- Exemplo para ler valores reais:

```
float salario;
```

```
printf("\nDigite o salario: ");
```

```
scanf("%f", &salario); // Na leitura sempre é  
                        // necessário usar o &
```



## scanf: Leitura de valores

---

- Exemplo para ler valores reais:

```
double salario;
```

```
printf("\nDigite o salario: ");
```

```
scanf("%lf", &salario); // Na leitura sempre é  
                        // necessário usar o &
```



## scanf: Leitura de valores

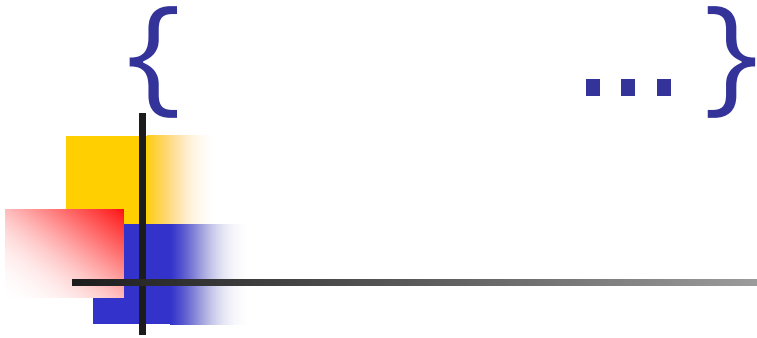
---

- Exemplo para ler caracteres:

```
char letra;
```

```
printf("\nDigite uma letra: ");
```

```
scanf("%c", &letra); // Na leitura sempre é  
// necessário usar o &
```



- “Abre chaves” e “fecha chaves”.  
Delimitam o início e o fim de um bloco.
- Os blocos devem sempre estar aninhados para facilitar a manutenção do código

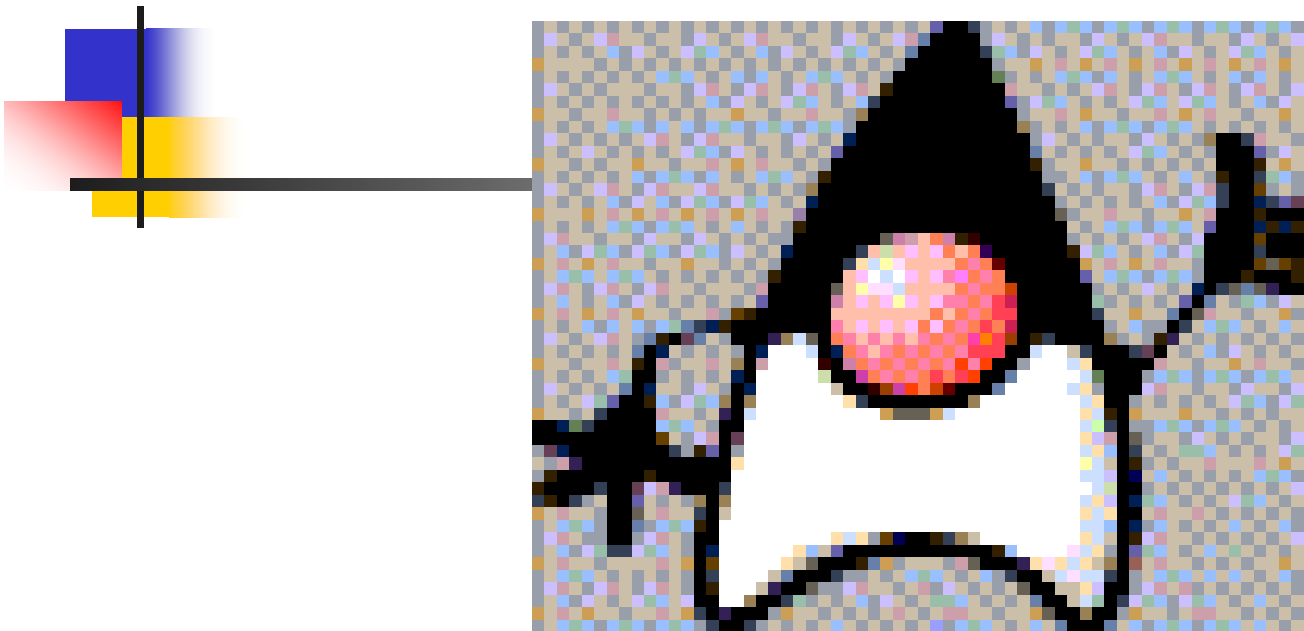
# Antes de prosseguir, assegure-se que sabe:



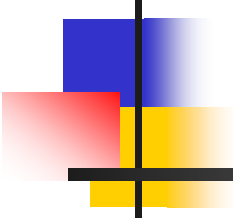
---

- Definir as principais características da linguagem C.
- Compilar e rodar aplicações.
- Identificar os principais pontos de uma aplicação.
- Saber a importância de ter o código comentado.
- Saber a importância de ter o código bem indentado.

## Parte 2: Estrutura da Linguagem



# Estrutura da linguagem

- 
- ✓ *Convenção identificadores*
  - ✓ *Tipos primitivos*
  - ✓ *Operadores*





# Identificadores

---

- O identificador deve começar com:

Grupo A:

- |                    |   |                 |
|--------------------|---|-----------------|
| 1) Letra           |   | Qq do grupo A   |
| 2) \$              | + | ou dígito + ... |
| 3) _ (caixa baixa) |   |                 |

- Obs: Não pode ser palavra reservada.



# Identificadores

---

- Exemplos:

Identificadores válidos:

userName2	User_name	
user_Name	_sys_var1	\$change

Identificadores inválidos:

user/Name	-User_name	
5userName	_sys*var1	çchange

Obs: C/C++ é case sensitive: nomeUsuario é diferente de NomeUsuario



# Palavras reservadas: Ansi C

---

**auto**  
**break**  
**case**  
**char**  
**const**  
**continue**  
**default**  
**do**  
**double**

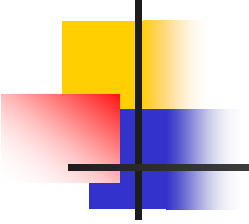
**else**  
**enum**  
**extern**  
**float**  
**for**  
**goto**  
**if**  
**int**  
**long**

**register**  
**return**  
**short**  
**signed**  
**sizeof**  
**static**  
**struct**

**switch**  
**typedef**  
**union**  
**unsigned**  
**void**  
**volatile**  
**while**

- Palavras reservadas não podem ser usadas como nome de variável

# Tipos Primitivos básicos (principais):



Tipo	Formatação	Bits	Menor	Maior
int	%d	16	-32.768	32.767
float	%f	32	3.4E-38	3.4E+38
double	%lf	64	1,7E-308	1,7E+308
long int (long)	%l	64	-2.147.483.648	2.147.483.647
char	%c	8	-	-

- Obs: String (Texto) não é tipo primitivo.
- Para formatação de string usa-se %s.
- Ver demais tipos no livro texto.

# Como declarar e atribuir valores a atributos

```
int main () {  
  
    int x, y;           // declarando variáveis inteiras  
    float z = 3.414;    // declarando e atribuindo ponto flutuante  
    double w = 3.1415;  // declarando e atribuindo double  
  
    char c;             // declarando variável caracter  
    c = 'A';            // atribuindo um valor char a variável  
    x = '6';  
    y=1000;             // atribuindo valores a variável  
    return 0;  
}
```



# Tópicos adicionais

---

- Documentação C/C++
- Listas de discussão
  - Listas de discussão

# Estrutura de um fonte em C

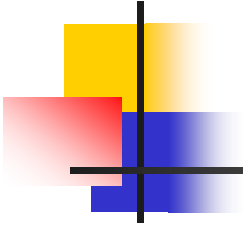


---

```
// includes de bibliotecas
```

```
int main {  
    // Variáveis  
    // Comandos  
}
```

# Laboratório

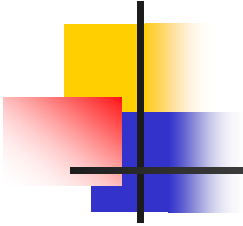


Ver roteiros do laboratório repassados pelo professor



## Módulo 03

### Parte 01 - Continuação sobre Fundamentos da Linguagem





# Operadores em C/C++

---

Unários            ++ -- + - ! ~ ()

Aritméticos        \* / % + -

Comparação        < <= > >= ==

Short-circuit      && ||

Atribuição        = “op=”



# Operadores Aritméticos

---

Operador	Significado	Exemplo
+	Adição	7 + 2
-	Subtração	3 - 2
*	Multiplicação	8 * 2
/	Divisão	6 / 3
%	Resto da divisão	7 % 2 => 1



# Operadores Aritméticos

---

Oper	Uso	Descrição
++	op++	Incrementa op de 1 unidade
--	op--	Decrementa op de 1 unidade

Exemplos:

```
int i = 5;  
i++;           // equivale a i = i+1;
```

ou

```
int i = 5;  
i--;           // equivale a i = i - 1;
```



# Operadores Relacionais

---

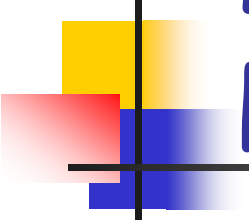
Operador	Significado	Exemplo
>	Maior que	<code>x &gt; 3</code>
>=	Maior ou igual	<code>x &gt;= 3</code>
<	Menor que	<code>x &lt; 4</code>
<=	Menor ou igual	<code>x &lt;= 7</code>
==	Igual a	<code>x == 8</code>
!=	Diferente	<code>x != 7</code>
=	Atribuição	<code>x = 3</code>



# Operadores Lógicos

---

Oper	Uso	Retorna <b>true</b> se
&&	op1 && op2	op1 <b>e</b> op2 <b>forem</b> true. Só avalia op2, <b>se</b> op1 <b>for</b> true.
	op1    op2	op1 <b>ou</b> op2 <b>for</b> true (ou ambos). Só avalia op2, <b>se</b> op1 <b>for</b> false.
!	!op	op <b>for</b> false.



# Módulo 04 (Opcional)

## Parte 02- Controle de Fluxo

### **Categoria**

decisão

loop

diversos

### **Comando**

if-else, switch-case

for, while, do-while

break, continue, label: , return



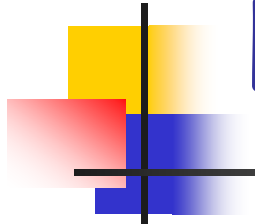
## Seja o Problema

---

Problema: Um Professor aplicou 2 provas N1, N2 com valor de 10 pontos cada. Será aprovado o aluno que tiver média final superior a 6.0

$Media \leftarrow (N1 + N2) / 2;$





# Fluxograma: Calcular Média

---

Ver diagrama no quadro ou no material da teoria.



# Portugol: Calcular Média

---

Programa Media

Variáveis

N1, N2, MEDIA: Real;

Início

Leia(N1, N2);

MEDIA  $\leftarrow$  (N1 + N2)/2;

Se(MEDIA  $\geq$  6.0) então

Escreva("Aprovado");

Senão

Escreva("Reprovado");

Fim Se

Fim



# Código em C

---

```
#include <stdio.h>
#include <stdlib.h>
int main (){
    // Declaração de variáveis
    float nota1, nota2, media;

    // Entrada de Dados
    printf ("Digite a nota da prova 1: ");
    scanf ("%f",&nota1);

    printf ("Digite a nota da prova 2: ");
    scanf ("%f",&nota2);
```



## Código em C

---

```
media = (nota1 + nota2) / 2;
printf ("\nMédia calculada é igual a %f", media);
printf ("\nMédia calculada é igual a %6.2f", media);
if (media >= 6.0){
    printf ("\nParabéns. Você foi aprovado!");
}
else{
    printf ("\nTente outra vez. Você foi reprovado!");
}
printf ("\nFim do programa. Tecle algo para continuar...\n");
system("pause");
return(0);
}
```



# Controle de Fluxo - if

---

```
if  (expressão booleana) {  
    comando ou bloco  
}  
else {  
    comando ou bloco  
}
```



# Controle de Fluxo - Exemplo if

---

```
#include <stdio.h>
int main (){
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10){
        printf ("\n\nO numero e maior que 10");
    }
    if (num==10){
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    if (num<10){
        printf ("\n\nO numero e menor que 10");
    }
    return(0);
}
```



# Controle de Fluxo - Exemplo

## if/else

---

```
#include <stdio.h>
int main (){
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num==10){
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\nVoce errou!\n");
        printf ("O numero e diferente de 10.\n");
    }
    return(0);
}
```



# Controle de Fluxo - Exemplo

## if/else if

---

```
#include <stdio.h>
int main (){
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num>10){
        printf ("\n\nO numero e maior que 10");
    }
    else if (num==10){
        printf ("\n\nVoce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else if (num<10){
        printf ("\n\nO numero e menor que 10");
    }
    return(0);
}
```





# Operadores - Cuidado!!!

---

**O operador = é de atribuição. Enquanto os testes booleanos devem ser feitos usando dois operadores = seguidos: ==**



# Controle de Fluxo - switch

```
switch (variavel) {  
    case expressão1:  
        comando ou bloco  
        break;  
  
    . . .  
  
    case expressãon:  
        comando ou bloco  
        break;  
  
    default:  
        comando ou bloco  
        break;  
}
```



# Exemplo de uso - switch

```
#include <stdio.h>

int main (){
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    switch (num){
        Case 1:
            printf ("\n\nO numero e igual a 1.\n");
            break;
        Case 2:
            printf ("\n\nO numero e igual a 2.\n");
            break;
        Case 3:
            printf ("\n\nO numero e igual a 3.\n");
            break;
        default:
            printf ("\n\nO numero nao e nem 1 nem 2 nem 3.\n");
    }
    return(0);}

```



# Loops - while

---

```
while (expressão booleana) {  
    comando ou bloco  
}
```

Obs: Repete o bloco enquanto a condição for verdadeira (true). O teste é feito no início.



# Exemplo de uso - while

---

```
#include <stdio.h>
int main (){
    int i = 0;
    while ( i < 100){
        printf(" %d", i);
        i++;
    }
    return(0);
}
```



# Exemplo de uso - while

---

```
#include <stdio.h>
int main (){
    int i = 0;
    while (1){        // Loop infinito
        printf(" %d", i);
        i++;
    }
    return(0);
}
```



# Loops - do while

---

```
do {  
    comando ou bloco  
    expressão de iteração;  
}while (expressão booleana);
```

Obs: Repete o bloco enquanto a condição for verdadeira (true). O teste é feito no final. O bloco é executado pelo menos uma vez mesmo a condição sendo falsa.



# Exemplo de uso - do while

---

```
#include <stdio.h>
int main (){
    int i = 0;
    do {
        printf(" %d", i);
        i++;
    } while ( i < 100);
    return(0);
}
```





# Exemplo de uso - do while

---

```
#include <stdio.h>
int main (){
    int i = 2000;
    do {
        printf(" %d", i);
        i++;
    } while ( i < 100);
    return(0);
}
```



# Exemplo de uso - do while

---

```
#include <stdio.h>
int main (){
    int i;
    do{
        printf ("\n\nEscolha a fruta pelo numero:\n\n");
        printf ("\t(1)...Mamão\n");
        printf ("\t(2)...Abacaxi\n");
        printf ("\t(3)...Laranja\n\n");
        scanf("%d", &i);
    } while ((i<1)||i>3));
    switch (i){
        case 1:
            printf ("\t\tVoce escolheu Mamão.\n");
            break;
        case 2:
            printf ("\t\tVoce escolheu Abacaxi.\n");
            break;
        case 3:
            printf ("\t\tVoce escolheu Laranja.\n");
            break;
    }
    return(0);}
```



# Loops - for

---

```
for(int_statament; boolean_cond; step)
{
    << comandos >>
}
```

Obs: Repete o bloco uma quantidade fixa de vezes.



# Loops - for

---

```
#include <stdio.h>
int main (){
    int count;
    for (count=1; count<=100; count++){
        printf ("%d ",count);
    }
    return(0);
}
```



# Loops - for loop sem conteúdo

---

```
#include <stdio.h>
int main ()
{
    long int i;

    printf("\a"); /* Imprime o caracter de alerta (um beep) */

    for (i=0; i<10000000; i++); /* Espera 10.000.000 de iteracoes */

    printf("\a"); /* Imprime outro caracter de alerta */

    return(0);
}
```



# Loops - for com Break

---

```
for(int_statament; boolean_cond; step)
{
    << comandos >>
    break;
}
```

```
// break dentro de qualquer loop, faz o
// laço abandonar o loop mais interno
```



# Loops - for com Break

---

```
#include <stdio.h>
int main (){
    int count;
    for (count=0; count<10; count++){
        printf ("%d ",count);
        if (i==5){
            break;        // Abandona o laço
        }
    }
    return(0);
}
```



# Loops - for com Continue

---

```
for(int_statament; boolean_cond; step)
{
    << comandos >>
    continue;
}
```





# Loops - for com Continue

---

```
#include <stdio.h>
int main (){
    int count;
    for (count=0; count < 10; count++){
        printf ("%d ",count);
        if (i==5){
            continue;    // Avança o contador
        }
    }
    return(0);
}
```



# Operadores - Cuidado!!!

---

**O operador = é de atribuição. Enquanto os testes booleanos devem ser feitos usando dois operadores = seguidos: ==**

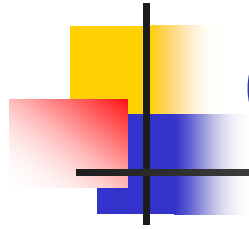


# Operadores - Cuidado!!!

---

Qual o problema com o código abaixo?

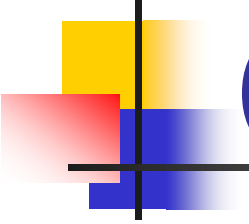
```
#include <stdio.h>
int main (){
    int i = 200;
    while ( i = 100){
        printf(" %d", i);
        i++;
    }
    return(0);
}
```



# Cast e conversão

---

- Conversão de primitivos
- Cast de primitivos



# Conversão de primitivos (tempo de compilação)

---

- Realizada em atribuições
- Chamada (passagem de parâmetro) de método
- Promoção aritmética



# 1 - Conversão na atribuição

---

- `int i = 10;`
- `double d = i; // ok`
  
- `double d = 10;`
- `int i = d; // erro`
- Será necessário um cast explícito



## cast:

---

- `double d = 10.5;`
- `int i = (int) d; // o double será  
// convertido no int  
// perdendo a parte decimal`

`// O resultado do i é igual a 10`



## 2 - Chamada de método

---

- Quando é passado um parâmetro, a conversão é realizada como na atribuição





# Laboratório

---

Ver roteiros

# Array (ou Vetor)



Estrutura de dados que pode conter um conjunto homogêneos de elementos.

---

Características:

- Tamanho fixo.
- O índice do primeiro elemento: 0

Definição:

```
int lista[] = {10, -2, 3};
```

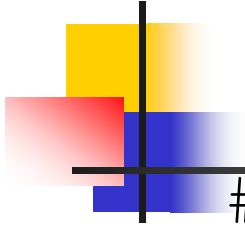
ou

```
int lista[3];
```

```
lista[0] = 10; lista[1] = -2;
```

```
lista[2] = 3;
```

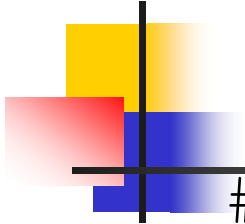
# Array (ou Vetor)



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

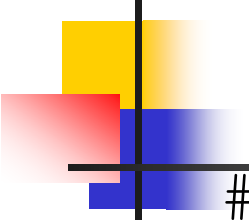
int main () {
    int lista[] = {10, -2, 3};
    int i=0, j=0;
    printf ("\n== Impressão do Vetor ==");
    for(i=0; i < 3; i++){
        printf ("\nVetor[%d]=%d", i, lista[i]);
    }
    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

# Array (ou Vetor)



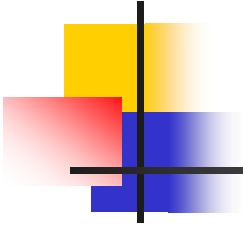
```
#include <stdio.h>
#include <stdlib.h>
#define TAMANHO 3
int main () {
    int lista[] = {10, -2, 3};
    int i=0, j=0;
    printf ("\n== Impressão do Vetor ==");
    for(i=0; i < TAMANHO ; i++){
        printf ("\nVetor[%d]=%d", i, lista[i]);
    }
    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

# Array (ou Vetor)



```
#include <stdio.h>
#include <stdlib.h>
#define TAMANHO 3
int main () {
    int lista[] = {10, -2, 3};
    int i=0, j=0;
    printf ("\n== Impressão Invertida ==");
    for(i= TAMANHO-1; i >= 0; i++){
        printf ("\nVetor[%d]=%d", i, lista[i]);
    }
    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

# Texto em C

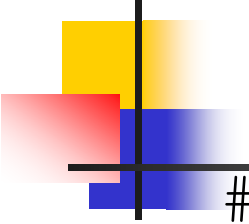


Texto ou string é considerado um array de caracteres com a terminação `'\0'` que marca o fim da string

```
char ch = 'a';    // Tipo char pode ter um  
                  // "character"
```

**Formatador para char é `"%c"`**

# Texto em C



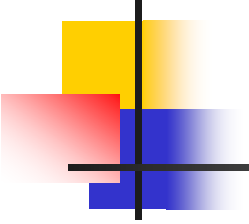
```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char ch;
    printf ("\nDigite o character:");

    scanf ("%c", &ch) ;

    printf ("\nCaracter digitado: %c", ch) ;

    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

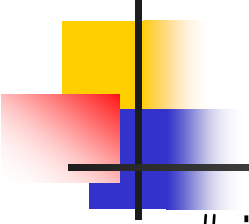
# Lendo e escrevendo string



```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char nome[40];
    printf ("\nDigite seu nome:");
    gets(nome); // Lê uma string
    printf ("\nBoa noite:");
    puts(nome); // Escreve uma string na tela
    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

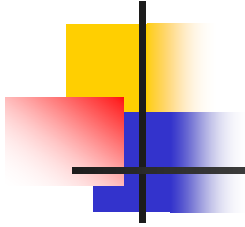


# Lendo e escrevendo string



```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char nome[40];
    printf ("\nDigite seu nome:");
    scanf("%s", nome);
    printf ("\nBoa noite: %s ", nome);
    printf ("\nFim do programa!!!");
    system("pause");
    return(0);
}
```

# Funções para manipular string



**gets** lê a string da entrada padrão

**puts** imprime a string na saída padrão

**strlen** retorna o número de caracteres da string

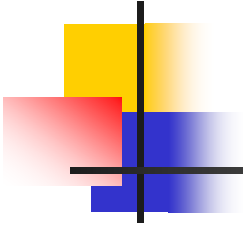
**strcat** concatena duas strings

**strcmp** compara duas strings

**strcpy** copia uma string para outra

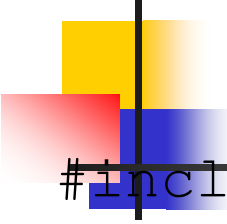
Obs: O **gets** e **puts** já controla o terminador da string `'\0'`. Para manipulação de texto Deve-se usar essas funções.

# Exemplo 1: Função: strlen



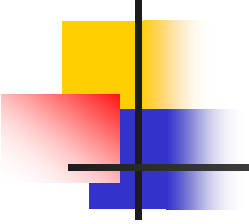
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main () {
    char nome[40];
    printf ("\nDigite seu nome:");
    gets(nome);
printf ("\nSeu nome tem %d letras", strlen(nome));
    system("pause");
    return(0);
}
```

## Exemplo 2: Função: strlen



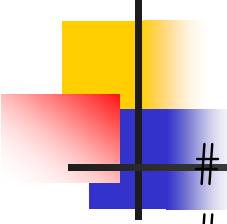
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main () {
    char nome[40];
    printf ("\nDigite seu nome:");
    gets(nome);
    printf ("\nNome Invertido:" );
    for (int i=(strlen(nome)-1); i>=0; i--) {
        printf("%c", nome[i]);
    }
    system("pause");
    return(0);
}
```

## Exemplo 2: Função: strcat



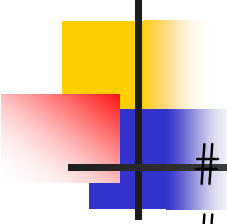
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main () {
    char nome[40];
    char frase[80] = "Bom dia: ";
    printf ("\nDigite seu nome:");
    gets(nome);
    strcat(frase, nome);
    printf ("\n%s", frase);
    system("pause");
    return(0);
}
```

## Exemplo 2: Função: strcpy



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main () {
    char nome[40];
    char nome2[40];
    printf ("\nDigite seu nome:");
    gets(nome);
    strcpy(nome2, nome); // copia nome para nome2
    printf ("\nNome: %s", nome);
    printf ("\nCópia de nome: %s", nome2);
    system("pause");
    return(0);
}
```

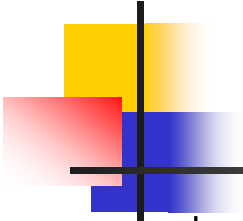
## Exemplo 2: Função: strcmp



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

```
int main () {
    char nome1[40];
    char nome2[40];
    printf ("\nDigite nome 1: ");
    gets(nome1);
    printf ("\nDigite nome 2: ");
    gets(nome2);
    int diferenca = strcmp(nome1, nome2);
```

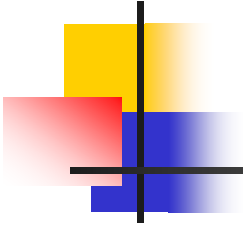
## Exemplo 2: Função: strcmp



```
printf ("\nDiferença entre %s e %s = %d",
        nome1, nome2, diferenca);
if(diferenca==0){
    printf ("\n%s é igual a %s", nome1, nome2);
}
else if (diferenca > 0){
    printf ("\n%s precede %s !!!", nome2, nome1);
}
else {
    printf ("\n%s precede %s !!!", nome1, nome2);
}
system("pause");
return(0);
}
```



# Resultado para:



Digite nome 1: AAA

Digite nome 2: BBB

**Diferença da comparação entre AAA e BBB = -1**  
**AAA precede BBB !!!**

Pressione qualquer tecla para continuar. . .



# Funções em C

---

- Exemplo de função em C
- Regra para criar uma função
- Protótipo de função



# Exemplo 1:

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

void saudacaoInicial(){
    printf ("\nSeja bem vindo usuário! ");
}

void saudacaoFinal(){
    printf ("\nVolte sempre usuário! ");
}

int main (){
    saudacaoInicial();
    saudacaoFinal();
    system("pause");
    return(0);
}
```



## Exemplo 2: com parâmetros

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

int somaNumeros(int x, int y){
    int resultado = x+y;
    return resultado;
}

int main (){
    int numero1, numero2;
    printf("\nDigite o primeiro número: ");
    scanf("%d", &numero1);
    printf("\nDigite o segundo número: ");
    scanf("%d", &numero2);
    printf("\nSoma dos números: %d ", somaNumeros(numero1, numero2));
    system("pause");
    return(0);
}
```



# Forma Geral para Função com parâmetros

```
void nomeDaFuncao(tipo nomeParametro1, tipo nomeParametro2, ...){
```

```
// Corpo da função
```

```
// Não pode colocar return porque a função declara retorno do tipo void
```

```
}
```



# Forma Geral para Função parâmetros

```
tipo_retorno nomeDaFuncao(tipo nomeParametro1, tipo, nomeParametro2, ...){
```

```
// Corpo da função
```

```
return valor;      // valor tem que ser do mesmo tipo de tipo_retorno
```

```
}
```



# Protótipo de função

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

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

```
int somaNumeros(int x, int y);
```

```
int main (){
```

```
    int numero1, numero2;
```

```
    printf("\nDigite o primeiro número: ");
```

```
    scanf("%d", &numero1);
```

```
    printf("\nDigite o segundo número: ");
```

```
    scanf("%d", &numero2);
```

```
    printf("\nSoma dos números: %d ", somaNumeros(numero1, numero2));
```

```
    system("pause");
```

```
    return(0);
```

```
}
```

```
int somaNumeros(int x, int y){
```

```
    int resultado = x+y;
```

```
    return resultado;
```

```
}
```



# Passagem de Parâmetro por valor

---

```
#include <stdio.h>
#include <stdlib.h>
int incrementaX(int x) {
    x = x + 10;
    return x;
}
int main(){
    int x = 5;
    printf("x=%d", x); // Qual valor de x?
    printf("x=%d",(incrementaX(x));
    printf("x=%d", x); // Qual valor de x?
}
```





# Passagem de Parâmetro por referência

---

```
int incrementaX(int *x) {
    *x = *x + 10;
    return *x;
}

int main(){
    int x = 5;
    printf("\nx=%d", x); // Qual valor de x?
    printf("\nx=%d",(incrementaX(&x)));
    printf("\nx=%d", x); // Qual valor de x?
    printf("\n\n\n");
    system("pause");
}
```

# Passagem de Parâmetro por valor – outro exemplo



```
void troca(int a, int b){
```

```
    int temp;
```

```
    temp=a;
```

```
    a=b;
```

```
    b=temp;
```

```
}
```

```
int main(){
```

```
    int a=2,b=3;
```

```
    printf("Antes de troca() :\na=%d\nb=%d\n",a,b);
```

```
    troca(a,b);
```

```
    printf("Depois de troca:\na=%d\nb=%d\n",a,b);
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

# Passagem de Parâmetro por valor – outro exemplo

```
void troca(int *a, int *b){
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
int main(){
    int a=2,b=3;
    printf("Antes de troca():\na=%d\nb=%d\n",a,b);
    troca(&a,&b);
    printf("Depois de troca():\na=%d\nb=%d\n",a,b);
    return 0;
}
```



# Por quê usar Função?

---

- Reuso de código: função pode ser chamada várias vezes e em outros programas;
- Melhora *manutenabilidade*: mais fácil de dar manutenção;
- Melhora *legibilidade*: mais fácil de ser entendido;



# TAD: Tipo Abstrato de Dados

---

- O usuário também pode definir seus próprios tipos de dados na linguagem C/C++



# TAD: Tipo Abstrato de Dados

---

- Suponha que você queira criar uma variável que seja capaz de armazenar o código de um aluno, 3 notas de uma prova e a média final.



# TAD: Tipo Abstrato de Dados

---

```
struct Aluno
{
    int codigo;
    double notas[3];
    double media;
};
```



# TAD: Tipo Abstrato de Dados

---

```
int main(){
    struct Aluno jose;
    jose.codigo=1; jose.notas[0]=7.5;
    jose.notas[1]=8; jose.notas[2]=5;
    jose.media =(jose.notas[0]+jose.notas[1]+jose.notas[2])/3;
    printf("\nMatrícula: %d", jose.codigo);
    printf("\nMédia: %.2lf", jose.media);
    return 0;
}
```

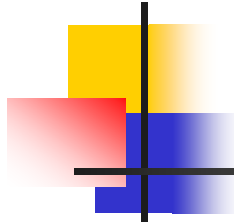




# TAD: Tipo Abstrato de Dados

---

```
int main(){  
    struct Aluno turma[40]; // Definição de uma turma de 40 alunos  
    // Restante do código do programa  
}
```



# Acessando Membros da Matriz de Estruturas

---

```
double media=turma[10].media
```

```
// Recupera a média do aluno que está na  
posição 10 do vetor
```



# Funções e Estruturas

---

```
struct Venda  
{  
    int pecas;  
    double preco;  
};
```



# Funções e Estruturas

---

```
Venda totalVendas(Venda v1, Venda v2)
{
    Venda total;
    total.pecas=v1.pecas+v2.pecas;
    total.preco=v1.preco+v2.preco;
    return total;
};
```



# Por que usar TAD?

---

- Definir seus próprios tipos em C/C++;
- Armazenar dados de tipos diferentes;
- Facilitar a programação;
- Para saber mais consultar o **capítulo 8** do livro texto;
- Fazer os exercícios do 1 ao 15 do livro texto;



# Manipulação de Dados em Memória Secundária: Arquivos

---

- As vezes o programa necessita buscar informações de um fonte externa ou enviá-las para um fonte externa.
- As informações podem estar nos seguintes lugares:
  - arquivo;
  - em algum lugar na rede;
  - em memória;
  - em outro programa;
  - Em um banco de dados;

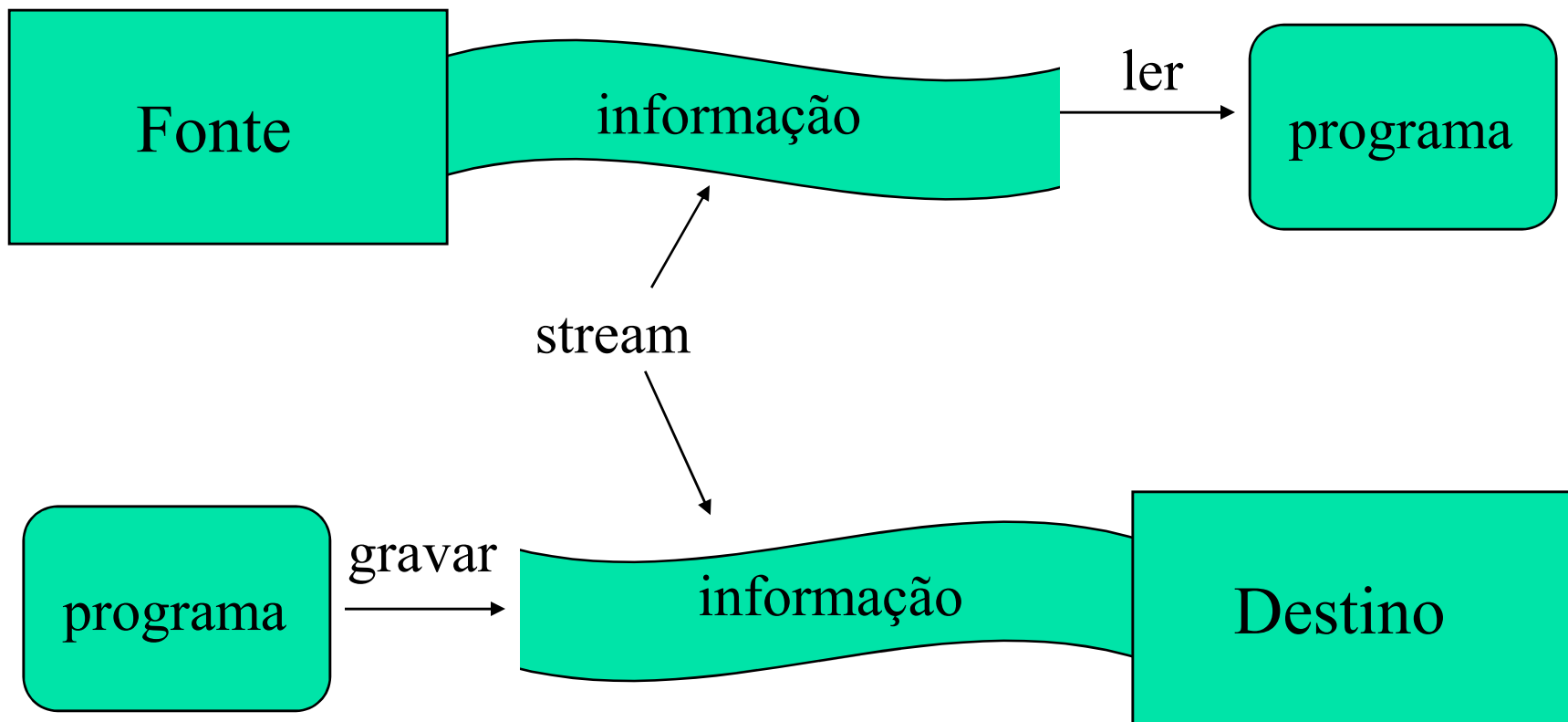


# Manipulação de informações

---

- As informações podem ser dos seguintes tipos:
  - binários;
  - caracteres;
  - imagens e
  - sons

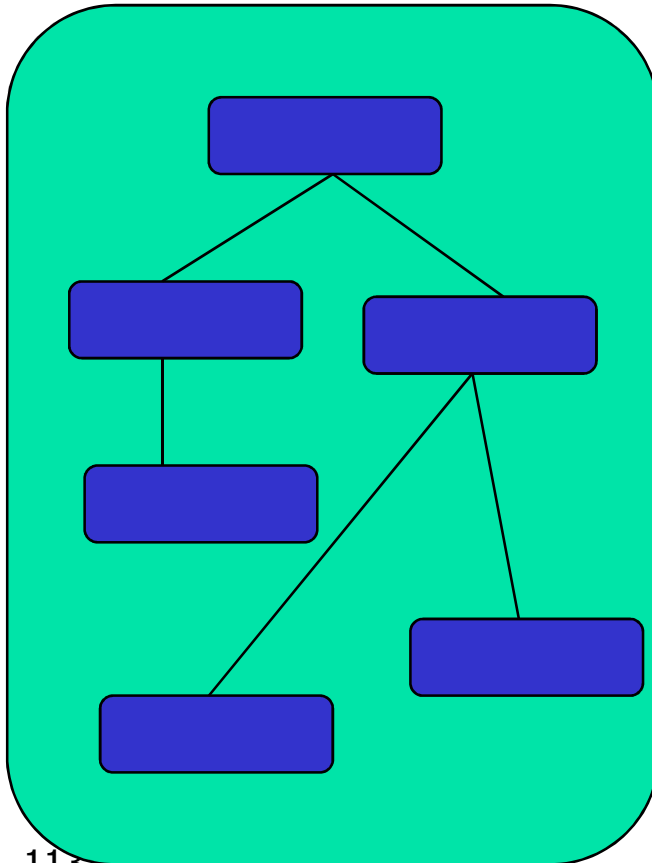
## Um programa lendo ou gravando informações em stream



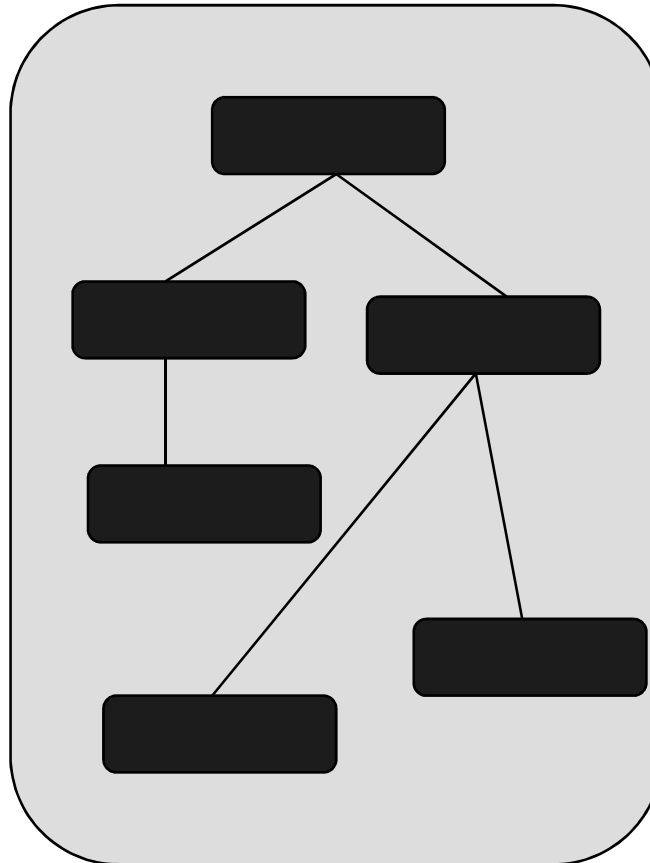


# Informações Texto x Binário

Stream de caracter



Stream de byte





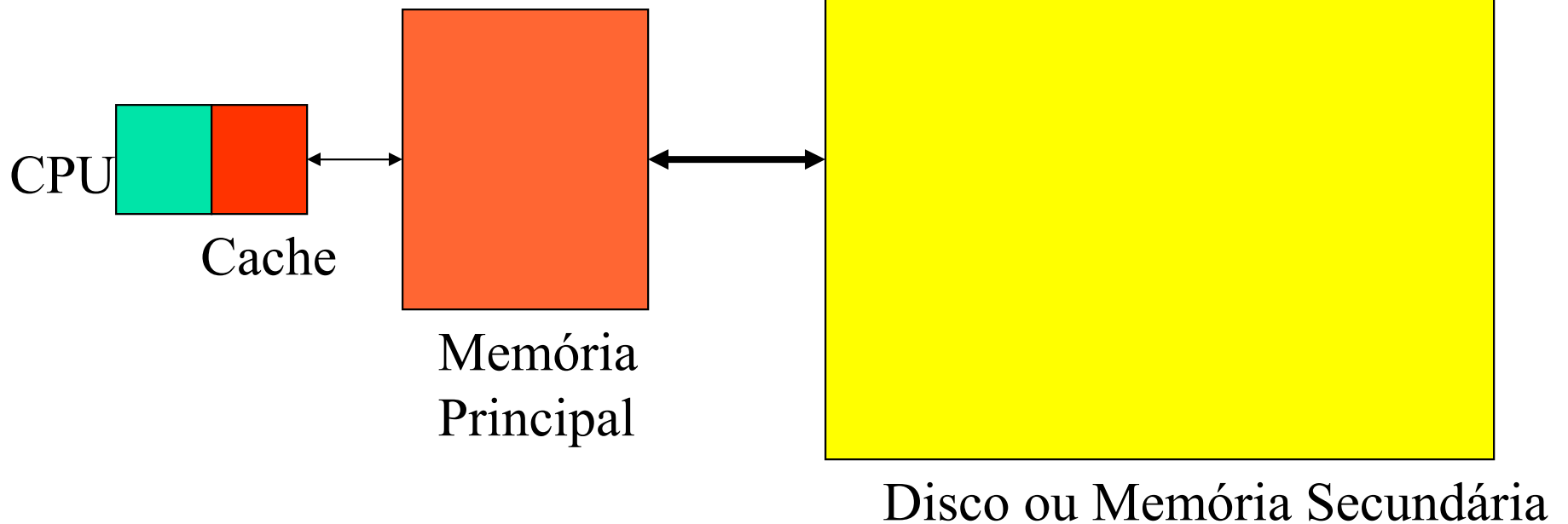
# Introdução

---

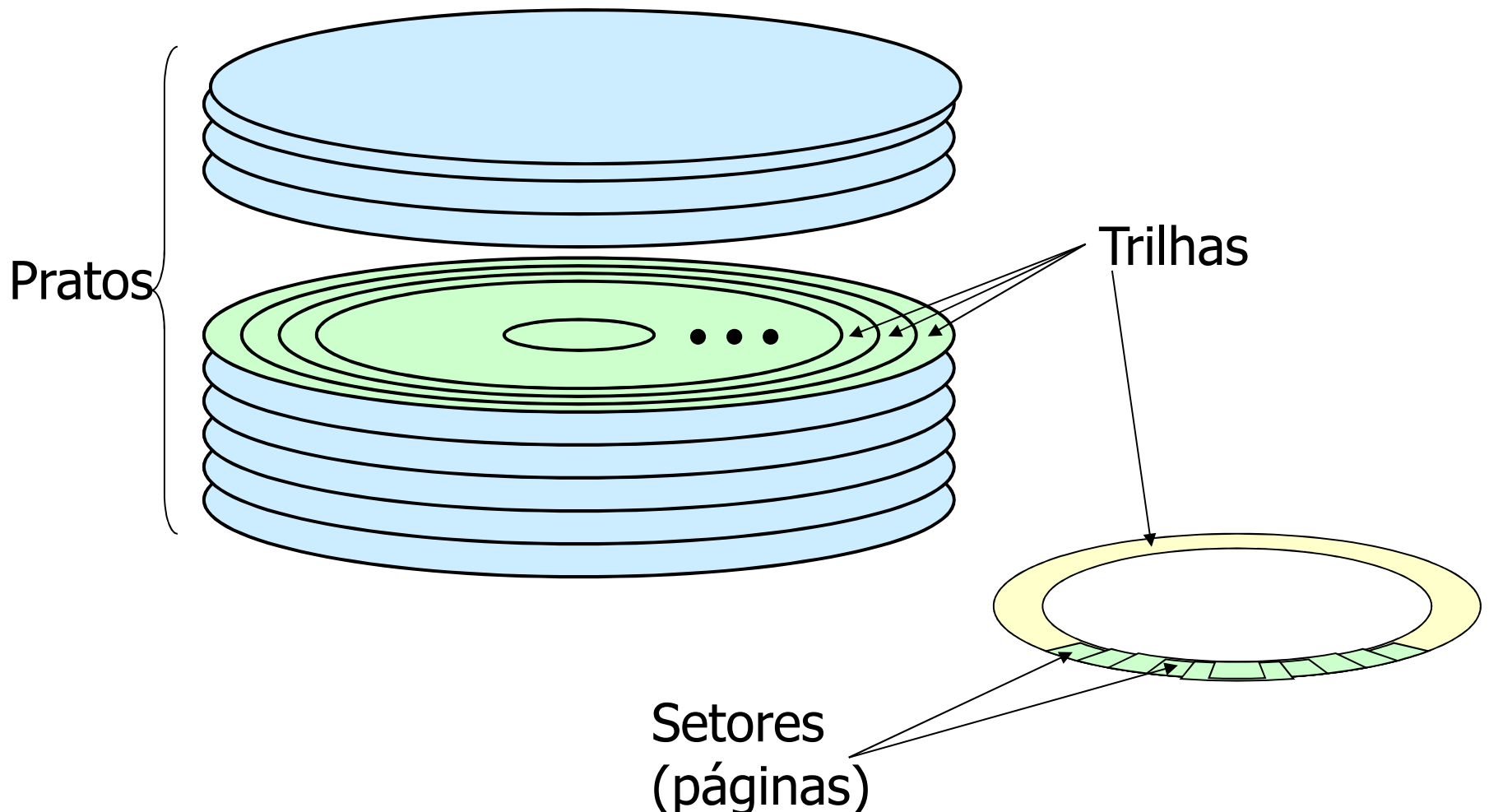
- Manipulação de dados armazenados em memória primária é eficiente, quando utilizamos estruturas de dados apropriadas:
  - Árvores binárias, tabelas hash, etc.
- Problema: utilizar tais estruturas quando a quantidade de dados é muito grande.
  - Geralmente, os dados estão em disco e transferi-los para a memória pode ser muito caro.

# Hierarquias de Memória

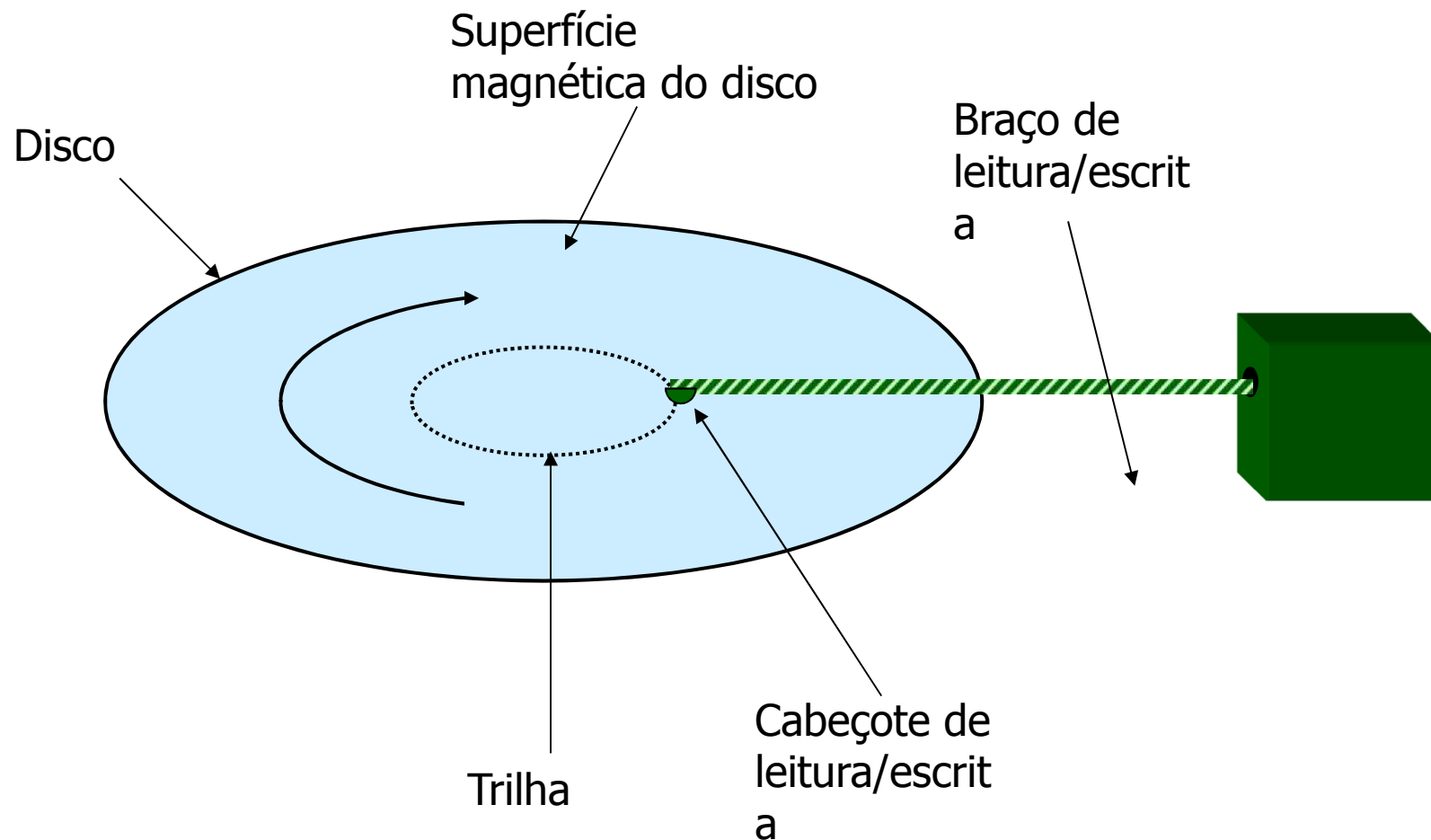
- A memória de uma máquina é normalmente dividida da seguinte maneira:

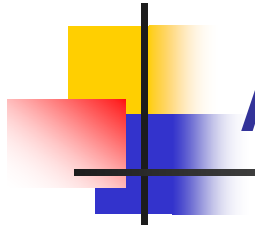


# Organização do Disco Magnético



# Organização do Disco Magnético...

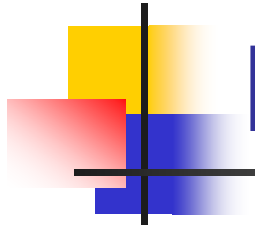




# Acesso a Disco

---

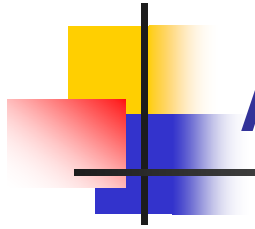
- Acesso a disco é lento;
- Todo arquivo aberto deve ser fechado!
- Estrutura de dados como árvore binária em disco podem ser usadas para otimizar performance;
- Busque blocos de informações em vez de caracteres;



# Leitura/Escrita em Arquivos

---

- O acesso a um arquivo pode ser feito de duas maneiras:
  - Seqüencial: a partir do início do arquivo, lê-se byte por byte até chegar ao final.
  - Direto: especifica-se uma posição a partir de onde os dados serão lidos.



# Acesso Seqüencial vs. Direto

---

- Quando somente um registro tiver que ser lido, utiliza-se acesso direto.
- Quando todo o arquivo tiver que ser lido, deve-se obrigatoriamente utilizar o acesso seqüencial.
- Quando poucos dados tiverem que ser lidos, deve-se analisar qual solução é a mais viável, se utilizar acesso direto ou acesso seqüencial.





# Manipulação de Arquivos em C/C++

---

- Arquivos binários vs. arquivos textos
- Principal Aplicação: **Bancos de Dados**
- As linguagens C e C++ provêem funções para manipulação de arquivos binários:
  - FILE\* em C
  - ifstream e ofstream em C++



# Abertura e Criação de arquivos

---

- Para criarmos/abrirmos um arquivo, utilizamos a função **fopen**:

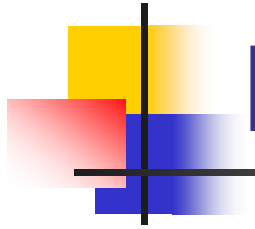
```
FILE * f;
```

```
f = fopen(nome_arquivo, modificadores)
```

- Quando a abertura falha, **fopen** retorna **NULL**.

- Para fecharmos o arquivos:

```
fclose(f);
```



# Modificadores

---

- "r": Abre para leitura em modo texto. Se o arquivo não existir, a operação falha.
- "w": Abre um arquivo vazio para escrita em modo texto. Se o arquivo existir, seu conteúdo é destruído.
- "a": Abre para escrita no final do arquivo (append) em modo texto. Cria o arquivo caso ele não exista.
- "r+": Abre em modo texto para leitura e escrita. O arquivo deve existir.
- "w+": Cria um arquivo em modo texto para atualização. Se o arquivo existir, seu conteúdo é destruído.



# Modificadores

---

- `"a+"`: Abre um arquivo em modo texto para atualização, ou seja, tanto para leitura como para gravação. Se o arquivo não existir, ele será criado.
- `"rb"`: Abre um arquivo em modo binário para leitura. Se o arquivo não existir, a operação irá falhar e `fopen` retornará `NULL`.
- `"wb"`: Cria um um arquivo em modo binário para gravação. Se o arquivo existir seu conteúdo será apagado e recomeça a gravação a partir do início.
- `"ab"`: Abre um arquivo em modo binário para gravação, a partir do seu final. Se o arquivo não existir, será criado.



# Modificadores

---

- `"rb+"`: Abre um arquivo em modo binário para atualização, ou seja, tanto para leitura quanto para gravação. Se o arquivo não existir, a operação irá falhar e `fopen()` retornará `NULL`.
- `"wb+"`: Cria um arquivo em modo binário para atualização, ou seja, tanto para leitura quanto para gravação. Se o arquivo existir, seu conteúdo será destruído.
- `"ab+"`: Abre um arquivo em modo binário para atualização gravando novos dados a partir do final do arquivo. Se o arquivo não existir, ele será criado.



# Escrita Binária

---

- Utiliza a função **fwrite**:

```
size_t fwrite(buffer, size, count, stream);
```

- Retorna o número de itens escritos, o que pode ser menor que count, caso ocorra um erro.
- Parâmetros:
  - buffer: local que contém os dados a serem escritos
  - size: tamanho dos registros em bytes
  - count: número máximo de registros a serem lidos
  - stream: Apontador para um estrutura FILE



# Leitura Binária

---

- Utiliza a função **fread**:

```
size_t fread(buffer, size, count, stream);
```

- Retorna o número de itens lidos, o que pode ser menor que count, caso ocorra um erro ou o fim de arquivo seja encontrado.
- Parâmetros:
  - buffer: local para onde os dados serão lidos
  - size: tamanho dos registros em bytes
  - count: número máximo de registros a serem lidos



# Posicionamento Aleatório

- Utiliza a função **fseek**:

```
int fseek(stream, offset, origem);
```

- Retorna zero caso consiga posicionar.
- Parâmetros:
  - stream: Apontador para um estrutura FILE
  - offset: número de bytes de deslocamento da origem
  - origem: pode ser um dos três abaixo:
    - SEEK\_CUR: Posição corrente no arquivo
    - SEEK\_END: Final do arquivo
    - SEEK\_SET: início do arquivo





# Obtendo Posicionamento

---

- Utiliza a função **ftell**:

```
long ftell(stream) ;
```

- Retorna a posição corrente do apontador de arquivo.



## *Status* do arquivo

---

- Para saber se chegou ao final, utiliza **feof**:

```
int feof(stream) ;
```

- EOF = End Of File
- Retorna zero se não estiver no final, ou algum valor diferente de zero, caso contrário.

- Para saber se houve algum erro, utiliza **ferror**:

```
int ferror(stream) ;
```

- Retorna zero se nenhum erro tiver ocorrido, ou algum valor diferente de zero, caso contrário.



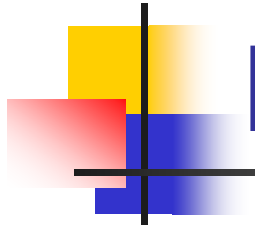
# Arquivo de Exemplo

---

Chave = 7182

Cada registro  
contém B bytes.

0	23	“Frederico Gabriel”
1	456	“Joaquim Fernando”
...	...	...
<i>i</i>	7182	“Cremilda Aparecida”
...	...	...
<i>n</i> - 1	25202	“Patrícia Amélia”



# Busca Seqüencial

---

- A busca é feita comparando-se registro por registro a partir do primeiro registro do arquivo.
- Algoritmo:
  - enquanto não for fim do arquivo faça
    - lê o próximo registro do arquivo;
    - se a chave do registro for igual à chave de busca:
      - fim da busca: registro encontrado;
  - se após ler todo o arquivo, a chave não foi encontrada, então a busca termina sem ter encontrado o arquivo.



# Busca Binária

---

- A busca é semelhante à busca binária em memória principal.
- Registram-se os índices limites inferior e superior da região do arquivo onde a busca está sendo feita: inicialmente o primeiro e o último registro.
- Lê o elemento do meio do arquivo ( $\text{meio} = (\text{inf} + \text{sup}) / 2$ ).
- Se a chave de busca for igual ao elemento do meio, encontrado.
- Se for menor, sup passa a ser ( $\text{meio} - 1$ ).
- Se for maior, inf passa a ser ( $\text{meio} + 1$ ).
- Pára-se quando a chave for encontrada ou quando o limite inferior ultrapassar o superior.



# Comparação

---

- Busca seqüencial:
  - O número de leituras e comparações é  $O(n)$ .
  - O acesso seqüencial é feito muito eficientemente, em geral utilizando *bufferring*, e a única leitura lenta é a primeira **de cada página**. As seguintes são como acessos à memória principal.
- Busca Binária:
  - O número de leituras e comparações é  $O(\log n)$ .
  - O acesso direto é muito ineficiente, só sendo recomendado quando o arquivo for muito grande.