

Unidade 0 - Nivelamento - Introdução à Linguagem C para Programadores Java



PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

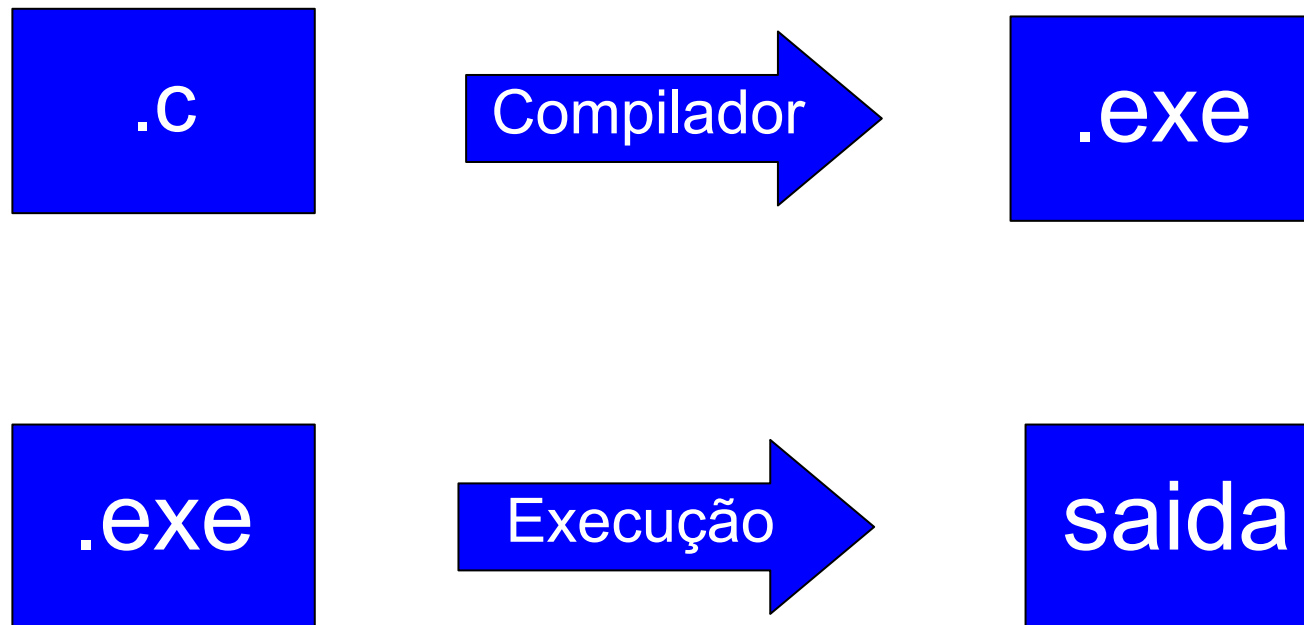
Sumário

- Introdução
- Leitura e Escrita
- String
- Registro
- Ponteiro
- Arquivo
- Passagem de Parâmetro

- **Introdução**
- Leitura e Escrita
- String
- Registro
- Ponteiro
- Arquivo
- Passagem de Parâmetro

Linguagem C

- Criada por Dennis Ritchie em 1972 a partir da linguagem B (criada por Ken Thompson a partir da BCPL)
- Propósito geral, estruturada, imperativa e procedural
- Características de linguagens de alto e baixo nível



Como Começar

- No Linux, o compilador gcc está normalmente instalado

`gcc fonte.c -o executavel` (compilação)

`./executavel` (execução)

- No Windows, baixar um compilador como, por exemplo, o DevC++ ou o Code::Blocks

Como Começar

- No Linux, o compilador gcc está normalmente instalado

`gcc fonte.c -o executavel` (compilação)

`./executavel` (execução)

- No Windows, baixar um compilador
DevC++ ou o Code::Blocks

Apenas o gcc é assunto
para muitas aulas...

Estrutura Básica do Código Fonte

```
/* Empresa  
 * Autor  
 * Data  
 * Objetivo  
 */  
  
#include <stdio.h>  
  
int main(int argc, char *argv[]) {  
  
    return 0;  
  
}
```


Estrutura Básica do Código Fonte

```
/* Empresa  
 * Autor  
 * Data  
 * Objetivo  
 */
```

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

```
int main(int argc, char *argv[]) {
```

```
    return 0;
```

```
}
```

Cabeçalho do programa:
contém informações para
identificação do código

Estrutura Básica do Código Fonte

```
/* Empresa  
 * Autor  
 * Data  
 * Objetivo  
 */
```

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

```
int main(int argc, char *argv[]) {
```

```
    return 0;
```

```
}
```

Bibliotecas: contém a
implementação de alguns
comandos que vamos
utilizar (printf e scanf)

Estrutura Básica do Código Fonte

```
/* Empresa  
 * Autor  
 * Data  
 * Objetivo  
 */
```

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

```
int main(int argc, char *argv[]) {
```

```
    return 0;
```

```
}
```

Função main

Meu Primeiro Programa

```
#include <stdio.h>

int main(int argc, char *argv[]) {

    printf ("Ola pessoal!!!\n\n");
    return 0;

}
```

- Declaração de variáveis, comentários e sintaxe do if, while, do-while, for, switch-case é igual em C, C++, C# e Java
- Em C e C++, chamamos métodos e arrays de funções e vetores, respectivamente
- Se tivermos dúvidas sobre uma função em C, basta digitar `man nomeFunção` (e.g., `man printf`) no terminal do Linux

Sumário

- Introdução
- **Leitura e Escrita**
- String
- Registro
- Ponteiro
- Arquivo
- Passagem de Parâmetro

Leitura e Escrita

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

int main(int argc, char *argv[]){
    char c; char s[100]; int i; double d;
    printf("\nEntre com um caractere: ");
    scanf("%c", &c);
    printf("\nEntre com uma palavra: ");
    scanf("%s", s);
    printf("\nEntre com um inteiro e um real: ");
    scanf("%i%lf", &i, &d);
    printf("\ninteiro(%d) real(%f) char(%c) s(%s) ", i, d, c, s);

    return 0;
}
```

printf: Função de Escrita

- Apresenta um ou mais argumentos para escrever no dispositivo padrão de saída (dps)
- O primeiro argumento é a string de controle que descreve tudo de o printf escreve no dps
- Além dos caracteres a serem escritos, a string de controle também define quais/onde variáveis serão escritas no dps

Exemplo: `printf("\ninteiro(%d) real(%f) char(%c) s(%s) ", i, d, c, s);`

printf: Função de Escrita

- Apresenta um ou mais argumentos para escrever no dispositivo padrão de saída (dps)
 - Indicamos as variáveis a serem escritas inserindo códigos de controle relativos aos tipos das mesmas
 - Os demais argumentos são as variáveis a serem exibidas no dps

Exemplo: `printf("\ninteiro(%d) real(%f) char(%c) s(%s) ", i, d, c, s);`

printf: Função de Escrita

Códigos de Controle	
%d	Número inteiro (int)
%i	Número inteiro (int)
%u	Número decimal natural (unsigned int)
%o	Número inteiro em octal
%x	Número inteiro em hexa (%X, letras maiúsculas)
%f	Número real (float ou double)
%e	Número em notação científica (%E, o e é maiúsculo)
%g	Escolha automática entre %f e %e (%G, o e é maiúsculo)
%p	Ponteiro (endereço em notação hexadecimal)
%c	Caractere (char)
%s	Sequência de caracteres (string).
%%	Imprime um %

printf: Função de Escrita

Exemplos

```
printf ("Teste %% %%")
```

Teste % %

```
printf ("%f",40.345)
```

40.345

```
printf ("Um caractere %c e um inteiro  
%d",'D',120)
```

Um caractere D e um inteiro 120

```
printf ("%s e um exemplo","Este")
```

Este e um exemplo

```
printf ("%s%d%%","Juros de ",10)
```

Juros de 10%

scanf: Função de Leitura

- Apresenta um ou mais argumentos para ler do dispositivo padrão de saída (dpe)
 - O primeiro argumento é a string de controle que descreve quais variáveis o scanf lê do dpe
 - Os demais argumentos são as variáveis a serem lidas e antes de cada variável, colocamos um & (exceto para as strings)

Exemplo: `scanf("%i%lf", &i, &d);`

scanf: Função de Leitura

Tipo	Tam.	Controle	Início	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	"	"
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	"	"
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,40E-38	3.4E+38
double	64	%lf	0,00E-01	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

```
#include <stdio.h>

int main(int argc, char *argv[]){
    int a = sizeof(char),
        b = sizeof(int),
        c = sizeof(double),
        d = sizeof(float);
    printf("tamanhos: %i --- %i --- %i --- %i", a, b, c, d);

    return 0;
}
```

Sumário

- Introdução
- Leitura e Escrita
- **String**
- Registro
- Ponteiro
- Arquivo
- Passagem de Parâmetro

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");

    ■ ■ ■
```


String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");

    ■ ■ ■
```

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");

    ■ ■ ■
```

s1

s2

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");

    ■ ■ ■
```

s1	Algoritmos
s2	e EDs

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");

    ■ ■ ■
```

s1 Algoritmos

s2 e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");
```

■ ■ ■

s1 Algoritmos

s2 e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

String

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

int main(int argc, char *argv[]){
    char s1[80], s2[80];

    strcpy(s1, "Algoritmos");
    strcpy(s2, " e EDs");

    printf("\nTamanho s1(%i)", (int)strlen(s1));
    printf("\nTamanho s2(%i)", (int)strlen(s2));

    if (strcmp(s1, s2) == 0) printf("\nIguais!!!");
    else printf("\nDiferentes!!!");
```

s1 Algoritmos

s2 e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!



String

```
■ ■ ■  
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);  
  
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);  
  
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
  
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);  
  
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;  
}
```

s1 Algoritmos

s2 e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);  
  
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);  
  
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
  
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);  
  
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;  
}
```

s1 Algoritmos e EDs

s2 e EDs

TELA

Tamanho s1(10)
Tamanho s2(6)
Diferentes!!!

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos e EDs

s2 e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos e EDs

s2 Algoritmos e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos e EDs

s2 Algoritmos e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos e EDs

s2 Algoritmos e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

Iguais!!!

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos\0e EDs

s2 Algoritmos\0e EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

Iguais!!!

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;
```

```
}
```

s1 Algoritmos\0a EDs

s2 Algoritmos\0b EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

Iguais!!!

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");  
return 0;  
}
```

s1 Algoritmos\0a EDs

s2 Algoritmos\0b EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

Iguais!!!

Nova s1 (Algoritmos)

Nova s2 (Algoritmos)

String

■ ■ ■

```
strcat(s1, s2);  
printf("\nNova s1 (%s)", s1);
```

```
strcpy(s2, s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
s1[10]=s2[10]='\0'; s1[11] = 'a'; s2[11] = 'b';  
printf("\nNova s1 (%s)", s1);  
printf("\nNova s2 (%s)", s2);
```

```
if (strcmp(s1, s2) == 0) printf("\nIguais!!!");  
else printf("\nDiferentes!!!");
```

```
return 0;
```

```
}
```

s1 Algoritmos\0a EDs

s2 Algoritmos\0b EDs

TELA

Tamanho s1(10)

Tamanho s2(6)

Diferentes!!!

Nova s1 (Algoritmos e EDs)

Nova s2 (Algoritmos e EDs)

Iguais!!!

Nova s1 (Algoritmos)

Nova s2 (Algoritmos)

Iguais!!!

Sumário

- Introdução
- Leitura e Escrita
- String
- **Registro**
- Ponteiro
- Arquivo
- Passagem de Parâmetro

Exemplo sobre Registro (1)

```
#include <stdio.h>
#define MAXTAM 100

struct Funcionario {
    int matricula;
    char nome[MAXTAM];
};

int main(int argc, char *argv[]){
    struct Funcionario f;

    printf("\nEntre com a matricula: ");
    scanf("%d", &f.matricula);
    printf("\nEntre com o nome: ");
    scanf("%s", f.nome);

    printf("\nMatricula: %d", f.matricula);
    printf("\nNome: %s", f.nome);
}
```

Exemplo sobre Registro (2)

```
#include <stdio.h>
#define MAXTAM 100

struct Funcionario {
    int matricula;
    char nome[MAXTAM];
};

typedef struct Funcionario Funcionario;

int main(int argc, char *argv[]){
    Funcionario f;

    printf("\nEntre com a matricula: ");
    scanf("%d", &f.matricula);
    printf("\nEntre com o nome: ");
    scanf("%s", f.nome);

    printf("\nMatricula: %d", f.matricula);
    printf("\nNome: %s", f.nome);
}
```

Exemplo sobre Registro (3)

```
#include <stdio.h>
#define MAXTAM 100

typedef struct Funcionario {
    int matricula;
    char nome[MAXTAM];
} Funcionario;

int main(int argc, char *argv[]){
    Funcionario f;

    printf("\nEntre com a matricula: ");
    scanf("%d", &f.matricula);
    printf("\nEntre com o nome: ");
    scanf("%s", f.nome);

    printf("\nMatricula: %d", f.matricula);
    printf("\nNome: %s", f.nome);
}
```

Exemplo sobre Registro (4)

```
#include <stdio.h>
#define MAXTAM 100

typedef struct Funcionario { int matricula; char nome[MAXTAM]; } Funcionario;

int main(int argc, char *argv[]){
    Funcionario vet[MAXTAM];
    for (int i = 0; i < 3; i++){
        printf("\nEntre com a matricula do funcionário %d: ", (i+1));
        scanf("%d", &vet[i].matricula);
        printf("\nEntre com o nome do funcionário %d: ", (i+1));
        scanf("%s", vet[i].nome);
    }
    for (int i = 0; i < 3; i++){
        printf("\nMatricula do funcionário %d: %d", (i+1), vet[i].matricula);
        printf("\nNome do funcionário %d: %s", (i+1), vet[i].nome);
    }
}
```

Sumário

- Introdução
- Leitura e Escrita
- String
- Registro
- **Ponteiro**
- Arquivo
- Passagem de Parâmetro

Introdução

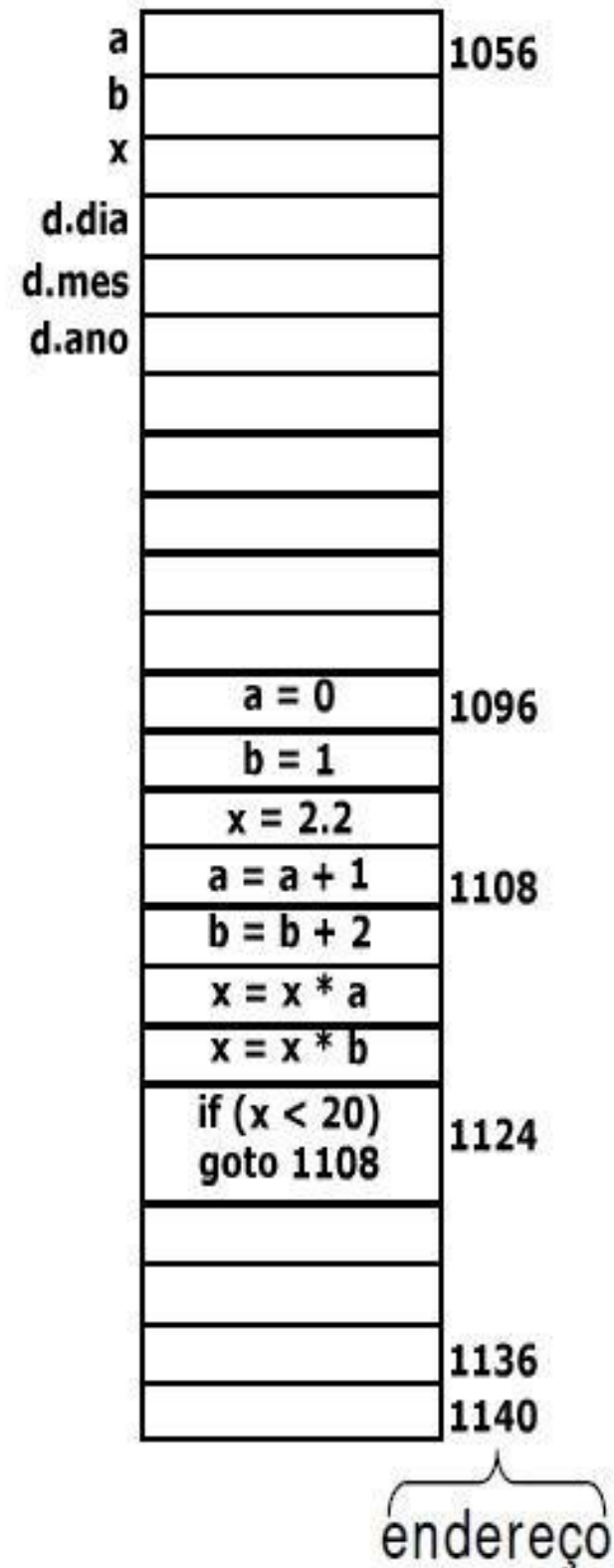
- Um programa utiliza duas áreas de memória: uma para os dados e outra para o código fonte (ou seja, as instruções)
- Uma variável possui conteúdo e endereço
 - Seu conteúdo pode alterar durante a execução do programa
 - Seu endereço é constante

```

void main() { // CONTEÚDO    ENDEREÇO
  int a;      // lixo        1056
  int b;      // lixo        1060
  float x;    // lixo        1064
  Data d;     // lixo        1068

  a = 0;      // 0           1056
  b = 1;      // 1           1060
  x = 2.2f;    // 2.2        1064
  do {
    a = a + 1;
    b = b + 2;
    x = x * a;
    x = x * b;
  } while (x < 20.0f);

```



Alocações Estática e Dinâmica

- São as duas formas para alocarmos de alocar os dados
- Na estática, o SO reserva o espaço de memória das variáveis quando ele começa a executar um programa e essa reserva não pode ser alterada

int a; int b[20];

- Na dinâmica, o SO reserva esse espaço durante a execução do programa e essa reserva pode ser alterada

Alocação Dinâmica

- A memória alocada dinamicamente está localizada em uma área chamada de *heap* e, basicamente, o programa aloca e desaloca porções de memória do *heap* durante a execução
- Acessamos as posições de memória alocadas dinamicamente através de apontadores ou ponteiros

Ponteiros

- São variáveis que armazenam um endereço de memória
- Da mesma forma que um int armazena inteiro; um double, número real; um ponteiro armazena um endereço de memória
- Os ponteiros possuem tipo, ou seja, temos ponteiro para endereços de memória de um int, de um float, de um char...

Declaração de Ponteiros

tipoPonteiro *nomeVariável;

- O asterisco na declaração de uma variável indica que essa não guardará um valor e sim um endereço para o tipo especificado

Operadores

- Operador endereço (**&**) determina o endereço de uma variável
- Operador de conteúdo de um ponteiro (*****) determina o conteúdo da posição de memória endereçada pelo ponteiro

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

Memória	
	75h
	...
	CDh

Tela

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = & x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

x	Memória	
	10	75h
		...
		CDh

Tela

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = &x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;
```

```
int * y = & x;
```

```
printf("\n%i", x);
```

```
printf("\n%p", &x);
```

```
printf("\n%p", y);
```

```
printf("\n%p", &y);
```

```
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h
CDh

Exemplo

- Faça o quadro de memória e mostre a saída na tela:

```
int x = 10;  
  
int * y = & x;  
  
printf("\n%i", x);  
  
printf("\n%p", &x);  
  
printf("\n%p", y);  
  
printf("\n%p", &y);  
  
printf("\n%i", *y);
```

	Memória	
x	10	75h
		...
y	75h	CDh

Tela
10
75h
75h
CDh
10

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	?	89Bh
x2	?	89Ch
x3	?	89Dh
p	?	89Eh

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	22	89Ch
x3	33	89Dh
p	?	89Eh

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	22	89Ch
x3	33	89Dh
p	89Bh	89Eh

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Bh	89Eh

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	33	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Bh	89Eh

Exemplo

```
int x1, x2, x3; int *p;
```

```
x1 = 11; x2 = 22; x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	33	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Dh	89Eh

Exemplo

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	33	89Bh
x2	11	89Ch
x3	0	89Dh
p	89Dh	89Eh

Exercício

- Faça o quadro de memória e mostre a saída na tela:

```
int *x1;      int x2;      int *x3;

x1 = (int *) malloc (sizeof(int));
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

*x1 = 20;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x2 = *x1;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

*x3 = x2 * *x1;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x3 = &x2;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x2 = 15;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);
```

```
double M [3][3];
double *p = M[0];
for (int i = 0; i < pow(MAXTAM, 2); i++, p++){
    *p=0.0;
}
```

R:

[illegible]

Observações

- Os símbolos usados para notação de ponteiros em C/C++ não são tão claros como deveriam ser
- Descuidado com ponteiros \Rightarrow problemas
- **Atenção:** Sempre inicialize os ponteiros

Observações

- $p1 = p2$: faz com que eles apontem para o mesmo local
- $*p1 = *p2$: Atribui o conteúdo apontado por p2 o por p1
- $p++$, $p--$, $p=p+5$ e $p+=3$: Incrementa e decrementa o valor do endereço apontado pelo ponteiro, fazendo com que o ponteiro antes $\text{sizeof}(\text{tipoPonteiro})$ bytes na memória

Observações

- $(*p)++$ e $(*p) --$: Incrementa / decrementa o conteúdo da variável apontada pelo ponteiro p
- Os testes relacionais como $>$, $<$, $>=$, $<=$, $==$ ou $!=$ são aceitos apenas para ponteiros do mesmo tipo, contudo, eles comparam endereços

Alocar Memória em C: malloc

- Protótipo da função malloc()

void* malloc (int tamanho)

- O malloc aloca o número de bytes passados como parâmetro e retorna um ponteiro para a primeira posição da área alocada

Desalocar Memória em C: free()

- Protótipo da função free()

void free (void*)

- O free desaloca o espaço de memória apontado pelo ponteiro recebido como parâmetro

Exemplo do malloc() e do free()

```
char* p1 = (char*) malloc (sizeof(char));  
int* p2 = (int*) malloc (sizeof(int));  
float* p3 = (float*) malloc (sizeof(float));  
Cliente* p4 = (Cliente*) malloc (sizeof(Cliente));  
int* p5 = (int*) malloc (MAXTAM * sizeof (int));  
Cliente* p6 =(Cliente*) malloc (MAXTAM * sizeof (Cliente));  
free(p1);  
free(p2);  
free(p3);  
free(p4);  
free(p5);  
free(p6);
```

Alocar/Desalocar Memória em C++: new e delete

```
char* p1 = new char;
```

```
int* p2 = new int;
```

```
float* p3 = new float;
```

```
Cliente* p4 = new Cliente;
```

```
int* p5 = new int [MAXTAM];
```

```
Cliente* p6 = new Cliente[MAXTAM];
```

```
delete p1;
```

```
delete p2;
```

```
delete p3;
```

```
delete p4;
```

```
delete [ ] p5;
```

```
delete [ ] p6;
```

Registro vs. Ponteiro

- Acessamos um atributo de um registro fazendo *registro.atributo*
- Acessamos um atributo de um registro apontado por um ponteiro fazendo *ponteiro->atributo*

```
Cliente registro;  
Cliente* ponteiro = (Cliente*) malloc (sizeof(Cliente));  
registro.codigo = 1;  
strcpy(registro.nome, "AA");  
printf("\nFuncionario (%i): %s", registro.codigo, registro.nome);  
ponteiro->codigo = 2;  
strcpy(ponteiro->nome, "BB");  
printf("\nFuncionario (%i): %s", ponteiro->codigo, ponteiro->nome);
```

Erros Comuns

- Esquecer de alocar memória e tentar acessar o conteúdo da variável
- Copiar o valor do apontador quando deveria ser o conteúdo da variável apontada
- Esquecer de desalocar memória
 - O SO a desaloca no final do programa ou da função onde a variável está declarada
- Tentar acessar o conteúdo da variável depois de desalocá-la

Exercício

- Mostre a saída na tela

```
double a;  
double *p, *q;  
a = 3.14;  
printf("%f\n", a);  
p = &a;  
*p = 2.718;  
printf("%f\n", a);  
a = 5;  
printf("%f\n", *p);
```

```
p = NULL;  
p = (double*) malloc(sizeof(double));  
*p = 20;  
q = p;  
printf("%f\n", *p);  
printf("%f\n", a);  
free(p);  
printf("%f\n", *q);
```

R: Saída da tela:
3.140000

Exercício

- Mostre o quadro de memória

```
int a[10], *b;
```

```
b = a;
```

```
b[5] = 100;
```

```
printf("\n%d -- %d", a[5], b[5]);
```

```
b = (int*) malloc(10*sizeof(int));
```

```
b[7] = 100;
```

```
printf("\n%d -- %d", a[7], b[7]);
```

//O comando a = b gera um erro de compilação

R:

Índice i | Valor de a[i] | Endereço de a[i] | Valor de b | Valor de b[i] | Endereço de b[i]

expression must be a modifiable lvalueC/C++(137)View Problem (Alt+F8)

Exercício

• Mostre o quadro de memória

```
int *x1;      int x2;      int *x3;
```

```
x1 = (int*) malloc(sizeof(int));
```

```
*x1 = 20; R:
```

```
x2 = *x1;
```

```
*x3 = x2 * *x1;
```

```
x3 = &x2;
```

```
x2 = 15;
```

```
x2 = 13 & 3;
```

```
x2 = 13 | 3;
```

```
x2 = 13 >> 1;
```

```
x2 = 13 << 1;
```

1. Após a alocação de memória para x1:

[x1] --> Endereço alocado para um inteiro
[x2]
[x3]

2. Após a atribuição de valor 20 a *x1:

[x1] --> Valor: 20
[x2]
[x3]

3. Após a atribuição de *x1 a x2:

4. [x1] --> Valor: 20
[x2] --> Valor: 20
[x3]

5. Após a tentativa de atribuição a *x3 (ERRO - x3 não aponta para local alocado):

[x1] --> Valor: 20
[x2] --> Valor: 20
[x3]

6. Após a atribuição de endereço de x2 a x3:

[x1] --> Valor: 20
[x2] --> Valor: 20
[x3] --> Aponta para o endereço de x2

7. Após a atribuição do valor 15 a x2:

[x1] --> Valor: 20
[x2] --> Valor: 15
[x3] --> Aponta para o endereço de x2

8. Após a operação $x2 = 13 \& 3$ (AND bit a bit):

[x1] --> Valor: 20
[x2] --> Valor: 1
[x3] --> Aponta para o endereço de x2

9. Após a operação $x2 = 13 | 3$ (OR bit a bit):

[x1] --> Valor: 20
[x2] --> Valor: 15
[x3] --> Aponta para o endereço de x2

10. Após a operação $x2 = 13 \gg 1$:

[x1] --> Valor: 20
[x2] --> Valor: 6
[x3] --> Aponta para o endereço de x2

11. Após a operação $x2 = 13 \ll 1$:

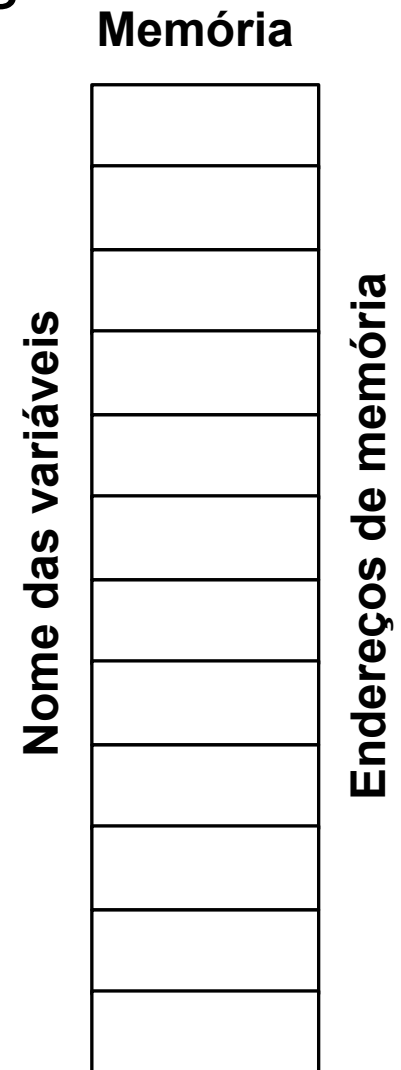
[x1] --> Valor: 20
[x2] --> Valor: 12
[x3] --> Aponta para o endereço de x2

Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;  
c.codigo = 5;  
Cliente *p = NULL;  
p = (Cliente*) malloc (sizeof(Cliente));  
p->codigo = 6;  
Cliente *p2 = &c;  
p2->codigo = 7;
```

Representação gráfica



Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;
```

```
c.codigo = 5;
```

```
Cliente *p = NULL;
```

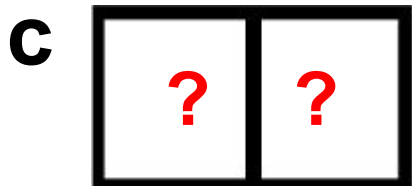
```
p = (Cliente*) malloc (sizeof(Cliente));
```

```
p->codigo = 6;
```

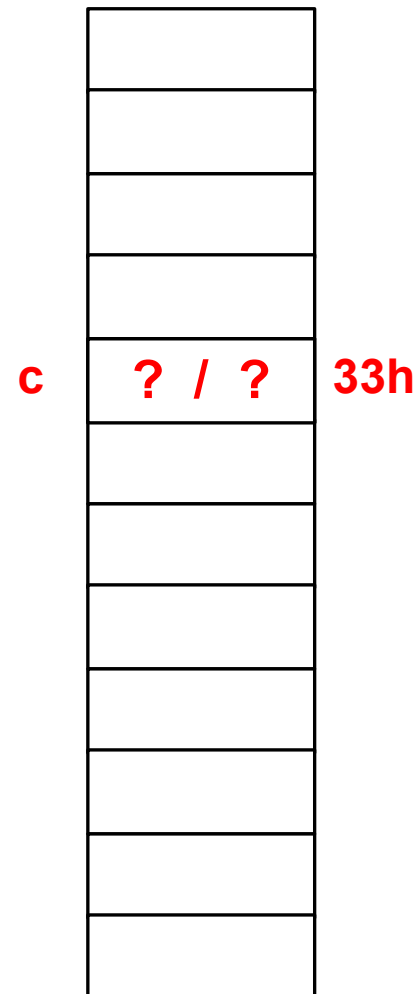
```
Cliente *p2 = &c;
```

```
p2->codigo = 7;
```

Representação gráfica



Memória

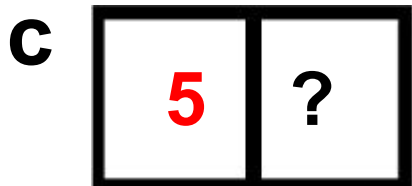


Exercício

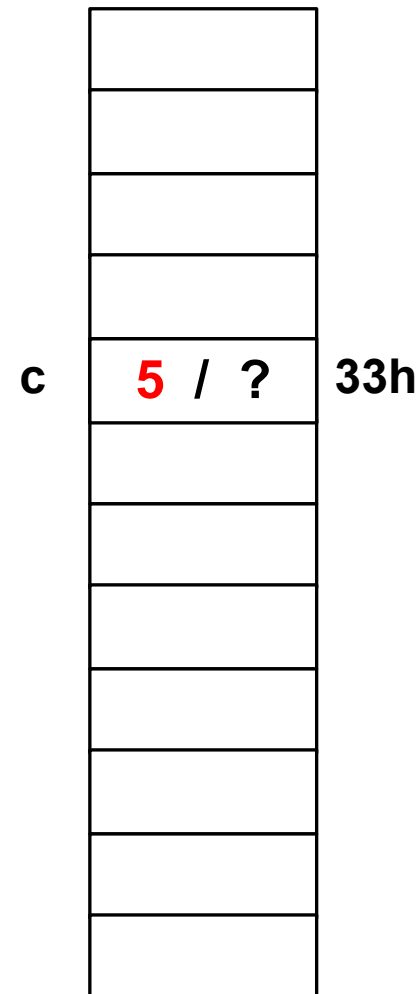
- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;  
c.codigo = 5;  
Cliente *p = NULL;  
p = (Cliente*) malloc (sizeof(Cliente));  
p->codigo = 6;  
Cliente *p2 = &c;  
p2->codigo = 7;
```

Representação gráfica



Memória

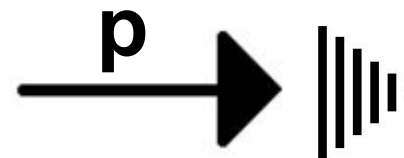
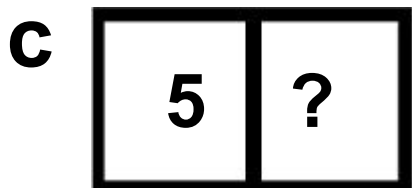


Exercício

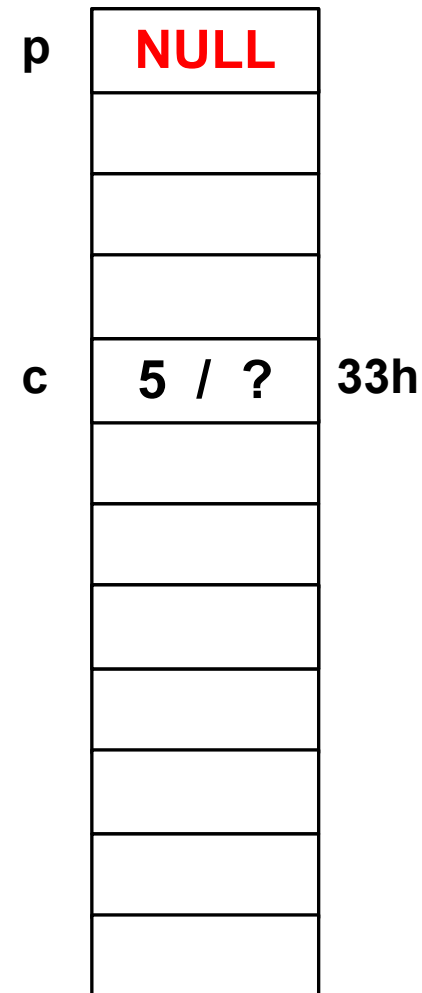
- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;  
c.codigo = 5;  
Cliente *p = NULL;  
p = (Cliente*) malloc (sizeof(Cliente));  
p->codigo = 6;  
Cliente *p2 = &c;  
p2->codigo = 7;
```

Representação gráfica



Memória



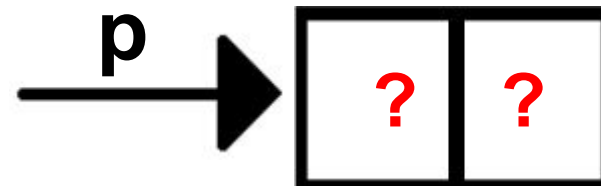
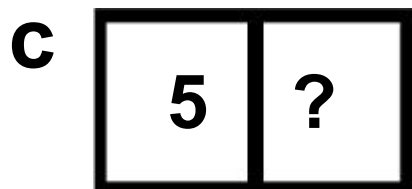
Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

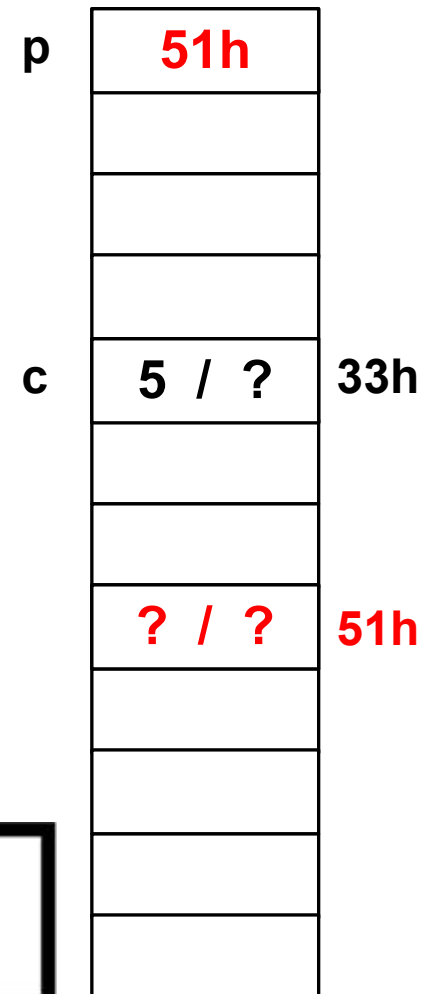
```

Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
    
```

Representação gráfica



Memória

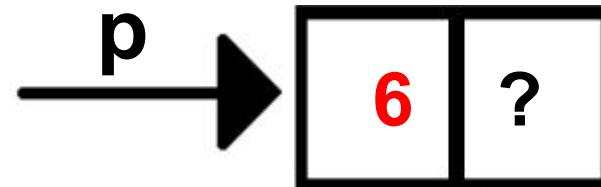
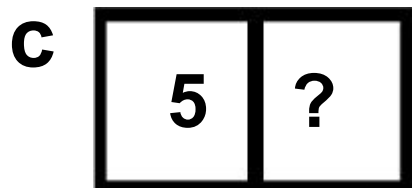


Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;  
c.codigo = 5;  
Cliente *p = NULL;  
p = (Cliente*) malloc (sizeof(Cliente));  
p->codigo = 6;  
Cliente *p2 = &c;  
p2->codigo = 7;
```

Representação gráfica



Memória

p	51h	
c	5 / ?	33h
		51h
	6 / ?	

Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
    
```

Memória

p	51h	
p2	33h	
c	5 / ?	33h
	6 / ?	51h

Representação gráfica



Exercício

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

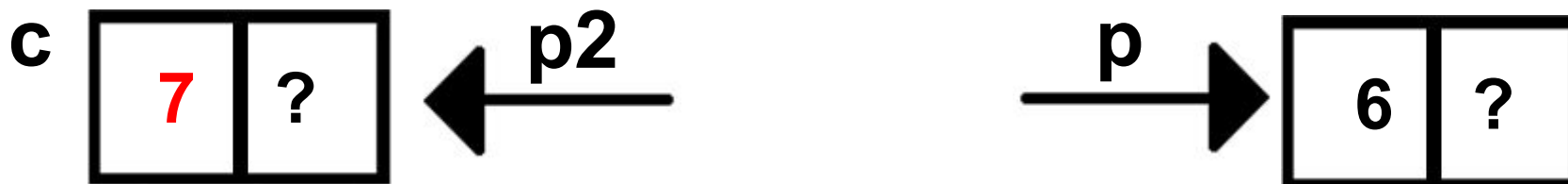
```

Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
    
```

Memória

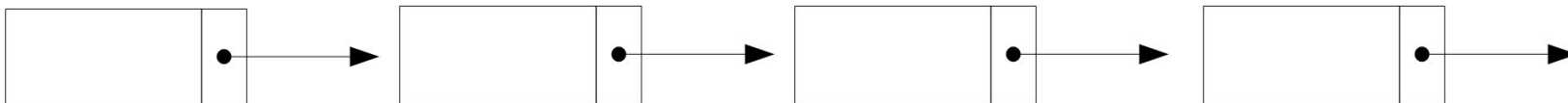
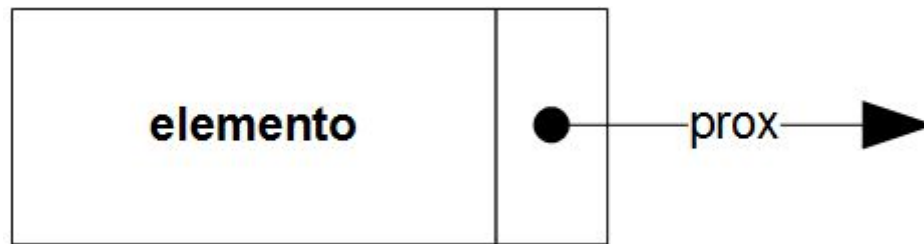
p	51h	
p2	33h	
c	7 / ?	33h
	6 / ?	51h

Representação gráfica



Exercício

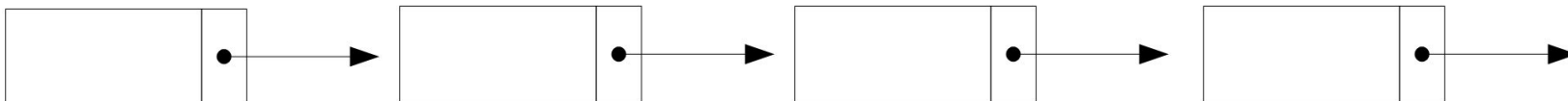
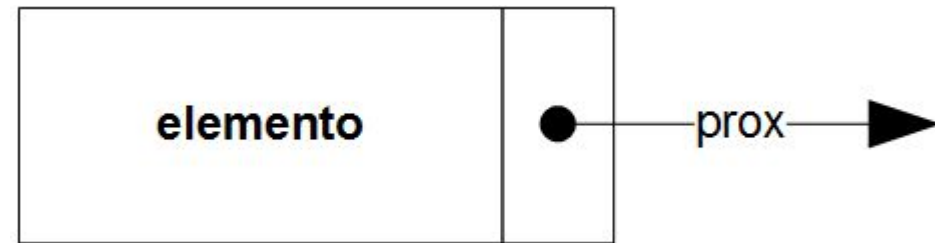
- Crie um registro célula contendo os atributos elemento (inteiro) e prox (apontador para outra célula)



Exercício

- Crie um registro célula contendo os atributos elemento (inteiro) e prox (apontador para outra célula)

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```



Exercício

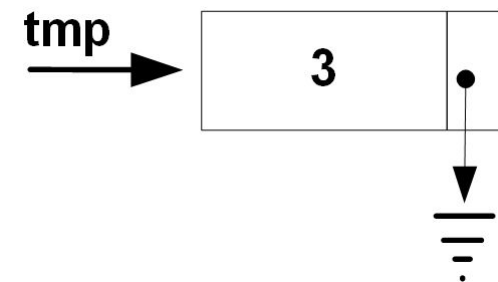
- Mostre o que acontece se um método tiver o comando *Celula*
**tmp = novaCelula(3)*.

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```

Exercício

- Mostre o que acontece se um método tiver o comando *Celula* $*tmp = novaCelula(3)$.

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```



Exercícios Gráficos: Java, C e C++

Exercícios Gráficos em Java

Exercícios Gráficos: Java, C e C++

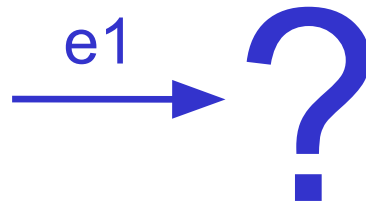
- Represente graficamente o código Java abaixo

Elemento e1;

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código Java abaixo

Elemento e1;



Exercícios Gráficos: Java, C e C++

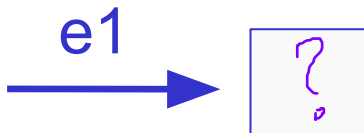
- Represente graficamente o código Java abaixo

```
Elemento e1 = new Elemento();
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código Java abaixo

```
Elemento e1 = new Elemento();
```



Exercícios Gráficos: Java, C e C++

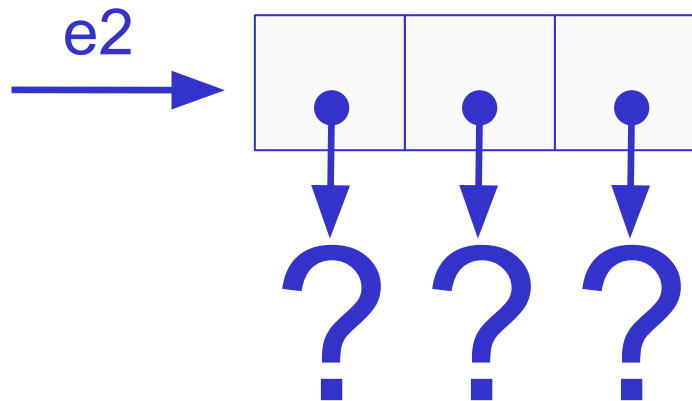
- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```



Exercícios Gráficos: Java, C e C++

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

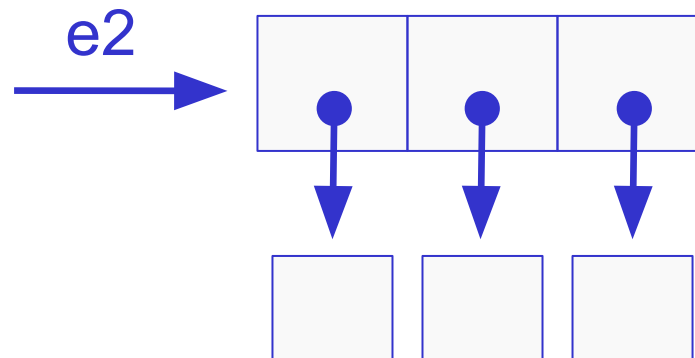
```
for (int i = 0; i < 3; i ++){  
    e2[i] = new Elemento();  
}
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

```
for (int i = 0; i < 3; i++){  
    e2[i] = new Elemento();  
}
```



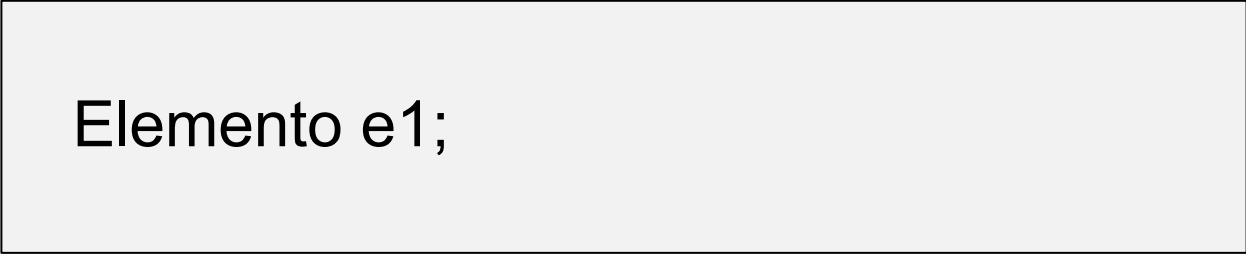
Exercícios Gráficos: Java, C e C++

Exercícios Gráficos

em C

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

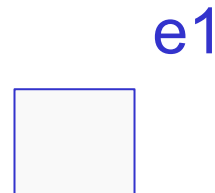


Elemento e1;

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento e1;
```



Representação da alocação de um espaço na memória!!

Exercícios Gráficos: Java, C e C++

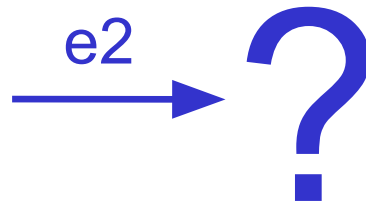
- Represente graficamente o código C abaixo

```
Elemento* e2;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento* e2;
```



Representação da alocação de um espaço na memória
de um ponteiro (e2), que por sua vez também aponta para algo!!

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

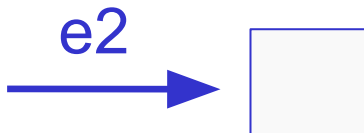
```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));
```

Representação da alocação de um espaço na memória
de um ponteiro (e2), que por sua vez também aponta para um
elemento de espaço equivalente à "Elemento"

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));
```



Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(3*sizeof(Elemento));
```

Representação da alocação de um espaço na memória de um ponteiro (e2), que por sua vez também aponta para um elemento de espaço equivalente à 3 "Elemento"

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(3*sizeof(Elemento));
```



Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento e3[3];
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

Elemento e3[3];



Representação da alocação de um espaço na memória
de um vetor de Elemento's (e3)

Exercícios Gráficos: Java, C e C++

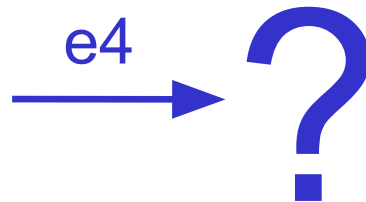
- Represente graficamente o código C abaixo

```
Elemento** e4;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento** e4;
```



Ponteiro de um ponteiro (e4)

Exercícios Gráficos: Java, C e C++

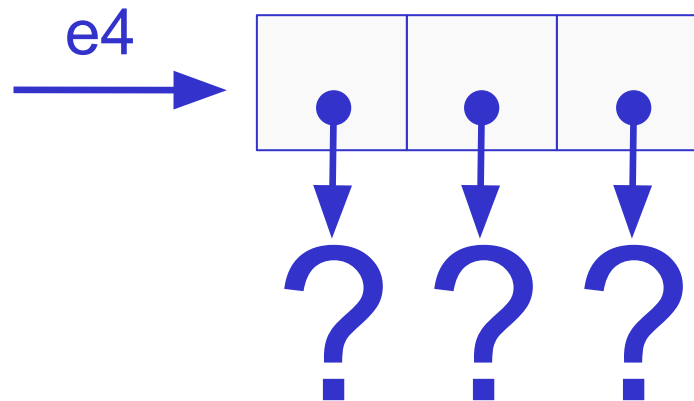
- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
```



Representação da alocação de um espaço na memória de um ponteiro de um ponteiro (e4), que por sua vez também aponta para um elemento de espaço equivalente à "Elemento"

Exercícios Gráficos: Java, C e C++

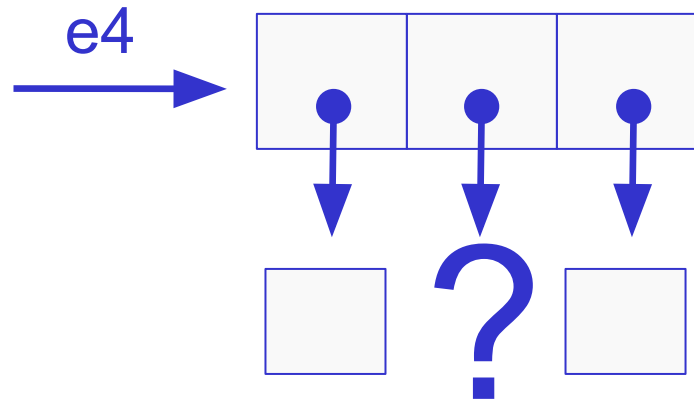
- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));  
e4[0] = (Elemento*) malloc(sizeof(Elemento*));  
e4[2] = (Elemento*) malloc(sizeof(Elemento*));
```


Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));  
e4[0] = (Elemento*) malloc(sizeof(Elemento*));  
e4[2] = (Elemento*) malloc(sizeof(Elemento*));
```



Exercícios Gráficos: Java, C e C++

Exercícios Gráficos

C++

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento e1;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

Elemento e1;

e1



Exercícios Gráficos: Java, C e C++

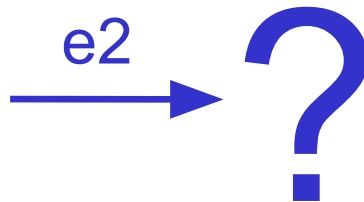
- Represente graficamente o código C++ abaixo

```
Elemento* e2;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento* e2;
```



Exercícios Gráficos: Java, C e C++

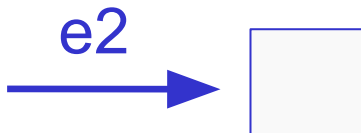
- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;
```



Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento[3];
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento[3];
```



Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento e3[3];
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

Elemento e3[3];



Exercícios Gráficos: Java, C e C++

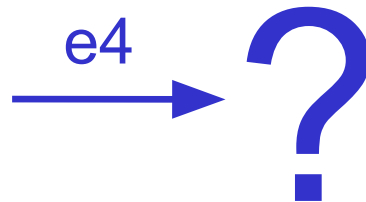
- Represente graficamente o código C++ abaixo

```
Elemento** e4;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento** e4;
```



Exercícios Gráficos: Java, C e C++

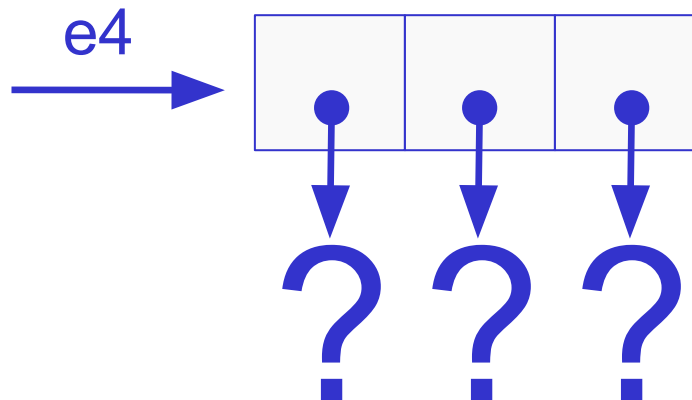
- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
```



Exercícios Gráficos: Java, C e C++

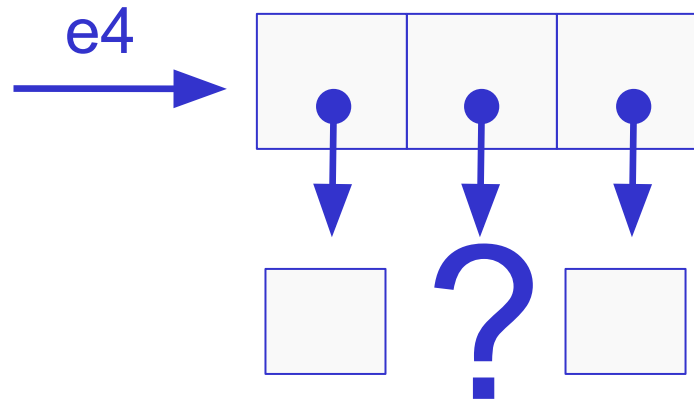
- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];  
e4[0] = new Elemento;  
e4[2] = new Elemento;
```

Exercícios Gráficos: Java, C e C++

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];  
e4[0] = new Elemento;  
e4[2] = new Elemento;
```



Sumário

- Introdução
- Leitura e Escrita
- String
- Registro
- Ponteiro
- **Arquivo**
- Passagem de Parâmetro

Definição de Arquivo

- Unidade lógica utilizada para armazenar dados em disco ou em qualquer outro dispositivo externo de armazenamento
- Pode-se abrir, fechar, ler, escrever ou apagar um arquivo



Biblioteca para Arquivos na Linguagem C

- A linguagem C manipula tanto arquivos quanto dispositivos de I/O usando o tipo ponteiro para arquivo
- A biblioteca *stdio.h* possui funções para trabalhar com arquivos

Declaração de um Ponteiro para Arquivo

```
FILE *p;
```

- Protótipo da função fopen()

FILE *fopen (char *nomeArquivo, char *modo);

- Protótipo da função fopen()

FILE *fopen (char *nomeArquivo, char *modo);

Abrir Arquivo

- Protótipo da função fopen()

r - abre um arquivo existente para leitura

w - cria um arquivo para escrita. Se ele existir, o SO
apaga o conteúdo atual

a - abre um arquivo para escrita no final. Se o arquivo
não existir, o SO tenta criá-lo

char *modo);

Abrir Arquivo

- Protótipo da função `fopen()`

r+ - igual ao **r** sendo que o **+** permite a escrita

w+ - igual ao **w** sendo que o **+** permite a leitura

a+ - igual ao **a** sendo que o **+** permite a leitura

char *modo);

Abrir Arquivo

- Protótipo da função `fopen()`

`rb`

`wb`

`ab`

`rb+`

`wb+`

`ab+`

igual aos anteriores, contudo,
para o modo binário

`char *modo);`

Modos Texto e Binário

- **Modo texto**: o arquivo fica armazenado como uma sequência de caracteres, permitindo a organização em linhas por um caractere de nova linha
- **Modo binário**: o arquivo fica armazenado como uma sequência de bytes, permitindo uma relação um para um com o arquivo real

Exemplo

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

int main(int argc, char *argv[]) {
    FILE *p = fopen ("teste.txt", "r");

    // testa se o arquivo foi aberto com sucesso
    if (p != NULL) {
        printf ("\nArquivo foi aberto com sucesso.");
    } else {
        printf ("\nNao foi possivel abrir o arquivo.");
    }

    return 0;
}
```

Fechar Arquivo

- Protótipo da função fclose()

int fclose (FILE *p);

- Devemos fechar um arquivo após a leitura / escrita do mesmo
- O fechamento de um arquivo garante que dados remanescentes no “buffer” serão grafos no arquivo

Exemplo

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

int main(int argc, char *argv[]) {
    FILE *p = fopen ("teste.txt", "r");

    // testa se o arquivo foi aberto com sucesso
    if (p != NULL) {
        printf ("\nArquivo foi aberto com sucesso.");
        fclose(p);
    } else {
        printf ("\nNao foi possivel abrir o arquivo.");
    }

    return 0;
}
```

Leitura de um Caractere

- Protótipo da função `fgetc()`

`int fgetc(FILE *p);`

- O `fgetc` retorna um inteiro que corresponde ao código ASCII de um caractere, um EOF (definido em `stdio.h`) ou alguma condição de erro
- O EOF é uma constante definida na `stdio.h` que indica *end of file*

Leitura de um Caractere

- Exemplo de programa que lê os caracteres de um arquivo até seu final

```
int main(int argc, char *argv[]) {  
    FILE *p = fopen ("teste.txt", "r");  
    int ch;  
    if (p != NULL) {  
        do {  
            ch = fgetc(p);  
            printf( "%c", (char)ch);  
        } while (ch != EOF);  
        fclose(p);  
    }  
    return 0;  
}
```

Leitura de Mais Caracteres

- Protótipo da função fgets()

char* fgets(char *s, int size, FILE *p);

Leitura de Mais Caracteres

- Protótipo da função `fgets()`

`char* fgets(char *s, int size, FILE *p);`

- O primeiro parâmetro é o endereço de um *array* de caracteres para receber os caracteres lidos
- Observa-se que o programador deve garantir que esse *array* tem um tamanho mínimo de `size` caracteres (cuidado com erros bizantinos)

Leitura de Mais Caracteres

- Protótipo da função fgets()

char* fgets(char *s, int size, FILE *p);

- O segundo parâmetro é o número máximo de caracteres a serem lidos
 - Observa-se que se o fgets encontrar um caractere de final de linha, ele pára nesse ponto e retorna os caracteres lidos

Leitura de Mais Caracteres

- Protótipo da função `fgets()`

`char* fgets(char *s, int size, FILE *p);`

- O último parâmetro é o manipulador do arquivo

Leitura de Mais Caracteres

- Protótipo da função `fgets()`

`char*` fgets(`char *s`, `int size`, `FILE *p`);

- O `fgets` retorna um ponteiro para o início do array contendo os caracteres lidos ou um `NULL` caso o final do arquivo tenha sido atingido.
- Observa-se que se o *array* tiver espaço para o final de linha (`'\0'`), o `fgets` o insere no final da sequência de caracteres lida

Leitura de Mais Caracteres

- Exemplo de programa que lê os caracteres de um arquivo até seu final

```
int main(int argc, char *argv[]) {  
    FILE *p = fopen ("teste.txt", "r");  
    char str[100+1];  
    char* resp;  
    if (p != NULL) {  
        do {  
            resp = fgets(str, 100, p);  
            if (resp != NULL) { printf("%s\n", str); }  
        } while (resp != NULL);  
        fclose(p);  
    }  
    return 0;  
}
```

Escrita de um Caractere

- Protótipo da função `fputc()` - a função `putc()` é idêntica

`int fputc(int ch, FILE *p);`

- O primeiro parâmetro é o caractere a ser inserido e o segundo, o manipulador do arquivo
- O `fputc` retorna o caractere escrito em caso de sucesso. Caso contrário, retorna o EOF

Escrita de um Caractere

- Exemplo de programa que escreve um caractere em um arquivo

```
int main(int argc, char *argv[]) {  
    FILE *p = fopen ("teste.txt", "w");  
    fputc('M', p);  
    fclose(p);  
    return 0;  
}
```

Escrita de Mais Caracteres

- Protótipo da função `fputs()`

`int fputs(const char *s, FILE *p);`

- O primeiro parâmetro é a sequência de caracteres a serem inseridos e o segundo, o manipulador do arquivo
- O `fputs` retorna um EOF em caso de erro

Escrita de Mais Caracteres

- Exemplo de programa que escreve vários caracteres em um arquivo

```
int main(int argc, char *argv[]) {  
    FILE *p = fopen ("teste.txt", "w");  
    fputs("Algoritmos e Estruturas de Dados II", p);  
    fclose(p);  
    return 0;  
}
```

Fim de Arquivo: feof()

- Protótipo da função feof()

```
int feof(FILE *p);
```

- Retorna verdadeiro se o final do arquivo foi alcançado e, caso contrário, falso

Fim de Arquivo: feof()

```
int main(int argc, char *argv[]) {  
  
    FILE *in = fopen ("teste.txt", "rb"),  
          *out = fopen ("copia.txt", "wb");  
  
    while ( ! feof(in) ) {  
        char ch = getc(in);  
        if ( ! feof(in)) putc(ch, out);  
    }  
  
    fclose(in);  
    fclose(out);  
    return 0;  
}
```

Leitura e Escrita de Blocos

- Protótipo da função fread()

size_t fread(void *buffer, size_t nByte, size_t cont, FILE *p);

- O primeiro parâmetro é um ponteiro para uma área de memória em que os dados serão armazenados
- O segundo é o número de bytes a serem inseridos no arquivo
- O terceiro é o número de itens (de tamanho nByte) serão lidos
- O quarto é o manipulador do arquivo

Leitura e Escrita de Blocos

- Protótipo da função fread()

size_t fread(void *buffer, size_t nByte, size_t cont, FILE *p);

- O fread retorna o número de itens lidos
- size_t é aproximadamente o mesmo que o **unsigned**

Leitura e Escrita de Blocos

- Protótipo da função fwrite()

size_t fwrite(void *buffer, size_t nByte, size_t cont, FILE *p);

- Os parâmetros e o retorno são similares aos do fread

Exemplo de Leitura e Escrita de Blocos

```
int main(int argc, char *argv[]) {  
    FILE *p;  
  
    double d = 12.23;  
    int i = 101;  
    long l = 123023L;  
  
    if ((p = fopen("teste.txt", "wb")) == NULL) {  
        printf ("Arquivo nao pode ser aberto!!!");  
        exit(1);  
    }  
  
    fwrite(&d, sizeof(double), 1, p);  
    fwrite(&i, sizeof(int), 1, p);  
    fwrite(&l, sizeof(long), 1, p);  
  
    fclose(p);  
  
    return 0;  
}
```

Exemplo de Leitura e Escrita de Blocos

```
int main(int argc, char *argv[]) {  
    FILE *p;  
  
    double d;  
    int i;  
    long l;  
  
    if ((p = fopen("teste.txt", "rb")) == NULL) {  
        printf ("\nArquivo nao pode ser aberto!!!");  
        exit(1);  
    }  
  
    fread(&d, sizeof(double), 1, p);  
    fread (&i, sizeof(int), 1, p);  
    fread (&l, sizeof(long), 1, p);  
    printf("%f %d %ld", d, i, l);  
    fclose(p);  
    return 0;  
}
```

Exemplo de Leitura e Escrita de Blocos

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

typedef struct Cliente {
    char nome[100];
    int codigo;
} Cliente;

void escrever(char*);
void ler(char*);

void main(int argc, char** argv){
    escrever("teste.txt");
    ler("teste.txt");
}
```



Exemplo de Leitura e Escrita de Blocos

```
void escrever(char* nomeArq){
    cliente c1, c2;
    strcat(c1.nome, "Ze da Silva");    c1.codigo = 1;
    strcat(c2.nome, "Lele da Cuca");   c2.codigo = 11;
    FILE *p = fopen(nomeArq, "ab");
    fwrite(&c1, sizeof(Cliente), 1, p);
    fwrite(&c2, sizeof(Cliente), 1, p);
    fclose(p);
}

void ler(char* nomeArq){
    cliente c1, c2;
    FILE *p = fopen(nomeArq, "rb");
    fread(&c1, sizeof(Cliente), 1, p);
    fread(&c2, sizeof(Cliente), 1, p);
    fclose(p);
    printf("%s -- %d\n", c1.nome, c1.codigo);
    printf("%s -- %d\n", c2.nome, c2.codigo);
}
```

Exercício

- Faça um programa que leia n números inteiros, armazene-os em um arquivo, leia-os do arquivo e mostre-os na tela

Cabeçote de Leitura e Escrita

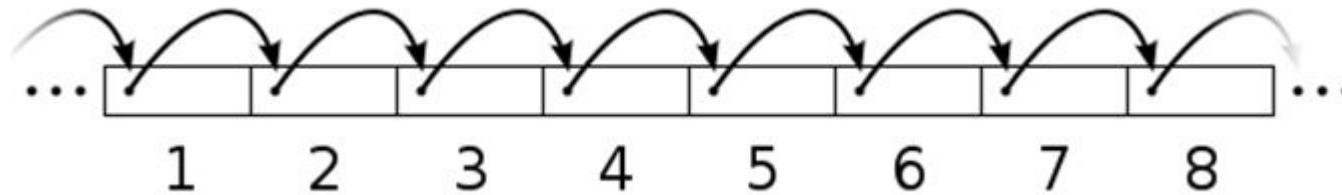
- Indica a posição atual de leitura/escrita no arquivo
- Após uma leitura/escrita o cabeçote se desloca em uma unidade em direção ao final do arquivo

Reiniciar o Cabeçote: rewind()

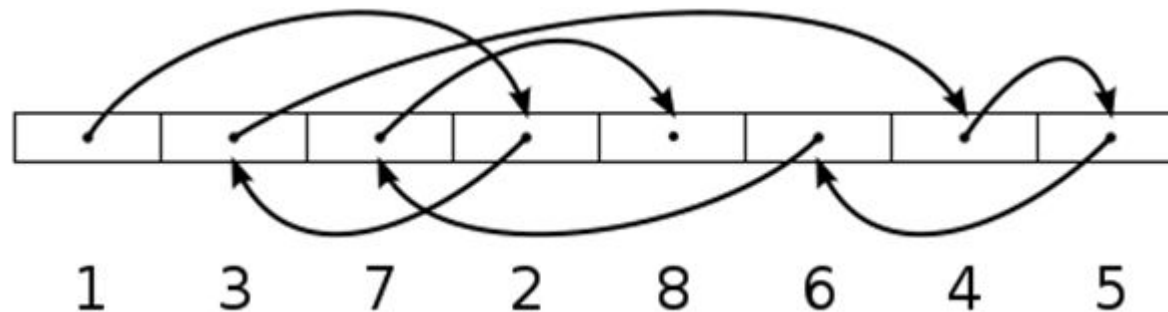
```
int main(int argc, char *argv[]) {  
    FILE *p = fopen("teste.txt", "wb");  
  
    double d = 12.23;      int i = 101;      long l = 123023L;  
  
    fwrite(&d, sizeof(double), 1, p);  
    fwrite(&i, sizeof(int), 1, p);  
    fwrite(&l, sizeof(long), 1, p);  
  
    rewind(p);  
  
    fread(&d, sizeof(double), 1, p);  
    fread (&i, sizeof(int), 1, p);  
    fread (&l, sizeof(long), 1, p);  
    printf("%f %d %ld", d, i, l);  
    fclose(p);  
  
    return 0;  
}
```

Acessos Sequencial e Aleatório

- Acesso Sequencial



- Acesso Aleatório



Acesso Aleatório com fseek() e ftell()

- Protótipo da função fseek():

int fseek(FILE* p, long nBytes, int origem);

Origem	Macro
Início do arquivo	SEEK_SET
Posição atual	SEEK_CUR
Fim do arquivo	SEEK_END

Acesso Aleatório com fseek() e ftell()

- Protótipo da função ftell():

```
long ftell(FILE* p);
```

Acesso Aleatório com fseek() e ftell()

```
int main(int argc, char *argv[]) {  
    FILE *p = fopen("teste.txt", "wb+");  
    int registro, valor, i;  
  
    for (i = valor = 0; i < 10; i++, valor = i * 10)    fwrite(&valor, sizeof (int), 1, p);  
  
    int numRegistro = ftell(p) / sizeof (int);  
  
    do {  
        printf ("\nEscolha um numero entre zero e %i: ", numRegistro-1);  
        scanf("%d", &registro);  
    } while (registro < 0 || registro >= numRegistro);  
  
    fseek(p, registro * sizeof (int), SEEK_SET);  
    fread(&valor, sizeof (int), 1, p);  
    fclose(p);  
    printf ("\nValor: %d\n\n", valor);  
  
    return 0;  
}
```

Exercício

- Faça um programa que leia n números inteiros e os mostre em ordem inversa sem usar *arrays*

Exercício

- Faça um programa que leia n números inteiros e os mostre a soma do primeiro e último, segundo e penúltimo e assim sucessivamente.

Novamente, sem usar *arrays*

Sumário

- Introdução
- Leitura e Escrita
- String
- Registro
- Ponteiro
- Arquivo
- **Passagem de Parâmetro**

Passagem de Parâmetro

- As linguagens de programação normalmente permitem as passagens de parâmetro por valor e por referência
- A Linguagem C (como o Java) permite somente a passagem de parâmetro por valor
- Na passagem de parâmetros por valor, passamos apenas o valor e qualquer alteração no método chamado não será refletida no que chama

Passagem de Parâmetro

- Na passagem por referência, passamos uma referência fazendo com que qualquer alteração no método chamado seja refletida no que chama
- Nesse caso, o argumento do método chamado ocupa a mesma área de memória da variável correspondente no método que chama
- Por exemplo, as linguagens C++ e C# possuem a passagem de parâmetros por referência

Passagem de Parâmetro

- Um erro comum na linguagem C (no Java também) é achar que ela tem a passagem de parâmetros por referência e essa confusão acontece quando o argumento é um ponteiro

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

Memória

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {
```

```
    int a = 0, b = 0;
```

```
    funcao(&a, b);
```

```
    printf("\n(%i) (%i)", a, b);
```

```
    return 0;
```

```
}
```

Tela

Memória

a

0

33h

...

b

0

51h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

Memória

a	0	33h
	...	
b	0	51h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}
```

Tela

Memória

a	0	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){
    *a = *a + 1;
    b = b + 1;
    printf("\n(%p) (%i) (%i)", a, *a, b);
}

int main(int argc, char *argv[]) {
    int a = 0, b = 0;
    funcao(&a, b);
    printf("\n(%i) (%i)", a, b);
    return 0;
}
```

Tela

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	0	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

(33h) (1) (1)

Memória

a	1	33h
	...	
b	0	51h
	...	
a	33h	7Bh
	...	
b	1	C2h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

(33h) (1) (1)

Memória

a	1	33h
	...	
b	0	51h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

(33h) (1) (1)
(1)(0)

Memória

a	1	33h
	...	
b	0	51h

Exemplo de Passagem por Valor com Ponteiro

```
void funcao(int* a, int b){  
    *a = *a + 1;  
    b = b + 1;  
    printf("\n(%p) (%i) (%i)", a, *a, b);  
}
```

```
int main(int argc, char *argv[]) {  
    int a = 0, b = 0;  
    funcao(&a, b);  
    printf("\n(%i) (%i)", a, b);  
    return 0;  
}
```

Tela

(33h) (1) (1)
(1)(0)

Memória

a	1	33h
	...	
b	0	51h

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
    printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
    printf("\n %i %i %d", a, b, c);
}
```

Tela

Memória

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

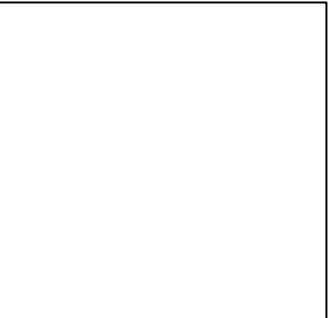
Memória

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela



Memória

a
b
c

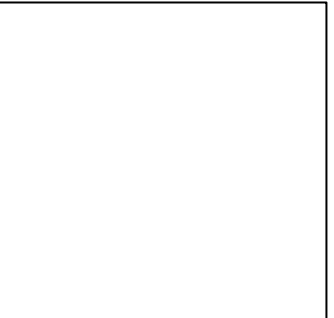
?
?
?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela



Memória

a
b
c

1
1
?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}  
  
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1 1 ?

Memória

a
b
c

1

1

?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}  
  
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1 1 ?

Memória

a
b
c

1

1

?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1 1 ?

Memória

a/x

1

b

1

c

?

y

1

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
    printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
    printf("\n %i %i %d", a, b, c);
}
```

Tela

1 1 ?

Memória

a/x

1

b

1

c

?

y

1

z

?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}  
  
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1	1	?
1	1	?

Memória

a/x	1
b	1
c	?
y	1
z	?

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}  
  
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1	1	?
1	1	?

Memória

a/x

2

b

1

c

?

y

2

z

2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){
    int z;
    printf(" %i %i ?", x, y);
    x = y = z = 2;
    printf("\n %i %i %d", x, y, z);
    return z;
}

int main(int argc, char **argv){
    int a, b, c;
    a = b = 1;
    printf(" %i %i ?", a, b);
    c = metodo(a, b);
    printf("\n %i %i %d", a, b, c);
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x	2
b	1
c	?
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x	2
b	1
c	?
y	2
z	2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}  
  
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1	1	?
1	1	?
2	2	2

Memória

a/x
b
c

y
z

2
1
2
2
2

Exemplo (1) de Passagem por Referência em C++

```
int metodo(int& x, int y){  
    int z;  
    printf(" %i %i ?", x, y);  
    x = y = z = 2;  
    printf("\n %i %i %d", x, y, z);  
    return z;  
}
```

```
int main(int argc, char **argv){  
    int a, b, c;  
    a = b = 1;  
    printf(" %i %i ?", a, b);  
    c = metodo(a, b);  
    printf("\n %i %i %d", a, b, c);  
}
```

Tela

1	1	?
1	1	?
2	2	2
2	1	2

Memória

a/x

2

b

1

c

2

y

2

z

2

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

a	10	33h
	...	
b	25	51h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

a	10	33h
	...	
b	25	51h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

a/a	10	33h
	...	
b	25	51h
	...	
b	51h	7Bh

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

a/a	5	33h
	...	
b	25	51h
	...	
b	51h	7Bh

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

--	--	--

E9h

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}  
  
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

10	20	30
----	----	----

 E9h

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

10	20	30	E9h
----	----	----	-----

Tela

Memória

a/a	5	33h
	...	
b	6	51h
	...	
b	51h	7Bh
	...	
resp	E9h	C2h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

10	20	30	E9h
----	----	----	-----

Tela

Memória

a	5	33h
	...	
b	6	51h
	...	
vet	E9h	88h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

10	20	30
----	----	----

 E9h

Tela

5 6

Memória

a 5 33h

...

b 6 51h

...

vet E9h 88h

Exemplo (2) de Passagem por Referência em C++

```
int* metodo(int& a, int* b){  
    a = 5;  
    *b = 6;  
    int* resp = new int[3];  
    resp[0] = 10; resp[1] = 20; resp[2] = 30;  
    return resp;  
}
```

```
int main(int argc, char **argv){  
    int a = 10, b = 25;  
    int* vet = metodo(a, &b);  
    printf("%d %d %d", a, b);  
    printf("%d %d %d", vet[0], vet[1], vet[2]);  
}
```

10	20	30
----	----	----

 E9h

Tela

5	6
10	20 30

Memória

a 5 33h

b 6 51h

vet E9h 88h