Vetores - Imagine ter que desenvolver um programa para armazenar as notas dos alunos, nomes, médias, nome dos pais, faltas e tudo mais
Um vetor é uma série de variáveis indexadas que podem ser acessadas por meio de uma indice inteiro

Por exemplo: vetor[5]

- Um vetor só guarda variáveis do mesmo tipo
    - vetor de inteiros
    - vetor de caracteres
    - vetor de ponto flutuante

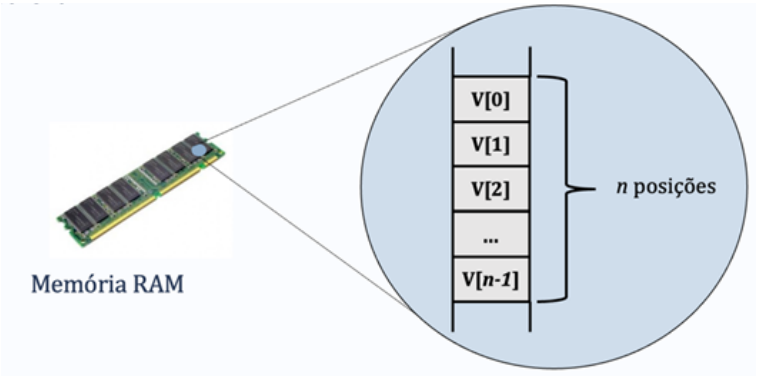
Declaração de vetores
para usar um vetor, primeiro declará-lo, como era feito para qualquer variável comum:

- tipo_de_dado nome_vetor[tamanho];

Exemplos:

- int lista[5];
- float salarios[3];
- char nome[30];

int v[10];
int v[3] = {5, 10, 15};
int v[] = {5, 10, 15}; //tamanho 3
int v[10] = [5, 10, 15}; //restante preencher com zero
int v[]; // Declaração incorreta. Provoca erro de compilação
As variáveis compostas derivam dos tipos primitivos e podem ser categorizadas como homogêneas ou heterogêneas. Além disso, podem ser unidimensionais ou multidimensionais. Quando armazenam valores do mesmo tipo primitivo, são classificadas como homogêneas, enquanto, se armazenam valores de diferentes tipos, são consideradas heterogêneas.



Diversas funções são utilizadas para a leitura e impressão de strings em C. Estudaremos duas delas. A primeira é a função já conhecida, scanf(), que agora é usada com o código de controle %s para indicar que uma string será armazenada. Portanto, para armazenar um nome digitado pelo usuário, utilizamos o seguinte formato:

```
char nome[16];

printf("\n Digite um nome:");

scanf("%s", nome);

printf("\n Nome digitado: %s", nome);
```

Para contornar essa limitação, uma opção é utilizar a função **fgets()**, que também faz parte da biblioteca padrão **<stdio.h>**. Essa função é utilizada com a seguinte sintaxe:

```
fgets(destino, tamanho, fluxo);
```

O parâmetro "destino" especifica o nome da string que será utilizada para o armazenamento. O parâmetro "tamanho" deve ser o mesmo declarado para a string. O parâmetro "fluxo" indica a origem da string; no caso em questão, sempre virá do teclado, e, portanto, utilizamos "stdin" (entrada padrão). Segue um exemplo prático:

```
char frase[101];

printf("\n Digite uma frase:");

fflush(stdin);

fgets(frase, 101, stdin);

printf("\n Frase digitada: %s", frase);
```

Matriz - Cada linha de uma matriz é um vetor-linho de n números, e a matriz é um vetor de m vetores-linha

- muitas vezes chamadas de vetores multidimensionais

sintaxe similar à declaração de vetores:
tipo nome_matriz[linhas][colunas];
Exemplos:
int valores[3][2] = {{2,3}, {5,7}, {9,11}}; //correto
int valores[][2] = {{2,3}, {5,7}, {9,11}}; //correto
int valores[][] = {{2,3}, {5.7}, {9,11}}; //inválido

Uma matriz possui dois índices,
precisamos de dois laços para percorrer todos os seus elementos

Imagine ter que declarar quatro cadastros para quatro pessoas diferentes:

```
chr nome[50], nome2[50], nome3[50], nome4[50];
int idade1, idade2, idade3, idade4;
char rua1[50], rua2[50], rua3[50], rua4[50];
int numero1, numero2, numero3, numero5;
```

Utilizando uma estrutura, o mesmo poder ser feito da seguinte maneira:

struct cadastro c1, c2, c3, c4;

```
struct Cadastro {
char nome[30];
int idade;
char rua[50];
int numero;
};
```

Declaração no main:
struct Cadastro c1;

OU

```
typedef struct {
char nome[30];
int idade;
char rua[50];
int numero;
} Cadastro;
```

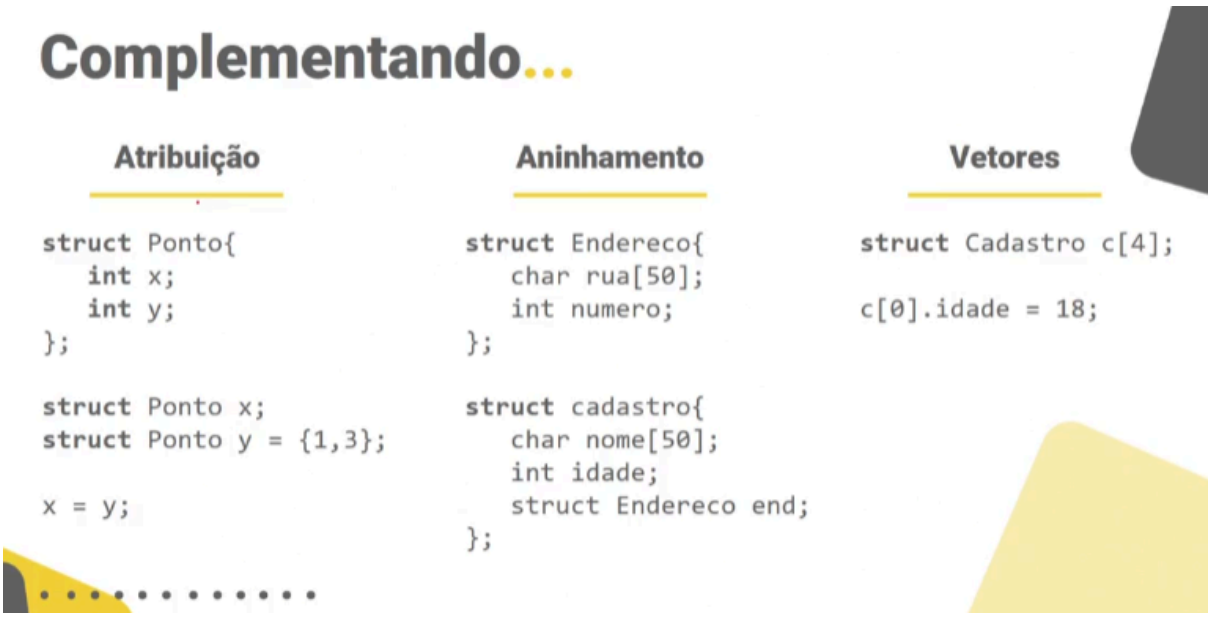
Declaração na main:
Cadastro c1;

Cada campo (variável) da estrutura pode ser acessado usando o operador "."(ponto)

```
struct Cadastro c1;

strcpy(c1.nome, "João");
c1.idade = 32;

fgets(c1.rua, 30, stdim);
scanf("%d", &c1.numero);
```



Caso se faça necessário ler uma entrada de dados do usuário, basta usar o scanf normalmente como fazemos com variáveis comuns, e de forma independente para cada variável; da estrutura, respeitando o tipo de cada uma para o código de controle. Veja o trecho de código a seguir que lê os valores para os campos “numMat” e “nome” da variável “aluno1”.

```
scanf(“%d”, &aluno1.numMat);

fgets(aluno1.numMat, 30, stdin);
```

Utilizados para fazer manipulação direto de endereços de memória
(*): usando para declarar o ponteiro
(&): utilizando para acessar o endereço de memória
<tipo> *<nome_no_ponteiro>;
int *idade;

Ponteiro só aceitam **endereços de memória**

Portanto, com ele conseguimos atribuir o endereço de memória de uma variável a um ponteiro

dentro de printf() use %x para exibir o endereço de memória, pois o mesmo se trata de um valor hexadecimal

Ponteiro para vetores - O nome de um vetor corresponde ao endereço do seu primeiro elemento, isto é, se v for um vetor v == &v[0]

int v[3] = {10, 20, 30, 40, 50};

```
int *ptr;
ptr = v;
ou
ptr = &v[0];
```

Não se deve fazer cargas iniciais de objetos apontados por um ponteiro que ainda não tenha sido iniciado

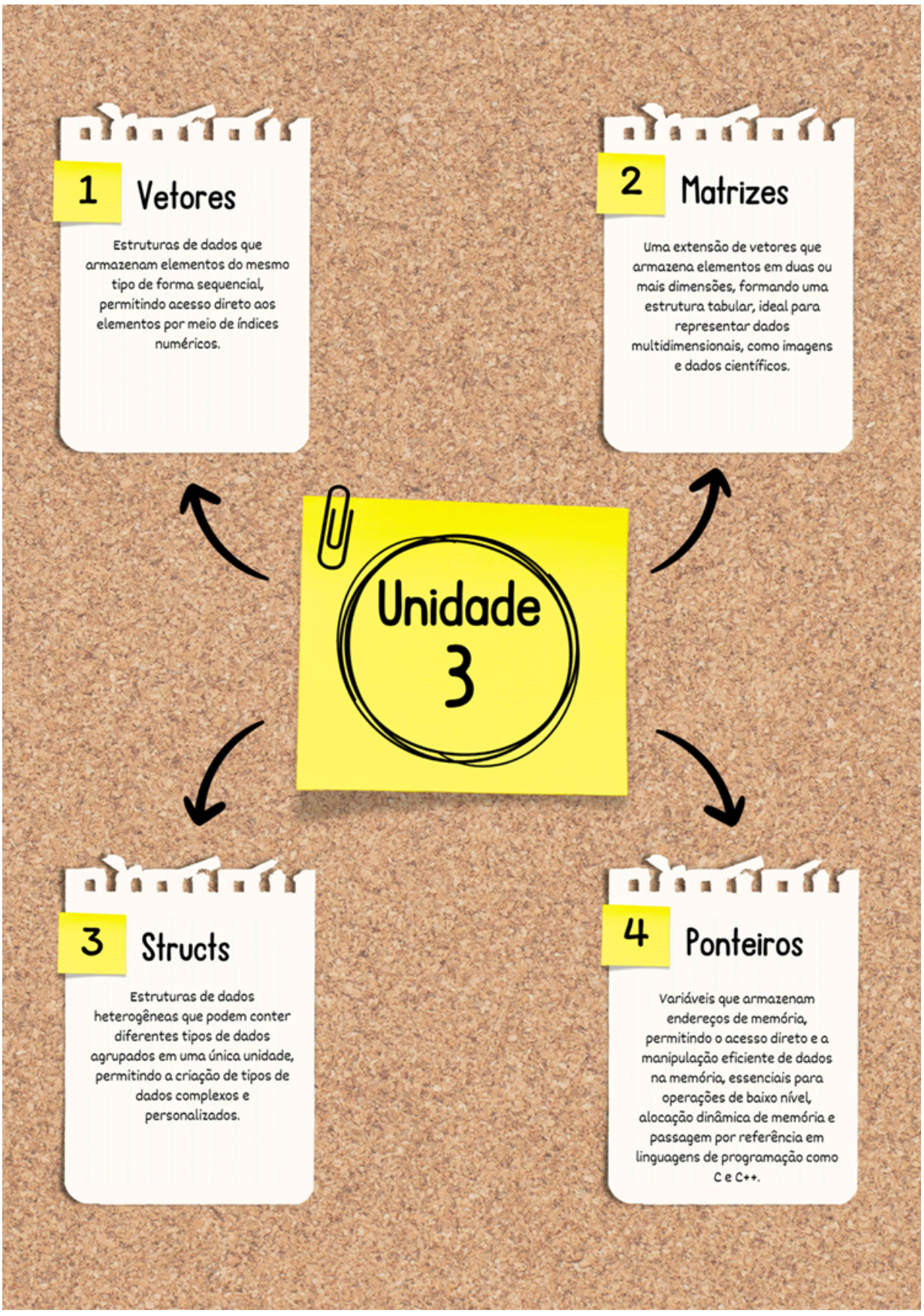
```
int *p; /*p fica com lixo no seu interior */

*p = 100; /* Coloca o valor 100 no local para onde p aponta que será qualquer lugar na memória */
```

Por segurança, inicie sempre os seus ponteiros. Se não souber para onde apontá-los, inicie com NULL

```
int *p = NULL
```

Atenção: Neste exemplo, é o ponteiro p que é iniciado, e não *p, embora a atribuição possa por vezes sugerir o contrário
O especificador de formato “%p” é usado para imprimir o endereço de memória armazenado em um ponteiro, em hexadecimal (também poderia ser utilizado “%x”).


1 Vetores
Estruturas de dados que armazenam elementos do mesmo tipo de forma sequencial, permitindo acesso direto aos elementos por meio de índices numéricos.
2 Matrizes
Uma extensão de vetores que armazena elementos em duas ou mais dimensões, formando uma estrutura tabular, ideal para representar dados multidimensionais, como imagens e dados científicos.
Unidade 3
3 Structs
Estruturas de dados heterogêneas que podem conter diferentes tipos de dados agrupados em uma única unidade, permitindo a criação de tipos de dados complexos e personalizados.
4 Ponteiros
Variáveis que armazenam endereços de memória, permitindo o acesso direto e a manipulação eficiente de dados na memória, essenciais para operações de baixo nível, alocação dinâmica de memória e passagem por referência em linguagens de programação como C e C++.