

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PIAUÍ
Campus Teresina - Central

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E
TECNOLOGIA DO PIAUÍ

CAMPUS TERESINA-CENTRAL

DIRETORIA DE ENSINO

Estrutura de Dados

Introdução

(CAP 1 do livro Estrutura de Dados Em C)

Professora: Elanne na O. dos Santos

elannecristina.santos@gmail.com
elannecristina.santos@ifpi.edu.br

Abstração De Dados

- Uma **estrutura de dados** é um tipo de dados abstrato que representa uma coleção de itens inter-relacionados.
- O **tipo de relacionamento** entre os itens é que define a classe de estrutura de dados que eles compõem.
- Uma coleção de itens:
 - onde não há ordem nem repetição é um **conjunto**.
 - organizados linearmente é uma **lista**.
 - Organizados hierarquicamente é uma **árvore**. Cada item com um único predecessor e vários sucessores, exceto o raiz e as folhas.
 - Organizados em rede é um **grafo**. Cada item pode ter vários predecessores e sucessores.

Principais classes de estrutura de dados

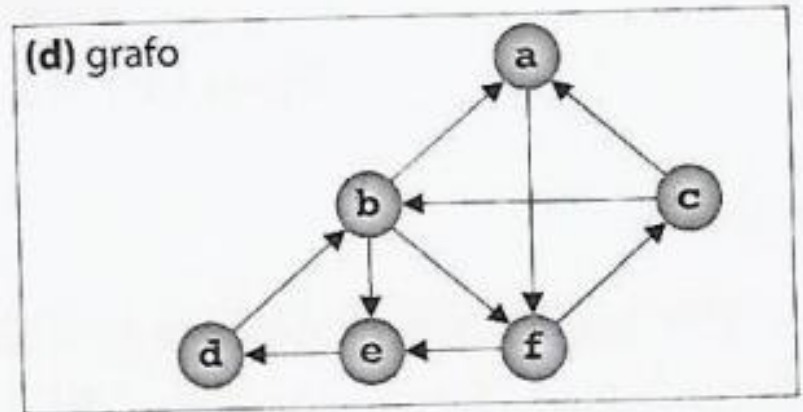
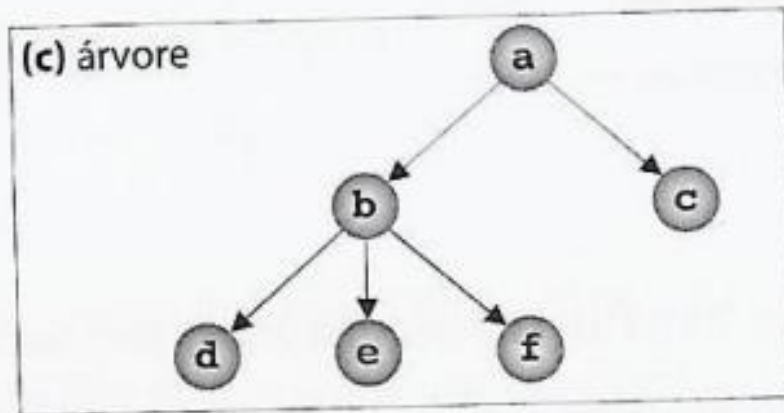
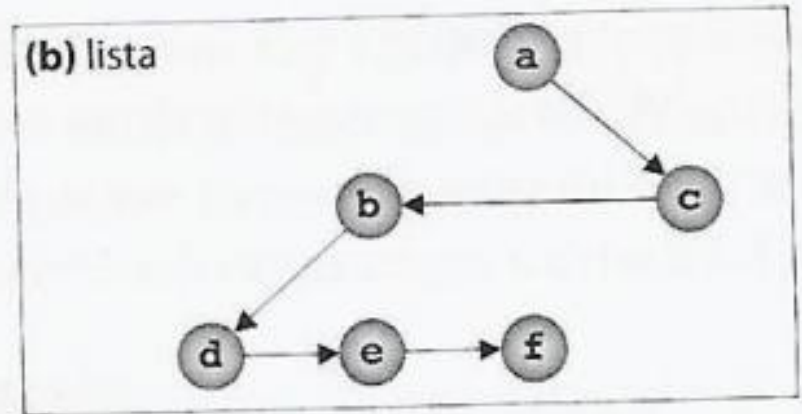
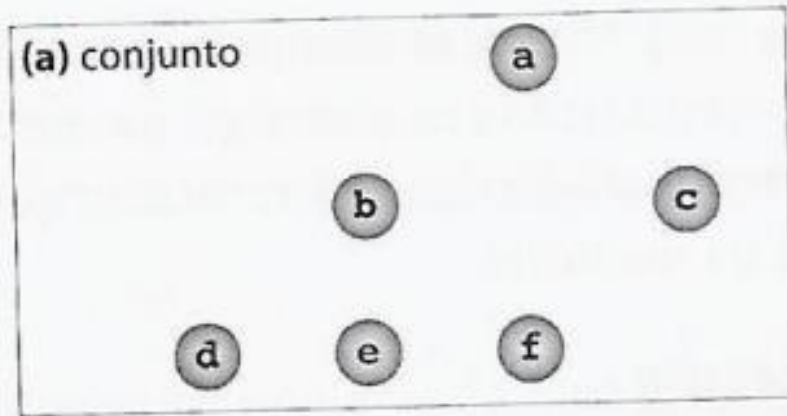


Figura 1.1 | Principais classes de estruturas de dados.

OBJETIVOS PRINCIPAIS DO ESTUDO DE ESTRUTURA DE DADOS

- Implementar estrutura de dados, usando mecanismos de agregação de dados e alocação de memória existentes na linguagem de programação.
- Mostrar que o uso de uma estrutura de dados adequada pode simplificar a criação de um programa que resolve um tipo de problema específico.

MECANISMOS DE AGREGAÇÃO DE DADOS

- Um das principais preocupações da ED é decidir como agregar seus itens.
- Para isso é necessário conhecer as **propriedades dessa estrutura e os mecanismos de agregação de dados usados existentes na linguagem utilizada.**

MECANISMOS DE AGREGAÇÃO DE DADOS EM LINGUAGEM C

❑ VETOR

- Agrega vários itens de um mesmo tipo, formando uma coleção **homogênea**.
- Usa posições adjacentes de memória e é acessado por meio de índices.

```
#include <stdio.h>
main()  {
.....
    int v[3]={9,6,7};
.....
}
```

❑ REGISTRO

- Agrega vários itens de tipos distintos.
- Os itens do registro ocupam posições adjacentes e são identificados por campos.
- Antes de criar o registro é preciso definir seu tipo. Ex:

```
#include <stdio.h>

typedef struct registro {
    int a; char b; float c;
} Reg;

main() {
    .....
    Reg r = {18, 'a', 2.5};
    .....}
```

```
struct aluno {  
int mat;  
float nota;  
char nome[30];  
};  
typedef struct aluno Aluno;
```

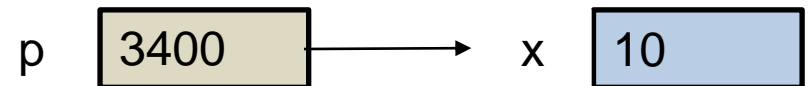
```
typedef struct aluno{  
    int matricula;  
    float nota;  
    char nome[30];  
}Aluno;
```


❑ PONTEIRO

- Variável que guarda o endereço de memória de outra variável. Ex:

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

```
main() {  
    int *p;  
    int x = 10;  
    p = &x;  
    ..... }  
}
```



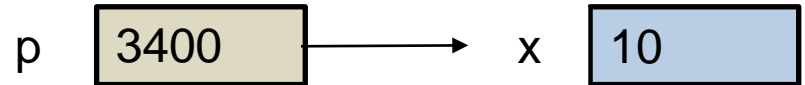
❑ PONTEIRO

- Para acessar a variável apontada por um ponteiro usa-se *<nome_do_ponteiro>. Ex:

```
#include <stdio.h>

main() {
    int *p;
    int x = 10;

    p= &x;
    printf("%d", *p) ;
    .....}
```



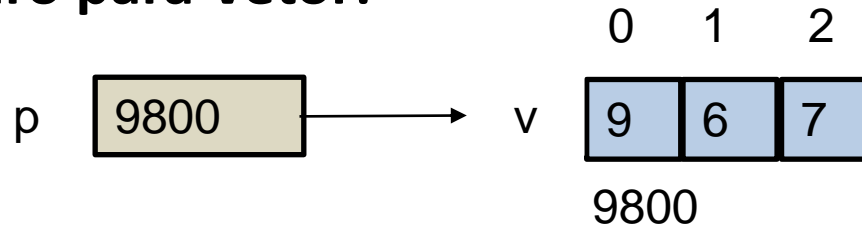
❑ PONTEIRO

```
main() {  
    int x=10;  
    p = &x;  
    printf("%d\n", *p) ;  
    printf("%p\n", p) ;  
    cout<<p<<endl ;  
}
```

❑ PONTEIRO

- Pode ser criado de qualquer tipo.

❑ Ponteiro para Vetor:



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

```
main() {
```

```
    int v[3] = {9,6,7};
```

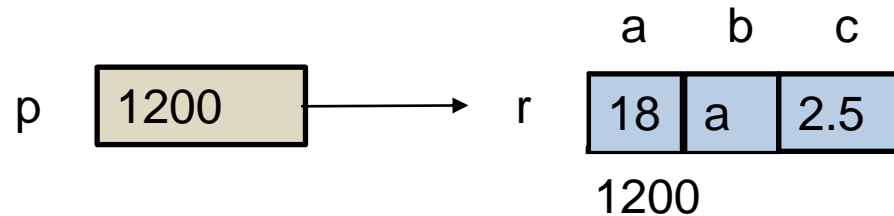
```
    int *p = v; //ponteiro para um vetor
```

```
    .....}
```

- Para acessar o *i*-ésimo item do vetor apontado por `p`, basta escrever `*(p+i)` ou `p[i]`.

❑ PONTEIRO

❑ Ponteiro para Registro:



```
#include <iostream>
using namespace std;
type struct {int a;char
b;float c;}Reg;
int main()
{
    Reg r = {18,'a',2.5};
    Reg *p= &r;
}
```

A biblioteca *iostream* e o tipo *string*

No estilo da linguagem C quando queremos representar um conjunto de caracteres colocamos todos eles em uma matriz sequenciada na memória:

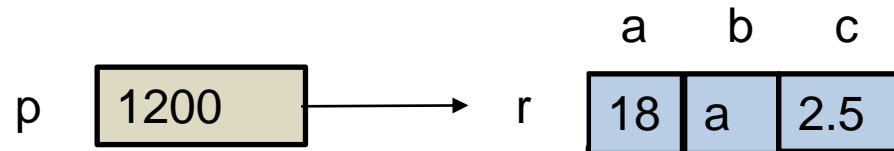
Endereço relativo	0x0	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09
Dado	U	m	a		f	r	a	s	e	\0

Por exemplo, para declarar um espaço na memória que contenha 20 caracteres fazemos:

```
char dados[20];
```

❑ PONTEIRO

❑ Ponteiro para Registro:



- Para acessar o campo "c" do registro apontado por `p`, use: `(*p).c` ou `p->c`. O operador `->` só pode ser usado com ponteiros para registro.
- Para indicar que um ponteiro não está apontando para uma variável use `NULL`, definido em `stdio.h`. Ex: `p=NULL`;
- NO caso do ponteiro ser NULO qualquer tentativa de acesso a `p` causará um erro que abortará a execução do programa.

FORMAS DE ALOCAÇÃO DE MEMÓRIA

- **ALOCAÇÃO ESTÁTICA:** alocada com base no tipo de variável e local que foi declarada no programa. O compilador determina o tamanho de memória, criação e destruição automaticamente.
- **ALOCAÇÃO DINÂMICA:** Ocorre quando a variável é necessária SOMENTE durante a execução do programa. O programador define quantidade de memória, criação e destruição da variável. Use para isso a função `malloc()` definida em `stdlib.h`. Exemplo:

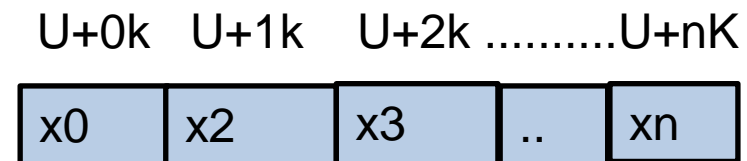
```
int *p=(int*)malloc(sizeof(int)) ;
```


ALOCAÇÃO DINÂMICA

- ***Malloc*** apenas aloca memória depois disso, é responsabilidade do usuário fazer o ***typecast*** para um tipo apropriado para que possa ser usado corretamente no programa.
- Ponteiro ***void*** é um ponteiro que pode apontar qualquer tipo de dado. ***Malloc*** retorna ponteiro ***void*** porque ele não sabe qual tipo de dado será armazenado dentro daquela memória.
- No exemplo apresentado, temos que fazer um ***typecast*** para um ponteiro do tipo inteiro porque queremos armazenar um inteiro naquela memória.

Alocação sequencial e encadeada

- Com relação ao modo como os itens de uma coleção são distribuídos na memória, há duas formas de alocação:
- Na **alocação sequencial**, os itens ocupam **posições adjacentes de memória**. A partir do endereço do primeiro item, é possível obter diretamente o endereço de qualquer outro item da coleção.



- No exemplo, **u** é o endereço do primeiro elemento e cada item ocupa **k** bytes.

- A alocação sequencial é feita com um vetor. O endereço do 1o item do vetor é indicado com o próprio nome do vetor.
- Supondo **v** o nome de um vetor, **v** é o endereço do 1o item e o endereço do **i-ésimo** item é **v+i**.
- A alocação sequencial pode ser **estática** ou **dinâmica**.

Ex:

```
int v[9];
```

```
int n;
```

```
int *w=(int*)malloc(n*sizeof(int));
```

- A alocação sequencial é feita com um vetor. O endereço do 1o item do vetor é indicado com o próprio nome do vetor.
- Supondo **v** o nome de um vetor, **v** é o endereço do 1o item e o endereço do **i-ésimo** item é **v+i**.
- A alocação sequencial pode ser **estática** ou **dinâmica**.

Ex:

```
int v[9];
```

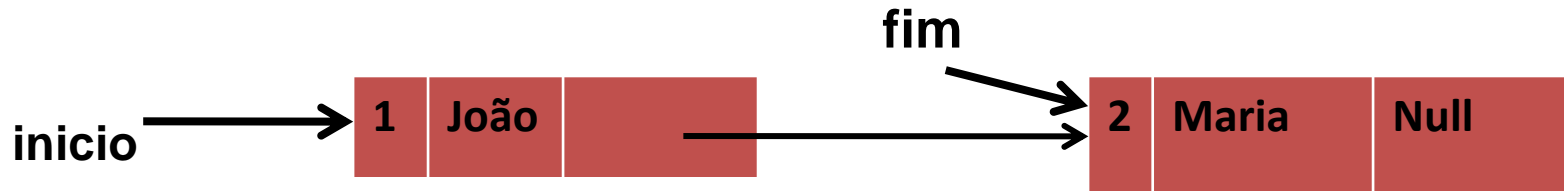
```
int n;
```

```
int *w=(int*)malloc(n*sizeof(int)) ;
```

- A **vantagem da alocação sequencial** é a rapidez de acesso aos dados. A **desvantagem** aparece quando é necessário inserir ou remover itens (tempo gasto em deslocamento de itens).
- O esquema da **alocação encadeada** é baseada no **conceito de nós**. Um **nó** é um registro que guarda **um item** e um **ponteiro para outro nó**. **Exemplo:**

```
typedef struct elemento{  
    int mat;  
    char nome[20];  
    elemento *prox;  
}Elemento;
```

- A estrutura formada pelo encadeamento de nós é chamada **lista encadeada**. O endereço do primeiro nó é guardado por um ponteiro “p” qualquer e a partir de p é possível obter indiretamente os outros nós da lista. Exemplo:



- A **vantagem** da **alocação encadeada** é a rapidez de inserção e remoção de itens. A **desvantagem** é quando um item arbitrário da coleção deve ser acessado.