Ponteiros & Alocação Dinâmica

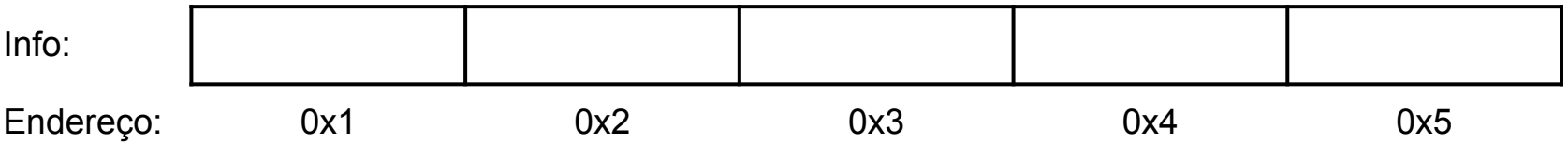
Uhuulll!!!

Ponteiros

- São variáveis.
- Fazem referência a objetos de memória (normalmente implementados como endereço de máquina).
- Declaração: *TIPO * var_pnt;*
  - TIPO: Tipo da variável para a qual o ponteiro faz referência. Pode ser inclusive um tipo ponteiro.
- Inicialização por referência: *var_pnt = & var_info;*
  - &: Diz o endereço da variável *var_info*;
- O valor que *var_pnt* armazena é diferente do valor que *var_info* armazena.

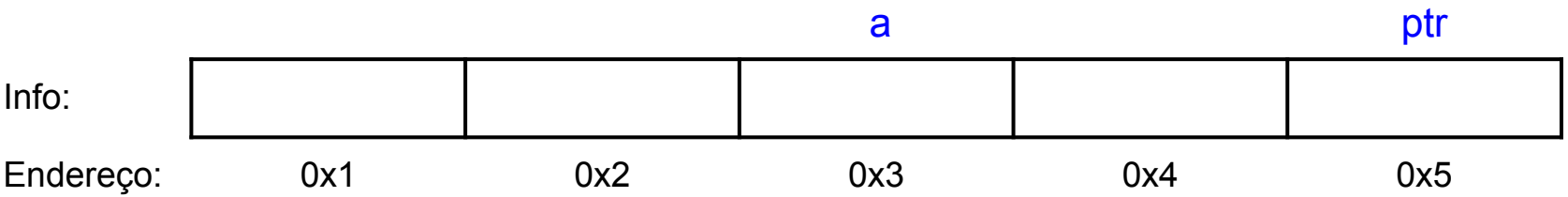
Ponteiros

Memória:



Ponteiros

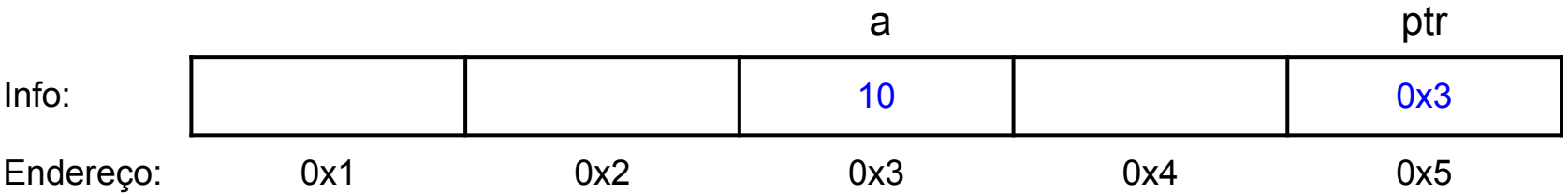
Memória:




```
int a;
int *ptr;
```

Ponteiros

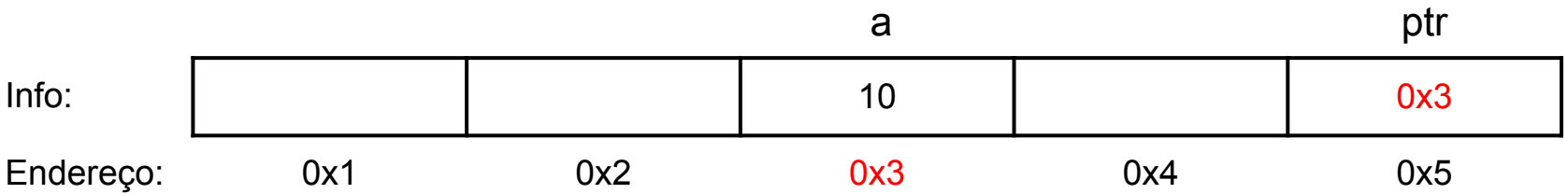
Memória:




```
int a;
int *ptr;

a = 10;
ptr = &a;
```

Ponteiros

Memória:

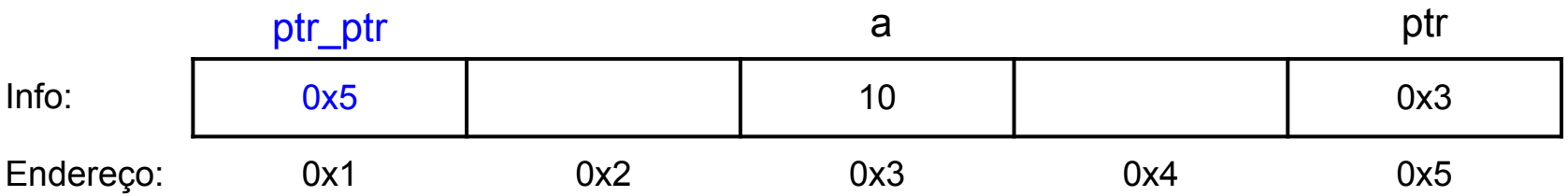



```
int a;
int *ptr;

a = 10;
ptr = &a;
```

ptr ⟶ a

Ponteiros

Memória:

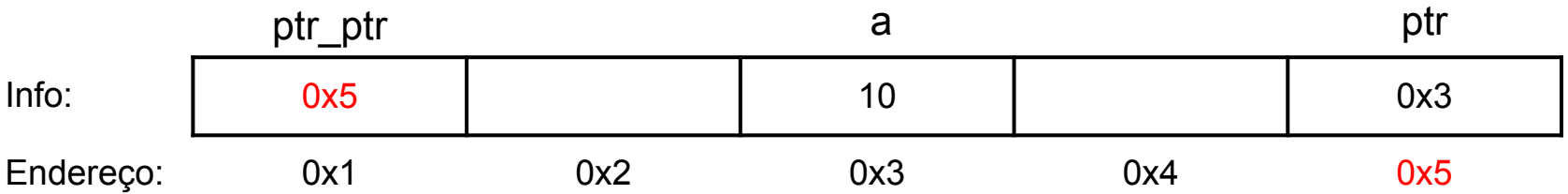



```
int a;
int *ptr;
int **ptr_ptr;

a = 10;
ptr = &a;
ptr_ptr = &ptr;
```

ptr ⟶ a

Ponteiros

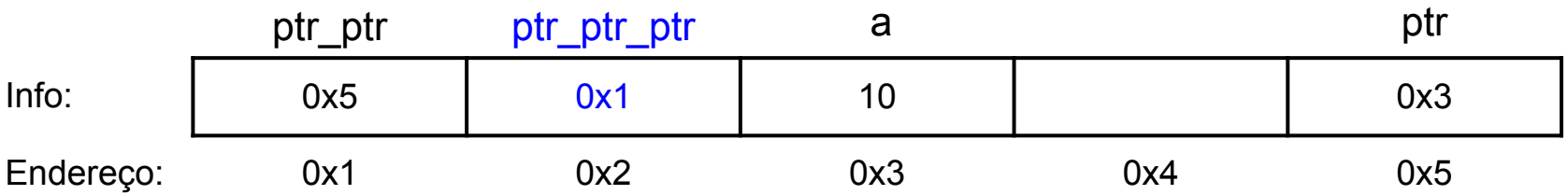
Memória:



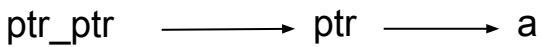

```
int a;
int *ptr;
int **ptr_ptr;

a = 10;
ptr = &a;
ptr_ptr = &ptr;
```

Ponteiros

Memória:

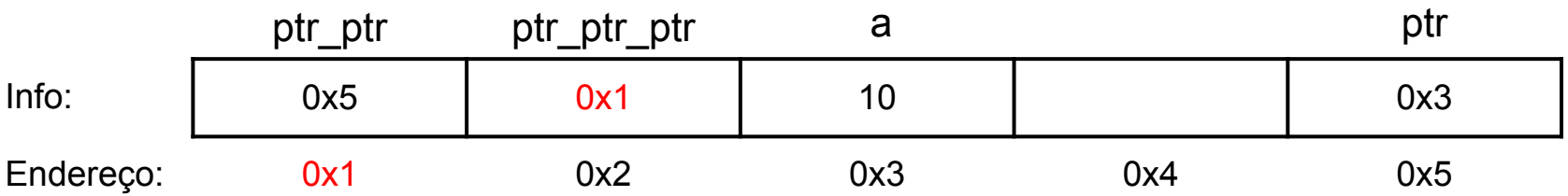



```
int a;
int *ptr;
int **ptr_ptr;
int ***ptr_ptr_ptr;

a = 10;
ptr = &a;
ptr_ptr = &ptr;
ptr_ptr_ptr = &ptr_ptr;
```



Ponteiros

Memória:

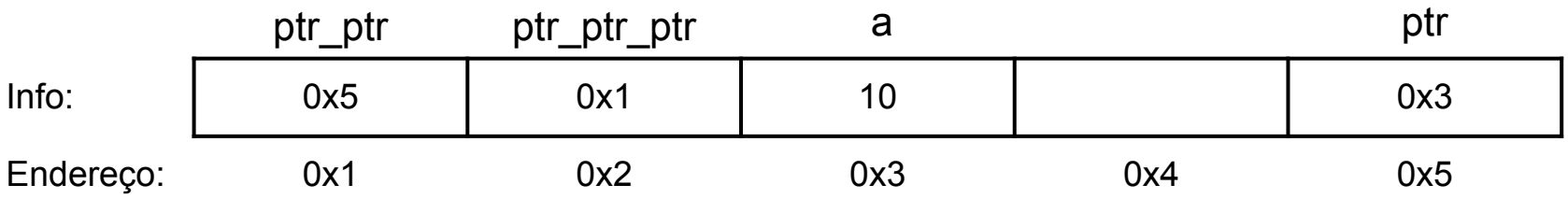



```
int a;
int *ptr;
int **ptr_ptr;
int ***ptr_ptr_ptr;

a = 10;
ptr = &a;
ptr_ptr = &ptr;
ptr_ptr_ptr = &ptr_ptr;
```

Ponteiros

- Valor armazenado: *var_pnt;*
- Valor referenciado: **var_pnt;*

Ponteiros



ptr_ptr_ptr → ptr_ptr → ptr → a

Variável	Valor
a	10

Variável	Valor
ptr	0x3
*ptr	10

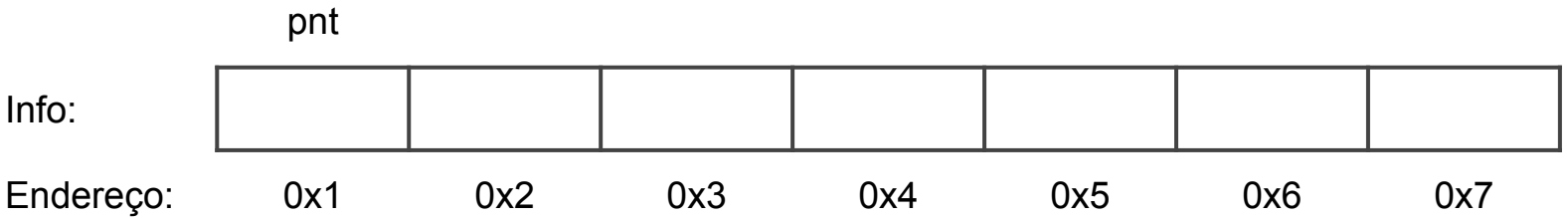
Variável	Valor
ptr_ptr	0x5
*ptr_ptr	0x3
**ptr_ptr	10

Variável	Valor
ptr_ptr_ptr	0x1
*ptr_ptr_ptr	0x5
**ptr_ptr_ptr	0x3
***ptr_ptr_ptr	10

Alocação Dinâmica

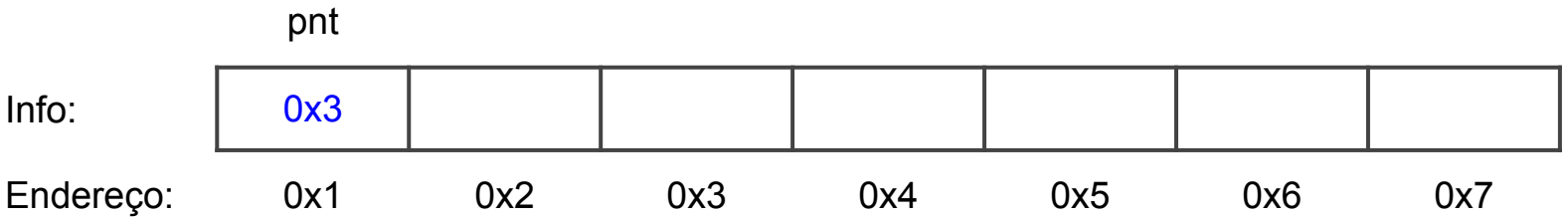
- Alocar/Criar uma variável em tempo de execução.
- Não conhecemos a variável, mas conhecemos um ponteiro para ela.
- Gerenciar melhor o espaço gasto na memória.
- O *malloc.*
- O *sizeof.*
- Alocação: *pnt = (TIPO *) malloc(N * sizeof(TIPO));*
    - N: quantidade de itens do tipo TIPO que o ponteiro faz referência.

Alocação Dinâmica



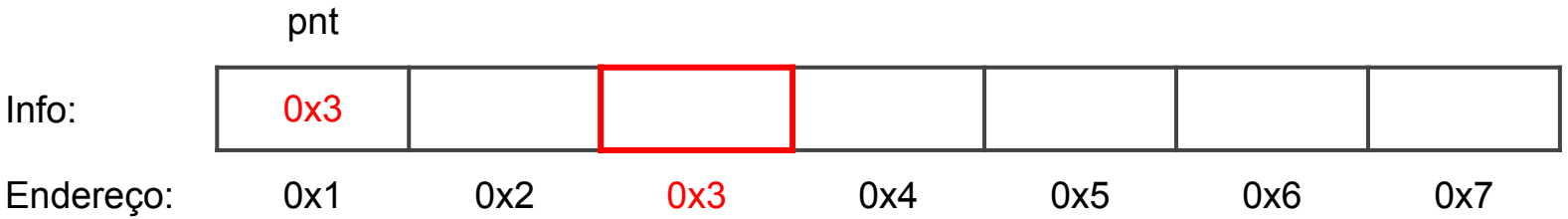

```
int *pnt;
```

Alocação Dinâmica




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

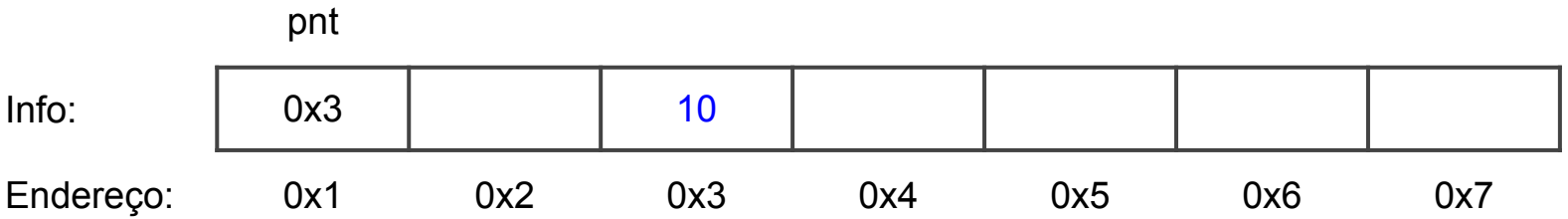
Alocação Dinâmica




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

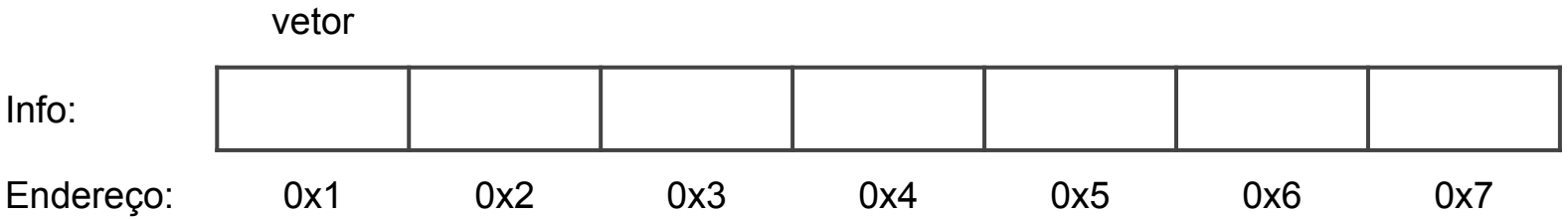
pnt ⟶ ?

Alocação Dinâmica



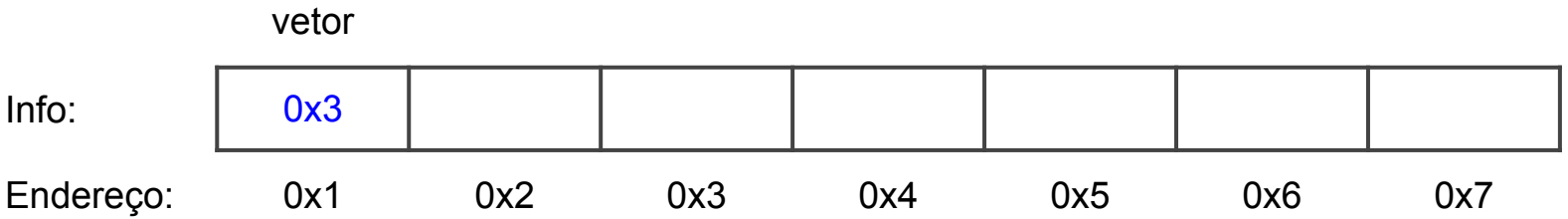

```
int *pnt;
pnt = (int*) malloc(sizeof(int));

*pnt = 10;
```

Alocação Dinâmica



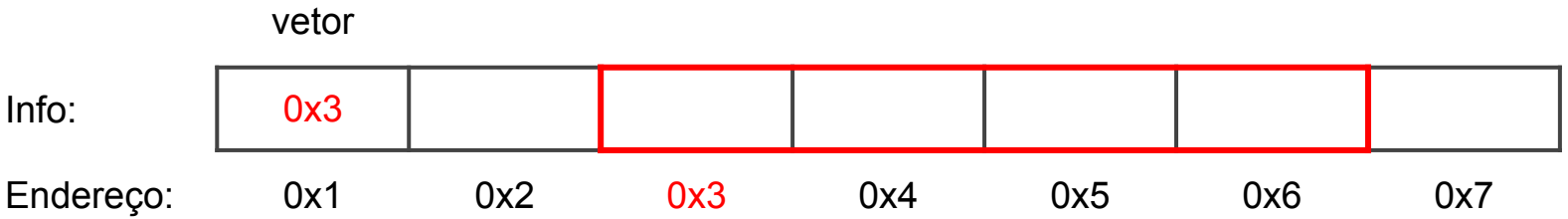

```
int *vetor;
```

Alocação Dinâmica



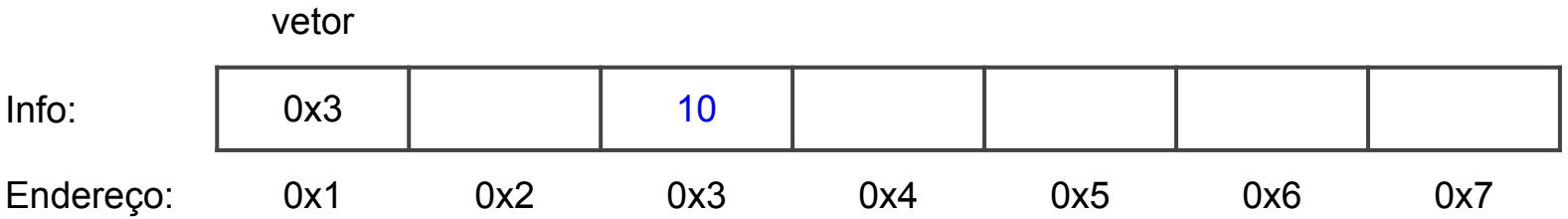

```
int *vetor;
int N = 4;
vetor = (int*) malloc(N * sizeof(int));
```

Alocação Dinâmica



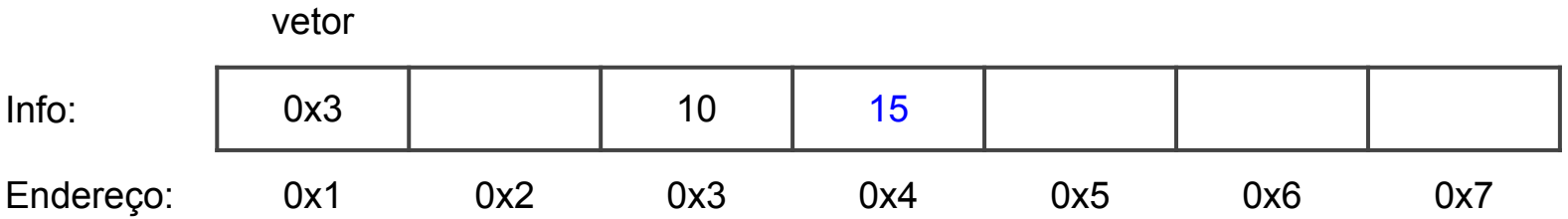

```
int *vetor;
int N = 4;
vetor = (int*) malloc(N * sizeof(int));
```

Alocação Dinâmica



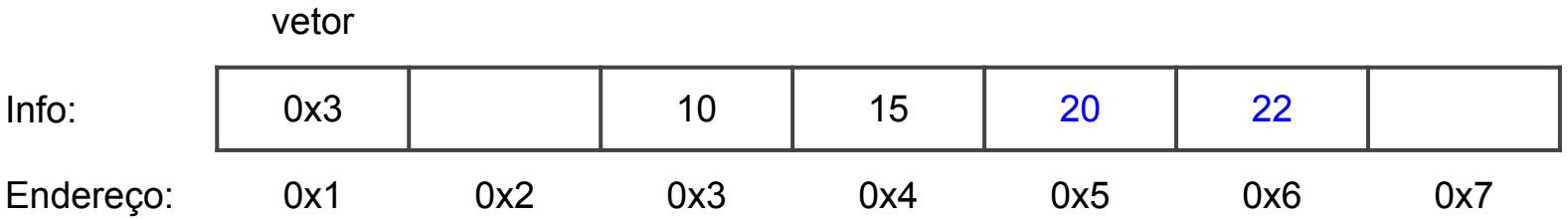

```
int *vetor;
int N = 4;
vetor = (int*) malloc(N * sizeof(int));

vetor[0] = 10;
```

Alocação Dinâmica




```
int *vetor;
int N = 4;
vetor = (int*) malloc(N * sizeof(int));

vetor[0] = 10;
vetor[1] = 15;
```

Alocação Dinâmica




```
int *vetor;
int N = 4;
vetor = (int*) malloc(N * sizeof(int));

vetor[0] = 10;
vetor[1] = 15;
vetor[2] = 20;
vetor[3] = 22;
```

TADs & Lista, Fila, Pilha

Uhuulll!!!

Estruturas (structs)

- O objetivo de uma estrutura é **agrupar** informações heterogêneas.
- Utiliza-se as estruturas para definir coleções de dados para que se possa manipular uma coleção inteira como uma unidade, mas ainda seja possível se referir individualmente aos componentes pelo nome.

Estruturas (structs) - Exemplo

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

typedef struct Livro {
    char titulo[50];
    char autor[50];
    int numeroPaginas;
    int id;
} Livro;

int main() {
    Livro livro1;

    strcpy(livro1.titulo, "Introduction to Algorithms");
    strcpy(livro1.autor, "Thomas H. Cormen");
    livro1.numeroPaginas = 1203;
    livro1.id = 1;

    printf("%d", livro1.numeroPaginas);

    return 0;
}
```

Declaração da estrutura Livro

Declaração de uma variável do tipo Livro

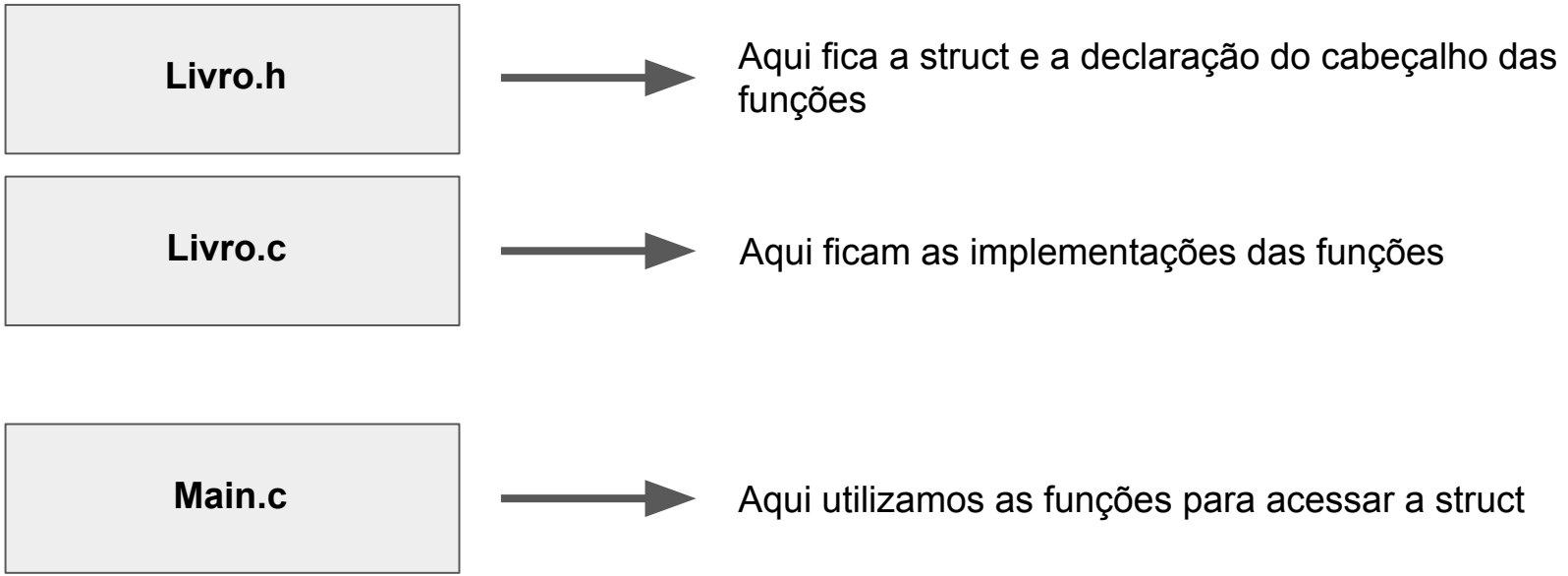
Definição dos valores da variável criada

Utilização da variável

Tipo Abstrato de Dados (TADs)

- O TAD contém uma estrutura
- Ele é responsável por encapsular essa estrutura.
- Esse encapsulamento, se bem feito, permite ao programador alterar os detalhes internos do funcionamento das funções sem prejudicar a utilização da estrutura pelo usuário.
- Dessa forma, o usuário “enxerga” apenas a interface, não a implementação.

Tipo Abstrato de Dados (TADs)



Lista

- O exemplo mais prático e simples de uma lista é o exemplo da lista de compras.
- Veja o exemplo ao lado:
- Uma lista pode ser implementada com array (vetor) ou como lista encadeada.

Pilha

- O exemplo mais prático e simples de uma pilha é o exemplo da pilha de pratos.
- Veja o exemplo ao lado:
- É o processo que chamamos de LIFO (Last in - First Out)

Fila

- O exemplo mais prático e simples de uma fila é o exemplo da fila nossa de cada dia para comer no RU - que felizmente esquecemos dela por enquanto.
- Veja o exemplo abaixo:
- É o processo que chamamos de FIFO (First in - First Out)

Array

Vantagem

Economia de memória (os apontadores são implícitos nesta estrutura)

Desvantagens

Custo para inserir ou retirar itens, que pode causar um deslocamento de todos os itens, no pior caso

Em aplicações em que não existe previsão sobre o crescimento da lista, a utilização de arranjos em linguagens como C pode ser problemática porque o tamanho máximo da lista tem de ser definido em tempo de compilação

Lista Encadeada

Vantagens

Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).

Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido a priori).

Desvantagens

Utilização de memória extra para armazenar os apontadores.

Percorrer a lista, procurando pelo i-ésimo elemento.

Métodos de Ordenação & Complexidade de Algoritmos

Análise Assintótica

- Análise do crescimento de funções
- Notação Big-O
  - Arredondamento para cima (definição) $0<=f(n)<=cg(n)$
  - Funções comumente utilizadas
- Exemplos:
  - $O(\log n)$
  - $O(n)$
  - $O(n^2)$

Métodos de Ordenação

- Selection Sort - $O(n^2)$
- Bubble Sort - $O(n^2)$
- Insertion Sort - $O(n^2)$
- Shell Sort - Knuth (?)
- Quick Sort - $O(n^2)$
- Heap Sort - $O(n \lg n)$
- Merge Sort - $O(n \lg n)$

Análise de Complexidade

- Selection Sort - $O(n^2)$
  - $n(n+1)/2$
- Relações de recorrência
  - Juros compostos
- Recursão
  - Caso Base
  - Passo Recursivo
- Resolução de relações de recorrência
  - Equação de recorrência