CURSO SUPERIOR DE ADS

Alocação Dinâmica de Memória e Tipos Abstrato de Dados

Prof. Fernando Marlon Soares Figueiredo

Disciplina: Programação Estruturada

SECTRAS

ADS

Aula de Hoje

- Alocação de Dinâmica de Memória
- Tipos Abstratos de Dados
- Listas

Problema

- Até agora estamos definindo os tamanhos das variáveis durante sua declaração.
- O maior problema dessa abordagem é que nem sempre sabemos de antemão quantos dados uma variável ou vetor vai precisar. Além disso, após declarar esse tamanho, não podemos modificá-lo em tempo de execução do programa.
- Seria interessante se, por exemplo, o usuário pudesse escolher a quantidade de elementos dos vetores durante a execução de um programa.

C possui uma forma de alocar espaço na memória dinamicamente

- É possível utilizar ponteiros e sua relação com vetores para manejar melhor a memória quando for necessário no programa.
- Duas funções são principais para alocação e desalocação de espaço na memória:

```
v = (tipo *) malloc (N * sizeof(tipo) ); //alocação
free(v); //desalocação
```

malloc

v = (tipo *) malloc (N * sizeof(tipo));

- A função malloc aloca um bloco de memória disponível, **retornando o endereço da primeira posição**, que **deverá ser armazenado em um ponteiro** que passa a apontar par ao primeiro elemento do vetor.
- Assim, como nos vetores, deve-se assumir que os **blocos alocados contêm lixo de memória**.
- Observe que antes da função em si há uma operação de casting para um ponteiro do tipo a ser alocado. Isso é necessário porque a função **malloc opera sobre ponteiros do tipo void**, justamente para que seja genérica o suficiente e possa se adaptar aos ponteiros de qualquer tipo.

malloc
v = (tipo *) malloc (N * sizeof(tipo));

- Como parâmetros na função, é necessário passar o número de bytes a ser alocado. Como o número de bytes dos tipos pode variar dependendo do sistema, costuma-se utilizar a função sizeof para descobrir o número de bytes que o tipo ocupa.
- Este valor corresponde à quantidade de bytes para um elemento, devendo ser multiplicado por **N elementos dos quais se deseja alocar**.
- A partir do momento em que a função é chamada, v pode ser utilizado da mesma forma que um vetor, com todas as suas características.

free(v)

- A função free serve para desalocar o espaço alocado na memória e deve sempre ser utilizado assim que o vetor serviu seu propósito no código.
- **Basta passar como parâmetro o ponteiro do início do espaço**, uma vez, uma vez que o programa já sabe quantos espaços estão associados àquela posição de memória, liberando-os.

Exemplo

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

int main()
{
    int *v;
    int n , i;

    printf("Digite o numero de inteiros alocados:");
     scanf("%d", &n);

     v = (int *) malloc(n *size of(int));
    for(i=0; i <n; i++)
    {
        printf("Digite o v[%d ]:", i);
        scanf("%d", &v[i]);
    }
    for(i=0; i <n; i++)
        printf("v[%d ] = %d\n", i , v[i]);

    free(v);

    return 0;
}
```

```
Digite o número de inteiros alocados: 5
Digite o v[0]:1
Digite o v[1]:2
Digite o v[2]:3
Digite o v[3]:4
Digite o v[4]:5
v[0] = 1
v[1] = 2
v[2] = 3
v[3] = 4
v[4] = 5
```

Biblioteca stdlib

- As funções malloc e free precisam da biblioteca stdlib pra funcionar.
- Uma vez alocado, é importante não perder a referência para o ponteiro do primeiro elemento, caso contrário o espaço só é desalocado quando a aplicação é finalizada.

A alocação dinâmica pode ser aplicada a qualquer tipo de dados

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

typedef struct {
    int ano;
    float preco;
    char autor[50];
} Livro;
```

```c
int main()
{
    Livro *v;
    int n, i;

    printf("Digite o numero de livros alocados:");
    scanf("%d", &n);

    v = (Livro *) malloc(n *size of(Livro));

    // ...

    free(v);

    return 0;
}
```

Exercícios

Exercício 1

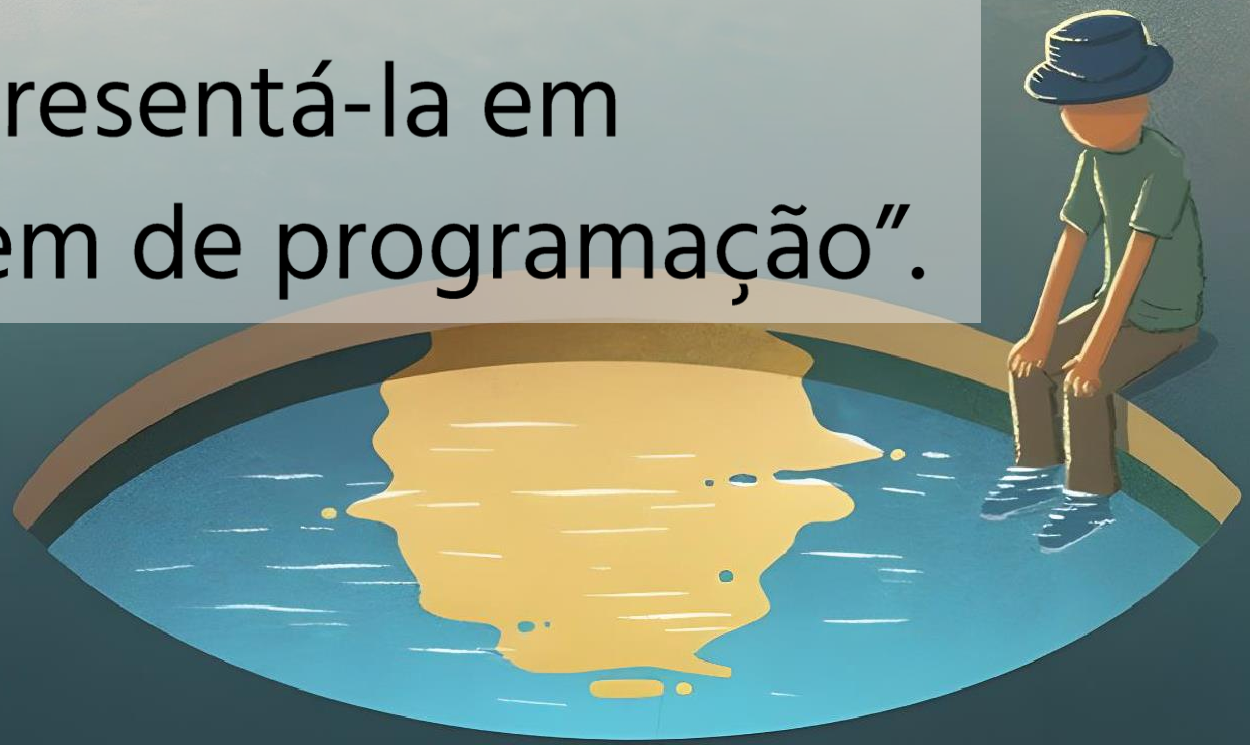
- Elabore um programa em C que aloque dinamicamente memória para armazenar uma string de até 50 caracteres.
- Solicite ao usuário que digite uma frase e armazene-a na string alocada dinamicamente.
- Após exibir a frase, libere a memória alocada.

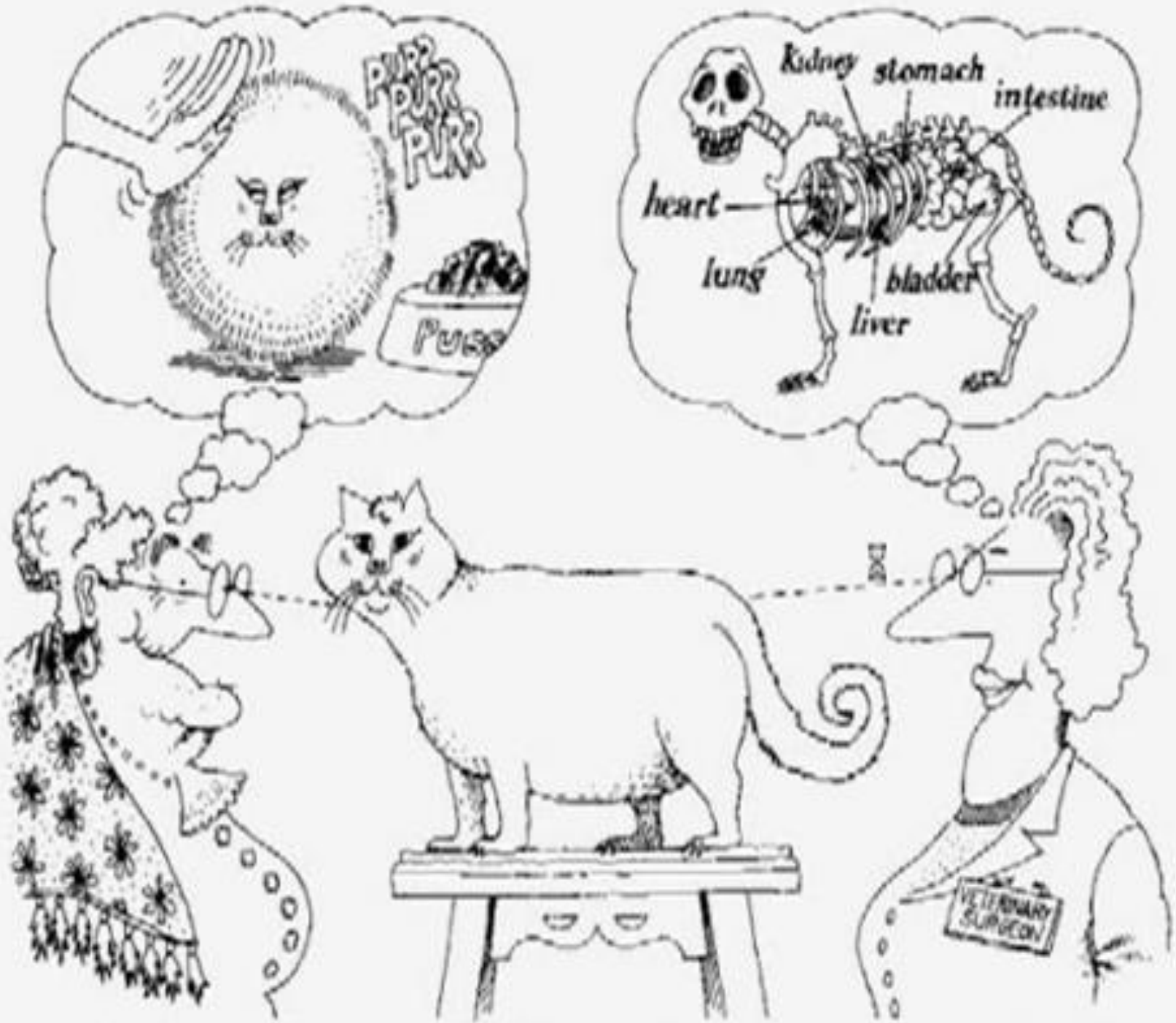
Exercício 1 - Resposta

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

int main() {
    char *frase = (char *)malloc(50 * sizeof(char));

    if (frase == NULL) {
        printf("Falha na alocação de memória!");
        return 1;
    }
    printf("Digite uma frase: ");
    fgets(frase, 50, stdin);

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

    free(frase);

    return 0;
}
```

Tipos Abstratos de Dados

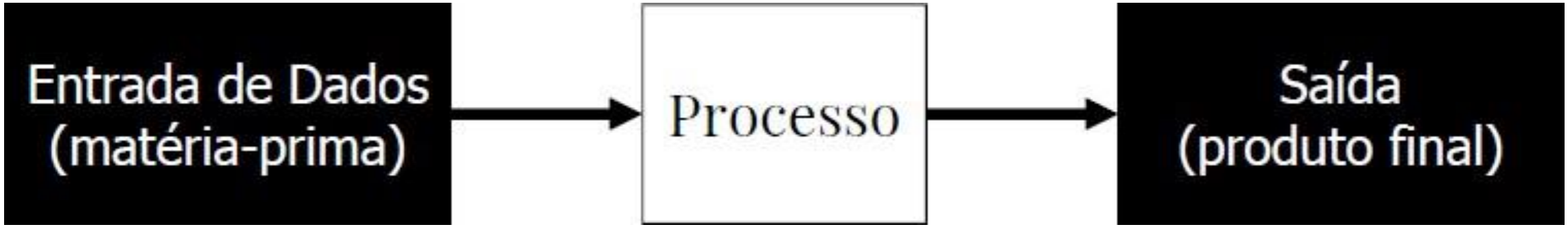
“No mundo da programação,
para resolver um problema
é necessário construir uma
abstração da realidade
e depois encontrar uma forma
de representá-la em
uma linguagem de programação”.




PURR PURR PURR
Kidney
stomach
intestine
heart
lung
bladder
liver
VETERINARY SURGEON

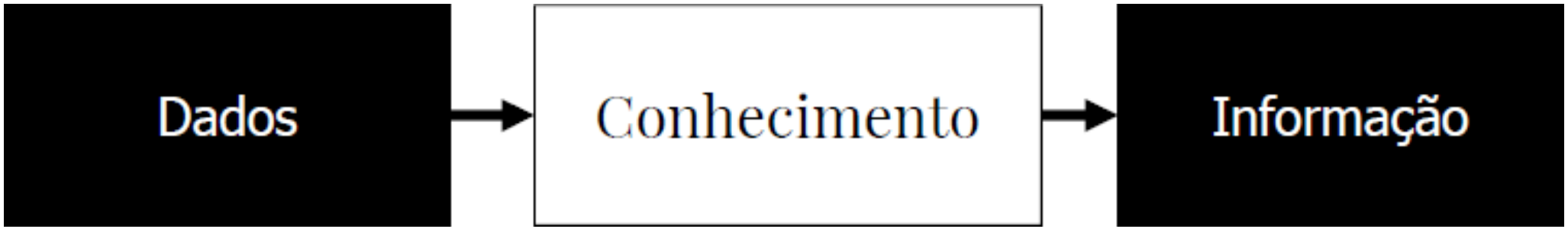
Abstração de Dados

Os dados processados representam uma abstração da realidade, ou seja, **são características selecionadas de ENTIDADES do mundo real**, necessárias para a solução de um determinado problema.



Abstração Conceitual

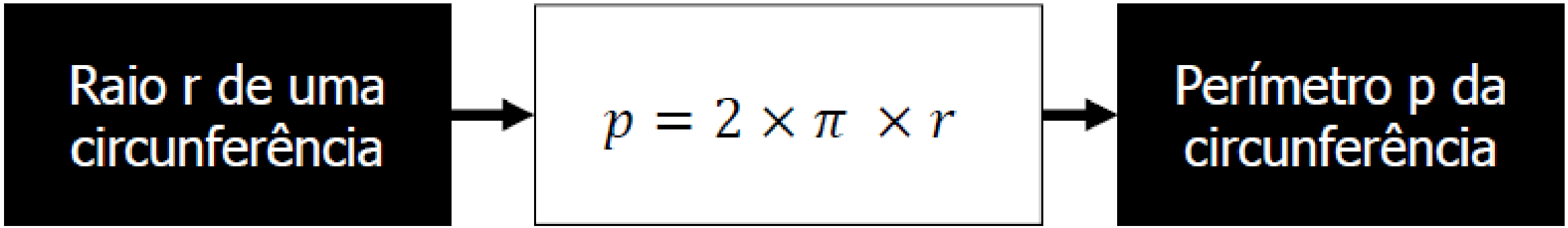
- A abstração de dados sempre é necessária para resolver problemas utilizando um computador.
- Numa abstração mais conceitual...



- Dados são manipulados a partir de um determinado conhecimento, visando produzir novas informações.

Por exemplo...

O valor **r** fornecido como **entrada ao processo** nada mais é que uma **abstração de objetos** do mundo real, que poderiam ser pizzas, pneus, CDs, ou qualquer outro objeto com a mesma forma geométrica.



Tipos Abstratos de Dados (TAD ou TDA)

Um TAD é um **modelo matemático** definido por **um conjunto de valores** e por um **conjunto de operadores** que atuam sobre esses valores.

TAD servem para:

1. Especificar **características relevantes** das **entidades** envolvidas nos **problemas**.
2. Definir **de que forma** as **entidades se relacionam** entre si.
3. Definir **como as entidades** podem ser **manipuladas**.

TAD

- Como o TAD é um **modelo matemático**, não considera **como** os valores são armazenados, nem se preocupa com o **tempo gasto** para efetuar **operações** com esses valores.
- Assim, para usar um TAD para solução de problemas no computador, é preciso transformá-lo num **tipo de dados concreto** ou simplesmente, **tipo de dados**.
- Essa transformação é chamada de **implementação**!
- Em C, essa implementação pode ser tanto pela criação de novos tipos de dados com struct, como para a estruturas de dados clássicas (lista, fila, pilha).

De Abstrato para Concreto

Tipo de Dados Abstrato → Implementação → Tipo de Dados Concreto

Disciplina
- código: String - nome: String - CH: int
+ Métodos Gets e Sets

*Disciplina.java

```java

public class Disciplina {
    private String código;
    private String nome;
    private int CH;

    public String getCódigo() {
        return código;
    }
    public void setCódigo(String código) {
        this.código = código;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getCH() {
        return CH;
    }
    public void setCH(int cH) {
        CH = cH;
    }

}

```