Estruturas de Dados
Tipos Abstratos de Dados

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa
paulo_regis@uvanet.br

Tipos Abstratos de Dados – TAD

- Níveis de abstração
- Modularidade
- TAD's em C

- Um tipo de dados **simples** é caracterizado por um conjunto de valores, tais como os definidos pelos tipos `char`, `int` ou `float`.
- Um tipo de dados **estruturado** define, em geral, uma coleção de mesmo tipo de valores estruturados (*arrays*), ou um agregado de valores de tipos diferentes (*structs*, *unions*).

Um **Tipo Abstrato de Dados** (TAD) pode ser visto como um modelo de dados e um conjunto de procedimentos que atuam com exclusividade sobre os dados encapsulados.

- Qualquer processamento a ser realizado sobre os dados encapsulados em um TAD só poderá ser executado através dos procedimentos definidos no modelo do TAD, sendo esta restrição a característica operacional mais útil dessa estrutura.

- Uma coleção de atividades, tais como:
  1. inserir,
  2. remover e
  3. consultar,

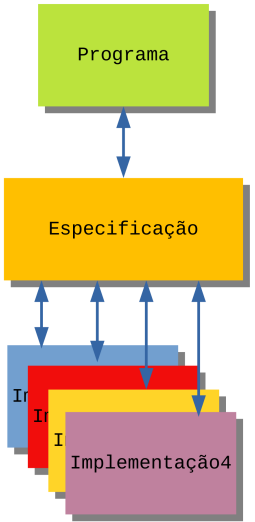
  encapsulada junto com uma estrutura, como uma LISTA, pode ser considerada um tipo abstrato de dados.
- Definido dessa forma, o TAD LISTA fica representado no nível conceitual o nível de abstração mais alto possível.

- Em um nível de abstração mais baixo, denominado nível de design, a estrutura deve ser representada por um **modelo de dados**, por exemplo:
  - sequência,
  - árvore ou
  - grafo

  e as operações devem ser especificadas através de procedimentos cuja representação não dependa de uma linguagem de programação.

- Em um nível de abstração ainda mais baixo, denominado nível de implementação, deve-se tomar como base o *design* do TAD e estabelecer representações concretas para os seus elementos em uma **linguagem de programação específica**.

- **Modularidade** consiste em dividir um software em componentes individuais, rotulados e endereçáveis, chamados módulos.
- A modularidade nos permite implementar o conceito de **encapsulamento**.
- O encapsulamento nos permite que só exista uma forma de acessar nossos dados e uma forma de alterá-los.
- O encapsulamento também **esconde os detalhes** internos de implementação do resto do código. Assim eles podem ser alterados dependendo de plataforma ou situação sem que o resto do software sofra alterações.



Usa o TAD

Define o TAD

Implementa o TAD

- Quando usamos TAD's, nossos sistemas ficam divididos em:
  - **Programas usuário**: as partes que usam o TAD.
  - **Implementações**: as partes que implementam o TAD.
- A linguagem C oferece mecanismos para especificação e uso de TAD's.
- A especificação é possível com o arquivo cabeçalho (`.h`)
  - O arquivo .h possui apenas os protótipos das operações
  - Usar a `#include` para incluir o arquivo `.h`. Inclui o arquivo antes da compilação.
- Os diferentes módulos são incluídos em um único programa executável na "linkagem".

Código 1: "User.h"

```
1 #define LOGIN_MAX_LENGTH 20
2 #define PASSWORD_MAX_LENGTH 20
3 typedef struct User User;
4
5 User *User_alloc(char *login, char *password);
6 void  User_free(User *usr);
7 int   User_authenticate(User *usr, char *login, char *password);
```

Código 2: "User-main.h"

```c
#include <stdio.h>
#include "User.h"

int main() {
    int sucessAuth = 0;
    char login[LOGIN_MAX_LENGTH], pass[PASSWORD_MAX_LENGTH];
    User *usr = User_alloc("admin", "1234");

    do {
        printf("login: ");
        scanf("%s", login);
        printf("password: ");
        scanf("%s", pass);

        sucessAuth = User_authenticate(usr, login, pass);
        if (sucessAuth)
            printf("Usuario autenticado com sucesso.\n");
        else
            printf("Login ou senha invalidos.\n");
    }
    while (!sucessAuth);
    return 0;
}
```

- Implemente a seguinte definição de TAD para um ponto no plano bidimensional.

```
* Point2d_alloc
  - Input: x e y (coordenadas no plano)
  - Output: P (ponto criado)

* Point2d_free
  - Input: P (o ponto)
  - Output: nenhuma. Todos os recusos de memória para P
    estão liberados

* Point2d_getX
  - Input: P (ponto)
  - Output: x
```

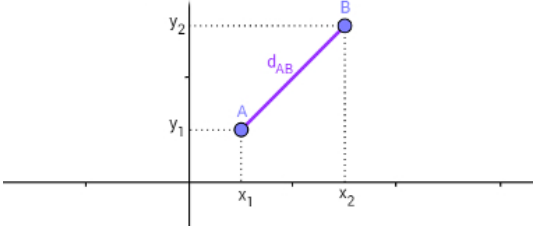
* Point2d_getY
  - Input: P (ponto)
  - Output: y

* Point2d_setX
  - Input: P (ponto), x (coordenada)
  - Output: P'

* Point2d_setY
  - Input: P (ponto), y (coordenada)
  - Output: P'

* Point2d_distance
  - Input: P1 (ponto), P2 (ponto)
  - Output: V (valor da distância entre os pontos)

$$d_{AB} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



- Implemente um TAD Ponto em um arquivo `point2d.h`.
- Implementação do tipo ponto no arquivo `ponto.c`.
- Módulo que usa a implementação do ponto é `prog.c`
  - `#include "Point2d.h"`
  - Inclui o cabeçalho na pré-compilação.
- Compilação:

```
1 gcc -c Point2d.c
2 gcc -c prog.c
```

- Linkagem:

```
3 gcc -o prog Point2d.o prog.o
```