# Estrutura de Dados

IDP

# Tipo abstrato de dados

- Técnica de programação baseada na definição de tipos estruturados, conhecida como **tipo abstrato de dados** (TAD);
- Ideia central: encapsular de quem usa determinado tipo de dado a forma concreta com o que o tipo foi implementado;
- **Exemplo**: um tipo para representar um ponto no espaço; um cliente usa-o de forma abstrata, baseado nas funcionalidades oferecidas;
- **Resultado:** desacoplamos a implementação do uso.
- **Qual a vantagem?**

# Módulos e compilação em separado

- Quando tempos um arquivo com funções que representam apenas parte da implementação de um programa completo, chamamos de **módulo**;
- **Programa**: composto por um ou mais módulos;
- Cada módulo é compilado separadamente, gerando um arquivo objeto para cada um (geralmente com extensão “.o” ou “.obj”);
- **Linker (ligador**): é usado para juntar os arquivos objetos em um único executável;
- É na ligação que os códigos objetos das funções da biblioteca padrão de C são incluídos no objeto. Ou seja, a ligação sempre ocorre.

# Módulos e compilação em separado

- Para programas pequenos, o uso de vários módulos pode não se justificar;
- Para programas de médio e grande porte, a divisão é mandatória;
  - Divisão de tarefa maior e mais complexa em tarefas menores e provavelmente mais fáceis de implementar;
- **Ideia:**
  - O TAD deve ter um arquivo de implementação do módulo (“ponto.c”);
  - Deve sempre incluir a interface do módulo: podem existir definições necessárias na interface; garantir que as funções implementadas correspondem às funções da interface;
- **Exemplo**;

# Vetores Dinâmicos

- Vetores são a forma mais primitiva de representar elementos agrupados;
- Espaço contíguo na memória -> permite acessar qualquer um de seus elementos a partir do ponteiro para o primeiro elemento;
- O nome de um vetor representa um ponteiro para o primeiro elemento do vetor;
- Acesso randômico aos elementos (podemos acessar qualquer um aleatoriamente);
- **Problema: dimensionar o número máximo de elementos;**

# Vetores Dinâmicos

- Temos uma função útil:

*vet = (float *) realloc(vet, m * sizeof(float));*

- A partir dessa realocação, se bem-sucedida, *vet* aponta para uma área de memória contígua suficiente para armazenar *m* valores reais;
- Realloc recebe dois parâmetros: o vetor a ser redimensionado, o novo número de bytes a ser reservado para o vetor;
- **Conceitualmente**:
  - É realizada a alocação de um novo vetor com o novo tamanho;
  - É feita a cópia das posições do vetor anterior para o novo;
  - O vetor anterior é liberado;

# Vetores Dinâmicos

- **Ou seja...**
  - A realocação do tamanho de vetores é possível;
  - É muito útil!
  - **Tem ALTO CUSTO COMPUTACIONAL;**
- O sistema operacional pode usar estratégias para deixar mais eficiente...
  - O sistema tenta preservar o vetor no espaço de memória já reservado para ele;
  - Se a nova dimensão é maior que a original, o sistema verifica se um espaço de memória adicional à frente do vetor está livre;
  - Se sim, o espaço é incorporado e, nesse caso, não é necessário cópia.
  - Mesmo se a nova dimensão for menor, o sistema pode optar por reposicionar o vetor para otimizar o uso da memória (por exemplo, para reduzir espaços fragmentados).
- **Exemplo de vetor dinâmico implementado via abordagem por TAD;**