Tipos abstratos de dados

Profª Simone de Oliveira Santos

Sumário

Introdução

Um **tipo de dado** define o conjunto de valores que uma variável, e ainda as operações que podem ser usadas com esses valores.

- Ex.: variáveis `int` podem assumir valores inteiros em uma faixa definida.

As linguagens de programação trazem tipos pré-definidos, que são os tipos primitivos, e adicionalmente os programadores podem **definir novos tipos de dados** em termos de outros já definidos.

- Ex.: tipos estruturados, vetores, e outros.

Introdução

Nesta aula, discutiremos uma importante técnica de programação baseada na definição de tipos estruturados conhecida como

Tipo Abstrato de Dados (TAD).

- A ideia central é encapsular (esconder) de quem usa determinado tipo a forma concreta com que o tipo foi implementado.

Introdução

- Ao criar um tipo para representar um ponto no espaço, um cliente deste tipo usa-o de forma abstrata, baseando-se apenas nas funcionalidades oferecidas pelo tipo.
- A forma como o tipo foi efetivamente implementado passa a ser um detalhe de implementação, e não afetar o uso do tipo.
- A implementação é desacoplada do uso, o que facilita a manutenção e o potencial de reutilização.
- A implementação do tipo pode ser alterada sem afetar o seu uso em outros contextos.

Módulos e compilação em separado

- Um programa em C pode ser dividido em vários arquivos-fonte (com extensão .c).
- No desenvolvimento é comum identificar funções afins e agrupá-las por arquivo.
- Quando tem-se um arquivo com funções que representam parte da implementação de um programa completo, este é denominado **módulo**.
- Assim, a implementação de um programa pode ser composta por um ou mais módulos.

Construindo um módulo

- Baixe o arquivo chamado `funcrec.c` no sigaa e coloque em uma pasta local reservada para o módulo.
- Na mesma pasta crie um outro arquivo para conter a função main.
- Escreva os protótipos das funções de `funcrec.c` no topo do arquivo principal.
- Faça um programa que armazene uma string qualquer e faça uso das funções que estão no módulo `funcrec.c`

Arquivo principal

```
#include <stdio.h>

int tamanho(char s[]);
//prototipo das demais funções
(...)

int main(){
   char str[50] = "simone";
   printf("Tamanho da string: %d\n", tamanho(str));
}
```

Construindo um módulo

Execute os comandos no terminal para compilar:

```
> gcc -c funcrec.c principal.c
> gcc -o principal funcrec.o principal.o
```

Execute o comando no terminal para executar:

```
> ./principal.exe
```

Construindo um módulo

- Esta forma de construir módulos oferece um problema para módulos com muitas funções.
- Ter que escrever manualmente todos os protótipos no arquivo principal.
- Para contornar esse problema, todo módulo de funções C costuma ter associado um arquivo que contem apenas **os protótipos das funções** oferecidas pelo módulo.

Construindo um módulo

- Este arquivo de protótipo caracteriza a **interface do módulo** e é feito **com o mesmo nome do módulo**, só que com extensão .h
- Assim, deve-se criar um arquivo `funcrec.h` para o módulo que conterá apenas os protótipos das funções.

Arquivo funcrec.h

```
/*Função que retorna o tamanho de uma string passada como parâmetro.*/
int tamanho(char *string);

/* Função que retorna a quantidade de ocorrencias de uma
letra em uma string passados como parâmetro.*/
int ocorrencia(char *str, char letra);

/*Função recursiva que imprime uma string.*/
void imprime(char *str);

/*Função recursiva que imprime o reverso de uma string.*/
void imprimeReverso(char *str);
```

Construindo um módulo

- Altere o arquivo principal e retire os protótipos.
- Faça a inclusão do módulo `funcrec.h` no topo do arquivo principal através da diretiva `include` da seguinte maneira:

```
#include "funcrec.h"
```

- O arquivo principal deverá ficar como segue:

Arquivo principal

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

int main(){
   char str[50] = "simone";
   printf("Tamanho da string: %d\n", tamanho(str));
}
```

Módulos

- Geralmente, um módulo agrupa vários tipos e funções com funcionalidades relacionadas, caracterizando assim uma funcionalidade bem definida.
- Nos casos em que o módulo define **um novo tipo de dado e o conjunto de operações** para manipular dados desse tipo, dizemos que o módulo é um **TAD**.

TAD

A interface de um TAD consiste basicamente:

- na definição do nome do tipo, e
- do conjunto de funções exportadas para sua criação e manipulação.

Prática:
Vamos criar um TAD para representar um ponto no espaço $\mathbb{R}^2$

TAD Ponto no $\mathbb{R}^2$

O tipo `Ponto` consiste de dois valores do tipo `float`, representando as coordenadas, e conterá as seguintes operações:

`cria`	cria um ponto com coordenadas `x` e `y`;
`libera`	libera memória alocada por um ponto;
`acessa`	retorna as coordenadas de um ponto;
`atribui`	atribui novos valores às coordenadas de um ponto;
`distancia`	calcula a distância entre dois pontos.

Prática

- Baixe os arquivos do TAD Ponto no Sigaa
- Crie um arquivo que contenha a função `main` para fazer uso do TAD Ponto.
- Crie dois pontos;
- Calcule e mostre a distância entre esses dois pontos;
- Mostre as coordenadas de um dos pontos;
- Altere o valor de um dos pontos;