TAD:
Tipo Abstrato de Dados
(parte 2)

SCC0202 – Algoritmos e Estruturas de Dados I

Prof. Thiago A. S. Pardo

Modularização em C

- Programa em C pode ser dividido em vários arquivos
  - Arquivos **fonte** com extensão **.c**
    - Denominados de módulos
- Cada módulo deve ser compilado separadamente
  - Para tanto, usa-se um **compilador**
  - Resultado: **arquivos objeto** não executáveis
    - Arquivos com extensão **.o** ou **.obj**
- Arquivos objeto devem ser juntados em um **executável**
  - Para tanto, usa-se um *ligador* ou **link-editor**
  - Resultado: um único arquivo em linguagem de máquina
    - Usualmente com extensão **.exe**

Modularização em C

- Módulos são muito úteis para construir bibliotecas de funções inter-relacionadas. Por exemplo:
  - Módulos de funções matemáticas
  - Módulos de funções para manipulação de strings
  - Etc.

- Em C, é preciso listar no início de cada módulo aquelas funções de outros módulos que serão utilizadas:
  - Isso é feito através de uma lista denominada **cabeçalho**

Modularização em C

- Módulos são muito úteis para construir bibliotecas de funções inter-relacionadas. Por exemplo:
  - Módulos de funções matemáticas
  - Módulos de funções para manipulação de strings
  - Etc.

- Em C, é preciso listar no início de cada módulo aquelas funções de outros módulos que serão utilizadas:
  - Isso é feito através de uma lista denominada **cabeçalho**

- **Exemplo**: considere um arquivo STR.c contendo funções implementadas para manipulação de strings, dentre elas:
  - **int** comprimento (**char*** strg) { ... }
  - **void** copia (**char*** dest, **char*** orig) { ... }
  - **void** concatena (**char*** dest, **char*** orig) { ... }

Modularização em C

- **Exemplo** (cont): Qualquer módulo que utilizar essas funções deverá incluir no início o cabeçalho das mesmas, como abaixo.

```
/* Programa Exemplo.c */
#include <stdio.h>
int comprimento (char* str);
void copia (char* dest, char* orig);
void concatena (char* dest, char* orig);
int main (void) {
        char str[101], str1[51], str2[51];
        printf("Entre com uma sequência de caracteres: ");
        scanf(" %50s[^\n]", str1);
        printf("Entre com outra sequência de caracteres: ");
        scanf(" %50s[^\n]", str2);
        copia(str, str1);  concatena(str, str2);
        printf("Comprimento total: %d\n",comprimento(str));
        return 0; }
```

Modularização em C

- **Exemplo** (cont):
  - A partir desses dois fontes (Exemplo.c e STR.c), podemos gerar um executável compilando cada um separadamente e depois ligando-os
  - Por exemplo, com o compilador C (gcc), utilizaríamos a seguinte sequência de comandos para gerar o arquivo executável Teste.exe:

```
> gcc -c STR.c

> gcc -c Exemplo.c

> gcc -o Teste.exe STR.o Exemplo.o
```

- **Questão:**
  - É preciso inserir manualmente e individualmente todos os cabeçalhos de todas as funções usadas por um módulo?
    - E se forem muitas e de diferentes módulos?

Modularização em C

- **Solução:**
  - **Arquivo de cabeçalhos** associado a cada módulo, com:
    - cabeçalhos das funções oferecidas pelo módulo e,
    - eventualmente, os tipos de dados que ele **exporta**
      - typedefs, structs, etc.
  - Segue o mesmo nome do módulo ao qual está associado
    - porém com a extensão **.h**

- Exemplo:
  - Arquivo STR.h para o módulo STR.c do exemplo anterior

Modularização em C

```
/* Arquivo STR.h */

/* Função comprimento:
   Retorna o no. de caracteres da string str */
int comprimento (char* str);

/* Função copia:
   Copia a string orig para a string dest */
void copia (char* dest, char* orig);

/* Função concatena:
   Concatena a string orig na string dest */
void concatena (char* dest, char* orig);
```

Modularização em C

- O programa Exemplo.c pode então ser rescrito como:

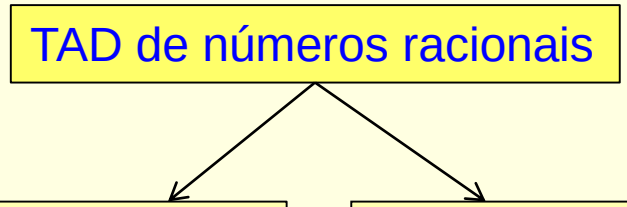
```
/* Programa Exemplo.c */
#include <stdio.h>  /* Módulo da Biblioteca C Padrão */
#include "STR.h"    /* Módulo Próprio */
int main (void) {
        char str[101], str1[51], str2[51];
        printf("Entre com uma sequência de caracteres: ");
        scanf(" %50s[^\n]", str1);
        printf("Entre com outra sequência de caracteres: ");
        scanf(" %50s[^\n]", str2);
        copia(str, str1);  concatena(str, str2);
        printf("Comprimento total: %d\n",comprimento(str));
        return 0; }
```

Nota: O uso dos delimitadores < > e " " indica onde o compilador deve procurar os arquivos de cabeçalho – na biblioteca interna (<>) ou começando pelo diretório corrente (" ")

TADs em C

- Módulos podem ser usados para definir um novo tipo de dado e o conjunto de operações para manipular dados desse tipo:
  - **Tipo Abstrato de Dados (TAD)**
- Definindo um tipo *abstrato*, pode-se “esconder” a implementação
  - Quem usa o tipo abstrato precisa apenas conhecer a funcionalidade que ele implementa, não a forma como ele é implementado
  - Facilita manutenção e re-uso de códigos, entre outras vantagens

Exemplo da aula anterior



Programa 1

```
programa ensino números racionais
    usar TAD de números racionais
início
    declarar r racional
    imprimir("Agora vamos aprender...")
    ler_numeros(r,1)
    ...
fim
```

Programa 2

```
programa cálculos matemáticos
    usar TAD de números racionais
início
    declarar i inteiro
    declarar r(10) racional
    para i=1 até 10 faça
            ler_numeros(r,i)
    calcular_media(r,10)
    ...
fim
```

Questão para responder na próxima aula

- Como compilar um arquivo .h?
  - No gcc
  - Em uma IDE como Codeblocks
    - Por que se usa um IDE?