TAD: Tipo Abstrato de Dados (parte 2)

Baseado no material de Thiago A. S. Pardo

SCC-202 – Algoritmos e Estruturas de Dados I

Debora Medeiros

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 em linguagem de máquina 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**
- **Exemplo**: considere um arquivo STR.c contendo funções 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 seqüência de caracteres: ");
        scanf(" %50s[^\n]", str1);
        printf("Entre com outra seqüência 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 Gnu C (gcc) utilizaríamos a seguinte seqüência 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
  - Como seria o STR.h?
  - E o Exemplo.c?

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 seqüência de caracteres: ");
        scanf(" %50s[^\n]", str1);
        printf("Entre com outra seqüência 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 no diretório indicado (" " – *default* se ausente).

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

TADs em C: Exemplo

```
/* TAD: Matriz m por n */

/* Tipo Exportado */
typedef struct matriz Matriz;

/* Funções Exportadas */
/* Função cria - Aloca e retorna matriz m por n   */
Matriz* cria (int m, int n);
/* Função libera - Libera a memória de uma matriz */
void libera (Matriz* mat);

                                        /* Continua... */
```

Arquivo matriz.h

TADs em C: Exemplo

```
/* Continuação... */

/* Função acessa - Retorna o valor do elemento [i][j]  */
float acessa (Matriz* mat, int i, int j);
/* Função atribui - Atribui valor ao elemento [i][j]   */
void atribui (Matriz* mat, int i, int j, float v);
/* Função linhas - Retorna o no. de linhas da matriz   */
int linhas (Matriz* mat);
/* Função colunas - Retorna o no. de colunas da matriz */
int colunas (Matriz* mat);
```

Arquivo matriz.h

TADs em C: Exemplo

```
#include <stdlib.h> /* malloc, free, exit */
#include <stdio.h> /* printf */
#include "matriz.h"

struct matriz {
        int lin;
        int col;
        float* v;
};

void libera (Matriz* mat){
   free(mat->v);
   free(mat);
}
```

Deve estar incluído no .c correspondente

Arquivo matriz.c

TADs em C: Exemplo

```
/* Continuação... */

Matriz* cria (int m, int n) {
   Matriz* mat = (Matriz*) malloc(sizeof(Matriz));
   if (mat == NULL) {
        printf("Memória insuficiente!\n");
        exit(1);
   }
   mat->lin = m;
   mat->col = n;
   mat->v = (float*) malloc(m*n*sizeof(float));
   return mat;
}
```

Arquivo matriz.c

TADs em C: Exemplo

```
/* Continuação... */

float acessa (Matriz* mat, int i, int j) {
   int k; /* índice do elemento no vetor */
   if (i<0 || i>=mat->lin || j<0 || j>=mat->col) {
        printf("Acesso inválido!\n");
        exit(1);
   }
   k = i*mat->col + j;
   return mat->v[k];
}

int linhas (Matriz* mat) {
   return mat->lin;
}
```

Arquivo matriz.c

TADs em C: Exemplo

```
/* Continuação... */

void atribui (Matriz* mat, int i, int j, float v) {
   int k; /* índice do elemento no vetor */
   if (i<0 || i>=mat->lin || j<0 || j>=mat->col) {
        printf("Atribuição inválida!\n");
        exit(1);
   }
   k = i*mat->col + j;
   mat->v[k] = v;
}

int colunas (Matriz* mat) {
   return mat->col;
}
```

Arquivo matriz.c

Programa cliente – que usa o TAD

```
#include <stdio.h>
#include <stdlib.h>
#include "matriz.h"

int main(int argc, char *argv[])
{
  float a,b,c,d;
  Matriz *M;

  // criação de uma matriz
  M = cria(5,5);

  // inserção de valores na matriz
  atribui(M,1,2,40);
  atribui(M,2,3,3);
  atribui(M,3,0,15);
  atribui(M,4,1,21);
```

Programa cliente – que usa o TAD

```
/* Continuação... */

// verificando se a inserção foi feita corretamente
  a = acessa(M,1,2);
  b = acessa(M,2,3);
  c = acessa(M,3,0);
  d = acessa(M,4,1);

  printf ("M[1][2]: %4.2f \n", a);
  printf ("M[2][3]: %4.2f \n", b);
  printf ("M[3][0]: %4.2f \n", c);
  printf ("M[4][1]: %4.2f \n", d);


  system("PAUSE");
  return 0;
}
```