ICMC USP SÃO CARLOS
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

TAD:
Tipo Abstrato de Dados
(parte 2)

SCE-202 – Algoritmos e Estruturas de Dados I

Lucas Antiqueira

Modularização em C

- Programa em C pode ser dividido em vários arquivos
  - Arquivos **fonte** com extensão **.c**
    - Denominados *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 unidos 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 (**cabeçalho**) aquelas funções de outros módulos que serão utilizadas:

- **Exemplo**: considere um arquivo STR.c contendo funções para manipulação de strings, dentre elas:
  - **int** comprimento (**char*** str)
  - **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

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 reescrito 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);

/* Continua... */
```

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;
}
```

Exercício: TAD Conjuntos (SET)

- Um conjunto é uma coleção de membros (ou elementos); cada membro ou é um conjunto ou um elemento primitivo chamado de átomo.

- Todos os membros são diferentes: nenhum conjunto contém 2 cópias do mesmo elemento.

- Ex:

{1,4} → certo

{1,4,1} → errado

Operações básicas: união, intersecção e diferença

- Se A e B são conjuntos, então $A \cup B$ é o conjunto de elementos que são membros de A ou de B ou de ambos
- Se A e B são conjuntos, então $A \cap B$ é o conjunto de elementos que estão em A e em B
- Se A e B são conjuntos, então $A - B$ é o conjunto de elementos em A que não estão em B
- Exemplo: $A = \{a,b,c\}$ $B = \{b,d\}$

  $A \cup B = \{a,b,c,d\}$

  $A \cap B = \{b\}$

  $A - B = \{a,c\}$

Conjuntos em C

- Como implementar um conjunto em C?

Conjuntos em C

- Como implementar um conjunto em C?

```
#define N 100 /* por exemplo, conjunto que tem
  números de 1 a 100 */
int conjunto[N]; /* conjunto[i]=1 se i está no
  conjunto; 0, caso contrário */
```

Operações?

Operações usuais

1. União(A,B,C)
2. Intersecção(A,B,C)
3. Diferença(A,B,C)
4. Membro(x,A)
5. Cria_Conj_Vazio(A)
6. Insere(x,A)
7. Remove(x,A)
8. Atribui(A,B)
9. Min(A)
10. Max(A)
11. Igual(A,B)

Definição das operações

- União(A,B,C): toma os argumentos A e B que são conjuntos e retorna A $\cup$ B à variavel C
- Intersecção(A,B,C): toma os argumentos A e B que são conjuntos e retorna A $\cap$ B à variavel C
- Diferença(A,B,C): toma os argumentos A e B que são conjuntos e retorna A - B à variavel C
- Membro(x,A): toma o conjunto A e o objeto x cujo tipo é o tipo do elemento de A e retorna um valor booleano – true se x $\in$ A e false caso contrário
- Cria_Conj_Vazio(A): faz o conjunto vazio ser o valor para a variável conjunto A

Definição das operações

- Insere(x,A): toma o conjunto A e o objeto x cujo tipo é o tipo do elemento de A e faz x um membro de A. O novo valor de A = A $\cup$ {x}. Se x já é um membro de A, então a operação insere não muda A

- Remove(x,A): remove o objeto x, cujo tipo é o tipo do elemento de A, de A. O novo valor de A = A – {x}. Se x não pertence a A então a operação remove não altera A

Definição das operações

- Atribui(A,B): Seta o valor da variável conjunto A = ao valor da variável conjunto B
- Min(A): retorna o valor mínimo no conjunto A. Por exemplo: Min({2,3,1}) = 1 e Min({‘a’,’b’,’c’}) = ‘a’
- Max(A): Similar a Min(A) só que retorna o máximo do conjunto
- Igual(A,B): retorna true se e somente se os conjuntos A e B consistem dos mesmos elementos

Exercício

- Em duplas, implemente em C o TAD conjunto de números inteiros
  - Use um arquivo .h

TADs em C: Exemplo

```
/* TAD: conjunto*/

/* Tipo Exportado */
typedef struct conjunto Conjunto;

/* Funções Exportadas */


/* Função união – Une os elementos do conjunto A e B em um
   conjunto C.Retorna 1 se a operação for bem sucedida e 0 caso
   contrario */
int uniao (Conjunto *A, Conjunto *B, Conjunto *C);
/* Função Intersecção – Armazena em C os mesmos elementos que
   estão no conjunto A e B*/
void interseccao (Conjunto *A, Conjunto *B, Conjunto *C);
/* Função libera – Libera a memória de um conjunto */
void libera (Conjunto *A);
```

Arquivo conjunto.h

TADs em C: Exemplo

```
/* Continuação... */

/* Função acessa – atribui ao conjunto C a diferença entre os
   conjuntos A e B  */
void diferença(Conjunto *A, Conjunto *B, Conjunto *C);
/* Função membro – verifica se o elemento elem está no Conjunto
   A   */
int membro(elem x, Conjunto *A);
/* Função Cria_Conj_Vazio – Cria um conjunto vazio e retorna o
   conjunto criado. A variável erro retorna 0 se o conjunto foi
   criado corretamente e 0 caso contrario. Deve ser usado como
   primeira operação sobre um conjunto. */
Conjunto *Cria_Conj_Vazio(int *erro);
/* Função insere – insere o elemento elem no conjunto A e
   retorna se a execução foi realizada com sucesso(1) ou não(0)*/
int insere(elem x, Conjunto *A);
```


Arquivo conjunto.h

TADs em C: Exemplo

```
/* Continuação... */
/* Função remove (diferente da de PASCAL) – remove o elemento
   elem do Conjunto A, retorna 1 se o elemento foi retirado e 0
   se o elemento não está no conjunto */
int remove(elem x, Conjunto *A);
/* Função Atribui – faz a copia do conjunto A para o B*/
void atribui(Conjunto *A, Conjunto *B);
/* Função min – retorna o menor elemento do conjunto A – se o
   conjunto está vazio retorna TAM */
elem min(Conjunto *A);
```

Arquivo conjunto.h

TADs em C: Exemplo

```
/* Continuação... */

/* Função max – retorna o maior elemento do conjunto A - se o
   conjunto está vazio retorna TAM */
elem max(Conjunto *A);
/* Função igual – verifica se o conjunto A é igual a
Conjunto B   */
int igual(Conjunto *A, Conjunto *B);

/* Função tamanho – retorna o tamanho do conjunto A */

int tamanho(Conjunto *A);

/* Função testa_vazio – verifica se o conjunto A é vazio 1 se
   for vazio e 0 caso contrario*/

int testa_vazio(Conjunto *A);
```


Arquivo conjunto.h

TADs em C: Exemplo

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


#define TAM 200 // trabalha com elementos do universo 0..TAM - 1
typedef int elem;

struct conjunto {
        elem* v; //vetor booleano que armazenará o conjunto sendo que
         //o índice armazena o valor sendo true se o elemento está
         // no conjunto, false caso contrário
};

void libera (Conjunto *A){
   free(A->v);
   free(A);
}
```


Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

int uniao (Conjunto *A, Conjunto *B, Conjunto *C) {
   int i;// variável auxiliar para realização do loop
   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) || (B->v[i]==1) {
        C->v[i]=1;
        }
   }
  return 1; /*na implementação com vetores de booleanos sempre é possível
   realizar a união, pois o conjunto de elementos é limitado a um universo*/
}

void interseccao(Conjunto *A, Conjunto *B, Conjunto *C) {
   int i; // variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) && (B->v[i]==1) {
        C->v[i]=1;
        }
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */
/* faz o conjunto vazio ser o valor para a variável conjunto A. Deve ser
   usado como primeira operação sobre um conjunto.*/
Conjunto* Cria_Conj_Vazio(int *erro) {
   int i; // variável auxiliar para realização do loop
   Conjunto* conj = (Conjunto*) malloc(sizeof(Conjunto));
   if (conj == NULL) {
        erro= 1;
        exit(1);
   }
   conj->v = (int*)malloc(TAM*sizeof(int));
   for(i=0;i<TAM;i++){
     conj->v[i]=0;
   }
   erro = 0;
   return conj;
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */
int insere(elem x, Conjunto *A) {
   if (x>=TAM || x<0 ) {
        return 0;
   }
   A->v[x]=1;
   return 1;
}

int membro(elem x, Conjunto *A) {
   if (x>=TAM || x<0 ||(A->v[x]==0)) {
     return 0;
   }
   return 1;
}
```

Arquivo conjunto.c

35

TADs em C: Exemplo

```
/* Continuação... */
int remove(elem x, Conjunto *A) {
   if (x>=TAM || x<0 ||(A->v[x]==0)) {
        return 0;
   }
   A->v[x]=0;
   return 1;
}

void atribui(Conjunto *A, Conjunto *B) {
   int i; // variável auxiliar para realização do loop
   for(i = 0; i<TAM;i++){
     B->v[i]=A->v[i];
   }
}
```

Arquivo conjunto.c

36

TADs em C: Exemplo

```
/* Continuação... */

elem min (Conjunto *A) {
   int i;
   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) {
        return i;
     }
   }
   return TAM; // condição de conjunto vazio
}

elem max (Conjunto *A) {
   int i;
   for(i = TAM - 1; i>-1;i--){
     if (A->v[i]==1) {
        return i;
     }
   }
   return TAM; // condição de conjunto vazio
}
```

Arquivo conjunto.c

TADs em C: Exemplo

```
/* Continuação... */

void diferenca(Conjunto *A, Conjunto *B, Conjunto *C){
   int i; // variável auxiliar para realização do loop

   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) && (B->v[i]==0) {
        C->v[i]=1;
        }
}

int igual(Conjunto *A, Conjunto *B){
   int i;
   for(i = 0; i<TAM;i++){
     if (A->v[i]!=B->v[i]) {
        return 0;
     }
   }
   return 1;
}
```

Arquivo conjunto.c

TADs em C: Exemplo

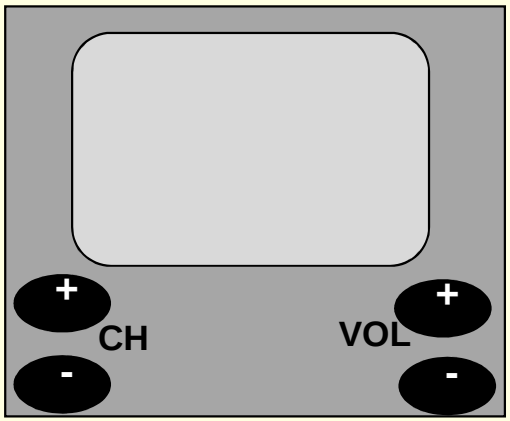
```
/* Continuação... */

int tamanho(Conjunto *A){
   int i,tam; // variável auxiliar para realização do loop e para a
   // verificação do tamanho do conjunto
   tam = 0;
   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) {
       tam++;
     }
   }
   return tam;
}

int testa_vazio(Conjunto *A);{
   int i;
   for(i = 0; i<TAM;i++){
     if (A->v[i]==1) {
        return 0;
     }
   }
   return 1;
}
```

Arquivo conjunto.c

Tipo abstrato de dados

- *Pensamento do dia*

Nunca desmonte uma TV para aumentar o volume. Use o botão!



Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*