Tipo Abstrato de Dados

Estrutura de Dados
Prof. Anselmo C. de Paiva
Dep. de Informática

Tipo Abstrato de Dados - TAD

- Estrutura de dados e coleção de funções c
  estrutura
  - Especifica o tipo de dado (domínio e operaç a
    implementação
  - Minimiza código do programa que usa detal
    - Mais liberdade para mudar implementação com menor impacto nos programas
    - Minimiza custos
- Programas que usam o TAD não "conhecem" as implementações dos TADs
  - Usam TAD através de operações → Interface do TAD

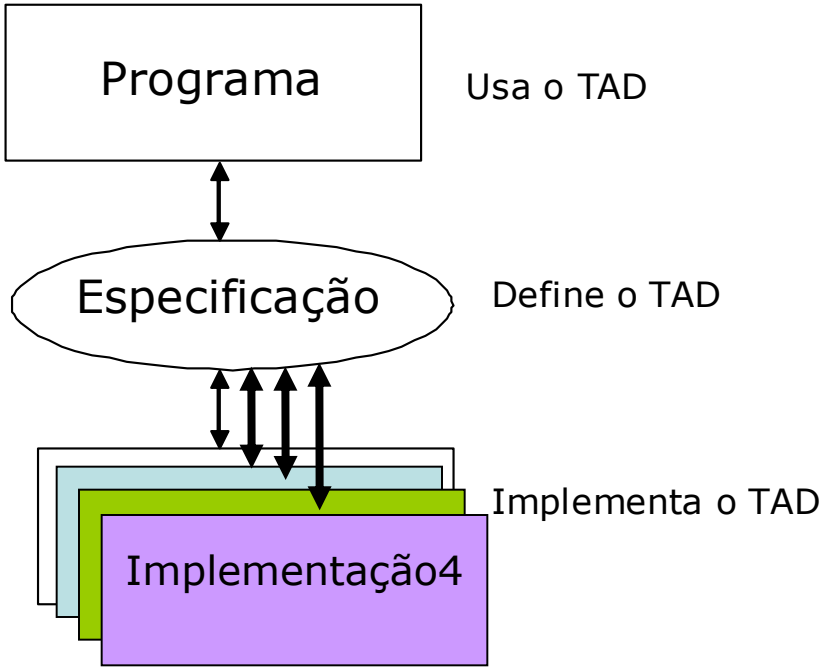
Tipo Abstrato de Dados

- Exemplo:
  - Programas sempre manuseiam coleções de itens (Analogia: Cofo de caranguejo)
  - Operações:
    - Criar → cria uma nova coleção
    - Inserir → adiciona um novo item à coleção
    - Remover → retira um item da coleção
    - Buscar → encontra um item na coleção atendendo algum critério
    - Destruir → Destroi a coleção

Resumindo (TAD)

- Especifica tudo que se precisa saber para usar um determinado tipo de dado
- Não faz referência à maneira com a qual o tipo de dado será (ou é) implementado
- Programas que usam TAD ficam divididos em:
  - Programas usuários: A parte que usa o TAD
  - Implementação: A parte que implementa o TAD

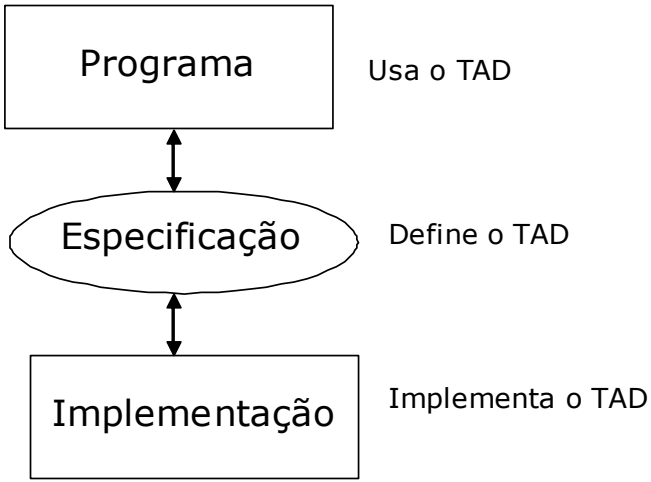
Resumindo (TAD)



Especificação do TAD

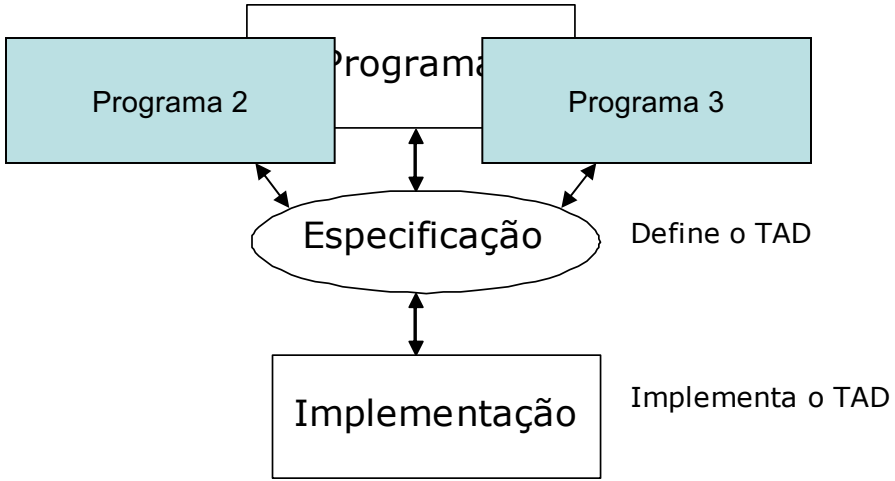
- Definir para cada operação:
  - Inputs, outputs
    - valores de entrada e a saída da operação
  - Pré-condições
    - Propriedades dos inputs que são assumidas pela operações
  - Pós-condições
    - Efeitos causados como resultado da execução da operação
  - Invariantes
    - Propriedades que devem ser sempre verdadeiras
      - antes,
      - durante
      - após

Software em Camadas



- Camadas de software são independentes
- Modificações na implementação do TAD não geram (grandes) mudanças no programa

Software em Camadas



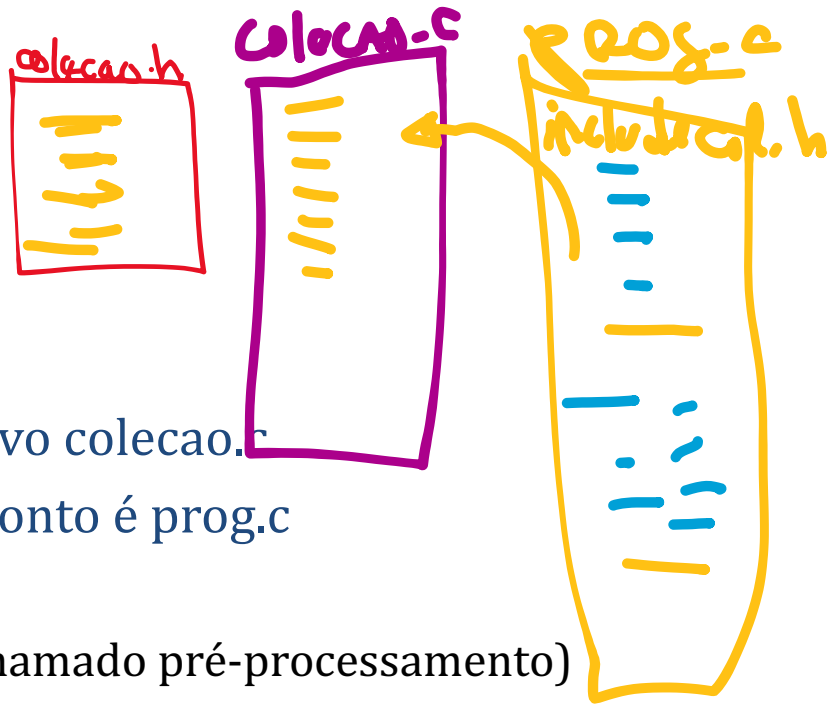
- Abordagem também permite o reuso de código
- Mesma implementação pode ser usada por vários programas

TADs em C

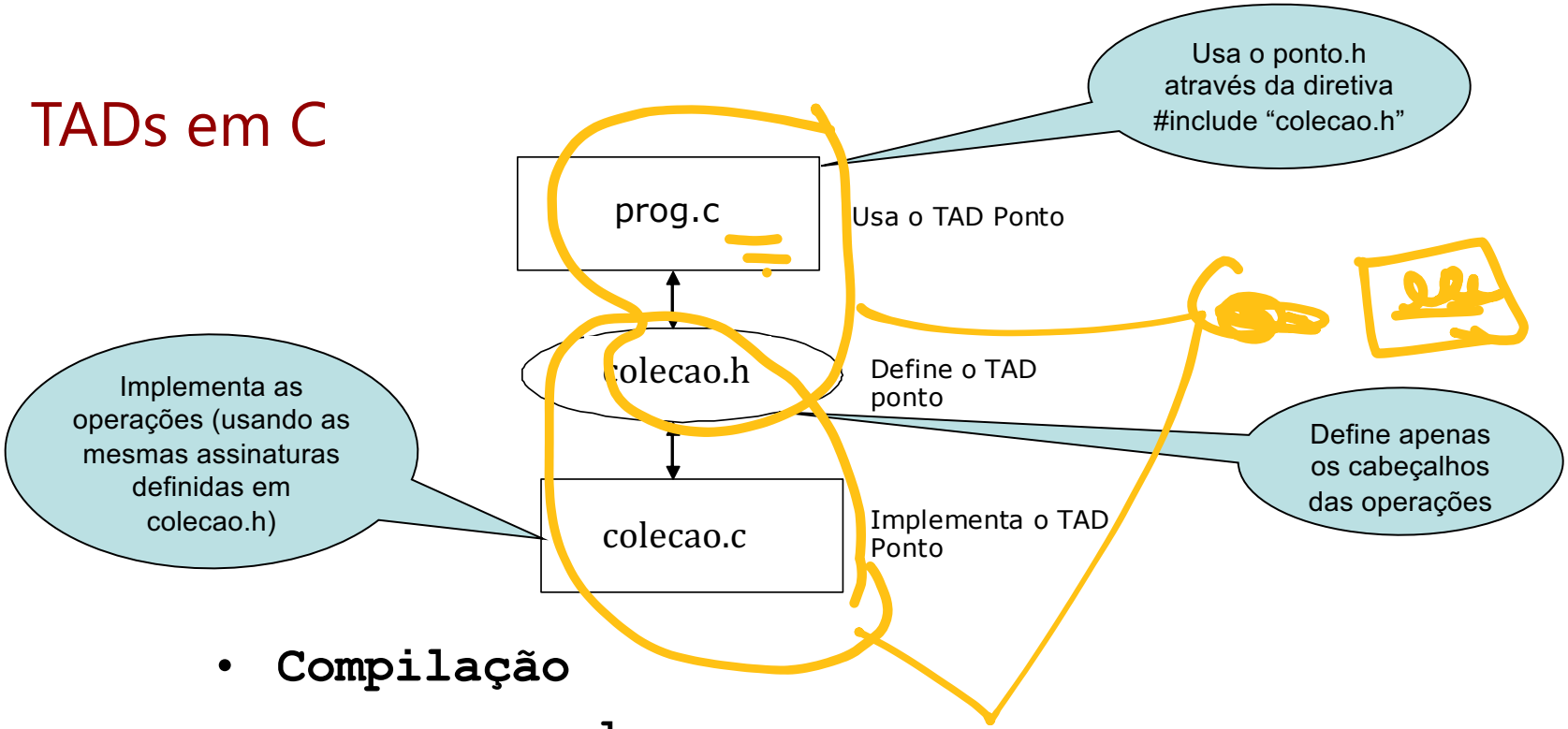
- Linguagem C oferece mecanismos para especificação e uso de TADs:
  - Mecanismos de modularização de programas
- Especificação do TAD
  - arquivo cabeçalho (.h)
    - protótipos das operações
  - Usar a #include para incluir o arquivo .h.
  - Inclui o arquivo antes da compilação
- Os diferentes módulos são incluídos em um único programa executável na “ligação”

TADs em C

- Exemplo:
  - TAD Colecao no arquivo colecao.h
  - Implementação do tipo ponto no arquivo colecao.c
  - Módulo que usa a implementação do ponto é prog.c
    - #include “colecao.h”
    - Inclui o cabeçalho na pré-compilação (chamado pré-processamento)



TADs em C



- **Compilação**
  - `gcc -c colecao.c`
  - `gcc -c prog.c`
- **Linkagem**
  - `gcc -o prog.exe colecao.o prog.o`

Abstração e Encapsulamento

- Abstração
  - “habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais”
    - Definição de TAD
      - concentra nos aspectos essencias do tipo de dado (operações) e abstrai como ele foi implementado
- Encapsulamento
  - “Consiste na separação de aspectos internos e externos de um objeto”.
  - TAD provê mecanismo de encapsulamento de um tipo de dado
    - separação
      - aspecto externo → Interface
      - implementação (aspecto interno)

Colecao – Conjunto Dinâmico

- conjuntos
  - fundamentais para a ciência da computação
- conjuntos dinâmicos (Cormen,2002)
  - Conjuntos manipulados por programas que podem crescer, encolher ou sofrer outras mudanças ao longo do tempo
- Elementos de um conjunto dinâmico
  - cada elemento é representado por um objeto cujos campos podem ser examinados e manipulados se tivermos um ponteiro para ele
  - um dos campos do elemento é um campo de chave de identificação.
  - Se as chaves são todas diferentes temos um conjunto de valores de chaves
  - Se chaves são extraídas de um conjunto totalmente ordenado
    - Pode ser definido: elemento mínimo, próximo elemento, elemento maior que um dado elemento

Colecao – Conjunto Dinâmico - Operações

- Consultas : simplesmente retornam informações
  - Busca(S,k)
    - retorna um ponteiro x para um elemento em S tal que chaue[x] = k, ou NULL
  - Minimo(S)
    - retorna o elemento de S com a menor chave.
  - Maximo (S)
    - retorna elemento de S com a maior chave.
  - Sucessor(S,x)
    - retorna o maior elemento seguinte em S, ou NIL se x é o elemento Máximo
  - Predecessor(S,x)
    - retorna o menor elemento seguinte em S, ou NIL se x é o elemento mínimo.

- Modificação: alteram o conjunto
  - Insere(S,x)
    - aumenta o conjunto S com o elemento x.
  - Remove (S,x)
    - remove x de S

TAD Coleção

- Coleção pode ser implementada em uma linguagem de programação com:
  - Declaração de tipo

    typedef struct _colecao_ Colecao;
  - Conjunto de funções representando as operações

    Colecao *colCreat(int maxItems, int itemSize);

    int colInsert(Colecao *c, int item);

    int colRemove(Colecao *c, int item);

    int colQuery(Coleca *c, int chave);
    - ficam no arquivo de cabeçalho (header) *colecao.h*
    - incluído em todos os arquivos que utilizarem o TAD

    int colDestroy (Colecao *c);
- Não sabemos como esta implementada a coleção
  - apenas sabemos sua especificação.

Coleção.h

```
/*-------------------------------------------------
 Colecao.h
Arquivo com a especificação para o TAD Colecao,
 tipo de dado para  uma coleção de inteiros
Exemplo do curso: Estrutura de Dados
-------------------------------------------------
 Autor:  Anselmo Cardoso de Paiva (ACP)
                        September/2000
-------------------------------------------------*/
#ifndef __COLECAO_H
#define __COLECAO_H
/*-------------------------------------------------
 Definicoes locais
-------------------------------------------------*/
typedef struct _colecao_ Colecao;
/*-------------------------------------------------
 Funcoes que implementam as operacoes do
 TAD ColecaoInt
-------------------------------------------------*/
```

```
/*    Cria um novo TAD ColecaoInt
  Pre-condicao: max_items > 0
  Pos-condicao: retorna um ponteiro para uma novo
   TAD ColecaoInt vazio ou NULL em caso de erro*/
Colecao *colCreate( int max_itens );
/*  Adiciona um item na Collecao
 Pre-condicao : (c é um TAD Colecao criado por uma chamada a colCriar)
e  (o TAD Colecao nao esta cheio) e   (item != NULL)
  Pos-condicao: item foi adicionado ao TAD c */
int colInsert( Colecao *c, int item );
/*  Retira um item da colecao
  Pre-condicao: (c é um TAD Colecao criado por uma chamada a  colCriar)
e && (existe pelo menos um item no TAD Colecao) e   (item != NULL)
  Pos-condicao: item foi eliminado do TAD c*/
int colRemove( Colecao *c, int *item );
```

Colecao.h

```
/*   Encontra um item em um TAD Colecao
 Pre-condicao: (c é um TAD Colecao criado por uma chamada a colCriar) e  (key != NULL)
Pos-condicao: retorna um item identificado por key se ele existir no TAD c, ou return NULL
caso contrátrio
*/
int colQuery( Colecao *c, int key );
/* Destroi um TAD Colecao
   Pre-condicao: (c é um TAD Colecao criado por uma chamada a  colCriar)
   Pos-condicao: a memoria usada pelo TAD foi liberada
*/
int  colDestroyr( Colecao *c );

#endif  /* __Colecao_INT_H */
```

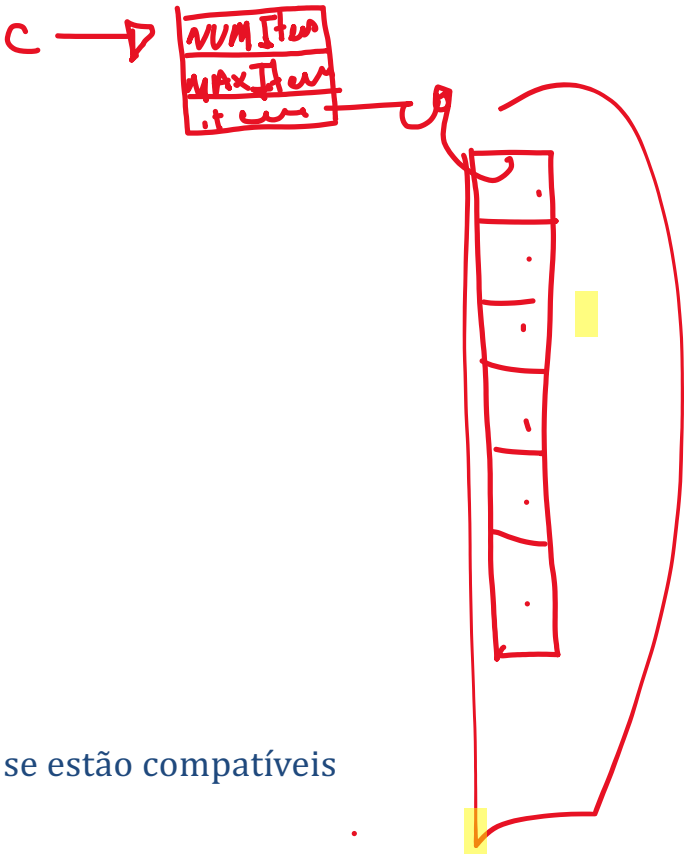
Questão:

- Suponha que alguém forneça a você um TAD Coleção implementado
  - pode ler apenas o arquivo de cabeçalho
  - pode ligar com o seu programa um arquivo com a implementação desse TAD.
- Pode escrever um programa que utiliza este TAD como um tipo de dado?
  - Sim
  - conhece o nome do novo tipo de dado, o suficiente para declarar variáveis ponteiros para Colecao
  - conhece os cabeçalhos e as especificações para cada operação

```
# include “colecao.h”
void main (void)
{
    Colecao *col; int elm=0; int cons;
    col = colCreate( 10 );
     if ( col != NULL) {
        while ( elm >= 0) {
              scanf( “%d”, &elm);
              colInsert( col, elm );
         }
        while ( elm >= 0) {
              scanf( “%d”, &elm);
              cons = colQuery( col, elm );
              if ( cons == elm) {
                 printf( “%d estava na coleção”, cons);
                  colRemove( col, elm );
                  cons = colQuery( col, elm );
                   if ( cons == elm) {
                      printf( “%d estava na coleção”, cons);
                    } else {
                        printf( “%d não estava estava na coleção”, cons);
                    }
        }
}
```

Prog.c

Exemplo de Implementação - Vetor



- Maneira mais simples de implementar uma colecao
  - usar um vetor para armazenar os itens da colecão.

```
/* Implementação do TAD Colecao como um vetor */
#include "colecao.h"      /* inclui a especificacao do TAD */
typedef struct _colecao_ {
        int numItens;
        int maxItens;
        int *item;
} Colecao;
```

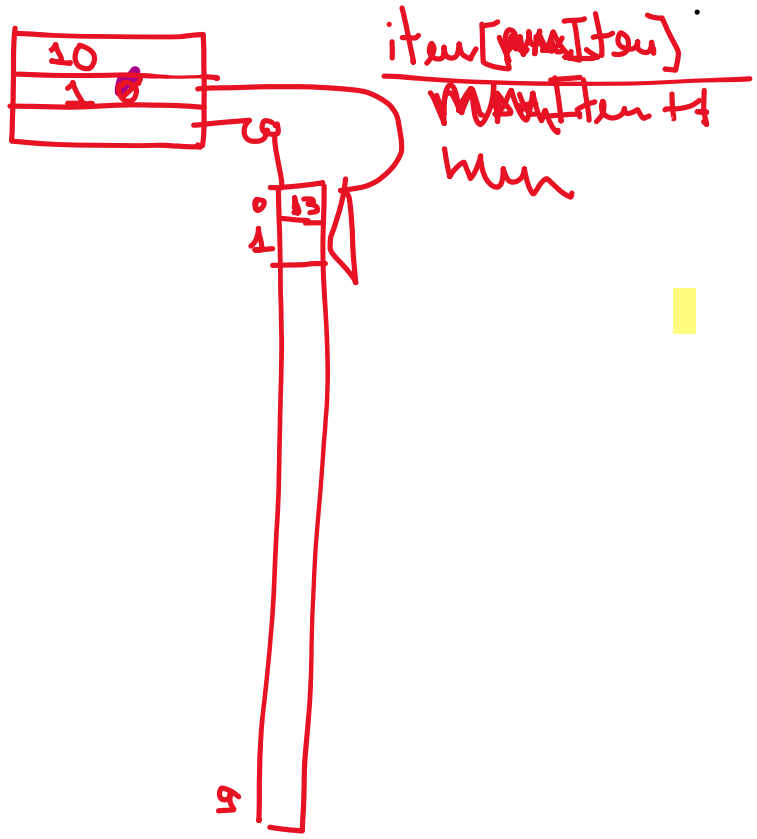
- Notas:
  - importação da especificação é para que o compilador verifique se estão compatíveis
  - *item* pode ser definido como *int item[]* ou *int *item*
  - implementação dos métodos é encontrada em *colecao.c*

Exemplo de Implementacao - Vetor

```
Colecao *colCreate(int maxItens)
{
    Colecao *c;
    if ( maxItens >  0 ) {
       c = (Colecao *)malloc(sizeof(Colecao) );
      if( c != NULL ) {
          c->item = (int *)malloc(sizeof(int)* maxItens);
       if ( c->item != NULL ) {
          c->maxItens = maxItens;
            c->numItens = 0;
            return c;
         }
        free(c);
      }
     }
     return NULL;
}/* fim de colCriar *
```

```
int colDestroy( Colecao *c )
{
    if ( c != NULL ) {
       if ( c->item != NULL ) {
         if ( c->numItens == 0 ) }
            free(c->itens)
            free(c);
            return TRUE;
         }
       }
     }
     return FALSE;
} /* fim de colDestruir */
```

```
int colInsert( Colecao *c, int item )
{
   if ( c != NULL ) {
      if ( c->item != NULL ) {
         if ( c->numItens  < c->maxItens){
           c->item[c->numItens]= item;
            c->numItens ++;
            return TRUE;
         }
      }
    }
    return FALSE;
}
```

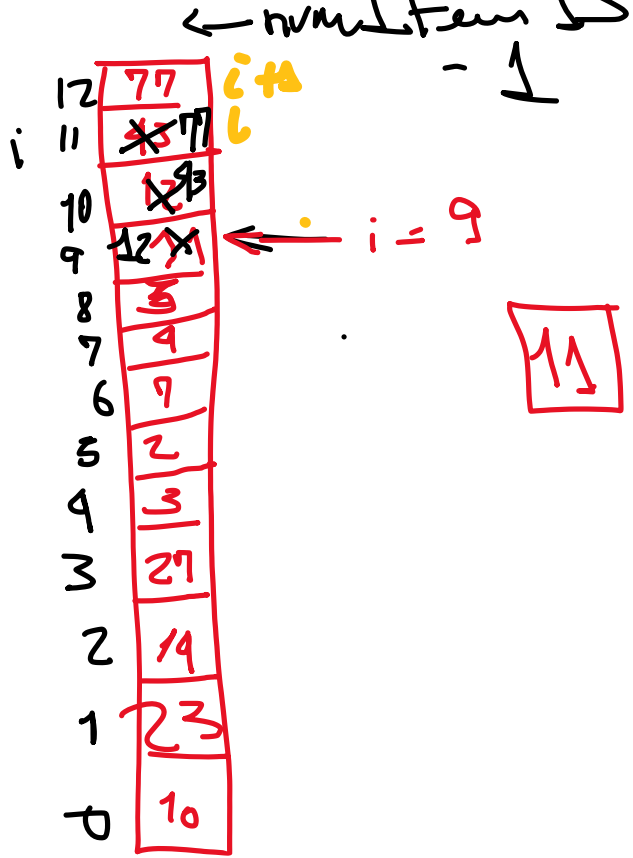


Exemplo de Implementacao - Vetor

```
int colRemove( Colecao *c, int item )
{
      int i; int elm;
      if( ( c != NULL ) {
        if ( c->numItens >0 )  {
          for(i=0; i<c->numItens; i++) {
              if ( item == c->itens[i] ) {
                  elm = c->itens[i] ;
                   while( i < c->numItens-1 ) {
                          c->itens[i] = c->itens[i+1];
                          i++;
                    }
                   c->numItens--;
                   return elm;
               } /* fi */
            }/* rof */
        }
      }
      return -1;
} /* fim de colRetirar */
```

```
int colQuery( Colecao *c, int key )
{
     int i;
     if (( c != NULL ) ) {
        if ( c->numItens >0 )  {
          for(i=0; i<c->numItens; i++) {
              if (c->itens[i] == key){
                  return c->items[i];
              }
           }
        }
      }
      return -1;
} /* fim de colBuscar */
```

23


numItems 13
- 1
i++
i
i = 9
11
12
11
10
9
8
7
6
5
4
3
2
1
0
77
93
12
3
4
7
2
3
27
14
23
10

TADS Genéricos

- Problema:
  - Necessário implementar um TAD Colecao para cada tipo de dados
- Solucao:
  - não especificar que tipo de dados será usado.
  - TAD de ponteiros para void.

TADs Genericos

- Especificação das operações

```
Colecao *colCreate( int maxItems );
int colInsert( Colecao *c, void * item );
void  *colRemove( Colecao *c, void *key );
void *colQuery( Colecao *c, void *key );
int colDestroy( Colecao *c);
```

- Reimplementar as funções

- Especificação

```
typedef struct _colecao_ {
        int numItens;
        int maxItens;
        int cur;
        void **item;
}Colecao;
```

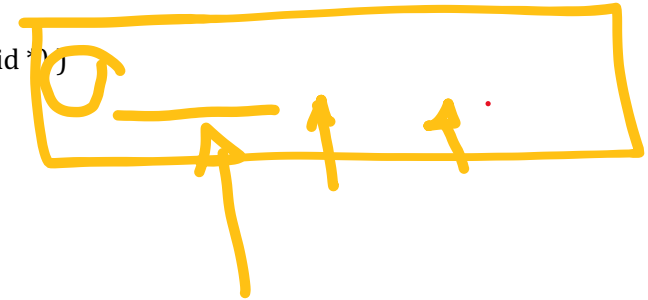
- Problema com consulta e remoção
  - necessário passar uma função como parâmetro.

```
Colecao *colCreate( int maxItens )
{
    Colecao *c;
    if (maxItens >0 ) {
        c = (Colecao *) malloc (sizeof(Colecao ));
        if ( c   !=  NULL) {
            c->item = (void **)malloc(sizeof(void *) * maxItens);
             if( c->item !=NULL) {
                 c->maxItens = maxItens;
                  c->numItens = 0;
                  c->cur = -1;
                  return c;
              }
          }
        }
        return NULL;
}
```

```
int colDestroy( Colecao *c)
{
     if ( c != NULL) {
        if( c->item != NULL) {
            if ( c->numItens == 0 ) {
                 free(c->item);
                  free(c);
                  return TRUE;
             }
          }
       }
       return FALSE;
}
```

```
int colInsert( Colecao *c, void * item )
{
   if ( c!= NULL) {
      if (c->item != NULL) {
         if ( c->numItens < c-> maxItens){
             c->item[c->numItens] = item;
              c->numItens++;
              return TRUE;
          }
       }
     }
     return FALSE;
}
```

```
void *colQuery( Colecao *c, void *key ,  int (*cmp)(void *,void *) )
{
    if ( c!= NULL) {
       if ( c->item != NULL) {
          if ( c->numItens > 0){
             for (i=0; i<vc->numItens; i++) {
                   // se item na posição i for identificado por key
                   if ( cmp(key, c->item[i]) == TRUE){
                      return c->item[i];
                   }
                }
             }
          }
       }
       return NULL;
}
```



```
int colRemove( Colecao *c, voi * key, int (*cmp)(void *, void *)  )
{
      int i; int elm;
      if( ( c != NULL ) {
        if ( c->numItens >0 )  {
           for(i=0; i<c->numItens; i++) {
                if ( cmp( key,  c->itens[i] == TRUE ) {
                   elm = c->itens[i] ;
                    while( i < c->numItens-1 ) {
                           c->itens[i] = c->itens[i+1];
                           i++;
                    }
                    c->numItens--;
                    return elm;
               } /* fi */
            }/* rof */
        }
      }
      return -1;
} /* fim de colRemove
 */
```

```
void *colGetFirst(Colecao *c)
{
     if ( c!= NULL) {
       if ( c->item != NULL) {
          c->cur = 0;
          return c->item[c->cur];
        }
      }
       return NULL;
}

void *colGetNext( Colecao *c)
{
     if ( c!= NULL) {
       if ( c->item != NULL) {
          c->cur++;
          if( c->cur < c->numItens) {
               return c->item[c->cur];
           }
        }
      }
       return NULL;
}
```

```
typedef struct _aluno_ {
        int id;
        char *nome;
} Aluno;
Int ComparaAluno( void *a, void *b)
{
       int *pa = (int *)a;
       int  *pb = ( Aluno *) b;
       if ( pb->id == *pa) {
           return TRUE;
       } else {
            return FALSE;
        }
}
void main ( void)
{
    Colecao *c;
    Aluno *a;
    int id
     c= colCreate(10);
     if ( c!= NULL) {
        for(i=0; i<4; i++) {
            a= ( Aluno *) malloc(sizeof(Aluno));
            if ( a!= NULL){
                a->id = i;
                scanf("%s", & (a->nome));
               colInsert(c, ( void* ) a);
            }
        }
        id = 2;
        a = ( Aluno *) colQuery(c, (void *) &id, ComparaAluno)
```

Prog.c

```
...
   a = (Aluno *) colGetFirst(c);
   while ( a != NULL ) {
        printf ("%d %s", a->id, a->nome);
        a = (Aluno *) colGetNext(c);
    }

}
```

Funções como Parâmetros - Exemplo

- Quicksort
  - void qsort(void *base, size_t n, size_t size, int (*compar)(void *, void *));

    base endereço de um vetor
    n número de elementos
    size tamanho do elemento
    compar função de comparação

- C permite passar uma função como parâmetro para uma outra função !!

Funções como tipos de dados - declaração

- Funções como tipos de dados
  - Declaração

    int (*compar)(void *, void *);
    - Parênteses em torno do * e do nome da função define que é um ponteiro para função
    - Diferente de:

      int *compar(void *, void *);

Função retorna um *int*

Possui dois argumentos void *

  - A função de comparação passada para quicksort retorna

    -1 *arg1 < *arg2

    0 *arg1 == *arg2

    +1 *arg1 > *arg2

Funções como tipo de dados

- Uso
  - Funções de bibliotecas que precisam de uma função com objetivo especial
    - Ordenação precisa do conceito de ordem
    - função compar fornece isto
      - Conceito de ordem pode ser complexo
        - eg nomes em uma lista telefônica
    - Busca também necessita de ordem

Funções como tipos de dados

- Generalizando TAD colecao
  - Usamos ponteiros para a struct que representa elemento da coleção
    - Permite que o ADT colecao armazene qualquer tipo de objetos
  - Precisamos assumir que existem duas funções externas (callback)
    - itemkey e itemcmp
    - Restringindo a uma coleção por programa
- Coloque a função de ordenação como atributo
  - Regra de ordenação é armazenada com os atributos da coleção
  - Quantas coleções eu quiser.

ADT Colecao Generica

- Redefina as funções que precisam identificar um element da coleção

```
void *colBuscar( Colecao *c, void *chave,
                        int (*compar)( const void *, const void * ) );
void  *colRetirar( Colecao *c, void *item,
                        int (*compar)( const void *,  const void * ));
```

Usando um ponteiro para uma função

- Use o atributo passado como o nome de uma função:

```
void *colBuscar( Colecao *c, void *chave,
                 int compar ( const void *, const void * ) );
{
    int k;
    for(k=0;k<c->numItens;k++) {
       if( compar(key, c->item[k]) == 0 ){
            return c->data[k];
       }
    }
    return NULL;
}
```

compar é um ponteiro para uma função - acesse o endereço nele para executar a função

… e forneça os argumentos , como se fosse o nome de uma função que vc ja esta acostumado a usar!

Funções como Tipo de Dados

- Códigos conduzidos por tabelas
  - eg menus
    - Cada entrada
      - String
      - Ação a ser realizada se a entrada for selecionada

```
struct menu {
        char *desc;
        int (*action)( void );
} file_menu[ ] = {
           { "Save",     save_function },
           { "Save as", save_as_function },
           …
        };
```

Ponteiro para uma função que não recebe argumentos

Strings que aparecem no menu

Funções a serem chamadas

Funções como tipos de dados

- Usando a tabela

```
struct menu {
    char *desc;
    int (*action)( void );
} file_menu[ ] = {
                    { "Save",  save_function },
                    { "Save as", save_as_function },
                    ...
                  };
void file_menu_callback()
{
  int k = item_selected( file_menu );
  file_menu[k].action();
}
```

Encontra o item a ser selecionado

Chama a função associada

Resumo

- Um TAD consiste de um novo tipo de dados juntamente com operações que manipulam esses dados
- O TAD é colocado em um arquivo .c separado de sua especificação que fica em um arquivo .h
- Qualquer programa pode usar o TAD
- Se a implementação for modificada os programas que utilizam o TAD não precisam ser alterados
- TADs genéricos são uma importante característica