Universidade Federal do Ma
Curso de Ciência da Computação

**Estrutura de Dados**
**Prof. Anselmo C. de Paiva**
**Depto de Informática**

*Tipo Abstrato de Dados*

## Tipo Abstrato de Dados
## Tipo Abstrato de Dados

- **Uma estrutura de dados e uma coleção de funções que operam sobre essa estrutura**
- **Uma nova forma de definir um tipo novo de dados juntamente com as operações que o manuseiam**
- **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**
    - **Destroir ® Destroi a coleção**



## Tipo Abstrato de Dados
## O TAD Coleção

- **Uma coleção pode ser implementada em uma linguagem de programação com:**
  - **uma declaração de tipo**

    *typedef struct _colecao_ * Colecao;*
  - **um conjunto de funções representando as operações**

    *Colecao colCriar( int maxItems, int itemSize );*
    *int colInserir( Colecao c, int item );*
    *int colRetirar( Colecao c, int item );*
    *int *colBuscar( Coleca c, int chave );*

  **Essas declarações ficam em um arquivo de cabeçalho (header) *colecao.h*, que deve ser incluído em todos os arquivos que utilizarem o TAD**
- **Até agora não sabemos como esta implementada a coleção, apenas sabemos sua especificação.**



## Tipo Abstrato de Dados
## 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;
/* ---------------------------------------------------
 Funcoesque implementam as operacoesdo
    TAD ColecaoInt
----------------------------------------------------- */
/*  Cria um novo TAD Colecao
  Pre-condicao: max_items > 0
  Pos-condicao: retorna um ponteiro para uma
    novo TAD Colecao vazio */
Colecao colCriar( int max_itens );
/* Adiciona um item na Collecao
  Pre-condition: (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 colInserir( 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 colRetirar ( Colecao c, void *item );
```



## Tipo Abstrato de Dados
## 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 colBuscar( Colecao c, void *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 colDestruir( Colecao c );


#endif /* __Colecao_INT_H */
```



## Tipo Abstrato de Dados
## Questão:

- **Suponha que alguém forneça a você um TAD Coleção implementado, mas vc pode ler apenas o arquivo de cabeçalho, e ele permite que vc ligue com o seu programa um arquivo com a implementação desse TAD. Vc pode escrever um programa que utiliza este TAD como um tipo de dado?**

  **Sim**
  - **Vc conhece o nome do novo tipo de dado, o suficiente para declarar variáveis ponteiros para Colecao**
  - **Vc também conhece os cabeçalhos e as especificações para cada operação**

## Tipo Abstrato de Dados
## Exemplo de Implementação - Vetor

- **Maneira mais simples de implementar uma colecao é 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;
 }Col;
```

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



## Tipo Abstrato de Dados
## Exemplo de Implementacao - Vetor

```
Colecao colCriar(int maxItens)
{
   Colecao c;
   if ( maxItens < 0 ) {   return NULL;   }
   c = (Colecao)calloc( 1, sizeof(Colecao) );
   if( c == NULL ) {  return NULL; }
   c->itens = (int
   *)calloc(maxItens,sizeof(int));
   if ( c->itens == NULL ) {
     free ( c );
    return NULL;
   }
   c->maxItens = maxItens;
   c->numItens = 0;
   return c;
} /* fim de colCriar *
```



## Tipo Abstrato de Dados
## Exemplo de Implementacao - Vetor

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



## Tipo Abstrato de Dados
## Exemplo de Implementacao - Vetor

```
int colInserir( Collection c, int item )
{
  if ( c == NULL || ( c->numItens < c->maxItens )) {
      return FALSE;
  }
  c->items[c->numItens] = item;
  c->numItens++;
} /* fim de colInserir */
```



## Tipo Abstrato de Dados
## Exemplo de Implementacao - Vetor

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



## Tipo Abstrato de Dados
## Exemplo de Implementacao - Vetor

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

## Tipo Abstrato de Dados
## TADS Genéricos

- **Problema:**
  - **Necessario implementar um TAD Colecao para cada tipo de dados**
- **Solucao:**
  - **reimplementar o TAD sem especificar que tipo de dados queremos colocar nele.**
  - **Usar um TAD de ponteiros para void.**



## Tipo Abstrato de Dados
## TADs Genericos

- ***Especificacao das operacoes***

```
Colecao colCriar( int maxItems );
int colInserir( Colecao c, void * item );
int colRetirar( Colecao c, void *item );
int *colBuscar( Coleca c, void *chave );
```

- ***Especificacao do vetor***

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

- ***Reimplementar as funcoes***
  - **Problema com consulta e remocao, necessario passar uma funcao como parametro.**



## Tipo Abstrato de Dados
## Fun'coes como Parametros - 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 !!**



## Tipo Abstrato de Dados
## 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 retornando um inteiro



## Tipo Abstrato de Dados
## Funções como tipo de dados

  - **Declaração**

```
int (*compar)( void *, void * );
```

**Função retorna um *int***

**Possui dois argumentos void ***

  - **Declara um ponteiro para uma função de nome *compar* que retorna *int* e recebe dois argumentos `void *`**



## Tipo Abstrato de Dados
## Funções como tipo de dados

  - **Declaração**

```
int (*compar)( void *, void * );
```

**Função retorna um *int***

**Possui dois argumentos void ***

- **A função de comparação passada para *quicksort* retorna**

| | | |
|---|---|---|
| **-1** | ***arg1 < *arg2** | **Interpretado como definindo ordem não magnitude** |
| **0** | ***arg1 == *arg2** | |
| **+1** | ***arg1 > *arg2** | |

## Tipo Abstrato de Dados
### 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**



## Tipo Abstrato de Dados
### Funções como tipos de dados

- **Generalizando nosso TAD colecao**
  - **Usamos ponteiros para a estrutura (struct) que representa o elemento da coleção**
    - **←Permite que o ADT colecao armazene qualquer tipo de objetos**
  - ***Mas* precisamos assumir que existem duas funções externas**
    - **`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.**



## Tipo Abstrato de Dados
### ADT Collection Generalizado

- **Redefina a função de criação da coleção**

```
Collection ConsCollection( int max_items,
      int (*compar)(const void *,const void * ) );
```

Assume que utilizamos um vetor - necessário modificar para listas, árvores, etc.

- **Adicione um elemento aos atributos**

```
struct t_collection {
    int max_items;
    int cnt;
    int (*compar)( const void *, const void * );
    void *data;
};
```



## Tipo Abstrato de Dados
### ADT Collection Generalizado

- **Na função de construção trate a função parâmetro como se fosse um ponteiro qualquer:**

```
Collection ConsCollection( int max_items,
      int (*compar)(const void *,const void * ) )
{
    Collection c;
    c = malloc( sizeof( struct collection ) );
    if ( c != NULL ) {
       c->max_items = max_items;
       c->compar = compar;
       ……
    }
    return c;
}
```

***Compar* é um ponteiro para uma função** - em geral o endereço do inicio do código da função

Copie o ponteiro (é um endereço) como outro tipo de dado qualquer



## Tipo Abstrato de Dados
### Usando um ponteiro para uma função

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

```
void *FindInCollection( void *key )
{
    int k;
    for(k=0;k<c->cnt;k++) {
       if( *c->compar( key, c->data[k] ) == 0 )
            return c->data[k];
    }
    return NULL;
}
```

**`c->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!**



## Tipo Abstrato de Dados
### 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 receb argumentos**

**Strings que aparecem no menu**

**Funções a serem chamadas**

## Tipo Abstrato de Dados
## 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



## Tipo Abstrato de Dados
## 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**