Programação com Linguagem C: Tipos Abstratos de Dados

Rômulo Silva de Oliveira
Departamento de Automação e Sistemas – DAS – UFSC

romulo@das.ufsc.br
http://www.das.ufsc.br/~romulo
Fevereiro/2011

Tipos Abstratos de Dados

- Tipo Abstrato de Dado – Introdução
- Tipo Abstrato de Dado – Lista
- Tipo Abstrato de Dado – Fila

Referências

- C Primer Plus, 5th Edition
  - Stephen Prata
- Livros e tutoriais sobre a linguagem C em geral

Tipos Abstratos de Dados

- **Tipo Abstrato de Dado – Introdução**
- Tipo Abstrato de Dado – Lista
- Tipo Abstrato de Dado – Fila

Introdução 1/3

- Em programação tenta-se casar o tipo de dado com o problema em questão
  - Quantidade de camisas como int
  - Preço médio das camisas como um float
- Por vezes vários dados são agregados
  - Struct para representar um aluno, com nome, matricula, nota
- Além da representação dos dados, existem as operações
  - Definir claramente o que pode ser feito com uma struct aluno

Introdução 2/3

- Um **Tipo Abstrato de Dado** é definido por **Atributos** e **Operações**
- Exemplo: *int*
  - Atributo é um valor inteiro
  - Operações são + - * / %
  - Trata-se da implementação de um conceito matemático mais abstrato
- Ao usar uma variável do tipo ***int***, estamos limitados a estas operações
- Como criar nossos próprios tipos de dados ?

Introdução 3/3

- 3 passos para a criação de um Tipo Abstrato de Dado
- (1) Descrição abstrata dos atributos e operações que podem ser realizadas sobre dados deste tipo
  - Este é o verdadeiro tipo abstrato de dado
- (2) Interface de programação que implementa o Tipo Abstrato de Dado
  - Indica como os atributos são armazenados
  - Indica como as operações são invocadas
  - Alguém que vai usar este Tipo Abstrado de Dado vai usar via esta interface
- (3) Código que implementa a interface
  - Quem usa o TAD não precisa conhecer este código

Tipos Abstratos de Dados

- Tipo Abstrato de Dado – Introdução
- **Tipo Abstrato de Dado – Lista**
- Tipo Abstrato de Dado – Fila

Exemplo: Lista de Filmes 1/5

- Aplicação lida com uma lista de filmes
- É necessário manipular uma lista de itens (neste caso são filmes)
- Cada item contém um nome de filme e uma nota de 0 a 10
- É necessário
  - Adicionar novos itens ao final da lista
  - Mostrar o conteúdo da lista
- Seria útil dispor de um tipo de dado **LISTA**, além do **int**, **char**, etc
- Como não existe este tipo nativo na linguagem C, vamos cria-lo

Exemplo: Lista de Filmes 2/5

- Quais seriam as operações úteis para uma lista ?
  - Inicializar uma lista vazia
  - Adicionar um item no final da lista
  - Determinar se a lista está vazia ou não
  - Determinar se a lista está está cheia ou não
  - Determinar quantos itens existem na lista
  - Visitar cada item da lista realizando uma operação
- As operações podem atender necessidades de várias aplicações
  - Inserir um item em qualquer posição da lista
  - Remover um item da lista
  - Recuperar um item da lista
  - Trocar um item da lista por algum outro item
  - Procurar por um item específico na lista

Exemplo: Lista de Filmes 3/5

- **Descrição de um Tipo Abstrato de Dado**
- Nome: Lista simples
- Atributos: Contém uma sequência de itens
- Invariante: (alguma propriedade que é sempre verdadeira)
- Operações:
  - Inicializa lista vazia
  - Determina se a lista está vazia
  - Determina se a lista está cheia
  - Determina o número de itens da lista
  - Adiciona um item ao fim da lista
  - Percorre a lista processando cada item da lista
  - Esvazia a lista

Exemplo: Lista de Filmes 4/5

- Agora precisamos de uma interface de programação para a lista
  - Na linguagem de programação C
- Interface possui duas partes:
  - Descreve como os dados serão representados
  - Descreve as funções que implementam as operações
- A interface deve ser o mais fiél possível ao Tipo Abstrato de Dado
- Em uma linguagem como C, fatalmente decisões relativas à implementação vão impactar a interface de programação
  - Difícil fazer a interface sem pensar na implementação
  - Fácil usar a interface sem pensar na implementação

Exemplo: Lista de Filmes 5/5

- A interface de programação será especializada para uma lista de filmes
- Item pode ser redefinido para listas de outras coisas
- Lista será implementada como uma lista encadeada
- A interface está descrita em um arquivo list.h

list.h 1/8

```
/* list.h -- header file for a simple list type */
#ifndef LIST_H_
#define LIST_H_
#include <stdbool.h>    /* C99 feature */

/* program-specific declarations */

. . .

/* general type definitions */

. . .

/* function prototypes */

. . .

#endif
```


list.h 2/8

```
/* list.h -- header file for a simple list type */
#ifndef LIST_H_
#define LIST_H_
#include <stdbool.h>    /* C99 feature */

/* program-specific declarations */

#define TSIZE     45   /* size of array to hold title  */

struct film
{
   char title[TSIZE];
   int rating;
};


/* general type definitions */
. . .
/* function prototypes */
. . .
#endif
```


list.h 3/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .

/* general type definitions */

typedef struct film Item;

typedef struct node
{
   Item item;
   struct node * next;
} Node;

typedef Node * List;

/* function prototypes */
. . .
#endif
```


list.h 4/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .
/* general type definitions */
. . .
/* function prototypes */

/* operation:       initialize a list*/
/* preconditions:   plist points to a list      */
/* postconditions:  the list is initialized to empty      */

void InitializeList(List * plist);

/* operation:           determine if list is empty          */
/*                      plist points to an initialized list */
/* postconditions:      function returns True if list is empty        */
/*                      and returns False otherwise         */

bool ListIsEmpty(const List *plist);

. . .
#endif
```


list.h 5/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .
/* general type definitions */
. . .
/* function prototypes */
. . .

/* operation:           determine if list is full                */
/*                      plist points to an initialized list      */
/* postconditions:      function returns True if list is full    */
/*                      and returns False otherwise              */
bool ListIsFull(const List *plist);

/* operation:       determine number of items in list            */
/*                plist points to an initialized list            */
/* postconditions:  function returns number of items in list     */
unsigned int ListItemCount(const List *plist);

. . .
#endif
```

list.h 6/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .
/* general type definitions */
. . .
/* function prototypes */
. . .

/* operation:       add item to end of list                */
/* preconditions:   item is an item to be added to list    */
/*                  plist points to an initialized list    */
/* postconditions:  if possible, function adds item to end */
/*                  of list and returns True; otherwise the*/
/*                  function returns False                 */

bool AddItem(Item item, List * plist);

. . .
#endif
```


list.h 7/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .
/* general type definitions */
. . .
/* function prototypes */
. . .

/* operation:  apply a function to each item in list      */
/*             plist points to an initialized list        */
/*             pfun points to a function that takes an    */
/*             Item argument and has no return value      */
/* postcondition:  the function pointed to by pfun is     */
/*                 executed once for each item in the list*/

void Traverse (const List *plist, void (* pfun)(Item item) );

. . .
#endif
```


list.h 8/8

```
/* list.h -- header file for a simple list type */
. . .
/* program-specific declarations */
. . .
/* general type definitions */
. . .
/* function prototypes */
. . .

/* operation:          free allocated memory, if any      */
/*                     plist points to an initialized list*/
/* postconditions:     any memory allocated for the list is freed */
/*                     and the list is set to empty       */

void EmptyTheList(List * plist);

#endif
```


Exemplo: Lista de Filmes

- A interface deve poder ser usada sem o conhecimento de mais nada a respeito da implementação da lista
- A seguir alguns exemplos de uso da interface

films3.c 1/6

```
/* films3.c -- using an ADT-style linked list */
/* compile with list.c                        */
#include <stdio.h>
#include <stdlib.h>    /* prototype for exit() */

#include "list.h"      /* defines List, Item   */
```


films3.c 2/6

```
int main(void)
{
   List movies;
   Item temp;


/* initialize */
   InitializeList(&movies);

   if (ListIsFull(&movies))
   {
      fprintf(stderr,"No memory available! Bye!\n");
      exit(1);
   }
```

films3.c 3/6

```
/* gather and store */
    . . .
    gets(temp.title);
    . . .
    scanf("%d", &temp.rating);
    . . .

    if (AddItem(temp, &movies)==false)
    {
        fprintf(stderr,"Problem allocating memory\n");          break;
    }

    if (ListIsFull(&movies))
    {
        puts("The list is now full.");          break;
    }
```


films3.c 4/6

```
/* display        */
   if (ListIsEmpty(&movies))
      printf("No data entered. ");
   else
   {
      printf ("Here is the movie list:\n");
      Traverse(&movies, showmovies);
   }
   printf("You entered %d movies.\n", ListItemCount(&movies));
```


films3.c 5/6

```
void showmovies(Item item)
{
   printf("Movie: %s  Rating: %d\n", item.title,
         item.rating);
}
```


films3.c 6/6

```
/* clean up        */

   EmptyTheList(&movies);
   printf("Bye!\n");

   return 0;
}
```


Exemplo: Lista de Filmes

- Falta implementar as funções do Tipo Abstrato de Dados
- Programa composto por 3 arquivos:
- **list.h**
  - Contem a interface do Tipo Abstrato de Dados “Lista de filmes”
- **list.c**
  - Contem a interface do Tipo Abstrato de Dados “Lista de filmes”
- **films3.c**
  - Contem a aplicação que usa o Tipo Abstrato de Dados “Lista de filmes”

list.c 1/9

```
/* list.c -- functions supporting list operations */
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

/* local function prototype */
static void CopyToNode(Item item, Node * pnode);
```

list.c 2/9

```
/* set the list to empty */
void InitializeList(List * plist)
{
    * plist = NULL;
}
```


list.c 3/9

```
/* returns true if list is empty */
bool ListIsEmpty(const List * plist)
{
    if (*plist == NULL)
        return true;
    else
        return false;
}
```


list.c 4/9

```
/* returns true if list is full */
bool ListIsFull(const List * plist)
{
    Node * pt;
    bool full;

    pt = (Node *) malloc(sizeof(Node));
    if (pt == NULL)
        full = true;
    else
        full = false;
    free(pt);

    return full;
}
```


list.c 5/9

```
/* returns number of nodes */
unsigned int ListItemCount(const List * plist)
{
    unsigned int count = 0;
    Node * pnode = *plist;    /* set to start of list */

    while (pnode != NULL)
    {
        ++count;
        pnode = pnode->next;  /* set to next node     */
    }

    return count;
}
```


list.c 6/9

```
/* creates node to hold item and adds it to the end of */
/* the list pointed to by plist (slow implementation)  */
bool AddItem(Item item, List * plist)
{
    Node * pnew;
    Node * scan = *plist;

    pnew = (Node *) malloc(sizeof(Node));
    if (pnew == NULL)              return false;    /* quit function on failure  */

    CopyToNode(item, pnew);
    pnew->next = NULL;
    if (scan == NULL)              /* empty list, so place */
            *plist = pnew;         /* pnew at head of list */
    else
    {
            while (scan->next != NULL)
                    scan = scan->next;     /* find end of list    */
            scan->next = pnew;             /* add pnew to end     */
    }
    return true;
}
```


list.c 7/9

```
/* visit each node and execute function pointed to by pfun */
void Traverse (const List * plist, void (* pfun)(Item item) )
{
    Node * pnode = *plist;    /* set to start of list   */

    while (pnode != NULL)
    {
        (*pfun)(pnode->item); /* apply function to item */
        pnode = pnode->next;  /* advance to next item   */
    }
}
```

list.c 8/9

```
/* free memory allocated by malloc() */
/* set list pointer to NULL          */
void EmptyTheList(List * plist)
{
   Node * psave;

   while (*plist != NULL)
   {
      psave = (*plist)->next; /* save address of next node */
      free(*plist);          /* free current node         */
      *plist = psave;        /* advance to next node      */
   }
}
```


list.c 9/9

```
/* local function definition  */
/* copies an item into a node */
static void CopyToNode(Item item, Node * pnode)
{
   pnode->item = item;  /* structure copy */
}
```


Comentários sobre list.c

- Muitas mudanças são possíveis na implementação de list.c sem alterar os programas que usam este Tipo Abstrato de Dado
  - Colocar um pointer para o fim da lista
  - Implementar a lista como um array
  - Implementar a lista usando encadeamento duplo
- Maior modularidade
- Maior reutilização de código
- Melhor legibilidade da aplicação e do módulo
- Cuidado com:
  - Efeitos colaterais dentro da implementação
  - Suposições implícitas

Tipos Abstratos de Dados

- Tipo Abstrato de Dado – Introdução
- Tipo Abstrato de Dado – Lista
- **Tipo Abstrato de Dado – Fila**

Exemplo: Fila Simples

- **Descrição de um Tipo Abstrato de Dado**
- Nome: Fila simples
- Atributos: Contém uma sequência ordenada de itens
- Operações:
  - Inicializa uma fila vazia
  - Determina se a fila está vazia
  - Determina se a fila está cheia
  - Determina o número de elementos na fila
  - Adiciona um elemento no fim da fila
  - Remove e recupera um elemento do início da fila
  - Esvazia a fila

queue.h 1/10

```
/* queue.h -- interface for a queue */

#ifndef _QUEUE_H_
#define _QUEUE_H_
#include <stdbool.h>
```

queue.h 2/10

```
/* INSERT ITEM TYPE HERE */
/* FOR EXAMPLE, */
/* use the following for use_q.c */
/* typedef int Item; */
/* OR typedef struct item {int gumption; int charisma;} Item; */
/* use the following for mall.c */

typedef struct item
{
    long arrive;          /* the time when a customer joins the queue   */
    int processtime;      /* the number of consultation minutes desired */
} Item;
```


queue.h 3/10

```
#define MAXQUEUE 10

typedef struct node
{
    Item item;
    struct node * next;
} Node;

typedef struct queue
{
    Node * front;  /* pointer to front of queue  */
    Node * rear;   /* pointer to rear of queue   */
    int items;     /* number of items in queue   */
} Queue;
```


queue.h 4/10

```
/* operation:        initialize the queue */
/* precondition:     pq points to a queue */
/* postcondition:   queue is initialized to being empty */

void InitializeQueue(Queue * pq);
```


queue.h 5/10

```
/* operation:       check if queue is full */
/* precondition:    pq points to previously initialized queue */
/* postcondition:   returns True if queue is full, else False */

bool QueueIsFull(const Queue * pq);
```


queue.h 6/10

```
/* operation:       check if queue is empty */
/* precondition:    pq points to previously initialized queue */
/* postcondition:   returns True if queue is empty, else False */

bool QueueIsEmpty(const Queue *pq);
```


queue.h 7/10

```
/* operation:       determine number of items in queue */
/* precondition:    pq points to previously initialized queue */
/* postcondition:   returns number of items in queue */

int QueueItemCount(const Queue * pq);
```

queue.h 8/10

```
/* operation:    add item to rear of queue */
/* precondition:     pq points to previously initialized queue */
/*                   item is to be placed at rear of queue */
/* postcondition:    if queue is not empty, item is placed at */
/*                rear of queue and function returns */
/*                True; otherwise, queue is unchanged and */
/*                function returns False */

bool EnQueue(Item item, Queue * pq);
```


queue.h 9/10

```
/* operation:       remove item from front of queue          */
/* precondition:    pq points to previously initialized queue  */
/* postcondition:   if queue is not empty, item at head of     */
/*                queue is copied to *pitem and deleted from */
/*                queue, and function returns True; if the   */
/*                operation empties the queue, the queue is  */
/*                reset to empty. If the queue is empty to   */
/*                begin with, queue is unchanged and the     */
/*                function returns False                     */

bool DeQueue(Item *pitem, Queue * pq);
```


queue.h 10/10

```
/* operation:       empty the queue                          */
/* precondition:    pq points to previously initialized queue  */
/* postconditions:  the queue is empty                       */

void EmptyTheQueue(Queue * pq);



#endif
```


Exemplo: Fila Simples

- Lista encadeada não é a única forma de implementar uma fila
- Pode-se usar array
- Existem vantagens e desvantagens
- Com array:
  - É necessário pré-determinar o tamanho máximo da fila
  - A memória é alocada mesmo que a fila seja pequena
  - Mas agora pode-se economizar o pointer "next"
  - Cópias de dados são evitadas usando-se uma estrutura circular

- A partir da interface é possível desenvolver um programa (**test-driver**) que faz chamadas às rotinas do Tipo Abstrato de Dado para testa-lo
- O **test-driver** pode (deve) ser escrito por uma pessoa diferente
- O **test-driver** pode (deve) ser escrito antes da própria implementação do Tipo Abstrato de Dado

queue.c 1/9

```
/* queue.c -- the Queue type implementation*/
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

/* local functions */
static void CopyToNode(Item item, Node * pn);
static void CopyToItem(Node * pn, Item * pi);
```


queue.c 2/9

```
void InitializeQueue(Queue * pq)
{
   pq->front = pq->rear = NULL;
   pq->items = 0;
}
```

queue.c 3/9

```
bool QueueIsFull(const Queue * pq)
{
   return pq->items == MAXQUEUE;
}
```


queue.c 4/9

```
bool QueueIsEmpty(const Queue * pq)
{
   return pq->items == 0;
}
```


queue.c 5/9

```
int QueueItemCount(const Queue * pq)
{
   return pq->items;
}
```


queue.c 6/9

```
bool EnQueue(Item item, Queue * pq)
{
    Node * pnew;

    if (QueueIsFull(pq))        return false;
    pnew = (Node *) malloc( sizeof(Node));
    if (pnew == NULL)
    {
    fprintf(stderr,"Unable to allocate memory!\n");          exit(1);
    }
    CopyToNode(item, pnew);
    pnew->next = NULL;
    if (QueueIsEmpty(pq))
    pq->front = pnew;           /* item goes to front     */
    else
    pq->rear->next = pnew;    /* link at end of queue   */
    pq->rear = pnew;            /* record location of end */
    pq->items++;    /* one more item in queue */
    return true;
}
```


queue.c 7/9

```
bool DeQueue(Item * pitem, Queue * pq)
{
    Node * pt;

    if (QueueIsEmpty(pq))
            return false;
    CopyToItem(pq->front, pitem);
    pt = pq->front;
    pq->front = pq->front->next;
    free(pt);
    pq->items--;
    if (pq->items == 0)
            pq->rear = NULL;

    return true;
}
```


queue.c 8/9

```
/* empty the queue                */
void EmptyTheQueue(Queue * pq)
{
   Item dummy;

   while (!QueueIsEmpty(pq))
         DeQueue(&dummy, pq);
}
```

queue.c 9/9

```
/* Local functions */

static void CopyToNode(Item item, Node * pn)
{
    pn->item = item;
}

static void CopyToItem(Node * pn, Item * pi)
{
    *pi = pn->item;
}
```


Comentários sobre queue.c

- Diferentes aplicações podem usar queue.c
- Muitas mudanças são possíveis na implementação de queue.c
  - Algoritmos novos, mais eficientes
  - Ampliação da interface
- Maior modularidade
- Maior reutilização de código
- Melhor legibilidade (depuração/manutenção mais fácil)
- Como evitar que código desnecessário acabe no arquivo executável:
  Criar bibliotecas a partir dos TADs e não apenas linkar com o resto

Tipos Abstratos de Dados – Sumário

- Tipo Abstrato de Dado – Introdução
- Tipo Abstrato de Dado – Lista
- Tipo Abstrato de Dado – Fila