Tipos Abstratos de Dados

1. Tipos de Dados

Em uma linguagem de programação, os dados armazenados pelas constantes e variáveis são classificados de acordo com o tipo de informação que contém. Um tipo de dados caracteriza o conjunto de valores que pode ser assumido por uma variável ou gerado por uma função ou mesmo uma expressão.

Um tipo de dados **elementar** (também chamado de **simples** ou **primitivo**) é caracterizado or um conjunto (domínio) de valores, como por exemplo os valores definidos pelos tipos inteiro, real, caracter ou string.

Cada tipo de dados também possui um conjunto de operações que podem ser realizadas. Por exemplo, os tipos inteiro e real suportam as operações aritméticas (+, -, *, /, **, raiz quadrada, etc), enquanto o tipo string suporta as operações de concatenação e segmentação, etc.

2. Tipo Abstrato de Dados (TAD)

Um tipo abstrato de dados, (TAD) é uma especificação de um conjunto de dados (frequentemente heterogêneos) e operações que podem ser executadas sobre esses dados.

Por exemplo, podemos definir um tipo abstrato de dados estudante, para representar as informações necessárias para uma “entidade” estudante, contendo o nome, a idade e a matrícula de um estudante, e as operações de duplicar a informação, validar a matrícula, verificar a idade, etc.

Na prática, o TAD é implementado usando-se um tipo composto (struct/record – estrutura/registro) com os valores pertensentes ao TAD (nome, idade, matrícula). E por funções que operam essa estrutura.

Exemplo:

```
struct Estudante
{
     char nome[50];
     int idade;
     int matrícula;
}

int maiorDeIdade (struct Estutante e);
int matriculaValida (struct Estudante e);
```

No exemplo acima foi definida a estrutura Estudante, com os três campos mencionados, e duas funções com retorno booleano, maiorDeIdade, que retorna TRUE se a idade do estudante e for maior ou igual a 18, e matriculaValida, que verifica se o campo matrícula do estudante e é válida. Nesse exemplo somente os protótipos das funções estão apresentadas.

3. Lista

Um tipo abstrato de dados comum é a lista. Na prática, usamos a representação de lista no nosso dia a dia, para armazenar informações das mais variadas. Num modo bem simples, pode ser armazenada em uma folha de papel, como uma relação de itens. As operações usuais sobre uma lista em geral incluem:
- Criação da lista;
- Inclusão de um elemento na lista;
- Exclusão de um elemento da lista;
- Contar o número de elementos na lista;
- Exibir todos os elementos da lista;

Um mesmo tipo abstrato de dados pode ser implementado de mais de uma maneira diferente, e cada forma de implementação pode apresentar vantagens e desvantagens.

Por exemplo, a implementação de uma lista pode ser feita através de um vetor de strings, onde cada string contém um elemento da lista. Isso é útil, por exemplo, para armazenar uma lista de compras, onde cada item da lista é armazenado em um string do vetor. Essa forma é conhecida como lista com implementação estática.

Nesse caso específico, como todos os itens são representados somente com um string, não há necessidade de usar um tipo struct, e o tipo Lista pode ser definido como:

```
#define MaxItens 10
#define TamItem 20

typedef char tpLista[MaxItens][TamItem];
```

Nesse caso estamos definindo um tipo tpLista como um vetor de 10 strings, cada um deles com 20 caracteres.

E a alocação do espaço é definida quando da criação da variável Lista com base no tipo tpLista.

```
tpLista L1;
```

Assim, o espaço pode ser representado como na tabela abaixo, com 10 linhas de 20 colunas.

L1

0	c	a	r	n	e														
1	a	r	r	o	z														
2	a	l	f	a	c	e													
3																			
4																			
5																			
6																			
7																			
8																			
9																			

Na lista `L1`, do tipo `tpLista`, representada acima, foram inseridos os itens **carne**, **arroz** e **alface**.

Para iniciar a operação de uma lista, é preciso criar uma lista vazia.
Isso pode ser feito zerando todos os strings da lista.
Uma forma simples de fazer isso é pela função criaLista abaixo.

```
void criaLista(tpLista l)
{
      int i;

      for (i=0;i<MaxItens;i++)
         l[i][0] = '\0';
}
```

Para imprimir o conteúdo de uma lista, basta imprimir todos os strings da lista
Isso pode ser feito com a função imprimeLista abaixo.

```
void imprimeLista(tpLista l)
{
      int i;

      printf("\nItens da lista\n");
      for(i=0; i<MaxItens && strlen(l[i])>0; i++)
            printf("\n%s",l[i]);
      printf("\n");
}
```

A operação que conta o número de elementos da lista é parecida com a impressão, mas ao invés de imprimir, precisa incrementar um contador, e retornar esse contador. Um exemplo de implementação segue abaixo.

```
int contaLista(tpLista l)
{
    int i;

    // Percorre a lista, até chegar ao final do espaço
    //                   ou encontrar um elemento vazio (tamanho = 0)
    for (i = 0; i<MaxItens && strlen(l[i])>0; i++);

    // Retorna a posição do elemento vazio, ou MaxItens, que equivale
    //   ao número de elementos na lista.
    return(i);
}
```

Para implementar a operação de inserir um elemento na lista pode-se fazer:

```
void insereLista(tpLista l, char *item)
{
    int i;
    // Laço que percorre a lista até encontrar o primeiro elemento vazio
    // (com tamanho = 0), ou ultrapassar o espaço alocado para a lista
    for(i=0; i<MaxItens && strlen(l[i])>0; i++);

    // Testa se ainda tem espaço na lista
    if (i<MaxItens)
    {
        strcpy(l[i],item); // copia o item para a posição vazia da lista
        printf("\nItem (%s) inserido com sucesso",item);
    }
    else // Não tem espaço na lista
        printf("\nLista cheia, não pode inserir");
}
```

A implementação da operação para retirar um elemento da lista precisa localizar o elemento a ser removido, e deslocar os elementos seguintes uma casa para trás, de modo a manter os itens sem espaços entre eles. Isso pode ser feito por um código aos moldes do código abaixo.

```
/ Operação que retira o elemento "item" da lista "l"
void retiraLista(tpLista l, char *item)
{
    int i;

    // Procura o elemento "item" na lista.
    // Encerra a busca se chegar ao final do espaço, ou
    //                 se encontrar um elemento vazio, ou
    //                 se encontrar o elemento "item"
    for(i=0; i<MaxItens && strlen(l[i])>0 && (strcmp(l[i],item) != 0); i++);

    // Pode ter saído do laço por três razões. Descobre qual
    if (i<MaxItens && (strcmp(l[i],item) == 0))
    {  // Encontrou o elemento buscado
       printf("\nItem (%s) encontrado, removendo",item);
       if (i < (MaxItens-1))
       { // Tem outros elementos depois dele, puxa todos para cima
             for (;i<(MaxItens-1) && strlen(l[i])>0;i++)
                 strcpy(l[i],l[i+1]);
       }
       // Zera o último elemento do espaço da lista
       l[MaxItens-1][0] = '\0';
    }
    else // Não encontrou o elemento buscado
       printf("\nItem (%s) não encontrado",item);
}
```

Ainda falta implementar o programa principal, que fará a interface com o usuário, oferecendo as opções de operações disponíveis e chamando cada uma das funções que o usuário escolher.