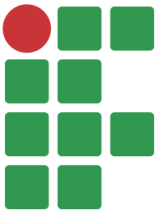

INSTITUTO FEDERAL DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA DE ESTRUTURAS DE DADOS
PROF. VICTOR ANDRÉ PINHO DE OLIVEIRA

Estruturas de Dados

Aula 4 - ED - Listas Sequenciais

Introdução

Olá,

Na semana anterior, nós concluímos o estudo a respeito da linguagem C. Foram algumas semanas de imersão na linguagem. Agora, com a bagagem obtida, estamos prontos para avançar ao próximo nível.

Este material inaugura uma nova etapa. A partir desta semana, estaremos voltando nossa atenção para a implementação das **Estruturas de Dados**. Começaremos com **Listas Lineares Sequenciais**, e veremos como implementá-la de maneira não ordenada e ordenada. Para tanto, certifique-se de ter compreendido muito bem sobre funções, arrays e structs.

Preparado?

Listas Lineares Sequenciais

Uma **Lista** é uma estrutura de dados que armazena elementos de mesmo tipo. Chamamos uma lista de **Linear** porque cada elemento tem apenas um antecessor e um sucessor[1]. Uma lista Linear pode ser Sequencial ou Encadeada. Uma Lista Linear é **Sequencial** quando os elementos estão dispostos sequencialmente na memória. Uma Lista Linear é **Encadeada** quando os elementos não estão em sequência na memória, mas cada elemento possui um ponteiro para o próximo elemento da Lista.

Na aula de hoje, voltaremos nossa atenção para **Listas Lineares Sequenciais**.

Listas Lineares Sequenciais não Ordenadas

As Listas podem ser Ordenadas e não Ordenadas. Chamamos uma Lista Linear de **Ordenada** quando os elementos são inseridos de maneira ordenada. Por outro lado, chamamos uma Lista de **não Ordenada** quando não há preocupação quanto a inserção ordenada dos elementos.

[1] À exceção, claro, do primeiro e do último elemento da lista.

Começaremos vendo as Listas não Ordenadas. Atente que existem várias formas de implementar uma Lista Linear Sequencial. Aqui, apresentarei uma variante. Mas exploraremos outras possibilidades nos exercícios.

Segue abaixo o código.

```
#include <stdio.h>
#include <stdlib.h>

const unsigned MAX = 10;
int l[MAX], pos = 0;

void inserir(int elemento);
int buscar(int elemento);
void remover(int elemento);

int obter(int indice);
int tamanho();
void imprimir();
void apagar();

int main(void) {
  return 0;
}

void inserir(int elemento){
  if (pos < MAX)
      l[pos++] = elemento;
  else {
      printf("Não foi possível inserir %d. Lista
cheia.\n",elemento);
  }
}

int buscar(int elemento){
  for (int i = 0 ; i < pos ; i++){
      if (l[i] == elemento)
      return i;
  }
  return -1;
}

void remover(int elemento){
```

```
  int p = buscar(elemento);

  if (p == -1)
      return;

  for (int i = p ; i < pos -1; i++)
      l[i] = l[i+1];
  pos--;
}

int obter(int indice){
  if (indice < 0 || indice >= pos) {
      printf("Indice %d fora dos limites da Lista.\n", indice);
      exit(1);
  }
  return l[indice];
}

int tamanho(){
  return pos;
}

void imprimir(){
  for (int i = 0 ; i < pos ; i++)
      printf("%d ",l[i]);
  printf("\n");
}

void apagar(){
  pos = 0;
}
```

O código acima mostra uma forma de implementar uma Lista de int.

Bem, para implementarmos uma Lista Linear Sequencial vamos precisar de:

- uma constante que armazena o tamanho máximo da Lista: MAX
- uma array de tamanho MAX
- uma variável que armazena o índice imediatamente posterior ao último elemento da Lista: pos
- de algumas funções que controlam a manipulação da estrutura:
    - inserir: permite inserir um elemento no fim da lista

    - remover: permite remover um determinado elemento da lista
    - buscar: retorna o índice de um elemento buscado, caso encontrado

Além disso, resolvi acrescentar umas funções de apoio:

- Funções de apoio:
    - obter: retorna o elemento do índice
    - tamanho: retorna a quantidade de elementos
    - imprimir: imprime na tela a lista
    - apagar: zera a lista

Como estamos implementando uma Lista de int, tanto o array quanto as funções foram implementados esperando receber tal tipo.

Adicionalmente, note que, conforme podemos ver no código acima, você pode declarar todas as suas funções (assinaturas) acima da main e deixar para implementá-las em qualquer outro lugar abaixo no arquivo. Isso te dá mais liberdade em algumas situações. Por hora, estou apenas deixando como uma boa dica.

Por estarmos implementando uma Lista, que é um tipo de Estrutura de Dados, não podemos manipular o array diretamente, mas somente por meio das funções. As funções ditam as regras de acesso à Estrutura. Assim, basta entendermos como cada função funciona.

Função inserir

A função inserir tem o objetivo de inserir um elemento no final da Lista.

```
void inserir(int elemento){
  if (pos < MAX)
      l[pos++] = elemento;
  else {
      printf("Não foi possível inserir %d. Lista
cheia.\n",elemento);
  }
}
```

A função inserir não retorna nada e recebe o elemento a ser introduzido na Lista. Primeiro verificamos se há espaço, pois nossa lista tem tamanho máximo. A variável pos sempre vai armazenar o próximo índice. Assim, se o próximo índice (pos) é igual a MAX é porque a Lista está cheia.

Caso haja espaço, inserimos o elemento ao fim da lista. E aqui, utilizamos um grande recurso da linguagem C: o pós-incremento. Fizemos l[pos++] = elemento. Isso significa que estamos armazenando em l[pos] o elemento. E, após e somente após a atribuição, pos é incrementado em 1. Isso nos permite fazer em uma linha de código, o que seria feito em duas. A variável pos é incrementada em 1 para conter o próximo índice.

Função buscar

A função buscar tem o objetivo de buscar um elemento na Lista e retornar sua posição - seu índice -, caso ele exista na Lista.

```
int buscar(int elemento){
  for (int i = 0 ; i < pos ; i++){
      if (l[i] == elemento)
          return i;
  }
  return -1;
}
```

A função retorna um índice (int) e recebe o elemento a ser buscado. Nesse caso, varremos o array de 0 até pos -1 comparando cada elemento do array com o elemento passado. Na primeira ocorrência, retornamos o índice do elemento. Note que, caso exista mais de uma ocorrência do elemento, somente a primeira é retornada. Caso o elemento não seja encontrado, a função retorna o índice -1.

Função remover

A função remover tem por objetivo remover um determinado elemento da Lista.

```
void remover(int elemento){
  int p = buscar(elemento);

  if (p == -1)
      return;

  for (int i = p ; i < pos -1; i++)
      l[i] = l[i+1];
  pos--;
}
```

A função não retorna nada e recebe o elemento a ser removido. A função começa invocando a função buscar, para obter o índice do elemento a ser removido. O retorno é armazenado em p. Se p for igual a -1 é porque tal elemento não existe na Lista, logo não há o que remover.

A grande sacada da função remover está no loop for. Basicamente, como a ideia é remover da Lista o elemento do índice p em questão, estamos trazendo, puxando, arrastando, todo o restante da Lista para cima do elemento removido, “espremendo” o elemento removido. O for corre a partir de p (o índice do elemento a ser removido) e roda enquanto i < pos-1 atribuindo ao elemento atual (l[i]) o próximo elemento (l[i+1]).

Claro que, como removemos um elemento, não podemos esquecer de decrementar pos (pos--).

Funções auxiliares

E agora, veremos as funções auxiliares.

Função obter

A função obter recebe um índice e retorna o elemento, caso ele esteja presente na Lista.

```
int obter(int indice){
  if (indice < 0 || indice >= pos) {
      printf("Indice %d fora dos limites da Lista.\n", indice);
      exit(1);
  }
  return l[indice];
}
```

Talvez você esteja se perguntando sobre a necessidade desta função, questionando se não seria melhor acessar o array direto e pronto. NÃO. Não seria melhor acessar o array diretamente porque perderíamos o controle sobre o acesso ao array. Note que a função obter verifica, antes de retornar o elemento, se o índice passado como argumento está dentro dos limites da Lista. Mais uma vez, é importante lembrar que, por estarmos implementando uma Estrutura de Dados, o acesso às variáveis da Estrutura deve se dar sempre por meio das funções, pois são elas que controlam o acesso e a manipulação das variáveis, mantendo a integridade da Estrutura.

Observe que caso um índice fora da Lista tente ser acessado, a função encerra o programa, pois não podemos retornar um valor que não esteja presente na Lista.

Função tamanho

Retorna o tamanho da Lista.

```
int tamanho(){
  return pos;
}
```

O valor da variável pos sempre vai coincidir com o tamanho da Lista. Quando a Lista está vazia, pos é igual a zero. Quando inserimos o primeiro elemento, pos se torna 1 etc. Mais uma vez, acessamos uma variável da Estrutura por meio de uma função.

Função imprimir

A função imprimir não recebe nem retorna nada. Apenas imprime a Lista na tela.

```
void imprimir(){
  for (int i = 0 ; i < pos ; i++)
```

```
      printf("%d ",l[i]);
  printf("\n");
}
```

Note que o for corre de 0 até pos -1, considerando apenas os elementos da Lista em si e não do array.

Função apagar

Talvez esta seja a mais divertida. A função apagar não recebe nem retorna nada. Apenas “apaga” a Lista.

```
void apagar(){
  pos = 0;
}
```

Note que apenas atribuímos 0 à pos. Não é necessário zerar nenhum elemento do array porque agora a Lista em si está vazia. Se invocar tamanho, a função vai retornar 0. Se inserir um elemento, ele vai ser inserido no índice zero do array e pos vai ser incrementado em 1. Se tentar remover um elemento que estava na Lista antes de ser apagada nada vai acontecer, pois a função remover invoca busca que trabalha em cima da variável pos, assim como todas as funções.

Listas Lineares Sequenciais Ordenadas

Agora chegou a hora de implementar uma Lista Sequencial Ordenada. Ora, se já sabemos que em uma Lista Ordenada os elementos já são inseridos de forma ordenada, basta modificarmos a função inserir.

Função inserir_ord

A função inserir_ord insere os elementos de forma ordenada na Lista. Não retorna nada e recebe o elemento a ser inserido.

```
void inserir_ord(int elemento){
  int i, p;
  if (pos < MAX) {
      for (i = 0 ; i < pos ; i++)
          if (l[i] >= elemento)
              break;

      for (p = pos ; p > i ; p--)
          l[p] = l[p-1];

      l[i] = elemento;
```

```
      pos ++;
  }
  else {
      printf("Não foi possível inserir %d. Lista
cheia.\n",elemento);
  }
}
```

Basicamente, o que a função acima faz é:

1. verifica se o novo elemento cabe na Lista:

```
  if (pos < MAX) {
```

2. encontra o índice (i) em que o novo elemento deve ser inserido, de modo que os elementos anteriores ao índice sejam menores que o elemento, e os elementos posteriores sejam maiores (ou iguais) que o elemento:

```
      for (i = 0 ; i < pos ; i++)
          if (l[i] >= elemento)
              break;
```

3. abrir espaço no array para o novo elemento, empurrando os elementos à direita do índice para frente:

```
      for (p = pos ; p > i ; p--)
         l[p] = l[p-1];
```

4. armazenar o novo elemento no espaço recém-aberto:

```
      l[i] = elemento;
```

5. incrementar pos

```
      pos ++;
```

E então, o que achou da nossa primeira Estrutura de Dados: Lista Linear Sequencial? Espero que tenha apreciado assim como apreciei prepará-la pra você!

Na próxima aula iremos tratar sobre Listas Encadeadas. Esteja com o estudo de ponteiros e alocação dinâmica em dia.

Bons estudos e até a próxima!