VOLTAR

Listas encadeadas

Apresentar o conceito de lista encadeada simples e mostrar alguns exemplos da utilização dessa estrutura de armazenamento de dados.

NESTE TÓPICO

Introdução

Estruturas de Dados é, muitas vezes, a pedra no sapato do programador inexperiente, por isso, aqui será mostrado de maneira simples como construir alguns exemplos. Quando se fala em listas, filas, pilhas e árvores pode-se dizer que todas, na verdade, são “listas de informações”, cuja diferença principal está no acesso a essas “listas” para inclusão e remoção de informações.

Arranjos (vetores ou matrizes) são mais simples de implementar: o conteúdo da lista é armazenado em um espaço de memória de tamanho para N elementos dispostos de forma em posições contínuas. Apesar de ser fácil a compreensão quanto à manipulação dos dados, os arranjos possuem limitações referentes à quantidade de elementos que o conjunto irá suportar, ou seja, o vetor possui um tamanho pré-determinado que pode ou não ser ocupado totalmente. Outro problema seria a necessidade de mais espaço do que o reservado inicialmente.

Já quando se fala em alocação dinâmica, são utilizadas posições descontinuadas de memória. Isso é possível, pois cada um dos elementos da lista deve possuir uma referência para os elementos seguinte e anterior. Essa referência é o endereço da posição de memória em que se encontra tal elemento.

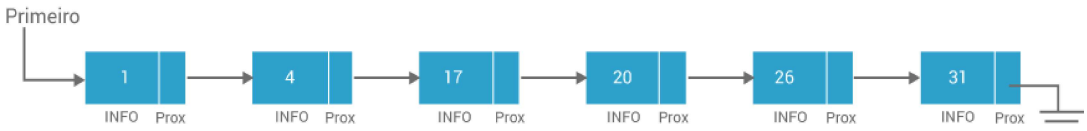
Numa lista encadeada, para cada novo elemento inserido na estrutura, se é alocado um espaço de memória para armazená-lo. Dessa forma, o espaço total de memória gasto pela estrutura é proporcional ao número de elementos nela armazenado. No entanto, não se pode garantir que os elementos armazenados na lista ocuparão um espaço de memória contíguo, portanto, não se tem acesso direto aos elementos da lista. Para que seja possível percorrer todos os elementos da lista, deve-se explicitamente guardar o encadeamento dos elementos, o que é feito armazenando-se, junto com a informação de cada elemento, um ponteiro para o próximo elemento da lista.

Listas Encadeadas Simples – Definição

Uma lista encadeada é uma representação de uma sequência de objetos na memória do computador.

A estrutura consiste numa seqüência encadeada de elementos, em geral chamados de nós da lista. A lista é representada por um ponteiro para o primeiro elemento (ou nó). Essa variável (também chamada de cabeça da lista) possibilita o acesso aos demais elementos contidos nela. Do primeiro elemento, pode-se alcançar o segundo seguindo o encadeamento, e assim por diante. O último elemento da lista aponta para NULL, sinalizando que não existe um próximo elemento.

Nesse caso, o primeiro elemento da lista, representado pelo campo info, é 1, o último elemento é 31, o predecessor de 31 é 26, o sucessor de 1 é 4 e assim por diante até o último elemento. Lembrem-se, os elementos dessa lista não estão organizados na memória seqüencialmente, igual a um arranjo. O campo Prox representa o ponteiro para o próximo elemento (endereço onde está armazenado o elemento seguinte da lista). A Lista é representada pela Figura abaixo



Representação da estrutura de uma lista encadeada simples

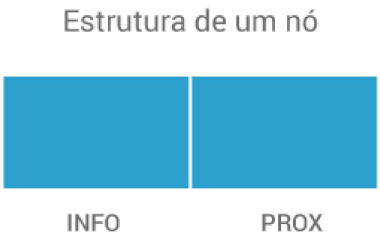
Fonte: Representação da estrutura de uma lista encadeada simples

Ao contrário de vetores, para acessar o i-ésimo elemento é necessário percorrer todos os elementos anteriores.

Estrutura de um nó da Lista Encadeada Simples

Uma lista encadeada (lista ligada) é uma sequência de nós, no qual cada nó contém um objeto de algum tipo e o endereço do nó seguinte. Se os objetos armazenados nos nós são do tipo int, a estrutura de cada nó de uma lista

pode ser definida como segue abaixo e é ilustrada pela Figura a seguir.



Estrutura de um nó de uma lista encadeada simples

Fonte: Elaborado pelo autor

```
1.  struct lista{
2.        int info;
3.        struct lista *prox;
4.  };
```

É conveniente tratar os nós como um novo tipo de dados e atribuir um nome a esse novo tipo:

```
1.  typedef struct lista *tipo_lista;
```

Principais operações para a Lista Encadeada Simples

Abaixo, seguem as principais operações que serão abordadas neste tópico:

- inserção de nós na lista.
- exibição de nós na lista.
- busca de um elemento na lista.
- remoção de nós na lista.

Operação de Inserção

Uma vez criada a lista vazia, pode-se inserir novos elementos nela. Para cada elemento inserido na lista, deve-se alocar dinamicamente a memória necessária para armazenar o elemento e encadeá-lo na lista existente. A função de inserção mais simples insere o novo elemento no inicio da lista.

Uma possível implementação dessa função é mostrada a seguir. Deve-se notar que o ponteiro que representa a lista deve ter seu valor atualizado, pois a lista deve passar a ser representada pelo ponteiro para o novo primeiro elemento. Por esta razão, a função de inserção recebe como parâmetros de entrada a lista onde será inserido o novo elemento e a informação do novo elemento, e tem como valor de retorno a nova lista, representada pelo ponteiro para o novo elemento.

```
1.  void insere_inicio (tipo_lista * primeiro, int elem)
2.  {
3.     tipo_lista no = (tipo_lista) malloc (sizeof(lista));
4.     no->info=elem;
5.     no->prox= *primeiro;
6.     *primeiro = no;
7.  }
```

Primeiramente, cria-se uma estrutura que representa um nó na lista com o comando *malloc*.

```
1.  tipo_lista no = (tipo_lista) malloc (sizeof(lista));
```

Criado o nó, o próximo passo é armazenar os dados nos campos do registro (info e prox). A princípio faz-se a atribuição do elemento ao campo elem.

```
1.  no->info = elem;
```

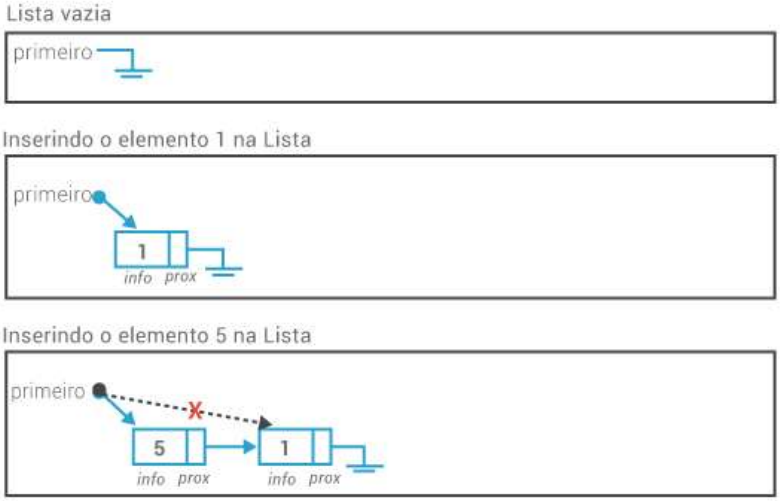
A próxima etapa é realizar a conexão do campo no->prox, ou seja, fazer o campo prox do nó recém criado apontar para onde o primeiro elemento da lista está apontando.

```
1.  no->prox= *primeiro;
```

Por último, atualizar o ponteiro da lista (primeiro), ou seja, “primeiro” vai apontar para o nó recém criado.

```
1.  *primeiro = no;
```

Vale lembrar que este procedimento aloca dinamicamente o espaço para armazenar o novo nó da lista, guarda a informação no novo nó e faz este nó apontar para (isto é, ter como próximo elemento) o elemento que era o primeiro da lista. Observe que não se pode deixar de atualizar a variável que representa o início da lista (primeiro) a cada inserção de um novo elemento. A figura a seguir ilustra a operação de inserção de um novo elemento no inicio da lista.



Operação de inserção de um novo elemento no início da lista encadeada simples

Fonte: Do próprio autor

Operação que exibe os elementos da lista

Para ilustrar a implementação de uma função que percorre todos os elementos da lista, considere a criação de um procedimento que imprima os valores dos elementos armazenados numa lista. Uma possível implementação dessa função é mostrada a seguir.

```
1.  void imprime_lista (tipo_lista primeiro)
2.  {
3.     tipo_lista aux;
4.
5.     printf ("\nA lista : \n");
6.     aux = primeiro;
7.     while (aux!=NULL)
8.     {
9.          printf("info=%d\t", aux->info);
10.         aux=aux->prox;
11.     }
12.    printf("\n");
13. }
```

Para buscar um elemento na lista é feito o percurso desde o início (primeiro) até que o elemento seja encontrado ou não. Caso o elemento seja encontrado, a busca retorna 1, caso não seja, retorna 0. Uma possível implementação dessa função é mostrada a seguir.

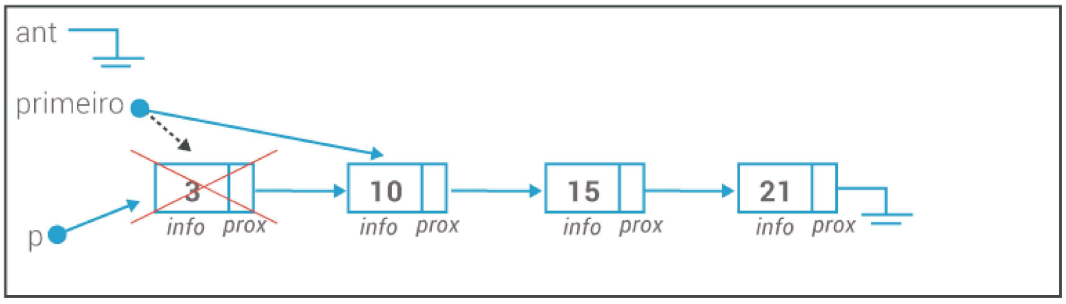
```
1.  int busca_lista (tipo_lista primeiro, int elem)
2.  {
3.     tipo_lista aux;
4.     aux = primeiro;
5.
6.
7.   while (aux!=NULL)
8.     {
9.        if (aux->info==elem)
10.          return 1;
11.       aux=aux->prox;
12.    }
13.    return 0;
14. }
```

Para completar o conjunto de funções que manipulam uma lista, deve-se implementar uma função que permita remover um elemento. A função tem como parâmetros de entrada o ponteiro para o primeiro da lista e o valor do elemento que se quer remover. Se o elemento a ser removido for o primeiro da lista, a variável deve ser atualizada.

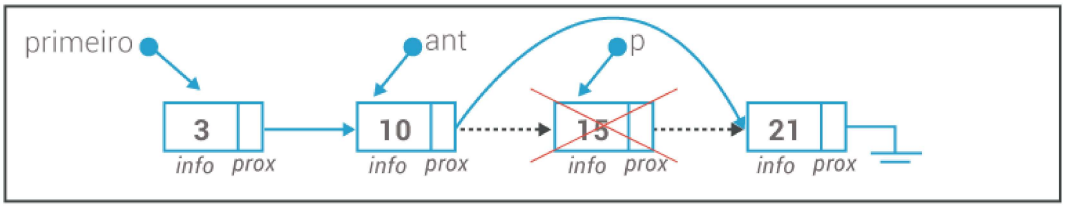
O procedimento para retirar um elemento da lista é mais complexo. Se for descoberto que o elemento a ser retirado é o primeiro da lista, deve-se fazer com que o novo valor da lista passe a ser o ponteiro para o segundo

elemento, e então se pode liberar o espaço alocado para o elemento que se quer retirar. Se o elemento a ser removido estiver no meio da lista, se deve fazer com que o elemento anterior a ele passe a apontar para o elemento seguinte, e então se pode liberar o elemento que se quer retirar. Deve-se notar que, no segundo caso, é preciso do ponteiro para o elemento anterior para se poder arrumar o encadeamento da lista. As figuras a seguir ilustram as operações de remoção.

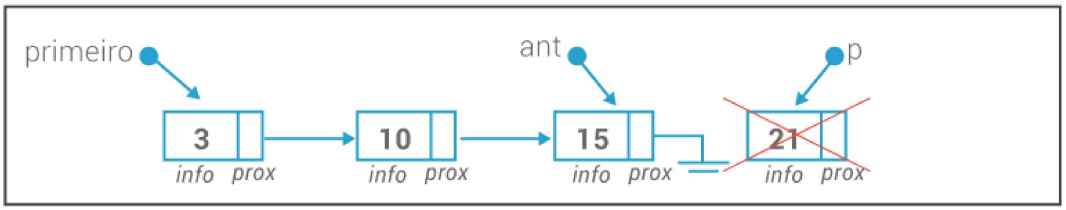
Remove o 1º elemento da Lista



Remove o 3º elemento da Lista (remoção no meio)



Remove o último elemento da Lista



Operações de remoção em uma lista encadeada simples

```
1.  int removeNo(tipo_lista *primeiro, int elem){
2.
3.       tipo_lista ant = NULL;
4.       tipo_lista p = *primeiro;
5.
6.       while (p != NULL && p->info != elem){
7.          ant = p;
8.          p = p->prox;
9.       }
10.
11.
12.      if (p == NULL)
13.        return 0;
14.      else {
15.              if (ant == NULL) {
16.              // retira o primeiro elemento
17.                 *primeiro = p->prox;
18.
19.              }
20.              else if (p->prox==NULL){
21.                   // retira o último elemento da lista
22.                     ant->prox = NULL;
23.
24.                   }
25.                   else {// retira elemento do meio da lista
26.                           ant->prox = p->prox;
27.
28.                      }
29.          free(p);
30.          }
31.      return 1;
32.  }
```

Exemplo 1

A seguir, apresentamos um exemplo completo de um algoritmo que trabalha com uma lista encadeada simples. O programa insere alguns valores na lista, exibe todos seus elementos, faz a busca por um elemento e remove um elemento da lista.

```c
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
/*****************************************************************************/
struct lista{
            int info;
               struct lista * prox;
             };

typedef struct lista * tipo_lista;
/*****************************************************************************/
void insere_inicio (tipo_lista * primeiro, int elem)
{
   tipo_lista no = (tipo_lista) malloc (sizeof(lista));
   no->info=elem;
   no->prox= *primeiro;
   *primeiro = no;
}/*****************************************************************************/
void imprime_lista (tipo_lista primeiro)
{
   tipo_lista aux;

   printf("\nA lista : \n");
   aux=primeiro;
   while (aux!=NULL)
   {

        printf("info=%d\t",aux->info);
        aux=aux->prox;
    }
   printf("\n");
}
/*****************************************************************************/
int busca_lista (tipo_lista primeiro, int elem)
{
   tipo_lista aux;
   aux=primeiro;

   while (aux!=NULL)
   {
      if (aux->info==elem)
         return 1;
      aux=aux->prox;
   }
   return 0;
}
/*****************************************************************************/
int remove_lista (tipo_lista * primeiro, int elem)
{
   tipo_lista aux=*primeiro;
   tipo_lista ant=NULL;

   while (aux!=NULL && aux->info!=elem)
   {
                ant=aux;
                     aux=aux->prox;
   }

   if (aux==NULL) /* Nao encontrou o elemento */
        return 0;
   else {
                if (ant==NULL)      /* O elemento está no inicio */
               *primeiro = aux->prox;
            else ant->prox=aux->prox;  /* O elemento está no meio */
                free(aux);
                return 1;
        }
}
/*****************************************************************************
                                                principal
*****************************************************************************/
```

```
72.  main()
73.  {
74.     tipo_lista L=NULL;
75.     int nro=0, elem;
76.
77.     insere_inicio(&L,5);
78.     insere_inicio(&L,7);
79.     insere_inicio(&L,3);
80.     insere_inicio(&L,8);
81.     insere_inicio(&L,15);
82.     imprime_lista (L);
83.     printf ("\n Digite o elemento a ser procurado na lista:");
84.     scanf ("%d", &elem);
85.
86.     if (busca_lista (L, elem)==1)
87.         printf ("\n Elemento foi encontrado na lista");
88.     else
89.         printf ("\n Elemento não foi encontrado na lista");
90.
91.     printf ("\n Digite o elemento a ser removida da lista:");
92.     scanf ("%d", &elem);
93.
94.     remove_lista (&L, elem);
95.     imprime_lista (L);
96.     getch();
97.  }
```

Quiz

Exercício

Listas encadeadas

INICIAR

Quiz

Exercício Final

Listas encadeadas

INICIAR

Referências

MIZRAHI, V. V. *Treinamento em linguagem C*, São Paulo: Pearson, 2008.

SCHILDT, H. C – Completo e Total, São Paulo: Pearson, 2006.

Avalie este tópico

ANTERIOR
Filas

Índice

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)