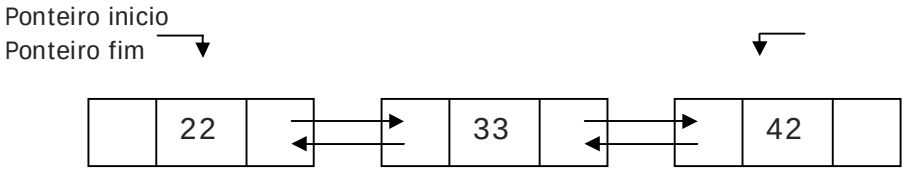
3.2 Listas Duplamente Encadeadas

As listas duplamente encadeadas utilizam nodos com dois campos de ligação (anterior e próximo), ou seja, uma lista duplamente encadeada é aquela em que cada nodo possui duas referências, a primeira é usada para indicar o nodo anterior e a segunda para apontar o próximo nodo. Permitindo dessa forma que a lista seja percorrida nos dois sentidos. Além dos dois campos são utilizados dois ponteiros: `inicio` e `fim`, onde, o ponteiro `inicio` aponta para o primeiro nodo da lista e o ponteiro `fim` aponta para o último nodo da lista.



Para a implementação das Listas Duplamente Encadeadas iremos utilizar o exemplo de armazenamento de um valor inteiro, além dos dois ponteiros de anterior (`ant`) e de próximo (`prox`), como é apresentado na estrutura nodod abaixo (em Linguagem C e Português Estruturado):

```
struct nodod{
 struct nodod *ant; //ponteiro para o nodo anterior
 int dados;
 struct nodod *prox; //ponteiro para o próximo nodo
 };

tipo
 nodod=registro
 *ant:nodod {ponteiro para o nodo anterior}
 dados:inteiro
 *prox:nodod {ponteiro para o próximo nodo}
 fim
```

3.2.1 Inserção em Listas Duplamente Encadeadas

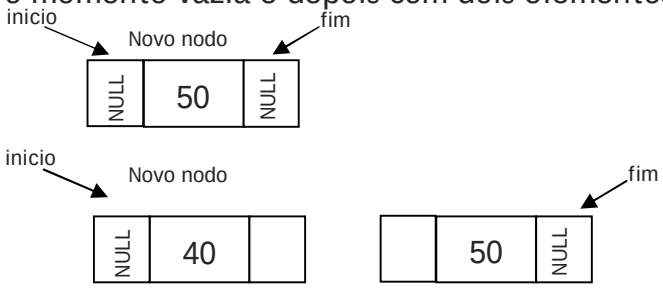
Antes de iniciar a construção da lista, o ponteiro que indica seu primeiro elemento deve ser inicializado com um endereço nulo (NULL), assim como seu ponteiro fim que indica o final da lista, indicando que a lista está vazia. Essa inicialização é implementada na função main (função principal), como apresentada abaixo.

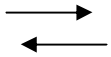
```
//Linguagem C
int main(){
struct nodod *ptri=NULL; //inicialização do ponteito *ptri como nulo
(ponteiro que indica o início da lista)
struct nodod *ptrf=NULL; //inicialização do ponteiro *ptrf como nulo
(ponteiro que indica o final da lista)
getch();
}

//Português Estruturado
Função principal
Variáveis
  *ptri,*ptrf:nodod
Início
 *ptri←nulo
 *ptrf←nulo
Fim.
```

3.2.1.1 Inserção no Início da Lista Duplamente Encadeada

Para inserir o novo nodo no início da lista deve-se, após alocar o nodo e preenchê-lo com o valor correspondente, apontar seus campos de elo (link, ponteiro) para o endereço daquele que era o primeiro, atribuir nulo (NULL) ao elo anterior (o elo do primeiro elemento sempre apontará para nulo), atualizar o ponteiro de início da lista para o novo nodo e atribuir o ponteiro de fim para o primeiro nodo a ser criado. Caso a lista esteja vazia, este passará a ser seu único nodo, tendo os ponteiros inicio e fim apontando para ele.

A figura abaixo apresenta a inserção de um novo nodo no início de uma lista, no primeiro momento vazia e depois com dois elementos.





Os algoritmos abaixo apresentam a função para a inserção a esquerda, ou seja, no início da lista, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int InsereEsquerda(struct nodod **inicio,struct nodod **fim,int
v){
2.     struct nodod *novo=NULL;
3.     novo=(struct nodod *)malloc(sizeof(struct nodod));
4.     if(novo!=NULL){
5.      novo->dados=v;
6.      novo->ant=NULL;
7.      novo->prox=*inicio;
8.      if(*inicio==NULL)
9.       *fim=novo;
10.     else (*inicio)->ant=novo;
11.    *inicio=novo;
12.    }else printf("\nNao foi possivel alocar!\n");
13.   }
```

Implementação em Português Estruturado

```
1.     Função InsereEsquerda(*inicio,*fim:nodod,v:inteiro)
2.     Variáveis
3.      *novo:nodod
4.     Início
5.      *novo←nulo
6.      aloca(novo)
7.      se(novo<>nulo)
8.      inicio
9.       novo->dados←v
10.     novo->ant←nulo
11.     novo->prox←*inicio
12.     se(*inicio=nulo)então
13.      *fim←novo
14.     senão (*inicio)->ant←novo
15.    *inicio←novo
16.    fim senão
17.     escreve("Nao foi possivel alocar!")
18.   Fim
```

Abaixo estão dispostos os comentários sobre a implementação:

Linha 1: procedimento para a inserção dos nodos a esquerda (início da lista), com recebimento dos ponteiros de inicio (`*inicio`) da lista e fim da lista (`*fim`) e do inteiro v para armazenamento.

Linha 3: declaração do ponteiro `novo` referente ao registro nodod, inicializado com o valor nulo (`NULL`), na linha 5.

Linha 5: alocação do ponteiro novo. Será alocado um novo nodo com um espaço para um inteiro e dois ponteiros (ant e prox). Esse ponteiro novo receberá o endereço de memória.

Linha 7: condição de verificação se o ponteiro novo foi alocado corretamente. Se sim, então executa a linha 8, senão executa a linha 17.

Linha 9: dados de novo (espaço alocado para o inteiro) recebe o valor de v passado por parâmetro.

Linha 10: ant de novo (espaço alocado para o ponteiro do nodo anterior da lista) recebe nulo (NULL). Sempre o primeiro nodo da lista terá o ponteiro ant como NULL.

Linha 11: prox de novo (espaço alocado para o ponteiro do próximo nodo da lista) recebe o ponteiro de inicio (pois estamos inserindo o nodo a esquerda, então sempre o último a ser alocado terá o endereço do ponteiro de início).

Linha 12: verificação se o ponteiro de inicio é nulo (NULL), se verdadeiro executa a linha 13, senão executa a linha 14.

Linha 13: se o ponteiro de inicio for igual a nulo então o ponteiro de fim recebe o endereço de memória do nodo novo, pois o primeiro nodo será o último.

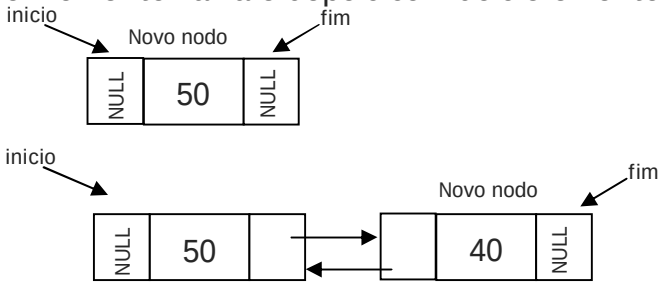
Linha 14: se o ponteiro de inicio for diferente de nulo, então o ponteiro de anterior (ant) do início receberá o endereço de novo, fazendo assim o encadeamento.

Linha 15: o ponteiro de inicio receberá o endereço de novo, ficando assim apontando para o primeiro nodo da lista.

3.2.1.2 Inserção no Final da Lista Duplamente Encadeada

Para inserir o novo nodo no final da lista requer encadear aquele que era o último nodo da lista com o novo nodo, ou seja, fazer com que seu campo de elo aponte para o endereço onde foi alocado o novo nodo. Sempre tendo o cuidado de atribuir o endereço do último nodo da lista para o elo anterior (ant). O ponteiro que guarda o início da lista somente será afetado no caso em que a lista estava vazia, quando então apontará para este novo nodo. Já o ponteiro fim será movido a cada nova inserção de nodo.

A figura abaixo apresenta a inserção de um novo nodo no final de uma lista, no primeiro momento vazia e depois com dois elementos.



Para localizar o último nodo da lista, basta acessar o ponteiro `fim`, pois este guarda o endereço do último nodo da lista, aqui não há a necessidade de percorrer a lista.

Os algoritmos abaixo apresentam a função para a inserção a direita, ou seja, no final da lista, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int InsereDireita(struct nodod **inicio,struct nodod **fim,int
v){
2.     struct nodod *novo=NULL;
3.     novo=(struct nodod *)malloc(sizeof(struct nodod));
4.     if(novo==NULL)
5.      printf("\nNao foi possivel alocar!\n");
6.       else{
7.        novo->dados=v;
8.        novo->prox=NULL;
9.        novo->ant=*fim;
10.    if(*inicio==NULL)
11.     *inicio=novo;
12.      else
13.       (*fim)->prox=novo;
14.    *fim=novo;
15.    }//fim do else
16.   }//fim da função
```

Implementação em Português Estruturado

```
1.     Função InsereDireita(*inicio,*fim:nodod,v:inteiro)
2.     Variáveis
3.      *novo:nodod
4.     Início
5.      *novo←nulo
6.      aloca(novo)
7.      se(novo=nulo)então
8.       escreve("Nao foi possivel alocar!")
9.       senão inicio
10.      novo->dados←v
11.      novo->prox←nulo
12.      novo->ant←*fim
13.    se(*inicio=nulo)então
14.     *inicio←novo
```

```
15.      senão
16.      (*fim)->prox←novo
17.     *fim←novo
18.     fim
19.   Fim
```

Abaixo estão dispostos os comentários sobre a implementação:

Linha 1: procedimento para a inserção dos nodos a esquerda (início da lista), com recebimento dos ponteiros de inicio (`*inicio`) da lista e fim da lista (`*fim`) e do inteiro v para armazenamento.

Linha 3: declaração do ponteiro `novo` referente ao registro nodod, inicializado com o valor nulo (`NULL`) na linha 5.

Linha 6: alocação do ponteiro `novo`. Será alocado um novo nodo com um espaço para um inteiro e dois ponteiros (`prox` e `ant`). Esse ponteiro `novo` receberá o endereço de memória.

Linha 7: condição de verificação se o ponteiro `novo` possui valor diferente de nulo (`NULL`). Se sim (foi alocado corretamente), então executa a linha 8, senão executa a linha 17, onde será emitida uma mensagem ao usuário.

Linha 9: `dados` de `novo` (espaço alocado para o inteiro) recebe o valor de v passado por parâmetro.

Linha 10: `ant` de `novo` (espaço alocado para o ponteiro do nodo anterior da lista) recebe o valor de nulo (sempre o campo `ant` do primeiro nodo apontara para `NULL`).

Linha 11: `prox` de `novo` (espaço alocado para o ponteiro do próximo nodo da lista) recebe o endereço de memória do ponteiro `inicio`.

Linha 12: verificação se a lista está vazia, ou seja, se o ponteiro `inicio` é nulo (`inicio=nulo`). Se verdadeiro então executa a linha 13, senão executa a linha 14.

Linha 13: se a lista é vazia, então o ponteiro de `fim` recebe o endereço de memória do nodo novo em função de ser o único nodo da lista e as demais inserções serão realizadas no início da lista.

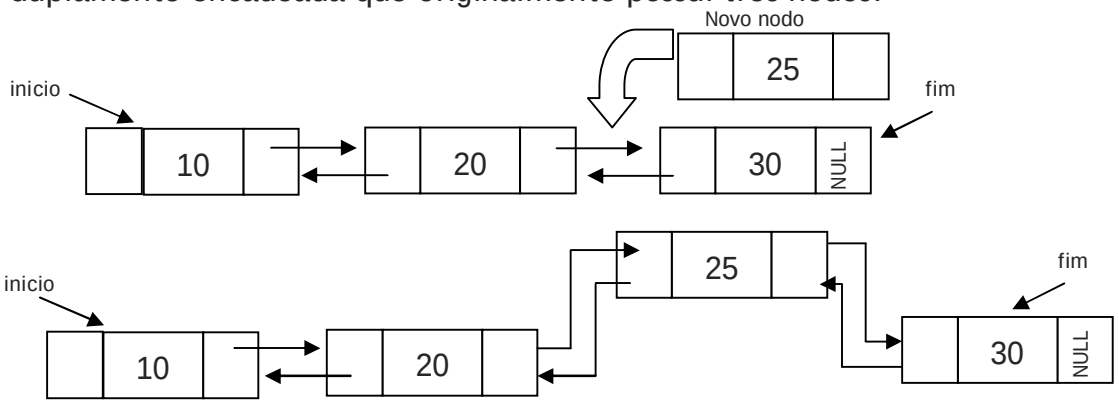
Linha 14: `ant` de `inicio` (campo `ant` do ponteiro `inicio`) recebe o endereço de memória do nodo `novo`.

Linha 14: ponteiro `inicio` recebe o endereço de memória do nodo `novo`. Atualizando assim o inicio da lista.

3.2.1.4 Inserção de nodo na Lista Duplamente Encadeada

Para inserir o novo nodo no início, meio ou fim, de uma lista duplamente encadeada é realizada adequando os campos de elo dos nodos anterior e posterior ao novo nodo.

A figura abaixo apresenta a inserção de um novo nodo no meio de uma lista duplamente encadeada que originalmente possui três nodos.



Para localizar o nodo da lista, é necessário receber como parâmetro a ordem que o novo nodo deverá ocupar na lista, além dos endereços dos ponteiros de `inicio` e `fim` da lista e do dado a ser inserido no novo nodo. Um parâmetro é retornado falso quando não existir espaço físico para alocar o novo nodo, ou quando a posição requerida para inserção não for compatível com o número de nodos da lista. No caso da inserção ter sucesso, o ponteiro `ant` do novo nodo receberá o valor do nodo anterior; o ponteiro `prox` do nodo anterior receberá o endereço do novo nodo; o ponteiro `prox` do novo nodo receberá o endereço do próximo nodo.

Os algoritmos abaixo apresentam a função para a inserção de um novo nodo na lista, tanto no início, meio ou fim, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int InsereNodo(struct nodod **inicio,struct nodod **fim,int v,int
k){
2.     struct nodod *novo=NULL,*aux;
3.     int pos=1,achou=0;
4.     aux=*inicio;
5.     while((aux!=NULL)&&(achou!=1)){
6.      if(pos==k){
7.       novo=(struct nodod *)malloc(sizeof(struct nodod));
8.       if(novo!=NULL){
9.        novo->dados=v;
10.      novo->ant=aux->ant;
```

```
11.       novo->prox=aux;
12.       if(*inicio==aux){
13.        novo->prox=aux;
14.        *inicio=novo;
15.        }else{
16.         aux->ant->prox=novo;
17.         aux->ant=novo;
18.         }//fim do else
19.      }else printf("\nNao foi possivel alocar!\n");
20.      achou=1;
21.      break;
22.      }//fim do if(pos==k)
23.     pos++;
24.     aux=aux->prox;
25.     }//fim do while
26.    return(achou);
27.    }
```

Implementação em Português Estruturado

```
1.      Função InsereNodo(*inicio,*fim:nodod,v,k:inteiro)
2.      Variáveis
3.       *novo,*aux:nodod
4.       pos,achou:inteiro
5.      Início
6.      *novo←nulo
7.      pos←1
8.      achou←0
9.      aux←*inicio
10.     enquanto((aux<>nulo)&&(achou<>1))faça
11.     inicio
12.      se(pos=k)então
13.      inicio
14.       aloca(novo)
15.       se(novo<>nulo)então
16.       inicio
17.        novo->dados←v
18.        novo->ant←aux->ant
19.        novo->prox←aux
20.        se(*inicio=aux)então
21.        inicio
22.         novo->prox←aux
23.         *inicio←novo
24.         fim
25.        senão inicio
26.         aux->ant->prox←novo
27.         aux->ant←novo
28.         fim
29.      fim senão
30.       escreve("Nao foi possivel alocar!")
31.       achou←1
32.       encerra o comando while
33.      fim
34.     pos←pos+1
35.     aux←aux->prox
```

```
36.     fim
37.    retorna(achou)
38.    fim
```

Abaixo estão dispostos os comentários sobre a implementação:

Linha 1: procedimento para a inserção do nodo no início, meio ou final da lista, com recebimento dos ponteiros de `inicio` e `fim` da lista e dos inteiros `v` para armazenamento e `k` para o valor a ser buscado na lista (o valor será inserido após esse valor).

Linha 3: declaração do ponteiro `novo` referente ao registro nodod, inicializado com o valor nulo (`NULL`) na linha 6 e do ponteiro `aux`.

Linha 4: declaração de uma variável inteira `achou`, que será utilizada para retornar se a inserção obteve ou não sucesso, inicializada com o valor zero (0) na linha 8 e a variável `pos` responsável pela posição dos nodos na lista, inicializada com o valor 1 na linha 7.

Linha 9: o ponteiro `aux` recebe o endereço de memória do ponteiro `inicio`, sendo assim aponta para o primeiro nodo da lista.

Linha 10: laço para percorrer a lista com o objetivo de achar o nodo que esta sendo buscado. Enquanto o ponteiro `aux` for diferente de nulo (enquanto houver nodos na lista) e a variável `achou` diferente de 1 (o valor zero significa que ainda não foi encontrado o nodo e o valor 1 significa que foi encontrado) será executado.

Linha 12: verificação se o valor buscado (k) é igual a posição contida na variável `pos`. Se verdadeiro executa a linha 13.

Linha 14: alocação do ponteiro `novo`. Será alocado um novo nodo com um espaço para um inteiro e os dois ponteiros (`prox` e `ant`). Esse ponteiro novo receberá o endereço de memória.

Linha 15: condição de verificação se o ponteiro `novo` foi alocado corretamente. Se sim, então executa a linha 16, senão executa a linha 29, onde será emitida uma mensagem ao usuário informando que não foi possível alocar.

Linha 17: `dados` de `novo` (espaço alocado para o inteiro) recebe o valor de `v` passado por parâmetro.

Linha 18: `ant` de `novo` (espaço alocado para o elo) recebe o endereço de memória contido em `ant` de `aux`.

Linha 19: `prox` de `novo` recebe o endereço de memória do ponteiro `aux`.

Linha 20: verifica se os ponteiros de `inicio` e `aux` são iguais (apontam para o mesmo nodo). Se verdadeiro executa a linha 21, senão executa a linha 25.

Linha 22: `prox` de `novo` recebe o endereço de memória do ponteiro `aux`.

Linha 23: ponteiro `inicio` recebe o endereço de memória do ponteiro `novo`.

Linha 15: finalização do então da verificação se os ponteiros `aux` e `inicio` são iguais. Início do comando senão (`else`) da mesma verificação.

Linha 26: `ant` de `aux` de `prox` recebe o endereço de `novo` (para o encadeamento). Primeiramente será executado o `ant->aux` que retornará o endereço contido no elo anterior do ponteiro `aux` (auxiliar). Depois será executado o `prox` do endereço recebido, ou seja, no elo `prox` de `aux` de `ant` será inserido o endereço do ponteiro `novo`.

Linha 27: `ant` de `aux` (elo anterior do ponteiro auxiliar) recebe o endereço de memória do ponteiro `novo`.

Linha 31: a variável de verificação `achou` recebe o valor de 1, que significa que a operação de inserção foi concluída com sucesso.

Linha 32: executa o comando `break` (interrompe o laço).

Linha 34: a variável `pos` é incrementada em mais 1.

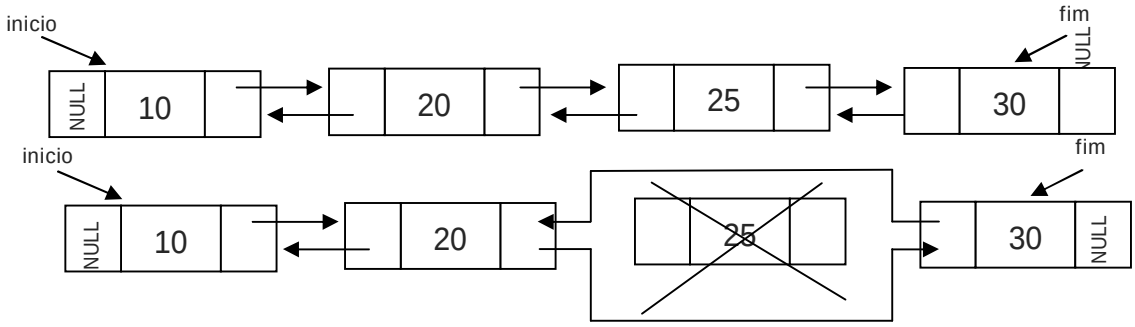
Linha 35: o ponteiro `aux` recebe o endereço de memória do próximo nodo, visto que está recebendo o elo (endereço) contido.

Linha 37: retorna para a função chamadora o valor da variável de verificação `achou`, sendo que 0 significa que a operação não foi concluída e 1 significa que foi concluída com sucesso.

3.2.2 Remoção de nodo na Lista Duplamente Encadeada

A remoção de um nodo no início, meio ou fim, de uma lista duplamente encadeada é realizada alterando os campos de elo dos nodos anterior e posterior ao novo nodo, ou seja, o nodo imediatamente anterior apontará para aquele que era o seguinte do nodo excluído da lista. Caso o nodo excluído seja o primeiro, o endereço do segundo deverá ser copiado para o ponteiro de início da lista. Caso seja o último, o anterior deverá ficar com o campo de elo nulo.

A figura abaixo apresenta a exclusão de um novo nodo no meio de uma lista encadeada que originalmente possui quatro nodos.



Para localizar o nodo da lista a ser excluído, é necessário receber como parâmetro a ordem (ou valor) que o nodo deverá ser excluído na lista, além dos endereços dos ponteiros de `inicio` e `fim` da lista. Um parâmetro é retornado falso quando não existir espaço físico para alocar o novo nodo, ou quando a posição requerida para remoção não for compatível com o número de nodos da lista. No caso da exclusão ter sucesso, o ponteiro `aux` percorre a lista até o nodo a ser excluído e os ajustes de seus campos de elo (`ant` e `prox`) são alterados para efetivar os devidos encadeamentos.

Os algoritmos abaixo apresentam a função para a exclusão de um nodo na lista duplamente encadeada, tanto no início, meio ou fim, tendo como parâmetro o valor inteiro do nodo a ser excluído, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int ExcluiNodo(struct nodod **inicio,struct nodod **fim,int
v){
2.     struct nodod *aux=NULL,*del=NULL;
3.     int achou=0;
4.     if(*inicio==NULL)
5.      printf("\nLista Vazia!\n");
6.      else{
7.       if((*inicio)->dados==v){
8.        (*inicio)->prox->ant=NULL;
9.        del=*inicio;
10.       *inicio=(*inicio)->prox;
11.      free(del);
12.      achou=1;
13.      }else{
```

```
14.       if((*fim)->dados==v){
15.      (*fim)->ant->prox=(*fim)->prox;
16.      del=*fim;
17.      *fim=(*fim)->ant;
18.      free(del);
19.      achou=1;
20.      }else{
21.       aux=*inicio;
22.       while(aux!=NULL){
23.        if(aux->dados==v){
24.          del=aux;
25.          aux->ant->prox=aux->prox;
26.          aux->prox->ant=aux->ant;
27.          free(del);
28.          achou=1;
29.          break;
30.        }//fim do if
31.        aux=aux->prox;
32.       }//fim do while
33.      }//fim do else *fim==v
34.     }//fim do else *inicio==v
35.    }//fim do else lista vazia
36.    return(achou);
37.   }
```

Implementação em Português Estruturado

```
1.   Função ExcluiNodo(*inicio,*fim:nodod,v:inteiro)
2.   Variáveis
3.   *aux,*del:nodod
4.   achou:inteiro
5.   Início
6.   achou←0;
7.   se(*inicio=nulo)então
8.    escreve("Lista Vazia!")
9.    Senão inicio
10.    se((*inicio)->dados=v)então
11.     (*inicio)->prox->ant←nulo
12.     del←*inicio
13.     *inicio←(*inicio)->prox
14.    desaloca(del)
15.    achou←1
16.    fim senão inicio
17.    se((*fim)->dados=v)então
18.    inicio
19.    (*fim)->ant->prox←(*fim)->prox
20.    del←*fim
21.    *fim←(*fim)->ant
22.    desaloca(del)
23.    achou←1
24.    fim senão inicio
25.     aux←*inicio
26.     enquanto(aux!=NULL)faça
27.     inicio
28.      se(aux->dados=v)então
29.      inicio
```

```
30.        del←aux
31.        aux->ant->prox←aux->prox
32.        aux->prox->ant←aux->ant
33.        desaloca(del)
34.        achou←1
35.        finalize comando enquanto
36.      fim
37.      aux←aux->prox
38.     fim
39.    fim
40.   fim
41.  fim
42.  retorna(achou)
43. Fim
```

Abaixo estão dispostos os comentários sobre a implementação:

Linha 1: procedimento para a exclusão de um nodo no início, meio ou final da lista duplamente encadeada, com recebimento dos ponteiros de `inicio` e `fim` da lista e do inteiro e `v` para o valor a ser buscado na lista (o nodo que contém esse valor será excluído da lista).

Linha 3: declaração dos ponteiros `aux` e `del`. O ponteiro `aux` será utilizado para percorrer a lista, ou seja, nodo a nodo, até encontrarmos o final da lista ou o nodo a ser excluído. O ponteiro `del`, será responsável por guardar o endereço de memória do nodo a ser excluído.

Linha 4: declaração da variável `achou`, que será inicializada com zero na linha 6. Essa variável é responsável por retornar a unidade chamado o valor 0 (zero) - não houve exclusão do nodo; ou o valor 1 (um) – a exclusão foi executada.

Linha 7: verificação se a lista está vazia, ou seja, não possui nodos inseridos. Se verdadeiro executa a linha 8, onde será emitida uma mesnagem ao usuário que a lista está vazia, senão executa a linha 9.

Linha 10: verificação se o conteúdo inteiro do ponteiro `inicio` é igual ao valor `v` passado por parâmetro. Se verdadeiro executa a linha 11, senão executa a linha 16.

Linha11: ponteiro `inicio->prox->ant` recebe `NULL` (nulo), pois o próximo nodo será o primeiro da lista após a exclusão.

Linha 12: o ponteiro `del` recebe o endereço de memória do ponteiro `inicio`, para posteriormente ser desalocado.

Linha 13: o ponteiro `inicio` recebe o endereço de memória contido do elo `prox`, dessa forma ele irá apontar para o próximo da lista, que passará a ser o primeiro nodo.

Linha 14: o ponteiro `del` é desalocado da memória.

Linha 15: a variável de verificação `achou` recebe o valor de 1, que significa que a operação de remoção foi concluída com sucesso.

Linha 17: verificação se o conteúdo inteiro do ponteiro `fim` é igual ao valor `v` passado por parâmetro. Se verdadeiro executa a linha 18, senão executa a linha 24.

Linha 19: ponteiro `(*fim)->ant->prox` recebe o endereço do elo `(*fim)->prox`, pois será o último nodo da lista após a exclusão.

Linha 20: o ponteiro `del` recebe o endereço de memória do ponteiro `fim`, para posteriormente ser desalocado, na linha 22.

Linha 21: o ponteiro `fim` recebe o endereço de memória contido do elo `ant`, dessa forma ele irá apontar para o anterior da lista, que passará a ser o último nodo.

Linha 23: a variável de verificação `achou` recebe o valor de 1, que significa que a operação de remoção foi concluída com sucesso.

Linha 25: o ponteiro `aux` recebe o endereço de memória do ponteiro `inicio`, para percorrer a lista a busca do nodo a ser excluído.

Linha 26: laço para percorrer a lista a fim de encontrar o nodo a ser excluído. Comando `while` (enquanto) o ponteiro `aux` for diferente de nulo.

Linha 28: verificação se o conteúdo inteiro do ponteiro `aux` é igual ao valor passado por parâmetro. Se verdadeiro executa a linha 29.

Linha 30: ponteiro `del` recebe o endereço de memória do ponteiro `aux`, para posteriormente ser desalocado na linha 33.

Linha 31: o campo `aux->ant->prox` recebe o conteúdo de memória do elo `aux->prox`, fazendo assim o encadeamento do elo `prox`.

Linha 32: o campo `aux->prox->ant` recebe o conteúdo de memória do elo `aux->ant`, fazendo assim o encadeamento do elo `ant`.

Linha 34: a variável de verificação `achou` recebe o valor de 1, que significa que a operação de inserção foi concluída com sucesso.

Linha 35: execução do comando `break` (na Linguagem C) que finaliza do laço.

Linha 37: ponteiro auxiliar (`aux`) recebe o endereço de memória do campo `prox`, passando assim a apontar para o próximo nodo.

Linha 42: retorna para a função chamadora o valor da variável de verificação `achou`, sendo que 0 significa que a operação não foi concluída e 1 significa que foi concluída com sucesso.

3.2.3 Pesquisa de nodo na Lista Duplamente Encadeada

A pesquisa a um determinado nodo necessita que a lista seja percorrida, a partir do seu primeiro nodo, até o nodo buscado. Este nodo pode ser identificado através de alguma informação contida nele (valor inteiro ou ponteiro), ou então em seu posicionamento na lista.

Os algoritmos abaixo apresentam a função para a pesquisa de um novo nodo na lista a partir de sua posição passada por parâmetro, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int PesquisaNodo(struct nodod **inicio,struct nodod **fim,int
k){
2.     struct nodod *aux;
3.     int achou=0,pos=1;
4.     aux=*inicio;
5.     while((aux!=NULL)&&(achou!=1)){
6.      if(pos==k)
7.       achou=1;
8.      pos++;
9.      aux=aux->prox;
10.     }//fim do while
11.   return(achou);
12.   }
```

Implementação em Português Estruturado

```
1.     Função PesquisaNodo(*inicio,*fim:nodod,k:inteiro)
2.     Variáveis
3.      *aux:nodod
4.      achou,pos:inteiro
5.     Início
6.      achou←0
7.      pos←1
8.      aux←*inicio
9.      enquanto((aux!=NULL)&&(achou!=1))faça
10.     inicio
11.      se(pos=k)
12.       achou←1
```

```
13.       pos←pos+1
14.       aux←aux->prox
15.       fim
16.     retorna(achou)
17.   Fim
```

Abaixo estão dispostos os comentários sobre a implementação, linha a linha.

Linha 1: procedimento para a pesquisa de um nodo na lista, com recebimento dos ponteiros `inicio` e `fim` da lista e do inteiro e k (posição do nodo na lista a ser pesquisado).

Linha 3: declaração do ponteiro `aux`, esse ponteiro será utilizado para percorrer a lista, ou seja, nodo a nodo, até encontrarmos o final da lista ou o nodo a ser pesquisado.

Linha 4: declaração de uma variável inteira `achou`, inicializada com zero na linha 6. Se o valor for zero, significa que não existe o nodo. Declaração da variável inteira `pos`, inicializada com o valor 1 na linha 7, responsável por armazenar a posição a cada verificação de nodo.

Linha 8: ponteiro `aux` recebe o endereço de memória do ponteiro `inicio`, com o propósito de percorrer a lista na busca do nodo.

Linha 9: laço para percorrer a lista, enquanto o ponteiro `aux` for diferente de nulo (`NULL`) e a variável `achou` diferente de 1 (um) será executado. Dessa forma é garantida que toda a lista será verificada.

Linha11: verificação se a variável `pos` é igual a posição do nodo a ser buscado. Se verdadeira executa a linha12.

Linha 12: a variável `achou` recebe o valor 1 (nodo encontrado).

Linha 13: a variável `pos` é incrementada com mais 1 (passa a contar a posição do próximo nodo, pois ainda não foi encontrado o nodo pesquisado).

Linha 14: o ponteiro `aux` recebe o endereço de memória do próximo nodo contido no elo `prox`.

Linha 16: retorno a unidade chamadora da variável `achou`, sendo que 1 (um) significa que foi encontrado o nodo e 0 (zero) significa que o nodo não foi encontrado na lista.

3.2.4 Remoção de uma Lista Duplamente Encadeada

Caso a lista duplamente encadeada não seja mais necessária, as posições ocupadas devem ser liberadas. Isto é feito percorrendo a lista a partir de seu primeiro nodo, liberando cada uma das posições ocupadas pelos nodos. Ao final os ponteiros de `inicio` e `fim` devem possuir o valor nulo (`NULL`).

Os algoritmos abaixo apresentam a função para a remoção de uma lista duplamente encadeada, em Linguagem C e Português Estruturado.

Implementação em Linguagem C

```
1.     int RemocaoLista(struct nodod **inicio,struct nodod **fim){
2.     struct nodod *aux=NULL;
3.     if(*inicio==NULL)
4.      printf("\nLista Vazia!\n");
5.      else{
6.       while((*inicio)!=NULL){
7.        aux=*inicio;
8.        *inicio=(*inicio)->prox;
9.        free(aux);
10.     }//fim do while
11.    *fim=NULL;
12.    }//fim do else
13.   }//fim da função
```

Implementação em Português Estruturado

```
1.     Função RemocaoLista(*inicio,*fim:nodod)
2.     Variáveis
3.      *aux:nodod
4.     Início
5.      se(*inicio=nulo)então
6.       escreve("Lista Vazia!");
7.       senão
8.        inicio
9.        enquanto (*inicio<>nulo)faça
10.       inicio
11.       aux←*inicio
12.       *inicio←(*inicio)->prox
13.       desaloca(aux)
14.       fim
15.     *fim←nulo
16.    fim
17.   fim
```

Abaixo estão dispostos os comentários sobre a implementação:

Linha 1: procedimento para a remoção de uma lista, com recebimento dos ponteiros `inicio` e `fim` da lista como parâmetro.

Linha 3: declaração do ponteiro `aux`.

Linha 5: verificação de a lista está vazia, ou seja, o ponteiro `inicio` igual a nulo (`NULL`). Se verdadeiro, executa a linha 6, que emiti uma mensagem ao usuário que a lista está vazia, senão executa a linha 7.

Linha 9: laço para percorrer a lista até que o ponteiro `inicio` seja diferente de nulo, ou seja, a lista ainda possui nodos.

Linha 10: o ponteiro auxiliar (`aux`) recebe o endereço de memória do ponteiro `inicio`.

Linha 11: ponteiro `inicio` recebe o endereço de memória do próximo elemento da lista, ou seja, o inicio da lista será o próximo elemento.

Linha 12: desaloca o ponteiro `aux`, ou seja, libera a posição de memória.

Linha 15: o ponteiro `fim` recebe nulo (`NULL`), pois a lista esta vazia.

Recapitulando

Listas Encadeadas são estruturas de armazenamento em tempo de execução, ou seja, por alocação dinâmica. São compostas por pelo menos dois nodos, um com o conteúdo e outro com o elo de ligação (ponteiro).

Listas Simplesmente Encadeadas são estruturas que possuem dois campos: um com o conteúdo e outro com o elo de ligação (ponteiro `prox`) para o próximo nodo. É utilizado um ponteiro de `inicio` no primeiro nodo.

Listas Duplamente Encadeadas são estruturas que possuem três campos: dois ponteiros e conteúdo. Os elos (ponteiros) apontam para o nodo anterior (`ant`) e para o próximo nodo (`prox`). São utilizados dois ponteiros: `inicio` (aponta para o primeiro nodo da lista) e `fim` (aponta para o último nodo da lista).

Referências Bibliográficas

ASCENCIO, Ana Fernanda Gomes e CAMPOS, Edilene Aparecida Veneruchi de. Fundamentos da Programação de Computadores: Algoritmos PASCAL, C/C++ (padrão ANSI) e JAVA. 3ª Edição. São Paulo: Pearson Education do Brasil, 2012.

EDELWEISS, Nina e GALANTE, Renata. Estruturas de Dados – Série Livros Didáticos Informática Ufrgs. Porto Alegre: Editora Bookman, 2009.

LORENZI, Fabiana; NOLL, Patrícia; CARVALHO, Tanisi. Estruturas de Dados. Porto Alegre: Editora Thomson, 2007.

SCHILDT, Herbert. C Completo e Total – 3ª edição revisada e atualizada. São Paulo: Pearson Makron Books, 1997.

CELES, Waldemar; CERQUEIRA, Reanato e RANGEL, José Lucas. Introdução a Estruturas de Dados: com técnicas de programação em C. Rio de Janeiro: Elsevier, 2004.