Exercicio Slide 69

As listas duplamente encadeadas tem uma vantagem simples durante a implementação dos métodos de manipulação da lista. Como exemplo temos a função de remoção de uma lista simples e uma lista duplamente encadeadas.

```
100  Node * remove_by_name (Node *list, char name[]) {
101      if (list != NULL) {
102          Node *tmp = list;
103          Node *previous = NULL;
104
105          if (compare_strings(tmp->name, name) == 0) {
106              previous = tmp;
107              tmp = tmp->next;
108              free(previous);
109              return tmp;
110          }
111          while (tmp != NULL && (compare_strings(tmp->name, name)
112              previous = tmp;
113              tmp = tmp->next;
114          }
115
116          if (tmp == NULL) {
117              return list;
118          } else {
119              previous->next = tmp->next;
120          }
121
122          free(tmp);
123          return list;
124      }
125  }
```

Se observarmos a linha 103 é criada uma variável do tipo Node que tem a função de guardar a referência do elemento anterior da lista, precisa-se desta referência para manter o encadeamento da lista intacto ao remover o elemento.

O código abaixo representa o método de remoção de um elemento de uma lista duplamente encadeada. Comparando com o trecho de código acima percebe-se que o membro anterior do elemento removido já está guardado na própria estrutura de dados. Esta comparação é apenas um exemplo de como as listas duplamente encadeadas, podem, algumas operações, ter uma performance melhor ante as listas simplesmente encadeadas tanto quanto ao custo operacional quanto de implementação.

```
93  D_linked_list * remove_any (D_linked_list *list, int e) {
94
95      if (list != NULL) {
96          D_linked_list *tmp = list;
97
98          while (tmp != NULL && tmp->v != e)
99              tmp = tmp->next;
100
101         if (tmp->prev == NULL) {
102             list = list->next;
103             list->next->prev = NULL;
104         } else if (tmp->next == NULL) {
105             tmp->prev->next = NULL;
106         } else {
107             tmp->next->prev = tmp->prev;
108             tmp->prev->next = tmp->next;
109         }
110
111         free(tmp);
112
113         return list;
114     }
115
116 }
```

Exercicio Apostila - Ex1:

Realizando o teste de mesa com a lista proposta e o código que irá atuar nesta lista chega-se à conclusão que após as operações realizadas a lista retornada será nula.