Nome: Julio Cesario de Paiva Leão
RA: 1916033
Estrutura de Dados
Engenharia de Software – 2018/01

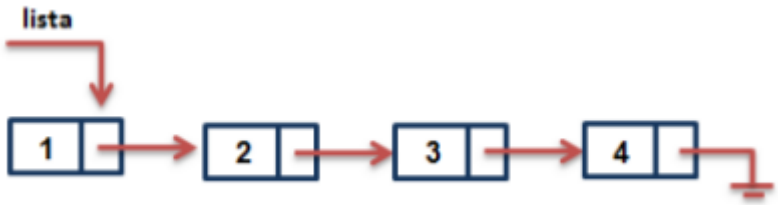
[SO33A-ES31] T08 - Listas Duplamente Encadeadas

- utilizando a apostila disponível em: https://goo.gl/4YRXRb
- exercícios obrigatórios (página 63):

  1, 3, 4;

- os outros exercícios ficam com ***complementares***;
- compactar todos os exercícios e enviar pelo Moodle;

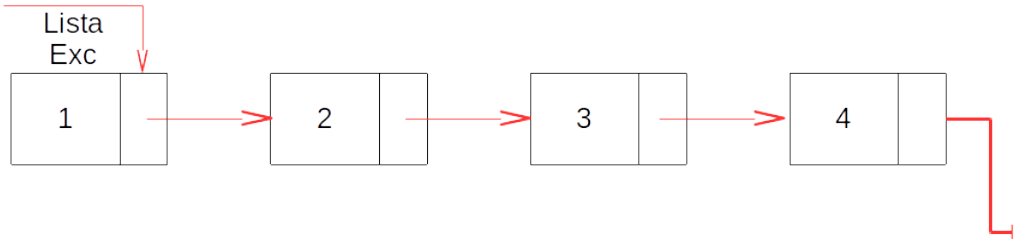
1) Dada a lista:



a) Qual será a configuração final da lista após a execução dos comandos abaixo?

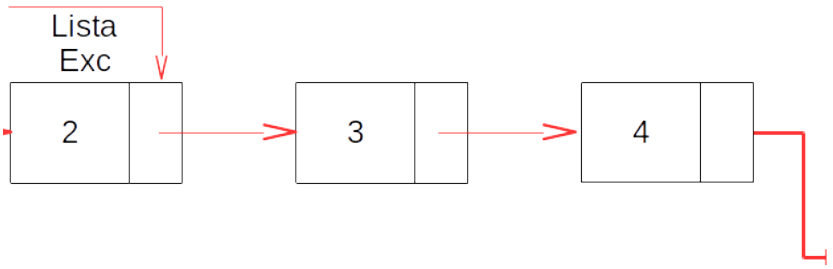
```
for(r = 1; r <= 3; r++){
   if(lista != NULL)
      exc = lista;
      lista = lista -> prox;
      free(exc);
}
```

RESP:
exc = lista;
lista = lista->prox;

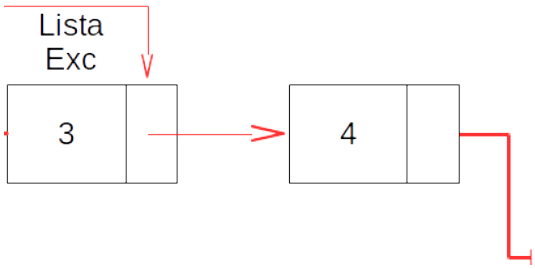


1. **exc** recebe o primeiro valor que lista aponta **(1)**.
2. Em seguida lista recebe o próximo valor **(2)**.
3. Finalizando com **free(exc)** onde o primeiro valor armazenado em **exc**, será apagado.
4. Incrementa **r**, e entra no laço de novo.

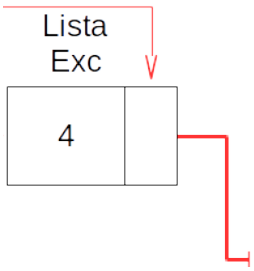
r==1



r==2



r==3



Com **r == 3**, encerra-se então o loop, e o único valor restante na lista é o numero **4**.

3) Considerando a seguinte estrutura referente ao cadastro de alunos em uma Universidade:

```
typedef struct aluno{
    int RA;
    char nome[30];
    float P1, P2, T, APS;
}Aluno;
```

Considerando uma lista duplamente encadeada, implemente as seguintes funções:

a) Incluir um aluno no final da lista.

```
106  // Função para inserir dados no final da lista
107  Aluno* insertAtEnd(Aluno *list){
108      Aluno* new = (Aluno*) malloc (sizeof(Aluno));
109      Aluno* last = list;
110      Aluno al;
111
112      if(new == NULL){
113          printf("\nERRO NA ALOCACAO DE MEMORIA\n");
114          exit(1);
115      }
116
117      insertData(&al);                        // Função para inseri novos dados
118
119                                              // Atribuindo os valores passado ao novo ponteiro da lista
120      new->RA = al.RA;
121      strcpy(new->nome, al.nome);
122      new->P1 = al.P1;
123      new->P2 = al.P2;
124      new->T = al.T;
125      new->APS = al.APS;
126
127
128      if (last == NULL){                      /*Verifica se a lista está vazia,
129                                                se estiver, apenas retorne a nova lista (new).*/
130          return new;
131      } else {                                // Se não for no inicio da lista, a inserção será no final
132
133          if(last->next == NULL){             /*Verifica se é o primeiro elemento da lista,
134                                                se for, o proximo elemento que new aponta (new->next) recebe last->next, ou seja NULL,
135                                                o anterior ao new (new->prev) recebe a posição de last (a primeira nesse caso)
136                                                e o próximo ponteiro que last aponta (last->next) recebe a nova lista (new).*/
137              new->next = last->next;
138              new->prev = last;
139              last->next = new;
140
141              if(last != NULL){               /*Se last é diferente de NULL.
142                                                o ponteiro last (list-> prev), aponta para o novo nó.*/
143                  last->prev = new;
144              }
145              return list;
146          } else {                            /*Caso contrário percorra até o ÚLTIMO elemento da lista,
147                                                antes de NULL, e insira os dados ao próximo elemento
148                                                sempre ao final, realizando a mesma operação da linha 130
149                                                retornando, então, a lista modificada (list)*/
150              while(last->next != NULL){
151                  last = last->next;
152              }
153              new->next = last->next;
154              new->prev = last;
155              last->next = new;
156
157              return list;
158          }
159
160      }
161  }
```

b) Inserir um aluno depois do enésimo elemento da lista. A posição passada como parâmetro.

```c
// Função de inserção normal
Aluno* insert(Aluno *list){
    Aluno *new = (Aluno*) malloc (sizeof(Aluno));
    Aluno al;

    if(new == NULL){
        printf("\nERRO NA ALOCACAO DE MEMORIA\n");
        exit(1);
    }

    insertData(&al);

    new->RA = al.RA;
    strcpy(new->nome, al.nome);
    new->P1 = al.P1;
    new->P2 = al.P2;
    new->T = al.T;
    new->APS = al.APS;

    return new;
}

// Função para inserir um aluno depois do enésimo elemento da lista, informado pelo usuario.
Aluno* insertNelement(Aluno* list){
    Aluno* aux = search(list);      // Um novo ponteiro recebe o retorno da função search()

    if(aux == NULL){                /*Se o retorno for NULL, significa que não foi encotnrado o elemento
                                      retornando assim a lista inalterada (list).*/
        return list;
    }
    Aluno* new = insert(list);      /*Caso tenha encontrado a posição do elemento
                                      um nuvo ponteiro new recebe os dados da função insert.*/

    new->next = aux->next;          /* O proximo elemento que new aponta (new->next) recebe aux->next, ou seja,
                                       o próximo valor que o retorno da função search aponta,
                                       o anterior ao new (new->prev) recebe o conteudo de aux (retorno da função search)
                                       e o próximo ponteiro que aux aponta (aux->next) recebe a nova lista (new).*/
    new->prev = aux;
    aux->next = new;

    if(aux != NULL){
        aux->prev = new;            /*Se o ponteiro aux for diferente de NULL,
                                      então o nó anterior aponta para o novo nó (new).*/
    }

    return list;                    // Retornando a lista alterada
}
```

c) Eliminar o enésimo nó da lista. A posição passada como parâmetro.

```
// Função para buscar um elemento na lista
Aluno* search(Aluno* list){
    int ra;
    Aluno *aux;

    printf("Informe o RA do aluno: ");              // Informa o valor a ser pesquisado
    scanf("%d", &ra);

    for(aux = list; aux != NULL; aux = aux->next){  /*realiza a busca dentro da lista, se encontrar o
                                                      valor igual ao informado, retorne o elemento
                                                      da lista (aux).*/
        if(aux->RA == ra){
            return aux;
        }
    }
    printf("\nAluno não encontrado!!\n");           // Caso contrário, retorne NULL
    return NULL;
}

// Função para remover um elemento, informado pelo usuario, da lista
Aluno* removeElement(Aluno* list){
    Aluno* aux = search(list);                      // Um novo ponteiro recebe o retorno da função search()

    if(aux == NULL){                                /*Se o retorno for NULL, significa que não foi encotnrado o elemento
                                                      retornando assim a lista inalterada (list).*/
        return list;
    }

    if(list == aux){                                /*Se o elemento estiver na primeira posição,
                                                      apenas aponte para a próxima posição*/
        list = aux->next;
    } else {
        aux->prev->next = aux->next;                /*Senão, o nó a ser retirado pode ser o último da lista ou estar entre dois outros nós,
                                                     Assim, o ponteiro next do nó anterior ao nó a ser removido (aux->prev->next)
                                                     aponta para o nó sucessor ao nó a ser retirado (aux->next).*/
    }

    if(aux->next != NULL){                          /*Se o ponteiro next do nó apontado pelo ponteiro aux (aux->next) for diferente de
                                                     NULL, então, o nó a ser removido encontra-se entre dois outros nós da lista.
                                                     Assim, o ponteiro prev do nó sucessor ao nó a ser removido (aux->prev->next)
                                                     aponta para o nó anterior ao nó a ser retirado (aux->prev).*/
        aux->next->prev = aux->prev;
    }
    free(aux);                                      //Desaloca o nó apontado pelo ponteiro
    printf("\n+--------------------+\n");
    printf("|   ALUNO REMOVIDO   |\n");
    printf("+--------------------+\n");
    return list;                                    // Retorna a lista alterada ou inalterada
}
```

4) Implemente uma função que realize a ordenação de uma lista duplamente encadeada.

```c
// Função para ordenar a lista de forma Crescente
void ordenationList(Counter *list) {
    Counter *ptr1, *ptr2;
    int i, j, aux;
    ptr1 = ptr2 = list;                                     // Dois ponteiros que receberão o primeiro valor de list

    if (list == NULL){
        printf("Lista vazia!\n");                           // teste para verificar se a lista esta vazia
        return;
    } else {
        for (i = 0; i < totNodes; i++) {                    /* no primeiro laço totNodes é uma variável global
                                                               que realiza a contagem de cada nó inserido
                                                               usado como indice de parada.*/

                aux = ptr1->number;                         // aux recebe o valor de ptr1

                for (j = 0; j < i; j++)                     // entra no segundo laço

                        ptr2 = ptr2->next;                  //ptr2 aponta para o próximo valor

                for (j = i; j > 0 && ptr2->prev->number > aux; j--) {   /*terceiro laço irá se repetir enquanto j for menor que 0
                                                                           ou o ptr2->prev->number (o valor ponteiro anterior)
                                                                           for menor que aux*/

                        ptr2->number = ptr2->prev->number;  /*o valor de ptr2 passa a ser o
                                                              valor do ponteiro anterior de ptr2*/

                        ptr2 = ptr2->prev;                  // ptr2 recebe o posição do ponteiro (ptr2) anterior
                }
                ptr2->number = aux;                         /*fora do laço ptr2 receber o valor de aux,
                                                              no qual armazena o valor de ptr1 */

                ptr2 = list;                                // ptr2 recebe list novamente

                ptr1 = ptr1->next;                          /*incrementa ptr1 para a proxima posição da lista
                                                              e assim recomeçar o laço novamente*/
        }
    }
    printf("\n+--------------------+\n");
    printf("|   LISTA ORDENADA   |\n");                     // Fim da ordenação
    printf("+--------------------+\n");
}
```

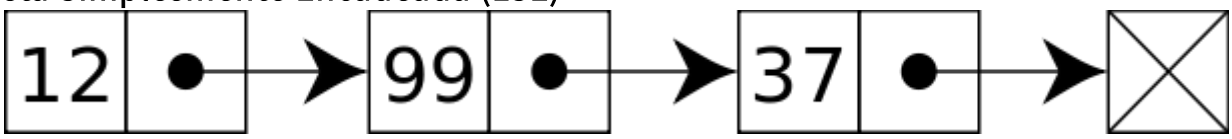
As estruturas: lista encadeada e lista duplamente encadeada são semelhantes;

- implemente as mesmas operações adicionais que foram explicadas na aula passada:
    - encontrar o maior elemento;
    - encontrar o menor elemento;
    - inserir de forma ordenada;
    - inserir no fim de uma lista;
    - remover do início de uma lista;
    - remover do fim de uma lista;
- escreva alguns parágrafos dizendo: o fato da lista ser duplamente encadeada pode melhorar o custo de alguma das operações? por que?

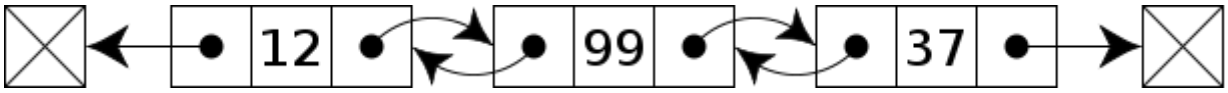
RESP.
Uma lista duplamente encadeada , cada elemento vai possuir um ponteiro que aponta para o elemento anterior e ao próximo, nos permitindo realizar busca e percorrer por ambas as direções. Além da inclusão e remoção ser realizadas mais facilmente, porque os nós não precisam ser armazenados na memória de forma sequencial, como na lista simplesmente encadeada.

Exemplo:

Lista Simplesmente Encadeada (LSE)



Lista Duplamente Encadeadas (LDE)



Na LSE, cada nó possui um valor e um ponteiro para o próximo nó, o primeiro nó poderia estar na posição de memória "3333", já o segundo poderia estar na posição

"4444" e o terceiro na posição "5555", sendo assim caso você precise adicionar um novo nó com o valor "6666" e ele precise estar na segunda posição, seria necessário apenas alterar o ponteiro do primeiro nó para que aponte para o novo nó, e neste novo nó você colocaria o endereço do ponteiro anterior do primeiro nó.

Caso você precise apagar o nó com o valor "37" tudo que precisaria fazer é alterar o ponteiro do nó anterior ("99") para que aponte para o nó que está a frente de "37".

Quanto a LDE, nelas você pode tanto percorrer a lista a partir do início quanto a partir do fim, em qualquer nó que esteja pode seguir quaisquer um dos caminhos. Para realizar operações de inserção e remoção, modificamos apenas os ponteiros que apontam para o elemento que queira inserir/remover e ponteiro que aponta para o antecessor a esse elemento .

OBS.: Todos os códigos seguirão em anexo para a execução.