Aula 10 – Exercício com Header (Descritor):

- A atividade é individual e prática, deverá ser implementada na Linguagem C;
- Utilize as estruturas vistas em aula, para os descritores das LSE e LDE;
- Identifique corretamente as Listas, pois iremos trabalhar com mais de uma Lista;
- A atividade deverá ser entregue até o dia 01 de outubro de 2017, até as 19:00, pelo Moodle;
- Organize seu tempo e Bom Trabalho!!!!

Exercício:

1. Implemente um programa em Linguagem C, com as seguintes Funções (as mesmas devem ser chamadas pela main – não faça menu, as Funções devem ser executadas na ordem do enunciado – exceto a impressão que deverá ser executada a cada modificação das Listas – após a chamada das funções):

a) (0.5 Pontos) Função para criar o header da LSE (descritor);

b) (0.5 Pontos) Função para criar o header da LDE (descritor);

c) (0.5 Pontos) Função para armazenar em uma LSE com Header à direita (no final da lista), os 50 valores sorteados na main [0-99];

d) (0.5 Pontos) Função para armazenar em uma LDE com Header à esquerda (no início da lista), os 50 valores sorteados na main (os mesmos valores sorteados para a letra c – use o mesmo laço);

e) (0.5 Pontos) Função para imprimir a LSE com Header;

f) (0.5 Pontos) Função para imprimir a LDE com Header;

g) (2.0 Pontos) Função que receba um novo valor, que deve ser inserido (nodo) antes do nodo que contém o maior elemento da LSE (implemente uma função auxiliar para descobrir o maior elemento da lista);

h) (2.0 Pontos) Função que receba um valor e exclua o nodo que contém esse valor da LDE – se o valor não estiver presente na lista o usuário deverá ser informado;

i) (1.0 Ponto) Função que insira um nodo no início da LDE. O valor desse nodo deverá ser o maior valor da LSE;

j) (0.5 Pontos) Função que retorne para a main a quantidade de nodos de cada uma das Listas, e imprima para o usuário com a devida identificação das listas;

k) (1.0 Ponto) Função que a partir da escolha do usuário, remove a LSE ou a LDE (não esqueça do descritor).

l) (0.5 Pontos) Função main que sorteia e realiza as chamadas de funções (identação, legibilidade, bons hábitos de programação).

```
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
#include<conio.h>
#define TAM 50

//Estrutura da LSE
struct nodo{
 int dados;
 struct nodo *prox;
 };
//Estrutura da LDE
struct nodod{
 struct nodod *ant;
 int dados;
 struct nodod *prox;
 };

//Estrutura do Descritor LSE
struct headerLSE{
 struct nodo *inicio;
 int qtde;
 struct nodo *fim;
 };

//Estrutura do Descritor LDE
struct headerLDE{
 struct nodod *inicio;
 int qtde;
 struct nodod *fim;
 };

//Função para criar o header LSE (descritor)
int Cria_headerLSE(struct headerLSE **lista){
 struct headerLSE *novo=NULL;
 novo=(struct headerLSE *)malloc(sizeof(struct headerLSE));
 if(novo!=NULL){
  novo->qtde=0;
  novo->inicio=NULL;
  novo->fim=NULL;
  }else printf("Nao foi possivel alocar!\n");
 *lista=novo;
 }

//Função para criar o header LDE (descritor);
int Cria_headerLDE(struct headerLDE **lista){
 struct headerLDE *novo=NULL;
 novo=(struct headerLDE *)malloc(sizeof(struct headerLDE));
 if(novo!=NULL){
  novo->qtde=0;
  novo->inicio=NULL;
  novo->fim=NULL;
  }else printf("Nao foi possivel alocar!\n");
 *lista=novo;
 }
```

```
/*Função para armazenar em uma LSE com Header à direita (no fim da lista), os 15
valores sorteados na main;*/
int InsereDireitaLSE(struct headerLSE **lista,int valor){
 struct nodo *novo=NULL;
 novo=(struct nodo *)malloc(sizeof(struct nodo));
 if(novo!=NULL){
  novo->dados=valor;
  novo->prox=NULL;
  if((*lista)->inicio==NULL)(*lista)->inicio=novo;
   else (*lista)->fim->prox=novo;
  ((*lista)->qtde)++;
  (*lista)->fim=novo;
  }else printf("\nNao foi possivel alocar!\n");
 }

/*Função para armazenar em uma LDE com Header à esquerda (no início da lista), os
15 valores sorteados na main;*/
int InsereEsquerdaLDE(struct headerLDE **lista,int valor){
 struct nodod *novo=NULL;
 novo=(struct nodod *)malloc(sizeof(struct nodod));
 if(novo!=NULL){
  novo->dados=valor;
  novo->ant=NULL;
  if((*lista)->inicio==NULL){
   (*lista)->fim=novo;
    novo->prox=NULL;
    }else{
     (*lista)->inicio->ant=novo;
     novo->prox=(*lista)->inicio;
     }
  ((*lista)->qtde)++;
  (*lista)->inicio=novo;
  }else printf("\nNao foi possivel Alocar!\n");
 }

//Função para imprimir a LSE com Header;
int ImprimeLSE(struct headerLSE **lista){
 struct nodo *aux=NULL;
 if((*lista)->inicio!=NULL){
  aux=(*lista)->inicio;
  while(aux!=NULL){
   printf("%i\t",aux->dados);
   aux=aux->prox;
   }
  }else printf("\nLista Vazia!\n");
 }

//Função para imprimir a LDE com Header;
int ImprimeLDE(struct headerLDE **lista){
 struct nodod *aux=NULL;
 if((*lista)->inicio!=NULL){
  aux=(*lista)->inicio;
  while(aux!=NULL){
   printf("%i\t",aux->dados);
   aux=aux->prox;
   }
```

```
  }else printf("\nLista Vazia!\n");
 }

//Função para pesquisar o maior valor da LSE
int PesquisaMaior(struct headerLSE **lista){
 struct nodo *aux=NULL;
 int maior=0;
 if((*lista)->inicio!=NULL){
  aux=(*lista)->inicio;
  maior=(*lista)->inicio->dados;
  while(aux!=NULL){
   if(aux->dados>maior)maior=aux->dados;
   aux=aux->prox;
   }
  }else printf("\nLista Vazia!\n");
 return (maior);
 }

/*Função que receba um novo valor, que deve ser inserido (nodo) antes do nodo que
contém o maior elemento da LSE (implemente uma função auxiliar para descobrir o
maior elemento da lista);*/
int InsereMaiorLSE(struct headerLSE **lista,int valor){
 struct nodo *novo=NULL,*aux=NULL,*ant=(*lista)->inicio;
 int maior;
 if((*lista)->inicio!=NULL){
  maior=PesquisaMaior(&*lista);
  novo=(struct nodo *)malloc(sizeof(struct nodo));
  if(novo!=NULL){
   novo->dados=valor;
   aux=(*lista)->inicio;
   while(aux!=NULL){
    if(aux->dados==maior){
     if(aux==(*lista)->inicio){
      novo->prox=aux;
      (*lista)->inicio=novo;
      }else{
       novo->prox=aux;
       ant->prox=novo;
       }
     break;
     }
    ant=aux;
    aux=aux->prox;
    }
   }else printf("\nNao foi possivel alocar!\n");
  }else printf("\nLista Vazia!\n");
 }

/*Função que receba um valor e exclua o nodo que contem esse valor da LDE - se o
valor não estiver presente na lista o usuário deverá ser informado;*/
int ExcluiNodoLDE(struct headerLDE **lista,int valor){
 struct nodod *aux=NULL,*del=NULL;
 int achou=0;
 if((*lista)->inicio!=NULL){
  if((*lista)->inicio->dados==valor){
   del=(*lista)->inicio;
   (*lista)->inicio=(*lista)->inicio->prox;
```

```
   (*lista)->inicio->ant=NULL;
   achou++;
   }else{
    if((*lista)->fim->dados==valor){
     del=(*lista)->fim;
     (*lista)->fim=(*lista)->fim->ant;
     (*lista)->fim->prox=NULL;
     achou++;
     }else{
      aux=(*lista)->inicio->prox;
      while(aux->prox!=NULL){
       if(aux->dados==valor){
        del=aux;
        aux->ant->prox=aux->prox;
        aux->prox->ant=aux->ant;
        achou++;
        break;
        }
       aux=aux->prox;
       }
      }
    }
  free(del);
  ((*lista)->qtde)--;
  }else printf("\nLista Vazia!\n");
 return(achou);
 }

/*Função que insira um nodo no início da LDE. O valor desse nodo deverá ser o
maior valor da LSE;*/
int InsereNodoInicioLDE(struct headerLSE **lista,struct headerLDE **listaD){
 struct nodod *novo=NULL;
 int valor;
 if(((*lista)->inicio!=NULL)&&((*listaD)->inicio!=NULL)){
  valor=PesquisaMaior(&*lista);
  novo=(struct nodod *)malloc(sizeof(struct nodod));
  if(novo!=NULL){
   novo->dados=valor;
   novo->ant=NULL;
   novo->prox=(*listaD)->inicio;
   (*listaD)->inicio->ant=novo;
   (*listaD)->inicio=novo;
   }else printf("\nNao foi possivel alocar!\n");
  }else printf("\nLista(s) Vazia(s)!\n");
 }

/*Função que retorne para a main a quantidade de nodos de cada uma das Listas,
informando ao usuário devidamente identificado;*/
int InformaQtdeLSE(struct headerLSE **lista){
 return((*lista)->qtde);
 }

int InformaQtdeLDE(struct headerLDE **lista){
 return((*lista)->qtde);
 }
```

```
/*Função que a partir da escolha do usuário, remove a LSE ou a LDE (não esqueça
do descritor).*/
int RemoveLSE(struct headerLSE **lista){
 struct nodo *aux=(*lista)->inicio,*del=NULL;
 if((*lista)->inicio!=NULL){
  while(aux!=NULL){
   del=aux;
   aux=aux->prox;
   free(del);
   }
  free(*lista);
  }else printf("\nLista Vazia!\n");
 }

int RemoveLDE(struct headerLDE **lista){
 struct nodod *aux=(*lista)->inicio,*del=NULL;
 if((*lista)->inicio!=NULL){
  while(aux!=NULL){
   del=aux;
   aux=aux->prox;
   free(del);
   }
  free(*lista);
  }else printf("\nLista Vazia!\n");
 }

//Função Principal
int main(){
 struct headerLSE *listaS=NULL;
 struct headerLDE *listaD=NULL;
 int i,nro,achou;
 Cria_headerLSE(&listaS);
 Cria_headerLDE(&listaD);
 for(i=0;i<TAM;i++){
  nro=rand()%51;
  InsereDireitaLSE(&listaS,nro);
  InsereEsquerdaLDE(&listaD,nro);
  }
 printf("\n--- Lista Simplemente Encadeada com Header ---\n\n");
 ImprimeLSE(&listaS);

 printf("\n--- Lista Duplamente Encadeada com Header ---\n\n");
 ImprimeLDE(&listaD);

 InsereMaiorLSE(&listaS,100);
 printf("\n--- Lista Simplemente Encadeada com Header Apos a Insercao Antes do Maior ---\n\n");
 ImprimeLSE(&listaS);

 printf("\n\nDigite um valor para excluir o nodo da LDE: ");
 scanf("%i",&nro);
 achou=ExcluiNodoLDE(&listaD,nro);
 if(achou==1)printf("\nNodo com o valor %i excluido com sucesso!\n",nro);
  else printf("\nNodo com o valor %i nao existe na Lista!\n",nro);
 printf("\n--- Lista Duplamente Encadeada com Header Apos a Exclusao de %i ---\n\n",nro);
 ImprimeLDE(&listaD);

 InsereNodoInicioLDE(&listaS,&listaD);
```

```
printf("\n\n--- LDE com Header Apos a insercao do primeiro nodo com o maior valor da LSE ---\n\n");
ImprimeLDE(&listaD);

printf("\nA LSE com Header possui %i nodos.\n",InformaQtdeLSE(&listaS));
printf("\nA LDE com Header possui %i nodos.\n",InformaQtdeLDE(&listaD));

printf("\nEscolha a Opcao para Exclusao das Listas:\n");
printf("\n1 - LSE\n2 - LDE\n3 - Nenhuma\nOpcao: ");
scanf("%i",&nro);
switch(nro){
 case 1:{RemoveLSE(&listaS); printf("\nRemovida com Sucesso!\n"); break;}
 case 2:{RemoveLDE(&listaD); printf("\nRemovida com Sucesso!\n"); break;}
 case 3:{printf("\nNenhuma Lista sera Removida!\n"); break;}
 default: printf("\nOpcao Invalida!\n");
 }
getch();
}
```