Atividade 7 - Listas Duplamente Encadeadas e Pilhas (T1)

15 de outubro de 2025

Problema A

```c
void Insere(TipoItem x, TipoLista *Lista)
{
  /* --- Insere no final ---*/
  Lista->Ultimo->Prox = (TipoApontador) malloc(sizeof(TipoCelula));
  TipoApontador NovoElemento = Lista->Ultimo->Prox;

  NovoElemento->Ant = Lista->Ultimo;
  NovoElemento->Item = x;
  NovoElemento->Prox = NULL;

  Lista->Ultimo = NovoElemento;
}

void Retira(TipoChave chave, TipoLista *Lista, TipoItem *Item)
{
  /* --- Remove o primeiro elemento da lista que contem a chave
     especificada --- */
  TipoApontador Aux;

  if (Vazia(*Lista))
  {
      printf(" Erro: Lista vazia\n");
      return;
  }

  // Percorre a lista procurando pela chave
  Aux = Lista->Primeiro;

  while (Aux != NULL && Aux->Item.Chave != chave)
  {
      Aux = Aux->Prox;
  }

  // Se nao encontrou a chave
  if (Aux == NULL)
  {
      printf(" Erro: Chave %d nao encontrada\n", chave);
      return;
  }

```

```
40    if(Aux->Ant && Aux->Prox){
41        Aux->Ant->Prox = Aux->Prox;
42        Aux->Prox->Ant = Aux->Ant;
43    } else if (Aux->Ant){
44        Aux->Ant->Prox = NULL;
45        Lista->Ultimo = Aux->Ant;
46    } else if (Aux->Prox){
47        Aux->Prox->Ant = NULL;
48        Lista->Primeiro = Aux->Prox;
49    } else {
50        Lista->Primeiro = NULL;
51        Lista->Ultimo = NULL;
52    }
53
54    free(Aux);
55 }
56
57 void ImprimeReverso(TipoLista Lista)
58 {
59    TipoApontador Aux;
60
61    if (Vazia(Lista))
62    {
63        printf("\n");
64        return;
65    }
66
67    Aux = Lista.Ultimo;
68
69    // Percorre do ultimo ate o primeiro elemento (depois da celula cabeca
        )
70    while (Aux != Lista.Primeiro)
71    {
72        printf("%d ", Aux->Item.Chave);
73        Aux = Aux->Ant;
74    }
75    printf("\n");
76 }
```

Problema B

```
void Desempilha(TipoPilha *Pilha, TipoItem *Item)
{
    *Item = Pilha->Topo->Prox->Item;
    TipoApontador Aux = Pilha->Topo->Prox->Prox;

    if(Pilha->Fundo == Pilha->Topo->Prox)
        Pilha->Fundo = Pilha->Topo;

    free(Pilha->Topo->Prox);
    Pilha->Tamanho -= 1;
    Pilha->Topo->Prox = Aux;
}
```

Problema C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct No {
    struct No* anterior;
    char c;
} No;

typedef struct Pilha {
    No* topo;
} Pilha;

Pilha* criaPilha(void){
    Pilha* pilha = (Pilha*) malloc (sizeof(Pilha));

    if(!pilha){
        printf("Erro! Nao foi possivel alocar a pilha\n");
        return NULL;
    }

    pilha->topo = NULL;
    return pilha;
}

void inserePilha(Pilha *pilha, char c){
    if(!pilha){
        printf("Erro! A pilha nao existe!\n");
        return;
    }

    No* aux = (No*) malloc (sizeof(No));

    if(!aux){
        printf("Erro! Nao foi possivel alocar o novo no para a pilha!\n"
            );
        return;
    }

    aux->c = c;
    aux->anterior = pilha->topo;
    pilha->topo = aux;
}

void removePilha(Pilha* pilha){
    if(!pilha){
        printf("Erro! A pilha nao existe!\n");
        return;
    }

    if(!pilha->topo){
        printf("Erro! A pilha esta vazia!\n");
        return;
    }

    No* aux = pilha->topo->anterior;
```

```
56      free(pilha->topo);
57      pilha->topo = aux;
58  }
59
60  void destroiPilha(Pilha *pilha){
61      if(!pilha) return;
62
63      while(pilha->topo){
64          removePilha(pilha);
65      }
66
67      free(pilha);
68  }
69
70  int main(){
71      int n;
72      scanf("%d\n", &n);
73
74      for(int i = 0; i < n; ++i){
75          int resposta = 0;
76
77          char str[1001];
78          scanf("%[^\n]\n", str);
79
80          int tam = strlen(str);
81
82          Pilha* pilha = criaPilha();
83
84          for(int i = 0; i < tam; ++i){
85              if(str[i] == '<'){
86                  inserePilha(pilha, str[i]);
87              } else if (str[i] == '>' && pilha->topo){
88                  removePilha(pilha);
89                  resposta += 1;
90              }
91          }
92
93          printf("%d\n", resposta);
94
95          destroiPilha(pilha);
96      }
97      return 0;
98  }
```

Problema D

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct No {
    int valor;
    struct No* anterior;
} No;

typedef struct Pilha {
    No* topo;
} Pilha;

Pilha* criaPilha(void){
    Pilha *pilha = (Pilha*) malloc (sizeof(Pilha));

    if(!pilha){
        printf("Erro! Nao foi possivel alocar a pilha!\n");
        return NULL;
    }
    pilha->topo = NULL;

    return pilha;
}

void inserePilha(Pilha* pilha, int valor){
    if(!pilha){
        printf("Erro! A pilha nao existe!\n");
        return;
    }

    No* aux = (No*) malloc (sizeof(No));
    aux->valor = valor;
    aux->anterior = pilha->topo;
    pilha->topo = aux;
}

void removePilha(Pilha* pilha){
    if(!pilha){
        printf("Erro! A pilha nao existe!\n");
        return;
    }

    if(!pilha->topo){
        printf("Erro! A pilha esta vazia!\n");
        return;
    }

    No* aux = pilha->topo->anterior;
    free(pilha->topo);
    pilha->topo = aux;
}

void destroiPilha(Pilha* pilha){
    if(!pilha) return;

    while(pilha->topo){
```

```
57          No* aux = pilha->topo->anterior;
58          free(pilha->topo);
59          pilha->topo = aux;
60      }
61
62      free(pilha);
63  }
64
65  int main(){
66      int n;
67      scanf("%d", &n);
68
69      Pilha *pilha = criaPilha();
70
71      for(int i = 0; i < n; ++i){
72          int operacao;
73          scanf("%d", &operacao);
74
75
76          switch(operacao){
77              case 0: {
78                  printf("%d\n", pilha->topo->valor);
79                  break;
80              }
81              case 1: {
82                  int valor;
83                  scanf("%d", &valor);
84
85                  if(pilha->topo){
86                      valor = pilha->topo->valor < valor ? pilha->topo->
                            valor : valor;
87                  }
88
89                  inserePilha(pilha, valor);
90                  break;
91              }
92              case 2: {
93                  removePilha(pilha);
94                  break;
95              }
96          }
97      }
98
99      destroiPilha(pilha);
100
101     return 0;
102 }
```