TAD Pilha de float e módulos Lista. Função inverte.

lista.h

```
struct elemento
{
   float info;
   struct elemento * prox;
};
typedef struct elemento Elemento;

Elemento * lista_insere(Elemento * n, float a);
Elemento * lista_retira(Elemento * n, float a);
void lista_libera(Elemento * n);
void lista_imprime(Elemento * n);
```

pilha.h

```
typedef struct pilha Pilha;

/* funcoes Pilha */
Pilha * pilha_cria(void);
void pilha_push(Pilha * p, float a);
float pilha_pop(Pilha * p);
int pilha_vazia(Pilha * p);
void pilha_libera(Pilha * p);
void pilha_imprime(Pilha * p);
```

lista.c

```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

Elemento * lista_insere(Elemento * n, float a)
{
   Elemento * p = (Elemento *)malloc(sizeof(Elemento));
   if (p==NULL) return NULL;
   p->info=a;
   p->prox=n;
   return p;
}

Elemento * lista_retira(Elemento * n, float a)
{
   Elemento * t;
   if(n!=NULL)
   {
      if (n->info==a)
      {
         t = n;
         n = n->prox;
         free(t);
      }
      else
      {
         n->prox= lista_retira(n->prox,a);
      }
   }
   return n;
}

void lista_libera(Elemento * n)
{
   if (n!=NULL)
   {
      lista_libera(n->prox);
      free(n);
   }
}
```

```c
void lista_imprime(Elemento * n)
{
   if (n!=NULL)
   {
      printf("%f\n",n->info);
      lista_imprime(n->prox);
   }
}
```

pilha.c

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

// Piha como Lista
/*
struct elemento
{
   float info;
   struct elemento * prox;
};
typedef struct elemento Elemento;

struct pilha
{
   Elemento * prim;
};

Pilha * pilha_cria(void)
{
   Pilha * p = (Pilha *) malloc(sizeof(Pilha));
   p->prim = NULL;
   return p;
}

int pilha_vazia(Pilha * p)
{
   return (p->prim == NULL);
}

void pilha_push(Pilha * p, float a)
{
   Elemento * novo = (Elemento *)malloc(sizeof(Elemento));
   if (novo==NULL) exit(1);
   novo->info = a;
   novo->prox = p->prim;
   p->prim = novo;
}

float pilha_pop(Pilha * p)
{
   Elemento * t;
   float a;
   if (pilha_vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1);
   }
   t = p->prim;
   a = t->info;
   p->prim = t->prox;
   free(t);
   return a;
}

void pilha_libera(Pilha * p)
{
```

```
   Elemento *t, * q = p->prim;
   while (q != NULL)
   {
      t = q->prox;
      free(q);
      q = t;
   }
   free(p);
}

void pilha_imprime(Pilha * p)
{
   Elemento * q;
   for (q=p->prim; q != NULL; q=q->prox)
      printf("%f\n",q->info);
}
*/
```

pilhaVetor.c

```
#include <stdio.h>
#include <stdlib.h>
#include "pilha.h"

// Pilha como vetor

#define N 100

struct pilha
{
   int n;
   float v[N];
};

Pilha * pilha_cria(void)
{
   Pilha * p = (Pilha *) malloc(sizeof(Pilha));
   if (p==NULL) return NULL;
   p->n = 0;   // inicializa com 0 elementos
   return p;
}

int pilha_vazia(Pilha * p)
{
   return (p->n == 0);
}

void pilha_push(Pilha * p, float a)
{
   if (p->n == N)
   {
      printf("Capacidade da pilha estourou.\n");
      exit(1);
   }
   p->v[p->n] = a;
   p->n++;              // incrementa numero de elementos
}

float pilha_pop(Pilha * p)
{
   float a;
   if (pilha_vazia(p))
   {
      printf("Pilha vazia.\n");
      exit(1);
   }
   a = p->v[p->n-1];
```

```
   p->n--;
   return a;
}

void pilha_libera(Pilha * p)
{
   free(p);
}

void pilha_imprime(Pilha * p)
{
   int i;
   for (i=p->n-1; i>=0; i--)  // topo e' v[n-1]
      printf("%f\n",p->v[i]);
}
```

inverte.c

```
#include <stdio.h>
#include "lista.h"
#include "pilha.h"

Elemento * inverte(Elemento * n)
{
  Pilha * p;
  Elemento * t;
  p = pilha_cria();
  for (t=n; t!=NULL; t = t->prox)
     pilha_push (p, t->info);
  for (t=n; t!=NULL; t = t->prox)
     t->info = pilha_pop(p);
  pilha_libera(p);
  return n;
}
```

prog1.c

```
#include <stdio.h>
#include "lista.h"
#include "pilha.h"
Elemento * inverte(Elemento * n);

int main(void)
{
   Pilha * p;
   Elemento * n;

   // Teste do TAD Pilha
   p = pilha_cria();
   pilha_push(p,2.0);
   pilha_push(p,4.0);
   pilha_push(p,6.0);
   pilha_push(p,8.0);
   printf("Pilha:\n");
   pilha_imprime(p);
   pilha_libera(p);
   // Teste do modulo Lista
   n = NULL;
   n = lista_insere(n,2.0);
   n = lista_insere(n,4.0);
   n = lista_insere(n,6.0);
   n = lista_insere(n,8.0);
   printf("Lista:\n");
   lista_imprime(n);

   // Inverter Lista
   printf("Lista invertida:\n");
   lista_imprime(inverte(n));

   return 0;
}
```