Pilha - Alocação Sequencial

Com base no que foi visto implemente a operação top_pop() que compõem o TAD PILHA_SEQ.

```
1  typedef struct
2  {
3      int TOPO;
4      int VAL[MAX];
5  }PILHA_SEQ;
6  void cria_pilha (PILHA_SEQ *);
7  int eh_vazia (PILHA_SEQ *);
8  void push (PILHA_SEQ *, int);
9  int top (PILHA_SEQ *);
10 void pop (PILHA_SEQ *);
11 int top_pop (PILHA_SEQ *);
```

UNIVASF

```c
int top_pop (PILHA_SEQ *p)
{
    if (eh_vazia(p))
    {
        printf ("\nERRO! Consulta e retirada na pilha vazia.\n");
        exit (4);
    }
    else
        return (p->VAL[p->TOPO--]);
}
```

Pilha - Alocação Sequencial

Utilizando-se dos TAD´s FILA_SEQ e PILHA_SEQ, vistos anteriormente, implemente a seguinte operação:

void inverte_fila (FILA_SEQ *f);

a qual recebe uma referência para uma fila sequencial de inteiros e inverte a ordem de seus elementos, utilizando-se para isto de uma pilha sequencial.

```c
void inverte_fila (FILA_SEQ *f)
{
    PILHA_SEQ p;
    cria_pilha(&p);
    while (!eh_vazia(f))
        push(&p,cons_ret(f));
    while (!eh_vazia_pilha(&p))
        ins(f, top_pop(&p));
}
```

Pilhas
Alocação Encadeada

UNIVASF

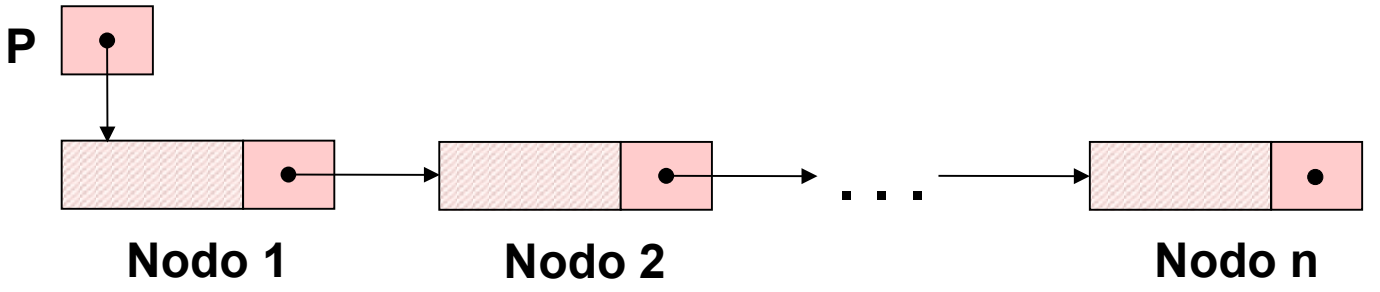
Pilha - Alocação Encadeada

Como já discutimos, a alocação sequencial apresenta algumas desvantagens.

Em virtude disso, podemos nós utilizar de uma lista encadeada para armazenarmos uma pilha, assim como fizemos com as filas.

Como todas as operações ocorrem numa das extremidades da lista, a representação da pilha se reduz a um único ponteiro para o primeiro nodo da lista.

Pilha - Alocação Encadeada



A implementação das operações é trivial.

Desta forma, definiremos e implementaremos, agora, o TAD PILHA_ENC (de valores inteiros).

```c
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODE;
typedef NODE * PILHA_ENC;
void create (PILHA_ENC *);
int is_empty (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
void destroy (PILHA_ENC);
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação create() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```
1   void create (PILHA_ENC *pp)
2   {
3       *pp=NULL;
4   }
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação is_empty() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
int is_empty (PILHA_ENC p)
{
    return (!p);
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação push() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
void push (PILHA_ENC *pp, int v)
{
    NODE *new;
    new = (NODE *) malloc (sizeof(NODE));
    if (!new)
    {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (1);
    }
    new->inf = v;
    new->next = *pp;
    *pp=new;
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação top() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
int top (PILHA_ENC p)
{
    if (!p)
    {
        printf ("\nERRO! Consulta em pilha vazia!\n");
        exit (2);
    }
    return (p->inf);
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação pop() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

UNIVASF

```c
void pop (PILHA_ENC *pp)
{
    if (!(*pp))
    {
        printf ("\nERRO! Retirada em pilha vazia!\n");
        exit (3);
    }
    else
    {
        NODE *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
    }
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação top_pop() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
int top_pop (PILHA_ENC *pp)
{
    if (!(*pp))
    {
        printf ("\nERRO! Consulta e retirada em pilha vazia!\n");
        exit (4);
    }
    else {
        int v=(*pp)->inf;
        NODE *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
        return (v);
    }
}
```

Pilha - Alocação Encadeada

Com base no que foi visto implemente a operação destroy() que compõem o TAD PILHA_ENC.

```
 1  typedef struct nodo
 2  {
 3      int inf;
 4      struct nodo * next;
 5  }NODO;
 6  typedef NODO * PILHA_ENC;
 7  void create (PILHA_ENC *);
 8  int is_empty (PILHA_ENC);
 9  void push (PILHA_ENC *, int);
10  int top (PILHA_ENC);
11  void pop (PILHA_ENC *);
12  int top_pop (PILHA_ENC *);
13  void destroy (PILHA_ENC);
```

```c
void destroy (PILHA_ENC l)
{
    PILHA_ENC aux;
    while (l)
    {
        aux = l;
        l = l->next;
        free(aux);
    }
}
```