Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
void cria_pilha (PILHA_SEQ *p)
{
   p->TOPO = -1;
}
```

Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
int eh_vazia (PILHA_SEQ *p)
{
   return (p->TOPO == -1);
}
```

Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
void push (PILHA_SEQ *p, int v)
{
   if (p->TOPO == MAX-1)
   {
      printf ("\nERRO! Estouro na pilha.\n");
      exit (1);
   }
   p->VAL[++(p->TOPO)]=v;
}
```

Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
int top (PILHA_SEQ *p)
{
   if (eh_vazia(p))
   {
      printf ("\nERRO! Consulta na pilha vazia.\n");
      exit (2);
   }
   else
      return (p->VAL[p->TOPO]);
}
```

Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
void pop (PILHA_SEQ *p)
{
   if (eh_vazia(p))
   {
      printf ("\nERRO! Retirada na pilha vazia.\n");
      exit (3);
   }
   else
      p->TOPO--;
}
```

Pilha - Alocação Sequencial

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

```
typedef struct
{
   int TOPO; /*índice do topo da pilha*/
   int VAL[MAX];    /*vetor de elementos*/
}PILHA_SEQ;
void cria_pilha (PILHA_SEQ *);
int eh_vazia (PILHA_SEQ *);
void push (PILHA_SEQ *, int);
int top (PILHA_SEQ *);
void pop (PILHA_SEQ *);
int top_pop (PILHA_SEQ *);
```

```
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 - Exercício

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.

UNIVASF

```
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));
}
```

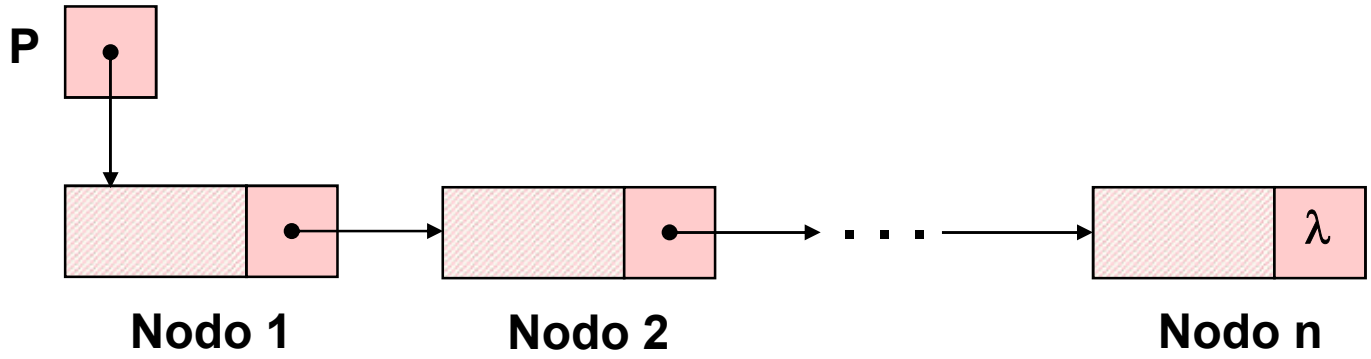
Pilha - Alocação Encadeada

Com já discutimos, a alocação sequência 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.

UNIVASF

Alocação Encadeada



A implementação das operações é trivial. Para fazer uma inserção, basta alocar um nodo para o novo valor, ligá-lo ao primeiro nodo da lista e fazer o ponteiro apontar para o novo nodo. Uma retirada exige apenas que o ponteiro passe a apontar para o segundo nodo da lista (ou ser anulado, se houver

Alocação Encadeada

apenas um nodo). Uma consulta exige apenas a recuperação do valor do primeiro nodo. OBS. : em uma retirada o espaço de memória ocupado pelo primeiro nodo deve ser liberado.

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

UNIVASF

```
/*Definição do TAD PILHA_ENC*/
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

Pilha - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

```
void cria_pilha (PILHA_ENC *pp)
{
   *pp=NULL;
}
```

Pilha - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

```
int eh_vazia (PILHA_ENC p)
{
   return (!p);
}
```

Pilha - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

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

Pilha - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

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

Pilha - Alocação Encadeada

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

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

```
void pop (PILHA_ENC *pp)
{
   if (eh_vazia(*pp))
   {
      printf ("\nERRO! Retirada em pilha vazia!\n");
      exit (3);
   }
   else
   {
      NODO *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.

```
typedef struct nodo
{
   int inf;
   struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
```

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