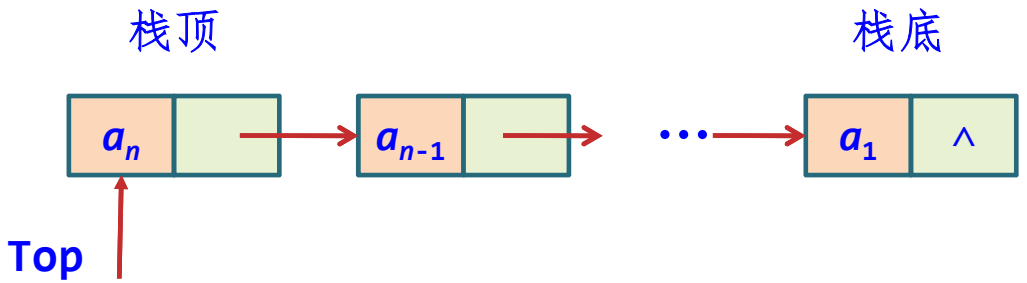
3.1.3 栈的链式存储结构

- 栈的链式存储结构是采用某种链表结构，栈的链式存储结构简称为链栈。
- 这里采用单链表作为链栈，该单链表是不带头结点的。



◆链栈的结点类型声明如下：

```
typedef char ElemType;
typedef struct node {
  ElemType data;
  struct node *next;
} LinkStack;
```

- 归纳起来，链栈top初始时top=NULL，其4个要素如下：

  - 栈空条件：top==NULL
  - 栈满条件：不考虑
  - 元素*x*进栈操作：创建存放元素*x*的结点*p*，将其插入到栈顶位置上
  - 出栈元素*x*操作：置*x*为栈顶结点的data域，并删除该结点

链栈的基本算法

（1）初始化栈运算算法

主要操作：用top==NULL标识栈为空栈。

```
void InitStack(LinkStack *&top)
{
    top=NULL;
}
```

（2）销毁栈算法

链栈的所有结点空间都是通过malloc函数分配的，在不再需要时需通过free函数释放所有结点的空间。

```
void DestroyStack(LinkStack *&top)
{  LinkStack *pre=top,*p;
   if (pre==NULL) return;     //考虑空栈的情况
   p=pre->next;
   while (p!=NULL)
   {  free(pre);              //释放pre结点
      pre=p;
      p=p->next;              //pre、p同步后移
   }
   free(pre);                 //释放尾结点
}
```

（3）进栈算法

主要操作：先创建一个新结点，其data域值为*x*；然后将该结点插入到top结点之后作为栈顶结点。

```
int Push(LinkStack *&top,ElemType x)
{
   LinkStack *p;
   p=(LinkStack *)malloc(sizeof(LinkStack));
   p->data=x;          //创建结点p用于存放x
   p->next=top;        //插入p结点作为栈顶结点
   top=p;
   return 1;
}
```

（4）出栈算法

主要操作：将栈顶结点（即top->next所指结点）的data域值赋给x，然后删除该栈顶结点。

```
int Pop(LinkStack *&top,ElemType &x)
{
   LinkStack *p;
   if (top==NULL)                  //栈空,下溢出返回0
      return 0;
   else                            //栈不空时出栈元素x并返回1
   {   p=top;                      //p指向栈顶结点
       x=p->data;                  //取栈顶元素x
       top=p->next;                //删除结点p
       free(p);                    //释放p结点
       return 1;
   }
}
```

（5）取栈顶元素算法

主要操作：将栈顶结点（即top->next所指结点）的data域值赋给x。

```
int GetTop(LinkStack *top,ElemType &x)
{
   if (top==NULL)      //栈空,下溢出时返回0
     return 0;
   else                //栈不空,取栈顶元素x并返回1
   {  x=top->data;
      return 1;
   }
}
```

（6）判断栈空算法

主要操作：若栈为空（即top->next==NULL）则返回值1，否则返回值0。

```
int StackEmpty(LinkStack *top)
{
   if (top==NULL)
      return 1;
   else
      return 0;
}
```

【示例】以下各链表均不带有头结点，其中最不适合用作链栈的链表是（　）。

A.只有表尾指针没有表头指针的循环单链表

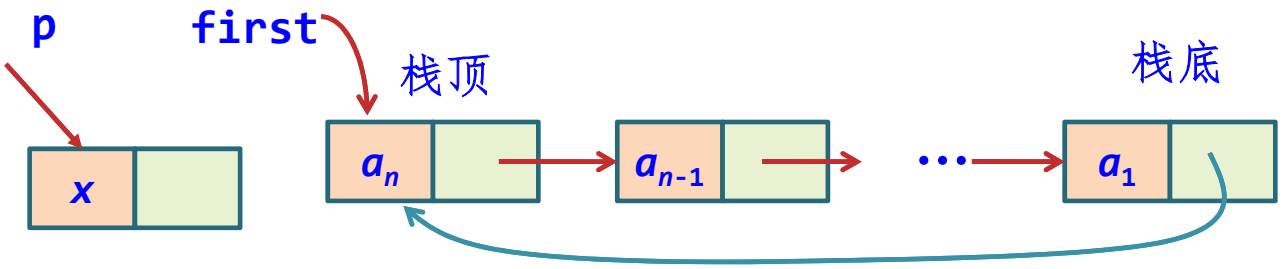
B.只有表头指针没有表尾指针的循环单链表

C.只有表头指针没有表尾指针的循环双链表

D.只有表尾指针没有表头指针的循环双链表

解：

B.只有表头指针没有表尾指针的循环单链表

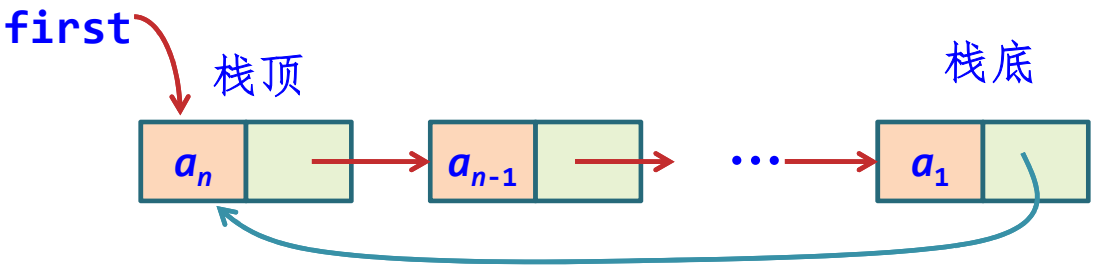


进栈操作：

```
q=first;
while (q->next!=first)
   q=q->next;          //查找尾结点q
p=(LinkStack *)malloc(sizeof(LinkStack));
p->data=x;
p->next=first;
first=p;
q->next=first;         //改为循环单链表
```

时间复杂度为O(*n*)

B.只有表头指针没有表尾指针的循环单链表



出栈操作：

```
q=first;
while (q->next!=first)
   q=q->next;          //查找尾结点q
p=first;
x=p->data;
first=first->next;
q->next=first;         //改为循环单链表
free(p);
```

时间复杂度为O(*n*)

END