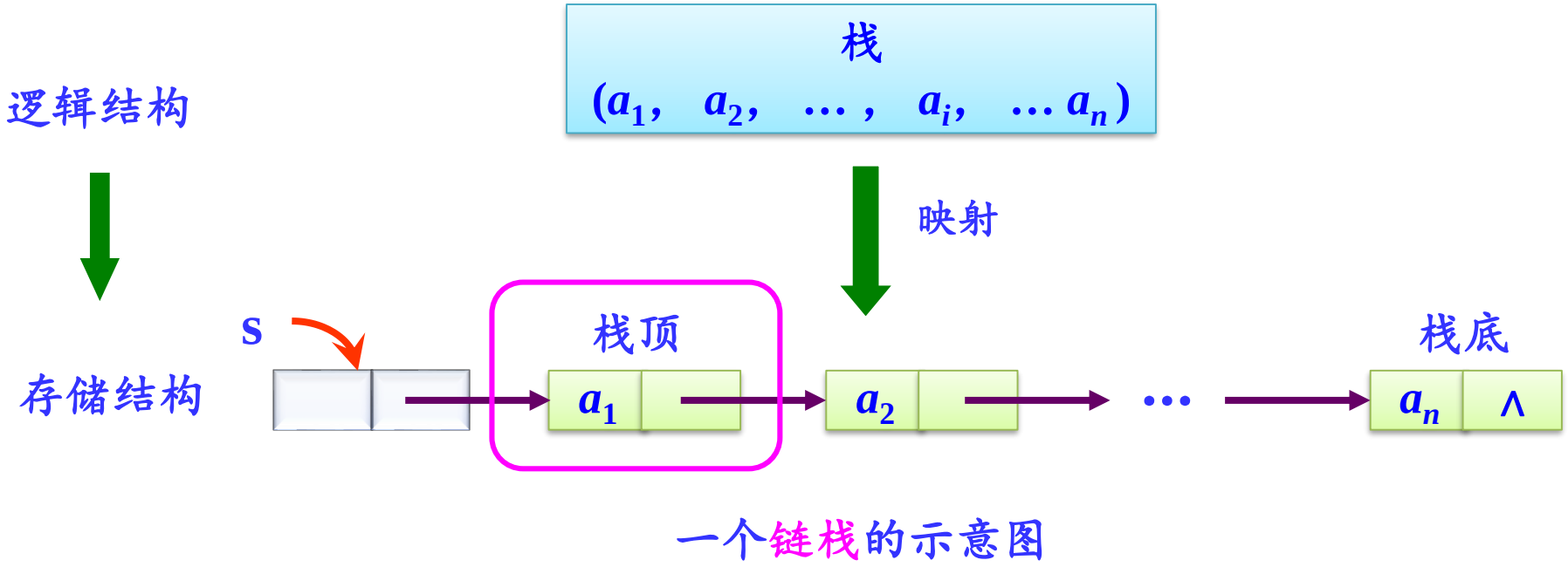
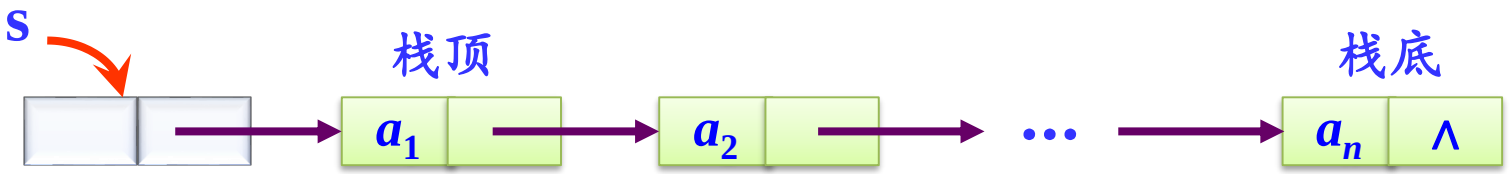
3.1.3 栈的链式存储结构及其基本运算的实现

采用链表存储的栈称为链栈，这里采用带头节点的单链表实现。



一个链栈的示意图



链栈的4要素：

- 栈空条件：s->next=NULL
- 栈满条件：不考虑
- 进栈e操作：将包含e的节点插入到头节点之后
- 退栈操作：取出头节点之后节点的元素并删除之

链栈中数据节点的类型LiStack定义如下:

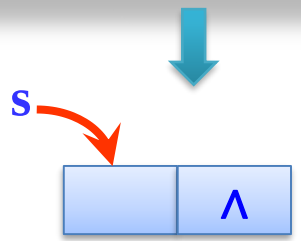
```
typedef struct linknode
{    ElemType data;                    //数据域
     struct linknode *next;            //指针域
} LiStack;
```

在链栈中，栈的基本运算算法如下。

（1）初始化栈initStack(&s)

建立一个空栈s。创建链栈的头节点，并将其next域置为NULL。

```
void (LiStack *&s)
{     s=(LiStack *)malloc(sizeof(LiStack));
      s->next=NULL;
}
```



（2）销毁栈DestroyStack(&s)

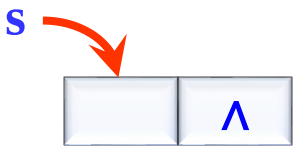
释放栈s占用的全部存储空间。

```
void DestroyStack(LiStack *&s)
{     LiStack *p=s,  *q=s->next;
      while (q!=NULL)
      {     free(p);
            p=q;
            q=p->next;
      }
      free(p);        //此时p指向尾节点，释放其空间
}
```

（3）判断栈是否为空StackEmpty(s)

栈S为空的条件是s->next==NULL，即单链表中没有数据节点。

```
bool StackEmpty(LiStack *s)
{
    return(s->next==NULL);
}
```

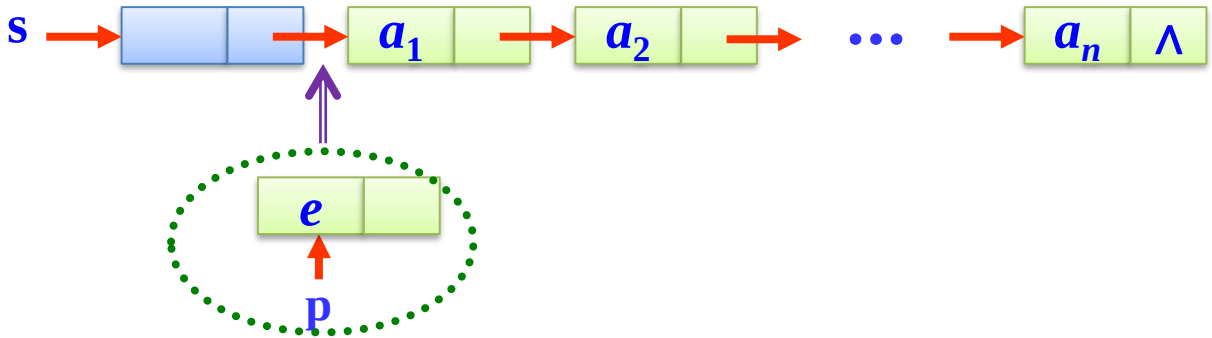


空栈的情况

（4）进栈Push(&s，e)

将新数据节点插入到头节点之后。

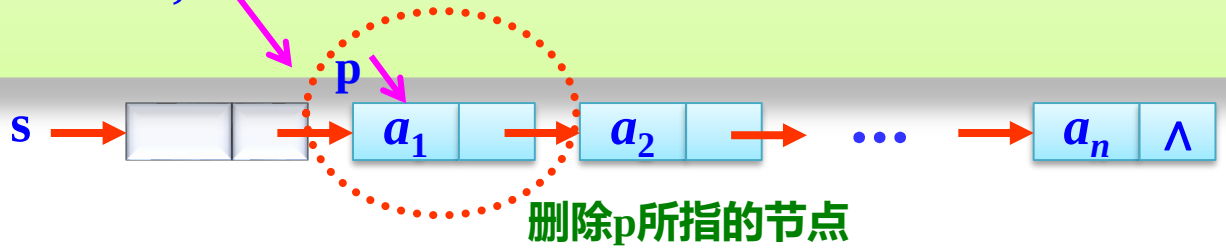
```
void Push(LiStack *&s，ElemType e)
{    LiStack *p;
     p=(LiStack *)malloc(sizeof(LiStack));
     p->data=e;                  //新建元素e对应的节点p
     p->next=s->next;            //插入p节点作为开始节点
     s->next=p;
}
```



（5）出栈Pop(&s，&e)

在栈不为空的条件下，将头节点后继数据节点的数据域赋给e，然后将其删除。

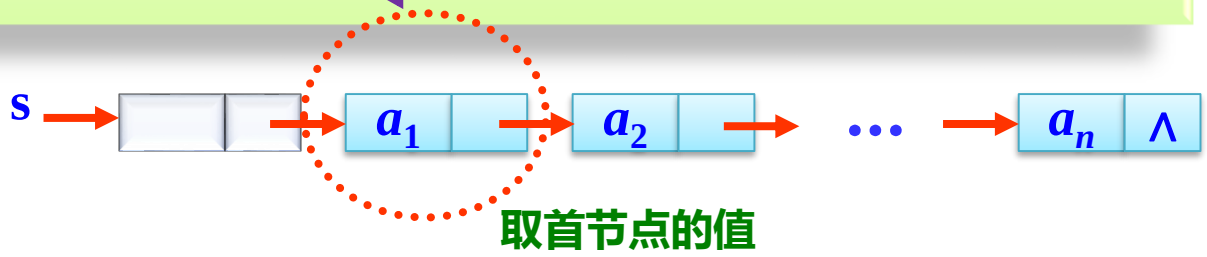
```
bool Pop(LiStack *&s，ElemType &e)
{    LiStack *p;
     if (s->next==NULL)                    //栈空的情况
            return false;
     p=s->next;                            //p指向开始节点
     e=p->data;
     s->next=p->next;                      //删除*p节点
     free(p);                              //释放*p节点
     return true;
}
```



（6）取栈顶元素GetTop(*s*，*e*)

在栈不为空的条件下，将头节点后继数据节点的数据域赋给*e*。

```
bool GetTop(LiStack *s，ElemType &e)
{     if (s->next==NULL)     //栈空的情况
            return false;
     e=s->next->data;
      return true;
}
```



思考题

链栈和顺序栈两种存储结构有什么不同？

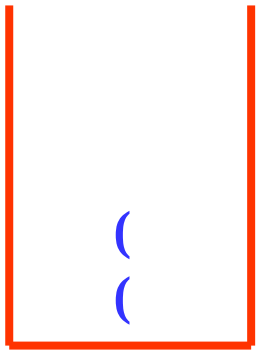
【例3-4】编写一个算法判断输入的表达式中括号是否配对（假设只含有左、右圆括号）。

算法设计思路

一个表达式中的左右括号是按最近位置配对的。所以利用一个栈来进行求解。这里采用链栈。

表达式括号不配对情况的演示

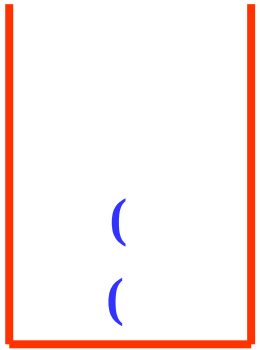
例如：exp=“(( )))”



遇到')'：栈为空，返回false：不匹配！

表达式括号配对情况的演示

例如：exp=“(( ))”



栈空且exp扫描完，返回true：匹配！

```
bool Match(char exp[],  int n)
{    int i=0;  char e;

    bool match=true;

    LiStack *st;

    InitStack(st);                    //初始化栈
    while (i<n && match)              //扫描exp中所有字符
    {
        if (exp[i]=='(')
            Push(st,  exp[i]);
```

配对时为true；否则为false

链栈指针

遇到任何左括号都进栈

```
    else if (exp[i]==')')              //当前字符为右括号
    {     if (GetTop(st，e)==true)
          {   if (e!='(')              //栈顶元素不为'('时表示不匹配
                  match=false;
              else
                  Pop(st，e);          //将栈顶元素出栈
          }
          else  match=false;           //无法取栈顶元素时表示不匹配
    }

    i++;                               //继续处理其他字符
}
```

遇到右括号判断是否匹配

```
    if (!StackEmpty(st))
        match=false;

    DestroyStack(st);                    //销毁栈
    return match;
}
```

栈不空时表示不匹配

只有在表达式扫描完毕且栈空时返回true。

——本讲完——