第三章 栈

一般的线性表：插入、删除操作位置任意

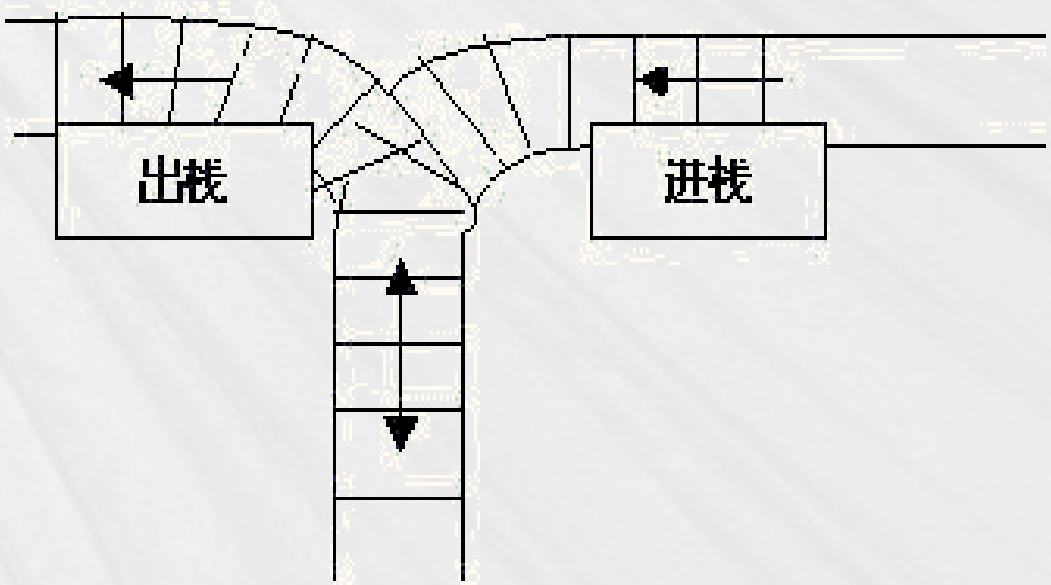
特殊的线性表：操作位置受限，栈和队列

栈：在表的一端进行操作(存、取数据）

队列: 分别在表的两端进行操作

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

本章节目录



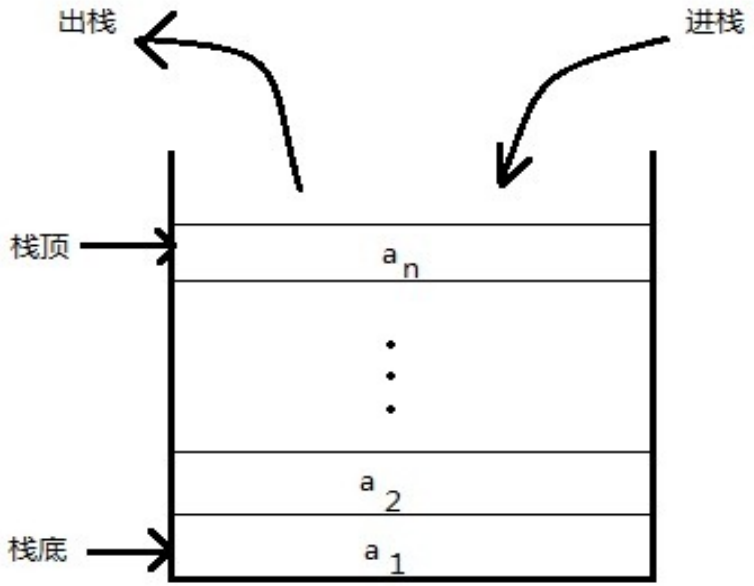
车库：在同一端进出

本章节目录

本章节目录

3.1 栈的抽象数据类型

- **定义：栈**(Stack)是限制在表的同一端进行插入和删除运算的**线性表**，通常称允许进行插入、删除的一端为**栈顶**(Top)，另一端为**栈底**(Bottom)。
- 栈的原则：**后进先出（LIFO）**
- 栈的实现方式
- 栈的抽象数据类型定义



图：栈的存储结构示意图

栈的抽象数据类型

ADT Stack{

数据对象：D={ai|ai∈ElemSet, i=1,2,…….n,n≥0}

数据关系：R={<a_i,a_{i+1}>|a_i,a_{i+1}∈D}

基本操作：

InitStack(&S);创建一个空的栈

Destroy(&S);销毁一个存在的栈

CLearStack(&S);清空一个已经存在的栈

StackEmpty（S）；判断栈是否为空

StackLength（S）

Push(&S，e)

Pop(&S，&e)

}

栈：特殊的线性表

栈的存储结构（物理结构）：？

3.2 栈的顺序存储结构

栈的顺序存储结构简称为顺序栈。
顺序栈：利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素，用一个变量top记录栈顶的位置（栈顶指针）

3.2.1 顺序栈的类型定义

```
#define  StackSize   100  //顺序栈的初始分配空间
typedef  struct
 {
   ElemType data[StackSize]; // 保存栈中元素
   int top;                  // 栈顶指针
    }   SqStack;
```

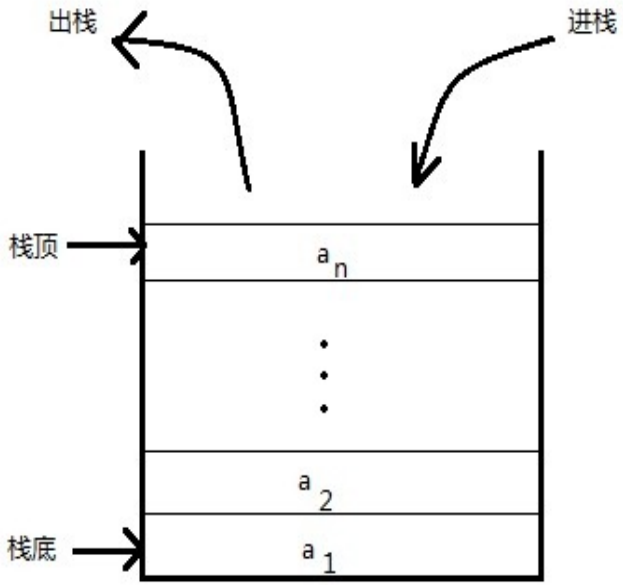
其中，StackSize是指顺序栈的初始分配空间，是栈的最大容量。
数组data用于存储栈中元素，top为栈顶指针（指向栈顶数据）。

- 由于C语言中数组的下标约定从0开始，因此，用top=-1表示栈空；非空栈中，top指出栈顶元素的位置
- 每当插入新的栈顶元素前，指针top先增1------入栈
- 删除栈顶元素后，指针top减1------出栈

3.2.2 栈基本运算在顺序栈上的实现

1. 初始化栈运算

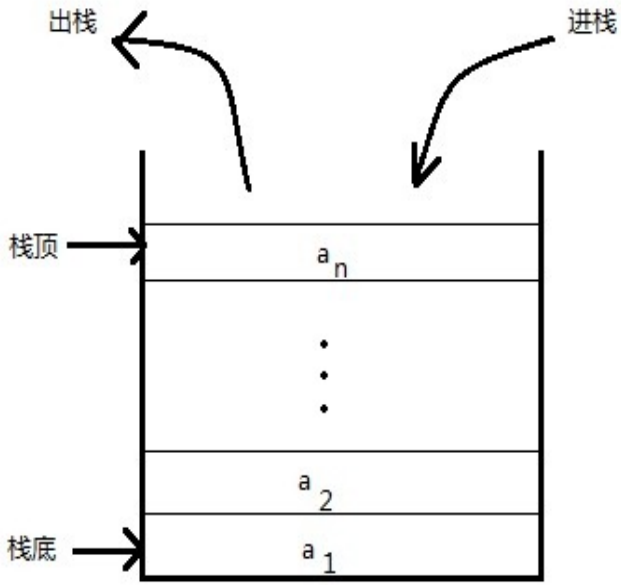
```
void InitStack(SqStack  &st)
{
    st.top=-1;
}
```



图：栈的存储结构示意图

2. 进栈运算

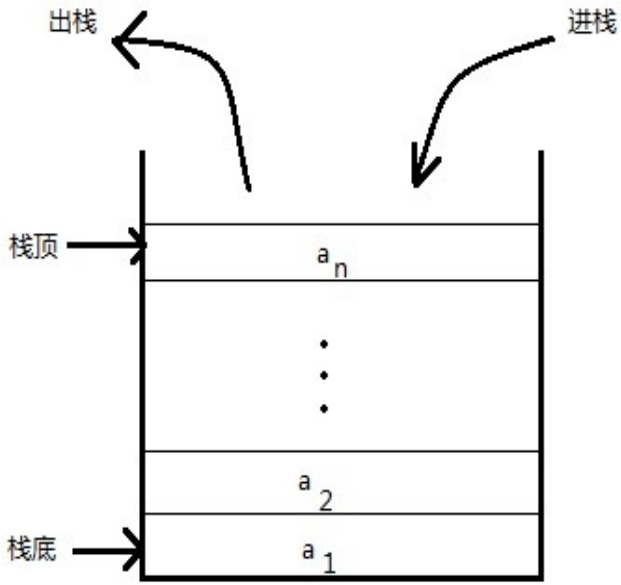
```
int Push(SqStack  &st,  ElemType e)
 {
    if (st.top==StackSize-1) return 0;  //溢出
    else {
          st.top++;
          st.data[st.top]=e;
          return 1;
         }
 }
```



图：栈的存储结构示意图

3. 出栈运算

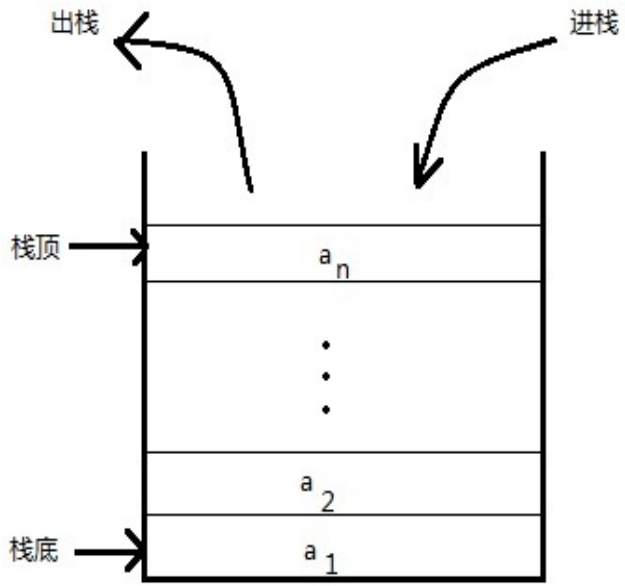
```
int Pop(SqStack   &st,  ElemType   &e)
 {
    if  (st.top==-1)  return 0;
    else {
              e=st.data[st.top];
              st.top--;
              return 1;
              }
 }
```



图：栈的存储结构示意图

4. 取栈顶元素运算（不弹出）

```
int GetTop(SqStack   st,  ElemType &e)
{
    if (st.top==-1) return 0;
    else {
            e=st.data[st.top];  //栈顶指针不变
            return 1;
         }
}
```



图：栈的存储结构示意图

5. 判断栈空运算

```
int StackEmpty(SqStack st)
{
    if  (st.top==-1) return 1;
    else return 0;
}
```

以上是顺序栈的几个基本操作！

3.2.3 顺序栈的应用举例

【例1】 设计一个算法，判断一个表达式中括号是否匹配。若匹配，则返回1；否则返回0。

算法思路：扫描表达式，当遇到左括号时，将其进栈，遇到右括号时，判断栈顶是否为相匹配的左括号。若不是，则退出扫描过程，返回0；否则栈顶元素出栈，直到扫描完整个表达式时，若栈为空，则返回1。

((x+y)*2+5)*3+a[i]*2

（（x+y)*[a+7)）+2

```
typedef   char  ElemType
int match(char   *exp)     //字符串是表达式，如何判断字符串结束？
{ SqStack   st;   int flag=0, i=0;  //flag记录匹配情况
  st.top=-1;
  while(exp[i]!='\0'&&flag==0)   {
   switch (exp[ i ])
   {       case  '('  :
           case  '['  :  st.data[++st.top]=exp[i]; break;
           case  ')'  :  if(st.data[st.top]==  '('  ) st.top--;   else  flag=1;
                         break;
           case  ']'  :  if(st.data[st.top]==  '['  ) st.top--; else flag=1;
                 }
    i++;                                }
   if(flag==0&&st.top=-1)   return 1;     else     return 0;
}
```

【例2】编写一个算法，将非负的十进制整数转换为其他进制的数输出，10及其以上的数字用从‘A’开始的字母表示。并给出完整的程序。

- 解题思路：
- 算法trans：先用“除基取余”法按照数制转换，求得从低位到高位的数字，转变为字符后依次暂时存放顺序栈中；当商为0时，转换结束；然后从栈顶到栈底取出从高位到低位的字符存到一个字符数组中，并通过数组返回给主调函数。
- 先定义顺序栈，并根据题意将ElemType设为char型；然后由算法trans来完成数制的转换；最后在主函数中调用实现转换算法的函数。
- **数制转换：除基取余法**

```
typedef char    ElemType
int tran(int d,  int b,  char string[ ])  //b:进制(2 to 36)，d：数据
  { SqStack   st;   char  ch;  int  r,  i=0;
   st.top=-1;
   while(d!=0)   //转换数制，从低位到高位依次进栈
    {   r=d%b;
        ch=(r<10 ? '0'+r  :  'A'+r-10);     //转变为对应的字符
        st.top++;   st.data[st.top]=ch;   //压入栈顶
         d=d/b;    }
   while(st.top!=-1)      //存入字符数组（由高位到低位）
       string[i++]=st.data[st.top--];
   string[i]='\0';
    return 1;
}
```

3.3 栈的链式存储结构

- 栈的链式存储结构称为**链栈**
- 它是操作受限的**单链表**，即**插入和删除操作**仅限制在**表头位置**上进行。

3.3.1链栈的类型定义

```
typedef  struct  stnode
        {
        ElemType   data;
        struct stnode  *next;
        }     StNode,    *LinkStack;
```

3.3.2 栈基本运算在链栈上的实现

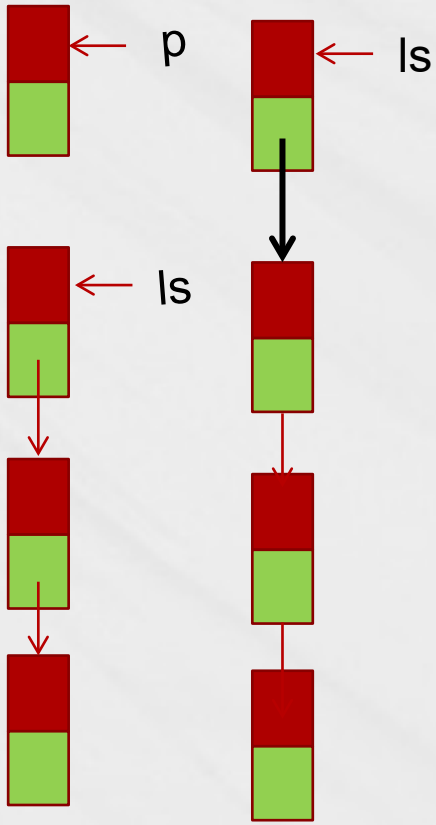
//用栈顶指针　ls　指出堆栈

1. 初始化栈运算

```
void InitStack(LinkStack   ls)
{
    ls=NULL;
  }
```

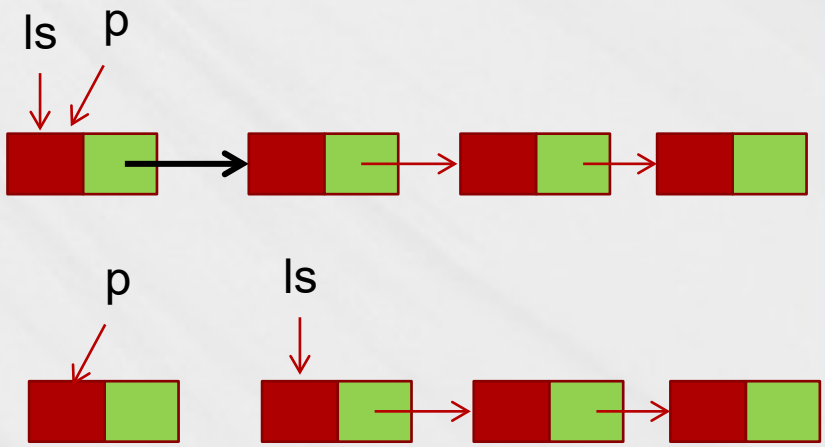
2. 进栈运算（值e入栈）

```
void Push(LinkStack  ls,  ElemType  x)
 {
   StNode  *p;
   p=( StNode * ) malloc(sizeof(StNode));
   p->data=x;
    p->next=ls;
    ls=p;
 }
```



3. 出栈运算 →x

```
int Pop(LinkStack  ls,  ElemType  &x)
{  StNode  *p;
   if (ls==NULL)  return 0;
   else  { p=ls;
           x=p->data;
           ls=p->next;
           free(p);
           return 1;
         }
}
```



4. 取栈顶元素运算（不弹出，存入x）

```
int GetTop(LinkStack  ls,  ElemType  &x)
{
  if  (ls==NULL)  return 0;
  else
     {
       x=ls->data;
       return 1;
      }
}
```

5. 判断栈空运算

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

❖ 3.3.3 链栈的应用举例

❖ 【例3】回文指的是一个字符串从前面读和从后面读都一样，编写一个算法判断一个字符串是否为回文。并给出完整的程序。字符串用数组存储。

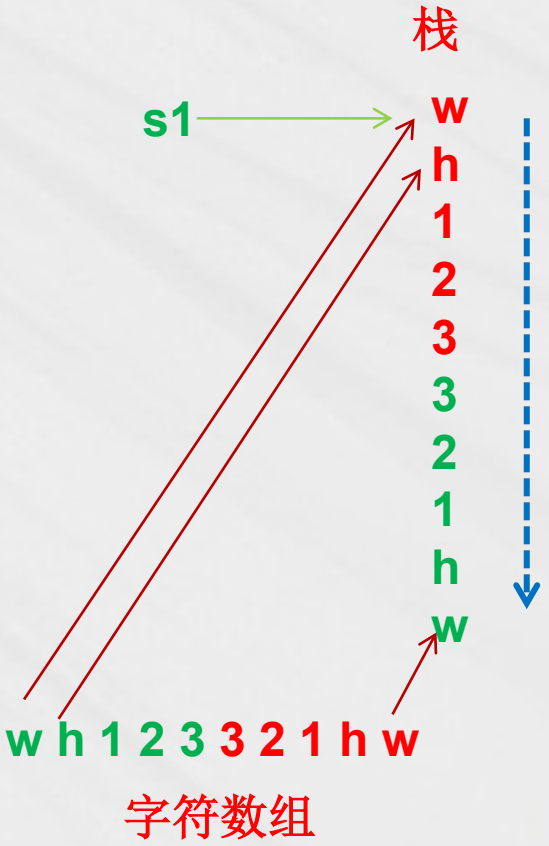
❖ 解题思路：先定义链栈的类型，并根据题意将ElemType设为char型；然后设计一个算法huiwei来完成判断；最后在主函数中调用实现判断算法的函数。

❖ 算法huiwei的思路：采用何种数据结构？

❖ 采用链栈：先将字符串从头到尾的各个字符依次放入一个链栈中(反序存储）；然后依次取栈顶到栈底的各个字符与字符串从头到尾的各个字符比较，如果两者不同，则表明该字符串不是回文，若相同，则继续比较；如果直到比较完毕相互之间都相匹配，则表明该字符串是回文。

```
int huiwen(char str[ ])
{  LinkStack  s1=NULL, p;   int i=0;   char  ch;
   while((ch=str[i])!='\0')     //反序存入栈
   {  p= (StNode *)malloc(sizeof(StNode));
      p->data=ch;
      p->next=sl;
      s1=p;     }
```

```
int huiwen(char str[ ])
{
    i=0;
    while(s1 != NULL)
    {  p=s1;
       ch=p->data;
       s1=s1->next;
        free(p);
        if(ch!=str[i++])   return  0;    }
     return 1;
}
```



3.4 栈与递归的实现

- 栈的一个重要应用是在程序设计语言中实现递归。

- 一个直接调用自己或通过一系列的调用语句间接地调用自己的函数，称做递归函数。

【例4】阶乘函数。

$$Fact(n)=\begin{cases}1 & \text{若}n=0\\ n*Fact(n-1) & \text{若}n>0\end{cases}$$

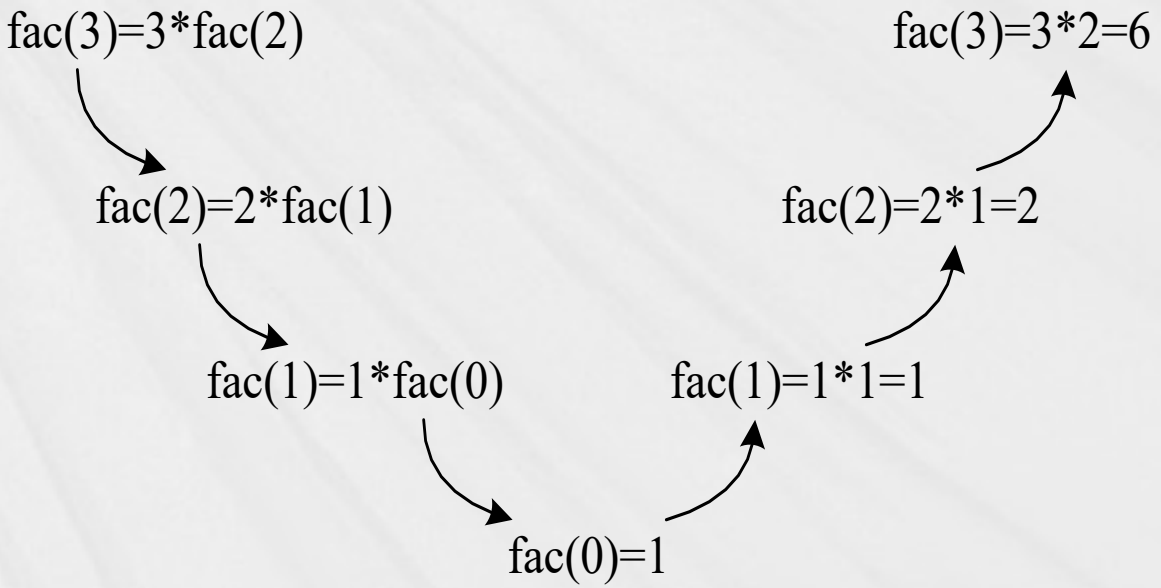
以求3！为例，完整的程序如下：

```
#include "stdio.h"
void main()
{
    int result,n;
    n=3;
0   result =fact(n);
    printf("3!=%d\n",result);
}
```

```
int fact(int n)
 {
1    int f;
2   if(n==0)     //递归终止条件
3       f=1;
4   else
5       f=n*fact(n-1);
6   return f;
 }
```

- 递归是把一个不能或不好直接求解的“大问题”转化成一个或几个“小问题”来解决，再把这些“小问题”进一步分解成更小的“小问题”来解决，如此分解，直至每个“小问题”都可以直接解决，此时递归终止，逐层返回。

- 计算fac(3)的过程如 下：




```
int fact(int n)
 {
1    int f;
2   if(n==0)
3       f=1;
4    else
5       f=n*fact(n-1);
6   return f;
 }
```

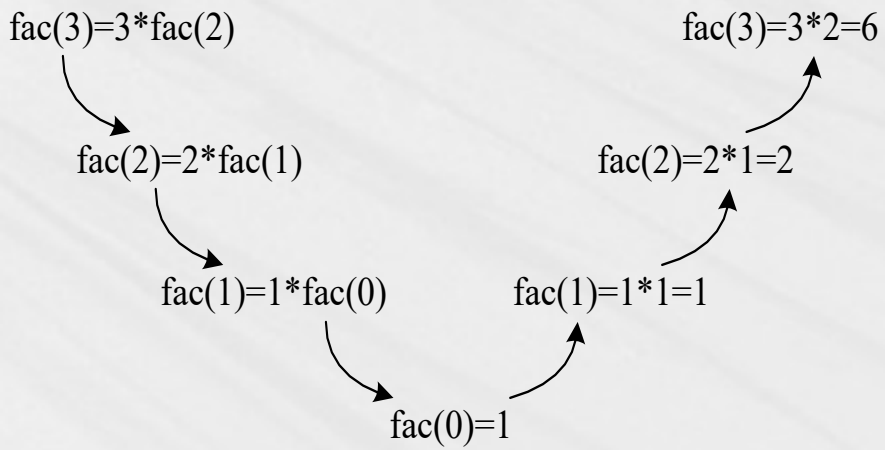
```
void main()
{
      int result,n;
      n=3;
0     result =fact(n);
      printf("3!=%d\n",result);
}
```

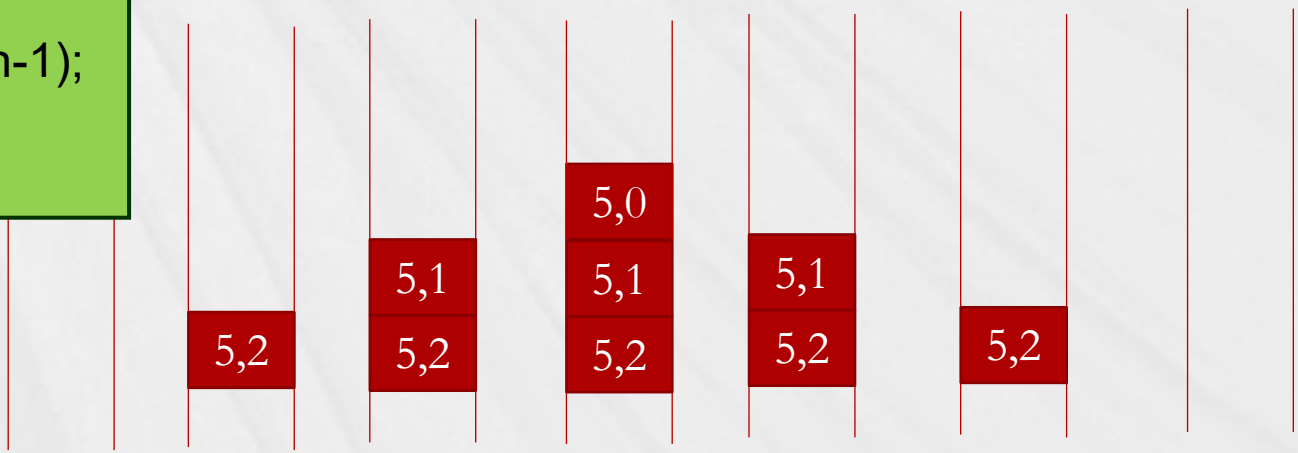
```
int fact(int n)
  {
1     int f;
2    if(n==0)
3       f=1;
4    else
5       f=n*fact(n-1);
6    return f;
  }
```

- 递归调用的特点是：主调函数和被调函数是同一个函数
- 为了保证递归函数正确执行，系统需设立一个“递归工作栈”，每一层递归所需信息构成一个“工作记录”，其中包括所有的实际参数、以及上一层的返回地址。
- 每进入一层递归，就产生一个新的工作记录压入栈顶。每退出一层递归，就从栈顶弹出一个工作记录。

```
void main()
{
      int result,n;
      n=3;
0     result =fact(n);
      printf("3!=%d\n",result);
}
```

```
int fact(int n)
 {
1    int f;
2   if(n==0)
3      f=1;
4    else
5      f=n*fact(n-1);
6   return f;
 }
```





调用---进栈

返回---出 栈

本章小结

❖栈是一种操作受限的线性表，即只允许在表的一端进行插入和删除操作。栈的结构特点是**后进先出（先进后出）**，在许多软件系统中都会用到这种结构。

❖栈也有顺序存储结构和链式存储结构。

❖栈的顺序存储结构称为顺序栈，数组：存储数据，下表：栈顶指针，主要通过改变栈顶指针来实现各种操作；

❖栈的链式存储结构称为链栈，其各种操作的实现类似于单链表。

本章习题

1.设一个栈的输入序列为A,B,C,D,则借助一个栈所得到的输出序列不可能是哪个？

（1）ABCD （2）BDCBA （3）ACDB （4）DABC

2.若元素进栈顺序为1234，为了得到1342的出栈顺序，试给出相应的操作序列。

3.链栈中为何不设头结点？

4.写一个算法，借助于栈将一个单链表逆置。