栈讲稿

一、栈的逻辑结构

引例、栈及其操作（P1509）

栈：插入元素和删除元素只能在线性表的一端进行，所以遵循先进后出原则，其中插入和删除的一端称为栈顶(top)。我们可以把栈比喻成一个箱子，只能在箱子的开口处放入和取出物体，而且是后放入的物体，会被先取出来。

【输入格式】

第 1 行一个整数 N，表示有 N 条关于 stack 的操作，在进行任何操作之前，stack 是空的。

接来的 N 行，每行一个关于 stack 的操作，格式和含义如下：

clear：把栈置空。

empty：判断栈是否为空。

push x：把整数 x 插入栈顶。

pop： 栈顶元素出栈。

top：获取栈顶元素的值。

【输出格式】

若干行，对应输入中的 top ， pop 和 empty 操作：

对于 top 操作，输出一个整数，如果这个操作失败，则输出单词 error。

对于 top 操作，如果这个操作失败，则输出单词 error。

对于 empty 操作，如果栈或队列是空，则输出’empty’，否则输出’not empty’。

【样例】

8 push 10 top push 15 pop top clear pop empty	10 10 error empty

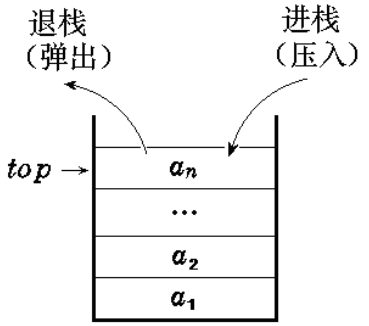
【数据范围】

N<=20000，x 是一个 int 范围的整数。

【讲解】

1、栈的逻辑结构

题目中对栈的逻辑结构和操作规则定义的非常清楚。图形描述如下：



2、栈的存储结构

现在来看他们的物理结构（存储结构），对于整数类型的栈，定义如下：

STL 之栈容器	数组栈
#include<stack> 定义格式： stack<元素类型>栈名称 例如： stack<int>stack;	#define SIZE 20005 //栈的容量 元素类型 stack[SIZE]; int top; //栈顶指针 说明： ◆栈元素的类型随着不同问题而不同 ◆变量 top：是栈顶指针，他始终指向**栈顶元素的位置** ◆当 top==0 时，栈为空 ◆当 top>0 时，栈为非空

3、栈的运算

运算	STL 之栈容器操作	数组栈运算代码
empty：判断栈是否为空	stack.empty();	如果 top==0 则栈为空
push x ：把整数 x 插入栈顶	stack.push(x);	stack[++top]=x;
pop： 栈顶元素出栈。	stack.pop();	top--;
clear：把栈置空	while(!stack.empty()) stack.pop();	top=0;
top：读取栈顶元素的值。	X=stack.top();	x=stack[top]
栈中元素个数	num=stack.size();	num=top;

4、本题算法流程

数组栈

```
int stack[20005];
int top;
void solve()
{
  char op[10];
  scanf("%d",&N);
  top=0;
  for(int i=1;i<=N;i++)
  {
     scanf("%s",op);
     if(strcmp(op,"clear")==0)  //清空
     {
          top=0;
     }
     else if(strcmp(op,"empty")==0) //判 空
     {
         if(top==0)
             printf("empty\n");
         else
             printf("not empty\n");
     }
     else if(strcmp(op,"push")==0)  //入栈
     {
         scanf("%d",&x);
         stack[++top]=x;
     }
```

STL 栈容器

```
void solve1()
{
  char op[10];
  scanf("%d",&N);
  top=0;
  for(int i=1;i<=N;i++)
  {
     scanf("%s",op);
     if(strcmp(op,"clear")==0)
     {
        while(!stack.empty()) stack.pop();
     }
     else if(strcmp(op,"empty")==0)
     {
         if(stack.empty())
             printf("empty\n");
         else
             printf("not empty\n");
     }
     else if(strcmp(op,"push")==0)
     {
         scanf("%d",&x);
         stack.push(x);
     }
```

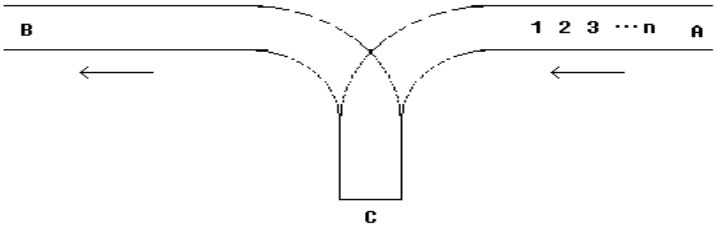
```
    else if(strcmp(op,"pop")==0)  //出栈
    {
       if(top==0)
          printf("error\n");
       else
          --top;
    }
    else if(strcmp(op,"top")==0) //读栈顶
    {
       if(top==0)
           printf("error\n");
       else
       {
            x=stack[top];
            printf("%d\n",x);
       }
    }
  }
}
```

```
    else if(strcmp(op,"pop")==0)
    {
       if(stack.empty())
          printf("error\n");
       else
          stack.pop();
    }
    else if(strcmp(op,"top")==0)
    {
       if(stack.empty())
           printf("error\n");
       else
       {
          x=stack.top();
          printf("%d\n",x);
       }
     }
  }
}
```

二、栈例题讲解

例 1、车站铁轨（P1502）

有 n 节车厢从 A 方向驶入车站，按进站顺序编号为 1..n。你的任务是让他们按照某中特定的顺序进入 B 方向的铁轨并驶出车站。为了重组车厢，你可以借助中转站 C。这是一个可以停放任意多节车厢的车站，但由于末端封顶，驶入 C 的车厢必须按照相反的顺序驶出 C。对于每个车厢，一旦从 A 移入 C，就不能再回到 A 了；一旦从 C 移入 B，就不能回到 C 了。换句话说，在任意时刻，只有两种选择：A->C 和 C->B。



现在需要你写一个程序，判断给定的 B 方向驶出车站的车箱顺序是否可行，若不可行输出'no'；若可行则输出'Yes'，并输出要能得到这个出站顺序，中转站 C 至少需要几个存放车厢的位置。

【样例】

5 //n 节车厢：n<=10000 1 2 3 4 5 //B 方向的出站序列	yes //可行 1 //中转站 C 至少需要 1 个存放车厢的位置。

【讲解】

中转站 C 操作规则需要按栈操作规则来进行，因此把 C 看成数据结构——栈。设出栈序列为 B[1]、B[2]、…、B[N]。则算法流程如下：

```
把栈置空;
int j=1;   //出栈序列的第 j 个元素;
int ans=0  //最少需要栈的大小
for(i=1;i<=N;i++)      //1，2，…,N 依次进栈
{
      ▶元素 i 进栈;
      ▶if(栈中元素个数大于 ans 时) ans=栈的元素个数;     //思考：为什么？？
      ▶当栈顶元素等于 B[j]，则出栈然后 j++，直到栈顶元素不等于 B[j]，或栈为空为止;
}
根据栈是否为空来判断是否有解。
```

时间复杂度 O(N)

例 2、括号平衡（P1503）

在本题中，题目会先给你一个包含小括号 () 及中括号 [] 的字串。当字串符合下列条件时我们称他为正确的运算式：

1．该字串为一个空字串。

2．如果 A 和 B 都为正确的运算式，则 AB 也为正确的运算式。

3．如果 A 为正确的运算式，则 (A) 及 [A] 都为正确的运算式。

现在，请你写一支程序可以读入这类字串並检查它们是否为正确的运算式。字串的长度不超过 128。

【样例】

3 //n 个字符串：1 <= n <= 100 ([]) (([()])) ([()[]()])()	Yes //匹配则输出 Yes，否则输出 No No Yes

【讲解】

括号表达式中，右括号一定要与左括号匹配，所以算法的思想是：

设置一个字符类型的栈，元素用于存储括号。然后从左到右扫描括号表达式（字符串），若遇到左括号'('或'['，则压入栈中，若遇到右括号，则读取栈顶并删除栈顶括号，若该元素与右括号匹配，则继续扫描，否则括号不匹配，终止扫描。

```
void solve(char *s)
{
   初始化栈;
   标记变量 ok=1;
   for(int i=0;s[i]!=0;i++)
   {
       如果 s[i]是 '(' 或'[', 则把 s[i]压入栈中;
       如果 s[i]为')'或']', 则
       {
            如果栈为空则 ok=0; break;
            读取栈顶元素到 x 中，并删除栈顶元素;
            如果 x 与 s[i]不匹配则 ok=0; break;
       }
   }
   根据 ok 输出解;
}
```

例 3、后缀表达式求值

不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行，不再考虑运算符的优先规则，这样的表达式称为**后缀表达式**，也叫**逆波兰表达式**。如：(2 + 1) * 3 的后缀表达式为 2 1 + 3 *。它是为了方便在计算机中进行表达式求值而出现的。给出一个仅由整数 、+、-、*、/（这里的 / 表示整除）等组成的后缀表达式，符号之间用空格分开，计算它的值。

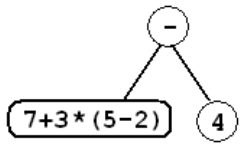
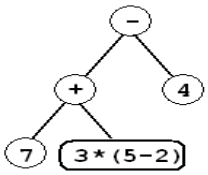
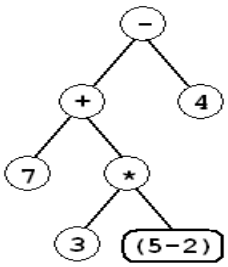
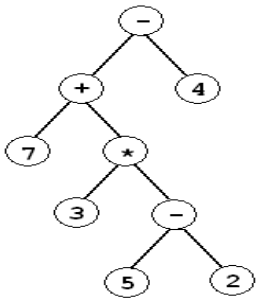
【样例】

2 1 + 3 * //后缀表达式（长度不超过 200）	9 //运算结果（最后的结果和中间结果都不会超过 $2*10^9$）

【讲解】

1、前、中、后缀表达式的认识

首先依据中缀表达式，按照优先级顺序构造一个达式树，它是一棵二叉树。例如：7+3*(5-2)-4

表达式中：7+3*(5-2)-4 最后计算'-'	表达式：7+3*(5-2) 最后计算'+'	表达式：3*(5-2) 最后计算'*'	表达式：(5-2) 最后计算'-'
			

前序遍历最后的表达式树得到**前缀表达式**：- + 7 * 3 - 5 2 4

后序遍历最后的表达式树得到**后缀表达式**：7 3 5 2 - * + 4 -

2、后缀表达式的计算

设置一个数据栈：用与存放参与运算的数或运算的中间结果。int nd[200],top=0;

运算的过程：从左至右扫描表达式，如果当前是常数则压入数据栈，如果是运算符，则将栈顶两个常数作相应的运算，并把计算结果压入栈。一直扫描到表达式的最右端时，最后数据栈的栈顶元素的值表达式的值。

```
int calc(int a,char op,int b)   //计算 a op b 的值
{
   if(op=='+') return a+b;
   if(op=='-') return a-b;
   if(op=='*') return a*b;
   if(op=='/' && b!=0) return a/b;
   if(op=='^') return mypow(a,b);   //计算 a 的 b 次方
}
int calcexp(char *s)  //计算后缀表达式的值
{
   for(int i=0;s[i]!=0;i++)
   {
     if(isdigit(s[i]))    //如果 s[i]是数字符号，计算常数 x，并把 x 压入数据栈
     {
         int x=s[i]-48;
         while(isdigit(s[i+1])) x=x*10+s[++i]-48;
         nd[++top]=x;
     }
     else if(s[i]=='+'||s[i]=='-'||s[i]=='*'||s[i]=='/'||s[i]=='^')  //如果是运算符
     {
         int b=nd[top--],a=nd[top--];   //获取数据栈顶的两个元素并删除他们，注意是 b,a 的顺序
         nd[++top]=calc(a,s[i],b);       //计算 a op b 的结果并压如数据栈
      }
   }
   return nd[top];   //栈顶元素就是表达式的值
}
```

例 5、中缀表达式计算（P1517）

一个数学表达式由下列元素组成：左括号，右括号，加号，减号，乘号，正号，负号，整数（无前导 0）。现在给出一个长度不超过 100 的数学表达式，求它的值。要求考虑括号和乘法的优先级，计算过程中的临时值的绝对值保证不会超过 int 范围。

给出的表达式保证合法以及符合人的书写习惯（但可能会有多余的括号对）。以下表达式被认为是合法的：((10+(-100)))，-3*(+5+7*2)-(0)，-0，(-3)*(-5+7)。

以下表达式被认为非法：1+-7，--3+8，-3+()

【样例】

-3*(+5-7*2)-(0)	27

【讲解】

1、整理表达式

本题出现了单目运算符：正号'+'和负'-'，可以把这两个符号转换为双目运算加和减来运算，比如：

-3*(**+5**-7*2)-(0) ➔ **0-3***(**0+5**-7*2)-(0) ➔ **(0-3***(**0+5**-7*2)-(0)**)**

因为根据题意**“除了括号以外，不会出现两个运算符号连续出现的情况”**，所以正负号出现的情况只可能在左括号后或表达式的开始处。因此在整理表达式的时候，就实现正负号的转换！

```
void ready(char *s)   //整理表达式，把正负号转换为双目运算加法或减法，并整理成(表达式格式)
{
   int j=0;
   t[j++]='(';
   for(int i=0;s[i]!='\0';i++)    //扫描表达式，把 单目运算符 转换成 双目运算符
   {
       if(s[i]=='+' || s[i]=='-')
          if(i==0 || s[i-1]=='(')     //如果'+'和'-'出现开始或在'(' 后，则说明这里的'+'和'-'是正负号
            t[j++]='0';
```

```
        t[j++]=s[i];
    }
    t[j++]=')';  t[j]='\0';
    strcpy(s,t);
}
```

2、算术运算符的优先级

常见的算术表达式包含的运算符有：+,-,*,/,^，按照运算的优级，数字化为：

```
int PRI(char op)  //获取运算符 op 的优先级
{
    if(op=='(') return 0;    //'('的优先级最低
    if(op=='+' || op=='-') return 1;    //+,-的优先级为 1
    if(op=='*' || op=='/') return 2;    //*, /的优先级为 2
    if(op=='^') return 3;               //^的优先级为 3
    if(op==')') return 4;
}
```

3、计算中缀表达式的值（请看动画演示）

```
int calcexp(char *s)  //计算算术表达式的值
{
   ready(s);
   stack<int>nd;     //数据栈
   stack<char>op;    //运算符栈
   for(int i=0;s[i]!=0;i++)
   {
     if(isdigit(s[i]))
     {
        int x=s[i]-48;
        while(isdigit(s[i+1])) x=x*10+s[++i]-48;
        nd.push(x);
     }
     else if(s[i]=='(') { op.push(s[i]);  }
     else if(s[i]==')')
     {
        while(op.top()!='(')
        {
           int b=nd.top();  nd.pop();
           int a=nd.top();  nd.pop();
           char c=op.top(); op.pop();
           nd.push(calc(a,c,b));
        }
        op.pop();  //把'('弹出
     }
     else if(s[i]=='+'||s[i]=='-'||s[i]=='*')
     {
        while(PRI(s[i])<=PRI(op.top()))
        {
            int b=nd.top();  nd.pop();
            int a=nd.top();  nd.pop();
            char c=op.top(); op.pop();
            nd.push(calc(a,c,b));
        }
        op.push(s[i]);
     }
   }
  return nd.top();
}
```

算法流程描述：

1、整理表达式为(exp)的格式；

2、设置数据栈和运算符号栈

3、依次扫描表达式 s[i]

如果 s[i]是数字符号

计算 s[i],s[i+1]..构成的常数 x

把常数 x 压入栈

如果 s[i]是左括号'('

把'('压入运算符栈

如果 s[i]是')'

则计算一对括号"(…)"中的表达式的值并把这些值得压入数据栈，

并且把与')'配对的'('出栈

如果 s[i]是运算符

当 s[i]的优先级小于等于运算符栈顶运算符的优先级，则计算栈顶运算符的运算结果，并压入数据栈；

最后把 s[i]入栈

三、关于栈的信息学初赛试题

1.（06 年）某个车站呈狭长形，宽度只能容下一台车，并且只有一个出入口。已知某时刻该车站状态为空，从这一时刻开始的出入记录为：“进，出，进，进，进，出，出，进，进，进，出,出”。假设车辆入站的顺序为 1，2，3，…，则车辆出站的顺序为（ ）。

A．1，2，3，4，5　　B．1，2，4，5，7　　C．1，4，3，7，6

D．1，4，3，7，2　　E．1，4，3，7，5

2.（05 年）设栈 S 的初始状态为空，元素 a，b，c，d，e，f，g 依次入栈，以下出栈序列不可能出现的有（ ）。

A．a，b，c，e，d，f，g　B．b，c，a，f，e，g，d　C．a，e，c，b，d，f，g

D．d，c，f，e，b，a，g　E．g，e，f，d，c，b，a

3.（07 年） 地面上有标号为 A、B、C 的 3 根细柱，在 A 柱上方有 10 个直径相同中间有孔的圆盘，从上到下次编号为 1，2，3,…，将 A 柱上的部分盘子经过 B 柱移入 C 柱，也可以在 B 柱上暂存。如果 B 柱上的操作记录为：“进，进，出，进，进，出，出，进，进，出，进，出，出”。那么，在 C 柱上从下到上的盘子的编号为(　　)。

A．2 4 3 6 5 7　　B．2 4 1 2 5 7　　C．2 4 3 1 7 6

D．2 4 3 6 7 5　　E．2 1 4 3 7 5

4.（02 年）若已知一个栈的入栈顺序是 1，2，3，…，n，其输出序列为 P1，P2，P3，…，Pn，若 P1 是 n，则 Pi 是( )

A.i　　B.n-1　　C.n-i+1　　D.不确定

5.（04 年）以下哪一个不是栈的基本运算(　　)

A.删除栈顶元素　B.删除栈底的元素　C.判断栈是否为空　　D.将栈置为空栈

6.（03 年）一个栈的输入顺序为 1、2、3、4、5，下列序列中可能是栈的输出序列是 ( )。

A.54312　　B.24315　　C.21345　　D.12534

7.（03 年）设栈 S 的初始状态为空，元素 a，b，c，d，e，f 依次入栈 S，出栈的序列为 b，d，c，f，e，a，则栈 S 的容量至少应该是（　　）。

A．6　　B．5　　C．4　　D．3　　E．2

8.(08 年)已知队列（13，2，11，34，41，77，5，7，18，26，15），第一个进入队列的元素是 13，则第五个出队列的元素是（　　）。

A．5　　B．41　　C．77　　D．13　　E．18

9.（10 年）元素 R1、R2、R3、R4、R5 入栈的顺序为 R1、R2、R3、R4、R5。如果第一个出栈的是 R3，那么第 5 个出栈的可能是（　　）

A．R1　　B．R2　　C．R4　　D．R5

10.（12 年）如果一个栈初始时为空，且当前栈中的元素从栈底到栈顶依次为 a,b,c，另有元素 d 已经出栈，则可能的入栈顺序是(　　)

A、a,b,c,d　　B、a,d,b,c　　A、a,c,b,d　　A、d,a,b,c

11. 队列 Q 的对首指针 front=3，队尾指针 rear=10，则队列中元素的个数为（　　）

A、6　　B、7　　C、8　　A、10

12.循环队列 Q 的下标范围为 1..N，队首指针为 front，队尾指针为 rear，则队列中元素的个数为(　　)

A、rear-front　B、rear-front+1　C、(rear-front+1)%10　A、(rear-front+N)%10

13. 设栈 S 和队列 Q 的初始状态为空，元素 e_1，e_2，e_3，e_4，e_5，e_6依次通过栈 S，一个元素出栈后即进入队列 Q，若出队的顺序为 e_2，e_4，e_3，e_6，e_5，e_1，则栈 S 的容量至少应该为（ ）。

A、2　　B、3　　C、4　　D、5

14、表达式 a*(b+c)-d 的后缀表达式是：

A）abcd*+-　　B）abc+*d-　　C）abc*+d-　　D）-+*abc

15.前缀表达式“+ 3 * 2 + 5 12 ” 的值是（ ）。

A. -31　　B. 13　　C. 17　　D. 31

16. 中缀表达式"23+(12*(5-3))+108/9　"的前缀表达式是（ ）。

A. + * - + / 23 12 5 3 108 9　　B. + + 23 * 12 - 5 3 / 108 9

C. + + 23 * 12 - 3 5 / 9 108　　D. + + 23 * - 5 3 12 / 108 9

E. + / 108 9 + 23 * 12 - 5 3　　F. 23 12 5 3 - * + 108 9 / +

17. 中缀表达式"57/(23*4-1)-23+45/9"的后缀表达式是（ ）。

A. 57 23 4 1 23 45 8 / * - - + /　　B. 57 1 23 4 * - / 23 - 45 9 / +

C. 57 23 4 * 1 - / 23 - 45 9 / +　　D. + - / 57 - * 23 4 1 23 / 45 9

18. 后缀表达式"10 2 3 + * 5 1 - /"，规定这里的'/'的运算取商的整数部分，表达式的值是（ ）。

A. 12　　B. -12　　C. 11　　D. 13

19. 请画出中缀表达式"a-(b*(c-b)+d*e)+f"的表达式树，然后写出其前缀和后缀表达式：

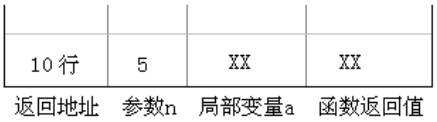
附录 1、C++函数调用栈

1、调用栈

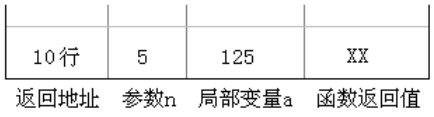
C++调用函数时，操作系统为每个函数申请开设一个栈帧——调用栈的一个元素，函数的返回地址、实际参数、局部变量、函数的返回值等都会进入这个栈，当函数执行完毕时，这个栈又被收回！例如：

```
1  int f(int n)
2  {
3       int a;
4       a=n*n*n;
5       return a;
6  }
7
8  int main()
9  {
10     int ans=f(5);
11     printf("%d %d\n",n,ans);
12      return 0;
13 }
```

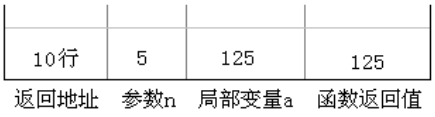
执行第 10 行调用 f(5)



执行第 4 行 a=n*n*n;



执行第 5 行 return a;

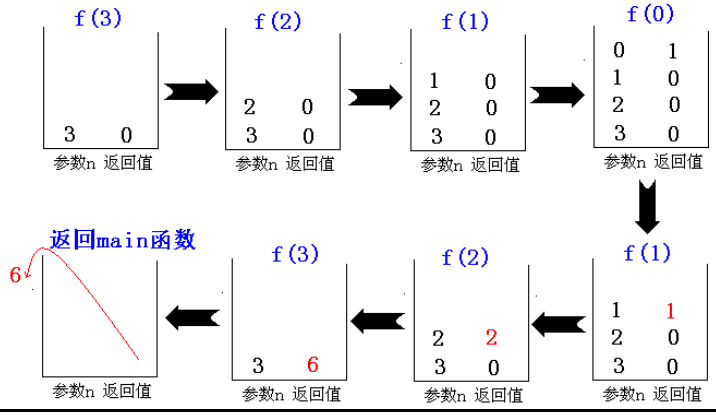


2、递归调用栈

递归调用就是函数自己调用自己，但在系统中，调用的并不是真的是自己！而是新开一个栈帧~！例如

```
int f(int n)
{
     if(n==0) return 1;
     return f(n-1)*n;
}
int main()
{
    int ans=f(20);
    printf("%d\n",ans);
    return 0;
}
```

递归调用栈帧的变化过程：



因此我们可以用自己定义则栈来代替系统调用栈：

```
struct data{int n,v;}st[20];     //自定义栈
int top;

int F(int n)
{
    int t;
    top=0;
    CALL:
        st[++top]=(data){n,0};     //参数和函数值进栈
        if(st[top].n==0)
        {
           st[top].v=1;
           goto RET;     //边界返回值
        }
        n=n-1;
        goto CALL;   调用
        RET:
          t=st[--top].v;   //退栈：t=f(n-1)
          if(top)
          {
               st[top].v=t*st[top].n;    //计算 n*f(n-1)
               goto RET;   //继续退栈
          }
          return t;
}
```

3、斐波那契数列递归改成非递归示例

```
int f(int n)
{
    int v;
    if(n<3) return 1;
    v=0;
    for(int i=1;i<3;i++)
       v=v+f(n-i);
    return v;
}
```

```
struct data{ int n,v,i;}st[1005];
int top=0;
int ff(int n)
{
    top=0;
    int v,i;
    CALL:
        st[++top]=(data){n,0,0};
        if(n<3)
        {
            st[top].v=1;
            goto END;
        }
       for(i=1;i<=2;i++)
       {
              st[top].i=i;
              n=n-i;
              goto CALL;
              RET:
               ;
       }
      END:
        v=st[top].v;
        top--;
        if(top)
         {
              st[top].v+=v;
```

```
            n=st[top].n;
            i=st[top].i;
            goto RET;
        }
    return v;
}
```