高级数据结构-栈

信智学院　陈琰宏

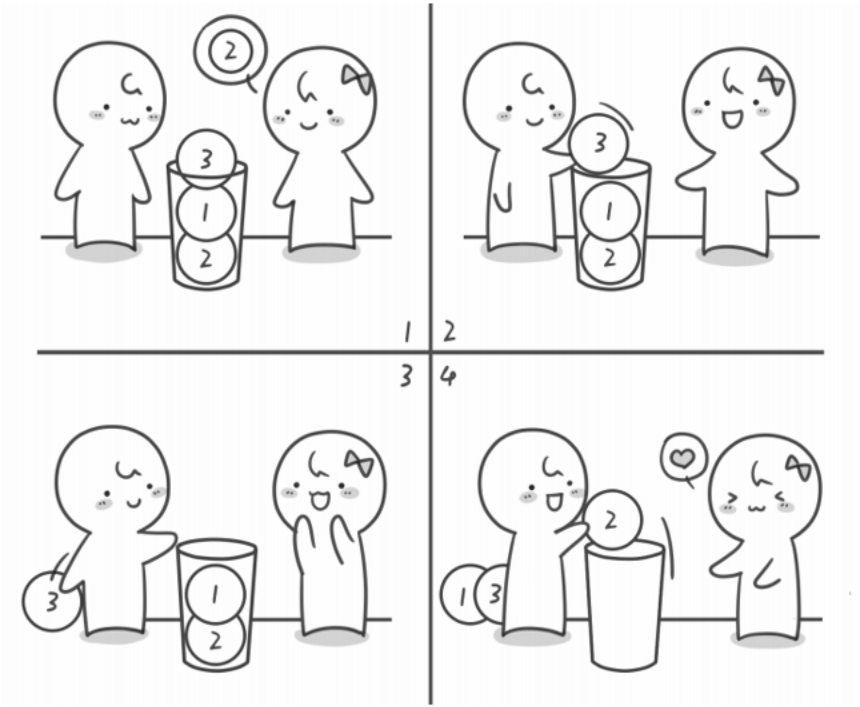
主要内容

01 栈的定义与存储

02 栈的操作（增删改查等）

03 栈的应用

3.1 栈的定义



3.1 栈的定义

注意：一般在绘制数组元素顺序图时按下标从上到下排列各元素。此处为了更形象的表达栈的结构，将第0个元素放在下方。

3.1 栈的定义

“栈（Stack）”可以认为是具有一定操作约束的线性表，插入和删除操作都作用在一个称为栈顶(Top)的端点位置。

- 把数据插入称为压入栈（Push）；
- 而数据删除可看作从堆栈中取出数据，叫做弹出栈（Pop）。
- 最后入栈的数据将被最先弹出，所以堆栈也被称为“后入先出”表（Last In First Out，简称LIFO）。

3.1 栈的定义

1、Stack CreateStack(int MaxSize)： 生成空堆栈，其最大长度为MaxSize；

2、int IsFull(Stack S, int MaxSize)：判断堆栈S是否已满。若S中元素个数等于MaxSize时返回1(TRUE)；否则返回0(FALSE) ；

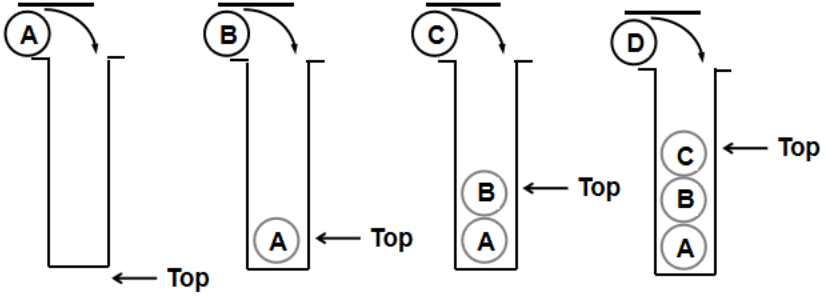
3、void Push(Stack S, ElementType item)：将元素item压入堆栈。若堆栈已满，返回堆栈为满信息；否则将数据元素item插入到堆栈S栈顶处；

4、int IsEmpty (Stack S)：判断堆栈S是否为空，若是返回1(TRUE)；否则返回0(FALSE)；

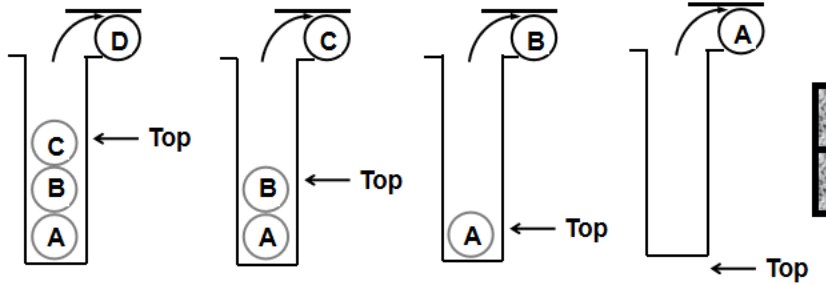
5、ElementType Pop(Stack S)：删除并返回栈顶元素。若堆栈为空，返回堆栈为空信息；否则将栈顶数据元素从堆栈中删除并返回。

6、 ElementType Top(Stack S)：……

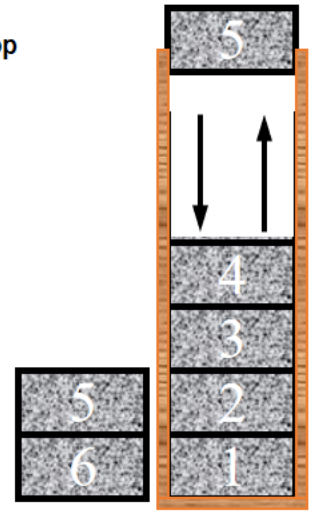
3.1 栈的定义



CreatStack( ); Push(S,A); Push(S,B); Push(S,C);



x=Pop(S); x=Pop(S); x=Pop(S); IsEmpty (S)



3.1 栈的定义-例1

[例1] 如果将ABCD四个字符按顺序压入堆栈，
是不是ABCD的所有排列都可能是出栈的序列？
可以产生CABD这样的序列吗？

【分析】如果有三个字符ABC出入栈时，全排列有3！ = 6 种可能。

- 其中的CAB是没法生成的。

1 [illegible]
进栈，所以C出栈时AB必然还在栈里，而且A还压在B下面。

2 [illegible] 如果进栈顺序为AB，那么出栈的系列AB、BA都有可能：

- 如果有四个字符ABCD出入栈时，排列有4!=24种可能。
- 也可以是所有排列都有可能是出栈的序列，象这样的序列是
这两个字符的排列也是正好是两个字符的全排列。
  - 四个字符出栈的所有可能序列有几种？ （14种）

3.1 栈的定义

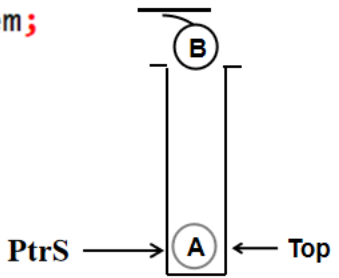
栈的顺序存储结构通常由一个一维数组和一个记录栈顶元素位置的变量组成。

```
#define MaxSize  <储存数据元素的最大个数>
typedef struct {
        ElementType Data[MaxSize];
        int Top;
} Stack;
```

3.2 栈的实现-push

(1)入栈

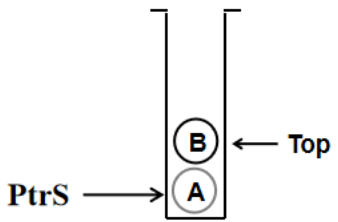
```
1  void Push( Stack *PtrS, ElementType item )
2  {
3      if ( PtrS->Top == MaxSize-1 ) {
4          printf(“ 堆栈满” );  return;
5      }
6      else
7      {
8          PtrS->Data[++(PtrS->Top)] = item;
9          return;
10     }
11 }
```



3.2 栈的实现-pop

(1)出栈

```
1  ElementType Pop( Stack *PtrS )
2  {
3      if ( PtrS->Top == -1 ) {
4          printf(“ 堆栈空” );
5          return ERROR; /* 标志错误 */
6      }
7      else
8          return ( PtrS->Data[(PtrS->Top)--] );
9  }
```



3.2 栈的实现-例2

[例2] 请用一个数组实现两个堆栈，要求最大地利用数组空间，使数组只要有空间入栈操作就可以成功。写出相应的入栈和出栈操作函数。

【分析】 一种比较聪明的方法是使这两个栈分别从数组的两头开始向中间生长；当两个栈的栈顶指针相遇时，表示两个栈都满了。此时，最大化地利用了数组空间。

```
#define MaxSize <存储数据元素的最大个数>
struct DStack {
    ElementType Data[MaxSize];
    int Top1;  /* 堆栈1的栈顶指针 */
    int Top2;  /* 堆栈2的栈顶指针 */
} S;
```

```
S.Top1 = -1;
S.Top2 = MaxSize;
```

3.2 栈的实现-例2

```
1  void Push( struct DStack *PtrS, ElementType item, int Tag )
2  {  /* Tag作为区分两个堆栈的标志，取值为1和2 */
3     if ( PtrS->Top2 -  PtrS->Top1 == 1) {  /*堆栈满*/
4        printf(“ 堆栈满” );   return ;
5     }
6     if ( Tag == 1 ) /* 对第一个堆栈操作 */
7        PtrS->Data[++(PtrS->Top1)] = item;
8     else                /* 对第二个堆栈操作 */
9        PtrS->Data[--(PtrS->Top2)] = item;
10 }
```

3.2 栈的实现-例2

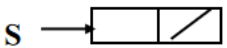
```
1  ElementType Pop( struct DStack *PtrS, int Tag )
2  {  /* Tag作为区分两个堆栈的标志，取值为1和2  */
3    if ( Tag == 1 ) {  /* 对第一个堆栈操作  */
4        if ( PtrS->Top1 == -1 ) { /*堆栈1空 */
5        printf(" 堆栈1空" );   return NULL;
6        } else return PtrS->Data[(PtrS->Top1)--];
7     }
8     else {           /* 对第二个堆栈操作 */
9        if ( PtrS->Top2 == MaxSize ) { /*堆栈2空 */
10           printf(" 堆栈2空" );  return NULL;
11       } else  return PtrS->Data[(PtrS->Top2)++];
12    }
13 }
```

3.2 栈的实现-链表

栈的链式存储结构实际上就是一个单链表，叫做链栈。插入和删除操作只能在链栈的栈顶进行；栈顶指针Top就是链表的头指针。

```
typedef struct Node{
        ElementType  Data;
        struct Node *Next;
} LinkStack;
LinkStack *Top;
```

(1) 堆栈初始化（建立空栈）
(2) 判断堆栈S是否为空




```
LinkStack *CreateStack()
{   /* 构建一个堆栈的头结点，返回指针 */
    LinkStack *S;
    S = malloc( sizeof(struct Node ));
    S->Next = NULL;
    return S;
}
int IsEmpty( LinkStack *S )
{   /*判断堆栈S是否为空，若为空函数返回整数
1，否则返回0 */
    return ( S->Next == NULL );
}
```

3.2 栈的实现-链表

```
1  void Push( ElementType item, LinkStack *S )
2  {  /* 将元素item压入堆栈S  */
3      struct Node *TmpCell;
4      TmpCell = malloc( sizeof( struct Node ) );
5      TmpCell->Element = item;
6      TmpCell->Next = S->Next;
7      S->Next = TmpCell;
8  }
9
10 ElementType Pop( LinkStack *S )
11 {  /* 删除并返回堆栈S的栈顶元素 */
12     struct Node *FirstCell;
13     ElementType TopElem;
14     if( IsEmpty( S ) ) {
15     printf(" 堆栈空" );  return NULL;
16     } else {
17     FirstCell = S->Next;
18     S->Next = FirstCell->Next;
19     TopElem = FirstCell ->Element;
20     free(FirstCell);
21     return TopElem;
22     }
23 }
```

3.3 栈的应用

- 进制转换
- 超市小车
- 表达式求解
- 递归模拟
- 迷宫求解

3.3.1 [7289]模拟栈

实现一个栈，栈初始为空，支持四种操作：
(1) “push x” – 向栈顶插入一个数x；
(2) “pop” – 从栈顶弹出一个数；
(3) “empty” – 判断栈是否为空；
(4) “query” – 查询栈顶元素。
现在要对栈进行M个操作，其中的每个操作3和操作4都要输出相应的结果。
输入格式：第一行包含整数M，表示操作次数。接下来M行，每行包含一个操作命令，操作命令为”push x”， ”pop”，”empty”，”query”中的一种。
输出格式
对于每个”empty”和”query”操作都要输出一个查询结果，每个结果占一行。其中，”empty”操作的查询结果为“YES”或“NO”，”query”操作的查询结果为一个整数，表示栈顶元素的值。
数据范围
1≤M≤100000 1≤x≤109所有操作保证合法。

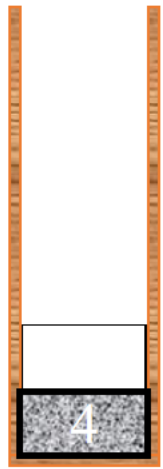
3.3.1 [7289]模拟栈

样例输入

```
10
push 5
query
push 6
pop
query
pop
empty
push 4
query
empty
```

样例输出

```
5
5
YES
4
NO
```



3.3.1 [7289]模拟栈-方法1.1结构体

方法1 结构体定义栈

```
1   #include<iostream>
2   using namespace std;
3   const int N = 1e5+5;
4   int x,m;
5   string str;
6   struct stack{
7       int sta[N] , topp;
8   }s;
9   void init(stack &st){ st.topp=0; }//结构体内的初始化
10  void push(stack &st,int x){ st.sta[++st.topp] = x;}
11  void pop(stack &st){ st.topp--;}
12  int top(stack &st){ return st.sta[st.topp];}
13  int size(stack &st){ return st.topp;}
14  bool empty(stack &st){ return st.topp==0?1 : 0;}
```

3.3.1 [7289]模拟栈-方法1.1结构体

方法1 结构体定义栈

```
16  int main(){
17      scanf("%d",&m);
18      while(m--) {
19          cin>>str;
20          if(str=="push") { cin>>x; push(s,x);}
21          else if(str=="pop") pop(s);
22          else if(str=="empty") empty(s)?cout<<"YES\n" : cout<<"NO\n";
23          else cout<<top(s)<<endl;
24      }
25      return 0;
26  }
```

3.3.1 [7289]模拟栈-方法1.2结构体

方法1 结构体定义栈

```
1  #include<iostream>
2  using namespace std;
3  const int N = 1e5+5;
4  int x,m;
5  string str;
6  struct stack{
7      int sta[N] , topp;
8  }s;
9  void init(stack *st){ st->topp=0; }//结构体内的初始化
10 void push(stack *st,int x){ st->sta[++st->topp] = x;}
11 void pop(stack *st){ st->topp--;}
12 int top(stack *st){ return st->sta[st->topp];}
13 int size(stack *st){ return st->topp;}
14 bool empty(stack *st){ return st->topp==0?1 : 0;}
```

3.3.1 [7289]模拟栈-方法1.2结构体

方法1 结构体定义栈

```
16  int main(){
17      scanf("%d",&m);
18      while(m--) {
19          cin>>str;
20          if(str=="push") { cin>>x; push(&s,x);}
21          else if(str=="pop") pop(&s);
22          else if(str=="empty")
23              empty(&s)?cout<<"YES\n" : cout<<"NO\n";
24          else cout<<top(&s)<<endl;
25      }
26      return 0;
27  }
```

3.3.1 [7289]模拟栈-方法2类

方法2 结构体模拟栈（类）

```
1  #include<iostream>
2  #define R register int
3  using namespace std;
4  const int N = 1e5+5;
5  int x,m;
6  string str;
7  struct stack{
8      int sta[N] , topp;
9      stack(){ topp=0; }//结构体内的初始化
10     void push(int x){ sta[++topp] = x;}
11     void pop(){ topp--;}
12     int top(){ return sta[topp];}
13     int size(){ return topp;}
14     bool empty(){ return topp==0?1 : 0;}
15 };
16 struct stack s;
```

3.3.1 [7289]模拟栈-方法2类

```
19 int main(){
20     scanf("%d",&m);
21     while(m--) {
22         cin>>str;
23         if(str=="push") { cin>>x; s.push(x);}
24         else if(str=="pop") s.pop();
25         else if(str=="empty")
26             s.empty() ? cout<<"YES\n" : cout<<"NO\n";
27         else cout<<s.top()<<endl;
28     }
29     return 0;
30 }
```

3.3.1 [7289]模拟栈-方法3数组

方法3 数组模拟栈

```
1   #include <iostream>
2   using namespace std;
3   const int N = 100010;
4   int stk[N], tt;
5   void push(int x){stk[++ tt] = x;}
6   void pop(){   tt --;}
7   int empty()
8   {
9       if(tt > 0) return 1;
10      else return 2;
11  }
12  int query(){  printf("%d\n", stk[tt]) ;}
```

3.3.1 [7289]模拟栈-方法3数组

方法3 数组模拟栈

```
15  int main()
16  {
17      int m;
18      cin >> m ;
19      while(m --)
20      {
21          string op;
22          int x;
23          cin >> op;
24          if(op == "push"){cin >> x;push(x); }
25          else if(op == "pop")pop();
26          else if(op == "query")query();
27          else
28          {
29              if(empty() == 1) printf("NO\n");
30              else printf("YES\n");
31          }
32      }
33      return 0;
34  }
```

3.3.1 [7289]模拟栈-方法4 STL

方法4 STL

```
4   const int N=100010;
5   int n;
6   int main(){
7       stack <int> stk;
8       cin>>n;
9       while(n--){
10          string op;
11          int x;
12          cin>>op;
13          if(op=="push")cin>>x,stk.push(x);
14          else if(op=="pop")stk.pop();
15          else if(op=="empty"){
16              if(!stk.empty())
17                  cout<<"NO"<<endl;
18              else cout<<"YES"<<endl;;
19          }
20          else
21            cout<<stk.top()<<endl;
22      }
23      return 0;
24  }
```

3.3.2 [1521]进制转换

把十进制到二进制的转换。
输入
234
输出
11101010

3.3.2 [1521]进制转换

```cpp
#include<iostream>
using namespace std;
struct stack{
    int st;
    int data[100005];
    stack()
    {
        st=-1;
    }
    bool empty(){
        if(st==-1)return 1;
        return 0;
    }
    void pop(){
        st--;
    }
    void push(int a){
        data[++st]=a;
    }
    int top(){
        return data[st];
    }
};
int main(){
    int n;
    while(cin>>n){
        stack st;
        while(!st.empty()){
            st.pop();
        }
        while(n)
        {
            st.push(n%2);
            n/=2;
        }
        while(!st.empty())
        {
            cout<<st.top();
            st.pop();
        }
        cout<<endl;
    }
    return 0;
}
```

3.3.2 [1521]进制转换

```cpp
#include<iostream>
#include<stack>
using namespace std;
int main(){
    int n,y;
    stack<int> k;
    while(cin>>n){
        while(n!=0){
            y=n%2;
            n=n/2;
            k.push(y);
        }
        while(!k.empty()){
            cout<<k.top();
            k.pop();
        }
        cout<<endl;
    }
    return 0;
}
```

```cpp
#include<iostream>
using namespace std;
int main()
{
    int n,i=0;
    cin>>n;
    int a[100000];
    memset(a,0,sizeof(a));
    while(n!=0)
    {
        i++;
        a[i]=n%2;
        n=n/2;
    }
    int re=i;
    for(int i=re;i>=1;i--)
        cout<<a[i];
    cout<<endl;
    return 0;
}
```

3.3.3 [2098]表达式求解

简单计算器

读入一个只包含 +, -, *, / 的非负整数计算表达式，计算该表达式的值。测试输入包含若干测试用例，每个测试用例占一行，每行不超过200个字符，整数和运算符之间用一个空格分隔。没有非法表达式。当一行中只有0时输入结束，相应的结果不要输出。

对每个测试用例输出1行，即该表达式的值，精确到小数点后2位。

样例输入

```
1 + 2
4 + 2 * 5 - 7 / 11
0
```

样例输出

```
3.00
13.36
```

3.3.3 表达式求解

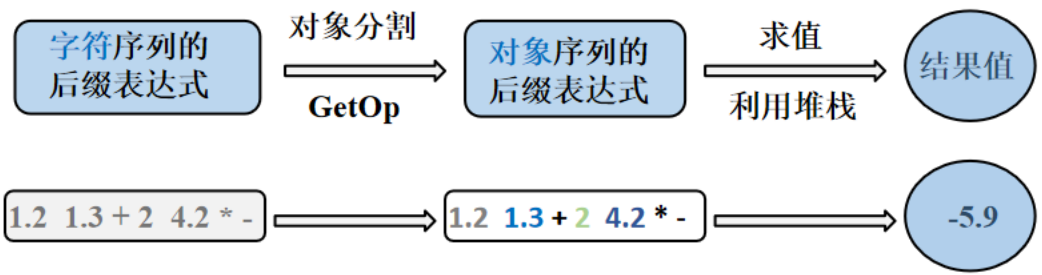
➢ 应用栈实现后缀表达式求值的基本过程：

. 从左到右读入后缀表达式的各项（运算符或运算数）；

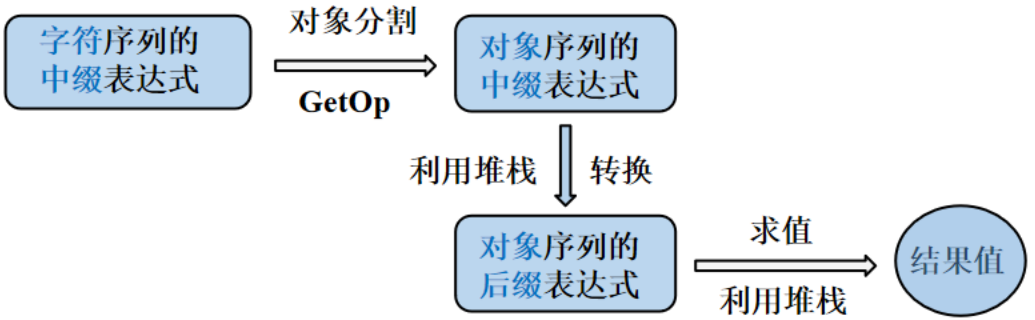
. 根据读入的对象（运算符或运算数）判断执行操作；

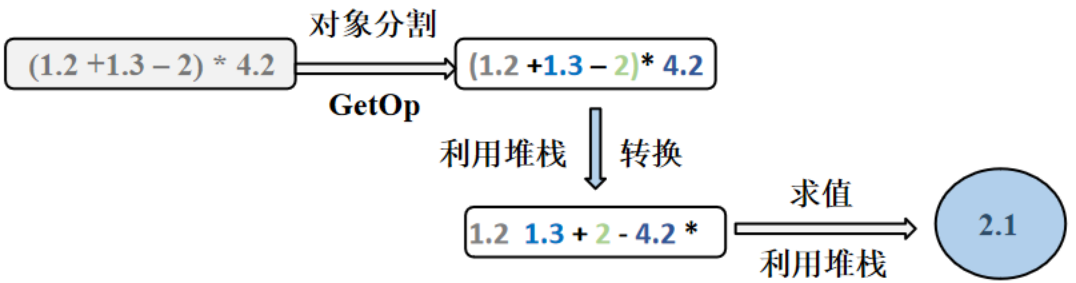
. 操作分下列3种情况：

1. 当读入的是一个运算数时，把它被压入栈中；
2. 当读入的是一个运算符时，就从堆栈中弹出适当数量的运算数，对该运算进行计算，计算结果再压回到栈中；
3. 处理完整个后缀表达式之后，堆栈顶上的元素就是表达式的结果值。



❖ 中缀表达式求值


字符序列的
中缀表达式
对象分割
GetOp
对象序列的
中缀表达式
利用堆栈
转换
对象序列的
后缀表达式
求值
利用堆栈
结果值


(1.2 +1.3 – 2) * 4.2
对象分割
GetOp
(1.2 +1.3 – 2)* 4.2
利用堆栈
转换
1.2 1.3 + 2 - 4.2 *
求值
利用堆栈
2.1

❖中缀表达式转换为后缀表达式

➢ 从头到尾读取中缀表达式的每个对象，对不同对象按不同的情况处理。对象分下列6种情况：

① 如果遇到空格则认为是分隔符，不需处理；
②若遇到运算数，则直接输出；
③若是左括号，则将其压入堆栈中；
④若遇到的是右括号，表明括号内的中缀表达式已经扫描完毕，将栈顶的运算符弹出并输出，直到遇到左括号（左括号也出栈，但不输出）；
⑤若遇到的是运算符，若该运算符的优先级大于栈顶运算符的优先级时，则把它压栈；若该运算符的优先级小于等于栈顶运算符时，将栈顶运算符弹出并输出，再比较新的栈顶运算符，按同样处理方法，直到该运算符大于栈顶运算符优先级为止，然后将该运算符压栈；
⑥若中缀表达式中的各对象处理完毕，则把堆栈中存留的运算符一并输出。

〖例〗 $a*(b+c)/d$ = ?　　$a\ b\ c\ +\ *\ d\ /$

输出: $a\ b\ c\ +\ *\ d\ /$

输入对象: a (操作数)	输入对象: $*$ (乘法)
输入对象: $($ (左括号)	输入对象: b (操作数)
输入对象: $+$ (加法)	输入对象: c (操作数)
输入对象: $)$ (右括号)	输入对象: $/$ (除法)
输入对象: d (操作数)	

$* \geq ($?

$(< +$?

$* \geq /$?

← top

No?!

$T(N) = O(N)$

❖中缀转换为后缀示例：（2*（9+6/3-5）+4）

步骤	待处理表达式	堆栈状态（底←→顶）	输出状态
1	2*（9+6/3-5）+4		
2	*（9+6/3-5）+4		2
3	（9+6/3-5）+4	*	2
4	9+6/3-5）+4	*（	2
5	+6/3-5）+4	*（	2 9
6	6/3-5）+4	*（+	2 9
7	/3-5）+4	*（+	2 9 6
8	3-5）+4	*（+/	2 9 6
9	-5）+4	*（+/	2 9 6 3
10	5）+4	*（-	2 9 6 3 / +
11	）+4	*（-	2 9 6 3 / + 5
12	+4	*	2 9 6 3 / + 5 -
13	4	+	2 9 6 3 / + 5 - *
14		+	2 9 6 3 / + 5 - * 4
15			2 9 6 3 / + 5 - * 4 +

❖中缀表达式转换为后缀表达式-总结

中缀表达式转后缀表达式遵循以下原则：

1.遇到操作数，直接输出；

2.栈为空时，遇到运算符，入栈；

3.遇到左括号，将其入栈；

4.遇到右括号，执行出栈操作，并将出栈的元素输出，直到弹出栈的是左括号，左括号不输出；

5.遇到其他运算符'+''-''*''/'时，*弹出所有优先级大于或等于该运算符的栈顶元素，然后将该运算符入栈*；

6.最终将栈中的元素依次出栈，输出。

❖中缀表达式转换为后缀表达式-总结

while (若exp未读完)
{　从exp读取字符ch;
ch为数字：将后续的所有数字均依次存放到postexp中,并以字符'#'标志数值串结束;
ch为左括号'('：将'('进栈;
ch为右括号')'：将op栈中'('以前的运算符依次出栈并存放到postexp中,再将'('退栈;
若ch的优先级高于栈顶运算符优先级，则将ch进栈;否则退栈并存入postexp中,再将ch进栈;
}
若字符串exp扫描完毕,则退栈op中的所有运算符并存放到postexp中。

例子

求 a+b*c+(d*e+f)*g 的后缀表达式

a+b*c+(d*e+f)*g ---> abc*+de*f+g*+

3.3.3 [2098]表达式求解

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   stack<double>d_st;//数字
4   stack<char>op_st;//运算符
5   int level(char s){//定义优先级
6       if(s=='+'||s=='-')return 1;
7       if(s=='*'||s=='/')return 2;
8   }
9   int main(){//40 + 21 * 100 - 7 / 11
10      char str[200];
11      double x;
12      while(gets(str)){//读入字符串
13          if(str[0]=='0')break;
14          for(int i=0;i<strlen(str);i++){//处理字符
44          while(!op_st.empty()){//字符处理完，栈内符号顺序出栈
56          printf("%.2lf\n",d_st.top());
57          while(!d_st.empty())d_st.pop();
58      }
59      return 0;
60  }
```

3.3.3 [2098]表达式求解

```
for(int i=0;i<strlen(str);i++){//处理字符
    if(str[i]>='0'&&str[i]<='9'){//处置数值-- 开始
        x=str[i]-'0';//转化为数值
        while(str[i+1]>='0'&&str[i+1]<='9') {
            x=x*10+str[i+1]-'0';
            i++;
        }
        d_st.push(x);
    }//处理数值--结束
```

3.3.3 [2098]表达式求解

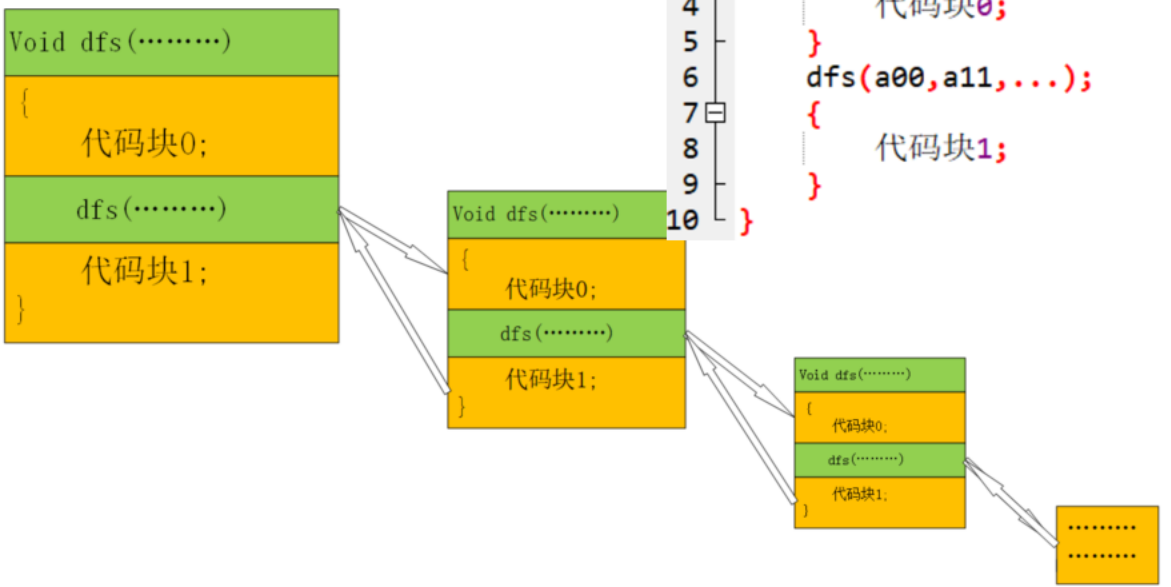
```
23          else if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/') {
24              //处理符号
25              if(op_st.empty()||level(str[i])>level(op_st.top()))
26                  op_st.push(str[i]);//栈为空或者当前操作符的优先级比栈顶操作
27              else if(level(str[i])<=level(op_st.top())) {//出栈，同时计算
28                  while(!op_st.empty()){
29                      double b,c;
30                      b=d_st.top();d_st.pop();
31                      c=d_st.top();d_st.pop();
32                      switch(op_st.top()){
33                          case '+':d_st.push(b+c);break;
34                          case '-':d_st.push(c-b);break;
35                          case '*':d_st.push(b*c);break;
36                          case '/':d_st.push(c/b);break;
37                      }
38                      op_st.pop();
39                  }
40                  op_st.push(str[i]);
41              }
42          } //end of   else if(str[i]=='+'||str....
43      } // end of   for(int i=0;i<strlen(str);i++)
```

3.3.3 [2098]表达式求解

```
44          while(!op_st.empty()){//字符处理完，栈内符号顺序出栈
45              double b,c;
46              b=d_st.top();d_st.pop();
47              c=d_st.top();d_st.pop();
48              switch(op_st.top()){
49                  case '+':d_st.push(b+c);break;
50                  case '-':d_st.push(c-b);break;
51                  case '*':d_st.push(b*c);break;
52                  case '/':d_st.push(c/b);break;
53              }
54              op_st.pop();
55          }
56          printf("%.2lf\n",d_st.top());
57          while(!d_st.empty())d_st.pop();
58      }
59      return 0;
60  }
```

3.3.4 栈求阶乘（模拟递归）

```
1   void dfs(int a0,int a1,...)
2   {
3       {
4           代码块0;
5       }
6       dfs(a00,a11,...);
7       {
8           代码块1;
9       }
10  }
```



3.3.4 栈模拟递归

递归过程分为以下五步：

1 执行代码块0
2 保存现场准备进入下一层
3 接受下层返回的数据
4 恢复现场
5 继续执行代码块1
上述五步相当重要

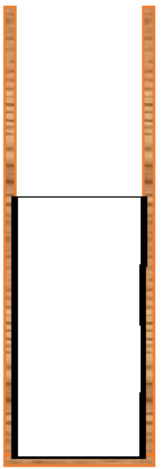
```
1  void dfs(int a0,int a1,...)
2  {
3      {
4          代码块0;
5      }
6      dfs(a00,a11,...);
7      {
8          代码块1;
9      }
10 }
```

在直接用递归程序实现递归时，**第二步和第四步都是编译器在帮助你完成**。而手写递归则需要自己用模拟栈来实现保存现场和恢复现场。

3.3.4 栈求阶乘

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

```
struct data
{
    int a0;//表示递归函数输入参数
    int v0;//该层递归结束时的返回值
    int ra;//记录应该执行哪个代码块了
};
```



在返回过程中返回值，求解 ret=n*ret

3.3.4 栈求阶乘

```
#include<bits/stdc++.h>
#include<stack>
using namespace std;

int ret;//不同递归层相互通信的信使，
        //也就是存放的普通递归里面return里面的值
struct data
{
    int a0;//表示递归函数输入参数和部分保存的“现场数据”
    int v0;//该层递归结束时的返回值
    int ra;//地址，记录应该执行哪个代码块了
};
stack<data> st;
```

3.3.4 栈求阶乘

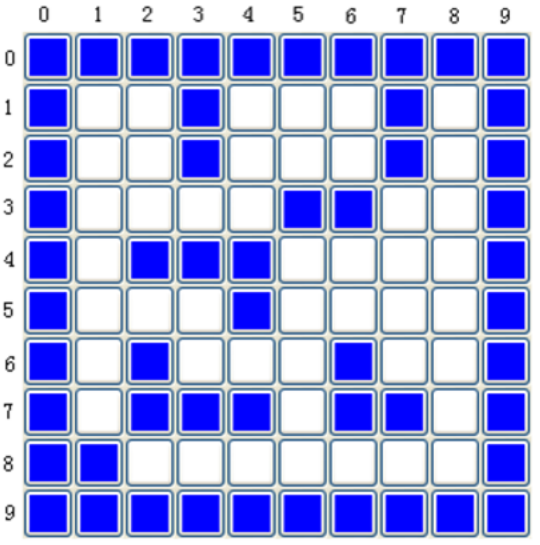
```
15  int fact(int n)
16  {
17      int ret;
18      st.push({n,0,0});//递归栈初始化，ra=0表示先进入代码块0
19      while(!st.empty())//栈不为空
20      {
21          data now=st.top();st.pop();
22          int a0=now.a0;
23          int ra=now.ra;
24          int v0=now.v0;
25          switch(ra)
26          {//0表示进入栈，1表示退出栈
27              case 0:
28              {
41              case 1:
42              {
47          }
48      }
49      return ret;
50  }
```

3.3.4 栈求阶乘

```
26          switch(ra)
27          {//0表示进入栈，1表示退出栈
28              case 0:
29              {
30                  if(a0==1)//递归结束条件
31                  {
32                      st.push({a0,1,1});//此条语句也可省略
33                      ret=1;//将本层返回值传给ret
34                  }
35                  else
36                  {
37                      st.push({a0,v0,1});//本层入栈，排队准备进入代码块1
38                      st.push({a0-1,v0,0});//下一层入栈，马上执行代码块0
39                  }
40                  break;
41              }
42              case 1:
43              {
44                  v0=a0*ret;//将下层的返回值和自己的a0相乘并变为自己的返回值
45                  ret=v0;//将自己的返回值赋给ret，供自己的上一层使用
46                  break;//不会有push语句，也就是对应正常递归里面的本层执行完毕就注销内
47              }
48          }
```

3.3.5 [3727] 迷宫3求路径次数

给定一个M×N的迷宫图，求从指定入口（1,1）到出口（M,N）有多少种走法。例如迷宫图如图所示（M=10，N=10），其中的方块图表示迷宫。对于图中的每个方块，用空白表示通道，用阴影表示墙。要求所求路径必须是简单路径，即在求得的路径上不能重复出现同一通道块。



3.3.5 [3727] 迷宫3求路径次数

```
1  #include<iostream>
2  #include<cstring>
3  #include<stack>
4  using namespace std;
5  int a[12][12];//定义迷宫
6  int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
7  int vis[12][12];//标记有没有走过
8  int sum=0,xe,ye;//xe,ye终点
9
10 struct node{
11     int x,y;
12     int dir;
13 };
```

3.3.5 [3727] 迷宫3求路径次数

```
15  int main(){
16      int m,n,i,j;
17      cin>>m>>n;
18      int xe=m,ye=n;
19      memset(a,1,sizeof(a));
20      for(i=1;i<=m;i++)
21          for(j=1;j<=n;j++)
22              cin>>a[i][j];
23
24      node start,t;
25      start.x=1,start.y=1,start.dir=0;
26      stack<node>st;
27      st.push(start);
28      while(!st.empty())
29      {
63      cout<<sum<<endl;
64      return 0;
65  }
```

3.3.5 [3727] 迷宫3求路径次数

```
28      while(!st.empty())
29      {
30          node e=st.top();
31          vis[e.x][e.y]=1;
32          if(e.x==xe&&e.y==ye)
33          {
34              sum++;
35              vis[e.x][e.y]=0;
36              st.pop();
37          }
38          while(!st.empty()&&st.top().dir==4)
39              vis[st.top().x][st.top().y]=0,st.pop();
40          if(st.empty())
41              break;
42          e=st.top();
43          st.pop();
```

3.3.5 [3727] 迷宫3求路径次数

```
44          for(int i=e.dir;i<4;i++)
45          {
46              int xx=e.x+dir[i][0];//下一个点
47              int yy=e.y+dir[i][1];
48              if(a[xx][yy]==0&&vis[xx][yy]==0){
49                  node r;
50                  r.x=xx,r.y=yy,r.dir=0;
51                  e.dir=i+1;
52                  st.push(e);
53                  st.push(r);
54                  break;
55              }
56              if(i==3)
57              {
58                  e.dir=i+1;
59                  st.push(e);
60              }
61          }
62      }
```

今天的课程结束啦……

下课了…
同学们再见！