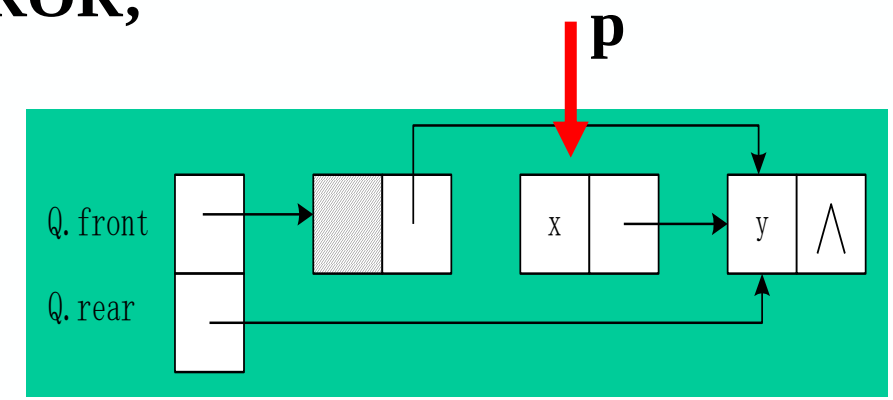
链队列出队

```
Status DeQueue (LinkQueue &Q,QElemType &e){
  if(Q.front==Q.rear) return ERROR;
  p=Q.front->next;
  e=p->data;
  Q.front->next=p->next;
  if(Q.rear==p) Q.rear=Q.front;
  delete p;
  return OK;
}
```



3.6 案例分析与实现

案例3.1： 数制的转换

【算法步骤】

① 初始化一个空栈S。

② 当十进制数N非零时，循环执行以下操作：

- 把N与8求余得到的八进制数压入栈S；
- N更新为N与8的商。

③ 当栈S非空时，循环执行以下操作：

- 弹出栈顶元素e；
- 输出e。

案例3.1： 数制的转换

【算法描述】

```
void conversion(int N)
{//对于任意一个非负十进制数，打印输出与其等值的八进制数
    InitStack(S);//初始化空栈S
    while(N)       //当N非零时,循环
    {
        Push(S,N%8);      //把N与8求余得到的八进制数压入栈S
        N=N/8;            //N更新为N与8的商
    }
    while(!StackEmpty(S))//当栈S非空时，循环
    {
        Pop(S,e);         //弹出栈顶元素e
        cout<<e;          //输出e
    }
}
```

案例3.2：括号的匹配

【算法步骤】

① 初始化一个空栈S。

② 设置一标记性变量flag，用来标记匹配结果以控制循环及返回结果，1表示正确匹配，0表示错误匹配，flag初值为1。

③ 扫描表达式，依次读入字符ch，如果表达式没有扫描完毕或flag非零，则循环执行以下操作：

- **若ch是左括号“[”或“(”，则将其压入栈；**
- **若ch是右括号“)”，则根据当前栈顶元素的值分情况考虑：若栈非空且栈顶元素是“(”，则正确匹配，否则错误匹配，flag置为0；**
- **若ch是右括号“]”，则根据当前栈顶元素的值分情况考虑：若栈非空且栈顶元素是“[”，则正确匹配，否则错误匹配，flag置为0。**

④ 退出循环后，如果栈空且flag值为1，则匹配成功，返回true，否则返回false。

案例3.3： 表达式求值

算术四则运算规则

(1) 先乘除,后加减

(2) 从左算到右

(3) 先括号内,后括号外

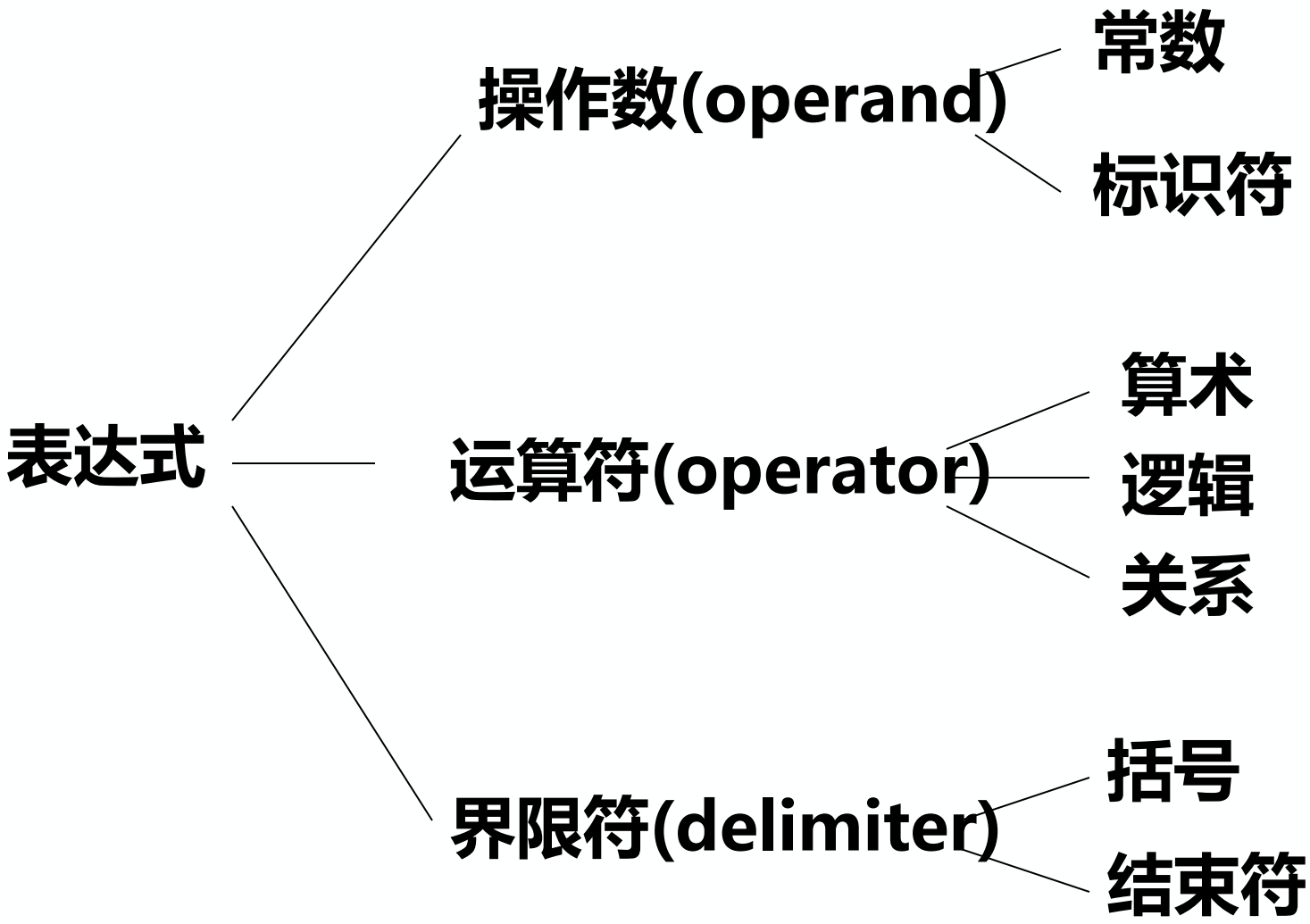

表达式
操作数(operand)
常数
标识符
运算符(operator)
算术
逻辑
关系
界限符(delimiter)
括号
结束符

表3.1　算符间的优先关系

θ_1 \ θ_2	+	-	*	/	(	)	#
+	>	>	<	<	<	>	>
-	>	>	<	<	<	>	>
*	>	>	>	>	<	>	>
/	>	>	>	>	<	>	>
(	<	<	<	<	<	=	x
)	>	>	>	>	x	>	>
#	<	<	<	<	<	x	=

2020年4月24日

【算法步骤】

设定两栈： OPND-----操作数或运算结果　OPTR------运算符

① 初始化OPTR栈和OPND栈，将表达式起始符“#”压入OPTR栈。

② 扫描表达式，读入第一个字符ch，如果表达式没有扫描完毕至“#”或OPTR的栈顶元素不为“#”时，则循环执行以下操作：

- 若ch不是运算符，则压入OPND栈，读入下一字符ch；
- 若ch是运算符，则根据OPTR的栈顶元素和ch的优先级比较结果，做不同的处理：
  - 若是小于，则ch压入OPTR栈，读入下一字符ch；
  - 若是大于，则弹出OPTR栈顶的运算符，从OPND栈弹出两个数，进行相应运算，结果压入OPND栈；
  - 若是等于，则OPTR的栈顶元素是“(”且ch是“)”，这时弹出OPTR栈顶的“(”，相当于括号匹配成功，然后读入下一字符ch。

③ OPND栈顶元素即为表达式求值结果，返回此元素。

```
OperandType  EvaluateExpression( ) {
 InitStack (OPTR);  Push (OPTR，'#') ;
 InitStack (OPND);  ch = getchar( );
 while (ch!= '#' || GetTop(OPTR)! = '#') {
  if (! In(ch)){Push(OPND,ch); ch = getchar(); }  // ch不是运算符则进栈
  else
    switch (Precede(GetTop(OPTR),ch)) { //比较优先权
      case '<' :   //当前字符ch压入OPTR栈，读入下一字符ch
          Push(OPTR, ch);  ch = getchar();  break;
      case '>' :   //弹出OPTR栈顶的运算符运算，并将运算结果入栈
          Pop(OPTR, theta);
          Pop(OPND, b);  Pop(OPND, a);
          Push(OPND, Operate(a, theta, b)); break;
      case '=' :   //脱括号并接收下一字符
          Pop(OPTR,x);    ch = getchar();   break;
      } // switch
 } // while
 return GetTop(OPND);} // EvaluateExpression
```

OPTR	OPND	INPUT	OPERATE
#		3*(7-2)#	Push(opnd,’3’)
#	3	*(7-2)#	Push(optr,’*’)
#,*	3	(7-2)#	Push(optr,’(’)
#,*,(	3	7-2)#	Push(opnd,’7’)
#,*,(	3,7	-2)#	Push(optr,’-’)
#,*,(,—	3,7	2)#	Push(opnd,’2’)
#,*,(,—	3,7,2	)#	Operate(7-2)
#,*,(	3,5	)#	Pop(optr)
#,*	3,5	#	Operate(3*5)
#	15	#	GetTop(opnd)