程序设计与算法(二)

郭 炜

递 归(二)

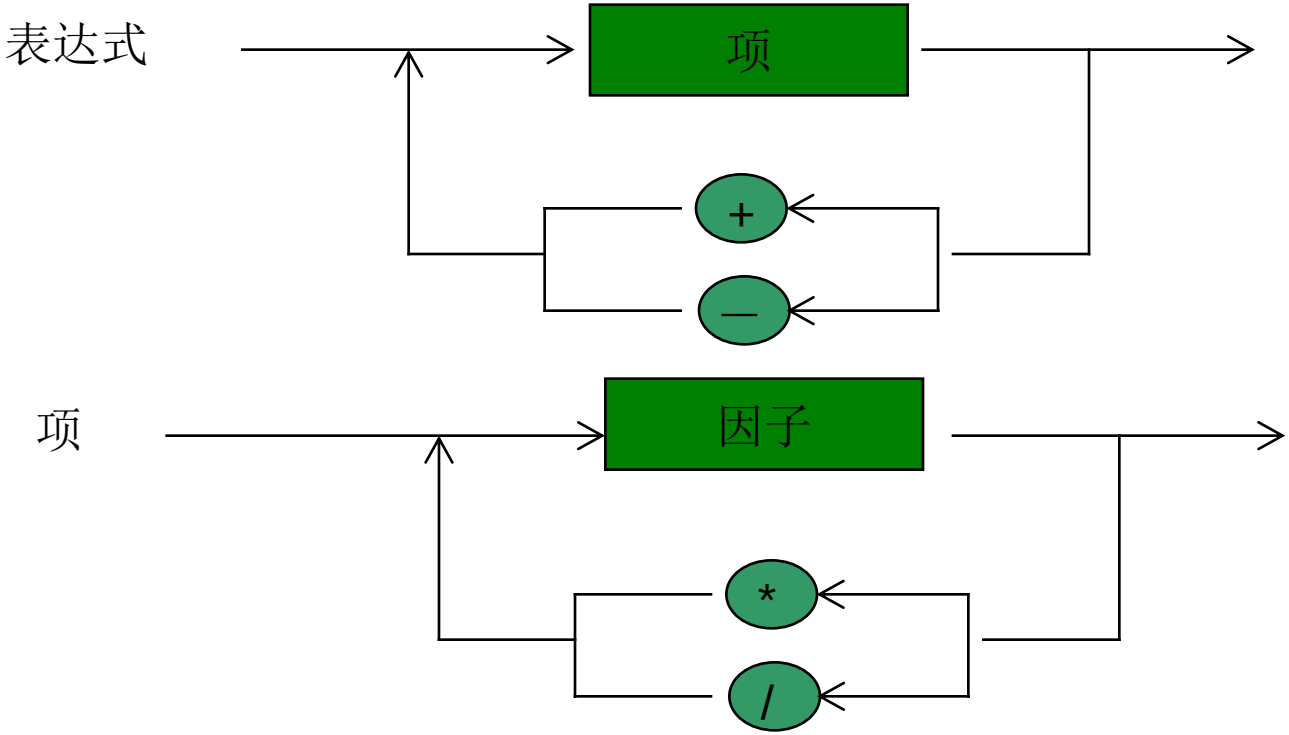
用递归解决递归形式的问题

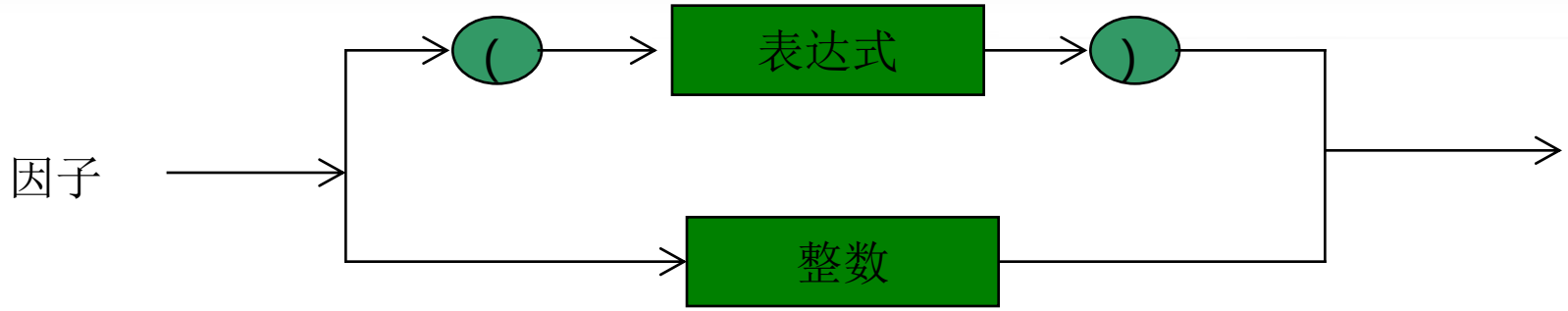
例题: 表达式计算

输入为四则运算表达式，仅由整数、+、—、*、/ 、(、)组成，没有空格，要求求其值。假设运算符结果都是整数。"/"结果也是整数

解题思想：

表达式是个递归的定义：





因此对于表达式可以进行递归分析处理

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
int factor_value();
int term_value();
int expression_value();
int main()
{
        cout << expression_value() << endl;
        return 0;
}
```

输入：(2+3)*(5+7)+9/3

输出： 63

```
int expression_value()    //求一个表达式的值
{
        int result = term_value(); //求第一项的值
        bool more = true;
        while( more) {
                char op = cin.peek();  //看一个字符,不取走
                if( op == '+' || op == '-' ) {
                        cin.get(); //从输入中取走一个字符
                        int value = term_value();
                        if( op == '+' )  result += value;
                        else   result -= value;
                }
                else  more = false;
        }
        return result;
}
```

```
int term_value()   //求一个项的值
{
        int result = factor_value(); //求第一个因子的值
        while(true) {
                char op = cin.peek();
                if( op == '*' || op == '/') {
                        cin.get();
                        int value = factor_value();
                        if( op == '*')
                                result *= value;
                        else   result /= value;
                }
                else
                        break;
        }
        return result;
}
```

```
int factor_value() //求一个因子的值
{
        int result = 0;
        char c = cin.peek();
        if( c == '(') {
                cin.get();
                result = expression_value();
                cin.get();
        }
        else {
                while(isdigit(c)) {
                        result = 10 * result + c - '0';
                        cin.get();
                        c = cin.peek();
                }
        }
        return result;
}
```

用递归将问题分解为规模更小的子问题进行求解

例题: 爬楼梯

树老师爬楼梯，他可以每次走1级或者2级，输入楼梯的级数，求不同的走法数

例如：楼梯一共有3级，他可以每次都走一级，或者第一次走一级，第二次走两级，也可以第一次走两级，第二次走一级，一共3种方法。

输入

输入包含若干行，每行包含一个正整数N，代表楼梯级数，1 <= N <= 30输出不同的走法数，每一行输入对应一行

爬楼梯

输出

不同的走法数，每一行输入对应一行输出

样例输入

5
8
10

样例输出

8
34
89

爬楼梯

n级台阶的走法 =

先走一级后，n-1级台阶的走法 +
先走两级后，n-2级台阶的走法

f(n) = f(n-1)+f(n-2)

边界条件：

爬楼梯

n级台阶的走法 =

先走一级后，n-1级台阶的走法 +
先走两级后，n-2级台阶的走法

f(n) = f(n-1)+f(n-2)

边界条件： n < 0   0
            n = 0   1

爬楼梯

n级台阶的走法 =

先走一级后，n-1级台阶的走法 +
先走两级后，n-2级台阶的走法

$f(n) = f(n-1)+f(n-2)$

边界条件：

$n < 0$	0	$n = 0$	1	$n = 1$	1
$n = 0$	1	$n = 1$	1	$n = 2$	2

```
#include <iostream>
using namespace std;
int N;
int stairs(int n)
{
	if( n < 0)
		return 0;
	if( n == 0 )
		return 1;
	return stairs(n-1) + stairs(n-2);
}
int main()
{
	while(cin >> N) {
		cout << stairs(N) << endl;
	}
	return 0;
}
```

例题：放苹果

把M个同样的苹果放在N个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？5，1，1和1，5，1 是同一种分法。

输入

第一行是测试数据的数目t（0 <= t <= 20）。以下每行均包含二个整数M和N，以空格分开。1<=M，N<=10。

输出

对输入的每组数据M和N，用一行输出相应的K。

样例输入

```
1
7 3
```

样例输出

```
8
```

例题：放苹果

设i个苹果放在k个盘子里放法总数是 **f(i,k)**，则：

例题：放苹果

设i个苹果放在k个盘子里放法总数是 **f(i,k)**，则：

k > i 时， f(i,k) = f(i,i)

例题：放苹果

设i个苹果放在k个盘子里放法总数是 **f(i,k)**，则：

k > i 时， f(i,k) = f(i,i)

k <= i 时，总放法 = 有盘子为空的放法+没盘子为空的放法

f(i,k) = f(i,k-1) + f(i-k,k)

边界条件？

```
#include <iostream>
using namespace std;
int f(int m,int n) {
        if( n > m )
                return f(m,m);
        if( m == 0)
                return 1;
        if( n <= 0 )
                return 0;
        return f(m,n-1) + f(m-n,n);
}
int main() {
        int t,m,n;
        cin >> t;
        while( t--) {
                cin >> m >> n;
                cout << f(m,n) << endl;
        }
        return 0;
}
```

例题：算24

给出4个小于10个正整数，你可以使用加减乘除4种运算以及括号把这4个数连接起来得到一个表达式。现在的问题是，是否存在一种方式使得得到的表达式的结果等于24。

这里加减乘除以及括号的运算结果和运算的优先级跟我们平常的定义一致（这里的除法定义是实数除法）。

比如，对于5，5，5，1，我们知道5 * (5 - 1 / 5) = 24，因此可以得到24。又比如，对于1，1，4，2，我们怎么都不能得到24。

例题：算24

输入

输入数据包括多行，每行给出一组测试数据，包括4个小于10个正整数。最后一组测试数据中包括4个0，表示输入的结束，这组数据不用处理。

输出

对于每一组测试数据，输出一行，如果可以得到24，输出“YES”；否则，输出“NO”。

样例输入

5 5 5 1
1 1 4 2
0 0 0 0

样例输出

YES
NO

例题：算24

n个数算24，必有两个数要先算。这两个数算的结果，和剩余n-2个数，就构成了n-1个数求24的问题

枚举先算的两个数，以及这两个数的运算方式。

边界条件?

注意：浮点数比较是否相等，不能用 ==

例题：算24

n个数算24，必有两个数要先算。这两个数算的结果，和剩余n-2个数，就构成了n-1个数求24的问题

枚举先算的两个数，以及这两个数的运算方式。

边界条件：一个数算24

注意：浮点数比较是否相等，不能用 ==

```
#include <iostream>
#include <cmath>
using namespace std;
double a[5];
#define EPS  1e-6
bool isZero(double x) {
        return fabs(x) <= EPS;
}
bool count24(double a[],int n)
{//用数组a里的 n个数，计算24
        if( n == 1 ) {
                if(isZero( a[0] - 24) )
                        return true;
                else
                        return false;
        }
```

```
double b[5];
for(int i = 0;i < n-1; ++i)
        for(int j = i+1;j < n; ++j) { //枚举两个数的组合
                int m = 0; //还剩下m个数, m = n - 2
                for(int k = 0; k < n; ++k)
                        if( k != i && k!= j)
                                b[m++] = a[k];//把其余数放入b
                b[m] = a[i]+a[j];
                if(count24(b,m+1))
                        return true;
                b[m] = a[i]-a[j];
                if(count24(b,m+1))
                        return true;
                b[m] = a[j]-a[i];
                if(count24(b,m+1))
                        return true;
                b[m] = a[i]*a[j];
                if(count24(b,m+1))
                        return true;
```

```
                        if( !isZero(a[j])) {
                                b[m] = a[i]/a[j];
                                if(count24(b,m+1))
                                        return true;
                        }
                        if( !isZero(a[i])) {
                                b[m] = a[j]/a[i];
                                if(count24(b,m+1))
                                        return true;
                        }
                }
        return false;
}
```

```
int main()
{
        while(true) {
                for(int i = 0;i < 4; ++i)
                        cin >> a[i];
                if( isZero(a[0]))
                        break;
                if( count24(a,4))
                        cout << "YES" << endl;
                else
                        cout << "NO" << endl;
        }
        return 0;
}
```