递归

递归应用场景

- 如果一个问题可以分解成子问题，并且子问题的求解方法何原问题相同，即：该问题可以“分而治之”，这样的情况可以考虑用递归函数实现问题的求解。
  - 快速排序、二分查找、树的遍历、阶乘、汉诺塔、八皇后、迷宫....
- 递归函数在定义中，一般都需要描述：
  - 什么时候进入递归
  - 什么时候不再递归

全排列

- 如果给定集合是{a,b,c,d}，可以用下面给出的简单算法产生其所有排列，即集合(a,b,c,d)的所有排列有下面的排列组成：

（1）以a开头后面跟着(b,c,d)的排列

（2）以b开头后面跟着(a,c,d)的排列

（3）以c开头后面跟着(a,b,d)的排列

（4）以d开头后面跟着(a,b,c)的排列

子问题和原问题解法相同，所以可以用递归实现

类似的，还有组合问题可以用递归实现

```
void permutation(char* a,int k,int m)  //全排列
{ int i,j;
  if(k == m) {
     for(i=0;i<=m;i++)   cout<<a[i];
     cout<<endl;
 } else
       for(j=k;j<=m;j++) {
         swap(a[j],a[k]);
         permutation(a,k+1,m);
         swap(a[j],a[k]);
      }
 }
```

练习：迷宫问题

- 知识点：二维数组+递归
- 需要从指定起始位置开始，上下左右平移进行递归（当前递归会生成4个子递归），只要有一个递归路径返回true，则最终输出yes，如果所有路径均返回false，则最终输出no。
- 递归实现的算法：
  - 如果当前位置就是终点，则return true;
  - 不是终点，根据当前位置的值

    1：不用处理，

    2：生命值-1；将当前位置的值修改为1；如果生命值=0，return false;

    3：生命值+3；将当前位置的值修改为1；拷贝现有位置访问记录；将访问记录数组清零

    上下左右递归，如果四个方向返回全部为0，复原当前位置的值，如果值为3，复原位置访问记录，return false；否则return true;

```
int dir[4][2] = {{-1,0},{1,0},{0,-1},{0,1}};
int vis[100][100]={0};
int vis2[100][100]={0};
int M[100][100]; int m,n;
bool Maze(int x1,int y1,int x2,int y2,int life)
{  if((x1==x2) &&(y1==y2))
      return true;
   int type = M[x1][y1];
   if(type == 2)
   {  life -= 1;
      M[x1][y1] = 1;
      if(life == 0)
        return false;
   }else if (type == 3) {
      life += 3;
      M[x1][y1] = 1;
for(int i=0;i<m;i++)
      for(int j=0;j<n;j++)
      {  vis2[i][j] = vis[i][j];
         vis[i][j] = 0;
      }
   }
   vis[x1][y1] = 1;
   int suc = 0;
   for(int i=0;i<4;i++)
   {  int nx = x1+dir[i][0];
      int ny = y1+dir[i][1];
      if(nx<0 || ny<0 || nx>=m || ny>=n ||
M[nx][ny]==0 || vis[nx][ny]==1)
         continue;
      suc += Maze(nx,ny,x2,y2,life);
   }
if(suc == 0)
   {  M[x1][y1] = type;
      vis[x1][y1] = 0;
      if(type == 3)
         for(int i=0;i<m;i++)
          for(int j=0;j<n;j++)
            vis[i][j] = vis2[i][j];
      return 0;
   }else
      return 1;
}
```

练习：折半查找

一组学生名单存于结构数组中，且已按学号从小到大排序。请设计C++函数，完成用折半法根据学号查找姓名的功能，函数原型为：

char *BiSearchR(Stu stu_array[], int first, int last, int id)

学生信息用如下类型表示：

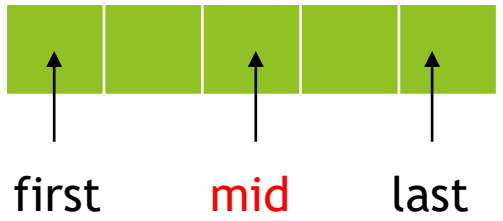
```
struct Stu
{  int id;
   char name[20];
};
```

完成main函数

```
const int N = 50;
char *BiSearchR(Stu stu_array[], int first, int last, int id);
int main( )
{     int num = 0;  Stu stu_a[N];
      for(int i=0;i<50;i++)
          cin>> stu_a[i].id>>stu_a[i].name;
      cout << "Input a student's id:";
      cin >> num;  //从键盘输入待查学生的学号
      char *x = BiSearchR(stu_a, 0, N-1, num);
      if(x == 0)
            cout << "The student's id is error.\n";
      else
            cout << "The student's name is: " << x << endl;
      return 0;
}
```

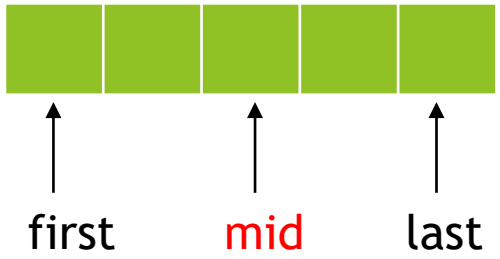
非递归算法

```
char *BiSearchR(Stu stu_array[], int first, int last, int id)
{    int mid;
     while(first<=last)
     {     mid = (first+last)/2;
          if(id == stu_array[mid].id)
               return  stu_array[mid].name;
          if(id >stu_array[mid].id )
               first = mid+1;
          else
               last = mid-1;
     }
     return 0;
}
```



练习：用递归函数实现折半查找

```
char *BiSearchR(Stu stu_array[], int first, int last, int id)
{   if(first > last)
       return 0;
    int mid = (first + last) / 2;
   if(id == stu_array[mid].id)
      return stu_array[mid].name;
   else if(id > stu_array[mid].id)
      return BiSearchR(stu_array, mid + 1, last, id);
   else
      return BiSearchR(stu_array, first, mid - 1, id);
}
```



练习：分别用循环与递归函数实现“台阶问题”

- 一个台阶总共有n级，如果一步可以跳1级，也可以跳2级，求到达第n级台阶的跳法总数。

分析：(1) n=1时，跳法总数为1

(2) n=2时,跳法总数为2

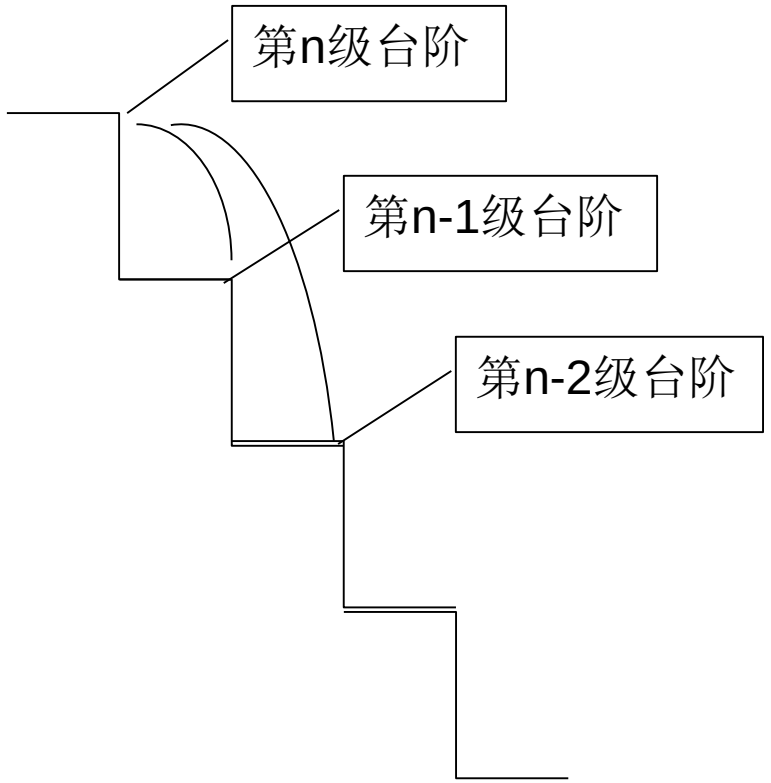
(3) 当n>2时

第n级台阶的跳法总数

= 第n-1级台阶的跳法总数

+

第n-2级台阶的跳法总数



类似斐波拉契数列

练习—台阶问题循环处理

```
int myStep(int n)
{  if(n==1)
        return 1;
   int step_1 = 1, step_2 = 2;
   for (int i = 3; i <= n; ++i)
   {   int temp = step_1 + step_2;
       step_1 = step_2 ;
       step_2 = temp;
    }
    return step_2;
}
```

练习：台阶问题递归处理

- 能分析出规律公式，直接翻译成C++语言

```
int myStepR(int n)
{  if(n == 1)
        return 1;
    else if( n == 2)
            return 2;
        else
            return myStepR(n-2) + myStepR(n-1);
}
```

练习：分苹果

- M个苹果放在N个盘子里，允许有的盘子空着不放，问共有多少种不同的放法？说明：假设3个盘子7个苹果，则5、1、1和1、5、1是同一种放法。

分析：

1、盘子数多于苹果数(N > M)：最多会放满M个盘子，放法和M个盘子时一样 。

2、 盘子数等于或少于苹果数 (N≤M)

(1) 每个盘子都有苹果：每个盘子至少1个苹果，剩下的M-N个苹果，放入N 个盘子中。(子问题)

(2) 如果至少有一个盘子是空的： 将M个苹果放入N-1个盘子中。(子问题)

```
int apple(int m, int n)
{  if (m == 0 || n == 1)      return 1;
   if (n > m)                 return apple(m, m);
   if (n<=m)                  return apple(m - n, n) + apple(m, n - 1);
}
```