第 6 章 递归算法

6.1 递归基础

一、什么是递归？

递归：函数的自我调用；

数列递归：如果一个数列的项与项之间存在关联性，那么可以使用递归实现；

原理：如果一个函数可以求 A(n)，那么该函数就可以求 A(n-1)，就形成了递归调用；

注意：一般起始项是不需要求解的，是已知条件；

递归求解问题的过程：

第一步：找出规律

第二步：函数调用自己求解前面的项

第三步：交代起始项，让递归能够停止

递归重要思想：

- 既然函数 fun()能够求第 n 项，那么它就能求第 n-1 项，也能求第 n+1 项；
- 既然函数 fun()能够解决一个问题的第 n 步，就能解决第 n-1 步，也能解决第 n+1 步；

递归算法解决问题的特点：

(1) 递归就是在过程或函数里调用自身。

(2) 在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。

(3) 递归算法解题通常显得很简洁，但递归算法解题的运行效率较低。所以一般不提倡用递归算法设计程序。

(4) 在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

由于递归的求解效率较低，且比较消耗内存，因此如果递归能够转化为循环，尽量用循环！

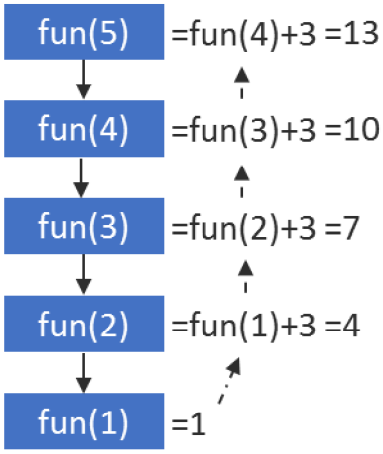
二、递归案例

例子：定义函数，递归求解等差数列 1　4　7　10　13……第 n 项的值

递归规律：A(n) = A(n-1) + 3

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

//求首项为 1，差值为 3 的等差数列的第 n 项
//A(n)=A(n-1)+3  function
int fun(int n){
    int r = 0;
    //递归一定要记得去交代递归的停止条件
    //让递归能够停下来
    if(n == 1){
        r = 1;
    }else{
        r = fun(n - 1) + 3;
    }

    return r;
```

```
}

int main(){
    for(int i = 1;i <= 10;i++){
        cout<<fun(i)<<endl;
    }
}
```



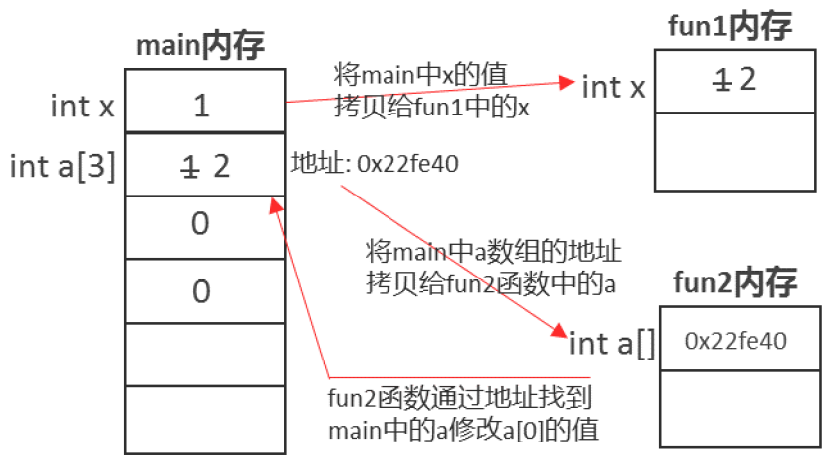
三、值传递和地址传递的区别

地址：变量在内存中的编号；

比如，数组的本质是 a[0]的地址！

指针：地址在 C++中叫做 指针 ！

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

void fun1(int x){
    x++;
}

void fun2(int a[]){
    a[0]++;
}

int main(){
    int x = 1;
    fun1(x);
    cout<<x<<endl;

    int a[3] = {0};
    a[0] = 1;
    cout<<a<<endl;

    fun2(a);
    cout<<a[0]<<endl;

    /*
    //以 0x 开头表示 16 进制
    int y = 0x1A;
    cout<<y<<endl;
    //以 0 开头表示 8 进制
    int z = 017;
    cout<<z<<endl;
    */
}
```



划定一块区域，供程序存储变量使用

注意：

原理：

1、每个函数运行时会生成一个独立的内存，来存储函数内部定义的变量，因此函数互相看不到对方内部定义的变量名，也不会出现变量名冲突的情况。

2、向函数中传递整数，本质是将整数的值拷贝给函数；向函数中传递数组，本质是将数组的地址拷贝给函数；

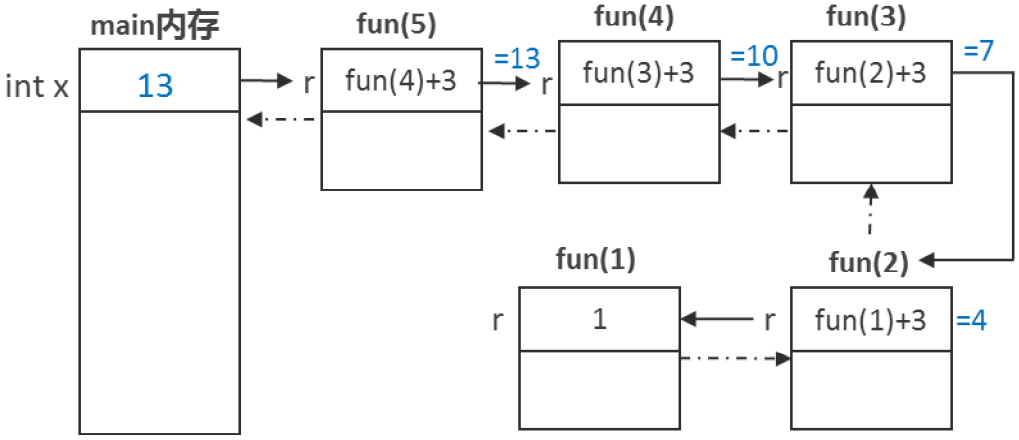
上述程序中，将 main 函数中的 x 拷贝给 change 函数的 x，实际上拷贝的是 main 函数中 x 的值；将 main 函数中 a 拷贝给 change2 函数中的 a，实际上拷贝的是 a 的地址；

结论： main 函数中的 x 和 change 函数中的 x 不是一个 x，但 main 函数中的 a 和 change 函数中的 a 是一个 a（地址）。

关键问题看传递给函数的是一个整数值还是一个数组的地址！

等差数列递归过程中的内存储存！

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

//求首项为 1，差值为 3 的等差数列的第 n 项
//A(n)=A(n-1)+3 function
int fun(int n){
    int r = 0;
    //递归一定要记得去交代递归的停止条件
    //让递归能够停下来
    if(n == 1){
        r = 1;
    }else{
        r = fun(n - 1) + 3;
    }

    return r;
}

int main(){
    int x = fun(5);
    cout<<x;
}
```



6.2 递归习题

要点：学会将循环问题改造成递归问题，并深入理解递归的执行过程！

一、数值类问题递归

1002：【入门】编程求解 1+2+3+...+n

解法一：使用公共变量，将每次递归产生的变量 i，加到总和上！

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

int n;
int s;//s 默认初始值 0

//递归数出 1~n 之间所有的数
void fun(int i){
    if(i <= n){
        //cout<<i<<endl;
        s = s + i;
        fun(i+1);
    }
}

int main(){
    cin>>n;
    fun(1);
    cout<<s;
}
```

思想：通过递归数数，fun(int)函数是为了输出 i 的值，既然 fun(int)可以输出 i 的值，就能输出 i+1 的值，也能输出 i+2 的值，也就是能输出 1~n 的每个数的值！

1、首先通过本问题让同学们理解递归的本质（函数自我调用），并从递归的功能角度先理解递归的作用（递归能解决第 n 项的问题，就能解决第 n+1 项的问题，也能解决第 n-1 项的问题）。本题中 fun(int)函数的作用，是为了递归产生 1~n 中的每个数，换言之，既然函数能 cout<<1，就能 cout<<2，就能 cout<<n，这就是递归！

2、能够通过 fun(int)打印出 1~n 的每一项，求和就很简单了。

3、理解局部变量和全局变量在递归中的区别，全局变量 s 和 n 在递归中，值是共享的，而局部变量（比如：i），是归每个函数独享的。需注意，如果是值不能共享的变量，在递归中，千万不要定义为公共的。

解法二：通过层层累加，然后将和层层返回，求和！

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

int n;//表示求和范围
//有返回值，通过层层累加，将和返回
int fun(int i){
    if(i <= n){
        return i + fun(i+1);
    }else{
        return 0;
    }
}

int main(){
    cin>>n;
    //从 1 开始递归
    cout<<fun(1);
}
```

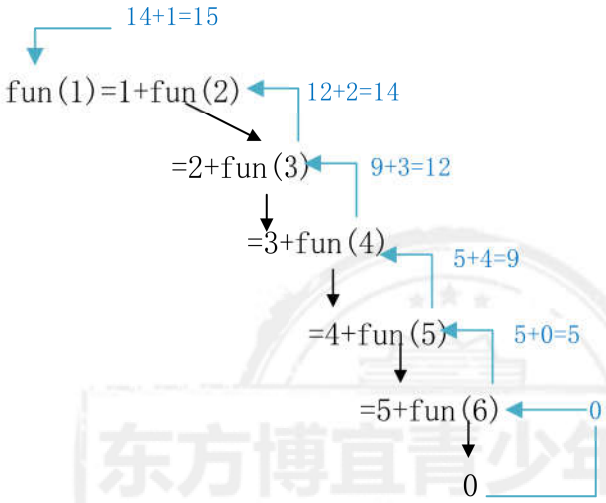
重点：

思想：fun(1~n 的和) = 1+fun(2~n)的和，因此 fun(int)函数的目的是为了求 1~n 之间所有数的和。

1、理解如何通过递归将值层层返回的过程。

2、本题的理解可以结合下方的图形，如下图所示，求 1~5 之间所有数的和，也就是 fun(1)表示从 1 开始求和。

fun(1)=1+fun(2)=1+2+fun(3)=1+2+3+fun(4)=1+2+3+4+fun(5)=1+2+3+4+5+fun(6)，由于 6>n（假设 n 为 5），因此递归停止，返回 0，那么 fun(1)=1+2+3+4+5+0=15。在递归图中，先通过黑色的线层层递归，再通过蓝色的线层层返回得到最终结果。



解法三：通过输入参数，层层累加求和！

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

int n;//表示求和范围
//有返回值，通过输入参数，层层累加，最后返回这个求和的输入参数
int fun(int i,int s){
    if(i <= n){
        return fun(i+1,s+i);
    }else{
        return s;
    }
}
```

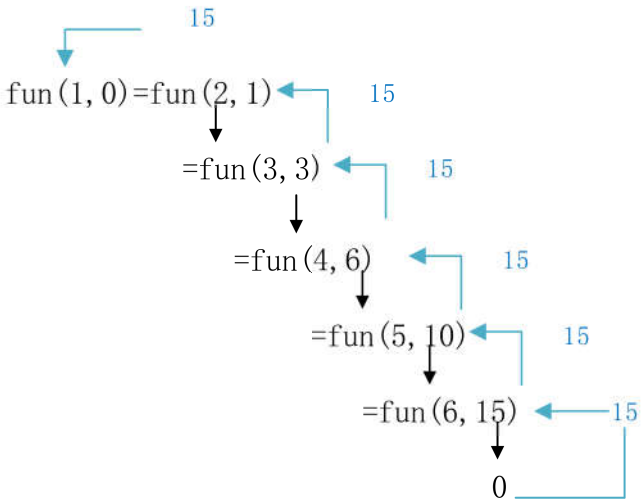
```
int main(){
    cin>>n;
    //从 1 开始递归，初始的和为 0
    cout<<fun(1,0);
}
```

重点：通过 fun(int)函数递归出 1~n 的每个数，每遇到一个数，就加到 s 上，然后层层递归深入，当递归结束，再层层返回求出的和 s 的值。

1、理接通过输入参数累加求和的过程；

2、如下图所示，每一层在递归下一层时，都会把这一层得到的 i 加到总和 s 上去，并将 s 的值带到下一层，直到递归结束，s 就是总和，此时，再层层返回 s，就能得到总和。

3、和上一个解法不同，上一个解法每一层得到的和并未带到下一层，而是再等待下一层的返回，在返回时计算总和；而本解法，是每一层都将和计算出来带入下一层继续计算，到递归结束，其实已经有和，只需要层层返回即可。



说明：本题我们采用了三种做法来解决本题，三种解法在后续的递归深入（深搜）类问题中都有应用，因此请大家在教学中让同学们都掌握一下。

不过，在讲解时，可以循序渐进，可以每隔 1、2 次课讲解 1 种做法，让同学们可以循序渐进得掌握熟练一种解法之后，再掌握其他的解法。

1241：【入门】角谷猜想

解法一：通过公共变量累计总次数

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

int c;//公共变量累计递归次数

void fun(int n){
    if(n != 1){
        if(n % 2 == 0){
            fun(n / 2);
        }else{
            fun(n * 3 + 1);
        }

        //每次递归，公共变量自增 1
        c++;
    }
}
```

```
int main(){
    int n;
    cin>>n;
    fun(n);
    cout<<c;
}
```

解法二：累计计算递归次数，然后层层返回

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

int fun(int n){
  if(n != 1){
    if(n % 2 == 0){
      return 1 + fun(n / 2);
    }else{
      return 1 + fun(n * 3 + 1);
    }
  }else{
    return 0;
  }
}

int main(){
  int n;
  cin>>n;
  cout<<fun(n);
}
```

解法三：通过输入参数逐层+1，最后返回 c 的值

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

//递归按照规则计算整数 n，直到 n 为 1
int fun(int n,int c){
    //如果 n!=1，则递归
    if(n != 1){
        if(n % 2 == 0){
            return fun(n / 2,1 + c);
        }else{
            return fun(n * 3 + 1,1 + c);
        }
    }else{
        return c;
    }
}

int main(){
    int n;
    cin>>n;
    cout<<fun(n,0);
}
```

1108：【入门】正整数 N 转换成一个二进制数

解法一：利用公共变量累加

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

string s;
//递归将 n 除 2
void fun(int n){
```

```
    char c;
    //递归条件：n!=0
    if(n != 0){
        c = n % 2 + '0';
        s = c + s;
        fun(n / 2);
    }
}

int main(){
    int n;
    cin>>n;
    fun(n);
    if(n == 0) cout<<0;
    else cout<<s;
}
```

解法二：通过层层累加计算结果，层层返回

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

//递归将 n 除 2
string fun(int n){
    char c;
    //递归条件：n!=0
    if(n != 0){
        c = n % 2 + '0';
        return fun(n / 2) + c;
    } else{
        return "";
    }
}

int main(){
    int n;
    cin>>n;
    if(n == 0) cout<<0;
    else cout<<fun(n);
}
```

解法三：利用输入参数逐层累加结果，在最后一层返回

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

//递归将 n 除 2
string fun(int n,string r){
    char c;
    //递归条件：n!=0
    if(n != 0){
        c = n % 2 + '0';
        return fun(n / 2,c + r);
    } else{
        return r;
    }
}

int main(){
    int n;
    cin>>n;
    if(n == 0) cout<<0;
    else cout<<fun(n,"");
}
```

1088：【入门】求两个自然数 M 和 N 的最大公约数

```
#include <bits/stdc++.h>
```

```
using namespace std;

//递归：将辗转相除法，重复用递归求解
long long fun(long long a,long long b){
    //递归停止条件：a % b == 0
    if(a % b != 0){
        return fun(b,a%b);
    } else{
        return b;
    }
}

int main(){
    long long a,b,t;
    cin>>a>>b;
    cout<<fun(a,b);
}
```

二、数值类递归作业

1083：【基础】回文数

1084：【入门】因子求和

1244：【入门】请问一个正整数能够整除几次 2？

1307：【基础】数的计数

1087：【入门】两个自然数 M 和 N 的最小公倍数