X 语言程序设计



本节内容

函数的声明语法

形参与实参

字符输入输出函数

函数的分类

系统函数（库函数）

自定义函数

函数声名格式

可以实现某个具体功能的代码块

增加代码的复用性

降低编程复杂度

函数不被调用是不会执行的

对内隐藏细节，对外暴露接口

```
<返回值类型|void>  函数名([参数列表])
{
    //函数体

    [return  返回值]
}
```

函数的四种形式

无参数，无返回值

无参数，有返回值

有参数，无返回值

有参数，有返回值

日常写代码，函数什么时候加参数，什么时候加返回值?

根据需求编写！！！

形参与实参

```
void add(int a, int b)
{
    printf("%d", a +b);
}
int main()
{
    int a = 5;
    int b = 10;
    add(a, b);
}
```

形参与实参

```
void add(int a, int b)
{
    printf("%d", a + b);
}
int main()
{
    int a = 5;
    int b = 10;
    add(a, b);
}
```

形式参数

实际参数

形参：函数声名中的参数

实参：函数调用进传入的参数

γετχηαρ() 与 πυτχηαρ()

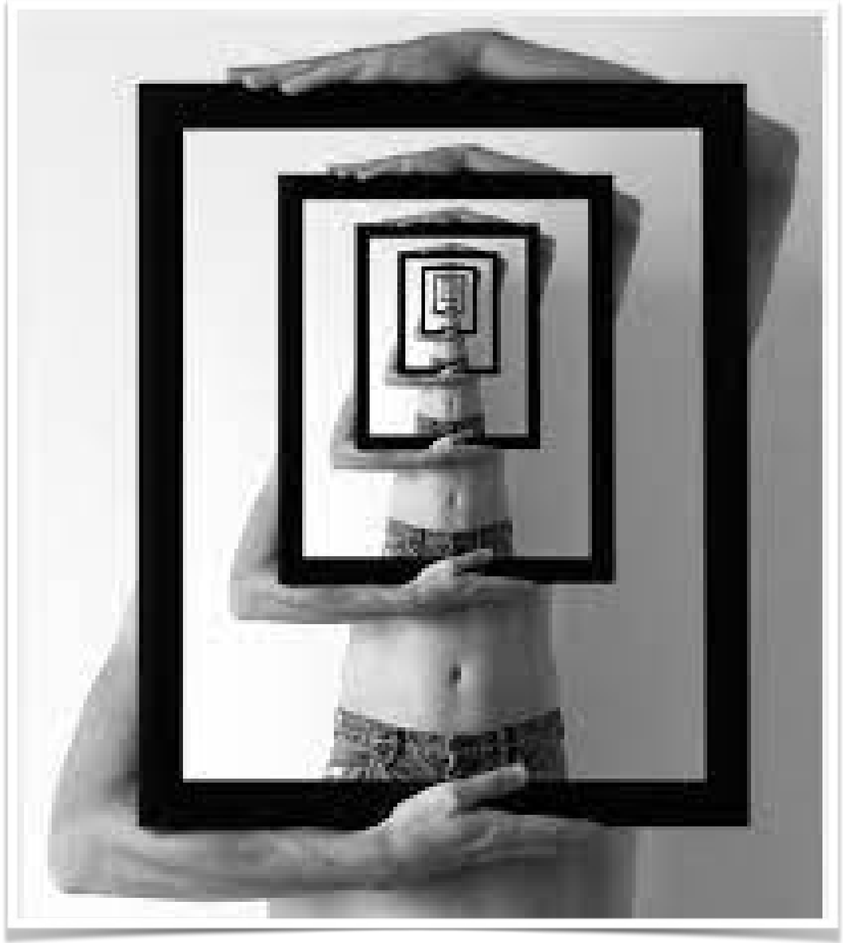
χηαρ γετχηαρ() ： 标准输入流中返回下一个字符。

ωοιδ πυτχηαρ(χηαρ χη) ： 输出参数 χη 。

递归

在函数调用过程中，调用自己本身





计算机 1～v 的和－非递归方式

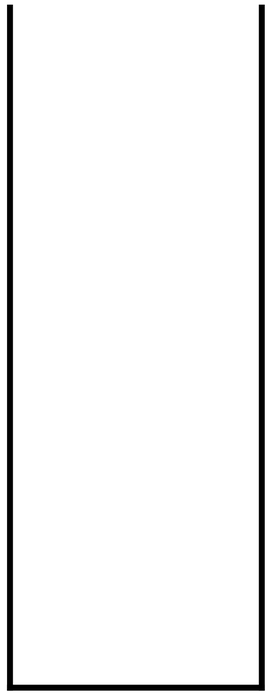
```
int fun(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}
```

计算机 1～v 的和－递归方式

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

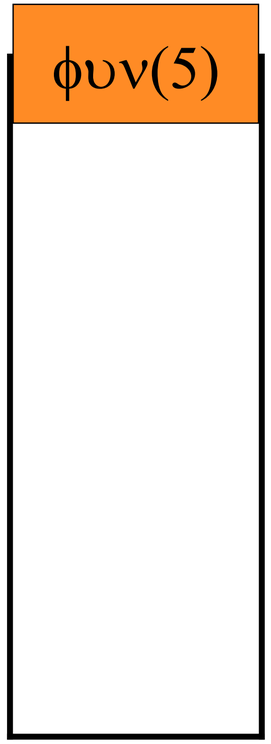
计算机 1～v 的和—递归方式

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



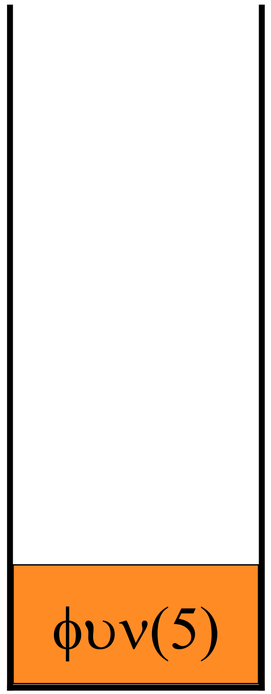
计算机 1～ν 的和－递归方式

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



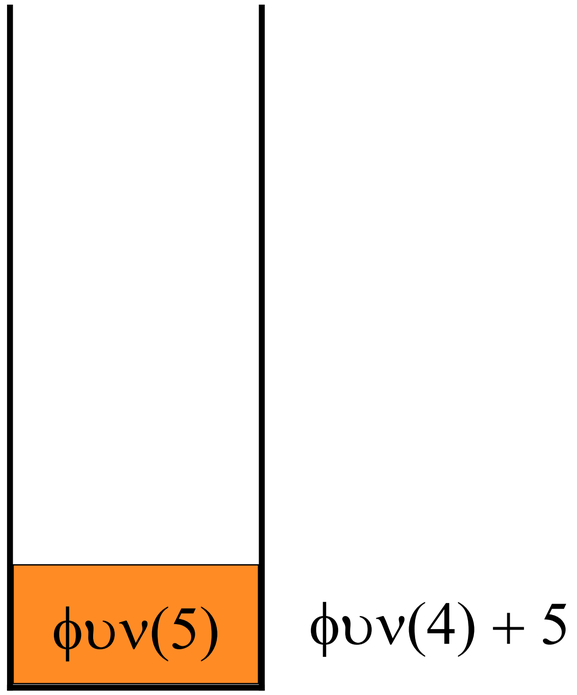
计算机 1～ν 的和－递归方式

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



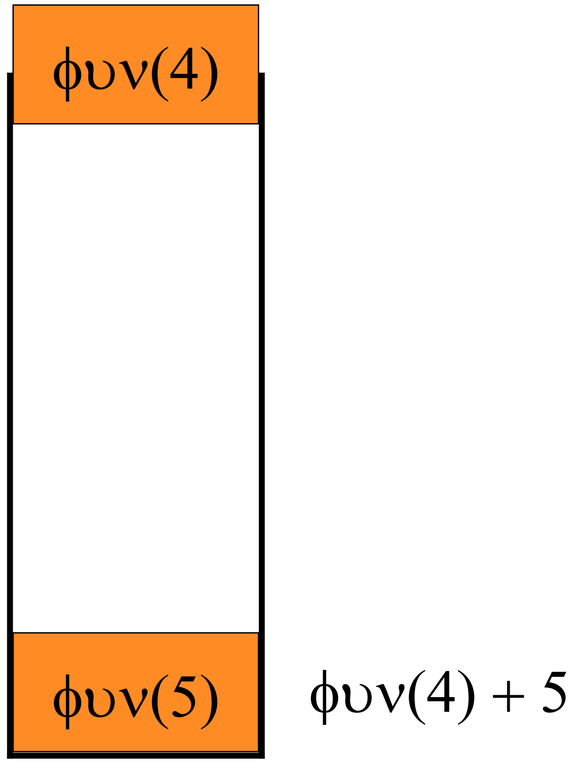
计算机 1～ν 的和—递归方式

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



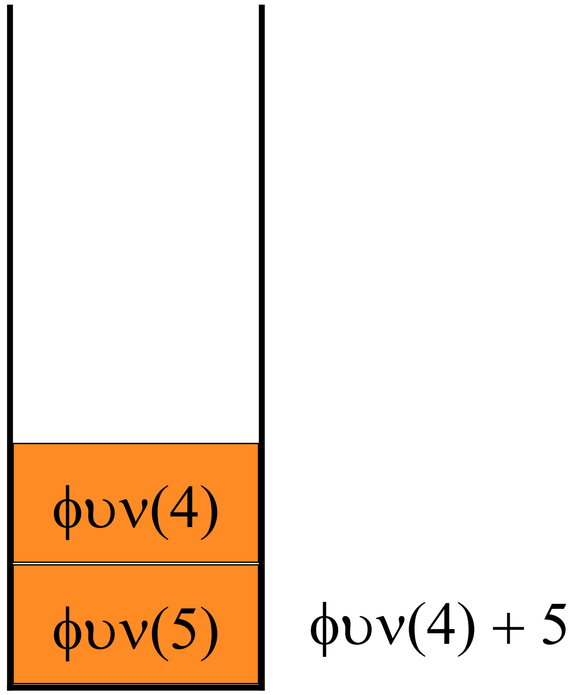
计算机 1～ν 的和—递归方式

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



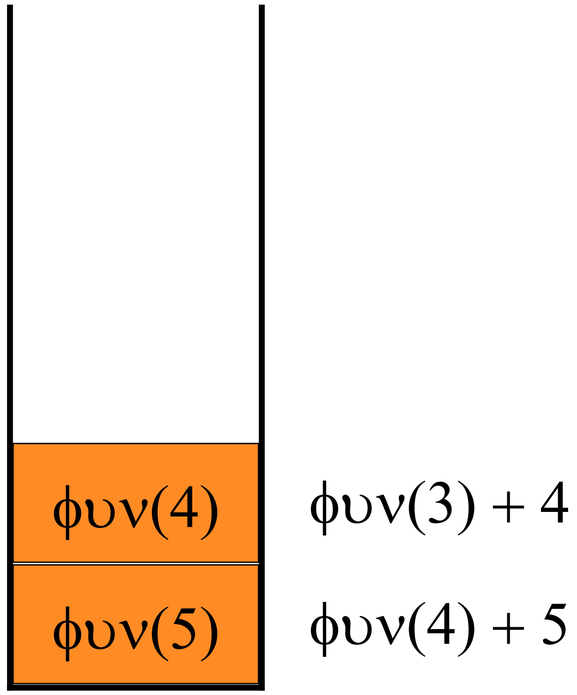
计算机 1～ν 的和—递归方式

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



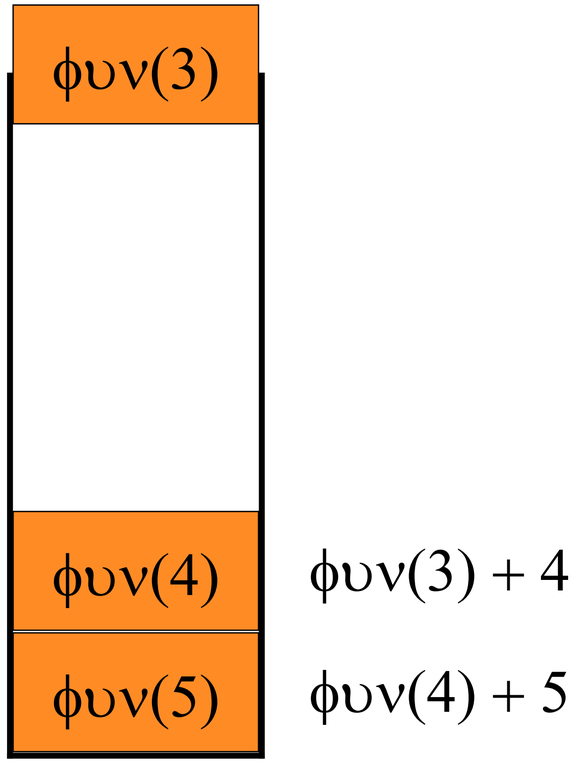
计算机 1～ν 的和—递归方式

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



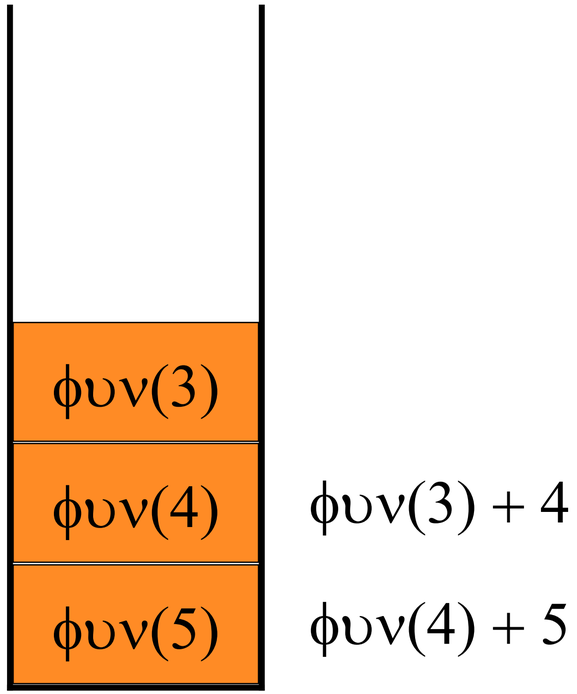
计算机 1～ν 的和—递归方式

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



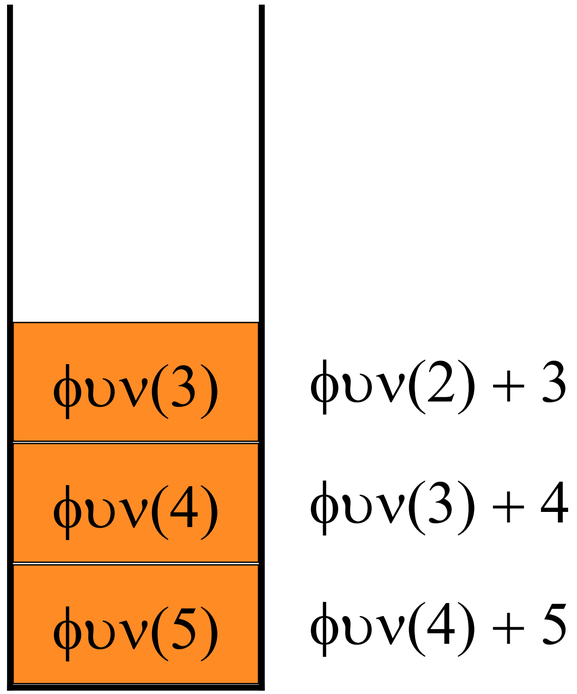
计算机 1～ν 的和—递归方式

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



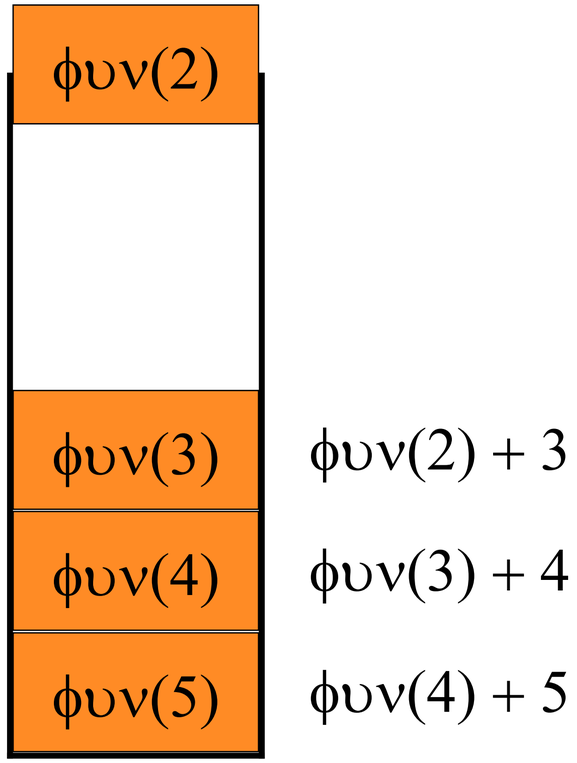
计算机 1～ν 的和—递归方式

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



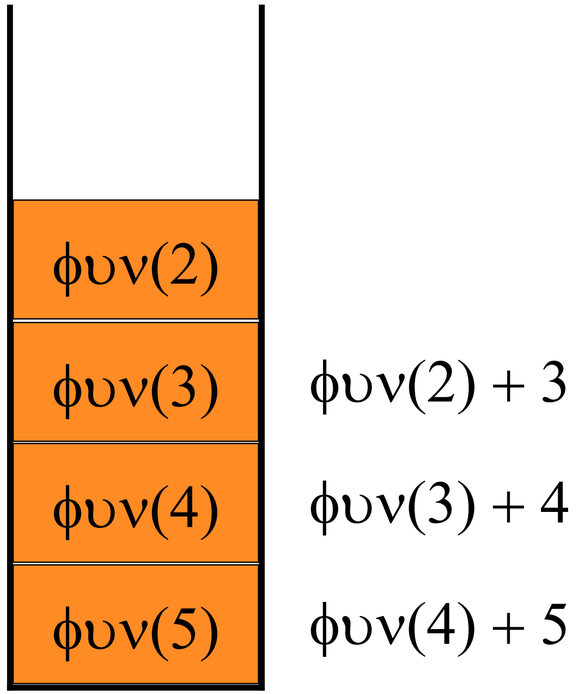
计算机 1～ν 的和—递归方式

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



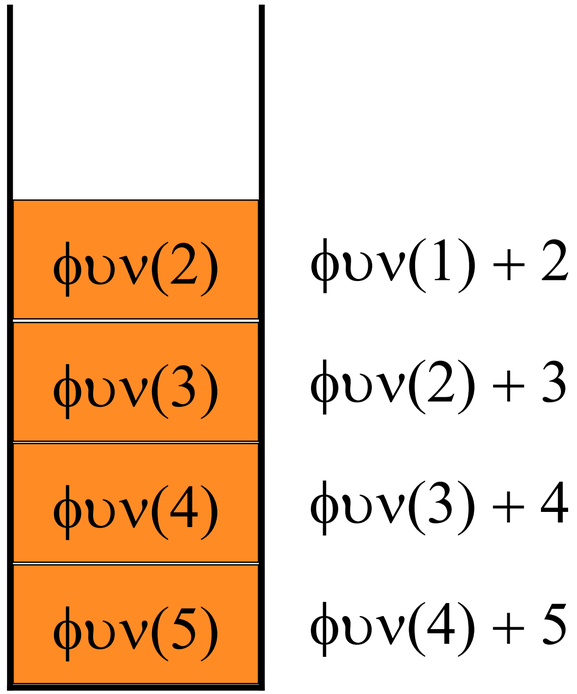
计算机 1～ν 的和—递归方式

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



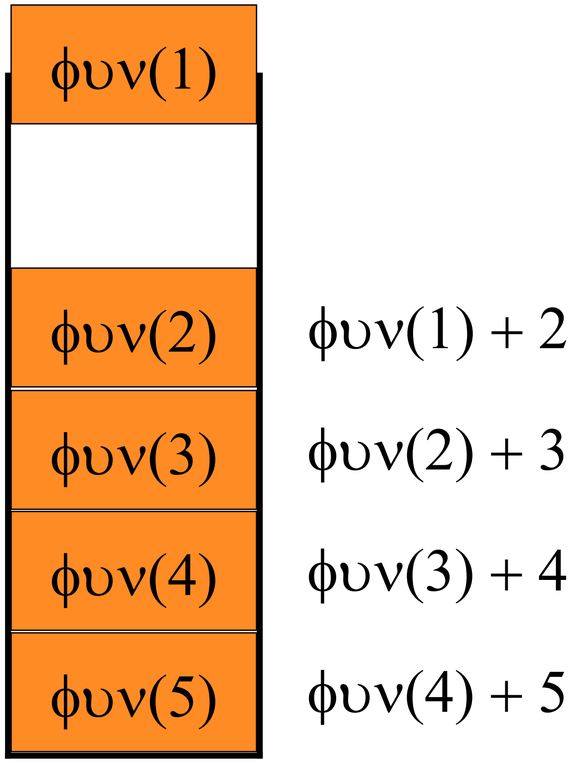
计算机 1～ν 的和—递归方式

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



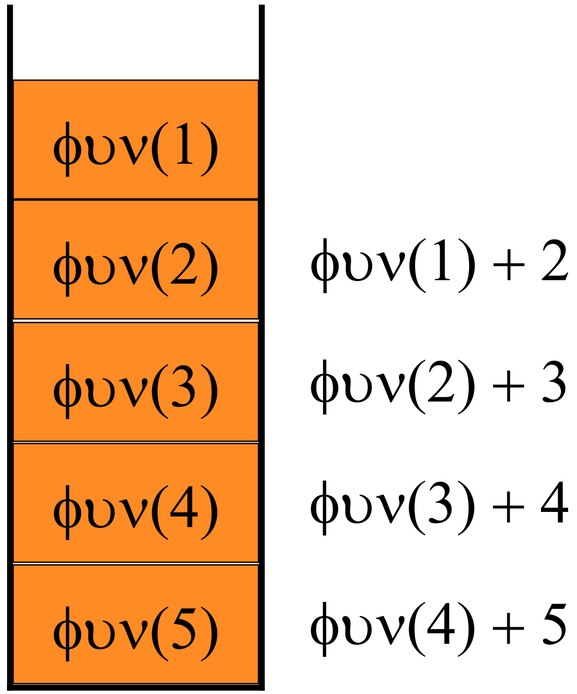
计算机 1～ν 的和—递归方式

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



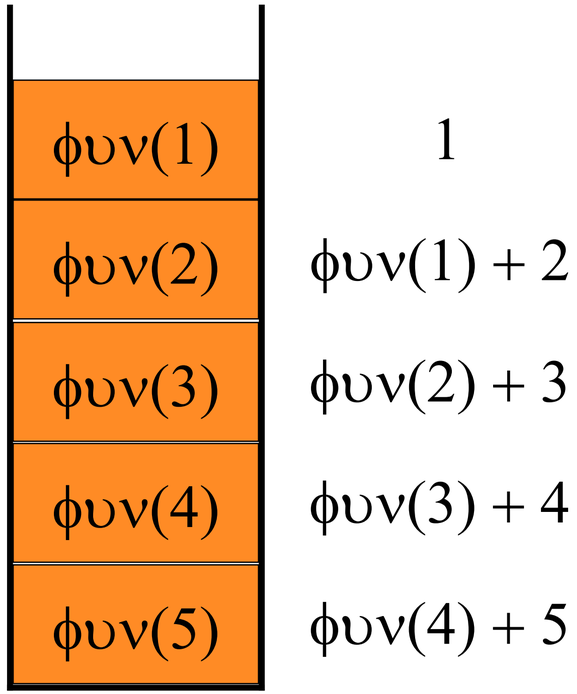
计算机 1～ν 的和—递归方式

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



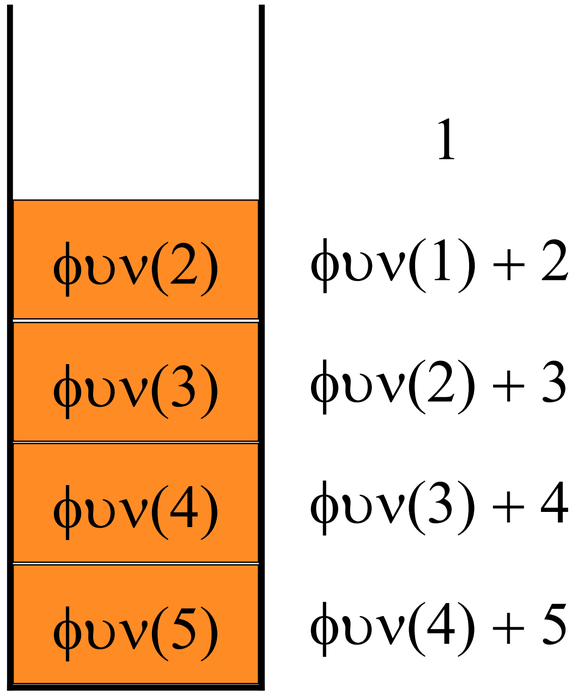
计算机 1～ν 的和—递归方式

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



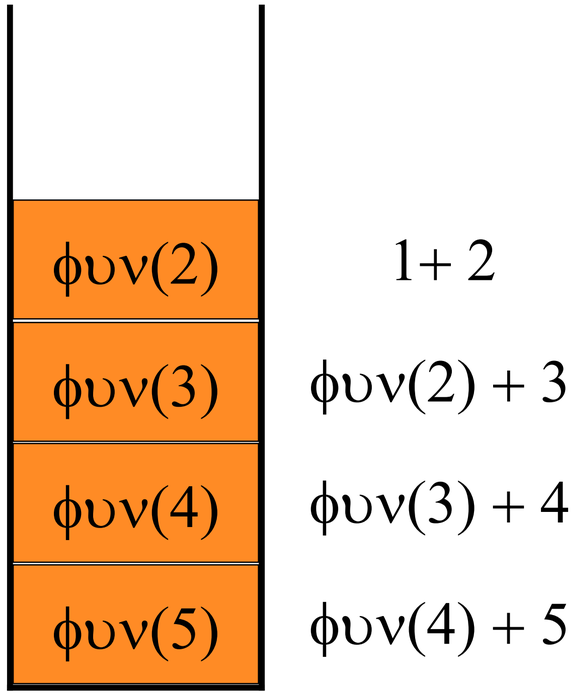
计算机 1～ν 的和—递归方式

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



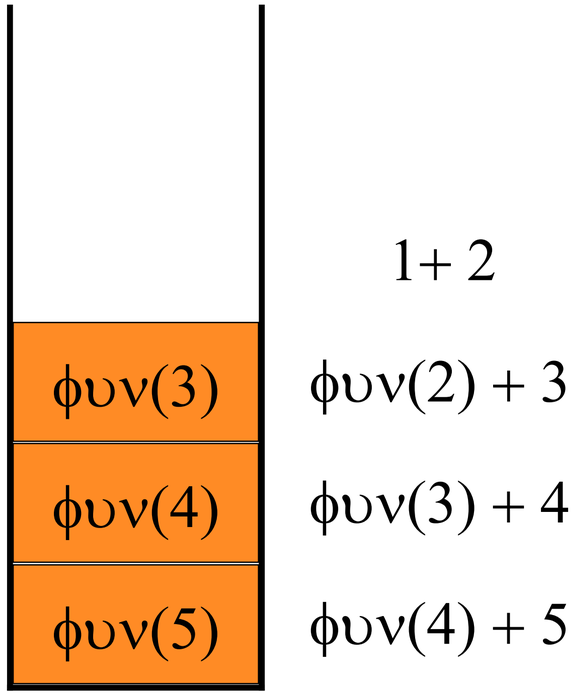
计算机 1～ν 的和—递归方式

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



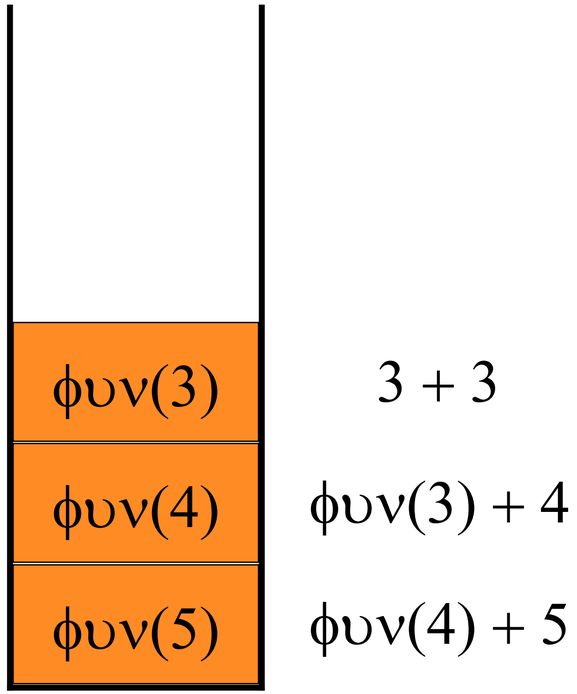
计算机 1～ν 的和—递归方式

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



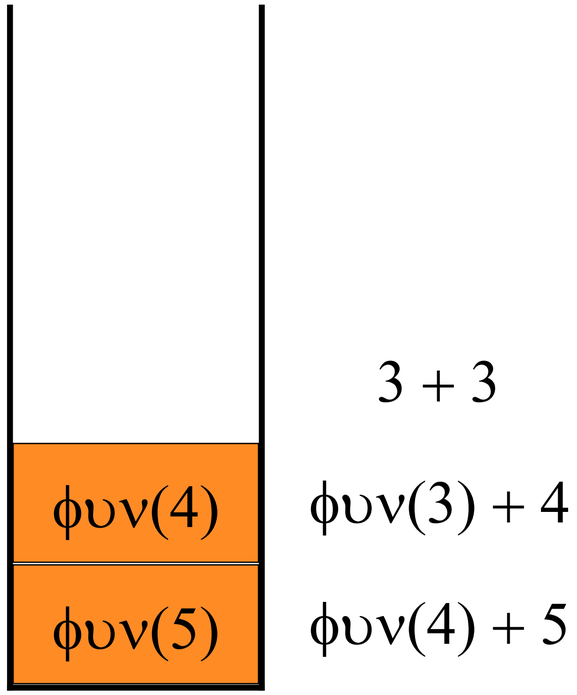
计算机 1～ν 的和—递归方式

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



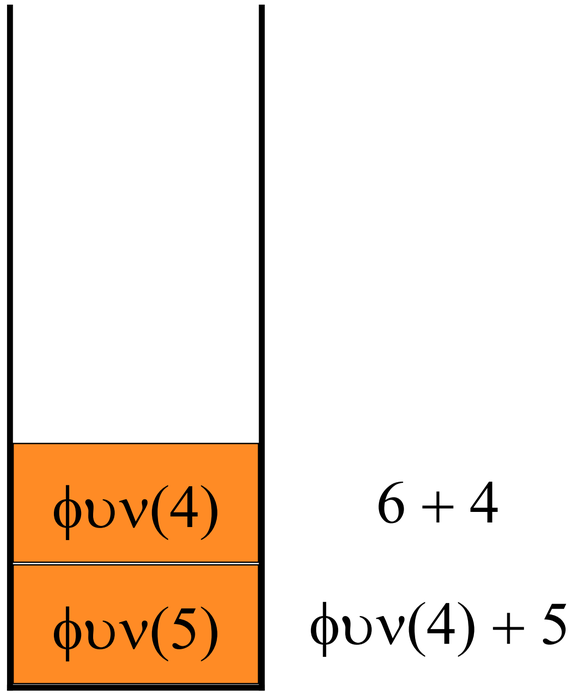
计算机 1～ν 的和—递归方式

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



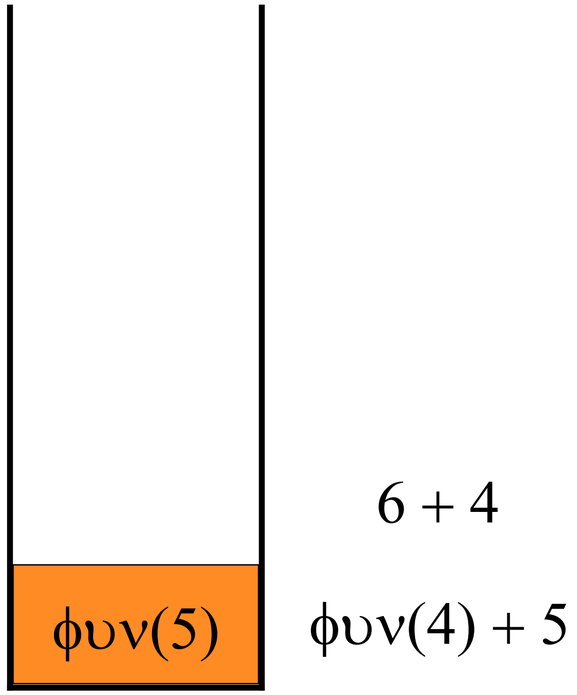
计算机 1～ν 的和—递归方式

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



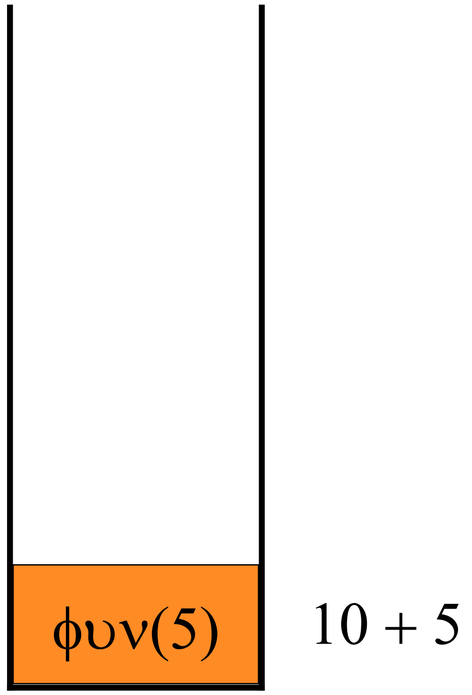
计算机 1～ν 的和—递归方式

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



计算机 1～ν 的和－递归方式

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



计算机 1～v 的和—递归方式

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



斐波那契数列

斐波那契数列（Φιβοναχχι σεθυενχε），又称黄金分割数列，因数学家莱昂纳多·斐波那契（Λεοναρδο Φιβοναχχι）以兔子繁殖为例子而引入，故又称“兔子数列”，其数值为：1、1、2、3、5、8、13、21、34.....

第 1、2 项为 1，从第 3 项开始每一项的值为前两项的和

求斐波那契数列第 n 项值－非递归方式

```
int fibonacci(int n)
{
    int last1 = 1;
    int last2 = 1;
    int result = 0;
    for (int i = 3; i <= n; i++)
    {
        result = last1 + last2;
        last2 = last1;
        last1 = result;
    }
    return result;
}
```

求斐波那契数列第 n 项值－非递归方式

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