函数相关概念

2024年4月26日 10:09

隐含了声明.

函数定义: void foo(int a) {...}

函数声明: void foo(int a);

函数调用: foo(3);

函数指针: foo, &foo

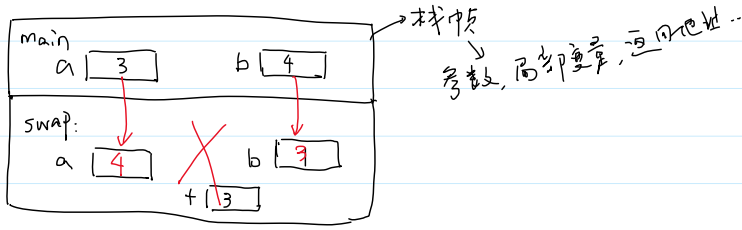
参数传递

2024年4月26日 10:56

```c
void swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
}

int main(void) {
    int a = 3, b = 4;
    printf("a = %d, b = %d\n", a, b);
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

形参 实参

```
Microsoft Visual Studio 调试控
a = 3, b = 4
a = 3, b = 4
```



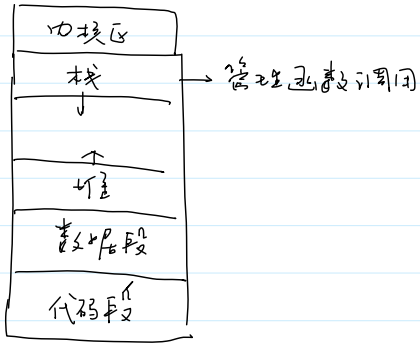
参数传递：值传递！（复制）

实参 → 形参

argument → parameter

问题：在被调函数（swap）里面是"不能"修改主调函数（main）里面值的。

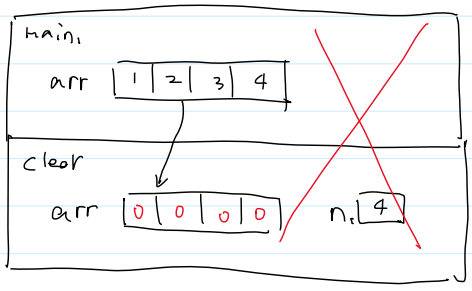
⇓ 指针



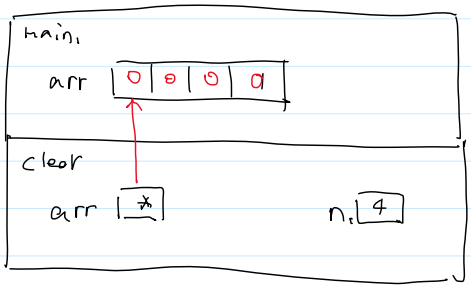
特例：数组作为参数传递时，会退化指向第一个元素的指针！

```c
void clear(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = 0;
    }
}

int main(void) {
    int arr[] = { 1, 2, 3, 4 };
    clear(arr, SIZE(arr));

    for (int i = 0; i < SIZE(arr); i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

why? 丢失了长度信息。



(√)



问题：为什么这样设计？

① 避免大量数据的复制

② 在被调函数（clear）中可以操作主调函数（main）中的数组。

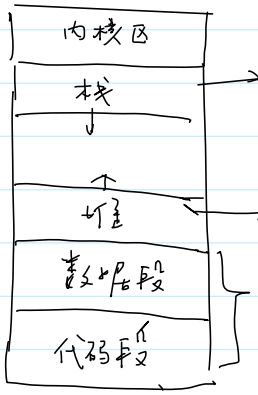
③ 让函数调用更灵活

局部变量和外部变量

2024年4月26日 11:34

作用域：编译时。
变量可以被引用的文本范围。

存储期限：运行时
变量绑定的值可以被引用的时间长度。



自动存储期限：栈中压入栈：开始
栈中弹出栈：消亡

动态存储期限：由程序员管理。 malloc：开始 free：消亡

静态存储期限：与天地同寿。
当进程启动时，开始
当进程退出时，消亡。

局部变量：定义在函数里面的变量。

作用域：块作用域。从定义开始，到块的末尾

存储期限：默认 → 自动存储期限。→ 栈
static → 静态存储期限。→ 数据段

(全局变量)

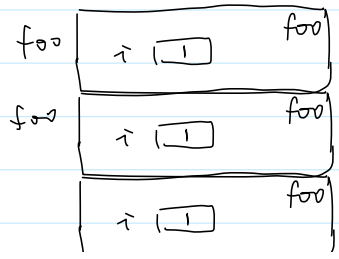
外部变量：定义在函数外面的变量

作用域：文件作用域。从定义开始，到文件末尾

存储期限：静态存储期限。

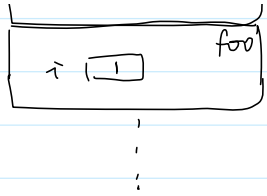
```c
void foo(void) {
    int i = 1;
    printf("&i = %p, i = %d\n", &i, i++);
    foo();
}
```

D:\工作\cpp58\1_C\CDay05\D



```
}
```

```
D:\工作\cpp58\1_C\CDay05\D
&i = 00E0710C, i = 1
&i = 00E07024, i = 1
&i = 00E06F3C, i = 1
&i = 00E06E54, i = 1
```

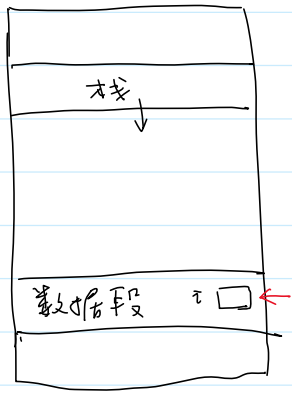


静态.

```
void foo(void) {
    static int i = 1;
    printf("&i = %p, i = %d\n", &i, i++);
    foo();
}
```

→ 只执行一次!

```
D:\工作\cpp58\1_C\CDay05\D
&i = 003DA008, i = 4722
&i = 003DA008, i = 4723
&i = 003DA008, i = 4724
&i = 003DA008, i = 4725
```



Q1. 静态存储的局部变量和外部变量有何区别?

作用域不同!

Q2: 静态存储期限的局部变量有什么作用? (使用场景)

↓

存储上一次函数调用的状态.

```
long long next_fib(void) {
    // 存储上一次函数调用的状态
    static long long a = 0;
    static long long b = 1;

    long long t = a + b;
    a = b;
    b = t;

    return a;
}
int main(void) {
    // foo();
    // Fibnacii: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
    printf("next_fib() = %lld\n", next_fib());  // 1
    printf("next_fib() = %lld\n", next_fib());  // 1
    printf("next_fib() = %lld\n", next_fib());  // 2
    printf("next_fib() = %lld\n", next_fib());  // 3
    printf("next_fib() = %lld\n", next_fib());  // 5
    return 0;
}
```

递归

2024年4月26日 14:33

recursion = re + cur + sion　　走重复的路.

re → 重复　　cur → 走，流通　　一个执行流程，走了一遍又一遍.

递归：一电影院例子.

递：将大问题分解时若干个子问题

子问题是跟求大问题最初等式一样，只是问题规模不一样.

归：将子问题的解合并时大问题的解

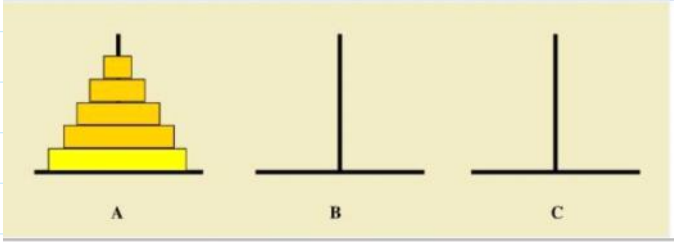
例子3：汉诺塔。 Hanoi

有三根杆子A，B，C。A杆上有 N 个 (N>1) 穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至 C 杆：

1. 每次只能移动一个圆盘；
2. 大盘不能叠在小盘上面。

提示：可将圆盘临时置于 B 杆，也可将从 A 杆移出的圆盘重新移回 A 杆，但都必须遵循上述两条规则。

问：最少需要移动多少次？如何移？



① N = 1,

A → C

③ N = 2:

A → B

A → C

B → C

⑦ N = 3:

A → C

A → B

C → B

A → C

B → A

B → C

A → C

前提：假定可以移动 n-1 个盘子，

如何移动 n 个盘子

递归公式：⇒ 这一层和下一层之间的关系.

A. 先将 n-1 盘移动到中间杆子上

B. 将最大的盘子移动到目标杆上

C. 将中间杆上 n-1 盘子移动目标杆上.

边界条件：

N = 1

起始杆子 → 目标杆子.

公理：数学归纳法 （皮埃诺公理）

```c
void hanoi(int n, char start, char middle, char target) {
    // 边界条件
    if (n == 1) {
        printf("%c -> %c\n", start, target);
        return;
    }
    // 递归公式
    // 将上面n-1个盘子从start,经过target, 移动到middle上
    hanoi(n - 1, start, target, middle);
    // 将最大的盘子从start直接移动到target上
    printf("%c -> %c\n", start, target);
    // 将上面n-1个盘子从middle,经过start, 移动到target上
    hanoi(n - 1, middle, start, target);
}
```

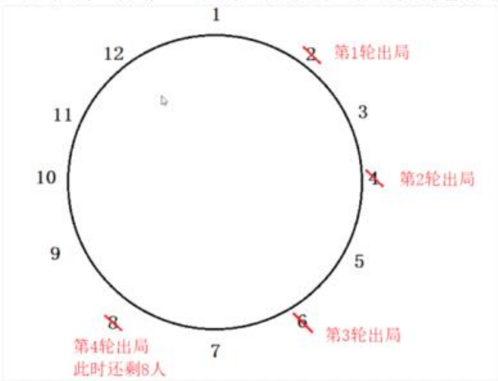
```
Total steps: 7
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

⇒ 用递归思维去思考问题！

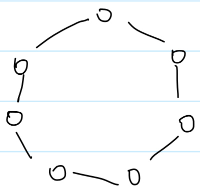
约瑟夫环

2024年4月26日 16:16

约瑟夫环是一个数学的应用问题：已知 n 个人（以编号1，2，3，...，n 分别表示）围坐在一张圆桌周围。从编号为 1 的人开始，每两个人出列一个人，直至只剩一个人。问：最终剩下的这个人的编号是多少？



思路1：循环链表，



时间：O(n) (n-1)×2

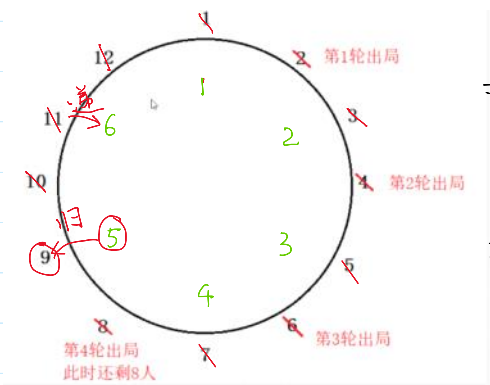
空间：O(n)

↓

每一个人出局的顺序.

思路2：递归

偶.



如何将子问题的解合并成大问题的解

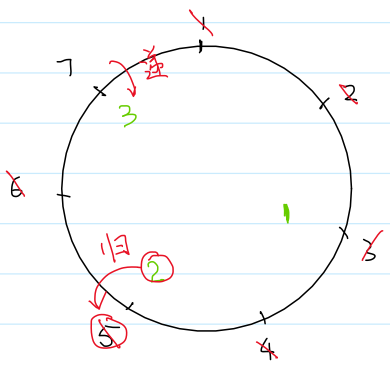
子问题的解 x： 1 2 3 4 5 6

大问题的解 f(x)： 1 3 5 7 9 11

$f(x) = 2x - 1$.

$joseph(n) = 2 \times joseph(\frac{n}{2}) - 1$ (1)

奇.



如何将子问题的解合并成大问题的解

子问题的解 x： 1 2 3

大问题的解 f(x)： 3 5 7

$f(x) = 2x + 1$

$joseph(n) = 2 * joseph(\frac{n-1}{2}) + 1$ (2)

边界条件:

joseph (1) = 1

joseph (2) = 1

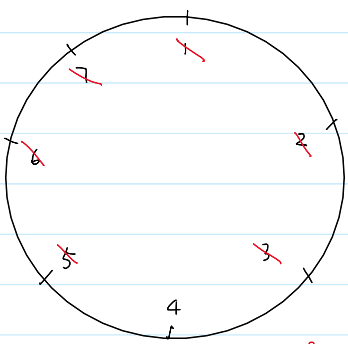
```
int joseph(int n) {
    // 边界条件
    if (n == 1 || n == 2) {
        return 1;
    }
    // 递归公式
    if (n & 0x1) {
        return 2 * joseph(n >> 1) + 1;
    }
    return 2 * joseph(n >> 1) - 1;
}
```

时间: O(log n)

空间: O(log n) → 栈

---

int joseph(int n, int m);



joseph(7, 3);

f(x):	1	2	3	4	5	6	7	→ 倒数第
x:	5	6		1	2	3	4	

$f(x) = (x+3) \% 7$

大问题的解.

↑ f(x):	0	1	2	3	4	5	6
x:	4	5		0	1	2	3

子问题的解

$f(x) = (x+3) \% 7$

⇓

递归公式. $joseph(n, m) = (joseph(n-1, m) + m) \% n$.

边界条件. $joseph(1, m) = 0$

递归三问

2024年4月26日 17:15

1. 找到问题的递归结构。

2. 要不要使用递归求解
   A. 存在重复计算
   B. 问题缩减幅度 ⟹ Stack Overflow
   (栈的深度)

栈空间受限的
主线程：8M
其它线程：2M

3. 大胆放心使用递归
   边界条件
   递归公式 → (这一层和下一层之间关系)

答疑：循环不变式

2024年4月26日 21:31

循环：{ 循环不变式 / 退出点

循环不变式：fib(i) 未求解

```
long long a = 0;
long long b = 1;
for (int i = 2; i <= n; i++) {
    long long t = a + b;
    a = b;
    b = t;
}  // i == n+1
```

控制表达式（指 i <= n）

→ 循环不变式维持到循环语句的结束