初识递归函数

huiw@suda.edu.cn

函数定义中使用了当前所定义函数的调用语句。这样的函数称为递归函数。

先看一个简单的代码

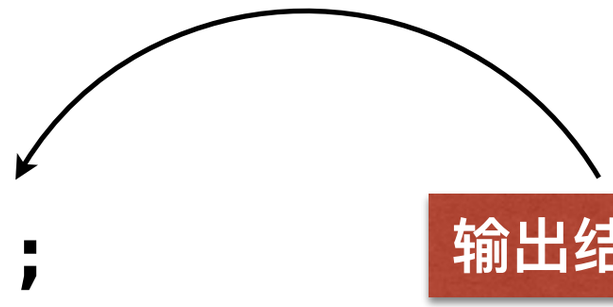
```
1 int forever() {
2    return forever();
3 }
4
5 int main (int argc, char* argv[]) {
6
7    forever();
8
9    return 0;
10 }
```

```c
#include <stdio.h>

int forever() {
   putchar('*');
   return forever();
}

int main (int argc, char* argv[]) {

   forever();

   return 0;
}

```

为递归函数的增加返回条件。

```c
#include <stdio.h>

void foreloop(unsigned int from, unsigned int to) {
  if (from > to).
    return;

  putchar('*');
  forloop(from+1, to);
}

int main (int argc, char* argv[]) {

  forloop(1, 5);

  return 0;
}
```

```c
#include <stdio.h>

void foreloop(unsigned int from, unsigned int to) {
   if (from > to).
      return;

   putchar('*');
   forloop(from+1, to-1);
}

int main (int argc, char* argv[]) {

   forloop(1, 5);

   return 0;
}
```



输出结果，有何变化?

经典递归代码举例

求最大公约数

```
1 unsigned int
2 gcd(unsigned int m, unsigned int n)
3 {
4     return (0==n)? m: gcd(n, m%n);
5 }
```

条件表达式

logical_or_expression '?' expression ':' expression

<逻辑表达式> ? <表达式1> : <表达式2>

注：条件表达式的优先级参见附件。

汉诺塔

```
void.
hanoi_solver(uint32_t n, char from_pole, char accessory_pole, char to_pole) {

  if (1==n) {
    move(from_pole, to_pole);
    return;
  }

  hanoi_solver(n-1, from_pole, to_pole, accessory_pole);
  move(from_pole, to_pole);
  hanoi_solver(n-1, accessory_pole, from_pole, to_pole);

}
```

```c
void
move(char from_pole, char to_pole) {
    printf("%c -> %c\n", from_pole, to_pole);
}
```

抽象数据类型

函数 与 复合数据类型

func()

unsigned double
char signed
float long short

存储操作

int
= unsigned int

& ~ -(单目)

运算符与表达式

+ - * / % ! || && == != < > <= >=

程序流程控制

goto if…else…

cos
fabs

putchar

库函数

WENZHENG COLLEGE OF SOOCHOW UNIVERSITY

2017.3.29

Soochow University

附录