函数

2020年7月14日 13:46

素数求和

```c
scanf("%d %d", &m, &n);
// m=10,n=31;
if ( m==1 ) m=2;
for ( i=m; i<=n; i++ ) {
    int isPrime = 1;
    int k;
    for ( k=2; k<i-1; k++ ) {
        if ( i%k == 0 ) {
            isPrime = 0;
            break;
        }
    }
    if ( isPrime ) {
        sum += i;
        cnt++;
    }
}
printf("%d %d\n", cnt, sum);
```

```c
int isPrime(int i)
{
    int ret = 1;
    int k;
    for ( k=2; k<i-1; k++ ) {
        if ( i%k == 0 ) {
            ret = 0;
            break;
        }
    }
    return ret;
}
```

```c
scanf("%d %d", &m, &n);
// m=10,n=31;
if ( m==1 ) m=2;
for ( i=m; i<=n; i++ ) {
    if ( isPrime(i) ) {
        sum += i;
        cnt++;
    }
}
printf("%d %d\n", cnt, sum);
```

什么是函数?

- 函数是一块代码，接收零个或多个参数，做一件事情，并返回零个或一个值
- 可以先想像成数学中的函数：
  - y = f(x)

调用函数时，进入函数内部、单步调试

求和函数

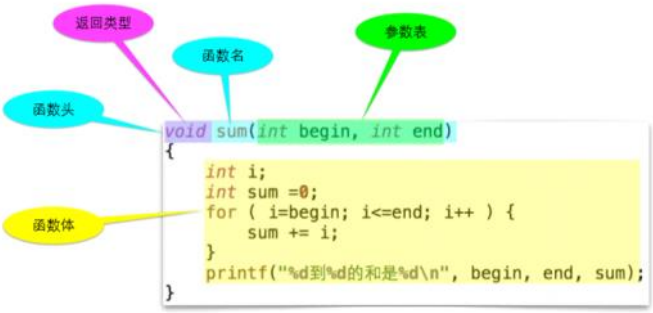
```c
int i;
int sum;

for ( i=1,sum=0; i<=10; i++ ) {
    sum += i;
}
printf("%d到%d的和是%d\n", 1, 10, s

for ( i=20,sum=0; i<=30; i++ ) {
    sum += i;
}
printf("%d到%d的和是%d\n", 20, 30,

for ( i=35,sum=0; i<=45; i++ ) {
    sum += i;
}
printf("%d到%d的和是%d\n", 35, 45,
```

```c
void sum(int begin, int end)
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}

int main()
{
    sum(1,10);
    sum(20,30);
    sum(35,45);

    return 0;
}
```

```c
int max(int a, int b)
{
    // int ret;
    if ( a>b ) {
        return a;
    } else {
        return b;
    }

    // return ret;
}
```

写函数最好遵从单一出口原则

函数定义



调用函数

- 函数名(参数值);
- ()起到了表示函数调用的重要作用
  - 即使没有参数也需要()
- 如果有参数，则需要给出正确的数量和顺序
- 这些值会被按照顺序依次用来初始化函数中的参数

```c
sum(1,10);
sum(20,30);
sum(35,45);
```

```c
void sum(int begin, int end)
{
int i;
int sum =0;
for ( i=begin; i<=end; i++ ) {
sum += i;
}
printf("%d到%d的和是%d\n", begin, end, sum);
}
```

从函数中返回值

```c
int max(int a, int b)
{
    int ret;
    if ( a>b ) {
        ret = a;
    } else {
        ret = b;
    }

    return ret;
}
```

- return停止函数的执行，并送回一个值
  - return;
  - return 表达式;
- 一个函数里可以出现多个return语句

```c
int max(int a, int b)
{
    int ret;
    if ( a>b ) {
        ret = a;
    } else {
        ret = b;
    }

    return ret;
}
```

```c
int a,b,c;
a = 5;
b = 6;
c = max(10,12);
c = max(a,b);
c = max(c, 23);
c = max(max(c,a), 5);
printf("%d\n", max(a,b));
max(12,13);
```

- 可以赋值给变量
- 可以再传递给函数
- 甚至可以丢弃
  - 有的时候要的是副作用

函数返回

- 函数知道每一次是哪里调用它，会返回到正确的地方

```
void sum(int begin, int end)
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```

```
sum(1,10);
sum(20,30);
sum(35,45);

return 0;
```

没有返回值的函数

- void 函数名(参数表)
- 不能使用带值的return
  - 可以没有return
- 调用的时候不能做带值的赋值

如果函数有返回值，则必须使用带值的return

```
void sum(int begin, int end)
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```

函数先后关系

```
void sum(int begin, int end)
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}

int main()
{
    sum(1,10);
    sum(20,30);
    sum(35,45);

    return 0;
}
```

- 像这样把sum()写在上面，是因为：
- C的编译器自上而下顺序分析你的代码
- 在看到sum(1,10)的时候，它需要知道sum()的样子
- 也就是sum()要几个参数，每个参数的类型如何，返回什么类型
- 这样它才能检查你对sum()的调用是否正确

```
1  #include <stdio.h>
2
3
4  int main()
5  {
6      sum(1,10);  //  int sum(int,int)
7      sum(20,30);
8      sum(35,45);
9
10     return 0;
11 }
12
13 void sum(int begin, int end)
14 {
15     int i;
16     int sum =0;
17     for ( i=begin; i<=end; i++ ) {
18         sum += i;
19     }
20     printf("%d到%d的和是%d\n", begin, end, sum);
21 }
22
```

```
/Users/wengkai/cc/7.2-1.c:6:2: warning: implicit dec
        sum(1,10);
        ^
/Users/wengkai/cc/7.2-1.c:13:6: error: conflicting t
void sum(int begin, int end)
     ^
```

如果不知道

- 也就是把要调用的函数放到下面了
- 旧标准会假设你所调用的函数所有的参数都是int，返回也是int
- 如果恰好不对...

```
int a,b,c;
a = 5;
b = 6;
c = max(10,12);
printf("%d\n", c);
max(12,13);

return 0;
}

double max(double a, double b)
{
```

自动类型转换

```
/Users/wengkai/cc/7.4.c:15:8: error: conflicting types for 'max'
double max(double a, double b)
       ^
```

```
1  #include <stdio.h>
2
3  void sum(int begin, int end);   //  声明
4
5  int main()
6  {
7      sum(1,10);  //  int sum(int,int)
8      sum(20,30);
9      sum(35,45);
10
11     return 0;
12 }
13
14 void sum(int begin, int end)    //  定义
15 {
16     int i;
17     int sum =0;
18     for ( i=begin; i<=end; i++ ) {
19         sum += i;
20     }
21     printf("%d到%d的和是%d\n", begin, end, sum);
22 }
23
```

猜的！和编译器有关

```
Finished in 0.2s]
```

函数原型

- 函数头，以分号";"结尾，就构成了函数的原型
- 函数原型的目的是告诉编译器这个函数长什么样
  - 名称
  - 参数（数量及类型）
  - 返回类型
- 旧标准习惯把函数原型写在调用它的函数里面
- 现在一般写在调用它的函数前面
- 原型里可以不写参数的名字，但是一般仍然写上

函数原型

根据原型判断

实际的函数头

```
double max(double a, double b);

int main()
{
    int a,b,c;
    a = 5;
    b = 6;
    c = max(10,12);
    printf("%d\n", c);
    max(12,13);

    return 0;
}

double max(double a, double b)
```

```
#include <stdio.h>

int sum(int begin, int end);    //  声明

int main()
{
    sum(1,10);  //  int sum(int,int)
    sum(20,30);
    sum(35,45);

    return 0;
}

void sum(int begin, int end)    //  定义
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```

```
Users/wengkai/cc/7.2-1.c:14:6: error: conflicting types for 'sum'
void sum(int begin, int end)    //      定义
     ^
Users/wengkai/cc/7.2-1.c:3:5: note: previous declaration is here
int sum(int begin, int end);    //      声明
    ^
```

```
#include <stdio.h>

void sum(int , int );   //  声明

int main()
{
    sum(1,10);  //  int sum(int,int)
    sum(20,30);
    sum(35,45);

    return 0;
}

void sum(int begin, int end)    //  定义
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}
```

```
[Finished in 1.3s]
```

返回结果与编译器有关

声明不是函数，只是告诉编译器sum是个函数，有几个参数，返回类型是啥，编译器不会猜了。
根据声明判断下面函数是否正确，下面的函数定义时也要检查定义和声明是否一致。如果不一致，前面检查白费了。

一般留着，增加程序可读性

调用函数

- 如果函数有参数，调用函数时必须传递给它数量、类型正确的值
- 可以传递给函数的值是表达式的结果，这包括：
  - 字面量
  - 变量
  - 函数的返回值
  - 计算的结果

```
int a,b,c;
a = 5;
b = 6;
c = max(10,12);
c = max(a,b);
c = max(c, 23);
c = max(max(23,45), a);
c = max(23+45, b);
```

只是个warning，整数，浮点数有损失
不同编译器结果不一样

```
#include <stdio.h>

void cheer(int i)
{
    printf("cheer %d\n", i);
}

int main()
{
    cheer(2.4);

    return 0;
}
```

```
#include <stdio.h>

void cheer(int i)
{
    printf("cheer %d\n", i);
}

int main()
{
    cheer(2.0);

    return 0;
}
```

```
Users/wengkai/cc/test.c:10:8: war
        cheer(2.4);
        ~~~~~ ^~~
 warning generated.
heer 2
Finished in 0.1s]
```

```
[Finished in 0.1s]
```

类型不匹配?

- 调用函数时给的值与参数的类型不匹配是C语言传统上最大的漏洞
- 编译器总是悄悄替你把类型转换好，但是这很可能不是你所期望的
- 后续的语言，C++/Java在这方面很严格

```
#include <stdio.h>

void cheer(int i)
{
    printf("cheer %d\n", i);
}

int main()
{
    double f = 2.0;
    cheer(f);

    return 0;
}
```

```
#include <stdio.h>

void cheer(int i)
{
    printf("cheer %d\n", i);
}

int main()
{
    double f = 2.4;
    cheer(f);

    return 0;
}
```

```
heer 2
Finished in 0.1s]
```

```
cheer 2
Finished in 0.1s]
```

传过去的是什么？

```
void swap(int a, int b);

int main()
{
    int a = 5;
    int b = 6;

    swap(a, b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

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

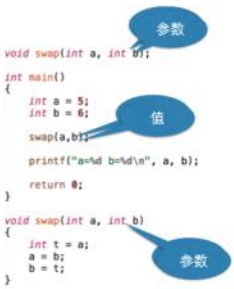
C语言在调用函数时，永远只能传值给函数

- 这样的代码能交换a和b的值吗？

这两个a,b没有关系！ 同名但是处在不同的地方
它们之间的联系仅仅是在调用的时候把值传过去了
在swap里面a,b做的事情，是swap里面a,b的事情，与外面的a,b无关！

传值

- 每个函数有自己的变量空间，参数也位于这个独立的空间中，和其他函数没有关系
- 过去，对于函数参数表中的参数，叫做“形式参数”，调用函数时给的值，叫做“实际参数”
- 由于容易让初学者误会实际参数就是实际在函数中进行计算的参数，误会调用函数的时候把变量而不是值传进去了，所以我们不建议继续用这种古老的方式来称呼它们
- 我们认为，它们是参数和值的关系



本地变量

- 函数的每次运行，就产生了一个独立的变量空间，在这个空间中的变量，是函数的这次运行所独有的，称作本地变量
- 定义在函数内部的变量就是本地变量
- 参数也是本地变量

变量的生存期和作用域

- 生存期：什么时候这个变量开始出现了，到什么时候它消亡了
- 作用域：在（代码的）什么范围内可以访问这个变量（这个变量可以起作用）
- 对于本地变量，这两个问题的答案是统一的：大括号内——块

```
    swap(a,b);

    printf("a=%d,b=%d\n", a,

    return 0;
}

void swap(int x, int y)
{
    int swap;
    int t = x;
    x = y;
    y = t;
}
```

运行到swap，a,b有，但是在当前上下文不能访问，a,b的生存期还有，但是不在作用域内。
运行完函数，t,x,y不存在了。t,x,y只在函数里面。

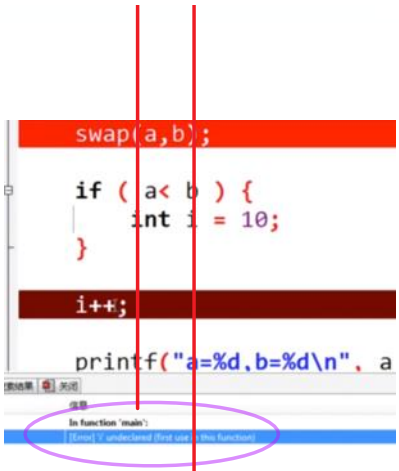
本地变量的规则

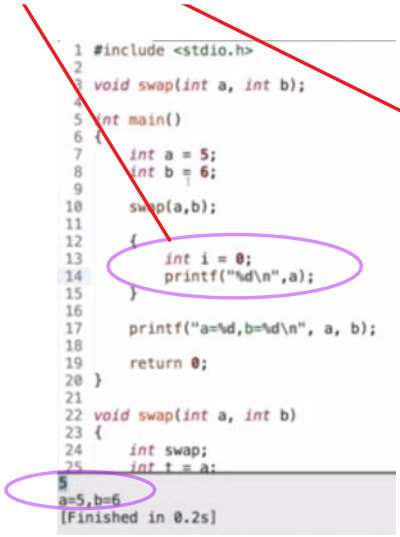
- 本地变量是定义在块内的
  - 它可以是定义在函数的块内
  - 也可以定义在语句的块内
  - 甚至可以随便拉一对大括号来定义变量
- 程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就消失了
- 块外面定义的变量在里面仍然有效
- 块里面定义了和外面同名的变量则掩盖了外面的
- 不能在一个块内定义同名的变量
- 本地变量不会被默认初始化
- 参数在进入函数的时候被初始化了

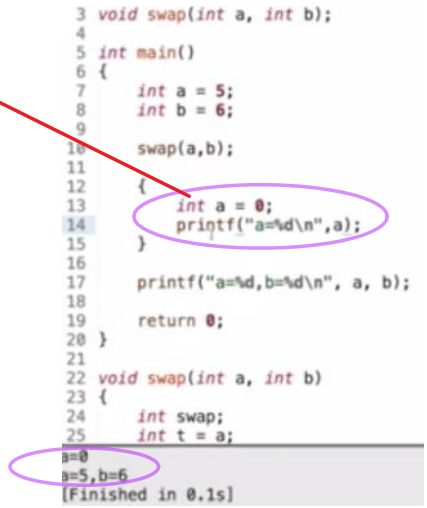
```
int main(void)
{
    int a = 5;
    int b = 6;

    swap(a,b);
    printf("a=%d,b=%d\n", a, b);

    return 0;
}
```

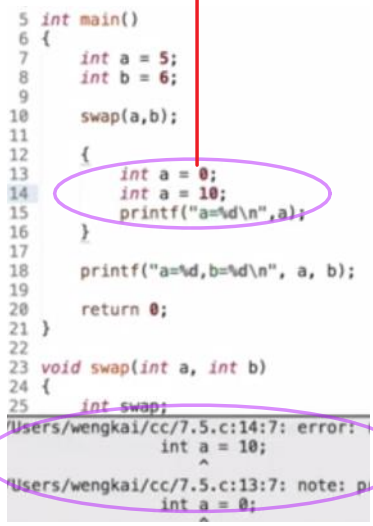
```
void swap(int a, int b)
{
    int t = a;
    a = b;
    b = t;
}
```

两个变量空间，每个空间有自己的参数，所以swap里面做的任何事情对main无关。
当做完swap要回去的时候，swap的空间就会没有了



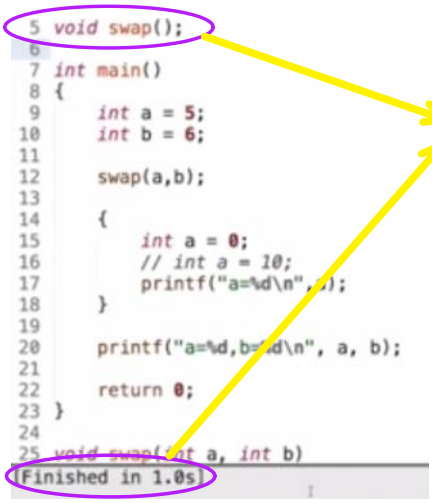






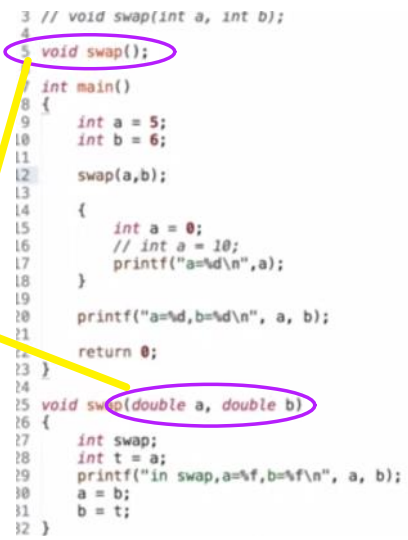
没有参数时

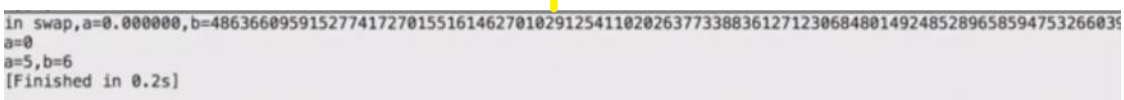
- void f(void);
- 还是
- void f();
  - 在传统C中，它表示f函数的参数表未知，并不表示没有参数



我只知道有个swap，我不确定里面有什么参数，编译器猜里面是两个int

欺骗了编译器，一开始以为swap要两个int原型也检查原型定义对不对，原型说了不确定是什么类型，所以两个double通过编译。但是发生了错误。所以不要写出这样的函数，确定函数没有参数，就写void





逗号运算符?

- 调用函数时的逗号和逗号运算符怎么区分?
- 调用函数时的圆括号里的逗号是标点符号，不是运算符
  - f(a,b)
  - f((a,b))

这是运算符了

函数里的函数?

可以放声明，不能放定义

- C语言不允许函数嵌套定义
- int i,j,sum(int a, int b);
- return (i);

定义了两个变量，声明了一个函数
不建议这么写!

圆括号没有意义，变量外面加圆括号还是个表达式
但是容易误解成是一个函数

关于main

- int main()也是一个函数
- 要不要写成int main(void)?
- return的0有人看吗?
  - Windows：if errorlevel 1 …
  - Unix Bash：echo $?
  - Csh：echo $status

可以这么写
有人看