第6章 函数

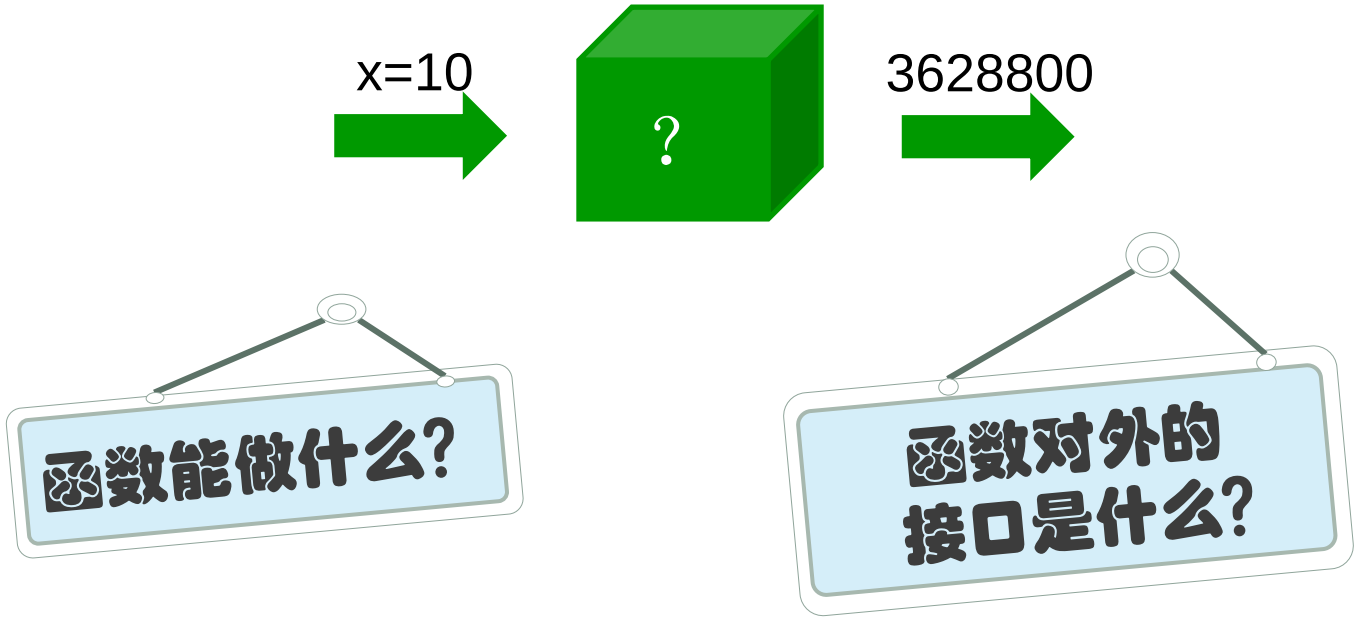
——函数封装与程序的健壮性

本节要讨论的主要问题

- 何为函数封装？
- 如何增强程序的健壮性？

函数封装（Encapsulation）

- 外界对函数的影响——仅限于入口参数
- 函数对外界的影响——仅限于一个返回值和数组、指针形参



函数封装

- 外界对函数的影响——仅限于入口参数
- 函数对外界的影响——仅限于一个返回值和数组、指针形参

```c
#include <stdio.h>
long  Fact(int  n);
int main()
{
    int  m;
    long ret;
    printf("Input m:");
    scanf("%d", &m);
    ret = Fact(m);
    printf("%d! = %ld\n", m, ret);
    return 0;
}
```

```c
long  Fact(int  n)
{
    int  i;
    long result = 1;
    for (i=2; i<=n; i++)
    {
        result *= i;
    }
    return result;
}
```

传入负数实参会怎样？

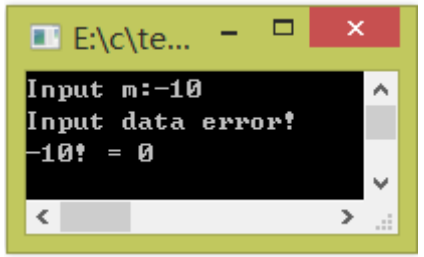
如何增强程序的健壮性?

- 在函数的入口处，检查输入参数的合法性

```
/* 函数功能：用迭代法计算 n! */
long Fact(int n)
{
    int i;
    long result = 1;
    if (n < 0)
    {
        printf("Input data error!\n");
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```

```
long Fact(int n)
{
    int i;
    long result = 1;
    for (i=2; i<=n; i++)
    {
        result *= i;
    }
    return result;
}
```

计算整数n的阶乘n！



CB

```
E:\c\test\buff...          In function 'Fact':
E:\c\test\buff... 31       warning: control reaches end of non-void function
                           === Build finished: 0 errors, 1 warnings ===
```

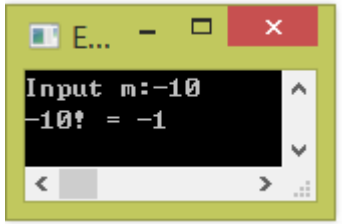
VC

```
warning C4715: 'Fact' : not all control paths return a value
```

如何增强程序的健壮性?

- 在函数的入口处，检查输入参数的合法性

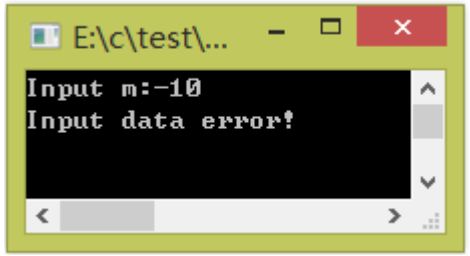
```
/* 函数功能：用迭代法计算 n！ */
long  Fact(int  n)
{
    int  i;
    long result = 1;
    if (n < 0)
    {
        printf("Input data error!\n");
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```




```
long  Fact(int  n)
{
    int  i;
    long result = 1;
    if (n < 0)
    {
        return -1;
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```

如何增强程序的健壮性?

- 在函数的入口处，检查输入参数的合法性
- 检查函数的返回值



防御性编程

```c
#include <stdio.h>
long  Fact(int  n);
int main()
{
    int  m;
    long ret;
    printf("Input m:");
    scanf("%d", &m);
    ret = Fact(m);
    if (ret == -1)          /* 增加对函数返回值的检验 */
        printf("Input data error!\n");
    else
        printf("%d! = %ld\n", m, ret);
    return 0;
}
```

```c
long  Fact(int  n)
{
    int  i;
    long result = 1;
    if (n < 0)
    {
        return -1;
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```

如何增强程序的健壮性?

- 改成无符号整型，传入负数实参Fact()会返回-1吗？

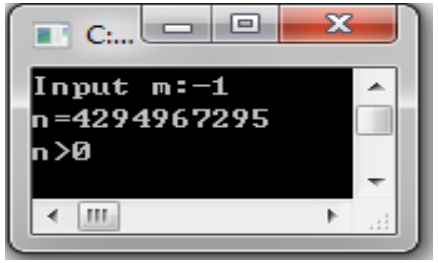
```c
#include <stdio.h>
unsigned long  Fact(unsigned int  n);
int main()
{
    int  m;
    long ret;
    printf("Input m:");
    scanf("%d", &m);
    ret = Fact(m);
    if (ret == -1)          /* 增加对函数返回值的检验 */
        printf("Input data error!\n");
    else
        printf("%d! = %ld\n", m, ret);
    return 0;
}
```

```c
unsigned long  Fact(unsigned int  n)
{
    unsigned int  i;
    unsigned long result = 1;
    if (n < 0)
    {
        return -1;
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```

如何增强程序的健壮性?

- 改成无符号整型，传入负数实参Fact()会返回-1吗？
  - 存在死代码的原因何在？

```
unsigned long  Fact(unsigned int  n)
{
    unsigned int  i;
    long result = 1;
    printf("n=%u\n", n);
    if (n < 0)
    {
        printf("n<0");
        return -1;
    }
    else
    {
        printf("n>0");
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```




```
unsigned long  Fact(unsigned int  n)
{
    unsigned int  i;
    unsigned long result = 1;
    if (n < 0)
    {
        return -1;
    }
    else
    {
        for (i=2; i<=n; i++)
            result *= i;
        return result;
    }
}
```

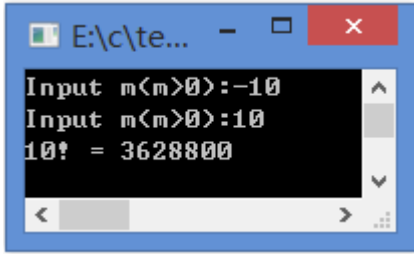
CB warning: comparison of unsigned expression < 0 is always false

如何增强程序的健壮性?

- 如何保证不会传入负数实参?

```c
#include <stdio.h>
unsigned long  Fact(unsigned int  n);
int main()
{
    int  m;
    do{
        printf("Input m(m>0):");
        scanf("%d", &m);
    }while (m<0);
    printf("%d! = %lu\n", m, Fact(m));
    return 0;
}
```

```c
unsigned long  Fact(unsigned int  n)
{
    unsigned int  i;
    unsigned long result = 1;
    for (i=2; i<=n; i++)
        result *= i;
    return result;
}
```



函数设计的基本原则

- 信息隐藏
  - 检查入口参数的有效性、合法性
  - 检查函数调用成功与否

1 函数规模要小

2 函数功能要单一

3 函数接口定义要清楚