



C语言程序设计基础

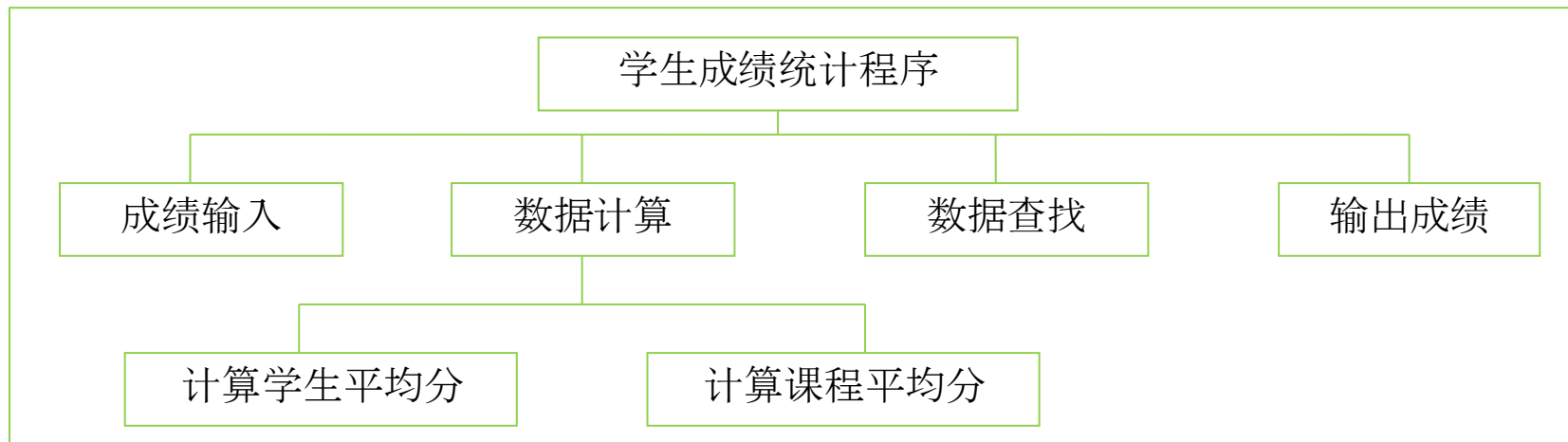
林川



第五章 函数

- 结构化程序设计(Structured Programming)
 - 将复杂程序分解为若干个简单的模块, 用函数进行实现
 - 模块之间相对独立, 通过参数进行调用
- C语言中的函数
 - 众多函数是**平等的兄弟关系**
 - 函数`main`是第一个被执行的函数
 - 函数之间通过**调用**结合在一起

学生成绩统计程序的层次结构图





一、函数是什么？

- 函数是指完成一个特定工作的独立程序模块。
 - 库函数: 由C语言系统提供定义
 - 如scanf()、printf()等函数
 - 自定义函数: 需要用户自己定义
 - 如计算圆柱体体积函数cylinder()
 - main()也是一个函数, C程序有且仅有一个main()

函数定义



```
/* 计算圆柱体体积 */  
double cylinder (double r, double h)  
{  
    double result;  
    /* 计算体积 */  
    result = 3.1415926 * r * r * h;  
    /* 返回结果 */  
    return result;  
}
```



二、函数的定义

函数类型 函数名 (形式参数表)

```
{  
    函数实现语句  
}
```

```
double cylinder (double r, double h)  
{  
    double result;  
    result = 3.1415926 * r * r * h;  
    return result;  
}
```

- 函数类型: 返回值的类型

例如void、int、float, 等等

- 如果返回类型不是void, 必须使用return语句返回函数值

- 形式参数表:

类型1 参数名1, 类型2 参数名2,...,类型n 参数名n

不能把cylinder的形式参数写作: double r, h

指出函数类型、函数名、形参、函数头、函数体、返回值？



```
double cylinder (double r, double h)
{
    double result;
    result = 3.1415926 * r * r * h;
    return result;
}
```



无结果的函数

函数也可以完成一系列操作步骤，不返回任何运算结果。

```
void 函数名(参数表)
{
    函数实现语句
}
```

- void不可少，否则默认为int
- 可以没有return语句
- 可用return提前结束函数(一般是在分支语句中)



三、函数的调用

- 定义一个函数后，就可以在程序中调用这个函数
- 调用标准库函数时，在程序的最前面用
`#include < >`
命令包含相应的头文件
- 调用自定义函数时，程序中必须有相应的函数定义。



函数调用的形式

函数名(实际参数表)

- 实际参数表与形式参数表对应
 - 可以是常量、变量、表达式
- 返回结果的函数调用(使用返回值)
 - `volume = cylinder(radius, height);`
 - `printf("%f\n", cylinder(radius, height));`
- 无返回结果的函数调用(完成操作)
 - `pyramid(5);`

函数调用的过程



- 计算机在执行程序时, 从主函数main开始执行
- 如果遇到某个函数调用, 主函数被暂停执行, 转而执行相应的函数, 该函数执行完后, 将返回主函数。然后再从原先暂停的位置继续执行。
- 在函数中, 如果执行完所有语句或者执行到return语句, 那么将返回主函数

分析函数调用的过程



```
#include <stdio.h>
int main( void )
{
    .....
    volume = cylinder (radius, height ); /* 调用函数 */
    .....
}

double cylinder (double r, double h)
{
    /* 执行函数中的语句 */
    .....
    return result; /* 返回调用它的地方 */
}
```



函数调用的参数传递

- 函数定义时的参数被称为**形式参数**(简称**形参**)
`double cylinder (double r, double h);`
- 函数调用时的参数被称为**实际参数**(简称**实参**)
`volume = cylinder (radius, height);`
- 实参□形参
 - 在参数传递过程中, **实参**把**值复制**给**形参**。
 - 形参和实参一一对应
 - 数量、类型、顺序
- 实参:**常量**、**变量**或**表达式**
- 形参:**变量**, 用于接受实参传递过来的值



函数结果返回

- 函数结果返回的形式：

- return 表达式；

- return (表达式)；

- 对于void类型的函数，表达式为空

- 函数返回的两种情况

- 完成运算，将结果返回给主调函数。

- 只是完成工作，无需返回结果给主调函数

- 函数类型为void。

判断奇偶数的函数



定义一个判断奇偶数的函数: 当参数为偶数时返回1, 否则返回0。

```
int even( int n )
```

判断奇偶数的函数



```
int even (int n)          /* 函数首部 */
{
    if( n % 2 == 0 )      /* 判别奇偶数 */
        return 1;        /* 偶数返回1 */
    else
        return 0;        /* 奇数返回0 */
}
```

如何调用该函数？

例如：

```
even( 3 );
if( even(x) )
    printf("x is even\n");
```




函数原型声明

函数类型 函数名(参数表);

即:函数定义中的第1行(函数首部), 并以分号结束。

例如

```
double cylinder (double r, double h);  
void pyramid (int n);
```

声明函数时, 可省略形式参数的名字, 因为无关紧要。

例如

```
double cylinder (double, double);  
void pyramid (int);
```

但是建议不要省略, 增加可读性

函数原型声明



函数类型 函数名(参数表);

- 函数必须先定义后调用
 - 将主调函数放在被调函数的后面, 就像变量先定义后使用一样
- 函数定义可以出现在主调函数之后
 - 需要在函数调用前, 声明函数原型
 - 说明函数的类型和参数的情况, 以保证程序编译时能判断对该函数的调用是否正确。



四、函数应用

[例5-2] 定义一个函数判定一个整数的奇偶

```
int even( int n )  
{  
    if( n%2==0 )  
        return 1;  
    else  
        return 0;  
}
```



函数程序设计

[例5-3] 使用格里高利公式，求 π 的近似值。输入精度 e ，精确到最后一项的绝对值小于 e 。

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

<https://zh.wikipedia.org/wiki/%CE%A0%E7%9A%84%E8%8E%B1%E5%B8%83%E5%B0%BC%E8%8C%A8%E5%85%AC%E5%BC%8F>

定义函数funpi, 求 π 的近似值



```
double funpi (double e)
{
    int denominator = 1, flag = 1;
    double item = 1, sum = 0;

    while (fabs (item) >= e)
    {
        item = flag * 1.0 / denominator;
        sum = sum + item;
        flag = -flag;
        denominator = denominator + 2;
    }

    return sum * 4;
}
```

源程序



```
#include <stdio.h>
#include <math.h>
int main (void)
{
    double error, pi;
    double funpi (double e);
    printf ("Enter error:");
    scanf ("%lf", &error);
    pi = funpi (error);
    printf ("pi = %f\n", funpi(error));

    return 0;
}
```



输出数字金字塔

1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

数字金字塔源程序



```
  1
 2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
#include <stdio.h>
int main (void)
{
    void pyramid (int n);
    pyramid(5);
    return 0;
}
```

```
void pyramid (int n)
{
    int k, j;
    for( k = 1; k <= n; k++ )
    {
        for( j = 1; j <= n-k; j++ )
            printf(" ");          /* 空格 */
        for( j = 1; j <= k; j++ )
            printf("%d ", k);     /* 数字 */
        putchar ('\n');          /* 换行 */
    }
}
```




复数运算

输入两个复数的实部和虚部，采用函数计算它们的和与积。

复数的计算结果有两个：实部和虚部

但是，return语句只能返回一个计算结果

怎么办？？？



[例5-6, P100]复数运算源程序

```
#include <stdio.h>
#include <math.h>
```

定义全局变量
保存和传递计算结果

```
float result_real, result_imag;
```

```
/* 复数之和 */
```

```
void complex_add(float r1, float m1, float r2, float m2)
{
    result_real = r1 + r2;
    result_imag = m1 + m2;
}
```

```
/* 复数之积 */
```

```
void complex_prod(float r1, float m1, float r2, float m2)
{
    result_real = r1*r2 - m1*m2;
    result_imag = r1*m2 + r2*m1;
}
```

复数运算源程序



```
int main(void)
{
    float real1, real2, imag1, imag2;
    ..... /* 输入两个复数, 此处略 */
    complex_add(real1, imag1, real2, imag2);
    printf("addition of complex is %f+%fi\n",
        result_real, result_imag);
    complex_prod(real1, imag1, real2, imag2);
    printf("product of complex is %f+%fi\n",
        result_real, result_imag);
    return 0;
}
```



五、变量分类

- 全局变量
 - 在函数外部定义的变量
 - [例5-6]中的 `result_real` 和 `result_imag`
 - 从定义起，之后的函数都可以使用
 - 定义之前的函数不可以
 - 作用范围广，使用方便
 - 省去参数传递

全局变量



- 典型应用
 - 软件运行的全局状态
 - 公共资源
 - 传递运算结果
- 避免滥用
 - 所用范围广，缺点+优点，
 - 不易管理、破坏程序的模块化
 - 慎用、尽量少用

局部变量



- 函数内部的变量
 - 局部变量
- 函数的形式参数
 - 特殊的局部变量
 - 函数调用时, 用于接收的实际参数值
- 复合语句内部的变量
 - 局部变量



局部变量和全局变量

- 局部变量

- 函数内部定义的变量(包括形参)

- 作用范围: 本函数内部

- 定义在复合语句内的变量

- 作用范围: 复合语句内部

- 全局变量

- 在所有函数外部定义的变量, 不属于任何函数

- 作用范围: 从定义处到源文件结束(包括各函数)

局部变量和全局变量



```
#include <stdio.h>
int x;
int f();
int main (void)
{
    int a = 1;
    x = a;
    a = f();
    {
        int b = 2;
        b = a + b;
        x = x + b;
    }
    printf ("%d %d" , a, x);
    return 0;
}
int f()
{
    int x = 4;
    return x;
}
```

运行结果
4 7

在函数f内部, 全局变量x被f的局部变量x屏蔽了

变量作用范围示例



```
int x=1;
void main( )
{
    int a=2;
    .....
    {
        int b=3;
        .....
    }
    f();
    .....
}
int t=4 ;
void f( )
{
    int x=5, b=6;
    .....
}
int a=7;
.....
```

x=? a=? b=?
x=1, a=2, b=3

b=?
b没有定义

x=? b=? t=? a=?
x=5 b=6 t=4 a没定义

变量的生命周期



- 变量的生命周期
 - 变量从分配存储单元开始, 到被回收存储单元的过程
- 局部变量
 - 函数调用时, 定义变量, 分配存储单元
 - 函数调用结束, 存储单元自动被收回
 - 包括形式参数
- 全局变量: 从程序执行开始, 到程序的结束, 存储单元**始终存在**



变量的存储类型

- 自动型(auto)
 - 普通的局部变量
- 静态型(static)
 - 静态的局部变量
 - 生命周期和全局变量一样
 - 静态的全局变量
 - 只能在本文件中使用
- 外部变量(extern)
 - 全局变量, 可跨文件使用
- 寄存器型(register)
 - 存储在硬件寄存器中的变量

操作系统(例如Windows)、语言系统		
程序区(代码) 主函数main、其他子函数等		
数据区	静态	全局变量
		静态局部变量
	动态	主函数局部变量区
		其他函数的局部变量区

静态变量



static 类型名 变量表

- 作用范围
 - 保持不变
- 生命周期
 - 程序开始执行直到程序结束
 - 等同于全局变量
- 静态变量的初值
 - 定义的时候给初值
 - 否则, 缺省为0(每一个bit都是0)



[例5-9] 静态变量示例:计算1-n的阶乘

```
#include <stdio.h>
double fact_s( int );
int main( void )
{
    int i, n;
    printf("Input n: ");
    scanf("%d", &n);
    for( i=0; i<n; i++ )
        printf("%3d!=%.0f\n", i, fact_s(i));
    return 0;
}
double fact_s(int n)
{
    static double f = 1;
    if (n == 0){
        return 1;
    }
    f = f * n;
    return f;
}
```

静态变量的初值为1

静态变量会记住前一次调用时的值



本章要点

- 函数定义、声明、调用
- 函数参数, 参数传递
- 变量的分类、生命周期、存储类型
- 局部变量、全局变量、静态变量
- 全局变量优缺点, 静态变量的特性



作业

- 以下函数都需要实现配套的测试用例
 - 实现计算长方体体积的函数

```
double cuboid(double length, double width, double height);
```

- 实现计算第 n 项斐波拉契数列的函数($n \geq 0$)

```
int fib(int n);
```

- 实现计算三个数中最大值的函数:

```
int max(int a, int b, int c)
```