高级语言程序设计

张伯雷
bolei.zhang@njupt.edu.cn
bolei-zhang.github.io
计算机学院，软件教学中心

高级语言程序设计

第05章　函数的基本知识

1. **初识计算机、程序与C语言**
2. **初识C源程序及其数据类型**
3. **运算符与表达式**
4. **程序流程控制**
   - 语句与程序流程
   - 顺序结构
   - 选择结构
   - 循环结构
   - break， continue

内容提要

- 模块化程序设计与函数
- 函数的定义
  - 函数的声明
  - 函数的定义
- 函数调用
- * 递归函数
- 变量的作用域与存储类型
- 应用举例

输出100以内的所有质数

```c
//输出100以内的所有质数
#include<stdio.h>
int main(){

    int i = 0, j = 0;
    for(i=2; i<=100; i++){
        int is_prime = 1;
        for(j=2; j<i; j++){
            if (i%j == 0){
                is_prime = 0;
            }
        }
        if(is_prime)
            printf("%d\n", i);
    }
}
```

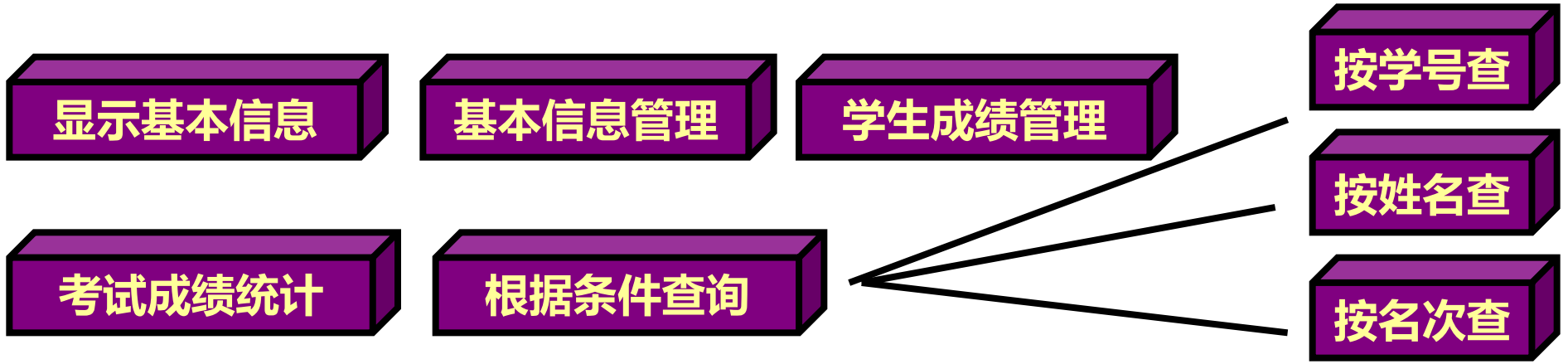
模块化程序设计与函数

- 函数（function）：C程序的基本单位
- 函数分两类 { 库函数 / 用户自定义函数
- 函数 - - 模块化程序设计
- 模块化程序设计：

  自顶向下、逐步分解、分而治之
- 函数：模块化程序设计的最小单位、基石。

模块化程序设计与函数

- 例：一个简单的学生成绩档案管理系统
  - 实现功能：读入学生信息、以数据文件的形式存储学生信息；增加、修改、删除学生的信息；按学号、姓名、名次查询学生信息；统计每门课的最高分、最低分以及平均分；计算每个学生的总分并排名
  - 分成5大模块，各大模块又可以分几个小模块：



函数的定义

- 举例5_2
- 自定义函数totalCost的完整定义由两个部分组成：
  - 函数声明或函数原型
    - 一般出现在main函数前
  - 函数定义
    - 一般出现在main函数后

例5.2

```c
#include<stdio.h>
double totalCost(int n, double p);
int main(){
    double price, bill;
    int number;
    scanf("%d", number);
    scanf("%lf", &price);
    bill = totalCost(number, price);
    printf("%.1f", bill);
    return 0;
}
```

```c
double totalCost(int n, double p){
    const double DISCOUNT = 0.2;
    double total;
    if(n > 1)
        total = n*p*(1-DISCOUNT);
    else
        total = n*p;
    return total;
}
```

函数的声明

- 函数（原型）声明：如果出现先调用后定义的情况，则必须在函数调用之前作原型声明
- 原型声明形式：

  函数返回类型 函数名（[形式参数表]）；
- 说明：
  - （1）分号：
  - （2）形式参数表：变量名可以省略，类型保留
  - （3）声明位置：建议预处理指令之后第一个函数定义前，或者在主调函数的调用点之前

函数的定义

- 函数定义的格式为：

  返回值类型 函数名（[形式参数表]）

  {

  一组语句

  }

- 函数首部（也称函数头）：
  - 返回值类型：结果的类型
  - 函数名：用户自定义标识符
  - 形式参数表：参数及对应类型

函数的定义

- 函数体就是函数代码实现部分：

```
{
    /* 说明性语句  */
    /* 执行语句  */
}
```

- 若返回值类型不是void，使用“return 表达式;”语句；
- 若返回值类型为void，使用“return ;”语句，也可以没有return语句

函数的定义

- 例5_3 定义一个函数int judgePrime(int n)，实现判断任意一个正整数是否是质数。

```
/*函数功能：判断一个正整数是否为质数
 函数参数： 一个整型形式参数
 函数返回值： int型，用值1表示n是质数，0表示不是质数
*/
int judgePrime(int n)
{
    int i,k ;
    int judge=1;
    if ( n==1 )
       judge=0 ;
    k = (int) sqrt ( n );
    for (i = 2; judge && i<=k ; i++)
        if (n % i == 0)
            judge=0 ;
    return judge;
}
```

函数的定义

- **例5_4 定义一个函数drawLine，实现画出一条由30个减号组成的横线**

```
/*函数功能：画一条由30个减号组成的横线
 函数参数： 无
 函数返回值：无
*/
void drawLine ( )
{   int i;
      for (i=1;i<=30;i++)
            printf("-");
      printf("\n");
     return ;
}
```

函数的调用

- 调用一个已定义函数的基本形式为：

  函数名([实际参数表]);

- 例5_5 从键盘上读入一个整数m，如果m小于等于0，则给出相应的提示信息；如果m大于0，则调用judgePrime函数判断它是不是质数，并在屏幕显示。

```
……
int prime=judgePrime(m) ;
if  ( prime )
      printf ( "%d is a prime!\n" , m ) ;
   else
      printf ("%d is not a prime!\n", m ) ;
     ……
```

函数的调用

- 例5_6 调用drawLine函数实现划线功能。

```
#include <stdio.h>
 /* 此处省略例5.3中函数定义的代码*/
int main ( )
{
   drawLine ( );       /*第一次调用drawLine 函数*/
   printf ("C is a beautiful language!\n");
   drawLine ( );       /*第二次调用drawLine 函数*/
   return 0;
}
```

函数的调用

- 函数调用的完整过程

  转向　传参　执行　返回　继续

- 实参（实际参数）与形参（形式参数）

  - （1）形参在函数定义时不占内存，被调用时占内存，用实参的值初始化。调用结束释放空间。
  - （2）实参与形参个数、对应数据类型完全一致

函数的调用

- 返回值类型：指明了函数执行结束后结果的类型

函数返回值类型	函数体内是否有return语句	函数执行到何处返回到调用点
void型	可以没有，但建议用return ; 形式的语句	如果没有return语句，则执行到函数体结束的右大括号返回调用处；如果有return语句，则执行到return语句处就返回调用处
非void型	必须有return 表达式 ; 形式的语句	执行到return 表达式 ; 处就返回到调用处，无论后续是否还有其他的语句

- （1）无名变量：当函数执行到return表达式；时，将自行定义无名变量接受return后表达式的值，该变量在被主调用函数使用过之后即消失
- （2）实参形式：表达式，其个数与形参个数必须完全相同，对应数据类型最好完全一致

递归函数*

- 函数的递归：在定义函数时直接或间接地调用了自己
  - 直接递归：A - -〉A （本章所讲）
  - 间接递归：A - -〉B - -〉A
- 用递归求解问题的3个条件：
  - 原问题可转化为小规模的新问题
  - 可以应用这个转化过程使问题得到解决
  - 必须有一个结束递归的终止条件

递归函数

- 很多问题适合用函数方法求解：

  - 阶乘问题：$n!=\begin{cases}1 & (当n=0时)\\ n*(n-1)! & (当n>0时)\end{cases}$
  - 乘幂问题：
  - $x^n=\begin{cases}1 & (当n=0时)\\ x*x^{n-1} & (当n>0时)\end{cases}$
  - 数制转换问题
  - 汉诺塔问题（经典，请自己查阅、学习）

- 例5_7 定义一个递归函数实现求n！ 主函数中读入任意一个整数，调用函数实现求阶乘。
- （注意 与非递归函数作对比，重在理解递归的过程）

递归函数

递归函数求解

```
double Fact( int  n)
{
      if (!n)
          return (1.0);
      return (n*Fact(n-1));
}
```

非递归函数求解

```
double Fact( int  n)
{
      int i;
      double f = 1.0;
      for (i=1; i<=n; i++)
            f *= i;
      return f;
}
```

- “递”即递推，表示将复杂的原问题转化为同类型同方法的简单问题的过程；
- “归”即回归，表示从递归调用终止处依次一层层向前返回处理结果。

递归函数

- 例5_8 数制转换：将十进制数n转化为B进制数
- 算法步骤：
  - 重复执行以下步骤（1）和（2），直到n为0。
  - （1）利用取余运算n%B得到B进制数的一位，值的范围肯定是0到B-1。
  - （2）利用整除运算n=n/B将B进制数降一阶。
  - （3）从后往前输出每一次的余数，也就是说，第一次得到的余数最后一个输出，最后一次得到的余数最先输出。

递归函数

```
void MultiBase(int n, int B)
{
      int m;
       if (n)
      {   MultiBase(n/B,B);
            m=n%B;
            if(m<10)
                        printf("%d",m);
            else
                        printf("%c",m+55);
        }
}
```

习题

- 1. 用递归实现斐波那契数列

- 2. 验证2000以内的哥德巴赫猜想：大于等于4的正偶数，都可以分解为两个质数之和。

习题

- 1. 用递归实现斐波那契数列

- 2. 验证2000以内的哥德巴赫猜想：大于等于4的正偶数，都可以分解为两个质数之和。

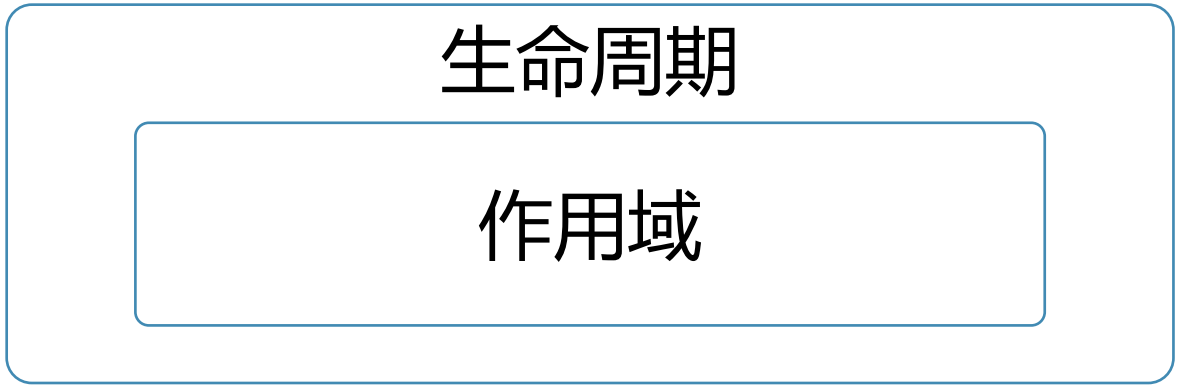
内容提要

- 模块化程序设计与函数
- 函数的定义
  - 函数的声明
  - 函数的定义
- 函数调用
- * 递归函数
- 变量的作用域与存储类型
- 应用举例

```
返回值类型  函数名（[形式参数表]）
{
                一组语句
}
```

变量的作用域与存储类型

- 自定义函数引出了两个问题：
  - 第一，每一个变量在什么范围内起作用
  - 第二，每一个变量何时生成、消失
  - 变量的作用域：在程序哪一部分可见并发挥作用
  - 变量的生命周期：变量所占用的空间从创建到撤消的时间。
  - 二者的关系：一个变量如果不在其生命周期，肯定无作用域可言；如果在其生命周期，也未必一直起作用。



变量的作用域

- 变量的作用域取决于变量定义的位置
- 位置有3种
  - **函数体外** **外部（全局）变量，定义点到程序结束，但去掉同名局部量范围** → 全局变量
  - **函数体内** **作用范围为本函数体内** → 局部变量
  - **函数语句块内** **作用范围仅限于本语句块内** → 局部变量
- 例5_9 找出2-100之间所有的质数，并统计个数。

变量的作用域

```
#include <stdio.h>
#include <math.h>
int count;
int judgePrime(int n);
int main()
{  …
    for (i= 2; i< 100; i++)
    if (judgePrime(i))   /
    {
        printf("%d ", i);
        count++;
    }
   …
}
```

变量的作用域

变量名	变量定义位置	变量性质	变量的作用域	特别说明
count	所有函数之外，程序的最开头	全局（外部）变量	从定义位置开始到程序结束，在下面的main和judgePrime函数中均可见	编译后一直占用空间直至程序结束，其值变化按程序整个执行过程连续变化
i	main函数体开头	局部变量	main函数体内	控制循环，并作为调用函数的实际参数变量
n	judgePrime函数形参表中	局部变量	judgePrime函数体内	其对应实参是main函数中的变量i
i	judgePrime函数体开头	局部变量	judgePrime函数体内	与main函数中局部变量同名，但作用域不同，无冲突
judge	judgePrime函数体开头	局部变量	judgePrime函数体内	用于存判断结果，1表示是质数，0表示不是质数
k	judgePrime函数体for循环语句块内	局部变量	只在for循环体内，不能在函数的其他位置访问	语句块的一对括号类似于函数体的一对边界，作用域不出此界

变量的作用域

对例5_9分别作以下几种修改，观察程序编译、运行结果，并分析原因。

① 在judgePrime函数体内增加一语句：

```
printf("count=%d\n",count);
```

② 随后，在main函数体内增加变量定义：

```
int count=0;
```


31

变量的存储类型

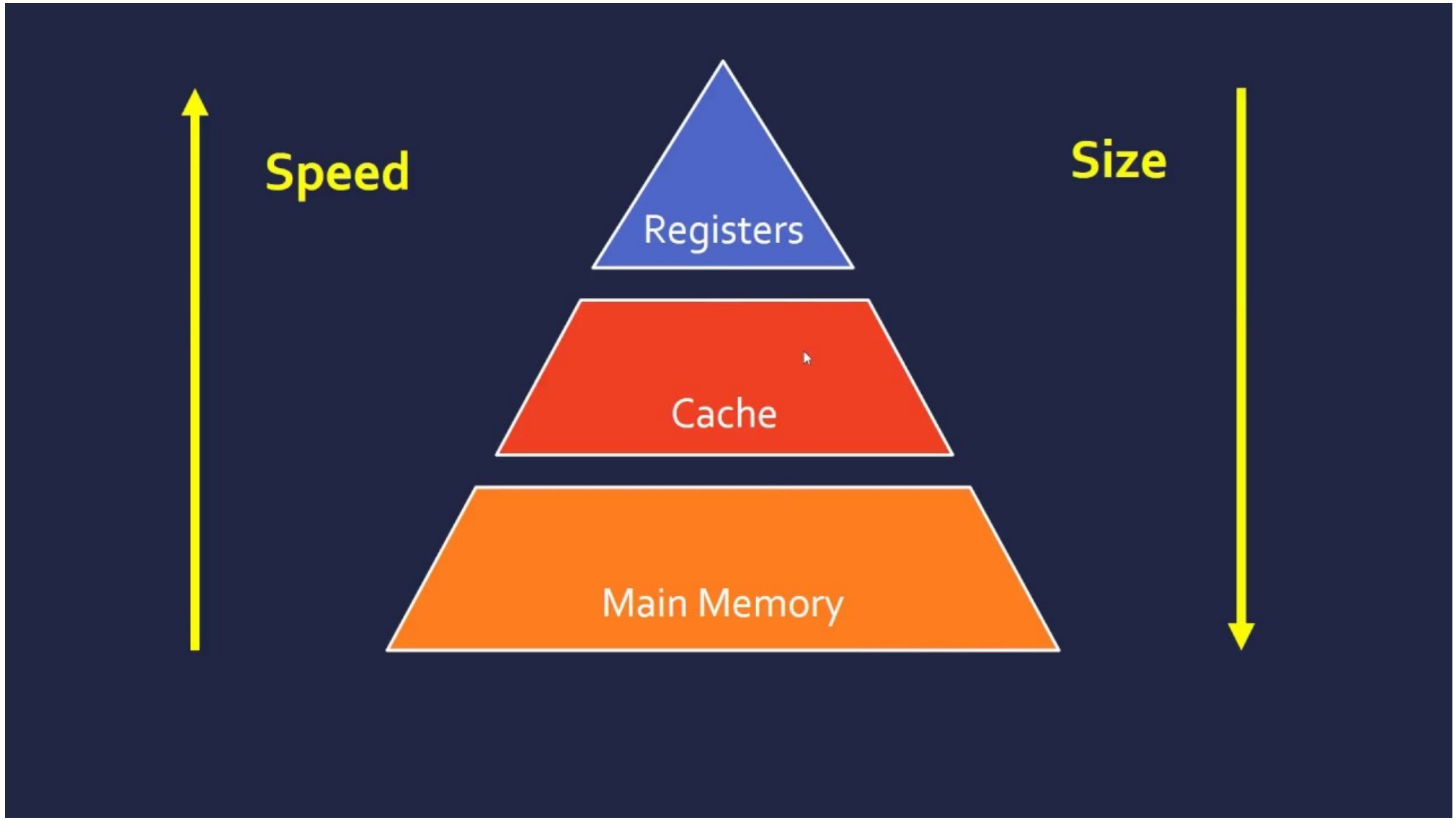
- 变量的生命周期取决于变量的存储类型
- C语言中变量有4种不同的存储类型(存储说明符)
  用关键字auto、 static、extern 、register表示

[存储说明符] [数据类型] [变量名称] [=初始化];

存储类型符	存储类型	存储位置	说明
auto	自动存储	内存堆栈区	变量定义，**缺省**存储类型
static	静态存储	内存数据区	变量定义，表明变量位于静态存储区
extern	外部存储	内存数据区	变量声明，表示变量来自同程序的其他文件
register	寄存器存储	CPU通用寄存器	变量定义，表明变量来自寄存器区域

32

计算机存储, register



extern

- 1. 要访问**后定义**的变量，需要变量前加extern做声明

- 2. 如果程序由多个文件组成，某文件中需要访问**另一个文件**的**全局变量**，则要做声明。

变量的存储类型

- 例5_10 利用静态局部变量求解从1到5的阶乘

```
#include<stdio.h>
int fun(int n) {
      static int f= 1; /*定义静态局部变量f*/
      f=f*n;                /*求n的阶乘f*/
      return f;             /*返回阶乘值*/
}
int main() {
     int i;                 /*定义局部变量i*/
     for(i =1; i <=5; i++)   /*循环，依次求1 到5的阶乘*/
           printf(  “%d != %d\ n" , i , fun(i));
     return 0;
}
```


35

变量的存储类型

- 静态局部变量的特点
  - （1）编译阶段在静态存储区分配存储空间，并且一直占用到程序结束。
  - （2）定义位置在函数内部，仍然是局部变量，作用域仅限于本函数，仅在本函数被调用时才能被访问。
  - （3）第一次进入函数时被初始化，若未指定初值，将自动初始化为0。
  - （4）第二次及以后进入函数时不再初始化，而从“休眠”状态“苏醒”，获取之前的值。每次退出函数时就进入“休眠”状态。

36

变量的存储类型

- 思考题：

对例5_10分别作以下几种修改，观察程序编译、运行结果，并分析原因。

① 将函数fun中的static int f= 1;改为int f= 1;

② f恢复成静态局部变量，将main函数中的for(i=1;i<=5;i++) printf("%d != %d\n" , i , fun(i)); 改为printf("%d != %d\n" , 5, fun(5));，那么输出结果是5的正确阶乘吗？

③ 如何在main函数中稍加修改，达到求sum=1!+2!+3!+4!+5!的效果。

37

应用举例——定义函数求解面积和体积

- 例5_11 求给定半径的圆形面积、给定半径的球形体积以及给定半径和高的圆柱体积。
- 一个大型的复杂的程序，按自顶向下、逐步细化、模块化的结构化程序设计方法设计
- 在程序设计时，每一个函数的功能力求简单清晰，代码量较少，可读性强。整个程序通过函数之间的调用及流程控制共同完成整个程序的功能。
- 用数学公式求解
- 程序共分三个层次五个函数，具体见编程环境下的演示

38

本章小结

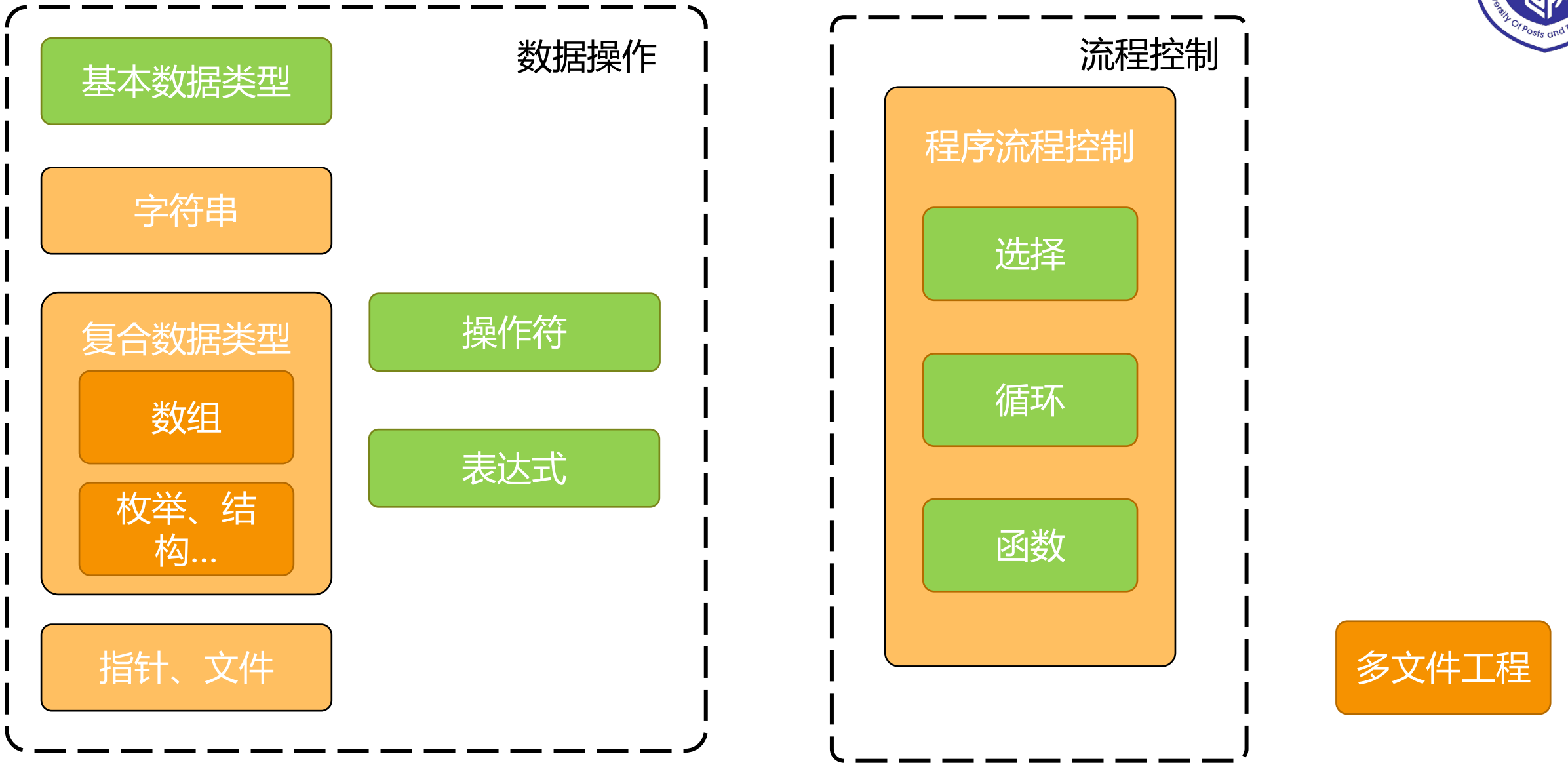
- 结构化程序设计思想与函数
- 函数的定义、调用 及声明
- 函数调用的完整过程
- 递归函数的定义及调用执行过程
- 变量的存储类别、生命期与作用域问题

习题

- 1. 找出100到999之间的水仙花数：$153 = 1^3+5^3+3^3$

- 2.通过辗转相除法计算两个非负整数a，b的最大公约数。

C语言


数据操作
基本数据类型
字符串
复合数据类型
数组
枚举、结构...
指针、文件
操作符
表达式
流程控制
程序流程控制
选择
循环
函数
多文件工程