

## 数据

- 数据类型
- 数据存储
- 数据访问
- 数据处理

## 运算

- 运算规则
- 运算符
- 表达式

## 控制

- 控制语句
- 控制算法
- 结构化设计
- 模块化设计

## 传输

- 函数参数
- 函数返回值
- 输入输出
- 文件访问

## ■ 数据类型包含的信息

- 数据值的特征
- 数据的编码方式
- 数据的取值范围
- 数据占用内存空间的大小
- 数据可以施加的运算类型

## ■ 常量的数据类型

- 整型常量缺省类型由`int`开始按照类型等级次序自动选取
- 字符型常量缺省类型是`int`（不是`char`）
  - 注意转义字符的写法, `'\n'`, `'\t'`, `'\\'`, `'\0'`, `'\123'`, `'\xab'`
- 浮点型常量缺省类型是`double`
- 字符串常量类型为字符数组
  - 如`"hello"`的数据类型是`char[6]`

# 运算符的优先级



## ■ 运算符的优先级

- 共15级
- 不需要记住运算符的具体优先级
- 只需要记住主要运算符的优先顺序
- 要能够按照正确的运算次序对表达式求值

## ■ 运算符的运算次序

- 误区：在表达式中，优先级最高的运算符最先被计算
- 例如，逻辑运算符的短路效应

第1优先级：各种括号，如()、[]等、成员运算符

第2优先级：所有单目运算符，如++、--、!、~等；

第3优先级：乘法运算符\*、除法运算符/、求余运算符%；

第4优先级：加法运算符+、减法运算符-；

第5优先级：移位运算符<<、>>；

第6优先级：大于运算符>、大于等于运算符>=、小于运算符<、小于等于运算符<=；

第7优先级：等于运算符==、不等于运算符!=；

第8优先级：按位与运算符&；

第9优先级：按位异或运算符^；

第10优先级：按位或运算符|；

第11优先级：逻辑与运算符&&；

第12优先级：逻辑或运算符||；

第13优先级：三目条件运算符?:；

第14优先级：各种赋值运算符，如=、+=、-=、

第15优先级：逗号运算，。

# 运算符的操作数

## ■ 操作数个数

- 一元运算符、二元运算符、三元运算符

## ■ 操作数的数据类型

- 运算符决定了操作数的数据类型
- 求余运算符`%`：两个操作数都必须是整数类型
- 指针运算符`*`：操作数必须是指针类型
- 下标运算符`[]`：一个操作数必须是指针类型，另一个操作数必须是整数类型



# 运算符的结合方向

## ■ 右结合

- 从右算到左，先右后左
- 除后缀++和后缀--之外的一元运算符
- 赋值运算符
- 条件运算符

## ■ 左结合

- 从左算到右，先左后右

## ■ 左值的概念

- 左值表达式具有存放数据的空间，并且存放总是被允许的

## ■ 左值的应用

- 左值表达式能够作为赋值运算符的左操作数
- 最常见的左值表达式是变量
- 赋值运算符左操作数、**++**和**--**的操作数只能是左值



# 运算符的注意事项

## ■ 求余运算符%

- 两个操作数必须都是整数类型
- 运算结果符号与左操作数符号相同

## ■ 前缀自增++和自减运算符--

- 一元运算符，优先级2，右结合
- 有副作用，运算后操作数的值加/减1，要求操作数必须是左值
- 前缀自增/自减表达式的值是操作数加/减1以后的值

## ■ 后缀自增++和自减运算符--

- 一元运算符，优先级1，左结合
- 有副作用，运算后操作数的值加/减1，要求操作数必须是左值
- 后缀自增/自减表达式的值是操作数的原值

## ■ 赋值运算符

- 优先级14，仅高于逗号运算符，右结合
- 有副作用，左操作数的值会发生改变，要求左操作数必须是左值
- 赋值表达式的值就是被赋的值
- 赋值表达式的数据类型就是左操作数的数据类型
- 注意复合赋值运算符的求值规则

## ■ 赋值表达式的求值过程

- 计算赋值运算符右操作数的值
- 将上述求值结果自动转换成左操作数的类型
- 将上述处理结果装入左操作数的内存空间



## ■ 长度运算符sizeof

- 单目运算符，优先级2，右结合
- 形式一：sizeof(类型名)
  - 必须有括号
  - 求该类型的数据占用内存字节数
- 形式二：sizeof 表达式
  - 不需要括号，但因为sizeof优先级较高，表达式经常需要括号保证运算次序
  - 求该表达式类型的数据占用内存字节数
  - 操作数表达式本身不会被求值
- sizeof表达式的数据类型是int

## ■ 逻辑运算符

- 逻辑表达式的数据类型是int
- 当逻辑表达式运算结果为真时，值为1
- 当逻辑表达式运算结果为假时，值为0

## ■ 逻辑运算符的短路效应

- `a && b`，当表达式a为假时，不计算表达式b
- `a || b`，当表达式a为真时，不计算表达式b
- 当表达式b存在副作用时（赋值、自增/自减），短路效应将影响变量的值

## ■ 条件运算符

- 唯一的三目运算符，优先级13，右结合

## ■ 条件表达式

- $\text{exp1} ? \text{exp2} : \text{exp3}$
- 如果 $\text{exp2}$ 和 $\text{exp3}$ 的类型不一致，则进行隐式类型转换
- $\text{exp1} ? \text{exp2} : \text{exp3} ? \text{exp4} : \text{exp5}$ 
  - 等价于：  $\text{exp1} ? \text{exp2} : (\text{exp3} ? \text{exp4} : \text{exp5})$
  - 而不是：  $(\text{exp1} ? \text{exp2} : \text{exp3}) ? \text{exp4} : \text{exp5}$

## ■ 声明的形式和使用形式相似

- `int *p;`           // \*p的类型是int
- `char a[1][4];`     // a[i][j]的类型是char, a[i]的类型是char[4]
- `double (*p)[3];`   // \*p的类型是double[3], (\*p)[i]的类型是double
- `float *a[4];`       // a[i]的类型是float\*, \*a[i]的类型是float

## ■ 移除声明中的标识符即为类型

- `int *`
- `char [3][4]`
- `double (*)[3]`
- `float *[4]`



# 字符串的输入方式

## ■ scanf()函数

- `scanf("%s", str);`
- 不能输入空白字符

## ■ gets()或fgets()函数

- `gets(str);`
- `fgets(str, MAX_SIZE, stdin);`
- 输入一行字符，以回车符结束

## ■ getchar()函数

- 使用循环逐个字符输入，可以输入所有字符
- 可以自己设定结束指示字符，编程比较麻烦，但使用更灵活
- 不要忘记字符串结束标志 `'\0'` 需要编程补上



# 函数的参数和返回值

## ■ 参数传递

- 实参至形参的“**单向值传递**”
- 实参和形参的类型必须相同或赋值兼容
- 形参的值发生的改变**不能传递**给实参
- 指针型实参和形参的传递同样遵循上述规则
- 指针型实参和形参指向了内存中的**同一数据块**

## ■ 返回值

- 函数返回值的数据类型在函数定义时通过“**函数类型**”指定
- 如果**return**语句中表达式值的类型与“函数类型”不一致，以“函数类型”为准
  - 若为数值类型的表达式，将自动进行类型转换
  - 若无法实现自动类型转换，则编译时发生错误



# 数组作为函数参数

## ■ 一维数组作为函数参数

- 形参可定义为数组或指针形式，但**形参实质上是指针类型**
- 形参定义为数组形式时，长度可以不指定，即使指定也无作用
- 实参可以使用相应类型的数组名，但不可附加多余的下标运算符
- 实参也可以使用相应类型的指针
- 函数调用时，实参表示的数组首地址传递给形参指针，则形参和实参都指向了同一个数组

## ■ 二维数组作为函数参数

- 形参定义为指针形式时，其基类型必须是二维数组的**行**，基类型数组的长度即为二维数组的列数，不可省略
- 形参定义为数组形式时，其行数可以不指定，但列数必须指定
- 实参可以使用二维数组名，或指向二维数组的行指针，**注意列数**

## ■ 如何访问内存数据

- 数据在哪里（地址、指针）
- 数据是什么（数据类型）

## ■ 主要内存数据的访问方法

- 变量：地址由编译器根据变量名判断、数据类型
- 数组元素：数组名（起始地址）、数组下标（地址偏移量）、数组元素类型
- 二维数组的行：数组名（起始地址）、行下标（地址偏移量）、数据类型（行）
- 指针指向的数据：指针值、指针基类型
- 字符串：起始地址（字符指针、字符数组名）、结束标志（'\0'）



# 指针的灵活运用

## ■ 指针的属性

- 地址值：来自内存数据的地址，如，变量地址、数组、字符串、指针赋值、函数名、动态存储分配
- 基类型：由程序指定，变量定义、取地址运算、强制类型转换、函数调用

## ■ 指针的运用

- 可以根据需要选择合适的基类型，并不一定和地址值来源同类型
- 例如，利用指向元素的指针访问二维数组

- `char a[M][N], *p=(char *)a; // p=*a, p=a[0]`

- `p[i*N+j]` 即 `a[i][j]`

## ■ 注意在任何情况下都要避免引用任何类型的无效指针

## ■ 数组与指针

- 数组名大部分情况下可以看做指针
  - 数组名表示数组的起始地址，基类型是数组元素类型
- 数组名不能被修改，称为“常量指针”
  - 不能用作左值：不能作为赋值、自增、自减运算符的操作数

## ■ 数组名什么时候不是指针

- 当需要关心数组在内存中的字节数时，数组不能看作指针
  - 数组的字节数 = 数组元素的字节数 \* 数组元素的个数
  - 指针的字节数 = 8（以64位编译系统为例）
- 什么时候需要关心数组的字节数
  - `sizeof`运算符的操作数
  - 取址运算符`&`的操作数
  - 用作指针的基类型：例如，行指针的基类型

## ■ 定义之后在内存中得到了什么

- 数组：存放所有元素的一段连续内存空间
  - 可以使用初始化或赋值的方式，将数据填充到数组元素的空间中
- 指针：存放一个指针（地址）的内存空间
  - 可以使用初始化或赋值的方式，将一个有效地址存入指针变量的空间中

## ■ 使用字符串常量对字符数组和字符指针初始化或赋值

### ■ 初始化

- `char astr[20] = "Hello World!";` // 字符串内容依次填充进字符数组的元素
- `char *pstr = "Hello World!";` // 字符串常量首地址存入字符指针变量

### ■ 赋值

- `astr = "Hello World!";` // 错误！数组不能整体赋值
- `pstr = "Hello World!";` // 正确！指针指向字符串常量的首地址

## ■ 下标运算符和指针运算的关系

- $a[i]$  等价于  $*(a+i)$  等价于  $*(i+a)$  等价于  $i[a]$
- $a[i]$  可以看作是  $*(a+i)$  的简写，便于书写和理解
- $*(a+i)$  才是其运算的本质
- 表达式中，下标运算符的操作数是（指向数据集合的）指针，而并不要求必须是数组
- 非形参的变量声明中，下标运算符表示数据类型为数组

## ■ 两次下标运算

- $a[i][j]$ ，其中， $a$  可能是二维数组、指向行的指针、一维指针数组、指向指针的指针等任何可以做两次指针运算的数据类型

# 行指针和指针数组

## ■ 行指针

- `int (*p)[N];`
- `p+i`表示指针移动*i*个`int[N]` (*i*行) 的距离, 即`i*N*sizeof(int)`字节
- 其为指针, 可以做下标运算`p[i]`
- 其基类型是数组, `p[i]`可以做下标运算`p[i][j]`

## ■ 指针数组

- `int *p[N];`
- `p+i`表示指针移动*i*个`int*`的距离, 即`i*sizeof(int*)`
- 其为数组, 可以做下标运算`p[i]`
- 其元素为指针, `p[i]`可以做下标运算`p[i][j]`



# 指向指针的指针

---

## ■ 无效指针问题

- 两级指针都必须避免使用无效指针

## ■ 下标运算

- 任何一级指针如果是指向数据集合的，都可以使用下标运算符
- 注意与一维数组、二维数组之间区分清楚

## ■ 二维动态数组

- 若干个用一维指针数组联系起来的不等长一维数组

## ■ 来源

- 回车和换行来自电传打字机
- 回车(Carriage Return, CR)表示回到一行的开头
- 换行(Line Feed, LF)表示换到下一行

## ■ 计算机系统中的换行(newline, line break, EOL)

- 回车符(CR, 0x0D, \r)、换行符(LF, 0x0A, \n)
  - Unix和Linux、Mac OS X系统：以LF (\n) 作为EOL
  - Mac OS 9之前版本系统：以CR (\r) 作为EOL
  - MS DOS/Windows系统：以CR+LF (\r\n) 作为EOL
- 当在键盘输入Enter键，实际输入的字符参照上述描述
- C语言中统一使用LF (\n) 表示Enter键的输入和EOL



# 文件的打开模式

## ■ 文件的打开模式和文本文件、二进制文件

- 打开文件时的“文本模式”和“二进制模式”，并不是说打开的文件是“文本文件”或“二进制文件”
- 任何文件都可以使用“文本模式”或“二进制模式”打开
- 用“文本模式”打开二进制文件可能无法得到期望的结果

## ■ 不同的文件打开模式对回车换行符的处理

- 以文本模式打开文件
  - 输入（读文件）时，“与平台相关的EOL”转换为换行符(LF)
  - 输出（写文件）时，换行符(LF)转换为“与平台相关的EOL”
  - Windows系统下，“与平台相关的EOL”即CR+LF
  - Unix/Linux系统下，“与平台相关的EOL”即LF（不需转换）
- 以二进制模式打开文件，不进行回车符、换行符的转换



## ■ 运算符和表达式

- 求值规则
- 注意运算结果的数据类型和范围

## ■ 格式化输入输出

## ■ 基本语句

- 注意switch、break、continue

## ■ 数组

- 一维数组、二维数组、字符串

## ■ 函数

- 参数传递规则
- 递归调用
- 变量存储类型

## ■ 指针运算

- 数组的指针
- 字符串的指针
- 指针数组和指针的指针
- 指针型函数参数、返回值
- 命名空间

## ■ 结构体

- 结构体指针
- 链表

## ■ 文件操作

- 打开、关闭、读写
- 注意数据块读写fread()和fwrite()

## ■ 字符串操作

- 复制、连接、比较、求长度、求子串位置
- 利用以上操作进行字符串的复杂处理

## ■ 数组操作

- 排序算法：插入排序、选择排序、冒泡排序、交换排序
- 查找算法：普通查找（顺序查找）、折半查找（二分查找）
- 其他算法：逆序、插入元素、删除元素
- 矩阵运算：转置、加法、乘法

## ■ 链表操作

- 建立、遍历、插入结点、删除结点
- 保存数据至文件、由文件重建链表