**八进制数以“0”开头，十六进制“0x”**

**ASCII码:’0’为048；‘A’为065；‘a’为097**

**1.字符序列:**关键字+标识符+字面常量+运算符+分隔符

2.P34 表2.3

**3.标识符（identifier）**

以字母或下划线“\_”开头，由字母、数字、下划线组成的字符串

不能与关键字重名

标识符区分大小写

有效长度有规定（一般有效长度为32）

4.**常量：固定不变的量**

字面常量（属于C++语言的词汇）

整型常量（int）：整数

浮点型常量（float）：小数

字符常量（char）：一个字符，由单引号标识‘a’

字符串常量：多个字符，以字符’\0’结尾，由双引号标识“abcde\0”

**（1）整型常量**

int 型常量即整型常量，实际上就是整数。

C++程序中除允许一般的十进制整数之外，还允许八进制整数和十六进制整数出现。

八进制：以0开头的整数

十六进制：以0和字母X(x)开头的整数

注意，在C++程序中，各种进制的整数都自动转换为十进制输出

例如：cout<<023<<“ ”<<23<<“ ”<<0x23;将输出不同的十进整数：19˽23˽35

**（2）浮点型常量**

小数点表示法：4.75，2.0 （整数2，浮点数2.0）

科学表示法：

注意，在C++程序中，浮点数以十进制的形式输入和输出，浮点数的存储格式与系统要求有关

例如：C++中float型浮点数占4个字节，double型浮点数占8个字节，long double型浮点数占10个字节。

浮点型常量按double型处理，占8个字节

加后缀f或F可按float型处理，占4个字节

浮点型常量有表示范围，见P63表3.2

**浮点类型的储存**

**（3）字符型常量**

用单引号括起的基本符号

‘A’，‘g’ ，‘3’，’˽’

占1个字节

表示范围：-128~127

ASCII码将字符型常量与整型常量建立对应关系

0~127，P32表2.2

**布尔型常量**

有两个值

false

true

**字符串常量**

用双引号括起来的字符序列

以字符‘\0’结尾

字符串长度为字符数+1，如“string constant”长度为16

**无值型**

用关键字void描述无值型

无值型的值域为空集

主要应用

声明函数的返回值

声明函数的参数

声明指针

void \*<指针变量名>

该指针可以指向任何数据类型

不能用void类型声明变量

**用户定义类型**

**用关键字enum描述枚举类型**

将数据的取值（枚举值）列举出来，不能超出此范围取值

定义枚举类型

enum<枚举类型名>{<枚举值表>}<枚举变量表>

枚举类型名：标识符

枚举值表：枚举类型数据的取值列表

值名（是一个标识符）

<值名> = <整型常量>

枚举变量表：声明为此类枚举类型的变量

enum <枚举类型名> <枚举变量表>

**说明：**

枚举类型是整型的子集

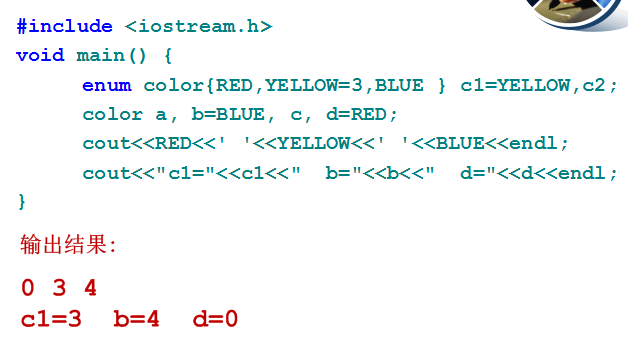
n个枚举值全部未赋整型常量值时，从左向右对应整数0~n-1

第i个枚举值赋予常量m，则从第i+1个枚举值开始，分别对应整数m+1，m+2，……，直到下一个赋了值的枚举值或结束

枚举类型变量不能赋整数值

枚举值按照常量处理，是整型常量

枚举值用标识符表示，但是输出为整数

****

**定点类型是整数类型**

整型及其派生类型

字符型及其派生类型

布尔型

枚举类型

**浮点类型是实数类型（可以理解为小数）**

单精度浮点型

双精度浮点型

长双精度浮点型

1. **变量说明**

[存储类] <数据类型> <变量名表>;

存储类，5种，可省略

auto：自动变量，默认

register：寄存器变量

static：静态变量

extern：外部变量

数据类型，不可省略

根据数据类型，在内存中开辟相应大小的空间

**全局变量和局部变量**

全局变量：程序运行的整个过程，内存中位置不变

局部变量：程序运行的过程中，所占内存反复占用和释放

**生存期和作用域**

生存期：变量所占内存空间由分配到释放的时期

作用域：变量有效的范围，说明变量时所在的语句

变量的存储类属性

变量的初始化

**在变量说明语句中对变量进行赋值.**

C++允许为不同变量起同一名字，同名变量按最近的说明处理，所以变量生存期与作用域可能不同

(1)尽量不要在一个变量名的生存期再定义一个同名变量

(2)不要在一个变量的生存期外使用该变量





**表达式包含多个同级运算符时，运算的先后次序**

左结合规则

从左向右依次计算

双目的算术运算符、关系运算符、逻辑运算符、位运算符、逗号运算符

右结合规则

从右向左依次计算

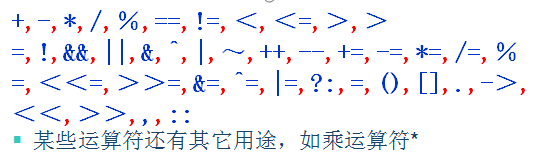
可以连续运算的单目运算符、赋值运算符、条件运算符

**5.运算符**

由字母、数字之外的第三类基本符号组成

个别关键字如sizeof、new、delete，也被认为是运算符

其余运算符为：



**6.分割符（separator）**

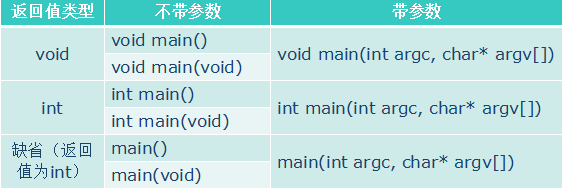
没有明确的含义，但在程序中必不可少

用来界定或分割程序中的语法成分，类似于“标点符号”

常用分割符有：

1546179610(1)

**7.主函数写法：**



1. **预处理命令：**

**文件嵌入命令**

**要求系统在编译之前把它指明的文件嵌入到该命令所在位置**

**＃include < <文件名> >嵌入的是系统提供的头文件**

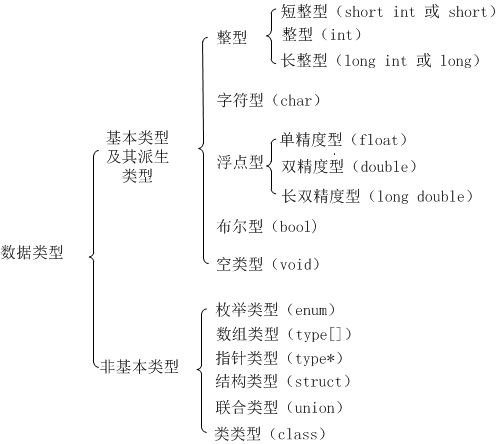
**＃include “ <文件名> ”嵌入的头文件是用户自己设计的头文件**

**每一include 命令只指定一个包含文件**

**文件的包含可以嵌套**

**例如**

**#inlcude<iostream>将文件iostream包含在C++源程序中调用该命令的位置**



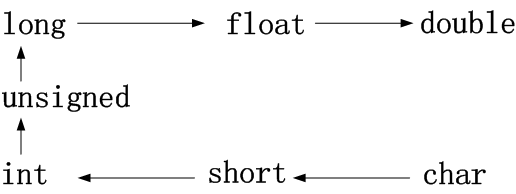
**数据类型转换**

算术运算分量的数据类型可以不同

整数、浮点数、字符

运算分量必须转换为同一类型才能够进行运算，运算结果与运算分量的数据类型相同

占空间少的类型自动转换为占空间多的类型



两个整数进行除法运算，结果仍然为整数

1/3的结果为0

对0.333333……取整，得到整数0

5/3的结果为1

**算术运算分量的说明**

%只能用于整数算术运算

布尔型能够参加整数运算

枚举型能够参加整数运算

**关系运算也称为比较运算，对相同类型数据进行关系运算，结果为布尔类型**

关系运算符

==、>、<、>=、<=、!=

关系表达式

<运算分量> <关系运算符> <运算分量>

运算分量为数值类型或指针类型

运算分量的数据类型相同（可支持隐式转换）

数据类型转换

布尔型与整型的相互转换

bool b = 7;（b的布尔值为true）

int n = true;（n的整型数值为1）

**逻辑表达式的短路问题**

在逻辑表达式求值过程中采用的优化技术

逻辑与运算表达式

如果第一个运算分量逻辑值为false，第二个运算分量值不再计算

表达式值为false

逻辑或运算表达式

如果第一个运算分量逻辑值为true，第二个运算分量值不再计算

表达式值为true

避免在逻辑表达式的运算分量中进行改值运算

**C++语句的分类**

**说明语句（Declaration statement）**

**表达式语句（Expression statement）**

空语句（Null statement）

**控制语句**

分支语句（Selection statement）

循环语句（iteration statement）

无条件转向语句（Jump statement）

**复合语句（Compound statemen**t**）**

**标签语句（Labeled statement）**

**数组(Array)**

数组是同类型元素（分量）的有序组合体。元素的类型可以是C++语言中允许使用的任何一种数据类型（包括任何用户自定义类型）

数组中的每个元素都有与其对应的下标以标明该元素在数组中的位置。对数组元素的访问通常借助于下标来进行，元素也被称为下标变量。每个数组元素（即下标变量）都可以当作单个变量来使用。

**说明一维数组的格式**

<类型名> <数组名> [ <元素数> ] = { <初值表> }

<类型名>指出数组元素的类型，也称为数组类型

<数组名>是一个标识符，是为数组起的名字，该名字还代表数组首元素的地址

<元素数>指定数组的大小，它必须是一个整数或一个整型的常量表达式

<初值表>部分可有可无，若有的话，用于为数组元素置初值,由一批以逗号分割的常量值所构成

举例：声明数组

int a[10];

在内存中开辟10个整型数据的空间（4字节×10）

通过下标访问数组中的每一个元素

<数组名>[下标表达式]

下标的范围：0~9

下标可以是值为整数的表达式，如a[3+5]、a[i\*2]等等，但表达式的值不能超出范围

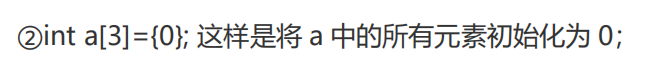
超出范围为“数组下标溢出”错误

数组中的每一个元素都等同于一个与数组类型相同的变量

int B[3]= {4,3,2}; //B[0]=4,B[1]=3,B[2]=2

int B[10] = {4,3,2}; //B[3]=0,为自动赋值

int B[]={4,3,2}; //数组B的大小为3



**二维数组**

具有两个下标的数组叫做二维数组

二维数组经常用来表示按行和列格式来存放信息的数据表

要区分表中某个特定的元素，必须指定两个下标。第一个下标表示该元素所在的行，而第二个下标则表示该元素所在的列

存储方式

转换为一维方式存储在内存

a[0][0] a[0][1] a[0][2] a[0][3]

a[1][0] a[1][1] a[1][2] a[1][3]

a[2][0] a[2][1] a[2][2] a[2][3]

从低维度（即后面的维度）开始排列

数组的首地址为第一个元素，即a[0][0]的地址

分配空间为L×N1×N2，L为数据的字长，N1为第一维的大小，N2为第二维的大小

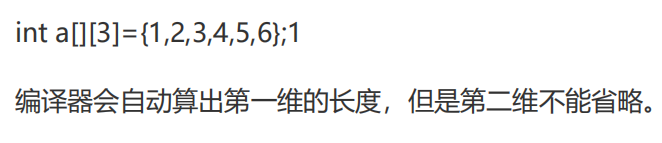
二维数组的初始化

嵌套一维数组

int a[3][4]={{1,3,5,7},{2,4,6,8},{3,5,7,11}};

int a[3][4]={{1,3,5,7},{2},{3,5}};

a[1][1]a[1][2]a[1][3]a[2][2]a[2][3]初始化为0



按数组元素存储顺序

int a[3][4]={1,3,5,7,2,4,6,8,3,5,7,11};

int a[3][4]={1,3,5,7,2,4,6,8,3};

a[3][1]a[3][2]a[3][3]初始化为0

二维数组元素的访问

访问二维数组的元素必须同时指明每一个维度的下标值

根据第一个下标得到该“行”元素的**首地址**

根据第二个下标得到该元素对于首地址的偏移量

例如，访问元素a[1][2]

找到a[1]的首地址（即a[1][0]的地址）

0x0012ff68

根据偏移量2，在首地址的基础上加2×4=8个字节（整型字长为4个字节），得到a[1][2]的地址

0x0012ff70

字符串可以存放在字符数组中，但该字符数组中必须存储一个显式的'\0'字符来作为字符串的结束标记

字符数组中， '\0'后的字符不属于该字符串

任一个字符数组不一定都是字符串，因为并不要求字符数组中必须存在'\0'字符!

如果字符数组中存在'\0' ，则'\0'及其前面的字符一起构成字符串

**对字符串进行整体处理的最常用系统函数（include<string.h>）**

strlen——求字符串长度，不包括\0

strlen(“abc”)的返回值为3

strcat ——连接字符串

strcat(str1,str2)将字符串str1和str2连接，将连接后得到的字符串作为函数的返回值

strcat(“abc”,”def”)的返回值为“abcdef”

Strcpy ——字符串拷贝，实现字符串的赋值

strcpy(str1,str2)将字符串str2赋值给str1

strcpy(str1,”abc”)将字符串”abc”赋值给字符串str1

对字符串进行整体处理的最常用系统函数

strcmp ——字符串比较

strcmp(str1,str2)

str1==str2：str1和str2的相同位置的每个字符都相同，而且长度相同。函数返回值为0

str1>str2:比较两个字符串第一个不相同的字符，str1的ASCII码大于str1的ASCII码。函数返回值为正整数

str1<str2:比较两个字符串第一个不相同的字符，str1的ASCII码小于str1的ASCII码。函数返回值为负整数

**说明字符串变量（include<string.h>）**

string str;

字符串变量的初始化

string str=“hello”;

字符串变量不包含\0

例如

char astr[10]=“hello”;

string str=astr;

字符串变量str中不包括字符数组astr中的\0

访问字符串中的字符

字符数组下标

例如

string word =“Then”;

word[2] = ‘a’;

字符串变量的值变为：“Than”

**结构类型的定义**

struct <结构类型名称> { <成员列表> };

类型名：标示符，可看作一种新的“数据类型”

成员表：<类型><成员1>;… <类型><成员1>;

成员的数据类型可以是该结构类型，也可以是其它结构类型

**结构类型变量的说明**

[struct] 结构类型名称 <变量列表>;

<结构类型定义><结构类型变量表>



**结构类型变量的初始化**

<结构类型名> <结构变量名> = <初始化列表>;

初始化列表形式为：

{成员1实例,成员2实例，……}

成员实例为与成员类型一致的数据



**结构类型成员的引用**

用运算符“.”对成员进行引用

雇员emp1的名字表示为：emp1.name

雇员emp1的电话表示为：emp1.phone

结构类型成员的赋值（保持类型一致性）

赋值语句： emp3.salary=21035.3;

输入语句: cin>>emp3.name;

相同结构类型的变量可以赋值

emp1 = emp2;

将数组说明为某种结构类型的数组

Employee emps[20];

数组的每个元素都是结构类型Employee

联合类型



**联合类型变量的初始化**

不允许进行初始化

可以随时对成员变量赋值

联合类型成员的引用

用运算符“.”对成员进行引用

学生stu1的年级表示为：stu1.grade

学生stu1的成绩表示为：stu1.score

**联合类型成员的赋值（保持类型一致性）**

赋值语句：stu2.grade =2009;

输入语句: cin>>stu3.score;

不能对联合变量名赋值

stu4 = stu3;//语法错误！

**函数的说明与使用**

一个函数既可以调用其它函数（包括该函数自身），亦可以被其它函数调用（包括该函数自身）

标准库函数

程序中可直接使用（调用）系统预定义的标准库函数，但要求在调用前使用编译预处理指令include将对应的头文件包含进来

用户自定义函数

由用户自定义的函数与系统预定义的标准库函数的不同点在于，自定义函数的函数名、参数个数、函数返回值类型以及函数所实现的功能等都完全由用户程序来规定（指定）

函数原型格式为：

[<属性说明>]<返回值类型><函数名>([<参数表>]);

属性说明：可缺省，一般可以是下面的关键字之一

inline：表示该函数为内联函数

static：表示该函数为静态函数

virtual：表示该函数为虚函数

friend：表示该函数为某类(class)的友元函数

返回值类型

函数处理得到的结果的数据类型

函数名

标识符

函数的说明和定义

两种说明方式的区别

函数原型的参数表中，参数名可以省略；函数定义的参数表中，必须给出参数名（省略参数名为无名参数）

函数原型的函数体，可以出现在函数调用之后；函数定义的函数体，必须出现在调用之前

函数原型的参数表后面加分号“;”,函数定义的参数表后面是函数体，即花括号“{”

函数定义不能出现在任何函数体中，函数原型可以出现在其它函数体中

函数的返回

函数的返回表示函数执行结束，将执行结果（无论是否有具体的数据）返回到调用函数的地方

函数返回时完成的任务

把运行控制从函数体返回到函数调用点

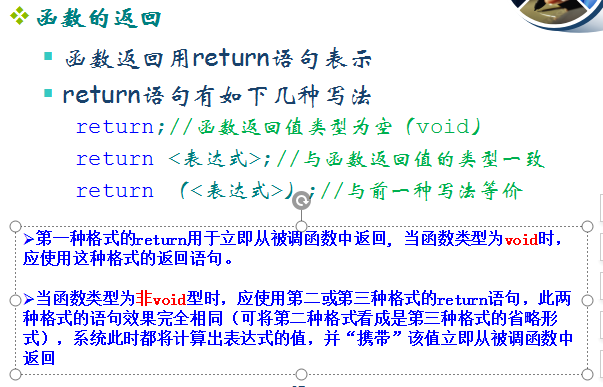
根据返回值要求，返回所需要的数据值

返回值类型

void

数值型

引用类型



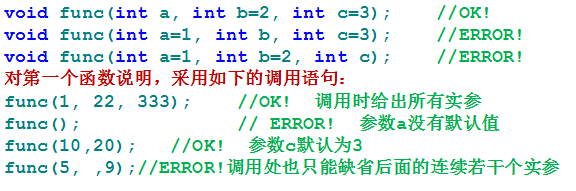
函数的参数

形参和实参

函数说明中的参数称为形式参数（形参），函数调用中的参数称为实际参数（实参）

实参表在参数个数、参数顺序、以及参数类型等方面要与被调函数的形参表之间有一个一一对应的相互匹配关系

编译器将根据参数的顺序，来逐一实现实参与对应形参的“结合”，而后执行一遍函数体（而完成本次的函数调用）



函数调用过程中参数的传递

一般传递过程（赋值传递）

发生函数调用，转到函数体执行

根据函数的形参，分配内存空间

将实参赋值给形参，即为形参分配的存储空间赋值，此时实参在函数体内失效，形参有效

函数执行完毕，返回到主调函数，形参所占的空间自动回收，形参失效

能够作为函数参数的数据类型

基本类型及其派生类型

赋值参数传递

数组、指针、引用等导出类型、

按地址传递参数

类类型、结构类型、联合类型等用户定义类型

赋值参数传递

按地址参数传递

一维数组以及多维数组的第一维大小，形参、实参可以不对应

实参为a[10]，形参可以定义为x[6]

数组作为函数参数，是将实参数组的首地址传递给形参，而不是将数组的所有元素传递给形参

在函数体中，根据形参数组首地址和下标指示的偏移量访问数组元素

能够表示地址的数据类型

数组

首地址（第一个元素的地址）

指针

引用

引用形参

函数定义的参数表中，名字前加上符号＆的参数为引用形参。例如void swap（int＆ a，int＆ b）；

引用形参在调用过程中的参数传递机制

函数的调用语句中对应于引用形参的实参必须是同一类型的变量，非变量的表达式则不允许。

参数传递的内容不是实参的值，而是地址，其实际的效果是令对应的引用形参在调用过程中，作为一个变量名指向作为实参的这个变量，在引用调用过程中并不创建新的参数变量。这一点有别于赋值调用

设计函数在下面两种情形时，建议采用引用参数

需要改变某些变量的值（上述函数swap 就是一例）

对于占内存较多的数据参数，为了不另建新的参数变量以节省内存

为了保证实参不在函数中被修改，可在形参说明中加上const 说明，例如：

complex add(const complex& a，const complex& b);

而对于赋值形参，则无此必要。

运算中的隐式数据类型转换（也称为数据类型的兼容）

整型、字符型、布尔型、枚举型之间的兼容性

整型和浮点型的兼容性

单精度浮点型与双精度浮点型的兼容性

程序改错：

错用分隔符

标识符未说明或者错误说明

错用运算符

if…else匹配问题

函数返回语句问题

数据类型不一致问题