第2章 基本数据类型与输入输出

- 2.1 字符集与保留字
- 2.2 基本数据类型
- 2.3 变量定义
- 2.4 字面量
- 2.5 常量
- 2.6 I/O流控制
- 2.7 printf与scanf

学习目标

- 熟悉基本数据类型
- 理解变量、常量的概念
- 掌握各种常量的性质和定义
- 学会I/O流的使用
- 了解printf和scanf输入输出的作用

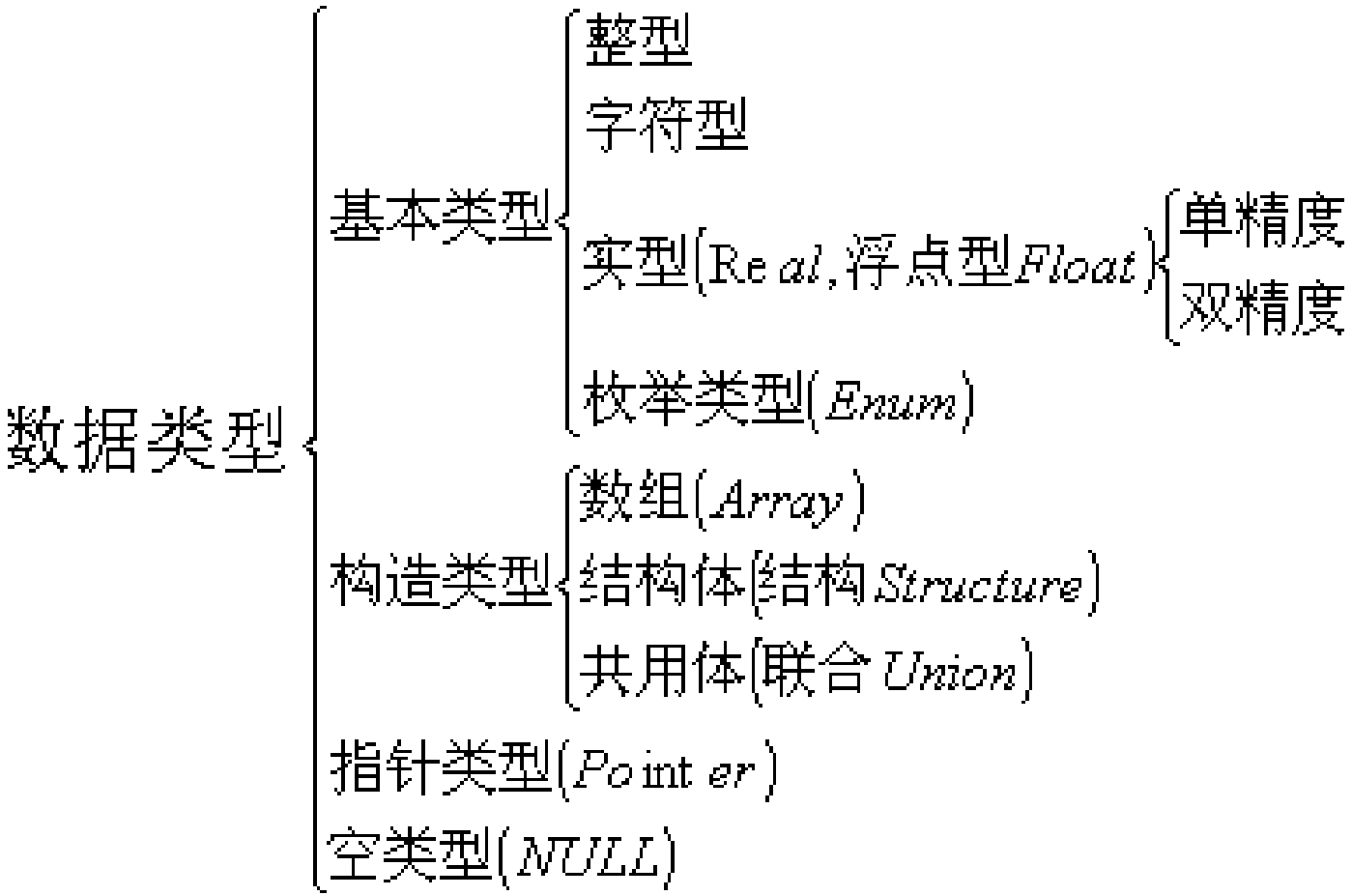
2.1 字符集与保留字

- 保留字 keyword
  - if, else, break, case, do , continue, goto, for, while,return,switch
  - auto, int, char, float, double,long, enum,sizeof, void,short,struct, unsigned,static….
- 表2.1
- 在程序中用到的其他名字不能与C/C++的关键字有相同的拼法和大小写

2.2 基本数据类型

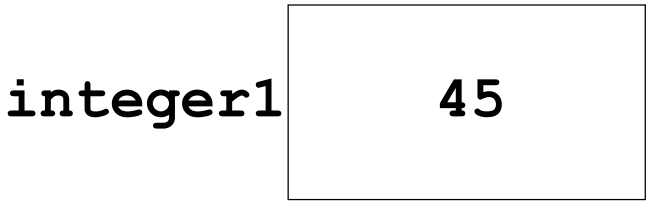
- 对程序当中所用到的所有数据都必须指定其数据类型。
- 程序中所用到（表达）的数据亦应有名字，或为变量或为常量，它们都对应某个内存空间。
- 数据类型的作用之一，是希望通过每个代表数据名字的性质来归类，不同数据类型之间不能进行混算，内部表达不同，空间占用不同，这都是编译器查错的重要依据。

C++的数据类型如下：


数据类型
基本类型
整型
字符型
实型(Real,浮点型Float)
单精度
双精度
枚举类型(Enum)
构造类型
数组(Array)
结构体(结构Structure)
共用体(联合Union)
指针类型(Point er)
空类型(NULL)

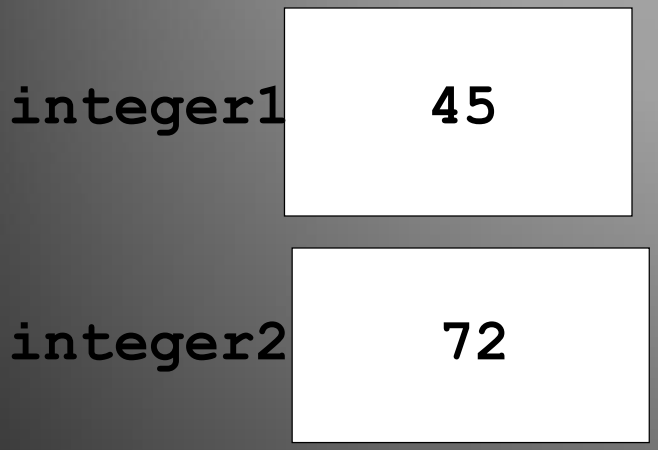
内存的概念

- 象以下的integer1, integer2和 sum这样的变量名实际上对应着计算机内存中的单元。
- 每个变量都有一个名字、一个数据类型和一个值。

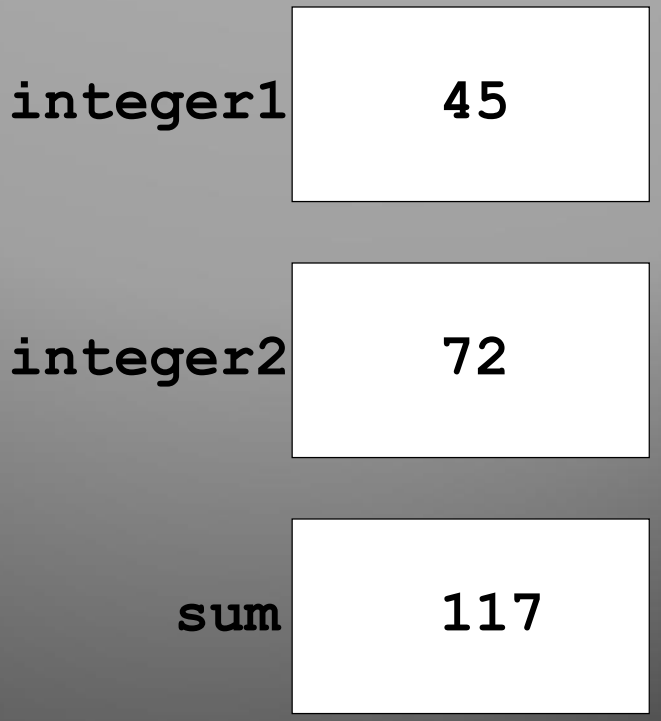


当一个值放在内存单元中时，这个值会取代内存单元中先前的值。

- 输入两个变量后的内存单元

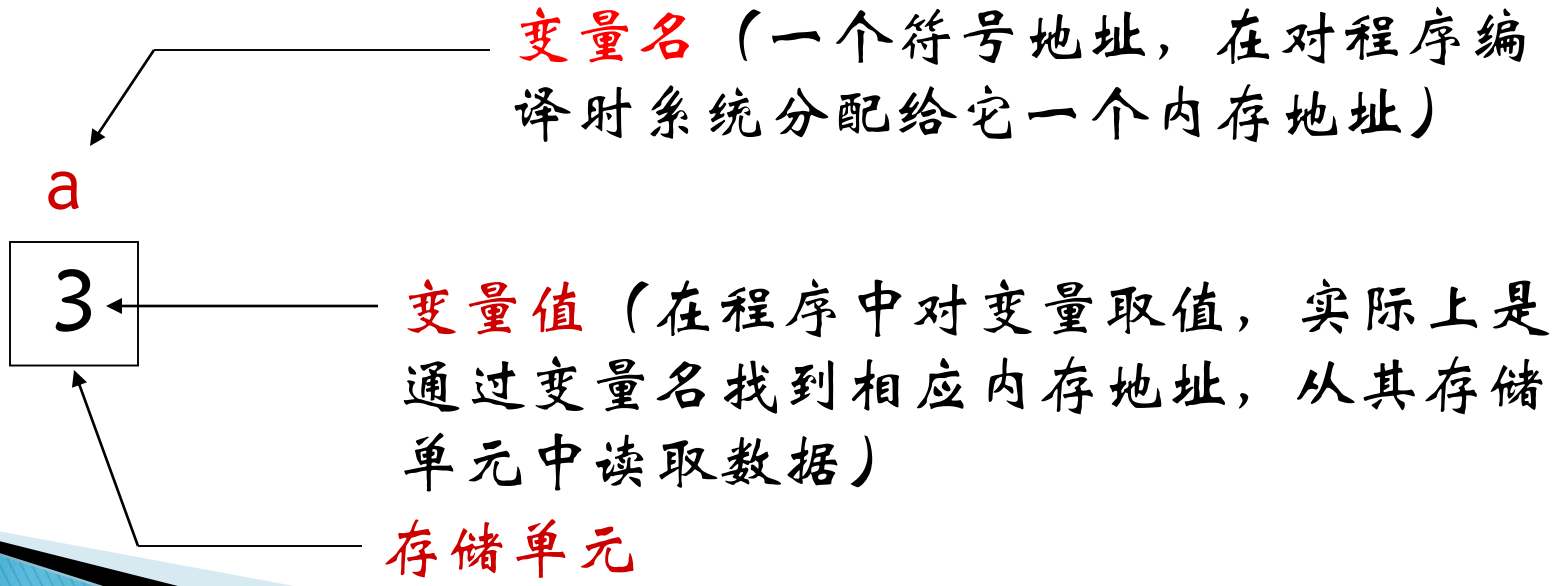


- 计算完成之后的内存单元



2.3 变量定义

- 在程序运行中其值可以改变的量称为变量。一个变量应该有一个名字，在内存中占据一定的存储单元。在该存储单元中存放该变量的值。
- 注意：变量名与变量值的区别。



2.3 变量定义

- 注意事项：
  - 变量是有表示范围的，使用时要避免数据溢出。
    - 例如：short int a=99999；//表示范围：-32768~32767
  - 实型变量不仅有表示范围，还有表示精度
    - 切记：不是所有的实数都能在计算机中精确表示。
    - 表示范围和表示精度的区别
  - 变量是有类型的
  - 布尔类型
    - C++中所有非零值都认为是逻辑真，只有0认为是逻辑假
  - 变量必须先定义，并且初始化才能使用。
    - 如果没有定义，产生编译错误
    - 如果没有初始化，不会产生编译错误，会产生不确定结果。

2.3 变量定义

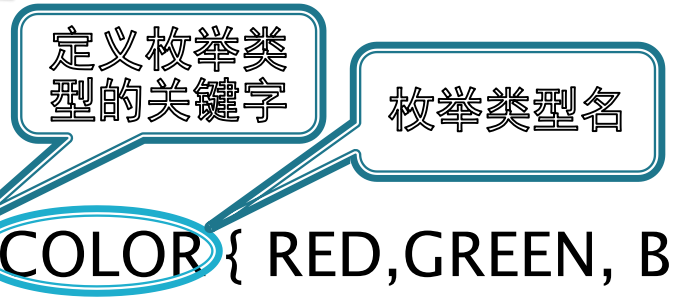
- 注意事项：
  - sizeof运算符
    - 计算变量类型在内存占有的字节数
    - 例如：sizeof(int)：4 //32位机
  - typedef
    - 为已有的类型名提供一个同义词，不定义新的类型，也不定义变量。
    - 例如： typedef int INT； INT a; //等价于 int a;
  - 字符变量
    - 内存占1个字节，保存字符的ASCII值。
    - 在表示范围内，与整数等价。
      - 例如： char c=96; // c表示字符'A'
        int a='A'; // a表示整数96

2.4 字面量

- 在程序中出现的，代表数据的文字。
  - 整型数：如12，0，－3，12L, 12l，12UL等（不带有小数）；
  - 实型数：如4.6，－1.23，1.23f, 1.23F等；
  - 字符：如 ‘a’， ‘d’等；
    - 转义字符： ‘\n’(换行), ‘\r’ (回车), ‘\\’(\), ‘\t’(tab)，等等
    - 单引号包含的单个字符
  - 字符串：如 “abc”
    - 双引号包含的一个或多个字符
    - 以字符串结束标记结尾: ‘\0’
  - 注意区分: ‘0’ 和0

2.4 字面量

- 枚举符
  - 一种数据类型
  - 例如：enum COLOR { RED,GREEN, BLUE};
  - 定义枚举类型变量
    - COLOR paint=RED; //
  - 注意：
    - 枚举变量的取值只能是枚举类型定义时规定的值。
      - 如paint只能取RED, GREEN, BLUE三个
    - 枚举常量的默认值从0开始，后面每一个递增1
      - 如：RED表示0， GREEN表示1， BLUE表示2
    - 可以规定枚举常量的值
      - 如：enum COLOR{RED=100,GREEN,BLUE=200,BLACK};
      - GREEN表示101， BLACK表示201



2.5 常量

- 在程序运行时保持不变的实体数据，用一个名字表示，该名字称为常量，在定义中加修饰const
- 常量在定义时必须初始化，常量名不能放在赋值语句的左边
  - 例如：**const int** a=123;
    a = 12; //error
- 另有一种使用伪指令#define的方法，C语言中使用，C++已经不用
  - 例如：#define a 123
- 优点：
  - 防止数据被意外修改。
  - 修改数据值时减少多处修改的麻烦。

常量的用途之一

- 常量的作用：

```
# include <iostream>
using namespace std;
int main( )
{
    float r, s, v;
    cin>>r;
    s=4.0*3.14*r*r ;
    v=4.0/3.0*3.14*r*r*r ;
    cout<<“s”<<s<<“v=\n“<<v;
     return 0;
}
```

```
s=4.0*3.14159*r*r ;
v=4.0/3.0*3.14159*r*r*r ;
```

常量的用途之一

- 常量的作用：

```
# define  PI  3.14159
# include <iostream>
using namespace std;
int main( )
{
    float r, s, v;
    cin>>r;
    s=4.0*PI*r*r ;
    v=4.0/3.0*PI*r*r*r ;
    cout<<s<<“  “<<v<<“\n";
     return 0;
}
```

```
# include <iostream>
int main( )
{
    float r, s, v;
const float pi=3.14159
    cin>>r;
    s=4.0*PI*r*r ;
    v=4.0/3.0*PI*r*r*r ;
    cout<<s<<“  “<<v<<“\n";
     return 0;
}
```

注意：

- 1、变量的定义必须放在执行语句之前；
  - 如果在执行语句中遇到一个变量，但是该变量还没有被定义，那么编译器会报语法错误。
  - 例如：int a =3;
  - c = a+1; // error c没有定义
- 2、每一个变量被指定为一确定数据类型，在编译时就能为其分配相应的存储单元；
  - 制定每一变量属于一个类型，这就便于在编译时，据此检查该变量所进行的运算是否合法。

变量名的命名

- C/C++语言中的变量名可以是任何有效的标识符。
- 标识符可以是由字母、数字和下划线(_)组成的一系列字符。
- 例如： integer1, integer2, sum
- 两种命名方式：
  - 骆驼命名方式：
    - int numOfStudent;
  - 匈牙利标记法
    - int iNumOfStudent;

命名规则

- 变量的命名符合一般标识符(名字)的命名规则 。

（1）变量为“字母数字串” ；

以字母开头，后边跟以字母或者数字，下划线等同于字母。（编程时不能用汉字作为名字，因C++语言以ASCII字符作为基本字符）

（2） 建议长度不超过8个字符（最早的C语言版本，只允许8个字符，现在C++可允许30个字符长，依可移植性要求）；

（3） 区分大小写 （一般使用小写字母）；

例如：int abc=3, Abc=7; //定义了不同的变量

（4） 尽量做到“见名知意”，避免使用代数符号（如a,b）；

例如：int length = 25, high=12;

（5） 不能有空格，不能有小数点。

2.6 I/O流控制

- 标准输入/出流
- 控制浮点数值显示
- 设置值的输出宽度
- 输出八进制和十六进制数
- 设置填充字符
- 左右对齐输出
- 强制显示小数点和符号

标准输入/出流

- 标准输入/出流： cin/cout
- 头文件包含：#include <iostream>
- 基本流状态控制符
  - 基于状态机制：
  - 不需要包含除iostream以外的其它头文件
- 流状态控制符

控制符	描述	控制符	描述
dec	十进制	setprecesion(n)	设置小数精度
hex	十六进制	setw(n)	设置宽度
oct	八进制	fixed	固定的浮点显示
setfill(c)	填充c	scientific	指数表示

标准输入/出流

- 举例

```
cout<<showpos<<12;                          // +12
cout<<hex<<18<< “ ” <<showbase<<18;         // 12  0x12
cout<<hex<<255<< “ ” <<uppercase<<255;      // ff  FF
cout<<123.0<< “  ” <<showpoint<<123.0;      // 123
  123.000
cout<<(2>3)<< “  ” <<boolalpha<<(2>3);      // 0  false
cout<<fixed<<12345.678;                     // 12345.678000
cout<<scientific<<123456.678;               // 1.234568e+05


int a;  cin>>a;                             //输入整数a
```

标准输入/出流

- 标准输入/出流
  - 宽度和填充字符的设置
    - 两种机制：使用<<流输出符 和 不使用<<流输出符

不使用<<流输出符	使用<<流输出符	功能
width(int)	setw(int)	设置显示宽度
fill(char)	setfill(char)	设置填充字符
precision(int)	setprecision(int)	设置有效位数或精度

    - 使用<<流输出符的机制下需要包含iomanip头文件，
  - width和fill为输出流类的成员函数，使用函数调用形式设置
  - 注意：
    - setw(n)是一次性的
    - 若要显示的内容超setw(n)中的n,则设置无效

标准输入/出流

- 标准输入/出流
  - 举例：

```
#include <iomanip>
….
cout<<setw(5)<<setfill(‘*’)<<12<<endl; //***12

cout<<setw(5)<<setfill('*')<<12<<";"<<34<<endl;
  // ***12；34
cout<<setw(4)<<12345<<endl;
 //12345
```

```
cout.width(5);
cout.fill('#');
cout<<12<<endl;
// ###12
```

浮点数显示

- 控制浮点数值显示

a) 普通格式：独立使用setprecision(n)表示有效位数n

例如：cout<<setprecision(3)<<12.2675;

显示：12.3

a) 定点表示格式：fixed与setprecision(n)配合，表示小数精度n位

例如：cout<<fixed<<setprecision(3)<<123567.89 <<"\n";

显示：123567.890

a) 科学表示格式：scientific与setprecision(n)配合，表示小数精度n位

例如：
cout<<scientific<<setprecision(4)<<12267.5<<"\n";

显示：1.2268e+05

2.7 printf与scanf(请各位教师自己补充)

- printf和scanf输出入格式是C的输入出方式，它输入出已有的C类型的数据。例如，int,double等

- printf函数
  f=format 在输出时，确定输出格式

- scanf函数
  f=format 将键盘的字符序列，按格式转为数据

多选题 1分

设置

下面描述正确的是( )

A C++中任何类型变量都只能存放一定范围的数据

B C++中任何数值都可以精确表示

C C++中字符型数据与整型数据等价

D 使用setprecision可以指定显示小数位数

提交