第2单元 C语言基础



摘要



- 2.1 数据类型
- 2.2 常量与变量
  - 2.2.1 常量和符号常量
  - 2.2.2 变量
- 2.3 整型数据
  - 2.3.1 整型常量
  - 2.3.2 整型变量
- 2.4 实型数据
  - 2.4.1 实型常量
  - 2.4.2 实型变量
- 2.5 字符型数据
  - 2.5.1 字符常量
  - 2.5.2 字符型变量
  - 2.5.3 字符串常量
- 2.6 运算符与表达式
  - 2.6.1 算术运算符和算术表达式
  - 2.6.2 赋值运算符和赋值表达式
  - 2.6.3 数据类型转换
  - 2.6.4 几个特殊的运算符
- 实例分析与实现

2.1 数据类型

数据类型



- 学生信息管理系统



数据：

姓名
性别
年龄
籍贯
成绩
…

- 超市信息管理系统



数据：

商品货号
名称
价格
数量
供货商
…



现实中有哪些类型的数据？

不同类型的数据可以进行哪些运算？

2.1 数据类型



- C数据类型
  - 基本数据类型
    - 整型 int
      - 短整型 short
      - 整型 int
      - 长整型 long
    - 字符型 char
    - 实型（浮点型）
      - 单精度实型 float
      - 双精度实型 double
  - 构造类型
    - 数组
    - 结构体 struct
    - 共用体 union
    - 枚举型 enum
  - 指针类型
  - 空类型 void

2.2 常量与变量

标识符与关键字



- 常量：在程序运行过程中，其值不会发生变化的量

  例如：整数123、字符‘a’
- 变量：在程序运行过程中，其值可以发生变化的量

标识符与关键字



给变量所取的名字叫变量名。

定义变量的名字需要遵循标识符的命名规则。

标识符是用来标识变量、符号常量、数组、函数、文件等名字的有效字符序列。

标识符的命名规则：



- 只能由字母、数字和下划线组成 （例如：Score1，Stu_Name）
- 第一个字符必须为字母或下划线 （例如：_sum，p1）
- 区分大小写字母 （例如：sum，SUM，Sum不同）
- C语言内部规定的标识符（即关键字）不能作为用户标识符

标识符与关键字



C语言中的关键字

auto	break	case	char	const	continue
default	do	double	else	enum	exern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

2.2.1 常量和符号常量

2.2.1 常量和符号常量



C语言提供的常量有：整型常量、实型常量、字符常量、字符串常量和符号常量。

用一个标识符来代表一个常量，称为符号常量。只符叫作符号常量。

一般用大写字母表示，以便与其他标志相区别

其一般形式为：

#define 标识符

命名尽量“见名知义”
例如： #define PI 3.14159

例如：

```
#define NUM 0                    //定义符号常量NUM
```

2.2.1 常量和符号常量



【例2.1】使用符号常量计算半径为5的圆形周长和面积。

```
#include <stdio.h>
#define PI 3.14
int main()
{
        printf("周长是：%f\n",2*PI*5);
        printf("面积是：%f\n",PI*5*5);
        return 0;
}
```

```
周长是：31.400000
面积是：78.500000
```

2.2.1 常量和符号常量



使用符号常量要

- 一个#define命

例如：
```
#define NUM 10
main()
{
    NUM=50;    //编译时会提示错误，符号常量NUM的值不能再被修改
}
```

- 用一行书写，不用分号结尾

- 符号常量一旦定义，就可以在程序中代替常量使用。由于是常量，所以在其作用域内不能再被修改或赋值。

2.2.2 变量

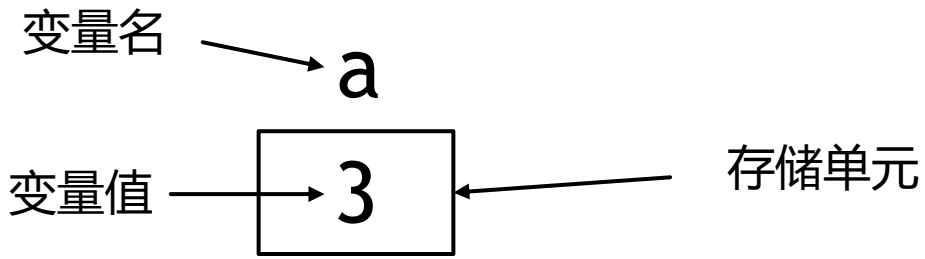
变量的定义与使用



变量在内存中占用一定的存储单元，在该存储单元中存放变量的值。

程序中用到的所有变量都必须有一个变量名。

变量名和变量值是两个不同的概念。



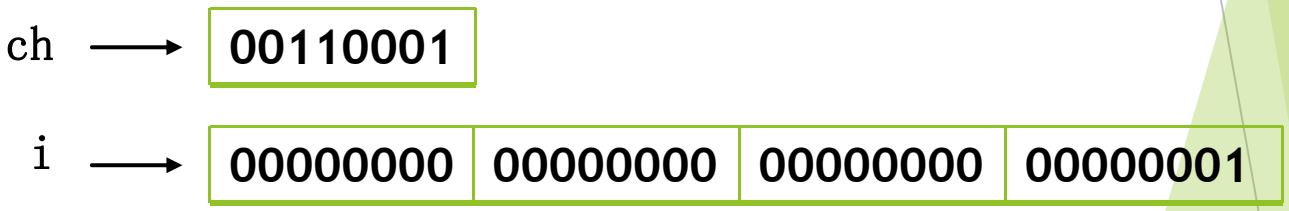
变量的定义与使用



不同类型的变量在内存中占据存储单元的数量及存储的格式不相同。

例如：char ch= ‘1’；

int i=1;



变量的定义与使用



变量必须"先定义后使用"。这样做的目的是:

1.便于编译程序检查对该变量的运

例如，整型变量可以进行求模（余
以进行求模运算

2.保证程序中变量名的正确使用

例如:

```
#include<stdio.h>
int main()
{
    int a=5,b=6,sum=0;
    svm=a+b;
    return 0;
}
```

程序第5行错将sum写为svm，编译时，系统会报告svm没定义，并给出错误提示“undeclared identifier”。

2.3 整型数据

2.3.1 整型常量

2.3.1 整型常量



整型常量就是整数。有3种表示形式：

①十进制整常数：就是通常整数的写法，数码取值为0~9。

例如：25、-623、0等。



②八进制整数：以0作为前缀，数码取值为0~7。

几个合法的八进制数：

020（十进制数16）　　016（十进制数14）

0177777（十进制数65535）　　-010（十进制数-8）

按权展开，逐项相加

020

=0*8^0 + 2*8^1

=0 +16

=16

几个不合法的八进制数：

256（无前缀0）　　079（包含了非八进制数码9）

2.3.1 整型常量



3.十六进制整常数：以数字0和字母X（x）开头，即0X或0x作为前缀，数码取值为数字0~9和字母A~F（或字母a~f）。

几个合法的十六进制数：

0X2A（十进制数42）　　0x1AB0（十进制数6832）
0XFFFF（十进制数65535）

几个不合法的十六进制数：

5A（无前缀）　　0X2H（包含了非十六进制数码H）

2.3 整型数据

2.3.2 整型变量

2.3.2 整型变量



1.整型变量的分类

整型变量分为：基本整型、长整型、短整型、无符号型

类型名	类型标识符	字节数	取值的范围
短整型	short[int]	2	-2^{15} ~ $2^{15}-1$（-32768 ~ 32767）
无符号短整型	UShort[int]	2	0 ~ 0xffff （0 ~ 65535）
基本整型	int	4	-2^{31} ~ $2^{31}-1$（-2147483648 ~ 2147483647）
无符号整型	Uint	4	0 ~ 0xffffffff （0 ~ 4294967295）
长整型	long[int]	4	-2^{31} ~ $2^{31}-1$（-2147483648 ~ 2147483647）
无符号长整型	Ulong	4	0 ~ 0xffffffffUL（0 ~ 4294967295）

2.3.2 整型变量



2.整型变量的定义及初始化

定义变量及初始化的一般形式为：

类型说明符　变量名1[=值1] [,变量名2[=值2],……];

例如：

```
int a,b,c;                          /* 定义a，b，c为整型变量*/
long x=10;                          /*定义x为长整型变量，并赋初值10*/
unsigned p=2,q=5，m;/*定义p，q，m为无符号整型变量，为p
                                    赋初值2，为q赋初值5*/
```

2.3.2 整型变量



例如:
int p=2,q=5,m;

注意:

①允许在一个类型说明符后，定义多个相同类型变量。类型说明符与变量名之间至少用一个空格间隔，各变量名之间用逗号间隔。

②最后一个变量名后面用“；”号结尾。

③变量定义必须放在变量使用之前，一般放在函数体开头部分。

④没有给变量赋初值，并不意味
没有确定的值，因此直接使用这
有可能导致运算错误。

例如:
int a,b,sum;
sum=a+b;（结果毫无意义）

2.3.2 整型变量



【例2.2】整型变量的定义与使用

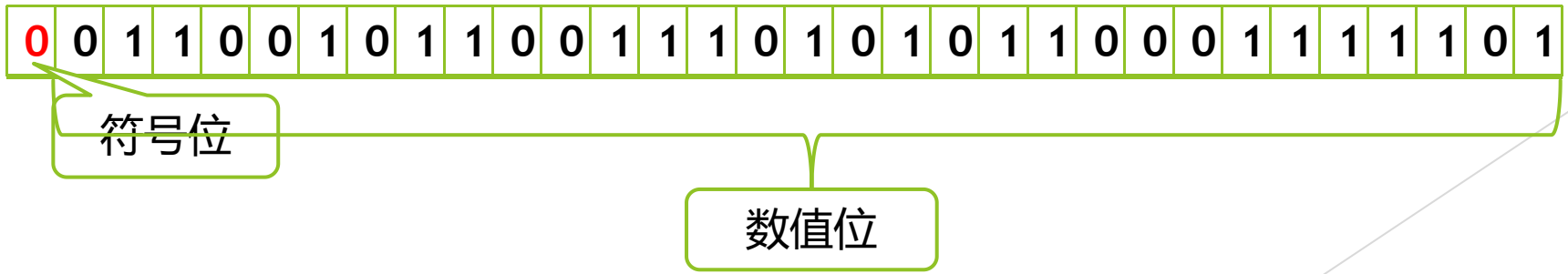
```
#include<stdio.h>
main()
{       int a=3,b=5,c,d
        c=a+b;
        d=a-b;
        printf("a+b=%d,a-b=%d\n",c,d);  //输出a与b的和与差
}
```

```
a+b=8, a-b=-2
```

原码、反码和补码

在计算机中数分为有符号数和无符号数。它们是以二进制形式表示的，有三种表示方法，即原码、反码和补码。

三种表示方法均由符号位和数值位两部分构成。符号位在最高位，用0表示“正”，用1表示“负”其余位是数值位。



原码、反码和补码

unsigned int i=132;



int i=132;



int i=-132;



原码、反码和补码



原码：按照二进制的方法来表示数的绝对值。用这样的表示方法得到的就是数的原码。

int i=132;

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1

int i=-132;

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1

原码、反码和补码

反码：正数的反码与其原码相同，负数的反码为其原码除符号位以外的各位按位取反。

int i=132;



int i=-132;



原码、反码和补码



补码：正数的补码与其原码相同，负数的补码是将其原码除符号位之外的各位求反之后，在末位再加1。

int i=132;

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1

int i=-132;

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1

原码、反码和补码



正数的原码、反码、补码相同。
负数的原码、反码、补码不同。
数值在计算机中以补码形式存放。

127的原码 ⟺ 取反加1 ⟺ -127的补码

原码、反码和补码

例如:求-127的补码:

第一步:确定符号位1，数据位是127的原码



第二步:除符号位外,各位取反



第三步:加1



例如：



问题：该二进制数是什么数的补码?

方法：除符号位外各位取反、加1、加上负号



除符号位外各位取反



加1



加上负号

-1

2.4 实型数据

2.4.1 实型常量

实型常量



实型数据又称实数或浮点数，指的是带有小数部分的非整数数值。

在C语言中，实数只用十进制数表示，有两种表示方法：

（1）小数形式：由数字0～9和小数点组成，当小数部分为零时，小数点不能省略。

例如：0.0、25.4、.12、22.、-56.33等均为合法的实型常量。

实型常量



（2）指数形式：由十进制数、加阶码标志小写字母“e”（或大写字母“E”）和阶码（必须是整数）组成。

例如：2.3E5（等于2.3×10^{5}）

5.6e-3（等于5.6×10^{-3}）

-1.8E-2（等于-1.8×10^{-2}）

以下不是合法的实型常量：

123（无小数点）

E4（阶码“E”之前无数字）

23.-E3（负号的位置不对）

2.7E（无阶码）

2.4 实型数据

2.4.2 实型变量

实型变量



1.实型变量可分为：单精度实型（float类型）和双精度实型（double类型）

类型标识符	所占字节数	有效位数	数值范围
float	4	6～7	-3.40E+38～ +3.40E+38
double	8	15～16	-1.79E+308～ +1.79E+308

实型变量



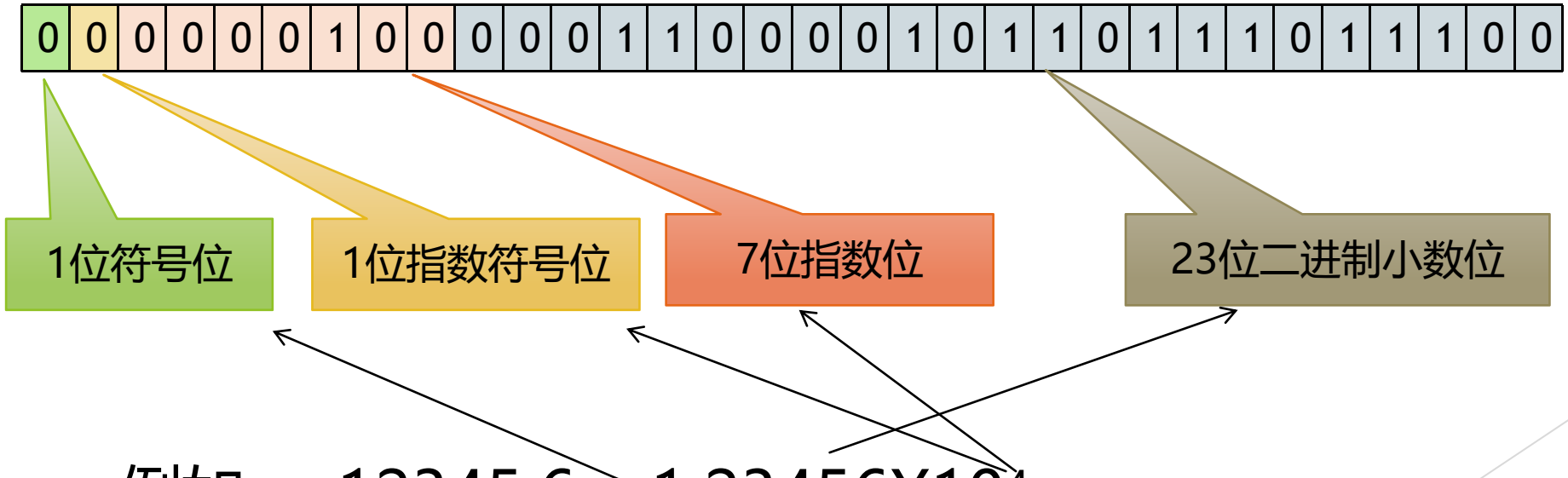

```
#include <stdio.h>
main()
{
        float a=1.23456789123456789;
        double b=1.23456789123456789;
        printf("%.20f\n%.20f\n",a,b);
 }
```

```
1.23456788063049320000
1.23456789123456790000
```

2.4.2 实型变量



2.实型数据在内存中的存放形式：以浮点数形式存放的。分为符号位、小数部分、指数符号位和指数部分。



例如：-12345.6=-1.23456X10^4

2.4.2 实型变量



3.实型变量的定义及初始化

定义变量及初始化的一般形式为：

类型说明符　变量名1[=值1] [,变量名2[=值2],……];

例如：

```
float a,b,c;                    /*定义a,b,c为单精度实型变量*/
double m=3.5,n=5.6;   /*定义m、n为双精度实型变量，并分别赋
                                初值3.5和5.6*/
```

2.4.2 实型变量



4.实型数据的舍入误差

【例2.3】实型数据的舍入误差

```
#include <stdio.h>
main()
{
        float a=1.234567E10,b;
        b=a+20;
        printf("a=%f\n",a);
        printf("b=%f\n",b);
}
```

1.要避免一个大数和一个小数直接相加、减。
2.可以将float类型改为double类型

运行程序，输出结果为：

```
a=12345669632.000000
b=12345669632.000000
```

2.5 字符型数据

2.5.1 字符常量

字符型常量



字符常量是用一对单引号括起来的一个字符。
例如：'b'、'Z'、'='、'？'、'5'。

字符常量的特点：

1.字符常量只能是单个字符，不能是多个字符。

2.字符常量是以ASCII码的形式存储的。一个字符常量占用一个字节空

美国标准信息交换码
American Standard Code for Information Interchange

字符型常量



3.ASCII编码能表示数据范围：

00000000~11111111 共2^8即256个数据。
例如：

字符	ASCII码值	二进制存储方式
‘A’	65	01000001
‘a’	97	01100001
‘0’	48	00011000

字符型常量



标准ASCII码有128个字符，其中：

（1）码值0～31为控制字符。例如，回车换行、文件结束标志、字符串结束标志等。

（2）10个阿拉伯数字‘0’～‘9’的码值是连续的。

（3）26个大写字母’ A ’ ～ ’ Z ’是连续的，26个小写字母’ a ’ ～ ’ z ’是连续的。

字符型常量



字符常量可以参与各种运算。

例如：

‘B’-‘A’值为1（字符‘B’的ASCII码值66减去字符‘A’的ASCII码值65结果为1）。

‘b’-32值为66（字符‘b’的ASCII码值98减去32等于65，是字符‘B’的ASCII码值）

‘c’<‘d’的值为true（字符‘c’的ASCII码小于字符‘d’的ASCII码）。

转义字符



转义字符：是由一个反斜杠“\”后跟规定字符构成，具有特定含义的字符。

常用转义字符

转义字符	含义	ASCII码
\n	回车换行	10
\t	水平制表（跳到下一个tab位置）	9
\b	退格(Backspace)	8
\0	空字符	0
\\	反斜杠字符	92
\’	单引号字符	39
\”	双引号字符	34
\ddd	1～3位八进制数据所代表的字符	
\xhh	1～2位十六进制数据所代表的字符	

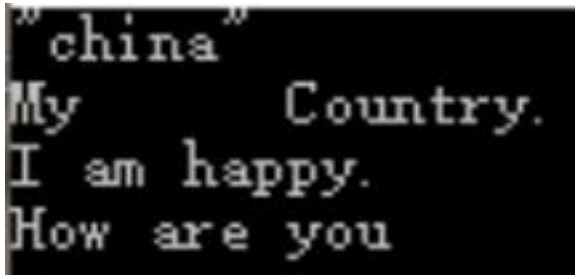
转义字符



【例2.4】转义字符的使用

```
#include <stdio.h>
int main()
{       printf("\"china\"\n");
        printf("My\tCountry.\n");
        printf("I am hap\160\x79.\n");
        printf("Hay a\b\b\b\bow are you\n");
        return 0;
 }
```

运行程序，输出结果为：



2.5 字符型数据

2.5.2 字符型变量

字符变量



字符变量是用来存放单个字符的变量，也包括转义字符。字符型变量的类型说明符是char。

定义变量及初始化的一般形式为：

char 变量名1[=值1] [,变量名2[=值2],……];

例如：

```
char ch1;                    /*定义ch1为字符型变量*/
char ch1，ch2='a';           /*定义ch1、ch2为字符型变量， 并
                               给ch2赋初值'a'*/
unsigned char ch3;           /*定义ch3为无符号的字符型变量*/
```

2.5.2 字符变量



说明：

- 字符型变量在内存中占用1个字节,只能存放一个字符，并以该字符的ASCII码的值的形式存放。

- 在C语言中，一个字符型数据可以按照它的ASCII码整数值进行算数运算。

2.5.2 字符变量



【例2.5】给字符变量赋整数值

```
#include <stdio.h>
main()
{
        char ch1=65,ch2=66;
        printf("%c,%c\n",ch1,ch2);
        printf("%d,%d\n",ch1,ch2);
}
```

运行程序，输出结果为：

```
A, B
65, 66
```

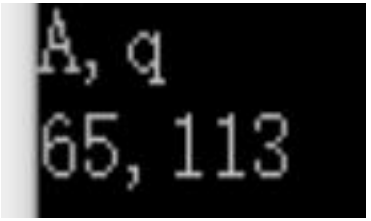
2.5.2 字符变量



【例2.6】大小写字母的转换

```
#include <stdio.h>
#define PI 3.14
main()
{
        char ch1,ch2;
        ch1='a';
        ch2='Q';
        ch1=ch1-32;
        ch2=ch2+32;
        printf("%c,%c\n%d,%d\n",ch1,ch2,ch1,ch2);
 }
```

程序运行结果：



字符变量赋值给整型变量



【例2.7】字符型变量赋值给整型变量

```
#include <stdio.h>
main()
{
        char x='\256';
        int y;
        y=x;
        printf("y=%d\n",y);
 }
```

X的二进制数据存放：



y的二进制数据存放：



运行程序结果为：

```
y=-82
```

字符变量赋值给整型变量



【例2.8】无符号字符型变量赋值给整型变量

```
#include <stdio.h>
main()
{
        unsigned char x='\256';
        int y;
        y=x;
        printf("y=%d\n",y);
 }
```

X的二进制数据存放：

1	0	1	0	1	1	1	0

y的二进制数据存放：

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0

运行程序结果：

```
y=174
```

2.5 字符型数据

2.5.3 字符串常量

2.5.3 字符串常量



字符串常量简称字符串，是用一对双撇号（西文双引号）括起来的一串字符，包含字符的个数称为字符串的长度。

例如"Hello world"、"a"、"C语言"都是字符串常量。

2.5.3 字符串常量



字符串常量和字符常量的区别：

（1）字符常量是由单引号括起来的字符，而字符串常量是由双引号括起来的字符。尽管'a'与"a"都含有一个字符，但在C程序中，它们具有不同的含义。

（2）字符常量只能是一个字符，字符串常量可以是多个字符。

2.5.3 字符串常量



字符串常量和字符常量的区别：

（3）可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。

2.5.3 字符串常量



（4）字符常量占1个字节内存空间，字符串常量所占内存的字节数等于其字符的个数加1。在字符串结尾，计算机自动加上字符'\0'，表示该字符串的结束。因此，字符串常量的存储单元要比实际的字符个数多一个。

例如：

- ✓ 字符串常量"a"在内存中占2个字节，分别存储'a'和'\0'。
- ✓ 字符串"This is a computer"字符数为18，但占用19个字节,最后一个字节存放'\0'。

2.6 运算符与表达式

2.6.1 算术运算符和算术表达式

运算符与表达式



运算符：用来表示各种运算的符号。

按运算对象个数可分为：单目运算符、双目运算符、三目运算符

运算符还有运算方向的规定，即结合性。
自左向右进行运算的结合方向是左结合
自右向左进行运算的结合方向为右结合

算术运算符和算数表达式



1.算术运算符

加（+）、减（-）、乘（*）、除（/）、求模（%）

【例2.9】除法运算符的应用举例

```
#include <stdio.h>
main()
{       printf("%d,%d\n",10/3,-10/3);
        printf("%f,%f,%f,%f",10.0/3,10.0/-3,-10.0/3,-10.0/-3);
}
```

运行程序，结果为：

```
3, -3
3.333333, -3.333333, -3.333333, 3.333333
```

算术运算符和算数表达式



【例2.10】求余数运算符的应用举例

```
#include <stdio.h>
main()
{
        printf("%d\n",7%3);
        printf("%d,%d,%d\n",(-7)%3,7%(-3),(-7)%(-3));
}
```

运行程序，结果为：

```
1
-1, 1, -1
```

算术运算符和算数表达式



2.算术表达式

用算术运算符和一对圆括号将操作数（常数、变量、函数等）连接起来，符合C语言语法的表达式称为算术表达式。

数学表达式	错误的程序表达式	正确的程序表达式
b^2-4ac	b*b-4ac	b*b-4*a*c
$\frac{b^2-4ac}{2a}$	(b*b-4*a*c) /2*a	(b*b-4*a*c) /(2*a)
$\frac{a+b}{a-b}$	a+b/a-b	(a+b)/(a-b)

2.6.1 算术运算符和算数表达式



注意：

- C语言中不允许使用方括号和花括号，只能使用圆括号。
- 圆括号是C语言中优先级别最高的运算符。
- 圆括号必须成对使用。当使用了多层圆括号时，先完成最里层的运算，最后处理最外层括号。
- 诸如求绝对值和平方根这类数学运算，C语言已经将它们定义成标准库函数，存放在数学库头文件“math.h”中，在使用时，用户只需要直接使用即可。

2.6.1 算术运算符和算数表达式



注意：

- 五种算术运算符中，单目运算符“-”的优先级最高，其次是乘“*”、除“/”和求模“%”三种运算符，最后是加“+”、减“-”运算。

- 算术运算符具有左结合性，即先左后右。

2.6 运算符与表达式

2.6.2 赋值运算符和赋值表达式

2.6.2 赋值运算符和赋值表达式



赋值号“=”称为赋值运算符。

由赋值运算符组成的表达式称为赋值表达式。

赋值表达式形式如下：

变量名 = 表达式

- 赋值号左边必须是一个变量名，赋值号右边允许是常量、变量和表达式。
- 赋值运算符的功能是首先求出右边表达式的值，然后将此值赋给左边变量。

2.6.2 赋值运算符和赋值表达式



赋值运算符的特点：

(1)赋值运算符的优先级别很低，在所有的运算符中，仅高于逗号运算符，低于其他所有运算符。

(2)赋值运算符不同于数学中的等号，等号没有方向，而赋值号具有方向性。

例如a=b和b=a在数学上意义是等价的，但是C语言中，执行a=b操作后，变量a的值被改写成了变量b的值。

2.6.2 赋值运算符和赋值表达式



赋值运算符的特点：

(3) C语言规定，赋值表达式左边变量得到的值作为赋值表达式的值。所以表达式a=5的值等于5。

(4)赋值运算符具有右结合性，因此a=b=5也是合法的，与a=(b=5)等价，最后a和b的值均等于5

2.6.2 赋值运算符和赋值表达式



注意：定义变量时不能使用连等的形式

例如：

```
int i=5;          //相当于：int i；i=5；
int i，j=5;       //相当于：int i,j；j=5；
int i=j=5;        //错误
```

2.6 运算符与表达式

2.6.3 自动类型转换与强制类型转换

数据类型转换



赋值号两侧类型不同时，右边自动转换为左边类型

赋值运算转换规则：

1.将实型数据赋给整型变量时，舍去实数的小数部分。

2.将整型数据赋给实型变量时，数值不变，以浮点形式存放，即增加小数部分（小数部分的值为0）。

数据类型转换



例如：

```
#include <stdio.h>
int main()
{       int a=98.123;
        char b=a;
        float c=b;
        printf("a=%d,b=%c,c=%f\n",a,b,c);
        return 0;
 }
```

```
a=98, b=b, c=98.000000
```

数据类型转换



3.将字符型数据赋值给整型变量时，需要进行数据的扩展填充。

将字符的ASCII码值复制到整型变量的低八位中，对于无符号字符型数据，整形变量高位补0；对于有符号字符型数据，整形变量高位补字符的最高位（0或1）。

数据类型转换

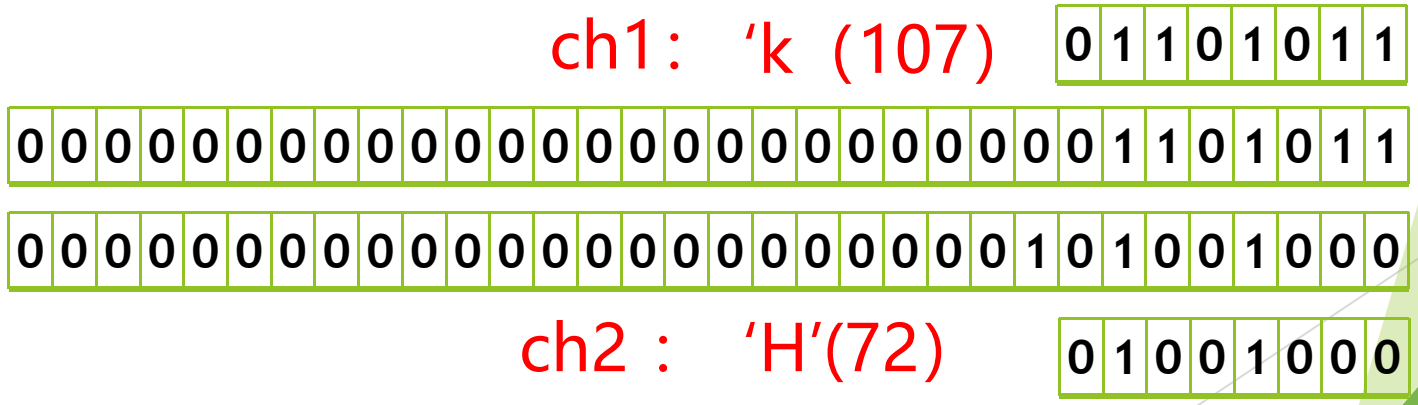
4.将整型数据赋给字符变量时，只把其低八位赋给字符变量。

例如：int a,b=328,c;

char ch1='k',ch2;

c=ch1;

ch2=b;



数据类型转换



5.整型赋值给短整型

```
#include <stdio.h>
int main()
{
        int a=0xaaaaaaaa;
        short b=a;
        printf("%d",b);
 }
```

```
-21846
```

a的二进制数据

1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

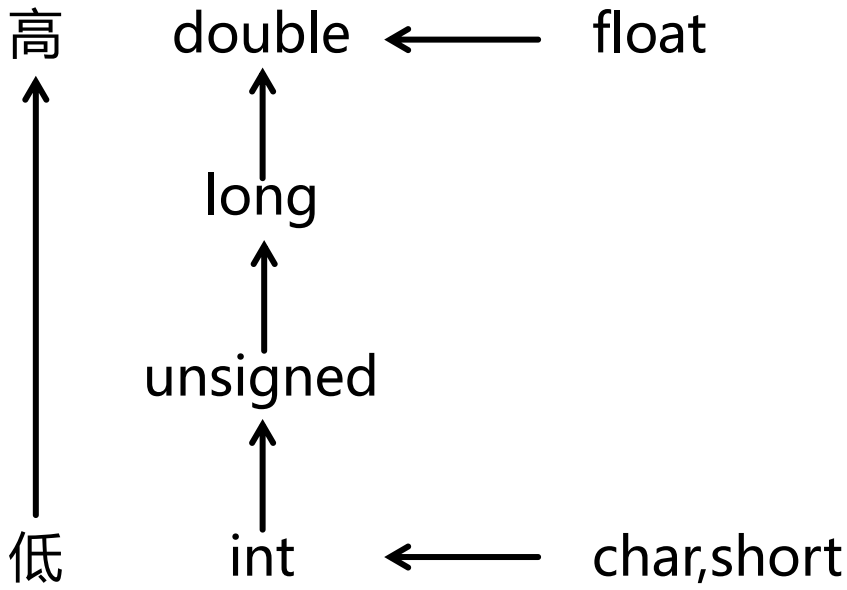
b的二进制数据

1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0

自动类型转换



- 当运算符两边的运算对象类型相同时，可以直接进行运算，并且运算结果和运算对象具有同一数据类型。
- 当运算符两边的运算对象类型不同时，系统自动地将操作数转换成同种类型。计算结果的数据类型为级别较高的类型。



自动类型转换



【例2.12】数据类型自动转换举例

```
#include <stdio.h>
main()
{
        float x=5.0,y;
        int a=6;
        char c='B';
        y=7+a*c+x;
        printf("y=%f",y);
 }
```

运行程序，结果为：

```
y=408.000000
```

强制类型转换



C语言中允许使用类型说明符关键字对操作数据进行强制类型转换，强制类型转换表达式的形式如下：

（类型说明符）（表达式）

例如：

(int)7.678=?

(double)（10%3）=?

int a=3，b=4

（float）a/(float)b=?

（float）（a/b）=?

强制类型转换



注意：

1.在强制类型转换时，取整类型转换不是按照四舍五入原则处理的。

例如：float a=2.8则（int）a的值为2。

2.强制类型转换并不改变变量原有的数据类型和变量的值。

上式中，虽然表达式（int）a的值为2,变量 a的类型仍然是float，其值仍然是2.8。

强制类型转换



3.强制类型转换运算符和表达式都必须加括号（单个变量可以不加括号）。

【例2.13】强制类型转换符的使用

```
#include <stdio.h>
main()
{       float x=45.2353;
        x=(int)(x*100+0.5)/100.0;
        printf("x=%.2f\n",x);
 }
```

运程程序，结果为：

```
x=45.24
```

2.6 运算符与表达式

2.6.4 几个特殊的运算符

2.6.4 几个特殊的运算符



1.复合赋值运算符

在赋值运算符之前加上其他运算符，可以构成复合赋值运算符。

常用的复合赋值运算符：+=、-=、*=、/=、%=

a+=x　相当于 a=a+x

a-= y　相当于 a=a-(x-y)

a*=x-y 相当于 a=a*(x-y)

a/=x*y 相当于 a=a/(x*y)

a%=y　相当于 a=a%y

2.6.4 几个特殊的运算符



2.自增运算符（++）和自减运算符（--）

二者都是单目运算符，运算对象必须是变量，不能为表达式或常量

- ++i，--i变量在使用之前先自增1，自减1
- i++，i-- 变量在使用之后再自增1，自减1

2.6.4 几个特殊的运算符



【例2.15】自增、自减运算符的使用

```
#include <stdio.h>
main()
{       int x=3,y=5,a,b;
        a=x++;
        b=++y;
        printf("x=%d,y=%d,a=%d,b=%d\n",x,y,a,b);
}
```

运行程序，结果为：

```
x=4, y=6, a=3, b=6
```

2.6.4 几个特殊的运算符



3.逗号运算符

逗号运算符“，”将多个表达式连接起来，组成逗号表达式。逗号表达式的一般形式为：

表达式1，表达式2，...,表达式n

例如：i=3，i++，++i，i+5；

这个逗号表达式的值10，i的值为5。

2.6.4 几个特殊的运算符



4.条件运算符

C语言中仅有一个三目运算符，叫条件运算符。条件运算符需要三个操作数，它能够实现简单的选择功能，条件表达式的形式是：

表达式1？表达式2：表达式3

由三目运算符组成的表达式称为条件表达式。

2.6.4 几个特殊的运算符



【例2.16】输出两个数中较小的数。

```
#include <stdio.h>
int main()
{
        int a,b,min;
        a=1,b=6;
        min=a<b?a:b;
        printf("min=%d\n",min);
        return 0;
 }
```

```
min=1
```

2.6.4 几个特殊的运算符



5.求字节运算符

求字节运算符是一元运算符，用来返回其后的类型说明符或表达式所表示的数在内存中所占有的字节数。

求字节运算符的使用形式如下：

sizeof(e)

其中，e可以是任意类型的变量、类型名或表达式。

2.6.4 几个特殊的运算符



【例2.17】字节运算符的使用举例。

```
#include <stdio.h>
int main()
{
        int a;
        printf("%d,%d,%d,%d\n",sizeof(a),sizeof(4+5),sizeof(double),sizeof("string"));
        return 0;
 }
```

```
4, 4, 8, 7
```

运行程序，结果：



实例分析与实现

实例1



假设某顾客买了3种商品，已知各商品的单价，现要求编写程序，用于实现当输入各种商品的购买数量后，输出顾客的应付价格总额。

实例2



有如下定义：

```
float b=3.4;
double c=10.0;
long int d=5;
int a=6;
```

分析表达式“('a'+'b')*2+a*b-c/d”结果的数据类型，并计算表达式的值。

【想一想】

（1）怎样判断一个数字型的变量可以容纳的最大值?

（2）浮点数比较数据大小可靠吗?



The chapter is over!