5 简单数据描述（II）—数据类型转换

郭延文

2019级计算机科学与技术系

基本类型的转换

- 程序执行过程中，要求参加双目操作的两个操作数类型相同；当类型不同时，会进行类型转换，即一种操作数的类型会转换成另一种数据类型
  - 常指基本类型，不是基本类型的两个不同类型操作数往往不能转换
- 类型转换方式有两种：
  - 隐式类型转换：由系统自动按一定规则进行的转换
  - 显式类型转换：由程序员在程序代码中标明，进行强制转换
- 不管哪一种方式，类型转换都是“临时”的，即在类型转换过程中，操作数本身的类型并没有被转换，只是被临时“看作”另一种类型的数值而已。

例：基本类型的转换示例

```
#include<stdio.h>
int main( )
{
   int r = 10;
   float c = 2 * 3.14 * r;          //隐式类型转换
   double s = 3.14 * (double)r * (double)r;//显式类型转换
   double v = 4.0 / 3 * 3.14 * r * r * r;          //隐式类型转换
   printf("%f, %f, %f \n", c, s, v);
   return 0;
}
```

- 对于含有多个操作符的表达式，其类型转换过程是逐步进行的，而不是一次性将所有操作数转换成同种类型的数据再分别参加操作：
  - 比如，“double v = 4/3*3.14*r*r*r;”，转换步骤为：

    1*3.14*r*r*r→ 1.0*3.14*r*r*r → 3.14*r*r*r →...

  - 思考：如果写成“double v = 3.14*r*r*r*4/3;”呢？

隐式类型转换规则

- 对于赋值操作(=)，右操作数的类型转换为左边变量定义的类型；
- 对于逻辑操作与条件操作中第一个表达式的操作数，不是bool型的数据，非0转换为true（非0暨真），0转换为false
  - 比如，在a为0时，!(a)为true

```
int i, sum=0;
scanf("%d", &i);
if(i)
    sum += i;
printf("%d \n", sum);
```

- 对于其他双目操作，按整型提升转换规则和算术类型转换规则进行转换（一般是低精度类型转换为高精度类型）

整型提升转换规则

1）bool、char、signed char、unsigned char、short int、unsigned short int型的操作数，如果int型能够表示它们的值，则其类型转换成int，否则，转换成unsigned int

bool型的操作数，false通常转换为0，true通常转换为1

2）wchar_t和枚举类型转换成下列类型中第一个能表示其值的类型：int、unsigned int、long int、unsigned long int

考虑以下表达式的值

- !3 的结果为：
  A) true　B) false

- -3的结果为：
  A)true　B) false

- true < false 的结果为：
  A) true　B) false

- 30 > 20 > 10的结果为：
  A) true　B) false

算术类型转换规则

1）如果其中一个操作数类型为long double，则另一个的类型转换成long double；

2）否则，如果其中一个操作数类型为double，则另一个的类型转换成double；

3）否则，如果其中一个操作数类型为float，则另一个的类型转换成float；

4）否则，先对操作数进行整型提升转换，如果转换后操作数的类型不一样，则按下列规则再进行转换：

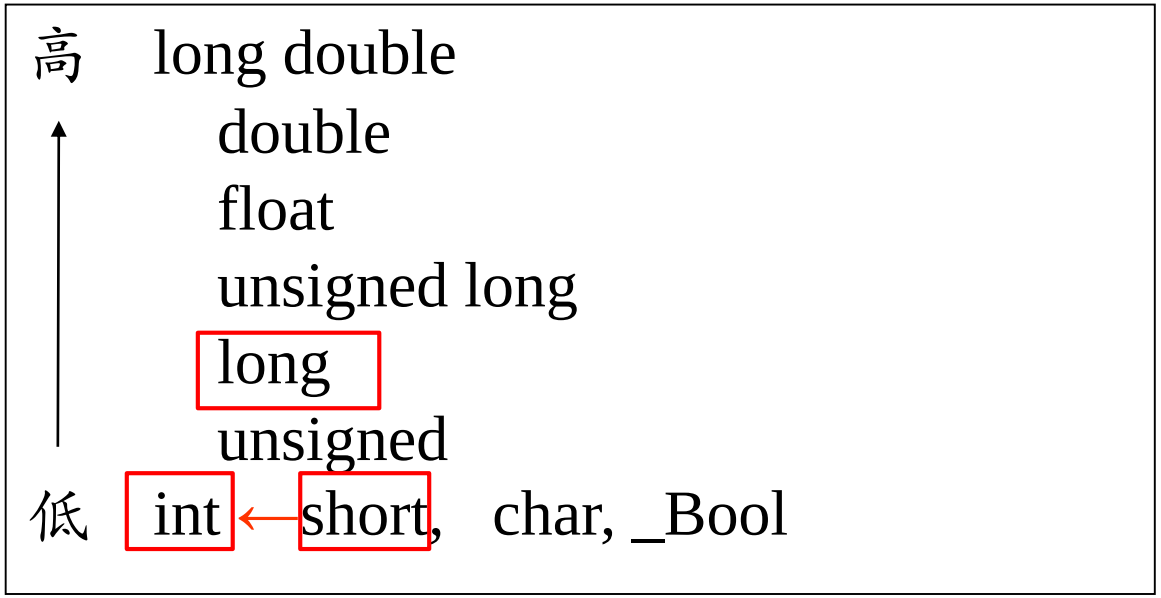
5）如果其中一个操作数类型为unsigned long int，则另一个的类型转换成unsigned long int；

6）否则，如果一个操作数类型为long int，另一个操作数类型为unsigned int，那么，如果long int能表示unsigned int的所有值，则unsigned int转换成long int，否则，两个操作数的类型都转化成unsigned long int；

7）否则，如果一个操作数类型为long int，则另一个的类型转换成long int；

8）否则，如果一个操作数类型为unsigned int，则另一个的类型转换成unsigned int。

类型优先级表


高
long double
double
float
unsigned long
long
unsigned
低
int
short,
char, _Bool

函数参数传递和返回过程中的类型转换

- 隐式类型转换还会发生在函数调用及其值的返回过程中
  - 调用函数时，通常要求实参与形参类型一致；当不一致时，系统会隐式地将实参的数据类型转换成形参的数据类型，再操作
  - 当函数的执行结果与函数定义时返回值类型不同时，系统会隐式地将函数执行结果的数据类型转换成函数定义的数据类型，再返回给调用者

枚举类型

- 程序员用关键词enum构造出来的数据类型，逐个列举出该类型变量所有可能的取值。先构造枚举类型，再定义枚举变量

- 比如，

  enum Color {RED, YELLOW, BLUE};

  Color c1, c2, c3;

  - Color是构造的枚举类型名，花括号里列出了Color类型变量可以取的值
  - c1、c2和c3是枚举变量，这三个变量的取值都只能是RED、YELLOW或BLUE

整形的补码表示

- 补码的简单求法
  - **正数**：同原码
  - **负数**：符号位同原码，其余各位取反，末位加1

真值X：	+1010111	-1010111
$[X]_{原}$(8位)：	01010111	11010111
$[X]_{原}$(16位)：	0000000001010111	1000000001010111
$[X]_{补}$(8位)：	01010111	10101001
$[X]_{补}$(16位)：	0000000001010111	1111111110101001

例：分离出一个整数m的个位、十位和百位

```
…
double x;
void main ()
{
    int m = …;
    x = double(m);
    int a = (x/10-m/10)*10;     // +0.5?
    int b = (x/100-m/100)*10;
    int c = (x/1000-m/1000)*10;
    …
}
```

类型转换！

注意：

C浮点直接转整是截尾取整

（不是四舍五入）

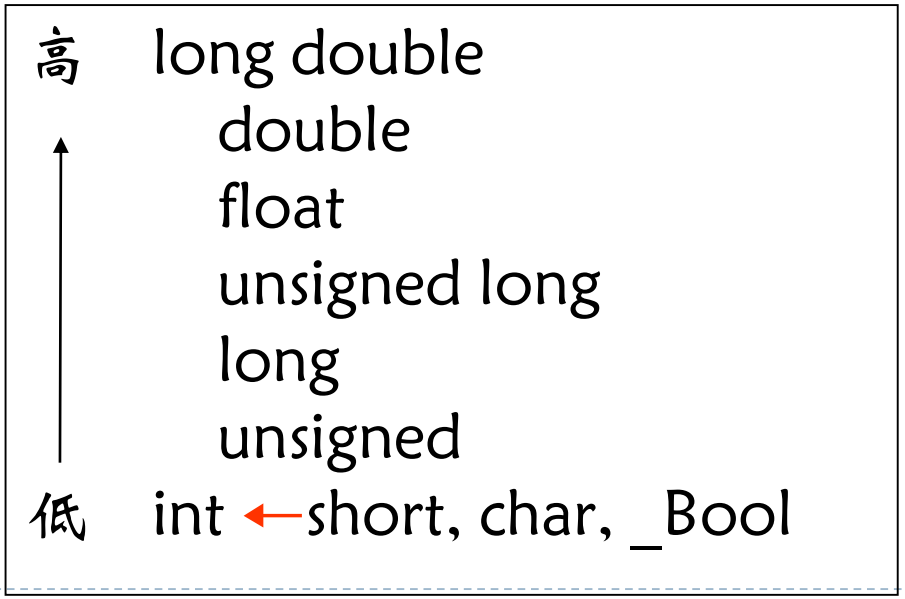
浮点数转整数+0.5

隐式类型转换规则小结

- C程序运行期间，函数和变量的类型以定义的类型为准

实型	去掉小数部分 →	整型
整型	不变 →	实型

- 逻辑操作：
  - 非0数→true，　0→false
- 其他：
  - 精度低→精度高



例：隐式类型转换存在的问题示例

```
#include<stdio.h>
int main( )
{
    int i = -10;
    unsigned int j = 3;
    ……
    if(i + j < 0)
        printf("-7 \n");
    else
        printf("error. \n");        //结果显示error

    if(i < j)
        printf("i<j \n");
    else
        printf("i>j \n");           //结果显示i>j
    return 0;
}
```

不同类型的数据（i和j）在一起操作（算术、比较操作），隐式类型转换的结果违背了常识。

例：隐式类型转换存在的问题示例

```
#include<stdio.h>
int main( )
{
    int i = -10;
    unsigned int j = 3;
    if(i + j < 0)
        printf("-7 \n");          //应改成if(i+(int)j<0)    printf("-7\n");
    else
        printf("error. \n");

    if(i < j)
        printf("i<j \n");         //应改成if(i<(int)j)      printf("i<j \n");
    else
        printf("i>j \n");

    return 0;
}
```

利用显式类型转换，则结果可显示-7和i<j。

显式（强制）类型转换

- 隐式类型转换有时不能满足要求，于是C语言提供了显式类型转换机制，由程序员用类型关键词明确地指出要转换的类型，强制系统进行类型转换：

```
i + (int) j ;
```

建议进行显式类型转换！

- 避免理解的“模糊”（提升程序可读性）和执行的“问题”

- 此外，对于一些对操作数类型有约束的操作，可以用显式类型转换保证操作的正确性。
  - 比如，C语言中的求余数运算要求操作数必须是整型数据，
    “int x = 10%3.4;”应改为
    “int x = 10%(int)3.4;”，否则编译会出错。

- 当把一个枚举值赋值给一个整型变量时，枚举值会隐式转换成整型；而当把一个整数赋给枚举类型的变量时，系统不会将整数转换成枚举类型数据，这时候可以用显式类型转换。
  - 比如，

```
Weekday d;
…
d = (Weekday)(d + 1);
d = d+1;  //编译器会报错，因为d+1的结果为int类型
```

类型转换后的数据精度问题

- 操作数类型转换后，有的精度不受损失，有的则会损失精度。损失精度的隐式类型转换会得到编译器的警告(warning!)。
- 隐式类型转换中，对于赋值运算，右操作数的类型转换为左边变量定义的类型，有可能会损失精度；对于其他运算，按“整型提升转换规则”和“算术类型转换规则”进行转换，一般精度不受损失。

```
int x = 4.3;
```

```
int a = 10;
float b = a + 3.4;
```

例：类型转换后的数据精度问题

```
//…
int main( )
{
    double a=3.3, b=1.1;
    int i = a/b;
    printf("%d \n", i);
    return 0;
}
```

2

```
//…
int main( )
{
    double a=3.3, b=1.1;
    printf("%.0f \n", a/b);
    return 0;
}
```

3

思考：左边的例子如何得到3？

```
printf("%.17f\n",3.3/1.1);
```

```
2.99999999999999960
```

要点：设计程序时，防止因为浮点数的不精确带来的问题

```
printf("%f\n",3.3/1.1);
```

```
3.000000
```

默认6位小数，且四舍五入

基本类型的应用

- 为了选择合适的基本类型定义变量或函数，需要注意考虑以下几个方面：
  - 表达是否自然，比如将一个表示人数的变量定义成float型显然不合适；
  - 可参与的操作与实际操作是否相符，比如需要对两个变量进行求余数运算，那么把其中任一变量定义成double型都不合适；
  - 值域与实际需求是否协调（是否浪费空间或溢出）

伪随机数的生成–强制类型转换的应用

- 实际应用与程序设计中常常需要生成随机数。随机数的特性是产生前其值不可预测，产生后的多个数之间毫无关系
  - 真正的随机数是通过物理现象产生的，比如掷骰子的结果、噪声的强度、福利彩票抽奖等，它们的产生对技术要求往往比较高
- 一般情况下，通过一个固定的、可以重复的计算方法产生的伪随机数就可以满足需求，它们具有与随机数类似的统计特征
  - 线性同余法是产生伪随机数的常用方法

例：生成随机数（范围无限制）

- 利用rand()函数，rand()会返回一随机数值，范围在0至RAND_MAX 间。RAND_MAX定义在stdlib.h，其值为2147483647

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    for(int i=0;i<10;i+)
        printf("%d/n", rand());
}
```

例：生成随机数（在一定范围内）

例如：随机生成10个0~100的数：

```
#include<stdio.h>
#include<stdlib.h>
#define random(x) (rand()%x)

void main()
{
    for(int x=0;x<10;x++)
        printf("%d/n",random(100));
}
```

例：生成随机数（几次操作得到不一样的随机数）

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define random(x) (rand()%x)

void main()
{
   srand((int)time(0));    // srand(time(NULL));
   for(int x=0;x<10;x++)
      printf("%d/n",random(100));
}
```

为什么用srand()函数

- srand和rand()配合使用产生伪随机数序列
- rand函数在产生随机数前，需要系统提供的生成伪随机数序列的种子，rand根据这个种子的值产生一系列随机数。如果系统提供的种子没有变化，每次调用rand函数生成的伪随机数序列都是一样
- srand(unsigned seed)通过参数seed改变系统提供的种子值，从而可以使得每次调用rand函数生成的伪随机数序列不同，从而实现真正意义上的“随机”
- 通常可以利用系统时间来改变系统的种子值，即srand(time(NULL))，可以为rand函数提供不同的种子值，进而产生不同的随机数序列

用rand()和srand()产生伪随机数的方法总结

课外阅读：

http://blog.chinaunix.net/uid-26722078-id-3754502.html

小 结

- 数据为什么要分成不同的类型
  - 表示具有不同取值“属性”的数据
  - 便于合理分配内存，产生高效代码
  - 便于运算，便于数据的处理
  - 便于自动进行类型一致性检查，保护数据，提高程序的可靠性

Q & A