**转换为浮点类型**

不是所有的整数都可以用浮点类型精确表示。例如，虽然类型浮点类型包括长整型和超长整型，浮点类型精确到小数点后六位。因此，一些长整型的值不能精确地存储在浮点对象中。这样的转换的结果是下一个低位或下一个高位表示了这个值，如下面的示例所示：

longl\_var = 123456789L;

floatf\_var = l\_var;

printf(“The rounding error (f\_var–l)var) is %f\n”, (double) f\_var–l\_var);

请注意，本示例中的减法使用至少双精度执行。显然下面代码的输出为：

The rounding error（f\_var - l\_var;）为3.000000

浮点类型中的任何值都可以精确地用更高精度的另一个浮点类型表示。因此，当double类型转换为long double时，或当float类型转换为double或long double时，该值被精确保留。然而，在从更精确的类型转换到不怎么精确的类型的时候，被转换的值可能超出新类型的取值范围。如果值超过目标类型的范围，转换的结果是未知的。如果转换的值在目标类型的范围内，但不能在目标类型的精度中精确表示，则结果是更小或更大的表示值。在2-2这个例子中通过转换为较低精确类型的值阐述了错误类型的产生。

当一个复杂的数转换为浮点类型时，虚部就被丢弃，而结果只是这个复杂数据的实部，这会使得转换离预期的目标类型越来越远。

**转换为复杂的浮点类型**

当一个整数或者一个浮点类型的实部被转换为复杂类型时，结果的实部是通过相应的浮点类型的实部转换获得的。虚部是0.

当一个复杂的数转换为不同的复杂类型时，实部和虚部是分开来转化的，参照下面的浮点类型的例子：

#include <complex.h>

double\_Complexdz = 2;

float\_Complexfz = dz+I;

在这两个初始化中的第一个，整数2隐式地转换为double\_Complex类型并赋值给dz。dz的结果值为2.0 + 0.0 \*I。在fz的初始化中，double\_CompLex类型的dz值的两个部分被转换为float类型，使得fz的实部等于2.0F，虚部1.0F。

**非算术类型的转换**

指针以及数组和函数的名称也受到某些隐式和显式类型转换的影响。 结构体不能被转换，尽管指向它们的指针可以转换为其他指针类型或从其他指针类型转换。

**数组和函数指示符**

数组或函数指示符是具有数组或函数类型的任何表达式。在大多数情况下，编译器将具有数组类型的表达式（例如数组的名称）隐式转换为指向数组的第一个元素的指针。数组表达式不会在下面的情况下转换为指针：

·当数组是sizeof运算符的操作数时

·当数组是地址运算符的操作数时，

·当一个字符串用于初始化一个char数组，wchar\_t，char16\_t ,或者是char32\_ t

下面的例子演示了使用转换规范％p打印指针值，将数组指示符隐式转换为指针：

#incLude<stdio.h>

int \* iPtr = 0;

intarray\_Length = sizeof（iArray）/ sizeof（int）;

printf(“The array stats at the address %p.\n”,iArray);

\* iArray = 5;

iPtr = iArray + array\_Length-1;

printf(“The last element of the array is %d.\n”,\*iPtr);

在本例中的array\_Length的初始化中，表达式sizeof（iArray）产生整个数组的大小，而不是指针的大小。然而，相同的标识符iArray被隐式转换为出现的其他三个语句中的指针：

作为第一个printf()中的参数

作为取消引用运算符的操作数\*

在指针算术运算和iPtr中的赋值

字符数组的名称在字符串操作中用作指针，如本示例所示：

#incLude<stdio.h>

#incLude<string.h>

charmsg [80] =“I’m a string literal.”;

printf（“string is％d characterslong.\n”，strlen（msg））;

pr-Lntf（“The array named msg is ％d byteslong.\n”,sizeof（msg））;

在本例中的函数strlen（msg）中，数组标识符msg被隐含地转换为指向数组的第一个元素的指针，函数参数的类型为const char \*。在内部：，strLen（）只计算从该地址开始的字符到第一个nu11字符或者是字符串终止符。

类似地，指定函数的任何表达式，例如函数名称，也可以隐式地转换为指向函数的指针。同样，当表达式是地址运算符＆的操作数时，此转换不适用。 sizeof运算符不能与函数类型的操作数一起使用。下面的例子指出函数名到指针的隐式转换:

#incLude<stdio.h>

void func0（）{puts(“This is void func0().”);}

void func1（）{puts(“This is void func1().”);}

void（\* funcTabLe [2]）（void）={func0,func1};

for(inti =0;i<2;++i)

funcTable[i]();

**显式指针转换**

要将指针从一种指针类型转换为另一种指针类型，您必须使用显式转换。 在某些情况下，编译器提供了一个隐式转换。 指针也可以被显式地转换为整数，反之亦然。

**对象指针**

您可以将对象指针（即指向完整或不完整对象类型的指针）显式转换为任何其他对象指针类型。 在你的程序中，你必须确保你使用转换的指针是有意义的。 这里是一个例子：

floatf\_var = 1.5F;

long \* l\_ptr =（long \*）＆f\_var;

double \*d\_ptr =（double \*）l\_ptr;

printf(“The %zu bytes that represent %f ,in hexadecimal:0x%lx\n”,sizeof(f\_var),f\_var,\*l\_ptr);

\*（float \*）d\_ptr = 2.5，

如果转换后的对象指针不具有新类型所需的准确，则使用指针的结果是未定义的。 在其这种情况下，将指针值转换回原指针类型将保证产生与原始指针相同的值。

如果将任何类型的对象指针转换为任何char类型的指针（char，signed char或unsigned char），则结果是指向对象的第一个字节的指针。 第一个字节在这里被认为是具有最多地址的字节，而不管系统的字节顺序结构如何。 下面的示例使用此功能打印结构变量的十六进制转储：

#incLude<stdio.h>

struct Data {

shortid;

doublevaL;

};

struct DatamyData = {0x123，77.7};

unsigned char \* cp =（unsigned char \*）＆myData;

prLntf（“％p：”，cp）;

for（int l\_ = 0; i <sizeof（myData）; ++ i）

printf（“％02X”，\*（cp + i））;

putchar（'\ n'）;

这个例子产生的输出：

Oxbffffd70: 23 01 00 00 00 00 00 00 CD CC CCCC CC6C53 40

前两个字节23 01的输出表明该代码在小端系统上执行：具有结构最早地址的字节是短成员的最高有效字节。

**函数指针**

函数的类型始终包括其返回类型，并且还可以包括其参数类型。 您可以显式将指向给定函数的指针转换为指向不同类型的函数的指针。 在下面的示例中，typedef语句定义了“具有一个doubLe参数并返回double值”函数的类型的名称：

#incLude<math.h>

typedef double（func\_t）（double）;

func\_t \* pFunc = sqrt;

double y = pFunc（2.0）;

printf(“The square root of 2 is %f.\n”,f);

pFunc =（func\_t \*）pow;

在该示例中，函数指针pFunc被分配具有不同类型的函数的地址。但是，如果程序使用指针来定义与精确的函数指针类型不匹配的函数，那么程序的行为是未定义的。

**隐式指针转换**

编译器会隐式转换某些类型的指针。 赋值，使用等式运算符==和！=的条件表达式，以及函数涉及三种情况下的隐式指针转换，这些情况在后面的小节中描述。三次隐式指针转换是：

·任何对象指针类型都可以隐式转换为指向void的指针，反之亦然。

·任何指向给定类型的指针都可以隐式转换为指向更确定该指针类型的限定版本-即具有一个或多个添加类型限定符的类型。

·一个nu11指针可以隐式转换为任何指针类型。

**空指针**

指向void的指针-即类型为void \*的指针，用作“多用途”指针来表示任何对象的地址，而不考虑其类型。例如，malloc()函数返回一个指向void的指针。 在你可以访问内存块之前，void指针必须转换为指向一个对象的指针。程序使用标准函数qsort（）对数组进行排序，该函数在头文件stdlib.h中用下面的声明：

void qsort（void \* array，size\_t n，size\_telement\_size, int（\* compare）（const void \*，const void \*））;

qsort（）函数使用快速排序算法以升序从地址array开始对数组进行排序。 假设数组有11个元素，其大小为element\_size。

第四个参数compare，是指向qsort（）的函数的指针，以比较任何两个数组元素。要比较的两个元素的地址在其指针参数中传递给此函数。通常这个比较函数必须由程序员定义。它必须返回一个不小于等于或者说是大于0的值，以指示第一个元素是否小于等于或大于第二个元素。

例4．对比qsort（）

#include <stdlib.h>

#define ARR LEN20

int floatcrr1p（const void \* p1，const void \* p2）

{

float x = \*（float \*）p1，

y = \*（float \*）p2;

return（x <y）？ -1：（（x == y）≠0：1）;

}}

int main（）

{

float \* pNumbers = malloc（ARR\_LEN \* sizeof（float））;

qsort（pNumbers，ARR\_LEN，sizeof（float），floatcmp）;

return 0;

}}

在Example4-1中，malloc（）函数返回一个void \*，它在pNumbers的赋值中隐式转换为float \*。在qsort（）函数中，第一个参数pNumbers是从float \*隐式转换为void \*，以及隐式解释为函数指针。最后,当floatcmp（）函数由qsort（）约束，它接收类型void \*的参数，“universa1”指针类型，并且必须将它们显式地转换为float \*，然后解除引用以初始化其float变量。

**指向合格对象类型的指针**

C中的类型限定符是const，volatile和restrict。例如，编译器必要时将任何指向int的指针隐式转换为指向constint的指针。如果你想删除一个限定条件，而不是增加一个限定条件，你必须使用显式类型转换，如下面的例子所示：

int n = 77;

constint \* ciPtr = 0;

ciPtr =＆n;

n = \* ciPtr + 3;

\* ciPtr \* = 2;

\*（int \*）ciPtr \* = 2;

在这个例子中的最后一行语句说明了指向constqualified指针类型的指针有时是只读指针的原因，虽然你可以修改值，你不能使用它们来修改他们指向的对象。

**空指针常量**

nu11指针常数是值为0的整数常量，或者是作为指向void的指针的固定整数值0。宏NULL在头文件stdlib.h，stdio.h和其他里定义为nu11指针常量。下面的示例使用宏NULL作为指针常量来初始化指针，而不是整数零或nu11字符：

#include <stdlib.h>

long \* lPtr = NULL;

if（LPtr！= NULL）

{

}}

当将空指针常量转换为另一个指针类型时，结果称为空指针。nu11指针的位模式不一定为零。然而，将空指针与零，NULL或另一个空指针进行比较时，结果总是为真。相反，将空指针与任何对象或函数的有效指针进行比较总是产生faLse。

**指针和Integer类型之间的转换**

您可以将指针显式转换为整数类型，反之亦然。这种转换的结果取决于编译器，并且应该与编译的可执行文件所运行的系统的寻址结构一致。指针和整数类型之间的转换可以在系统编程中使用，当程序需要访问特定的物理地址时，如ROM或存储器映射I/O寄存器，这是必要的。

当将指针转换为整数类型时，其范围不足以表示指针的值，结果是未定义的。相反，将整数转换为指针类型不一定产生有效的指针。头文件stdint.h可选地定义整数类型intptr\_t（signed）和uintptr\_t（unsigned）。任何有效的指针都可以转换为这些类型之一，并且后续转换回指针可以保证产生原始指针。因此，你应该使用这些类型之一，如果stdint.h定义它们，任何时候你需要在指针和整数之间执行转换。

这里有几个例子：

float x = 1.5F，\* fPtr =＆x;

unsignedlongLong adr\_va1 =（unsignedlonglLong）fPtr;

uintptr\_tadr\_val =（uintptr\_t）fPtr;

unsigned short \* biosPtr =（unsigned short \*）0x400000L;

unsigned short com1\_io = \* biosPtr;

printf(“COM1 has the I/O base address %Xh.\n”,com1\_io);

最后三个语句从系统数据表中获取有关硬件配置的信息，假设操作环境允许程序访问该存储区。在使用大内存模式1编译的DOS程序中，指针为32位宽，由高16位的段地址和低16位的偏移量组成.因此，先前示例中的指针biosPtr可以用一个常数初始化。