* 强制转化的原理

int a; float b;

变量就是由以地址100开始到103结束的4个字节内存空间内的0-1代码组成。b变量则是由以地址104开始到112结束的8个字节内存空间内的0-1代码组成。而在机器中，这些内存都是连续的0-1代码，机器并不知道100~103是整型而104~111是float型，所有这些类型都是编译器告知的。当我们用a时，由于前面把a定义为int型，则编译器知道从a的地址开始向后取4个字节再把它解释成int型。那么(float)a，就是先按照int类型取出该数值，再将该数值按照int to float的规则转换成float型。

所以强制类型转换就是按照某个变量的类型取出该变量的值，再按照\*\*\*to\*\*\*的规则进行强制转转换。如果是(类型名)常数，则是将该常数按照常数to类型 的规则进行强制转换。

* 指针强制转化的理论可行性

指针也是一个变量，它自己占据一个4个字节的地址空间（由于程序的寻址空间是2^32次方，即4GB，所以用4个字节表示指针就已经能指向任何程序能够寻址到的空间了，所以指针的大小为4字节），他的值是另一个东西的地址，这个东西可以是普通变量，结构体，还可以是个函数等等。由于，指针的大小是4字节，所以，我们可以将指针强制转换成int型或者其他类型。同样，我们也可以将任何一个常数转换成int型再赋值给指针。所有的指针所占的空间大小都是4字节，他们只是声明的类型不同，他们的值都是地址指向某个东西，他们对于机器来说没有本质差别，他们之间可以进行强制类型转换。

指针to指针的强制类型转换是指将指针所指的内容的类型由原先的类型转换为后面的类型。

例子：

int a = 1; int \*p = &a;

float \*p1 = (float\*)p;

则p和p1的值都是&a，但是\*p是将&a地址中的值按照int型变量进行解释，而\*p1则是将&a地址中的值按照float型变量进行解释。

* 万能指针的引入

鉴于指针之间这种灵活的强制类型转换的需求和出于简化代码的考虑，ANSI C引入了空指针即void\*。void指针又名万能指针，在现在的很多程序中，当参数不确定时就用万能指针代替，这一类的指针在线程\进程函数里特别常见。

ANSI C规定，void指针可以复制给其他任意类型的指针，其他任意类型的指针也可以复制给void指针，他们之间复制不需要强制类型转换。当然任何地址也可以复制给void型指针。我们在《网络编程》中经常会看到accept(socket, (struct sockaddr \*)&saddr\_c, &lenth)之类的语句在&saddr\_c之前需要增加代码(struct sockaddr \*)是因为当此函数被设计的时候ANSI C还没有提出void\*的概念。所有的地址统一用struct sockaddr类型标识，该函数的第二个参数也是指向struct sockaddr类型的指针，此处是强制类型转换。

当然，在某些编译器中不同类型的指针也可以进行直接赋值，但一般情况下会给出类型不匹配的警告。要求程序员显示的给出指针强制类型转换可以提醒程序员小心使用指针，对于明确程序目的具有一定的好处。

1、指针类型强制转换：

int m; int \*pm = &m;

char \*cp = (char \*)&m;

pm指向一个整型，cp指向整型数的第一个字节

2、结构体之间的强制转换

struct str1 a; struct str2 b;

a=(struct str1) b; //this is wrong

a=\*((struct str1\*)&b); //this is correct