C++ const引用与转换类型

< 上一页 下一页 >

不同类型的数据占用的内存数量不一样，处理方式也不一样，指针的类型要与它指向的数据的类型严格对应。下面的例子演示了错误的指针使用方式：

```
int n = 100;
int *p1 = &n;  //正确
float *p2 = &n;  //错误

char c = '@';
char *p3 = &c;  //正确
int *p4 = &c;  //错误
```

虽然 int 可以自动转换为 float，char 也可以自动转换为 int，但是 `float *` 类型的指针不能指向 int 类型的数据，`int *` 类型的指针也不能指向 char 类型的数据。

为什么「编译器禁止指针指向不同类型的数据」是合理的呢？

以 int 类型的数据和 `float *` 类型的指针为例，我们让 `float *` 类型的指针强制指向 int 类型的数据，看看会发生什么。下面的代码演示了这一幕：

```
#include <cstdio>
using namespace std;

int main(){
    int n = 100;
    float *p = (float*)&n;
    *p = 19.625;
    printf("%d\n", n);

    return 0;
}
```

运行结果：
1100808192

将 float 类型的数据赋值给 int 类型的变量时，会直接截去小数部分，只保留整数部分，本例中将 19.626 赋值给 n，n 的值应该为 19 才对，这是我们通常的认知。但是本例的输出结果是一个毫无意义的数字，它与 19 没有任何关系，这颠覆了我们的认知。

虽然 int 和 float 类型都占用 4 个字节的内存，但是程序对它们的处理方式却大相径庭：
- 对于 int，程序把最高 1 位作为符号位，把剩下的 31 位作为数值位；
- 对于 float，程序把最高 1 位作为符号位，把最低的 23 位作为尾数位，把中间的 8 位作为指数位。

> 关于整数和小数在内存中的存储形式，我们已在《整数在内存中是如何存储的》《小数在内存中是如何存储的》两节中讲到，不了解的读者请猛击链接学习。

n 存储的二进制位是不变的，只是当以不同的形式展现出来的时候，我们看到的结果是不一样的。读者可以尝试通过 `printf("%f\n", *p);` 输出 n 的值，得到的结果就是 19.625000。

让指针指向「相关的（相近的）但不是严格对应的」类型的数据，表面上看起来是合理的，但是细思极恐，这样会给程序留下很多意想不到的、难以发现的 Bug，所以编译器禁止这样做是非常合理的。当然，如果你想通过强制类型转换达到这个目的（如上例所示），那编译器也会放任不管，给你自由发挥的余地。

引用（Reference）和指针（Pointer）在本质上是一样的，引用仅仅是对指针进行了简单的封装，「类型严格一致」这条规则同样也适用于引用。下面的例子演示了错误的引用使用方式：

```
01.  int n = 100;
02.  int &r1 = n;  //正确
03.  float &r2 = n;  //错误
04.
05.  char c = '@';
06.  char &r3 = c;  //正确
07.  int &r4 = c;  //错误
```

const 引用与类型转换

「类型严格一致」是为了防止发生让人匪夷所思的操作，但是这条规则仅仅适用于普通引用，当对引用添加 const 限定后，情况就又发生了变化，编译器允许引用绑定到类型不一致的数据。请看下面的代码：

```
01.  int n = 100;
02.  int &r1 = n;  //正确
03.  const float &r2 = n;  //正确
04.
05.  char c = '@';
06.  char &r3 = c;  //正确
```

```
const int &r4 = c;  //正确
```

当引用的类型和数据的类型不一致时，如果它们的类型是相近的，并且遵守「数据类型的自动转换」规则，那么编译器就会创建一个临时变量，并将数据赋值给这个临时变量（这时候会发生自动类型转换），然后再将引用绑定到这个临时的变量，这与「将 const 引用绑定到临时数据时」采用的方案是一样的。

注意，临时变量的类型和引用的类型是一样的，在将数据赋值给临时变量时会发生自动类型转换。请看下面的代码：

```
float f = 12.45;
const int &r = f;
printf("%d", r);
```

该代码的输出结果为 12，说明临时变量和引用的类型都是 int（严格来说引用的类型是 int &），并没有变为 float。

当引用的类型和数据的类型不遵守「数据类型的自动转换」规则，那么编译器将报错，绑定失败，例如：

```
char *str = "http://c.biancheng.net";
const int &r = str;
```

`char *` 和 `int` 两种类型没有关系，不能自动转换，这种引用就是错误的。

结合上节讲到的知识，总结起来说，给引用添加 const 限定后，不但可以将引用绑定到临时数据，还可以将引用绑定到类型相近的数据，这使得引用更加灵活和通用，它们背后的机制都是临时变量。

引用类型的函数形参请尽可能的使用 const

当引用作为函数参数时，如果在函数体内部不会修改引用所绑定的数据，那么请尽量为该引用添加 const 限制。

下面的例子演示了 const 引用的灵活性：

```
#include <cstdio>
using namespace std;

double volume(const double &len, const double &width, const double &hei){
    return len*width*2 + len*hei*2 + width*hei*2;
}

int main(){
```

```
    int a = 12, b = 3, c = 20;
    double v1 = volume(a, b, c);
    double v2 = volume(10, 20, 30);
    double v3 = volume(89.4, 32.7, 19);
    double v4 = volume(a+12.5, b+23.4, 16.78);
    double v5 = volume(a+b, a+c, b+c);
    printf("%lf, %lf, %lf, %lf, %lf\n", v1, v2, v3, v4, v5);

    return 0;
}
```

运行结果：
672.000000, 2200.000000, 10486.560000, 3001.804000, 3122.000000

volume() 函数用来求一个长方体的体积，它可以接收不同类型的实参，也可以接收常量或者表达式。

概括起来说，将引用类型的形参添加 const 限制的理由有三个：

- 使用 const 可以避免无意中修改数据的编程错误；
- 使用 const 能让函数接收 const 和非 const 类型的实参，否则将只能接收非 const 类型的实参；
- 使用 const 引用能够让函数正确生成并使用临时变量。

< 上一页 下一页 >