C++ 类型转换 讲义

1. 引言

- **字符的储存**
  字符 `'A'` 在计算机底层又是用哪个数字存储的呢？计算机的本质是处理数字，我们看到的所有文本、字符，在它眼中都是一个个特定的数字。最著名的编码表就是 **ASCII** (美国信息交换标准代码)，在书本P17页有部分的ASCII编码表可做参考。

2. 核心思想

自动类型转换
我们有一个整数 `int number = 65;`。我们想得到它对应的字符。一个想法是：C++ 那么聪明，我直接把 `int` 赋值给 `char` 会怎么样？其实在 C++ 中，`char` 和 `int` 可以自由转换和运算。

```
int number = 65;
char c = number; // C++ 允许这样做，它会自动转换
cout << c;       // 输出 'A'
```

这种由编译器自动完成的转换，叫做 **自动类型转化(隐式转换)**。在很多简单场景下，它完全没有问题！但是，这种有时会隐藏一些问题。如果代码很复杂，**其他人读到 `char c = 65;` 时，可能会疑惑：这是一个笔误吗？还是故意这么写的？为什么两边的数据类型不一样？**

因为正常情况下，**赋值符号的右边的数据类型**应该和**左边的变量数据类型相匹配。**

但是有些情况也可以不一致。源类型向目标类型转换时(例子中 `char c = number;` 源类型就是 `int`，目标类型是 `char`)。如果源类型和目标类型不一致，**在编译器知道怎么转换的情况下，会进行自动类型转换！**

- **例如**：`int` 赋给 `double` 是安全的（`int` 转换为 `double`）。
- **但请注意：** `double` 赋给 `int` 时，虽然编译器会尝试转换（截断小数部分），但这通常被认为是**不安全的**，因为它会丢失精度(小数点后的数据没有了)。
- **整型和浮点型混合运算时，整型会向浮点型转换以避免数据丢失。核心原则是** “向高精度、大范围类型转换”，以避免数据丢失。例如，`int` 和 `double` 相加时，`int` 会被转换为 `double`。

简单来说，C++的自动类型转换是尽力让你在不同的数据类型之间进行操作时，得到一个合理的结果，并且尽量不丢失数据。

强制类型转换(推荐使用！)
为了让我们的代码意图更加清晰，我们常常需要**强制类型转换。**

这相当于我们明确地告诉编译器：**我需要在这里把这个值的类型变一下！**

在 C++ 中，强制类型转换语法是：
`(目标类型) 变量名 或 数值`

- 例1：要把变量 `int number` 转换为 `char` 类型，我们就可以写：`(char)number`。
- 例2：要把变量 `int number` 转换为 `double` 类型，我们就可以写：`(double)number`。

请注意：如果我们想对 `a + b` 这种**整个表达式的计算结果**进行转换时，**就必须**使用括号 `()` 将表达式完整地包裹起来，写成 `(char)(a + b)`，以此来明确地告诉编译器：先算括号里的，再把算出来的最终结果进行转化！

而不能写成 `(char)a + b`，如果这样写，**它会先把变量 `a` 转换为 `char` 类型**。然后再去去和变量 `b` 相加。和我们的目的：将 `a + b` 的结果转化为 `char` 类型就不符了。

步骤拆解
使用强制类型转换将数字转为字符的步骤很简单：

1. **准备一个整数**：定义一个 `int` 类型的变量，并存入一个在 ASCII 码范围内的数字。
2. **执行转换**：使用 `(char)(你的整数变量)` 来进行转换。
3. **存储结果**：将转换后的结果赋值给一个 `char` 类型的变量。
4. **输出验证**：打印输出这个 `char` 变量，看看结果是否符合预期。

3. 实例

- **具体案例**
  让我们把数字 `97` 转换为它对应的 ASCII 字符。
- **分步演示**
  1. **准备一个整数**：

     ```
     int number = 97;
     ```

  2. **执行转换**：我们使用强制类型转换。

     ```
     (char)number // 这个表达式的结果就是字符 'a'
     ```

  3. **存储结果**：

     ```
     char res = (char)number;
     ```

  4. **输出验证**：

     ```
     cout << res; // 屏幕上将会打印出 a
     ```

4. 代码实现

现在，让我们把上面的步骤整合起来，写一个完整的程序。这个程序的作用是输入一个数字，输出这个数字在ASCII 码表中对应的字符。

- **完整代码**

```
#include <iostream>
using namespace std;
int main() {
    cout << "请输入一个整数 (例如 65 到 90 之间): ";

    // 定义变量并接收用户的输入
    int number;
    cin >> number;

    // 使用强制类型转换
    // 将 int 类型的 number 转换为 char 类型
    char ascii_char = (char)number;
```

```
    // 输出结果
    cout << "数字 " << number << " 在 ASCII 码表中对应的字符是: " << ascii_char <<
endl;

    return 0;
}
```

5. 验证

让我们用几个例子来测试一下我们的程序，看看会发生什么。

- **正常情况**:
  - 输入 `65`，输出 `'A'`。
  - 输入 `97`，输出 `'a'`。
  - 输入 `48`，输出 `'0'`。
- **常见问题**
  - **问**：为什么不直接用 `char c = number;`？
  - **答**：在简单情况下可以，但它属于自动类型转化转换，意图不明确。像 `(char)number` 这样的强制转换，能让你的代码意图更清晰，一眼就能让人知道你是故意这样做的。
  - **问**：强制类型转换安全吗？
  - **答**：不一定。比方说： `double` 强制转换为 `int` 类型时，虽然编译器会转换（但是会截断小数部分）。所以，在转换前你要对数据的范围有的认识。例如你想转化一个整数到 `char` 字符类型，那就要确保这个整数的数值在ASCII码表中有对应的字符，如果这个数字根本不存在对应的字符，那计算机就也不知道应该如何转化了。

---

6. C++ 字符大小写转换（两种办法）

一：直接观察法

- 观察 ASCII 码表，你会发现一个规律：
  - `'A'` 是 65, `'a'` 是 97
  - `'B'` 是 66, `'b'` 是 98
  - ...
  - `'Z'` 是 90, `'z'` 是 122

  任意一个相同字母的小写形式，其 ASCII 码值都比大写形式大 32！（`'a' - 'A' = 97 - 65 = 32`）

  基于这个规律，我们有实现方式：
  - **小写转大写**：`大写字符 = 小写字符 - 32`
  - **大写转小写**：`小写字符 = 大写字符 + 32`
  - **案例：大小写转换**

```
char upper = 'B';
char lower = upper + 32; // 66 + 32 = 98 ('b')
cout << lower;   // 输出 b
```

二：差值计算法：

例如想要将大写字符 `'C'` 转为小写

1. **计算偏移量**：首先，我们需要知道目标字母 `'C'` 相对于大写字母起点 `'A'` 的“位置”或“差值量”。
    - 这个偏移量可以通过减法计算得出：`'C' - 'A'`。我们知道结果是 `2`。这告诉我们，`'C'` 是排在 `'A'` 后面的第2个字母（从0开始计数）。
2. **找到新起点**：我们的目标是小写字母，那么小写字母的“起点”是谁？是 `'a'`。
3. **应用偏移量**：我们在新的起点 `'a'` 上，加上刚才计算出的差值量 `2`，就能得到我们想要的小写字母。
    - `'a' + ('C' - 'A')`
    - `'a' + 2` --> 结果就是 `'c'`。

- **公式**：
    - **大写转小写**：`小写字母 = 任意大写字母 - 'A' + 'a'`
    - **小写转大写**：`大写字母 = 任意小写字母 - 'a' + 'A'`

总结

1. 字符在计算机底层是以数字（如 ASCII 码）存储的。
2. `char` 和 `int` 之间可以相互转换。
3. 使用 `(目标类型)值` 的形式可以进行**强制类型转换**，这能让代码的意图更清晰。
4. 进行类型转换时，要注意目标类型的**取值范围**，防止数据**溢出**，例如你想转化一个整数到`char`字符类型，那就要确保这个整数的数值在ASCII码表中有对应的字符，如果这个数字根本不存在对应的字符，那计算机就也不知道应该如何转化了。
5. 英文字母的大小写形式在 ASCII 码表中具有固定的数值差 (`32`)。