C++字符与 ASCII 码 讲义

1. 概念

在计算机的世界里，一切皆是 0 和 1（二进制）。计算机天生只认识数字，不认识字母、标点符号。
为了让计算机能显示和处理文字，人类制定了一套规则：**给每一个字符指定一个唯一的数字编号。**

- **字符 (Character)**：我们在屏幕上看到的符号（如 `'A'`, `'@'`, `'1'`）。
- **ASCII 码**：一种最基础的编码标准，规定了字符和数字之间的对应关系。

---

2. C++ 中的字符类型 (`char`)

在 C++ 中，我们学习了如何储存一个整数或者小数，但是字符是使用 `char` 关键字来定义并储存字符变量的。

2.1 基本语法

- **书写方式**：必须使用**单引号** `' '` 括起来。
- **本质**：`char` 类型在内存中实际上存储的是一个**整数**（即该字符对应的 ASCII 码值）。

```
char letter = 'A';  // 正确
char numChar = '5'; // 正确，这是字符5，不是数字5
// char error = "A"; // 错误！双引号代表字符串
```

2.2 使用 `char`

题目描述:请你实现普通计算器的功能，输入两个整数和一个*，输出这两个整数的乘积。

样例输入: 7 * 2

样例输出: 14

逻辑解析:

- **定义容器**：我们需要两个 `int` 类型的变量来存放参与运算的数字，**以及一个 `char` 类型的变量来存放中间的运算符** `*`，我们就可以定义：`int num1, num2;  char ch;`
- **获取数据**：接受用户输入，将三个值分别放入三个变量中，并且按照顺序读取 `cin >> num1 >> ch >> num2;`
    - **读取 `num1`**：读取的第一个是整数变量，于是 `cin` 接着期待一个整数，例如它读取了 `7`，接着遇到了空格。所以它会**自动结束读取**，将 `7` 存入 `num1` 中。
    - **读取 `op`**：接着是一个字符类型变量，于是 `cin` 接着期待一个字符。拿到了字符 `*`，并存入字符变量 `op` 中。
    - **读取 `num2`**：最后，`cin` 再次期待一个整数。它跳过空格，读取到数字 `2`，存入 `num2`。
- **计算结果：** `num1` 与 `num2` 已经存入了输入的两个整数，最后仅需按照题目要求输出这两个整数的乘积即可：`cout << num1 * num2;`

3. 什么是 ASCII 码?

全称：American Standard Code for Information Interchange（美国信息交换标准代码）。

因为计算机却只认识数字，不认识字符，那计算机怎么存储“Hello”这个单词呢?

为了解决这个问题，人类制定了一张**强制转换表**，这就是 **ASCII 码表**。 它规定了每一个符号对应一个特定的整数。字符在计算机内部，本质上就是一个整数。

它是一张“对照表”，规定了 128 个字符对应的数字。

- 当你按下键盘上的 `A`。
- 计算机查表：哦，`A` 对应的是 `65`。
- 计算机存储：在内存里存下 `65` 。
- 当你 `cout` 时：计算机读取 `65`，查表发现是 `A`，然后就输出一个 `A` 在屏幕上

3.1 ASCII 码表的构成

1. **0 - 31 (控制字符)**：不可打印，用于控制外围设备，这部分**了解**即可，一般不会用到。
    - 例如在我们键盘上的：换行、回车、响铃、退格。
2. **32 - 126 (可打印字符)**：键盘上能打出来的符号，**这部分需要掌握。**
    - 包括：空格、标点、数字、大写字母、小写字母。
3. **127 (Delete)**：删除控制符，**了解**即可。

---

4. 必须记住的关键 ASCII 值

你不需要背诵整个表，但以下几个数值应当记住，因为它们常用于编程逻辑中。

字符类别	起始字符	ASCII 值	结束字符	ASCII 值	备注
数字	`'0'`	**48**	`'9'`	**57**	注意：`'0'` 和 `0` 不是同一个东西
大写字母	`'A'`	**65**	`'Z'`	**90**	
小写字母	`'a'`	**97**	`'z'`	**122**	

规律总结：

1. 数字、大写字母、小写字母都是**连续**排列的。
2. **大小写转换规律**：小写字母的 ASCII 值比对应的大写字母大 **32**。
    - `'a' - 'A' = 32`
    - `'A' + 32 = 'a'`

---

5. 字符与整数的互换 (C++ 的自动类型转换特性)

在 C++ 中，其实 `char` 和 `int` 可以自由转换和运算。

1. 源类型向目标类型转换时。如果目标类型能够容纳源类型的值，且转换是安全的，编译器就会自动完成。
    - **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。在严格的编译环境下可能会有警告，甚至在某些情况下需要显式转换。
    - **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。

5.1 查看字符的 ASCII 码

使用 `int` 类型的变量来接住 `char` 类型的值。

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

int main() {
    char c = 'A';
    cout << "字符: " << c << endl;          // 输出 A
    int num = c;
    cout << "ASCII值: " << num << endl; // 输出 65
    return 0;
}
```

5.2 字符运算 (算术操作)

因为字符本质是整数（通过ASCII码来进行对照），所以可以进行加减乘除。

案例 1：大小写转换

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

案例 2：数字字符转整数数字
这是算法题中非常常见的操作。

```
char c_num = '8';
int i_num = c_num - '0'; // '8'(56) - '0'(48) = 8
cout << i_num << endl;   // 输出数字 8
```