C++ 数学运算~条件表达式 知识巩固讲义

第一章：数学运算

核心思想：
计算机最强大的能力之一就是进行快速而精确的计算。C++内置了我们数学中常用的算术运算符，让我们可以在程序中轻松完成各种计算任务。

详细讲解：

1. **五种基本算术运算符：**
   - `+` (加法): `5 + 3` 结果是 `8`。
   - `-` (减法): `5 - 3` 结果是 `2`。
   - `*` (乘法): `5 * 3` 结果是 `15`。
   - `/` (除法): 这个运算符的行为取决于参与运算的数字类型，需要特别注意。
   - `%` (取模/取余): `5 % 3` 结果是 `2`。
2. **整数除法 vs. 浮点数除法 (重点！)：**
   - **整数除法**：当 `/` 运算符两边的数**都是**整数时，C++会执行整数除法。计算结果会直接**舍弃所有小数部分**，只保留整数部分（向零取整）。
     - 例如：`10 / 3` 的结果是 `3`。`9 / 10` 的结果是 `0`。
   - **浮点数除法**：只要 `/` 运算符两边**至少有一个**是浮点数（如 `double` 或 `float`），C++就会执行浮点数除法，结果会是一个精确的、带小数的数值。
     - 例如：`10.0 / 3`、`10 / 3.0`、`10.0 / 3.0` 的结果都是 `3.33333...`。
     - **技巧**：若想用两个整数 `int` 类型的变量 `a` 和 `b` 计算出浮点数结果，可以写成 `(double)a / b`，这叫做强制类型转换。
3. **取模运算符 `%` 的特性：**
   - 取模就是计算整数除法中的余数。
   - **注意**：`%` 运算符两边的数**必须都是整数**，不能用于浮点数。`10.0 % 3` 这样的代码是错误的，无法通过编译。

示例代码：

```
#include <iostream>

using namespace std;

int main() {
    int a = 13, b = 4;
    double x = 13.0;

    cout << "a + b = " << a + b << endl;       // 输出: a + b = 17
    cout << "a - b = " << a - b << endl;       // 输出: a - b = 9
    cout << "a * b = " << a * b << endl;       // 输出: a * b = 52

    // 重点看除法和取模
    cout << "a / b = " << a / b << endl;       // 整数除法, 输出: a / b = 3
    cout << "a % b = " << a % b << endl;       // 取模, 输出: a % b = 1
```

```
    // 浮点数除法
    cout << "x / b = " << x / b << endl;        // 浮点数除法，输出: x / b = 3.25

    return 0;
}
```

第二章：变量与常量

核心思想：
程序处理的数据需要一个地方存放。变量和常量就是我们向内存申请的、用来存储数据的空间。

详细讲解：

1. **变量:**
    - **是什么？** 变量是程序中用于存储数据的一块内存，它的值在程序运行时**可以被改变**。你可以把它想象成一个可以随时更换内部物品的储物盒。
    - **如何创建（声明与初始化）？**
        - 格式：`数据类型 变量名 = 初始值;`
        - 例如：`int score = 90;` 创建了一个名为 `score` 的整型变量，并存入了初始值 `90`。
        - **好习惯**：定义变量时就给它一个初始值，可以避免很多潜在的错误。
    - **如何使用（赋值）？**
        - 使用 `=` 赋值运算符来更新变量的值。
        - 例如：`score = 95;` 这行代码执行后，`score` 变量中存储的值就变成了 `95`。
    - **命名规则 (重要！):**
        - 只能由字母、数字、下划线 `_` 组成。
        - 不能以数字开头。
        - 不能是C++的关键字（如 `int`, `const`, `double` 等）。
        - 大小写敏感 (`myScore` 和 `myscore` 是两个不同的变量)。
        - **建议**：变量名要“见名知意”，例如命名学生年龄的时候最好用 `studentAge` 而不是 `a`。
2. **常量 (Constant):**
    - **是什么？** 常量的值一旦设定，就**不能再被修改**。
    - **为什么需要常量?**
        - **提高安全性**：防止程序在不经意间修改了那些本应固定的不会改变的值。
    - **如何创建?**
        - 使用 `const` 关键字。
        - 格式：`const 数据类型 常量名 = 值;`
        - **注意**：常量在定义时**必须**进行初始化。
        - **习惯**：常量名通常全部大写，以便和变量区分开，例如：`const double PI = 3.1415926;`

变量与常量 示例代码：

```
#include <iostream>
```

```cpp
using namespace std;

int main() {
    // 定义一个常量 PI
    const double PI = 3.14159;
    cout << "圆周率的值是: " << PI << endl;

    // 以下代码错误！因为尝试修改常量的值会导致编译错误
    // PI = 3.14;

    // 定义一个变量来存储圆的半径
    double radius = 10.0;
    cout << "半径是: " << radius << endl;

    // 变量的值可以被修改
    radius = 12.5;
    cout << "修改后的半径是: " << radius << endl;

    double area = PI * radius * radius;
    cout << "圆的面积是: " << area << endl;

    return 0;
}
```

第三章：`cin` 和 `cout` 控制台输入输出

核心思想：
程序需要与用户进行交互。`cout` 负责将信息显示到屏幕上给用户看，而 `cin` 负责从用户的键盘输入中获取信息。

详细讲解：

1. **基本概念：**
    - `cout` (console output): 用于标准输出，通常指向屏幕输出。
    - `cin` (console input): 用于标准输入，通常从键盘中输入。
2. `cout` **— 输出到屏幕：**
    - 使用**插入运算符** `<<` 将数据“插入”到输出流中。
    - 可以**连续使用** `<<` 来连接多个输出项。
    - `endl` (end line) 是一个特殊的操作符，它作用为换行。
3. `cin` **— 从键盘读取：**
    - 使用**提取运算符** `>>` 从输入流中“提取”数据并存入变量。
    - `cin` 会自动根据你提供的变量类型来解析输入。
    - **工作方式 (重点！)**：`cin` 以**空白符**（**空格**、**换行符** `\n`）作为输入数据的分隔符。当你输入 `18 1.75` 然后按回车时，`cin` 会先读取 `18`，遇到空格后停止，将 `18` 存入第一个变量；接着它会跳过空格，继续读取 `1.75`，遇到最后的换行符停止，将 `1.75` 存入第二个变量。

示例代码：

```cpp
#include <iostream>
```

```cpp
using namespace std;

int main() {
    int age;
    double height;

    // 使用 cout 提示用户输入
    cout << "请输入你的年龄和身高 (用空格隔开), 然后按回车: ";

    // 使用 cin 读取一个整数和一个double
    cin >> age >> height;

    // 使用 cout 打印出读取到的值，进行验证
    cout << "你的年龄是: " << age << " 岁" << endl;
    cout << "你的身高是: " << height << " 米" << endl;

    return 0;
}
```

第四章：转义符

核心思想：
在字符串中，我们有时需要表示一些特殊的、无法直接打印的字符，比如换行。转义符就是一种特殊的表示法，用来在代码中代表这些特殊字符。

详细讲解：

1. **什么是转义符?**
    - 它是一个以反斜杠 `\` 开头的字符序列。
    - 编译器看到 `\` 时，会将其与紧跟其后的字符组合起来，解释成一个具有特殊含义的单一字符。
2. **最常用的转义符：**
    - `\n`：换行符。在 `cout` 中，它的效果与 `endl` 中的换行效果相同，但它不刷新缓冲区。
    - `\\`：表示一个反斜杠 `\` 字符本身。
    - `\"`：表示一个双引号 `"` 字符本身。因为字符串本身是用双引号包围的，所以想在字符串内部打印一个双引号，就需要用 `\"`。

示例代码：

```cpp
#include <iostream>

using namespace std;

int main() {
    // \n 的使用：换行
    cout << "这是第一行。\n这是第二行。" << endl;

    // \\ 和 \" 的使用
    cout << "\n他说: \"C++很有趣!\"" << endl;
    cout << "输出一个反斜杠: \\" << endl;
```

```
    return 0;
}
```

第五章：字符与ASCII码

核心思想：
计算机内部只存储数字。我们所使用的每一个字符，无论是 'A'、'?' 还是 '5'，在计算机内部都有一个对应的整数编码，这个编码标准就是ASCII码。

详细讲解：

1. `char` **数据类型：**
    - `char` 用于存储单个字符。定义时，字符值必须用**单引号** `' '` 括起来。例如：`char grade = 'A';`。
    - `'A'` (字符) 和 `"A"` (字符串) 是完全不同的东西。
2. **ASCII (美国信息交换标准代码)：**
    - 它定义了字符与整数之间的对应关系。
    - **关键点**：`char` 变量在内存中存放的其实是它的ASCII码值（一个整数）。
    - **必须记住的ASCII码：**
        - `'A'` -> `65`
        - `'a'` -> `97`
        - `'0'` (字符'0') -> `48`
    - 英文字母和数字字符的ASCII码都是连续的。所以 `'B'` 的码是 `66`，`'1'` 的码是 `49`，以此类推。
3. `char` **与** `cout`**：**
    - `cout` 很智能，当它看到一个 `char` 类型的变量时，它会默认打印出其对应的**字符**。
    - 如果我们想查看s输出它的ASCII码值，我们需要把它**强制转换**成 `int` 类型然后再进行输出。
4. **字符的运算：**
    - 由于 `char` 的本质是整数，我们可以对它进行数学运算。
    - 例如：`'A' + 1` 的计算结果是 `65 + 1 = 66`。如果将这个结果存入一个 `char` 变量，`cout` 打印它时会显示 `'B'`。这个特性是实现很多字符操作的基础。

示例代码：

```
#include <iostream>

using namespace std;

int main() {
    char ch1 = 'B';
    char ch2 = '8';

    // cout 默认打印字符
    cout << "字符ch1是: " << ch1 << endl;

    // 强制类型转换为int，打印其ASCII码
```

```
    cout << "字符ch1的ASCII码是: " << (int)ch1 << endl;
    cout << "字符ch2的ASCII码是: " << (int)ch2 << endl;

    // 对字符进行运算
    char next_char = ch1 + 1; // 'B' + 1 -> 66 + 1 = 67
    cout << "ch1的下一个字符是: " << next_char << endl; // cout会打印出 'C'

    // 大写转小写示例 ('a' - 'A' = 32)
    char lower_char = ch1 + 32; // 'B' + 32 -> 66 + 32 = 98
    cout << "字符ch1的小写形式是: " << lower_char << endl; // cout会打印出 'b'

    return 0;
}
```

第六章：格式化输入输出 (printf 和 scanf)

核心思想：
虽然 `cin` 和 `cout` 非常方便，但在需要精确控制输出格式（例如控制小数位数）时，源自C语言的 `printf` 函数会更加强大和灵活。同样，`scanf` 在读取特定格式的输入时也很有用。

详细讲解：

1. **必需的头文件**：要使用 `printf` 和 `scanf`，必须包含头文件 `#include <cstdio>`。
2. `printf` **— 格式化输出：**
   - **基本语法**：`printf("格式控制字符串", 参数1, 参数2, ...);`
   - **格式控制字符串**：它是一个普通的字符串，但里面包含了一些以 `%` 开头的**格式说明符**，这些说明符会被后面对应的参数值替换。
   - **常用格式说明符**：
     - `%d`：用于输出一个十进制整数（`int`）。
     - `%f`：用于输出一个浮点数（`float` 或 `double`）。
     - `%c`：用于输出一个单一字符（`char`）。
     - `%%`：如果你想输出一个 `%` 符号本身，需要写成 `%%`。
   - **控制小数位数**：`%.nf` 表示浮点数保留 `n` 位小数（会四舍五入）。这是**最常用的功能之一**。例如 `printf("%.2f", 3.14159);` 输出 `3.14`。
3. `scanf` **— 格式化输入：**
   - **基本语法**：`scanf("格式控制字符串", &变量地址1, &变量地址2, ...);`
   - `&` **(取地址符，至关重要！)**：`scanf` 必须知道要把读取的数据存放到内存的哪个位置，所以我们必须提供变量的地址。**忘记 `&` 是最常犯的、会导致程序崩溃的错误！**
   - **常用格式说明符 (注意与 `printf` 的区别！)**：
     - `%d`：读取一个整数（`int`）。
     - `%lf`：**必须使用** `%lf` 读取一个 `double` 类型的浮点数。这是 `scanf` 和 `printf` 的一个重要区别！
     - `%f`：读取一个 `float` 类型的浮点数。
     - `%c`：读取一个字符。

示例代码：

```cpp
#include <cstdio> // 使用 printf/scanf 需要的头文件
#include <iostream>

int main() {
    // --- printf 演示 ---
    int id = 101;
    double price = 4.586;

    printf("ID: %d, Price: %f\n", id, price);

    // --- scanf 演示 ---
    int age;
    double height;
    // 注意 double 类型的身高要用 %lf 来接收
    scanf("%d %lf", &age, &height);

    printf("年龄: %d, 身高: %.2f\n", age, height);

    return 0;
}
```

第七章：交换变量的值

核心思想：
这是一个非常经典的编程逻辑问题：如何安全地交换两个变量里存放的值。最通用、最易于理解的方法是使用一个临时的第三方变量。

详细讲解：

1. **错误的做法：**
   - 假设 `int a = 10, b = 20;`
   - 如果直接写 `a = b;`，此时 `a` 的值变成了 `20`，而 `a` 原本的值 `10` 已经丢失了。
   - 再写 `b = a;`，`b` 的值仍然是 `20`。结果是两个变量都变成了 `20`，交换失败。
2. **正确的思路（三变量法）：**
   - 想象你有两个杯子，A杯装可乐，B杯装雪碧，你需要一个**空的C杯**来辅助交换。
   - **步骤分解：**
     1. **备份**：声明一个临时变量 `temp`。将 `a` 的值（可乐）存入 `temp`。（`temp = a;`）
     2. **覆盖**：现在 `a` 的值已经安全备份，可以放心用 `b` 的值（雪碧）来覆盖 `a`。（`a = b;`）
     3. **还原**：将 `temp` 中备份的原始 `a` 的值（可乐）赋给 `b`。（`b = temp;`）
   - 经过这三步，`a` 和 `b` 的值就完美地交换了。

示例代码：

```cpp
#include <iostream>

using namespace std;

int main() {
    int a, b;
```

```
    cin >> a >> b;

    // --- 开始交换 ---
    // 1. 声明临时变量
    int temp;

    // 2. 备份 a 的值
    temp = a;

    // 3. 用 b 的值覆盖 a
    a = b;

    // 4. 将 a 的原始值赋给 b
    b = temp;
    // --- 交换结束 ---

    cout << "交换后: a = " << a << ", b = " << b << endl;
    return 0;
}
```

第八章：逻辑运算初步

核心思想：
程序不仅仅是计算，更重要的是能够进行判断和决策。逻辑运算就是程序进行判断的基础，其结果只有两种：**真** (true) 或 **假** (false)。

详细讲解：

1. `bool` **类型：**
    - C++ 中有一个专门表示"真/假"的数据类型，叫做 `bool` (布尔类型)。
    - 它只有两个值：`true` 和 `false`。
    - 当使用 `cout` 输出一个 `bool` 变量时，默认情况下 `true` 会显示为 `1`，`false` 会显示为 `0`。
2. **条件表达式：**
    - 这是一个用于比较两个值的表达式，其最终的计算结果是一个 `bool` 值。
    - **六种关系运算符：**
        - `>`: 大于
        - `<`: 小于
        - `>=`: 大于或等于
        - `<=`: 小于或等于
        - `==`: **等于** (用于比较，是**两个等号**！这是初学者最常犯的错误，容易和赋值的单个 `=` 混淆！)
        - `!=`: **不等于**
3. **如何使用：**
    - 你可以将一个条件表达式的结果赋值给一个 `bool` 类型的变量，以便后续使用。
    - 也可以直接在 `cout` 中输出一个条件表达式的结果，看看它是 `1` 还是 `0`。

示例代码：

```cpp
#include <iostream>

using namespace std;

int main() {
    int score = 85;
    int pass_line = 60;

    // 定义bool变量来存储判断结果
    bool is_passed; // 及格线
    bool is_excellent; // 满分

    // 进行逻辑判断，并将结果存入bool变量
    is_passed = (score >= pass_line); // 判断是否及格，85 >= 60，结果是 true
    is_excellent = (score == 100);    // 判断是否是满分，85 == 100，结果是 false

    // cout 默认将 true 输出为 1，false 输出为 0
    cout << "分数是: " << score << endl;
    cout << "及格线是: " << pass_line << endl;
    cout << "是否及格? (1是，0否): " << is_passed << endl;
    cout << "是否满分? (1是，0否): " << is_excellent << endl;

    // 也可以直接输出表达式的结果
    cout << (score < pass_line) << endl;

    return 0;
}
```