C++基础语法教程

1 C++简介

1.1 C++的历史

C++ 语言的发展始于 20 世纪 80 年代初，由贝尔实验室的 Bjarne Stroustrup 开发。起初，它被命名为“C with Classes”，意在 C 语言的基础上添加面向对象的特性。1983 年，语言正式更名为 C++，其中“++”在 C 语言中表示“递增”操作，象征着该语言在 C 的基础上进行了增强。

C++ 的设计目标是提供一种高效、灵活且功能强大的编程语言，它不仅保留了 C 语言的低级特性，还引入了面向对象编程、泛型编程和异常处理等高级特性。这些特性使得 C++ 成为开发操作系统、浏览器、游戏引擎等高性能软件的首选语言。

1.2 C++的特点

1.2.1 面向对象编程

C++ 支持面向对象编程（OOP），允许开发者定义类和对象，实现封装、继承和多态。这有助于创建可重用的代码和模块化的软件设计。

示例：

```
// 定义一个简单的类
class Circle {
public:
    Circle(double radius) : m_radius(radius) {} // 构造函数
    double area() const { return 3.14159 * m_radius * m_radius; } // 计算面积
    double perimeter() const { return 2 * 3.14159 * m_radius; } // 计算周长

private:
    double m_radius; // 圆的半径
};

// 使用类
int main() {
    Circle c(5.0); // 创建 Circle 对象
    std::cout << "Circle area: " << c.area() << std::endl; // 输出面积
    std::cout << "Circle perimeter: " << c.perimeter() << std::endl; // 输出周长
    return 0;
}
```

1.2.2 泛型编程

C++ 的模板功能支持泛型编程，允许编写可以处理多种数据类型的代码，提高了代码的复用性和灵活性。

示例：

```
// 定义一个模板函数，用于交换两个变量的值
template <typename T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

// 使用模板函数
int main() {
    int x = 10, y = 20;
    double d1 = 1.5, d2 = 2.5;

    swap(x, y); // 交换整数
    swap(d1, d2); // 交换双精度浮点数

    std::cout << "x: " << x << ", y: " << y << std::endl; // 输出交换后的整数
    std::cout << "d1: " << d1 << ", d2: " << d2 << std::endl; // 输出交换后的双精度浮点数
    return 0;
}
```

1.2.3 异常处理

C++ 提供了异常处理机制，允许在程序中捕获和处理运行时错误，增强了程序的健壮性和可维护性。

示例：

```
// 异常处理示例
int main() {
    try {
        int a = 10;
        int b = 0;
        if (b == 0) {
            throw std::runtime_error("Division by zero!");
        }
        int result = a / b;
        std::cout << "Result: " << result << std::endl;
    } catch (const std::runtime_error& e) {
        std::cerr << "Caught exception: " << e.what() << std::endl;
```

```
}
```

```
    return 0;
}
```

1.2.4 性能优化

C++ 提供了对底层硬件的直接访问，使得开发者能够编写出性能极高的代码。它支持多种优化技术，如内联函数、常量表达式和类型安全的指针操作。

示例：

```
// 使用内联函数优化性能
inline int square(int x) {
    return x * x;
}

// 使用函数
int main() {
    int a = 5;
    std::cout << "Square of " << a << " is " << square(a) << std::endl;
    return 0;
}
```

1.2.5 标准库和标准模板库

C++ 标准库提供了丰富的函数和数据类型，如输入输出流、字符串处理和数学函数等。标准模板库（STL）则提供了高效的容器（如向量、列表和映射）和算法，极大地简化了编程工作。

示例：

```
// 使用标准库和 STL 的示例
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> numbers = {5, 3, 8, 1, 2};
    std::sort(numbers.begin(), numbers.end()); // 使用 STL 的 sort 函数排序

    for (int num : numbers) {
        std::cout << num << " "; // 使用标准库的 for-each 循环输出排序后的数字
    }
    std::cout << std::endl;
    return 0;
}
```

C++ 的这些特性使其成为一种功能强大、应用广泛的编程语言，适用于从系统编程到游戏开发的众多领域。

2 环境搭建

2.1 安装编译器

在开始 C++编程之旅前，首先需要安装一个 C++编译器。编译器是将你编写的源代码转换为可执行程序的工具。以下是一些流行的 C++编译器：

- **GCC (GNU Compiler Collection)**: 这是开源社区中最常用的编译器，适用于 Linux 和 Windows 系统。
- **Clang**: 由 LLVM 项目开发，提供了一个更现代的编译器前端，支持多种编程语言，包括 C++。
- **Microsoft Visual C++**: Windows 平台上的官方 C++编译器，集成在 Visual Studio 中。

2.1.1 安装 GCC

如果你使用的是 Linux 系统，可以通过包管理器轻松安装 GCC。例如，在 Ubuntu 或 Debian 系统上，可以使用以下命令：

```
sudo apt-get update
sudo apt-get install build-essential
```

在 Windows 系统上，可以使用 MinGW 或 TDM-GCC 等工具，它们提供了 GCC 编译器的 Windows 版本。

2.1.2 安装 Clang

对于使用 Linux 的用户，安装 Clang 也很简单：

```
sudo apt-get install clang
```

在 Windows 上，可以通过安装 LLVM 包来获取 Clang 编译器。

2.1.3 安装 Microsoft Visual C++

对于 Windows 用户，最直接的方式是安装 Visual Studio，它包含了 Microsoft Visual C++编译器。可以从 Microsoft 官方网站下载 Visual Studio 社区版，它是免费的。

2.2 配置开发环境

配置开发环境是确保你的编译器能够正确编译和运行 C++代码的关键步骤。这通常涉及到设置环境变量、选择合适的 IDE（集成开发环境）或文本编辑器，以及配置编译器选项。

2.2.1 设置环境变量

在安装编译器后，需要确保系统能够找到编译器的可执行文件。这通常通过设置环境变量来实现。在 Linux 系统中，可以通过编辑~/.bashrc 或~/.profile 文件来添加编译器的路径。例如，如果 GCC 安装在/usr/bin 目录下，可以添加以下行：

```
export PATH=/usr/bin:$PATH
```

在 Windows 系统中，可以通过系统设置来添加环境变量。

2.2.2 选择 IDE 或文本编辑器

- **Visual Studio**: Microsoft 的官方 IDE，提供了强大的 C++开发工具，包括智能感知、调试工具和项目管理。
- **Code::Blocks**: 一个免费的、开源的 IDE，支持 GCC 和 Clang 编译器。
- **Sublime Text**: 一个流行的文本编辑器，通过安装插件可以支持 C++开发。
- **Visual Studio Code**: 微软的轻量级代码编辑器，通过安装 C++插件可以提供 C++开发支持。

2.2.3 配置编译器选项

编译器选项可以帮助你优化代码、检查错误和警告。例如，使用 GCC 时，可以添加以下选项来提高代码质量：

```
g++ -Wall -Wextra -pedantic -std=c++17 hello.cpp -o hello
```

这里，-Wall 表示显示所有警告，-Wextra 显示额外的警告，-pedantic 遵循 C++标准，-std=c++17 指定 C++17 标准，hello.cpp 是源代码文件，-o hello 指定输出的可执行文件名为 hello。

2.2.4 示例：使用 GCC 编译和运行 C++代码

假设你有以下 C++代码，保存为 hello.cpp：

```cpp
// hello.cpp
#include <iostream>

int main() {
    std::cout << "Hello, C++!" << std::endl;
    return 0;
}
```

你可以在终端中使用以下命令来编译和运行它：

```
g++ -Wall -Wextra -pedantic -std=c++17 hello.cpp -o hello
./hello
```

这将编译代码并生成一个名为 hello 的可执行文件，然后运行该文件，输出“Hello, C++!”。

通过遵循上述步骤，你可以成功搭建 C++的开发环境，开始你的编程之旅。

3 C++基础语法

3.1 基本语法

3.1.1 变量与数据类型

在C++中，变量是存储数据的容器。定义变量时，需要指定其数据类型，这决定了变量可以存储的数据范围和类型。C++提供了多种内置数据类型，包括但不限于：

- int：用于存储整数。
- float 和 double：用于存储浮点数。
- char：用于存储单个字符。
- bool：用于存储布尔值，即 true 或 false。
- string：用于存储字符串。

3.1.1.1 示例：变量声明与初始化

```
#include <iostream>
#include <string>

int main() {
  // 声明并初始化变量
  int age = 25;
  float height = 1.75;
  char grade = 'A';
  bool isStudent = true;
  std::string name = "张三";

  // 输出变量值
  std::cout << "姓名: " << name << std::endl;
  std::cout << "年龄: " << age << std::endl;
  std::cout << "身高: " << height << std::endl;
  std::cout << "成绩: " << grade << std::endl;
  std::cout << "是否是学生: " << (isStudent ? "是" : "否") << std::endl;

  return 0;
}
```

3.1.2 运算符与表达式

C++支持多种运算符，包括算术运算符、比较运算符、逻辑运算符和赋值运算符。运算符用于执行特定的计算或操作，表达式则是由一个或多个运算符和操作数组成的组合。

3.1.2.1 示例：使用运算符

```cpp
#include <iostream>

int main() {
    int a = 10;
    int b = 5;

    // 算术运算符
    int sum = a + b;
    int difference = a - b;
    int product = a * b;
    int quotient = a / b;
    int remainder = a % b;

    // 比较运算符
    bool isEqual = (a == b);
    bool isNotEqual = (a != b);
    bool isGreater = (a > b);
    bool isLess = (a < b);
    bool isGreaterOrEqual = (a >= b);
    bool isLessOrEqual = (a <= b);

    // 逻辑运算符
    bool andResult = (a > b) && (b > 0);
    bool orResult = (a > b) || (b > a);
    bool notResult = !(a > b);

    // 输出结果
    std::cout << "a + b = " << sum << std::endl;
    std::cout << "a - b = " << difference << std::endl;
    std::cout << "a * b = " << product << std::endl;
    std::cout << "a / b = " << quotient << std::endl;
    std::cout << "a % b = " << remainder << std::endl;
    std::cout << "a == b: " << isEqual << std::endl;
    std::cout << "a != b: " << isNotEqual << std::endl;
    std::cout << "a > b: " << isGreater << std::endl;
    std::cout << "a < b: " << isLess << std::endl;
    std::cout << "a >= b: " << isGreaterOrEqual << std::endl;
    std::cout << "a <= b: " << isLessOrEqual << std::endl;
```

```
    std::cout << "(a > b) && (b > 0): " << andResult << std::endl;
    std::cout << "(a > b) || (b > a): " << orResult << std::endl;
```

```
    std::cout << "!(a > b): " << notResult << std::endl;

    return 0;
}
```

3.1.3 控制结构

控制结构允许程序根据条件执行不同的代码路径，或重复执行一段代码直到满足特定条件。C++中的主要控制结构包括：

- if 语句：用于基于条件执行代码。
- for 循环：用于重复执行一段代码特定次数。
- while 循环：用于在条件为真时重复执行一段代码。
- switch 语句：用于基于不同条件执行不同的代码块。

3.1.3.1 示例：控制结构

```
#include <iostream>

int main() {
    int number = 15;

    // if 语句
    if (number > 10) {
        std::cout << "数字大于 10" << std::endl;
    } else {
        std::cout << "数字小于或等于 10" << std::endl;
    }

    // for 循环
    for (int i = 0; i < 5; i++) {
        std::cout << "循环次数: " << i << std::endl;
    }

    // while 循环
    int counter = 0;
    while (counter < 3) {
        std::cout << "计数: " << counter << std::endl;
        counter++;
    }

    // switch 语句
    char grade = 'B';
    switch (grade) {
        case 'A':
```

```
            std::cout << "优秀" << std::endl;
            break;
        case 'B':
            std::cout << "良好" << std::endl;
            break;
        case 'C':
            std::cout << "及格" << std::endl;
            break;
        default:
            std::cout << "不及格" << std::endl;
    }

    return 0;
}
```

以上示例展示了如何在 C++中声明和初始化变量，使用不同类型的运算符，以及如何通过控制结构来控制程序的流程。这些是 C++编程的基础，掌握它们是学习更高级概念的前提。

4 C++基础语法：函数与作用域

4.1 函数定义与调用

在 C++中，函数是程序的基本构建块，用于执行特定任务。函数可以提高代码的重用性和模块化，使得程序更易于理解和维护。

4.1.1 函数定义

函数定义的基本语法如下：

```
返回类型 函数名(参数列表) {
    // 函数体
}
```

其中，返回类型指明函数执行完毕后返回给调用者的数据类型；函数名是函数的标识符；参数列表包含函数接收的参数类型和名称。

4.1.2 函数调用

函数调用是在程序中使用函数的地方，通过函数名和传递相应的参数来执行函数。

4.1.2.1 示例

下面是一个简单的函数定义和调用的例子：