

预备节 C++基本语法知识

本节将带您了解C++编程语言的基本概念和语法知识，
已掌握C++基本语法的同学可以略过此节。



M学长的考研Top帮

编程语言分类

01
10

低级语言

低级语言更接近机器语言，能够直接控制硬件。它们通常更难编写和理解，但执行效率更高。



汇编语言

汇编语言是一种低级语言，它通过助记符和指令直接与CPU交互，例如MOV和ADD。



高级语言

高级语言更接近人类自然语言，提供了更抽象的概念，更加易于阅读和维护。它们通常更易于学习和使用，但执行效率可能略低于低级语言。



C++

C++是一种高级语言，其代码更易于阅读和维护，提供面向对象编程功能。

从高级到低级到二进制的过程

高级语言代码

高级语言代码是人类易于理解的代码，例如 C++、Python、Java 等。

编译器

编译器将高级语言代码转换为汇编语言，这是一个更接近机器指令的中间语言。

汇编语言

汇编语言使用助记符来表示机器指令，它更接近机器语言，但仍然需要进一步转换。

二进制代码

二进制代码是计算机能够直接执行的指令，由 0 和 1 组成，表示机器指令的具体操作。

从高级到低级到二进制的过程

高级语言代码经过编译器编译后，转换为汇编语言，然后再转换为机器语言（二进制代码），最后由CPU执行。例如：

```
// C++代码  
int main() {  
    int a = 5;  
    int b = 10;  
    return a + b;  
}
```

从高级到低级到二进制的过程

//汇编

MOV EAX, 5 ; 将5移动到EAX寄存器

ADD EAX, 10 ; EAX加上10

//二进制

10111000 00000101 10111000 00001010

变量的定义与声明

变量是计算机内存中存储数据的命名位置。在C++中，每个变量需要在使用前声明，声明时需要指定数据类型。

变量类型决定了它可以存储的数据类型，例如整数、浮点数或字符等。声明时，可以使用数据类型关键字，例如：`int`、`float`、`char`。

```
int age; // 声明一个整数类型的变量age  
age = 25; // 给变量赋值
```

基本数据类型

整数类型（int）用于存储整数，如10、-5、0等。它是最常用的数据类型之一。整数类型可以分为有符号整数和无符号整数，有符号整数可以表示负数，而无符号整数只能表示非负数。

浮点数类型（float）用于存储实数，如1.5、-2.7、3.14等。它可以表示小数部分。浮点数类型在计算机中通常采用 IEEE 754 标准进行存储，该标准规定了浮点数的表示方法和精度。

字符类型（char）用于存储单个字符，如 'A'、'b'、'#' 等。字符类型通常使用 ASCII 码或 Unicode 码来表示字符。

布尔类型（bool）用于存储真值或假值，用 true 和 false 表示。布尔类型通常用于表示条件判断的结果。

控制结构

条件语句

条件语句用于根据条件执行不同的代码块。C++ 提供两种主要的条件语句：if 语句和 switch 语句。

1. if 语句：用于判断一个条件是否成立，如果成立则执行 if 代码块，否则执行 else 代码块。

示例

if 语句和 switch 语句的代码示例展示了两种语句的使用场景。if 语句用于判断成绩是否及格，switch 语句用于根据星期数显示星期几。

```
int score = 85;
if (score >= 60) {
    std::cout << "及格" << std::endl;
} else {
    std::cout << "不及格" << std::endl;
}
```


循环语句

for循环

```
for (int i = 0; i < 5; i++) {  
    std::cout << "当前  
计数: " << i <<  
std::endl;  
}
```

while循环

```
int i = 0;  
while (i < 5) {  
    std::cout << "当前  
计数: " << i <<  
std::endl;  
    i++;  
}
```

do-while循环

```
int j = 0;  
do {  
    std::cout << "当前  
计数: " << j <<  
std::endl;  
    j++;  
} while (j < 5);
```

循环语句的作用

重复执行相同或类似的任务，从而简化编程过程并提高代码的效率。

跳转语句

break

break用于跳出循环或switch语句，它会终止整个循环或switch语句的执行。

continue

continue用于跳过当前循环的剩余部分，直接进入下一次循环，它不会终止整个循环。

return

return用于退出函数并返回值，它会将函数的执行流程返回到调用该函数的地方。

函数的定义与调用

函数是执行特定任务的代码块，可以接收输入参数并返回结果。定义函数时，需要指定返回类型、函数名称、参数列表以及函数体。

函数的调用是指在程序中使用函数，并将实际参数传递给函数，执行函数体内的代码，并最终得到返回值。

函数的使用示例

```
void increment(int num) {  
    num ++;  
}
```

```
int main() {  
    int number = 5;  
    increment(&number);  
    return 0;  
}
```

函数的调用

1

1. 传递参数

函数调用时，可以将实际的值或变量作为参数传递给函数。

2

2. 新的执行上下文

函数调用会创建一个新的执行上下文，函数在该上下文中执行。

3

3. 返回控制权

函数执行完毕后，控制权会返回到调用函数的地方。

调用栈原理

函数调用过程

当一个函数被调用时，程序会将当前执行的位置（返回地址）压入调用栈，并为函数的参数和局部变量分配内存。

栈的结构

调用栈的结构使得函数可以在执行完后恢复到之前的状态，类似于一个“后进先出”的堆栈。

例子

例如，当调用`add(5, 10)`时，调用栈会压入返回地址、参数5和10，并分配a和b的内存空间，执行函数体后，返回结果并恢复到之前的状态。

参数传递方式

参数传递是函数调用过程中，将实参传递给形参的过程。



值传递是一种常用的方式，适用于小型数据类型。

引用传递则适用于大型数据类型，可以提高效率。

指针传递可以控制数据的地址，更灵活。

值传递

一种常用的参数传递方式，适用于小型数据类型。在函数调用过程中，实参的副本会被传递给形参，函数在执行过程中只操作形参的副本，不会对原始变量产生影响。这样可以确保函数调用不会改变原始变量的值。

```
void increment(int num) {  
    num ++;  
}  
  
int main() {  
    int number = 5;  
    increment(number);  
    // now the value of number is 5  
    return 0;  
}
```

引用传递

1

直接访问

引用传递允许函数直接访问参数的原始变量，而不是其副本。

2

修改影响

在函数内部对参数的修改将影响外部变量。

3

性能提升

引用传递在处理大型对象时特别有用，因为它避免了不必要的复制，提高了性能。

代码示例

```
// 使用引用传递
```

```
void increment(int& num) { // 使用引用参数
```

```
    num++;
```

```
}
```

```
int main() {
```

```
    int number = 5;
```

```
    increment(number); // 直接传递变量，不需要取地址
```

```
    std::cout << "number = " << number << std::endl; // 输出: number = 6
```

```
    return 0;
```

```
}
```


指针传递

通过指针传递参数，可以在函数内部更直接地控制和修改数据。指针传递允许函数操作内存中的特定位置，而不是操作值的副本。这种方式可用于处理复杂的数据结构和大型数据对象，以提高性能和灵活性。

代码示例：

```
void increment(int* num) {  
    (*num)++;  
}  
  
int main() {  
    int number = 5;  
    increment(&number);  
    // now the value of number is 6  
    return 0;  
}
```

指针和引用的区别

- **指针 (Pointer)** : 指针是一个变量, 用于存储另一个变量的内存地址, 它可以动态地指向不同的内存地址。
- **引用 (Reference)** : 引用是一个别名, 直接引用另一个变量。一旦引用被初始化为某个变量, 就不能再引用其他变量。

函数重载与默认参数

函数重载

C++ 允许使用相同的函数名称定义多个函数，只要它们的参数类型或数量不同。这称为函数重载。

编译器会根据传入参数的类型和数量选择合适的重载版本。

默认参数

在函数声明时，可以为参数指定默认值。在调用函数时，可以省略某些参数，使用默认值。

例如，可以定义一个带默认参数的函数 `printMessage()`，它可以输出默认消息或自定义消息。

函数重载

- 函数重载:

```
int add(int a, int b) {  
    return a + b; // 整数相加  
}
```

```
double add(double a, double b) {  
    return a + b; // 浮点数相加  
}
```

在调用时，编译器会根据传入参数的类型和数量选择合适的重载版本。

默认参数

```
void printMessage(const std::string &message = "Hello, World!") {  
    std::cout << message << std::endl; // 输出消息  
}  
  
int main() {  
    printMessage(); // 输出默认消息  
    printMessage("Hello, C++!"); // 输出自定义消息  
}
```


内联函数

内联函数可以减少函数调用带来的开销，从而提高程序性能。

编译器会在调用处将内联函数的代码直接替换到调用位置，避免了函数调用过程。

内联函数适合小型、简单的函数，过多的内联函数可能会导致代码膨胀，反而降低性能。

```
inline int square(int x) {  
    return x * x; // 返回x的平方  
}
```

```
int main() {  
    std::cout << "4的平方是: " << square(4) << std::endl; // 编译时替换为4 * 4  
}
```

C++ class

`class` 是 C++ 中最重要的特性之一，用于实现面向对象编程（OOP）。它将数据和操作这些数据的方法（函数）封装在一起，形成一种数据类型，从而增强代码的可读性、可维护性和复用性。

什么是 class?

- class 是一种用户定义的数据类型，它包含属性（成员变量）和行为（成员函数/方法）。
- class 是 "对象的蓝图" 或模板，而对象（object）是 class 的实例。

基本语法

```
class ClassName {  
public:  
    // 构造函数  
    ClassName();  
  
    // 成员变量  
    int attribute1;  
    double attribute2;  
  
    // 成员函数（方法）  
    void memberFunction();  
};
```


示例

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

class Student {
public:
    // 成员变量
    std::string name;
    int age;

    // 成员函数
    void displayInfo() {
        std::cout << "Name: " << name << ", Age: " << age << std::endl;
    }
};
```

如何使用 `class`

你可以通过创建对象来使用 `class`，然后访问成员变量和调用成员函数。

```
int main() {  
    // 创建 Student 类的对象  
    Student student1;  
  
    // 访问成员变量并赋值  
    student1.name = "John";  
    student1.age = 20;  
  
    // 调用成员函数  
    student1.displayInfo(); // 输出: Name: John, Age: 20  
  
    return 0;  
}
```

成员变量

- **定义：**成员变量（也叫**数据成员**）是类中的变量，表示对象的**属性或状态**。它们存储在类的实例（对象）中，每个对象都有自己独立的成员变量。
- **访问控制：**成员变量可以设置为 `public`（公有）、`private`（私有）或 `protected`（受保护），以控制外部对这些变量的访问权限。
- **生命周期：**成员变量的生命周期与对象相同，当对象被创建时，成员变量被分配内存，当对象被销毁时，成员变量也被销毁。

成员函数

- **定义：**成员函数（也叫**方法**）是类中的函数，用来表示对象的行为或功能。它们可以访问和操作成员变量，定义了类的对象可以执行哪些操作。
- **访问控制：**成员函数同样可以设置为 public、private 或 protected，以控制是否允许从类的外部调用这些函数。
- **作用：**成员函数主要用于对对象的成员变量进行操作，实现对象的行为。

访问修饰符

访问修饰符定义了类的成员的可访问性。C++ 中常用的访问修饰符有：

- `public`：类的成员在类的外部可访问。
- `private`：类的成员在类的外部不可访问，只能在类的内部使用。
- `protected`：与 `private` 类似，但允许子类访问。

构造函数与析构函数

- **构造函数**：构造对象时自动调用，用于初始化对象。构造函数的名称与类名相同，没有返回值。
- **析构函数**：对象销毁时自动调用，用于清理资源，名称是 `~ClassName`，没有参数和返回值。

代码示例

```
class Person {  
public:  
    std::string name;  
  
    // 构造函数  
    Person(std::string n) : name(n) {  
        std::cout << "Person " << name << " is created." << std::endl;  
    }  
  
    // 析构函数  
    ~Person() {  
        std::cout << "Person " << name << " is destroyed." << std::endl;  
    }  
};
```


代码示例

```
int main() {  
    // 构造函数：在 p1 创建时，初始化 name，并打印出创建信息。  
    Person p1("Alice");  
    // 输出：Person Alice is created.  
  
    return 0;  
}  
// 析构函数：在 p1 超出作用域时，自动调用析构函数，释放资源。  
// 当 p1 超出作用域时，输出：Person Alice is destroyed.
```

本课总结

这节课我们学习了C++的基本语法，更多的语法后续将随着课程进展学习。



M学长的考研Top帮



谢谢大家

M学长的考研Top帮