两小时从 `C` 过渡到 `C++`

简介

- `Author` : *GoogTech*
- `Website` : *https://algorithm.show*
- `Email` : *GoogTech@qq.com*
- `Date` : *2021,6,15 ~ 2021,6,22*

目录

第一个C++程序

```
#include <iostream>

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

基础教程：头文件与命名空间

```
/**
 * Video 01: 头文件, 注释, 命名空间
 *
 * 1. C++ 头文件不必是 ".h" 结尾, C++ 语言中的标准库头文件如 "math.h",
 * "stdio.h" 在 C++ 中被名为: "cmath", "cstdio"
 *
 * 2. C++ 支持多行注释
 */
#include <cmath>
#include "cstdio"
int main() {
    double a = 1.2;
    double b = sin(a);
    // %f代表单精度浮点型数据(float), %lf代表双精度浮点型数据(double)
    printf("%lf\n", b); // 0.932039
}
```

```
/**
 * Video 01: 命名空间 namespace
 * 为了防止名字冲突(出现同名), C++ 引入了命名空间(namespace)
 * 通过 "::" 运算符限定某个名字属于哪个名字空间
 *
 * 如区分: "计科01: GoogTech" 与 "计科02: GoogTech"
 *
 * 通常有 3 中方法适用命名空间 X 的名字 name:
```

```
 * 1. 引入整个名字空间: using namespace X;
 * 2. 使用单个名字: using X::name;
 * 3. 程序中加上名字空间前缀: X::name;
 */
#include "cstdio"
namespace first {
    int a;
    void f() { a = a + 1; }
    int g() { return a + 2; }
}

namespace second {
    int a;
    double f() { return a + 1; }
    char g;
}

int main() {
    first::a = 2;
    second::a = 6;
    first::a = first::g() + second::f();
    second::a = first::g() + 6;
    printf("%d\n", first::a); // 11
    printf("%lf\n", second::a); // 0.000000
}
```

基础教程：标准输入输出流

```
/**
 * Video 02: 标准输入输出流
 *
 * C++ 新的输入输出流库 <iostream> 将输入输出看成一个流, 并用输出运算符
 * ">>" 和输入运算符 "<<" 对数据进行输入输出
 *
 * 其中 "count" 和 "bin" 分别代表标准输出流对象(窗口)和标准输入流对象
 * (键盘)
 *
 * 标准库中的名字都属于标准名字空间 std
 */
#include "iostream"
#include <cmath>

using std::cout;

int main() {
    double a;
    // std::endl 是一个操纵符,作用为换行和将缓冲区中的内容刷新到屏幕
    cout << "Please input a number: " << std::endl;
    std::cin >> a; // 100
    double b = sin(a);
    cout << b; // -0.506366
    return 0;
```

```
}
```

```
/**
 * Video 02: 标准输入输出流
 *
 * 引入整个名字空间 std 中的所有名字, 其 cout, cin 都属于名字空间 std;
 */
#include "iostream"
#include "cmath"

using namespace std;

int main() {
    double a;
    cout << "Please input a number: " << endl;
    cin >> a; // 100
    double b = sin(a);
    cout << b; // -0.506366
    return 0;
}
```

基础教程：变量及其作用域

```
/**
 * Video 03: 变量及其作用域
 *
 * 较 C 而言 C++ 变量 "即用即定义", 且可用表达式初始化
 */
#include "iostream"

using namespace std;

int main() {

    double a = 12 * 3.25;
    double b = a + 1.112;
    cout << "a's value :" << a << endl; // a's value :39
    cout << "b's value :" << b << endl; // b's value :40.112

    a = a * 2 + b;
    double  c = a + b * a;
    cout << "c's value :" << c << endl; // c's value :4855.82
    return 0;
}
```

```
/**
 * Video 03: 变量及其作用域
 *
 * 程序块 "{}" 内部作用域可定义与外部作用域同名的变量,在该块里就隐藏了外部变量
 */
```

```cpp
#include "iostream"

using namespace std;

int main() {

    double a;
    cout << "Please input a number :"; // Please input a number :100
    cin >> a;

    {
        int a = 1;
        a *= 10;
        // 输出程序块 {} 内部定义的变量 "int a"
        cout << "Local number :" << a << endl; // Local number :10
    }

    // 输出 main 作用域的变量 "double a"
    cout << "You inputed number :" << a << endl; // You inputed number :100
}
```

```cpp
/**
 * Video 03: 变量及其作用域
 *
 * for 循环语句可以定义局部变量
 */
#include "iostream"

using namespace std;

int main() {

    int i = 0;

    for(int i = 0; i < 4; i++) {
        cout << i << ""; // 0123
    }

    for(i = 0; i < 1; i++) {
        for(int i = 0; i < 4; i++) {
            cout << i << ""; // 0123
        }
        cout << endl;
    }
    return 0;
}
```

基础教程：引用

```cpp
/**
 * Video 04: 引用
 *
 * C++ 引入了 "引用类型", 即一个变量是另一个变量的别名
 */
#include "iostream"

using namespace std;

int main() {
    double a = 3.14;
    double &b = a; //  b 引用 a, 即 b 为 a 的别名, 进而 b 就是 a
    b = 100;
    cout <<  "a's value :" << a << endl; // a's value :100
}
```

```cpp
/**
 * Video 04: 引用
 *
 * 引用经常用作函数的形参,表示形参和实参实际上是同一个对象.
 * 在函数中对形参的修改也就是对实参的修改.
 */
#include "iostream"

using namespace std;

void swap(int x, int y) {
    cout << "swap before: " << x << " " << y << endl; // swap before: 3 4
    int t = x; x = y; y = t;
    cout << "swap after: " << x << " " << y << endl; // swap after: 4 3
}

int main() {
    int a = 3, b = 4;
    swap(a, b);
    // a, b 的最终值并未因 swap 函数而改变
    cout << "main: " << a << " " << b << endl; // main: 3 4
}
```

```cpp
/**
 * 方法一 : 使用指针
 * 修改上述程序, 使得 a, b 的最终值因 swap 函数而发生交换
 */
#include "iostream"

using namespace std;

void swap(int *x, int *y) {
    cout << "swap before: " << *x << " " << *y << endl; // swap before: 3 4
    int t = *x; *x = *y; *y = t;
    cout << "swap after: " << *x << " " << *y << endl; // swap after: 4 3
}
```

```
int main() {
    int a = 3, b = 4;
    // &a 将 a 值赋值给 x, x 为 int* 指针, 指向 a
    swap(&a, &b);
    // a, b 的最终值因 swap 函数而发生交换
    cout << "main: " << a << " " << b << endl; // main: 4 3
}


/**
 * 方法二 : 使用引用
 * 修改上述程序, 使得 a, b 的最终值因 swap 函数而发生交换
 */
#include "iostream"

using namespace std;

void swap(int &x, int &y) {
    cout << "swap before: " << x << " " << y << endl; // swap before: 3 4
    int t = x; x = y; y = t;
    cout << "swap after: " << x << " " << y << endl; // swap after: 4 3
}

int main() {
    int a = 3, b = 4;
    // x, y 分别是 a, b 的引用, 即 x 就是 a, y 就是 b
    swap(a, b);
    // a, b 的最终值因 swap 函数而发生交换
    cout << "main: " << a << " " << b << endl; // main: 4 3
}
```

```
/**
 * 当实参占据内存较大时,用引用代替传值( 需要复制 )可提高效率,
 * 如果不希望无意中修改实参, 可以用 const 修改符, 实例如下:
 */
#include "iostream"

using namespace std;

void change(double &x, const double &y, double z) {
    x = 100;
    // Error: Cannot assign to variable 'y' with const-qualified type 'const
double &'
    y = 200;
    z = 300;
}
```

基础教程：内联函数，异常处理

```cpp
/**
 * Video 05: 内敛函数
 *
 * 对于不包含循环的简单语句，建议用 inline 关键字声明为 "inline 内联函数"，
 * 编译器将内联函数调用其代码展开，称为 "内联展开"，避免函数调用开销，进而提
 * 高程序的运行效率.
 */
#include "iostream"
#include "cmath"

using namespace std;

inline double distance(double a, double b) {
    return sqrt(a * a + b * b);
}

int main() {

   double  a = 1, b = 2;

   // 下面两行代码的作用相同
   cout << distance(a, b) << endl; // 2.23607
   cout << sqrt(a * a + b * b) << endl; // 2.23607

   return 0;
}
```

```cpp
/**
 * Video 05: 异常处理
 *
 * C++ 通过 try-catch 处理异常情况，
 * 正常代码放在 try{} 中，catch{} 代码块用于捕获 try {} 代码块中抛出的异常
 */
#include "iostream"

using namespace std;

double division(int a, int b) {
    if(b == 0) {
        throw "Division By Zero Condition !";
    }
    return (a / b);
}

int main() {

    int x = 50, y = 0;
    double z = 0;

    // 抛出一个除以零的异常，并在 catch 块中捕获该异常
    try {
        z = division(x, y);
        cout << z << endl;
    }catch(const char* msg) {
```

```
        cerr << msg << endl; // Division By Zero Condition !
    }

    return 0;
}
```

基础教程：函数的默认参数

```
/**
 * Video 06: 函数的默认参数
 *
 * 函数的形参可带默认值，但必须一律靠右
 */
#include "iostream"

using namespace std;

double add(double a, double b = 2) {
    return a + b;
}

int main() {
    cout << add(1) << endl; // 3
    cout << add(1, 3) << endl; // 4
    return 0;
}
```

基础教程：函数的重载

```
/**
 * Video 07: 函数的重载
 *
 * C++ 允许函数同名，只要它们的形参不一样(个数或对应参数类型)，
 * 调用函数时将根据实参和形参的匹配选择最佳函数.
 */
#include "iostream"

using namespace std;

double add(double a, double b) {
    return a + b;
}

double add(int a, int b) {
    return a + b;
}
```

```
int main() {
    cout << add(1.1, 2.2) << endl; // 3.3
    cout << add(1, 2) << endl; // 3
    return 0;
}
```

> 当然还有`运算符`重载, 这里就不在举例了...

基础教程：模板函数

```
/**
 * Video 08: 模板函数
 *
 * 对于任何不同类型间的大小比较，可使用如下模板函数让编译器自动生成
 * 一个针对该数据类型的具体函数
 */
#include "iostream"

using namespace std;

template<class T1, class T2>
T1 minValue(T1 t1, T2 t2) {

    return t1 ? t2 > t1 : t2;
}

int main() {
    cout << "INT Min :" << minValue(1, 2) << endl; // INT Min :1
    cout << "DOUBLE Min :" << minValue(1.1, 2.2) << endl; // DOUBLE Min :1
    cout << "INT AND DOUBLE Min :" << minValue(1, 1.1) << endl; // INT AND
DOUBLE Min :1
    return 0;
}
```

基础教程：虚函数

```
/**
 * C++ 基础教程: 虚函数
 *
 * 定义一个函数为虚函数，不代表函数为不被实现的函数。
 * 定义他为虚函数是为了允许用基类的指针来调用子类的这个函数。
 *
 * 下面这个例子是虚函数的一个典型应用，
 * 通过这个例子，也许你就对虚函数有了一些概念。
 * 它虚就虚在所谓 "推迟联编 " 或者 "动态联编 "上，
```

```
 * 一个类函数的调用并不是在编译时刻被确定的，而是在运行时刻被确定的。
 * 由于编写代码的时候并不能确定被调用的是基类的函数还是哪个派生类的函数，所以被成为 "虚 " 函数。
 *
 * 注: 虚函数只能借助于指针或者引用来达到多态的效果。
 *
 * https://www.runoob.com/w3cnote/cpp-virtual-functions.html
 */
#include <iostream>

using namespace std;

class A {
public:
    virtual void foo() {
        cout << "A::foo() is called" << endl;
    }
};

class B : public A {
public:
    void foo() {
        cout << "B::foo() is called" << endl;
    }
};

int main() {
    A *a = new B();
    // a 虽然是指向 A 的指针，但是被调用的函数 foo() 却是 B 的!
    a->foo(); // B::foo() is called

    return 0;
}
```

基础教程 : new与delete

```
/**
 * Video 09: new 与 delete
 *
 * C++ 中使用 new 运算符来为任意的数据类型动态分配内存，使用 delete 操作符释放它所占用的内存，
 * new 与 malloc() 函数相比，其主要的优点为 new 不只是分配了内存，它还创建了对象.
 */
#include "iostream"

using namespace std;

int main() {

    // 初始化为值 NULL 的指针
    double* pvalue = NULL;
    // 为变量请求内存
```

```
    pvalue = new double;
    // 为分配的地址赋值
    *pvalue = 3.14;
    // 打印内存地址
    cout << "Address of pvalue :" << pvalue << endl; // Address of pvalue
:0xd21790
    // 打印内存地址中存储的值
    cout << "Value of pvalue :" << *pvalue << endl; // Value of pvalue :3.14
    // 释放内存
    delete pvalue;
    return 0;
}
```

面向对象：类与对象

```
/**
 * C++ 面向对象: 类与对象
 *
 * C++ 在 C 语言的基础上增加了面向对象编程, C++ 支持面向对象程序设计.
 *
 * https://www.runoob.com/cplusplus/cpp-classes-objects.html
 */
#include "iostream"

using namespace std;

// 类定义
class Box {
public:
    double length;
    double breadth;
    double height;

    double get(void);
    void set(double len, double bre, double hei);
};

// 定义成员函数 get()
double Box::get() {
    return length * breadth * height;
}

// 定义成员函数 set()
void Box::set(double len, double bre, double hei) {
    length = len;
    breadth = bre;
    height = hei;
}

int main() {

    Box box1;
```

```
    Box box2;
    double volume = 0.0; // 面积

    box1.length = 1;
    box1.breadth = 2;
    box1.height = 3;
    volume = box1.length * box1.breadth * box1.height;
    cout << "box1 area: " << volume << endl; // box1 area: 6

    box2.set(1, 2, 3);
    volume = box2.get();
    cout << "box2 area: " << volume << endl; // box2 area: 6

    return 0;
}
```

面向对象：继承

```
/**
 * C++ 面向对象: 继承
 *
 * 一个类可以派生自多个类，这意味着，它可以从多个基类继承数据和函数.
 * 定义一个派生类，我们使用一个类派生列表来指定基类. 类派生列表以一个或多个基类命名，形式如下 :
 * class derived-class: access-specifier base-class
 *
 * https://www.runoob.com/cplusplus/cpp-inheritance.html
 */
#include "iostream"

using namespace std;

// 基类
class Shape {

protected:
    int width;
    int height;

public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
};

// 派生类
class Rectangle: public Shape {
public:
    int getArea() {
        return (width * height);
```

```
    }
};

// 主方法
int main() {
    Rectangle rectangle;
    rectangle.setWidth(10);
    rectangle.setHeight(20);

    // The area of rectangle: 200
    cout << "The area of rectangle: " << rectangle.getArea() << endl;

    return 0;
}
```

```
/**
 * C++ 面向对象: 继承
 *
 * 多继承即一个子类可以有多个父类, 它继承了多个父类的特性.
 * C++ 类可以从多个类继承成员, 语法如下:
 *
 * class <派生类名>:<继承方式1><基类名1>, <继承方式2><基类名2>, ...
 * {
 * <派生类类体>
 * };
 *
 * https://www.runoob.com/cplusplus/cpp-inheritance.html
 */
#include "iostream"

using namespace std;

// 基类 Shape
class  Shape {

protected:
    int width;
    int height;

public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
};

// 基类 PaintCost
class PaintCost {
public:
    int getCost(int area) {
        return area * 100;
    }
};

// 派生类继承 Shape 及 PaintCost
```

```
class Rectangle: public Shape, public PaintCost {
public:
    int getArea() {
        return (width * height);
    }
};

// 主方法
int main() {
    int area;
    Rectangle rectangle;

    rectangle.setHeight(10);
    rectangle.setWidth(20);

    area = rectangle.getArea();

    // The Total of area: 200
    cout << "The Total of area: " << rectangle.getArea() << endl;
    // The Total of paint cost: 20000
    cout << "The Total of paint cost: " << rectangle.getCost(area) << endl;

    return 0;
}
```

面向对象：重载运算符与函数

```
/**
 * C++ 面向对象: 重载运算符和重载函数
 *
 * 在同一个作用域内, 可以声明几个功能类似的同名函数,
 * 但是这些同名函数的形式参数(指参数的个数, 类型或者顺序)必须不同.
 * 您不能仅通过返回类型的不同来重载函数.
 *
 * https://www.runoob.com/cplusplus/cpp-overloading.html
 */
#include "iostream"

using namespace std;

class printData {
public:
    void print(int i) {
        cout << "Int value: " << i << endl;
    }
    void print(double f) {
        cout << "Double value: " << f << endl;
    }
    void print(char c[]) {
        cout << "Char value: " << c << endl;
    }
};
```

```
int main() {
    printData p;

    p.print(1); // Int value: 1
    p.print(1.1); // Double value: 1.1
    char c[] = "hi"; // Char value: hi
    p.print(c);

    return 0;
}
```

```
/**
 * C++ 面向对象: 重载运算符和重载函数
 *
 * 您可以重定义或重载大部分 C++ 内置的运算符. 这样, 您就能使用自定义类型的运算符.
 * 重载的运算符是带有特殊名称的函数, 函数名是由关键字 operator 和其后要重载的运算符符号构成的.
 * 与其他函数一样, 重载运算符有一个返回类型和一个参数列表.
 *
 * https://www.runoob.com/cplusplus/cpp-overloading.html
 */
#include "iostream"

using namespace std;

class Box {

private:
    double length;
    double breadth;
    double height;

public:
    double getVolume() {
        return length * breadth * height;
    }

    void setLength(double len) {
        length = len;
    }

    void  setBreadth(double bre) {
        breadth = bre;
    }

    void setHeight(double hei) {
        height = hei;
    }

    // 重载运算符 " + ", 用于把两个 Box 对象相加
    Box operator+(const Box& b) {
        Box box;
        box.length = this -> length + b.length;
        box.breadth = this -> breadth + b.breadth;
        box.height = this -> height + b.height;
        return box;
    }
```

```
};

int main() {
    Box b1;
    Box b2;
    Box b3;
    double volume = 0.0; // 体积

    // 初始化 b1 对象中的属性值
    b1.setLength(6.0);
    b1.setBreadth(7.0);
    b1.setHeight(5.0);

    // 初始化 b2 对象中的属性值
    b2.setLength(12.0);
    b2.setBreadth(13.0);
    b2.setHeight(10.0);

    // 输出 b1 的体积
    volume = b1.getVolume();
    cout << "Volume of Box1: " << volume << endl; // Volume of Box1: 210

    // 输出 b2 的体积
    volume = b2.getVolume();
    cout << "Volume of Box2: " << volume << endl; // Volume of Box2: 1560

    // 把两个对象相加得到 b3, 然后输出其体积
    b3 = b1 + b2;
    volume = b3.getVolume();
    cout << "Volume of Box3: " << volume << endl; // Volume of Box3: 5400

    return 0;
}
```

```
/**
 * C++ 面向对象: 多态
 *
 * C++ 多态意味着调用成员函数时, 会根据调用函数的对象的类型来执行不同的函数.
 */
#include "iostream"

using namespace std;

class Shape {
protected:
    int width, height;

public:
    Shape(int a = 0, int b = 0) {
        width = a;
        height = b;
    }
    // 虚函数是在基类中使用关键字 virtual 声明的函数.
    // 在派生类中重新定义基类中定义的虚函数时, 会告诉编译器不要静态链接到该函数.
    virtual int area() {
        cout << "Parent class area" << endl;
        return 0;
```

```
    }
};

class Rectangle: public Shape {
public:
    Rectangle(int a = 0, int b = 0): Shape(a, b ) {}
    int area() {
        cout << "Rectangle class area" << endl;
        return (width * height);
    }
};

class Triangle: public Shape {
public:
    Triangle(int a = 0, int b = 0): Shape(a, b) {}
    int area() {
        cout << "Triangle class area" << endl;
        return (width * height / 2);
    }
};

int main() {
    Shape *shape;
    Rectangle rectangle(10, 7);
    Triangle triangle(10, 5);

    // 获取矩形类的地址，并调用矩形求面积的函数 area()
    shape = &rectangle;
    shape -> area(); // Rectangle class area

    // 获取三角形类的地址，并调用三角形求面积的函数 area()
    shape = &triangle;
    shape -> area(); // Triangle class area

    return 0;
}
```

面向对象：数据封装

```
/**
 * C++ 面向对象：数据封装
 *
 * 封装是面向对象编程中的把数据和操作数据的函数绑定在一起的一个概念，
 * 这样能避免受到外界的干扰和误用，从而确保了安全.
 * 数据封装引申出了另一个重要的 OOP 概念，即数据隐藏.
 *
 * 通常情况下，我们都会设置类成员状态为私有(private)，除非我们真的需要将其暴露,这样才能保证良好的封装性.
 *
 * https://www.runoob.com/cplusplus/cpp-data-encapsulation.html
 */
#include "iostream"
```

```
using namespace std;

class Adder {
private:
    int total;

public:
    Adder(int i = 0) {
        total = i;
    }

    void addNum(int number) {
        total += number;
    }

    int getTotal() {
        return total;
    }
};

int main() {
    Adder adder;
    adder.addNum(10);
    adder.addNum(20);
    cout << "Total value :" << adder.getTotal() << endl; // Total value :30

    return 0;
}
```

面向对象：接口(抽象类)

```
/**
 * C++ 面向对象: 接口(抽象类)
 *
 * C++ 接口是使用抽象类来实现的，抽象类与数据抽象互不混淆，数据抽象是一个把实现细节与相关的数据分离开的概念.
 * 如果类中至少有一个函数被声明为纯虚函数，则这个类就是抽象类.
 *
 * https://www.runoob.com/cplusplus/cpp-interfaces.html
 */
#include "iostream"

using namespace std;

// 基类
class Shape {

protected:
    int width;
    int height;
```

```
public:
    // 提供接口框架的纯虚函数.
    // 基类 Shape 提供了一个接口 getArea(),
    // 在两个派生类 Rectangle 和 Triangle 中分别实现了 getArea().
    virtual int getArea() = 0;

    void setWidth(int w) {
        width = w;
    }

    void setHeight(int h) {
        height = h;
    }
};

// 派生类 Rectangle
class  Rectangle: public Shape {
public:
    int getArea() {
        return (width * height);
    }
};

// 派生类 Triangle
class Triangle: public Shape {
public:
    int getArea() {
        return (width * height) / 2;
    }
};

int main() {
    Rectangle rectangle;
    Triangle triangle;

    rectangle.setWidth(5);
    rectangle.setHeight(7);
    // The total area of rectangle: 35
    cout << "The total area of rectangle: " << rectangle.getArea() << endl;

    triangle.setWidth(5);
    triangle.setHeight(7);
    // The total area of triangle: 17
    cout << "The total area of triangle: " << triangle.getArea() << endl;

    return 0;
}
```

高级教程：文件和流

```cpp
/**
 * C++ 高级教程: 文件和流
 *
 * ofstream : 该数据类型表示输出文件流, 用于创建文件并向文件写入信息.
 * ifstream : 该数据类型表示输入文件流, 用于从文件读取信息.
 * fstream  : 该数据类型通常表示文件流, 同时具有 ofstream 和 ifstream 两种功能.
 * close    : 在程序终止前关闭所有打开的文件.
 *
 * https://www.runoob.com/cplusplus/cpp-files-streams.html
 */
#include "iostream"
#include "fstream"

using namespace std;

int main() {
    char data[100];

    // 以写模式打开文件
    ofstream write;
    write.open("test.dat");

    // 接收用户输入的数据
    cout << "Write the data to file" << endl;
    cout << "Enter your name: ";
    cin.getline(data, 100);
    // 将数据写入到文件中
    write << data << endl;

    // 再次接收用户输入的数据
    cout << "Enter you age: ";
    cin >> data;
    cin.ignore();
    // 将数据写入到文件中
    write << data << endl;

    // 关闭打开的文件
    write.close();

    // 以读模式打开文件
    ifstream read;
    read.open("test.dat");

    // 读取文件中的数据并输出
    cout << "Read the data from file" << endl;
    read >> data;
    cout << data <<endl;

    // 再次从文件中读取数据并输出
    read >> data;
    cout << data << endl;

    // 关闭打开的文件
    read.close();
```

```
    return 0;
}

// Write the data to file
// Enter your name:GoogTech
// Enter you age:24
// Read the data from file
// GoogTech
// 24s
```

高级教程：命名空间

```
/**
 * C++ 高级教程: 命名空间
 *
 * 命名空间这个概念专门用于解决上面的问题,
 * 它可作为附加信息来区分不同库中相同名称的函数、类、变量等.
 * 使用了命名空间即定义了上下文. 本质上,命名空间就是定义了一个范围.
 *
 * https://www.runoob.com/cplusplus/cpp-namespaces.html
 */
#include "iostream"

using namespace std;

// 自定义命名空间
namespace firstSpace {
    void func() {
        cout << "this is first space" << endl;
    }
}

namespace secondSpace {
    void func() {
        cout << "this is second space" << endl;
    }
}

int main() {

    // 调用自定义命名空间中的函数
    firstSpace::func(); // this is first space
    secondSpace::func(); // this is second space

    return 0;
}
```

```
/**
 * C++ 高级教程: 命名空间
 *
 * 您可以使用 using namespace 指令,
```

```
 * 这样在使用命名空间时就可以不用在前面加上命名空间的名称.
 * 这个指令会告诉编译器, 后续的代码将使用指定的命名空间中的名称.
 *
 * https://www.runoob.com/cplusplus/cpp-namespaces.html
 */
#include "iostream"

using namespace std;

namespace firstSpace {
    void func() {
        cout << "this is first space" << endl;
    }
}

using namespace firstSpace;

int main() {

    func(); // this is first space
    return 0;
}
```

```
/**
 * C++ 高级教程: 命名空间
 *
 * 命名空间可以嵌套, 您可以在一个命名空间中定义另一个命名空间.
 *
 * https://www.runoob.com/cplusplus/cpp-namespaces.html
 */
#include "iostream"

using namespace std;

namespace firstSpace {
    void func() {
        cout << "this is first space" << endl;
    }

    namespace innerSpace {
        void func() {
            cout << "this is inner space" << endl;
        }
    }
}

using namespace firstSpace;

int main() {

    // 调用自定义命名空间 firstSpace 中的函数
    func(); // this is first space

    // 调用自定义命名空间 innerSpace 中的函数
    firstSpace::innerSpace::func(); // this is inner space

    return 0;
```

```
}
```

高级教程：模板

```
/**
 * C++ 高级教程: 模板
 *
 * 模板是泛型编程的基础，泛型编程即以一种独立于任何特定类型的方式编写代码.
 * 模板是创建泛型类或函数的蓝图或公式.
 * 库容器，比如迭代器和算法，都是泛型编程的例子，它们都使用了模板的概念.
 *
 * 模板函数的一般形式如下所示:
 * template <typename type> ret-type func-name(parameter list)
 * {
 *  // 函数的主体
 * }
 *
 * https://www.runoob.com/cplusplus/cpp-templates.html
 */
#include "iostream"

using namespace  std;

// 自定义函数模板，返回不同类型数据中的最大值
template <typename T>
inline T const& Max(T const& a, T const& b) {
    return a < b ? b : a;
}

int main() {

    cout << Max(1, 2) << endl; // 2
    cout << Max(1.1, 2.2) << endl; // 2.2
    cout << Max("A", "B") << endl; // A

    return 0;
}
```

```
/**
 * C++ 高级教程: 模板
 *
 * 也可以定义类模板，泛型类声明的一般形式如下所示:
 * template <class type> class class-name {
 * }
 *
 * 向量(Vector)是一个封装了动态大小数组的顺序容器(Sequence Container).
 * 跟任意其它类型容器一样，它能够存放各种类型的对.
 * 可以简单的认为，向量是一个能够存放任意类型的动态数组.
 *
 * https://www.runoob.com/cplusplus/cpp-templates.html
 */
```

```cpp
#include "iostream"
#include "vector"
#include "string"
#include "stdexcept"

using namespace std;

// 定义类 Stack<>, 并实现泛型方法来对元素进行入栈出栈操作
template <class T>
class Stack {

private:
    vector<T> element;

public:
    void push(T const&); // 添加元素
    void pop(); // 弹出元素
    T top() const; // 查看栈顶元素
    bool isEmpty() const { // 判断栈是否为空
        return element.empty();
    }
};

template <class T>
void Stack<T>::push(const T & e) {
    element.push_back(e);
}

template <class T>
void Stack<T>::pop() {
    if(isEmpty()) {
        throw out_of_range("Pop error: Stack is empty");
    }
    element.pop_back();
}

template <class T>
T Stack<T>::top() const {
    if (isEmpty()) {
        throw out_of_range("Top error: Stack is empty");
    }
    return element.back();
}

int main() {
    try {
        Stack<int> intStack; // int 类型的栈
        Stack<string> stringStack; //string 类型的栈

        // 操作 int 类型的栈
        intStack.push(1);
        cout << intStack.top() << endl; // 1
        intStack.pop();

        // 操作 string 类型的栈
        stringStack.push("googtech");
        cout << stringStack.top() << endl; // googtech
        stringStack.pop();
```

```
        stringStack.pop(); // Pop error: Stack is empty

    }catch (exception const& msg) {
        cerr << msg.what() << endl;
        return -1;
    }
}
```

高级教程：多线程

```
/**
 * C++ 高级教程: 多线程
 *
 * 多线程是多任务处理的一种特殊形式,
 * 多任务处理允许让电脑同时运行两个或两个以上的程序.
 *
 * https://www.cnblogs.com/DOMLX/p/10945309.html
 */
#include "iostream"
#include "chrono" // 用于时间延时
#include "thread" // 用于创建线程

using namespace std;

// 两线程共享的变量
int number = 1;

int firstThread() {
    while(number < 10) {
        cout << "First  Thread :" << number << endl;
        ++ number;
        this_thread::sleep_for(std::chrono::milliseconds(10)); // 间隔 10 毫秒
    }
    return 0;
}

int secondThread() {
    while(number < 10) {
        cout << "Second Thread :" << number << endl;
        ++ number;
        this_thread::sleep_for(std::chrono::milliseconds(10));
    }
    return 0;
}


int main() {

    // 创建两个线程
    thread t1(firstThread);
    thread t2(secondThread);
```

```
    // join()为阻塞线程,
    // 阻塞的目的就是让 Main 主线程等待一下创建的线程
    t1.join();
    t2.join();

    return 0;
}

// First  Thread :1
// Second Thread :2
// Second Thread :3
// First  Thread :4
// Second Thread :5
// First  Thread :6
// Second Thread :7
// First  Thread :8
// Second Thread :9
// First  Thread :10
```

致谢

以上部分代码参考了如下教程, Thanks .

- *https://www.bilibili.com/video/BV1kW411Y76d*
- *https://www.runoob.com/cplusplus/cpp-tutorial.html*

基础选择题

1. C++基本语法

1. C++源程序文件的缺省扩展名为(　)。
   A. cpp　　B. exe　　C. obj　　D. lik

   > 答案 : A

2. 由C++源程序文件编译而成的目标文件的缺省扩展名为(　)。
   A. cpp　　B. exe　　C. obj　　D. lik

   > 答案 : C

3. 由C++目标文件连接而成的可执行文件的缺省扩展名为(　)。
   A. cpp　　B. exe　　C. obj　　D. lik

   > 答案 : B

4. 编写C++程序一般需经过的几个步骤依次是(　　)。
   A. 编译、编辑、连接、调试
   B. 编辑、编译、连接、调试
   C. 编译、调试、编辑、连接
   D. 编辑、调试、编辑、连接

> 答案 : B

5. 在多文件结构的程序中，通常把类的定义单独存放于（　）中。
   A. 主文件　　B. 实现文件　　C. 库文件　　D. 头文件

> 答案 : D

6. 在多文件结构的程序中，通常把类中所有非内联函数的定义单独存放于（　）中。
   A. 主文件　　B. 实现文件　　C. 库文件　　D. 头文件

> 答案 : A

7. 在多文件结构的程序中，通常把含有main()函数的文件称为（　）。
   A. 主文件　　B. 实现文件　　C. 程序文件　　D. 头文件

> 答案 : C

8. 一个C++程序文件的扩展名为（　）。
   A. .h　　B. .c　　C. .cpp　　D. .cp

> 答案 : C

9. 在C++程序中使用的cin标识符是系统类库中定义的（　）类中的一个对象。
   A. istream　B. ostream　　C. iostream　D. fstream

> 答案 : A

10. 在C++程序中使用的cout标识符是系统类库中定义的（　）类中的一个对象。
   A. istream　B. ostream　　C. iostream　D. fstream

> 答案 : B

11. 程序中主函数的名字为（　）。
   A. main　　B. MAIN　　C. Main　　D. 任意标识符

> 答案 : A

12. 采用重载函数的目的是（　）。

A. 实现共享　　B. 减少空间

C. 提高速度　　D. 使用方便，提高可读性

> 答案 : D

13. int i=0,s=0; while(s<20) {i++; s+=i;}

A. 4　B. 5　C. 6　D. 7

> 题目有误, 请跳过哈~

14. 在下面循环语句中循环体执行的次数为（ A ）。

int i=0; do i++; while(i*i<10);

A. 4　B. 3　C. 5　D. 2

> 答案 : A

15. 当处理特定问题时的循环次数已知时，通常采用（　）来解决。

A. for循环　B. while循环　C. do循环　D. switch语句

> 答案 : A

16. 循环体至少被执行一次的语句为（　）。

A. for循环　B. while循环　C. do循环　D. 任一种循环

> 答案 : C

17. switch语句能够改写为（　）语句。

A. for　　B. if　　C. do　　D. while

> 答案 : B

18. 函数重载是指（　）。

A. 两个或两个以上的函数取相同的函数名，但形参的个数或类型不同

B. 两个以上的函数取相同的名字和具有相同的参数个数，但形参的类型可以不同

C. 两个以上的函数名字不同，但形参的个数或类型相同

D. 两个以上的函数取相同的函数名，并且函数的返回类型相同

> 答案 : A

19. 下列（　）的调用方式是引用调用。

A. 形参和实参都是变量　　B.形参是指针，实参是地址值

C. 形参是引用，实参是变量　D.形参是变量，实参是地址值

> 答案 : C

20. 用new运算符创建一个含10个元素的一维整型数组的正确语句是( )。
    A. int *p=new a[10];　　B. int *p=new float[10];
    C. int *p=new int[10];　　D. int *p=new int[10]={1,2,3,4,5}

> 答案 : C

21. 假定变量m定义为“int m=7;”，则定义变量p的正确语句为（ ）。
    A. int p=&m;　　B. int *p=&m;*　　C. int &p=m;　　D. int *p=m;

> 答案 : B
>
> 题目描述不清晰.

22. 变量s的定义为“char *s=”Hello world!”;”，要使变量p指向s所指向的同一个字符串，则应选取（ ）。
    A. char *p=s;　　B. char *p=&s;
    C. char *p;p*=s;　　D. char *p; p=&s;

> 答案 : A

23. 假定一条定义语句为“int a[10], x, *pa=a;”，若要把数组a中下标为3的元素值赋给x，则不正确的语句为（ ）。
    A. x=pa[3]　　B. x=*（a+3）*　　C. x=a[3]*　　D. *x=pa+3

> 答案 : D

24. 假定指针变量p定义为“int *p=new int(100);”，要释放p所指向的动态内存，应使用语句（ ）。
    A. delete p;　　B. delete *p;　　C. delete &p;　　D. delete []p;

> 答案 : A

---

2. 类的封装

1. 关于封装，下列说法中不正确的是（ ）。
    A. 通过封装，对象的全部属性和操作结合在一起，形成一个整体
    B. 通过封装，一个对象的实现细节被尽可能地隐藏起来（不可见）
    C. 通过封装，每个对象都成为相对独立的实体
    D. 通过封装，对象的属性都是不可见的

> 答案 : D

2. 面向对象方法的多态性是指（ ）。
    A. 一个类可以派生出多个特殊类
    B. 一个对象在不同的运行环境中可以有不同的变体
    C. 针对一消息，不同的对象可以以适合自身的方式加以响应

D. 一个对象可以是由多个其他对象组合而成的

> 答案 : C

3. 在一个类的定义中，包含有（ ）成员的定义。
A. 数据　B. 函数　C. 数据和函数　D. 数据或函数

> 答案 : C

4. 在类作用域中能够通过直接使用该类的（ ）成员名进行访问。
A. 私有　B. 公用　C. 保护　D. 任何

> 答案 : D

5. 在关键字public后面定义的成员为类的（ ）成员。
A. 私有　B. 公用　C. 保护　D. 任何

> 答案 : B

6. 在关键字private后面定义的成员为类的（ ）成员。
A. 私有　B. 公用　C. 保护　D. 任何

> 答案 : A

7. 当类中一个字符指针成员指向具有n个字节的存储空间时，它所能存储字符串的最大长度为( )。
A. n　B. n+1　C. n-1　D. n-2

> 答案 : C

8. 对于一个类的构造函数，其函数名与类名( )。
A. 完全相同　B. 基本相同　C. 不相同　D. 无关系

> 答案 : A

9. 对于一个类的`析构函数`，其函数名与类名( )。
A. 完全相同　B. 完全不同　C. 只相差一个字符　D. 无关系

> 答案 : C

10. 类的构造函数是在定义该类的一个( )时被自动调用执行的。
A. 成员函数　B. 数据成员　C. 对象　D. 友元函数

> 答案 : C

11. 类的`析构函数`是一个对象被( )时自动调用的。
A. 建立　B. 撤消　C. 赋值　D. 引用

> 答案 : B

12. 一个类的`构造函数`通常被定义为该类的(　)成员。
    A. 公用　　B. 保护　　C. 私有　　D. 友元

> 答案 : B

13. 一个类的`析构函数`通常被定义为该类的(　)成员。
    A. 私有　　B. 保护　　C. 公用　　D. 友元

> 答案 : C

---

3. 继承

1. 从一个基类派生出的各个类的对象之间(　)。
    A. 共享所有数据成员，每个对象还包含基类的所有属性
    B. 共享部分数据成员，每个对象还包含基类的所有属性
    C. 不共享任何数据成员，但每个对象还包含基类的所有属性
    D. 共享部分数据成员和函数成员

> 答案 : C

2. 如果是类B在类A的基础上构造，那么，就称（　）。
    A. 类A为基类或父类，类B为超类或子类
    B. 类A为基类、父类或超类，类B为派生类或子类
    C. 类A为派生类，类B为基类
    D. 类A为派生类或子类，类B为基类、父类或超类

> 答案 : B

3. C++的继承性允许派生类继承基类的（　）。
    A. 部分特性，并允许增加新的特性或重定义基类的特性
    B. 部分特性，但不允许增加新的特性或重定义基类的特性
    C. 所有特性，并允许增加新的特性或重定义基类的特性
    D. 所有特性，但不允许增加新的特性或重定义基类的特性

> 答案 : A

4. 派生类的成员函数可以直接访问基类的（　）成员。
    A. 所有　　B. 公有和保护　　C. 保护和私有　　D. 私有

> 答案 : B

5. 对于公有继承，基类的公有和保护成员在派生类中将( )成员。

A. 全部变成公有　　B. 全部变成保护

C. 全部变成私有　　D. 仍然相应保持为公有和保护

> 答案 : D

6. 对于公有继承，基类中的私有成员在派生类中将 ( )。

A. 能够直接使用成员名访问　　B. 能够通过成员运算符访问

C. 仍然是基类的私有成员　　D. 变为派生类的私有成员

> 答案 : C

7. 当保护继承时，基类的（ ）在派生类中成为保护成员，在类作用域外不能够通过派生类的对象来直接访问该成员。

A. 任何成员　　B.公有成员和保护成员

C. 保护成员和私有成员　　D.私有成员

> 答案 : B

8. 在定义一个派生类时，若不使用保留字显式地规定采用何种继承方式，则默认为（ ）方式。

A. 私有继承　　B. 非私有继承

C. 保护继承　　D. 公有继承

> 答案 : A

9. 建立包含有类对象成员的派生类对象时，自动调用构造函数的执行顺序依次为（ ）的构造函数。

A. 自己所属类、对象成员所属类、基类

B. 对象成员所属类、基类、自己所属类

C. 基类、对象成员所属类、自己所属类

D. 基类、自己所属类、对象成员所属类

> 答案 : C

10. 当派生类中有和基类一样名字的成员时，一般来说，（ ）。

A. 将产生二义性　　B. 派生类的同名成员将覆盖基类的成员

C. 是不能允许的　　D. 基类的同名成员将覆盖派生类的成员

> 答案 : B

11. C++中的`虚基类机制`可以保证：（ ）。

A. 限定基类只通过一条路径派生出派生类

B. 允许基类通过多条路径派生出派生类，派生类也就能多次继承该基类

C. 当一个类多次间接从基类派生以后，派生类对象能保留多份间接基类的成员

D. 当一个类多次间接从基类派生以后，其基类只被一次继承

> 答案 : D

12. 下列对派生类的描述中错误的说法是：（ ）。

A. 派生类至少有一个基类
B. 派生类可作为另一个派生类的基类
C. 派生类除了包含它直接定义的成员外，还包含其基类的成员
D. 派生类所继承的基类成员的访问权限保持不变

> 答案 : D

13. 派生类的对象对其基类中（ ）可直接访问。

A. 公有继承的公有成员
B. 公有继承的私有成员
C. 公有继承的保护成员
D. 私有继承的公有成员

> 答案 : A

---

4. 多态

1.在派生类中重新定义虚函数时，除了（ ），其他方面都必须与基类中相应的`虚函数`保持一致。

A.参数个数 B.参数类型 C.函数名称 D.函数体

> 答案 : D

2.在`重载一个运算符`时，其参数表中没有任何参数，这表明该运算符是（ ）。

A. 作为友元函数重载的1元运算符 B. 作为成员函数重载的1元运算符
C. 作为友元函数重载的2元运算符 D. 作为成员函数重载的2元运算符

> 答案 : B

3.在成员函数中进行`双目运算符重载`时，其参数表中应带有（ ）个参数。

A. 0 B. 1 C. 2 D. 3

> 答案 : B

4.`双目运算符重载`为普通函数时，其参数表中应带有（ ）个参数。

A. 0 B. 1 C. 2 D. 3

> 答案 : C

5.关于运算符重载，下列说法正确的是（ ）。

A. 重载时，运算符的优先级可以改变。
B. 重载时，运算符的结合性可以改变。
C. 重载时，运算符的功能可以改变。
D. 重载时，运算符的操作数个数可以改变。

> 答案 : C

6.关于运算符重载，下列说法正确的是（ ）。

A. 所有的运算符都可以重载。

B. 通过重载，可以使运算符应用于自定义的数据类型。

C. 通过重载，可以创造原来没有的运算符。

D. 通过重载，可以改变运算符的优先级。

> 答案 : B

7.假定M是一个类名，且M中重载了操作符=，可以实现M对象间的连续赋值，如“m1=m2=m3;”。重载操作符=的函数原型最好是（ ）。

A. int operaotor=(M);　　B. int operator=(M&);

C. M operator=(M&);　　D. M& operator=(M);

> 答案 : D

8.在重载一运算符时，若运算符函数的形参表中没有参数，则不可能的情况是（ ）。

A. 该运算符是一个单目运算符。

B. 该运算符函数有一个隐含的参数this。

C. 该运算符函数是类的成员函数。

D. 该运算符函数是类的友元函数。

> 答案 : D

9.在C++中，用于实现运行时多态性的是（ ）。

A）内联函数

B）重载函数

C）模板函数

D）虚函数

> 答案 : D

10.如果一个类至少有一个纯虚函数，那么就称该类为（ ）。

(a)抽象类　(b)派生类　(c)虚基类　(d)以上都不对

> 答案 : A

11.下列关于抽象类的说明中不正确的是（ ）。

(a)含有纯虚函数的类称为抽象类

(b)抽象类不能被实例化，但可声明抽象类的指针变量

(c)抽象类的派生类可以实例化

(d)纯虚函数可以被继承

> 答案 : C

12.下面描述中，正确的是（　）。

A．`virtual`可以用来声明虚函数
B．含有纯虚函数的类是不可以用来创建对象的，因为它是虚基类
C．即使基类的构造函数没有参数，派生类也必须建立构造函数
D．静态数据成员可以通过成员初始化列表来初始化

> 答案：A

13.关于`虚函数`的描述中，正确的是（）。

A．虚函数是一个静态成员函数
B．虚函数是一个非成员函数
C．虚函数即可以在函数说明定义，也可以在函数实现时定义
D．派生类的虚函数与基类中对应的虚函数具有相同的参数个数和类型

> 答案：D

14.要实现动态联编，可以通过（）调用`虚函数`。

A．对象指针
B．成员名限定
C．对象名
D．派生类名

> 答案：A

15.以下（）成员函数表示`纯虚函数`。

A．virtual int vf(int);
B．void vf(int)=0;
C．virtual void vf()=0;
D．virtual void vf(int) { };

> 答案：C

16.下列关于`动态联编`的描述中，错误的是（）。

A．动态联编是以虚函数为基础
B．动态联编是运行时确定所调用的函数代码的
C．动态联编调用函数操作是指向对象的指针或对象引用
D．动态联编是在编译时确定操作函数的

> 答案：D

基础编程题

1. 两数相加

方法一

```
/**
 * 1.两数相加
 *
 * 法一: 最简单的方法如下所示, 缺点为数据类型固定(可利用模板方法改进哈), 且无异常捕获.
 */
#include "iostream"

using namespace std;

int main() {
    int a, b;

    cout << "Please input a number: ";
    cin >> a;
    cout << "Please input a number again: ";
    cin >> b;

    cout << "The result: " << a + b;
}
```

```
// 运行结果如下:
Please input a number: 1
Please input a number again: 2
The result: 3
```

方法二

```
/**
 * 1.两数相加
 *
 * 法二: 利用函数模板进行改进, 这里就不加异常捕获了哈
 */
#include "iostream"

using namespace std;

// 自定义函数模板, 返回不同类型数据中的最大值
template <typename T1, typename T2>
inline T1 sum(T1 t1, T2 t2) {
    return t1 + t2;
}

int main() {
    cout << "Int type: "<< sum(1, 2) << endl;
    cout << "Double type: " << sum(1.1, 2.2) << endl;
    cout << "Int and Double type: " << sum(1, 2.2) << endl;
    return 0;
}
```

```
// 运行结果如下所示:
Int type: 3
Double type: 3.3
Int and Double type: 3
```

> 在 c/c++ 中, 为了解决一些频繁调用的小函数大量消耗栈空间(栈内存)的问题,特别的引入了 **inline** 修饰符,表示为内联函数.

---

2. 求 N 阶乘

方法一

```
/**
 * 2. 求 n 阶乘
 *
 * 法一: 最简单的方法是利用 for / while 循环求阶乘. 当然最好利用递归法哈
 */
#include "iostream"

using namespace std;

int main() {
    int n;
    long double result = 1; // int 类型易益出, 即错得结果0

    cout << "Please input a number: ";
    cin >> n;

    for (int i = n; i > 0; i --) {
        result =  result * i;
    }

    cout << result;

    return 0;
}
```

```
// 运行结果如下:
Please input a number: 5
The result: 120
```

方法二

```
/**
 * 2. 求 n 阶乘
 *
 * 法二: 利用递归求阶乘
 */
#include "iostream"
```

```
using namespace std;

inline int func(int n) {
    return (n == 1 || n == 0) ? n : n * func(n - 1);
}

int main() {
    cout << "The result: " << func(5);
}
```

```
// 运行结果如下:
The result: 120
```

---

3. swap()

题目要求: 求两个数交换函数`swap()`

方法一

```
/**
 * 3.求两个数得交换函数 swap()
 *
 * 法一: 利用引用 &, 其比指针易理解哈
 * 当实参占据内存较大时,用引用代替传值( 需要复制 )可提高效率
 */
#include "iostream"

using namespace std;

void swap(int &x, int &y) {
    int t = x; x = y; y = t;
}

int main() {
    int a = 1, b = 2;

    cout << "Before a: " << a << " and b: " << b << endl;
    swap(a, b);
    cout << "After a: " << a << " and b: " << b;

    return 0;
}
```

```
// 运行结果如下:
Before a: 1 and b: 2
After a: 2 and b: 1
```

方法二

```
/**
 * 3.求两个数得交换函数 swap()
 *
 * 法二: 利用指针
 */
#include "iostream"

using namespace std;

inline void swap(int *x, int *y) {
    int t = *x; *x = *y; *y = t;
}

int main() {
    int a = 3, b = 4;
    cout << "Before a value: " << a << " and b value: " << b << endl;
    // &a 将 a 值赋值给 x, x 为 int* 指针, 指向 a
    swap(&a, &b);
    cout << "After a value: " << a << " and b value: " << b;

    return 0;
}
```

```
// 运行结果如下:
Before a value: 3 and b value: 4
After a value: 4 and b value: 3
```

方法三

```
// 加减法: a = a + b; b = a - b; a = a - b;
// 缺点: 处理浮点型的时候有可能出现精度的损失

// 乘除法: a = a * b; b = a / b; a = a / b;
// 缺点: 在处理浮点型变量时也存在精度损失问题,而且乘除的溢出比加减法严重

// 异或法:
// a ^= b; // a = a ^ b
// b ^= a; // b = b ^ (a ^ b) = b ^ a ^ b = b ^ b ^ a = 0 ^ a = a
// a ^= b; // a = (a ^ b) ^ a = a ^ b ^ a = a ^ a ^ b = 0 ^ b = b
// 缺点: 异或法可以完成对整型变量的交换,对于浮点型变量它无法完成交换
```

swap源码分析

以下C++ 标准库源码来摘自`bits/move.h`

```
#if __cplusplus >= 201103L
#define _GLIBCXX_MOVE(__val) std::move(__val)
#else
#define _GLIBCXX_MOVE(__val) (__val)
#endif

template<typename _Tp>
inline void swap(_Tp& __a, _Tp& __b {

      _Tp __tmp = _GLIBCXX_MOVE(__a);
      __a = _GLIBCXX_MOVE(__b);
      __b = _GLIBCXX_MOVE(__tmp);
```

分析:

- 使用 `inline`, 即声明为内联函数来优化函数调用的开销
- 使用`右值引用(move)`, 进而节省拷贝所引发的不必要的资源浪费

---

4. 使用new运算符

题目要求: 使用 `new` 运算符生成一个数组, 其长度为6

```
int *p = new int[6];
```

> 删除所申请得空间 : `delete[] p`

---

5. 设计一个类

题目要求: 设计一个类, 包含得属性学号(`string ID`) 和姓名(`string name`)

```
/**
 * 5.设计一个类, 包含得属性学号(string ID) 和姓名 (string name)
 */
#include "iostream"

using namespace std;

class Student {
public:
    string id;
    string name;
public:
    void print() {
        cout << "id: " << id << " and name: " << name;
    }
};
```

```
int main() {
    Student student;
    student.id = "Z001";
    student.name = "GoogTech";
    student.print();
}
```

```
// 运行结果如下:
id: Z001 and name: GoogTech
```

---

6. 派生一个矩形类

题目要求：设计一个形状类, 其`派生`一个矩形类

```
#include "iostream"

using namespace std;

// 形状类
class Shape {
protected:
    int width;
    int height;
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
};

// 继承形状类的矩形类
class Rectangle: public Shape{
public:
    int getArea() {
        return width * height;
    }
};

int main() {
    Rectangle rectangle;
    rectangle.setHeight(10);
    rectangle.setWidth(20);

    cout << "The area of rectangle :" << rectangle.getArea() << endl;

    return 0;
}
```

```
// 运行结果如下:
The area of rectangle : 200
```

---

7. 重载"+"运算符

题目要求：编写程序定义类PT, 有数据成员ab, 为其重载 `+` 运算符, 实现两个PT类对象相加

```
/**
 * 7.编写程序定义类 PT, 有数据成员 ab,
 * 为其重载 `+` 运算符, 实现两个 PT 类对象相加.
 */
#include "iostream"

using namespace std;

class Box {
private:
    double length;
    double width;
public:
    double getAare() {
        return length * width;
    }

    void setLength(double len) {
        length = len;
    }

    void setWidth(double wid) {
        width = wid;
    }

    // 重载运算符 "+", 用于把两个 Box 对象相加
    Box operator+(const Box& b) {
        Box box;
        box.width = this -> width + b.width;
        box.length = this -> length + b.length;
        return box;
    }
};

int main() {
    Box box1;
    Box box2;
    Box box3;

    // 初始化 box1 对象中的属性值
    box1.setWidth(10);
    box1.setLength(20);

    // 初始化 box2 对象中的属性值
```

```
    box2.setWidth(30);
    box2.setLength(40);

    // 输出 box1, box2 的面积
    cout << "The area of box1: " << box1.getAare() << endl;
    cout << "The area of box2: " << box2.getAare() << endl;

    // 把两个对象相加得到 b3, 然后输出其体积
    box3 = box1 + box2;
    cout << "The area of box3: " << box3.getAare() << endl;

    return 0;
}
```

```
// 运行结果如下:
The area of box1: 200
The area of box2: 1200
The area of box3: 2400
```

基础程序分析题

阅读如下程序, 回答指定问题.

```
/**
 * 8.阅读程序如下程序, 回答指定问题.
 */
#include <iostream>

using namespace std;

class one {
public:
    virtual void f() {
        cout << "1";
    }
};

class two : public one {
public:
    two() {
        cout << "2";
    }
};

class three : public two {
public:
    void f() {
        two::f();
        cout << "3";
    }
};
```

```
int main() {
    // aa, bb not used
    one aa, *p;
    two bb;

    three cc;
    p = &cc;
    p -> f();

    return 0;
}
```

1. 上述程序中定义了几个类？这几个类具有怎样的关系？

   ```
   答: 定义了三个类, 其中 one 为基类, two 和 three 为派生类, 即 two 继承自 one, 而 three 继承自 two.
   ```

2. 面向对象具有哪三大特点？

   ```
   答: 那你要问 "封继多" 这个人了嘿嘿, 即封装、继承、多态.
   ```

3. 类 three 中, 函数`f()`是虚函数吗？为什么？

   ```
   答: 是, 因为函数名前有 virtual 关键字哈.
   ```

4. 类 two 中, 有函数`f()`吗？如果有, 其访问权限是怎么样的？

   ```
   答: 有的, 因为 two 类继承自 one 类. 其访问权限为 public.
   ```

5. main函数中, `p -> f()`的输出结果是什么？这体现了面向对象的哪个特点？

   ```
   答: 2213

   two bb;
   (1).初始化 two 类时会自动调用 two 中的无参构造函数, 输出: 2

   three cc;
   (2).因 three 类继承自 two 类, 故初始化 three 类时会自动调用 two 类的无参构造函数, 输出: 2

   two::f();
   (3).调用 two 类中继承自 one 类的 f() 函数, 输出 1

   cout << "3";
   (4).最后输出: 3

   体现了面向对象的继承与多态的特点哈.
   ```

总结

我学过并使用过很多种编程语言, 如 `Java`、`Python`、`Golang`、`C`、`JS`... 我想说的是编程语言之间是互通的, 其本身并不重要, 它仅仅是一个可以帮助我们解决不同问题的工具而已, 重要的是要学会培养及锻炼自己的逻辑思维、以及动手解决实际问题的能力 .

之所以花几个小时去学习 `C++`, 是因为临近期末, 总结 `408科目` 中的数据结构与算法题时需要用到它 .