**Part I.**

1. 下列关于C++函数的叙述中，正确的是(  **C**  )。
2. 每个函数至少要具有一个参数   B. 每个函数都必须返回一个值
3. 函数在被调用之前必须先声明   D. 函数不能自己调用自己

2．下述概念中不属于面向对象的是（ **D** ）。

A．对象                       B．继承、多态

C．类、封装                     D．过程调用

3．下列说法中能准确地描述对象的是（**A**  ）。

A．对象是类的具体实例，可以操作数据的方法

B．对象是抽象的，类可以通过对象来生成

C．对象只是方法的集合

D．对象是一组具有共同的结构和行为的类

4．下面有关变量及其作用域的陈述哪一项是错误的？（**D**）

A．在方法里面定义的局部变量在方法退出的时候被撤销

B．局部变量只在定义它的方法内有效

C．在方法外面定义的实例变量在对象被构造时创建

D．在方法中定义的方法的参变量只要该对象被需要就一直存在

5．下列方法的声明中不合法的是（**B** ）。

A．float play(){ return 1; }

B．void play(int d,e) { }

C．double play(int d) { return 2.0; }

D．int play(int r) { return 1; }

6．下列哪个方法不能与方法void add(int a){ }重载？（**A** ）

A．int add(int b) { }

B．void add(double b) { }

C．void add(int a, int b) { }

D．void add(float g) { }

7．类Test定义如下：

class Test {

float use(float a, float b) {

}

<1. >

}

将以下哪种方法插入<1. >处是不合法的？（**B** ）

A．float use(float a, float b, float c) {  }

B．float use(float c, float d) {  }

C．int use(int a,int b) {  }

D．float use(int a, int b, int c) {  }

8．为了区分重载多态中同名的不同方法，要求（  **A**）。

A．采用不同的参数列表

B．返回值类型不同

C．调用时用类名或对象名做前缀

D．参数名不同

9．下列有关构造方法描述正确的是（   **D**）。

A．所有类都必须定义一个构造方法

B．构造方法必须有返回值

C．构造方法必须访问类的非静态成员

D．构造方法可以初始化类的成员变量

10．下列关于构造方法的叙述中，错误的是（**C**   ）。

A．构造方法名与类名必须相同

B．构造方法没有返回值，但不用void声明

C．构造方法不可以重载

D．构造方法只能通过生成对象时自动调用

11．设A为已定义的类名，下列声明对象a的语句中正确的是（**B**）。

A．A a =new A( );

B．A a = A( );

C．A a = new A;

D．a A;

12．给出如下类定义：

class Test {

public:

Test(int i) {

}

}

如果要创建一个该类的对象，正确的语句是（ **B**）。

A．Test t = Test();

B．Test t = Test(5);

C．Test t = Test("5");

D．Test t = Test(3.4);

13．以下代码的调试结果为（**B**  ）。

class Square {

public:

int a;

    void Square() {

        a = 10;

}

}

   int main() {

        Square s;

        cout << s.a << endl;

    }

A．输出10                       B．编译错误

C．输出0                        D．运行错误

14. 下列关于类和对象的叙述中，错误的是(  **A**  )。

  A)一个类只能有一个对象         B)对象是类的具体实例

  C)类是对某一类对象的抽象       D)类和对象的关系是一种数据类型与变量的关系

15. 11、实现运行时的多态性要使用(  D  )。

  A)重载函数    B)构造函数    C)析构函数    D)虚函数

**Part II**

1. 下列程序的输出结果为2，请将程序补充完整。

class Base

{ public:

\_\_1.\_\_ void fun()

{ cout<<1; }

};

class Derived: public Base

{ public

void fun( ) { cout<<2;

};

int main( )

{ Base \*p= new Derived;

p->fun( );

delete p;

return 0; }

1. **Virtual**
2. **以下程序是定义一个计数器类counter，对其重载运算符“+”，请填空。**

class counter

{ private: int n;

public:

counter() {n=0;}

counter(int i){n=i;}

\_\_2.\_\_ //运算符重载函数

{ counter t; t.n=n+c.n; return t; }

void disp() {cout<<"n="<<n<<endl;}

};

void main()

{ counter c1(5),c2(10),c3;

c3=c1+c2;

c1.disp(); c2.disp(); c3.disp(); }

**2.counter operator + (counter c)**

**Part III.**

1. 编写一个程序计算“三角形、正方形、圆形”三种图形的面积，要求：

a)抽象出一个基类base；b)在其中说明一个虚函数用来求面积；c)利用派生类定义“三角形、正方形、圆形”；d)编写主函数并测试。

1. #include <iostream>
3. **using** **namespace** std;
5. **class** base
6. { **public**:
7. **virtual** **float** area() **const**
8. { **return** 0; }
9. };
11. **class** triangle: **public** base
12. { **protected**:
13. **float** bian,height;
14. **public**:
15. triangle(**float** a=0, **float** h=0) { bian=a; height=h; }
16. **virtual** **float** area() **const**
17. { **return** bian\*height/2; }
18. };
20. **class** square: **public** base
21. { **protected**:
22. **float** bian;
23. **public**:
24. square(**float** a=0) { bian=a; }
25. **virtual** **float** area() **const**
26. { **return** bian\*bian; }
27. };
29. **class** circle: **public** base
30. { **protected**:
31. **float** radius;
32. **public**:
33. circle(**float** r=0) { radius=r; }
34. **virtual** **float** area() **const**
35. { **return** 3.14\*radius\*radius; }
36. };
38. **int** main()
39. { triangle t(3,4);  square s(5);  circle c(2);
40. base \*p;
41. p=&t; cout<<"triangle's area:"<<p->area()<<endl;
42. p=&s; cout<<"square's   area:"<<p->area()<<endl;
43. p=&c; cout<<"circle's   area:"<<p->area()<<endl;
44. **return** 0;
45. }

2.设计一个动物园管理系统，该系统需要能够管理动物园中的不同种类的动物，记录它们的种类、年龄、体重等信息，并提供以下功能：

1. 添加动物：能够添加新的动物到动物园中，并记录相关信息。
2. 显示所有动物：能够显示当前动物园中所有动物的信息。
3. 动物喂食：为所有动物分配食物，不同种类的动物需要的食物量不同。
4. 动物成长：模拟动物随时间成长的过程，增加其年龄和体重。
5. 搜索动物：根据种类或名称搜索特定的动物。

**要求**：

* 使用面向对象的设计方法，创建相应的类和对象。
* 动物类（Animal）应包含种类、年龄、体重等属性，并提供相应的访问和修改方法。
* 为每种具体的动物（如狮子、大象、企鹅等）创建派生类，并实现特定的食物需求量。
* 创建一个动物园类（Zoo），用于管理动物的添加、显示、喂食和搜索等操作。
* 系统应能够处理异常情况，如尝试添加已存在的动物或无效的输入。

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <unordered\_map>
6. **using** **namespace** std;
8. // 基类 Animal
9. **class** Animal {
10. **public**:
11. string species;
12. **int** age;
13. **float** weight;
15. **public**:
16. Animal(string s, **int** a, **float** w) : species(s), age(a), weight(w) {}
18. **virtual** ~Animal() {}
20. **virtual** **void** eat() = 0; // 纯虚函数，派生类需要实现
21. **void** grow() {
22. age++;
23. weight += 10; // 假设每年增加10kg体重
24. }
26. **void** displayInfo() **const** {
27. cout << "Species: " << species << ", Age: " << age << ", Weight: " << weight << endl;
28. }
30. string getName(){ **return** species; }
31. };
33. // 派生类 Lion
34. **class** Lion : **public** Animal {
35. **public**:
36. Lion(**int** a, **float** w) : Animal("Lion", a, w) {}
38. **void** eat() override {
39. cout << "Lion eats 20kg of meat." << endl;
40. // 假设狮子每次吃20kg食物
41. }
42. };
44. // 派生类 Elephant
45. **class** Elephant : **public** Animal {
46. **public**:
47. Elephant(**int** a, **float** w) : Animal("Elephant", a, w) {}
49. **void** eat() override {
50. cout << "Elephant eats 100kg of vegetation." << endl;
51. // 假设大象每次吃100kg食物
52. }
53. };
55. // 派生类 Penguin
56. **class** Penguin : **public** Animal {
57. **public**:
58. Penguin(**int** a, **float** w) : Animal("Penguin", a, w) {}
60. **void** eat() override {
61. cout << "Penguin eats 5kg of fish." << endl;
62. // 假设企鹅每次吃5kg食物
63. }
64. };
66. // 类 Zoo
67. **class** Zoo {
68. **private**:
69. unordered\_map<string, vector<Animal\*>> animals;
71. **public**:
72. **void** addAnimal(Animal\* animal) {
73. animals[animal->species].push\_back(animal);
74. }
76. **void** displayAllAnimals() {
77. **for** (**const** auto& pair : animals) {
78. cout << "Species: " << pair.first << endl;
79. **for** (Animal\* animal : pair.second) {
80. animal->displayInfo();
81. }
82. }
83. }
85. **void** feedAnimals() {
86. **for** (**const** auto& pair : animals) {
87. cout << "Feeding " << pair.first << "s:" << endl;
88. **for** (Animal\* animal : pair.second) {
89. animal->eat();
90. }
91. }
92. }
94. **void** ageOneYear() {
95. **for** (auto& pair : animals) {
96. **for** (Animal\* animal : pair.second) {
97. animal->grow();
98. }
99. }
100. }
102. Animal\* findAnimal(**const** string& species, **const** string& name) {
103. **for** (**const** auto& pair : animals) {
104. **for** (Animal\* animal : pair.second) {
105. **if** (animal->species == species && animal->getName() == name) {
106. **return** animal;
107. }
108. }
109. }
110. **return** nullptr;
111. }
112. };
114. **int** main() {
115. // 创建动物实例
116. Lion lion(5, 150);
117. Elephant elephant(10, 5000);
118. Penguin penguin(3, 20);
120. // 创建动物园实例
121. Zoo zoo;
123. // 添加动物到动物园
124. zoo.addAnimal(&lion);
125. zoo.addAnimal(&elephant);
126. zoo.addAnimal(&penguin);
128. // 显示所有动物
129. zoo.displayAllAnimals();
131. // 喂食所有动物
132. zoo.feedAnimals();
134. // 动物成长一年
135. zoo.ageOneYear();
137. // 再次显示所有动物
138. zoo.displayAllAnimals();
140. **return** 0;
141. }