Part I.

1．下列关于 C++函数的叙述中，正确的是(**C**)。

A. 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

C. 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>
2.
3.  using namespace std;
4.
5.  class base
6.  { public:
7.        virtual float area() const
8.        { return 0; }
9.  };
10.
11. class triangle: public base
12. { protected:
```

```
    float bian,height;
  public:
      triangle(float a=0, float h=0) { bian=a; height=h; }
      virtual float area() const
      { return bian*height/2; }
};

class square: public base
{ protected:
    float bian;
  public:
    square(float a=0) { bian=a; }
    virtual float area() const
    { return bian*bian; }
};

class circle: public base
{ protected:
    float radius;
  public:
    circle(float r=0) { radius=r; }
    virtual float area() const
    { return 3.14*radius*radius; }
};

int main()
{ triangle t(3,4);  square s(5);  circle c(2);
  base *p;
  p=&t; cout<<"triangle's area:"<<p->area()<<endl;
  p=&s; cout<<"square's   area:"<<p->area()<<endl;
  p=&c; cout<<"circle's   area:"<<p->area()<<endl;
  return 0;
}
```

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。

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>

using namespace std;

// 基类 Animal
class Animal {
public:
    string species;
    int age;
    float weight;

public:
    Animal(string s, int a, float w) : species(s), age(a), weight(w) {}

    virtual ~Animal() {}

    virtual void eat() = 0; // 纯虚函数，派生类需要实现
    void grow() {
        age++;
        weight += 10; // 假设每年增加 10kg 体重
    }

    void displayInfo() const {
        cout << "Species: " << species << ", Age: " << age << ", Weight: " << weight << endl;
    }

    string getName(){ return species; }
};
```

```cpp

// 派生类 Lion
class Lion : public Animal {
public:
    Lion(int a, float w) : Animal("Lion", a, w) {}

    void eat() override {
        cout << "Lion eats 20kg of meat." << endl;
        // 假设狮子每次吃 20kg 食物
    }
};

// 派生类 Elephant
class Elephant : public Animal {
public:
    Elephant(int a, float w) : Animal("Elephant", a, w) {}

    void eat() override {
        cout << "Elephant eats 100kg of vegetation." << endl;
        // 假设大象每次吃 100kg 食物
    }
};

// 派生类 Penguin
class Penguin : public Animal {
public:
    Penguin(int a, float w) : Animal("Penguin", a, w) {}

    void eat() override {
        cout << "Penguin eats 5kg of fish." << endl;
        // 假设企鹅每次吃 5kg 食物
    }
};

// 类 Zoo
class Zoo {
private:
    unordered_map<string, vector<Animal*>> animals;

public:
    void addAnimal(Animal* animal) {
        animals[animal->species].push_back(animal);
    }

```

```
    void displayAllAnimals() {
        for (const auto& pair : animals) {
            cout << "Species: " << pair.first << endl;
            for (Animal* animal : pair.second) {
                animal->displayInfo();
            }
        }
    }

    void feedAnimals() {
        for (const auto& pair : animals) {
            cout << "Feeding " << pair.first << "s:" << endl;
            for (Animal* animal : pair.second) {
                animal->eat();
            }
        }
    }

    void ageOneYear() {
        for (auto& pair : animals) {
            for (Animal* animal : pair.second) {
                animal->grow();
            }
        }
    }

    Animal* findAnimal(const string& species, const string& name) {
        for (const auto& pair : animals) {
            for (Animal* animal : pair.second) {
                if (animal->species == species && animal->getName() == name
) {
                    return animal;
                }
            }
        }
        return nullptr;
    }
};

int main() {
    // 创建动物实例
    Lion lion(5, 150);
    Elephant elephant(10, 5000);
    Penguin penguin(3, 20);
```

```
119.
120.     // 创建动物园实例
121.     Zoo zoo;
122.
123.     // 添加动物到动物园
124.     zoo.addAnimal(&lion);
125.     zoo.addAnimal(&elephant);
126.     zoo.addAnimal(&penguin);
127.
128.     // 显示所有动物
129.     zoo.displayAllAnimals();
130.
131.     // 喂食所有动物
132.     zoo.feedAnimals();
133.
134.     // 动物成长一年
135.     zoo.ageOneYear();
136.
137.     // 再次显示所有动物
138.     zoo.displayAllAnimals();
139.
140.     return 0;
141. }
```