3.3多态和虚函数

- 有时，需要针对同一问题的不同对象使用不同的处理方法，而不仅是简单地继承基类的定义。
- 多态是指属于不同类的对象对同样的消息可以作出完全不同的响应，即同一个消息可能会导致调用不同的方法。
- 多态可解决上述问题，多态有2种：
  - 编译时多态（静态多态）：在函数名或运算符相同的情况下，编译器在编译阶段就能够根据函数参数类型的不同来确定要调用的函数—通过重载实现。
  - 运行时多态（动态多态）：在函数名、函数参数和返回类型都相同的情况下，只能在程序运行时才能确定要调用的函数—通过虚函数实现。

基类指针指向派生类对象

- C++允许一个基类对象的指针指向其派生类的对象，但不允许派生类对象的指针指向其基类的对象。
- 将一个基类对象的指针指向其派生类的对象，通过该指针只能访问派生类中从基类继承的公有成员，不能访问派生类中新增加的成员，除非通过强制类型转换将基类指针转换为派生类指针。

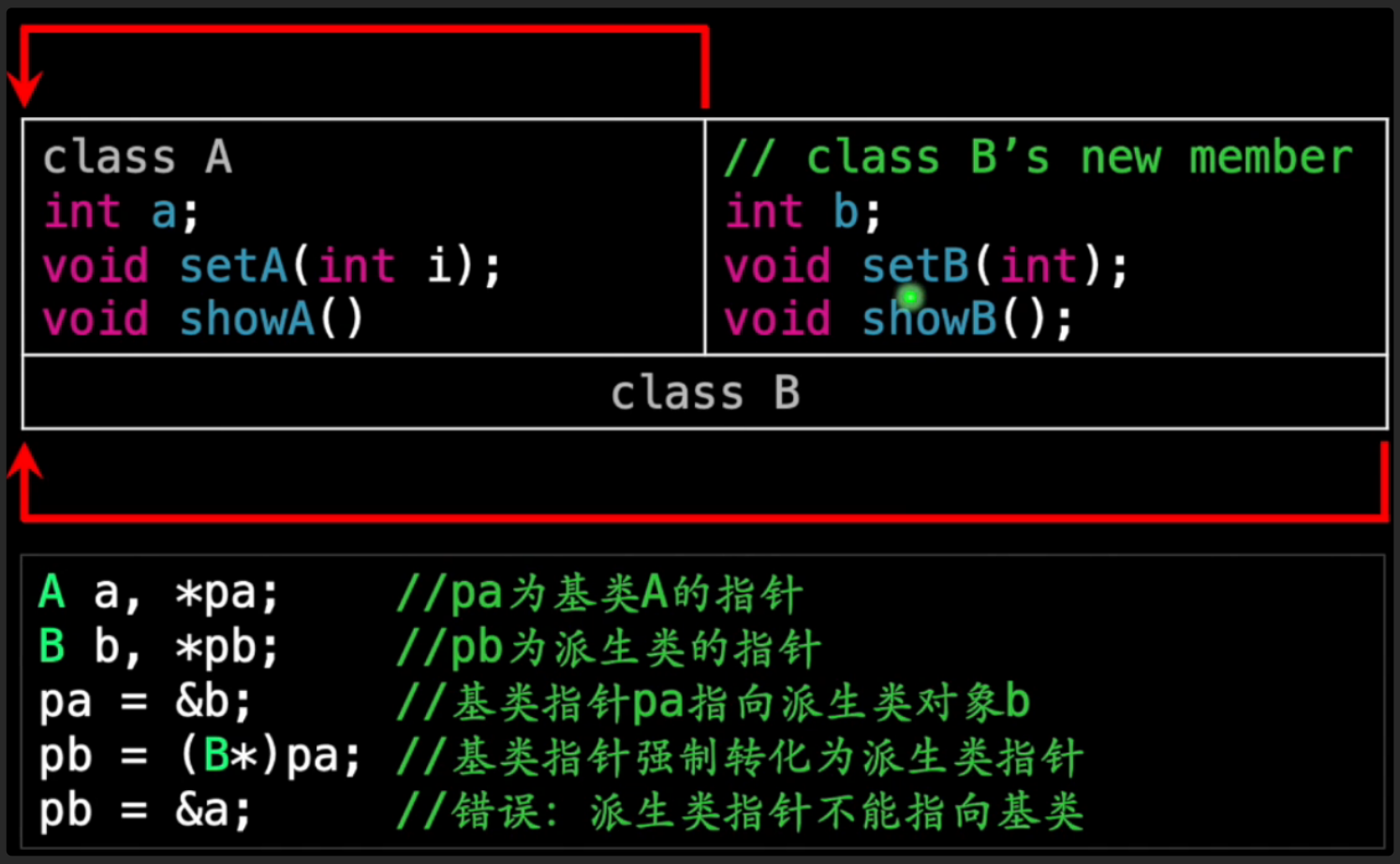
例子：基类指针指向派生类对象

```
#include <bits/stdc++.h>
using namespace std;

class A
{
private:
    int a;
public:
    void setA(int i) { a = i; }
    void showA() { cout <<"a = " << a << endl;}
};

class B : public A
{
private:
    int b;
public:
    void setB(int i) { b = i; }
    void showB() { cout << "b = " << b << endl; }
};

int main()
{
    A a, *pa; // pa为基类A的指针
    B b, *pb; // pb为派生类B的指针
    pa = &b; // 基类指针pa指向派生类对象b
    //通过基类指针pa访问B中从基类A继承的公有成员
    pa->setA(100);
    pa->showA();
    pb = (B*)pa; // 基类指针强制转化为派生类指针
    //不能通过基类指针pa访问派生类自己定义的成员
    pb->setB(200);
```

```
    pb->showB();
}

//以下语句错误
// pb = &a; // 基类指针不能指向基类对象
// pb->setB(200); // 基类指针不能访问派生类自己定义的成员
// pb->showB();
```

原理解析*

长度不同



- 思路：用基类指针指向派生类对象，通过基类指针来统一调用不同派生类的成员，从来实现多态。
- 问题：这样只能调用基类的成员，不能调用派生类的成员。

虚函数

- 基类指针指向派生类对象，只能调用基类成员。
- 基类指针指向派生类对象，通过虚函数，**可调用派生类成员**。
  - 此时，多态形成，运行时的多态，动态多态。
  - 同一个指针，可调用不同对象的**同名方法**，得到不同响应（执行不同的代码），对同一消息作出不同响应。
  - 为同一类继承结构中所有类的同一行为提供统一接口。虚函数声明形式如下：

```
virtual<函数类型><函数名>(<形参声明>)
```

- 声明为虚函数，意味着该成员函数在派生类中可能被重新定义。

例子：虚函数

```cpp
#include <bits/stdc++.h>
using namespace std;

class A
{
public:
    virtual void show() { cout << "A::show()" << endl; }
};
class B : public A
{
public:
    void show() { cout << "B::show()" << endl; }
};
class C : public B
{
public:
    void show() { cout << "C::show()" << endl; }
};

int main()
{
    A a, *pa;
    B b;
    C c;
    pa = &a; pa->show();
    pa = &b; pa->show();
    pa = &c; pa->show();
    return 0;
}
```

例子：虚函数 *

```cpp
#include <bits/stdc++.h>
using namespace std;

class CAnimal
{
public:
    virtual void run(){};
};

class CPerson : public CAnimal
{
    void run() { cout << "Person: Run on 2 Legs" << endl; }
};

class CDog : public CAnimal
{
    void run() { cout << "Dog: Run on 4 Legs" << endl; }
};

class CBug : public CAnimal
{
    void run() { cout << "Bug: Run on 6 Legs" << endl; }
};
```

```
24
25  void animalRun(CAnimal *pAnimal)
26  {   // some code
27      pAnimal->run();
28      // some code
29  }
30  int main()
31  {
32      CPerson person1;
33      CDog dog1;
34      CBug bug1;
35      animalRun(&person1);
36      animalRun(&dog1);
37      animalRun(&bug1);
38      return 0;
39  }
40
```

虚构造函数、虚析构函数

- 构造函数可以定义为虚函数吗?
  - **不能**，虚函数需要通过对应的虚指针vtable来调用。
- 析构函数的必要性
  - 释放对象时，只有定义为虚析构函数才能**保证**先释放派生类再释放基类。
- **建议：所有析构函数一律定义为虚函数。**

虚析构函数

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   class Base
5   {
6   public:
7       Base() { cout << "Base constructer." << endl; }
8       virtual ~Base() { cout << "Base destructor." << endl; }
9   };
10
11  class Derived : public Base
12  {
13  public:
14      Derived() { cout << "Derived constructer." << endl; }
15      virtual ~Derived() { cout << "Derived destructor." << endl; }
16  };
17
18  int main()
19  {
20      Base *pBase = new Derived;
21      delete pBase;
22      return 0;
23  }
```

内存泄露/僵尸内存

- 原因：没有把动态分配的内存完全释放掉

```
Base constructor.
Derived constructor.
Derived destructor.
Base destructor.
```

```
Base constructor.
Derived constructor.
Base destructor.
```

纯虚函数与抽象类

- 有时，在基类中无法给出基类中虚函数的实现代码，这时可以把虚函数声明为纯虚函数。

```
virtual<函数类型><函数名>(<形参声明>)=0;
```

- 纯虚函数没有函数体，仅**为类族提供一个统一的接口**。
- 含有纯虚函数的类称为**抽象类**，抽象类不能实例化，为了多态实现基类指针指向派生类对象

```
class CAnimal {
public:
    virtual void run() {};
};
```

```
class CAnimal {
public:
    virtual void run() = 0;
};
```

例子：纯虚函数与抽象类

```
#include <bits/stdc++.h>
using namespace std;

class CShape
{
protected:
    double s;
public:
    CShape() { s = 0; }
    virtual double area() = 0;
};
```

```
12
13  class CCircle : public CShape
14  {
15  private:
16      double r;
17  public:
18      CCircle(double r) { this->r = r; }
19      double area() { return 3.14159 * r * r; }
20  };
21
22  int main()
23  {
24      CShape* pShape;
25      CCircle circle(3);
26      pShape = &circle;
27      cout << "area = " << pShape->area() << endl;
28      return 0;
29  }
```

纯虚函数与抽象类语法规则

- 若必须定义为抽象类，而无合适的函数可作为纯虚函数，可将析构函数定义为纯虚函数。
- 为通过编译，必须在类外实现该纯虚函数。（有的编译系统不需要实现。）

```
1   class CShape
2   {
3   protected:
4       double s;
5   public:
6       CShape() { s = 0; }
7       virtual double {return 0; };
8       virtual ~CShape() {}
9   };
10  CShape::~CShape(){}
```