#include <iostream>

#include<string>

using namespace std;

class Base

{

public:

virtual string print() const

{

return "This is Base class";

}

};

class Derived : public Base

{

public:

virtual string print() const

{

return "This is Derived class";

}

};

**void describe(Base& p)**

{

cout << p.print() << endl;

}

int main()

{

Base b;

Derived d;

describe(b);

describe(d);

return 0;

}

**Output:**

This is Base class

This is Derived class

#include <iostream>

#include<string>

using namespace std;

class Base

{

public:

virtual string print() const

{

return "This is Base class";

}

};

class Derived : public Base

{

public:

virtual string print() const

{

return "This is Derived class";

}

};

**void describe(Base p)**

{

cout << p.print() << endl;

}

int main()

{

Base b;

Derived d;

describe(b);

describe(d);

return 0;

}

**Output**:

This is Base class

This is Base class

[Object slicing](http://en.wikipedia.org/wiki/Object_slicing) happens when a derived class object is assigned to a base class object, additional attributes of a derived class object are sliced off to form the base class object.

We can avoid above unexpected behavior with the use of pointers or references. Object slicing doesn’t occur when pointers or references to objects are passed as function arguments since a pointer or reference of any type takes same amount of memory.

#include <iostream>

#include<string>

using namespace std;

class Base

{

public:

virtual string print() const

{

return "This is Base class";

}

};

class Derived : public Base

{

public:

virtual string print() const

{

return "This is Derived class";

}

};

**void describe(Base \*p)**

{

cout << p**->**print() << endl;

}

int main()

{

Base **\*b** = new Base();

Derived **\*d** = new Derived();

describe(b);

describe(d);

return 0;

}

**Output:**

This is Base class

This is Derived class