Contents

Polymorphism sample programs

Static Linking

Program:

```
// ------------------------- static linking ---------------------------------
-
#include <iostream>
using namespace std;


class base_account{
    public:
        base_account(){
            cout << "base class constructor" << endl;
        }
        void func(){
            cout << "base_account class function" << endl;
        }
```

```
};

class derived_account:public base_account{
    public:
        derived_account(){
            cout << "derived class constructor" << endl;
        }
        void func(){
            cout << "derived_account class function" << endl;
        }
};

void global_func(base_account obj){
    cout<< "global function" << endl;
    obj.func();
}



int main()
{
    cout << "========================" << endl;
    base_account base_obj;
    base_obj.func();
    global_func(base_obj);
    cout << "========================" << endl;
    derived_account derived_obj;
    derived_obj.func();
    global_func(derived_obj);
    cout << "========================" << endl;

    return 0;
}
```

Result:

```
========================
```

base class constructor

base_account class function

global function

base_account class function

```
========================
```

base class constructor

derived class constructor

derived_account class function

global function

base_account class function

==========

---

static binding with pointers

program:

```
// ---static binding with pointers -----


#include <iostream>

using namespace std;


class base_account{
    public:
        base_account(){
            cout << "base class constructor" << endl;
        }
        void func(){
            cout << "base_account class function" << endl;
        }
};

class derived_account:public base_account{
    public:
        derived_account(){
            cout << "derived class constructor" << endl;
        }
        void func(){
            cout << "derived_account class function" << endl;
        }
};

void global_func(base_account obj){
    cout<< "global function" << endl;
    obj.func();
}
```

```
int main()
{
    cout << "========================" << endl;
    base_account base_obj;
    base_obj.func();
    global_func(base_obj);
    cout << "========================" << endl;
    derived_account derived_obj;
    derived_obj.func();
    global_func(derived_obj);
    cout << "========================" << endl;
    base_account *ptr = new derived_account();
    ptr->func();
    cout << "========================" << endl;

    return 0;
}
```

Result:

```
========================
base class constructor
base_account class function
global function
base_account class function
========================
base class constructor
derived class constructor
derived_account class function
global function
base_account class function
========================
base class constructor
derived class constructor
base_account class function
========================
```

dynamic binding eg1

program:

```
//-------- dynamic binding  eg1 -----------

#include <iostream>

using namespace std;


class base_account{
    public:
        base_account(){
            cout << "base class constructor" << endl;
        }
        virtual void func() const{
            cout << "base_account class function" << endl;
        }
};

class derived_account:public base_account{
    public:
        derived_account(){
            cout << "derived class constructor" << endl;
        }
        void func() const{
            cout << "derived_account class function" << endl;
        }
};

void global_func(const base_account &obj){
    cout<< "global function" << endl;
    obj.func();
}



int main()
{
    cout << "================" << endl;
    base_account base_obj;
    //base_obj.func();
    global_func(base_obj);
    cout << "================" << endl;
    derived_account derived_obj;
    derived_obj.func();
    global_func(derived_obj);
    cout << "================" << endl;
```

```
    base_account *ptr = new derived_account();
    ptr->func();
    cout << "================" << endl;

    return 0;
}
```

Result:

==========

base class constructor

global function

base_account class function

==========

base class constructor

derived class constructor

derived_account class function

global function

derived_account class function

==========

base class constructor

derived class constructor

derived_account class function

==========

dynambic binding eg2 constructors

program:

```
// ------- dynambic binding eg2  constructors---------
#include <iostream>
using namespace std;


class base_account{
    public:
        base_account(){
            cout << "base class constructor" << endl;
        }
        virtual void func() const{
            cout << "base_account class function" << endl;
        }
};

class derived_account_a:public base_account{
    public:
        derived_account_a(){
            cout << "derived_a class constructor" << endl;
        }
        virtual void func() const{
            cout << "derived_account_a class function" << endl;
        }
};

class derived_account_b:public base_account{
    public:
        derived_account_b(){
            cout << "derived_b class constructor" << endl;
        }
        virtual void func() const{
            cout << "derived_account_b class function" << endl;
        }
};


void global_func(const base_account &obj){
    cout<< "global function" << endl;
    obj.func();
}



int main()
{
```

```
    cout << "================" << endl;
    base_account base_obj;
    //base_obj.func();
    global_func(base_obj);

    cout << "================" << endl;
    derived_account_a derived_obj;
    //derived_obj.func();
    global_func(derived_obj);

    cout << "================" << endl;
    base_account *ptr1 = new base_account();
    base_account *ptr2 = new derived_account_a();
    base_account *ptr3 = new derived_account_b();
    ptr1->func();
    ptr2->func();
    ptr3->func();

    cout << "================" << endl;

    delete ptr1;
    delete ptr2;
    delete ptr3;
    cout << "================" << endl;

    return 0;
}
```

Result:

==========

base class constructor

global function

base_account class function

==========

base class constructor

derived_a class constructor

global function

derived_account_a class function

==========

base class constructor

base class constructor

derived_a class constructor

base class constructor

derived_b class constructor

base_account class function

derived_account_a class function

derived_account_b class function

==========

==========

---

dynambic binding eg2 constructors & destructors

program:

```
// ------- dynambic binding eg2  constructors & destructors --------

#include <iostream>
using namespace std;

class base_account{
    public:
        base_account(){
            cout << "base class constructor" << endl;
        }
        virtual void func() const{
            cout << "base_account class function" << endl;
        }
        virtual ~base_account(){
            cout << "base class destructor" << endl;
        }
};

class derived_account_a:public base_account{
    public:
        derived_account_a(){
            cout << "derived_a class constructor" << endl;
        }
        virtual void func() const{
            cout << "derived_account_a class function" << endl;
        }
        virtual ~derived_account_a(){
            cout << "derived_account_a destructor" << endl;
        }
```

```
};

class derived_account_b:public base_account{
    public:
        derived_account_b(){
            cout << "derived_b class constructor" << endl;
        }
        virtual void func() const{
            cout << "derived_account_b class function" << endl;
        }
        virtual ~derived_account_b(){
            cout << "derived_account_b destructor" << endl;
        }
};

void global_func(const base_account &obj){
    cout<< "global function" << endl;
    obj.func();
}

int main()
{
    cout << "================" << endl;
    base_account *ptr1 = new base_account();
    base_account *ptr2 = new derived_account_a();
    base_account *ptr3 = new derived_account_b();
    ptr1->func();
    ptr2->func();
    ptr3->func();
    cout << "================" << endl;
    delete ptr1;
    delete ptr2;
    delete ptr3;
    cout << "================" << endl;

    return 0;
}
```

Result:

```
================
```

base class constructor

base class constructor

derived_a class constructor

base class constructor

derived_b class constructor

base_account class function

derived_account_a class function

derived_account_b class function

==========

base class destructor

derived_account_a destructor

base class destructor

derived_account_b destructor

base class destructor

==========