浙江大学
ZHEJIANG UNIVERSITY

2022 OOP辅学
四月网课

张道泽

2022.4.24

内联函数

在调用函数处用内联函数体的代码来替换，节省调用开销。

将成员函数设为内联函数
法一：隐式声明 直接将函数声明在类内部
法二：成员函数返回类型前冠以关键字inline

Eg:

```
class Coord{
public:
     void setCoord(int a, int b)
     { x=a; y=b;}
     int getx()
     { return x;}
     int gety()
     { retrun y;}
private:
     int x,y;
};
```

```
inline 返回类型 类名::成员函数名(参数表)
{
    // 函数体
}
```

1-3 The reason inline functions are introduced into the C++ is to reduce the complecity of space, i.e. to shorten the code. (2分)

引用

引用：变量的别名，对引用的操作与对变量直接操作完全一样

- 声明引用时，必须同时对其进行初始化
- 相当于目标变量有两个名称，且不能再把该引用名改为其他变量名的别名
- 声明一个引用，不是新定义一个变量，本身不是一种数据类型
- 引用本身不占存储单元。因此，&ra == &a
- 不能建立数组变量的引用

常引用：用这种方式，不能通过引用修改目标变量的值，达到了引用的安全性。

```
int    i;
double d;

int&    ri = i;
double& rd = d;
```

```
int a ;
const int &ra=a;

ra=1;  //错误
a=1;   //正确
```

传地址 & 传引用

指针：运行时可以改变其所指向的值，即可以重新指向另一个对象。

引用：一旦和某个对象绑定后就不再改变，但所指对象的内容可以改变。
受限制的指针、一个别名、原变量的“诅咒娃娃”

```
void myswap(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
}
int main() {
    int a = 1, b = 2;
    myswap(a, b);
    printf("%d %d", a, b); // 1 2
    return 0;
}
```

```
void myswap(int* a, int* b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
int main() {
    int a = 1, b = 2;
    myswap(&a, &b);
    printf("%d %d", a, b); // 2 1
    return 0;
}
```

```
void myswap(int &a, int &b) {
    int tmp = a;
    a = b;
    b = tmp;
}
int main() {
    int a = 1, b = 2;
    myswap(a, b);
    printf("%d %d", a, b); // 2 1
    return 0;
}
```

```
#include <iostream>
using namespace std;
int& f(int &i )
{
    i += 10;
    return i ;
}
int main()
{
    int k = 0;
    int& m = f(k);
        cout << k << "#";
    f(m)++;
        cout << k << endl;
        return 0;
}
```

子类对象的构造销毁

当ctor( )遇到继承……

创建子类对象时，执行顺序：
① 父类成员变量初始化
② 父类ctor( )
③ 子类成员变量初始化
④ 子类ctor( )

当dtor( )遇到继承……

销毁子类对象时，执行顺序：
① 子类dtor( )
② 子类成员变量销毁
③ 父类dtor( )
④ 父类成员变量销毁

The output of the code below is:

```
#include <iostream>
using namespace std;

class A {
public:
        A() { cout << 1; }
} a;

int main()
{
        cout << 2;
        A a;

        return 0;
}
```

The output of the code below is:

```
#include<iostream>
using namespace std;
class AA {
public:
    AA() { cout << 1; }
    ~AA() { cout << 2; }
};
class BB: public AA {
    AA aa;
public:
    BB() { cout << 3; }
    ~BB() { cout << 4; }
};
int main() {
    BB bb;
    return 0;
}
```

多态性

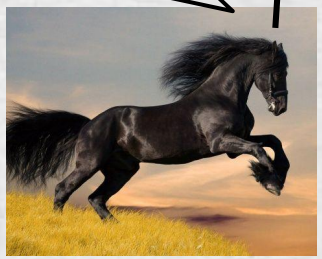
多态性：不同对象收到相同的消息时，产生不同的动作

- 静态多态性：函数重载和运算符重载
- 运行时的多态：继承时的虚函数

马类

调用成员函数：sayColor()

我是黑色的



黑马类

我是白色的

白马类

多态的实现：

- 函数重载
- 运算符重载
- 虚函数

virtual function

虚函数：函数调用与函数体之间的绑定关系，在运行时才建立

在运行时才决定如何动作（dynanmic binding 动态联编/动态绑定）

虚函数是动态联编的基础

声明：`virtual<函数类型><函数名>(<arg list>)`

动态绑定的条件：

- **父类声明虚函数**，**子类重写**了它
- 通过**指针/引用**来调用该函数

virtual function

```
class Grandam {
    public:
        virtual void introduce_self()
        { cout<<"I am grandam."<<endl; }
};

class Mother:public Grandam {
    public:
        void introduce_self
        { cout<<"I am mother."<<endl;}
};

class Daughter:public Mother {
    public:
        void introduce_self()
        { cout<<"I am daughter."<<endl;}
};
```

```
void main()
{
    Grandam* ptr;
    Grandam g;
    Mother m;
    Daughter d;

    ptr=&g;
    ptr->introduce_self();

    ptr=&m;
    ptr->introduce_self();

    ptr=&d;
    ptr->introduce_self();
}
```

virtual function

```
class Grandam {
    public:
        virtual void introduce_self()
        { cout<<"I am grandam."<<endl; }
};

class Mother:public Grandam {
    public:
        void introduce_self
        { cout<<"I am mother."<<endl;}
};

class Daughter:public Mother {
    public:
        void introduce_self()
        { cout<<"I am daughter."<<endl;}
};
```

```
void main()
{
    Grandam* ptr;
    Grandam g;
    Mother m;
    Daughter d;

    ptr=&g;
    ptr->introduce_self();

    ptr=&m;
    ptr->introduce_self();

    ptr=&d;
    ptr->introduce_self();
}
```

```
I am grandam.
I am mother.
I am daughter.
```

virtual function

```
class Grandam {
    public:
        virtual void introduce_self()
        { cout<<"I am grandam."<<endl; }
};

class Mother:public Grandam {
    public:
        void introduce_self
        { cout<<"I am mother."<<endl;}
};

class Daughter:public Mother {
    public:
        void introduce_self()
        { cout<<"I am daughter."<<endl;}
};
```

```
void main()
{
    Grandam* ptr;
    Grandam g;
    Mother m;
    Daughter d;

    ptr=&g;
    ptr->introduce_self();

    ptr=&m;
    ptr->introduce_self();

    ptr=&d;
    ptr->introduce_self();
}
```

virtual function

```
class Grandam {
    public:
        virtual void introduce_self()
        { cout<<"I am grandam."<<endl; }
};

class Mother:public Grandam {
    public:
        void introduce_self
        { cout<<"I am mother."<<endl;}
};

class Daughter:public Mother {
    public:
        void introduce_self()
        { cout<<"I am daughter."<<endl;}
};
```

```
void main()
{
    Grandam* ptr;
    Grandam g;
    Mother m;
    Daughter d;

    ptr=&g;
    ptr->introduce_self();

    ptr=&m;
    ptr->introduce_self();

    ptr=&d;
    ptr->introduce_self();
}
```

```
I am grandam.
I am grandam.
I am grandam.
```

virtual function

“同一类族中不同类的对象，对同一函数调用作出不同的响应”

Note：

- 子类重写父类的虚函数时，virtual可加可不加
- 使用 object.method() 也可以调用虚函数，但只能静态联编，不构成多态
- 虚函数是父类的非static的成员函数
- 内联函数、构造函数不能是虚函数；析构函数可以是虚函数

virtual function

```
class Grandam{
public:
    Grandam() { }
    virtual ~Grandam()
    {
        cout<<"This is Grandam::~Grandam()."<<endl;
    }
};
class Mother: public Grandam{
public:
    Mother() { }
    ~Mother()
    {
        cout<<"This is Mother::~Mother()."<<endl;
    }
};
```

纯虚函数

纯虚函数：

- 特殊的虚函数
- 在父类里：没有实现
- 在子类里：要么子类实现它，要么子类继续声明它是纯虚函数

声明：`virtual<函数类型><函数名>(arg list) = 0`

一个具有纯虚函数的类称为抽象类，抽象类不能被实例化

纯虚函数

```
class Polygon {
protected:
        int width, height;
public:
        void set_values (int a, int b) { width=a; height=b; }
        virtual int area (void) =0;
};

class Rectangle: public Polygon {
public:
        int area (void) { return (width * height); }

};

class Triangle: public Polygon {
public:
        int area (void) { return (width * height / 2); }
};
```

2-2 About virtual function, which statement below is correct? (2分)

A. Virtual function is a static member function
B. Virtual function is not a member function
C. Once defined as virtual, it is still virtual in derived class without virtual keyword,.
D. Virtual function can not be overloaded.

1-4 Dynamic binding is used as default binding method in C++. (1分)

1-2 An abstract class is a class with at least one pure virtual function. (1分)

2-6 Given:

```
class A {
        A() {}
        virtual f() = 0;
        int i;
};
```

which statement below is **NOT** true: (2分)

A. i is private
B. Objects of class A can not be created
C. i is a member of class A
D. sizeof(A) == sizeof(int)

```
#include <iostream>
using namespace std;

class A {
    A() {};
    virtual int f() {};
    int i;
};
int main()
{
    cout << sizeof(A) << endl;
    cout << sizeof(int) << endl;
}
```

```
C:\WIND
8
4
请按任意键
```

```
enum NOTE { middleC, Csharp, Cflat };
class Instrument {
public:
  virtual void play(NOTE) const = 0;
  virtual char* what() const = 0;
  virtual void adjust(int) = 0;
};

class Wind : public Instrument {
public:
  void play(NOTE) const {
    cout << 1 << endl;
  }
  char* what() const { return "Wind"; }
  void adjust(int) {}
};

class Percussion : public Instrument {
public:
  void play(NOTE) const {
    cout << 2 << endl;
  }
  char* what() const { return "Percussion"; }
  void adjust(int) {}
};
```

```
class Stringed : public Instrument {
public:
  void play(NOTE) const {
    cout << 3 << endl;
  }
  char* what() const { return "Stringed"; }
  void adjust(int) {}
};

class Brass : public Wind {
public:
  void play(NOTE) const {
    cout << 11 << endl;
  }
  char* what() const { return "Brass"; }
};

class Woodwind : public Wind {
public:
  void play(NOTE) const {
    cout << 12 << endl;
  }
  char* what() const { return "Woodwind"; }
};
```

```
void tune(Instrument& i) {
  i.play(middleC);
}

void f(Instrument& i) { i.adjust(1); }

int main() {
  Wind flute;
  Percussion drum;
  Stringed violin;
  Brass flugelhorn;
  Woodwind recorder;
  tune(flute);
  tune(drum);
  tune(violin);
  tune(flugelhorn);
  tune(recorder);
  f(flugelhorn);
  return 0;
}
```

```cpp
#include <iostream>

struct Base
{
  virtual ~Base()
  {
    std::cout << "Destructing Base" << std::endl;
  }
  virtual void f()
  {
    std::cout << "I'm in Base" << std::endl;
  }
};

struct Derived : public Base
{
  ~Derived()
  {
    std::cout << "Destructing Derived" << std::endl;
  }
  void f()
  {
    std::cout << "I'm in Derived" << std::endl;
  }
};
```

```cpp
int main()
{
  Base *p = new Derived();
  (*p).f();
  p->f();
  delete p;
}
```