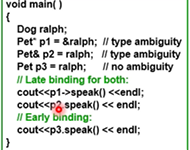
1. 常对象（所有值值不可变）只能调用常成员函数（eg.int t const()），且常成员函数不存在mutaul型变量值改变的情况;void t const()和void t拥有不同的签名，重载时可区分（分别用于对象和常对象调用）。
2. Static对象只能在类定义外初始化且不能再写static.（类似的还有default值只能在一处定义）
3. const int\*=int const \* int \*const const int\*const 三种注意区分

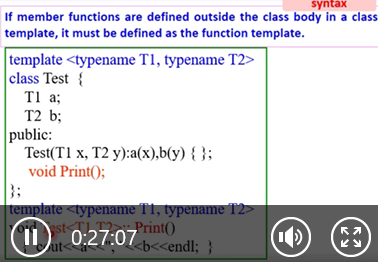
返回值类型为const int\*或const int\* const时，必须用const int\*或const int\* const的变量承接，而不能用int\*（也即不可变的返回值不能赋给一个可变的地址）。

1. 注意宏定义仅为替换，可能造成因括号缺省导致的错误。
2. 内联函数可以有效替代宏定义，弥补其不足的同时保证高效。
3. 类的子对象、常量、常引用只能放在初始化列表初始化，不能放在构造函数内部（如果子对象的构造函数没有缺省值，则必须要在初始化列表处赋值）。
4. 初始化的顺序与初始化列表的顺序无关，只与定义的顺序有关。
5. 引用的是地址空间，因此和地址一样可以读取、可以赋值。即int& f();a=f()√,f()=a√。
6. 当函数的参数是类对象或返回值为类对象（会产生匿名对象，但只有一次调用）的时候，需要拷贝构造函数。（编译器会提供缺省的拷贝构造函数，但当拷贝有地址的对象时会复制地址可能因此出错）
7. Class operator ++():前缀，++(int):后缀
8. CComplex（复数）重载+运算符时，a=b+c√，a=b+5也√，因为此时相当于a=b+CComplex
9. 重载强制转换时，如把CRational转化为CComplex时，要把强制转换operator CComplex()写在CRational类定义中。
10. Friend ostream& operator<<(ostream& os,const CComplex& cp){os<<......;return os;}
11. 静态存储区：全局对象、静态对象-》程序编译时分配空间。栈上空间：函数内部创建的局部对象。堆上空间：new、malloc创建的对象
12. 作为基类的类在使用之前要有完整定义。
13. 派生类的同名函数（仅名字相同，参数、返回值不同）会屏蔽掉基类函数。
14. A、C为基类，B为派生类，则B.A：：f（）的形式可调用A中函数。
15. 虚基类的构造函数由最派生类的构造函数调用。（虚基类构造函数的调用优于非虚基类）
16. 虚函数要求基类和派生类函数的接口完全一样（但派生类可不加virtual）
17. 
18. Overriding：动态绑定（虚函数） Overloading：静态绑定

动态绑定的调用函数的参数不能为const，而应该是类似于（A& a）；

如果基类虚函数给定默认值，派生类就不要给了。

必须通过指针或引用来操作对象实现动态多样性。

1. public:任何函数中都可以调用；protected:仅在自身函数和派生类中可以调用；private:仅在自身函数中可以调用。
2. 一个类中，所有的函数由一个指针指向。因此函数的多少不影响类的大小。Eg.A a，则&a存的就是vptr（指针）的地址。
3. 构造函数不适用虚函数，因为动态绑定会失效。而析构函数本身则经常为虚函数，以让内存得到释放。但构造函数和析构函数内部不存在虚机制。
4. 抽象类（abstract）中至少有一个纯虚函数。 virtual void f()=0和virtual void f(){}意义不同，后者是完整的。抽象类起到作为基类提供接口的作用。
5. 如果派生类不对基类中的纯虚函数进行完整定义，那么派生类仍为抽象类。
6. 

28.template <typename T>

T max(T a,int b)

{

if(a>b)return a;

return (T)b;

}

template <typename T>

T max(T a,T b)

{

if(a<b)return a;

return b;

} 如此定义时，若调用max（1,2）则会有歧义导致无法运行。

**函数重载、友元实例：**

#include <iostream>

#define M 3.14

using namespace std;

class B{

public:

int x,y;

B(int xx,int yy)

{

x=xx;y=yy;

}

};

class A

{

public:

int m,n;

A(int x=3,int y=2)

{

m=x;n=y;

}

A(int x=2)

{

m=n=x;

}

~A()

{

printf("!\n");

}

static int mm;

A& operator ++()

{

m++;n++;

return \*this;

}

int operator [](int i)

{

if(i==0)return m;else return n;

}

friend ostream& operator<<(ostream& os,A& a)

{

os<<a.m<<","<<a.n<<endl;

return os;

}

operator B()

{

return B(m,n);

}

};

int A::mm=3;

int main()

{

A a(1,2);

++a;

printf("%d\n",B(a).x);

cout<<a;

return 0;

}

**Virtual派生实例：**

#include <iostream>

using namespace std;

class person

{

public:

char sex;

void show()

{

printf("this %c's\n",sex);

}

person(char b)

{

sex=b;

}

};

class male:virtual public person

{

public:

int height;

void mshow()

{

printf("he is %dcm tall.\n",height);

}

male(int a):person('m')

{height=a;}

};

class female:virtual public person

{

public:

int height;

void fshow()

{

printf("she is %dcm tall.\n",height);

}

female(int a):person('f')

{height=a;}

};

class couple:public male,public female

{

public:

int year;

couple(int a,char b,int c):person(b),male(a+10),female(a)

{year=c;}

void showw()

{

printf("this couple has married for %d years\nthe male's height is%dcm\nthe female's height is%dcm\n''",year,male::height,female::height);

}

};

int main()

{

couple cc(165,'m',23);

cc.showw();

cc.show();

}

**动态多样性和template：**

#include <iostream>

using namespace std;

class base{

public:

int a,b;

virtual void show()

{

cout<<"base!"<<endl;

}

virtual ~base()

{

cout<<"base over"<<endl;

}

};

class A:public base

{

public: void show()

{

cout<<"A!"<<endl;

}

~A()

{

cout<<"A over"<<endl;

}

};

class B:public base

{

public: void show()

{

cout<<"B!"<<endl;

}

~B()

{

cout<<"B over"<<endl;

}

};

void pr(base& b)

{

b.show();

}

template<typename T>

class TT{

public:

T a;

void sh();

};

template<typename T>

void TT<T>::sh()

{

cout<<"OK!"<<endl;

}

int main()

{

A a;

B b;

pr(a);pr(b);

TT<int>t;

t.sh();

}

**数组结构实例**

#include <iostream>

#include <cstring>

#include <malloc.h>

using namespace std;

class CArray

{

private:

char\*str;

int length; // elements number

int capacity; // storage number

// Other variables are here.

public:

CArray(int a);

// Member functions are here.

void zeng(int m=10)

{

capacity+=10;

str=(char\*)realloc(str,strlen(str)\*sizeof(char)+10\*sizeof(char));

}

void Copy(char\* s)

{

int m=strlen(s);

while(m>capacity)zeng();

while(length<m)

{

str[length]=s[length];

length++;

}

}

void show()

{

printf("%s\n",str);

}

};

CArray::CArray(int a)

{

str=(char \*)malloc(sizeof(char)\*a);

length=0;

capacity=a;

}

int main()

{

CArray arr(10);

// “Hello world!” is stored in arr.

arr.Copy("Hello world!");

arr.show();

}