# C++ exercise code

## basic concept

### aboutarraypoint.cpp

|  |  |
| --- | --- |
| /\*  \* 关于复杂对象指针数组和数组指针  \* \*/  class MyClass  {  public:  int num;  char name[10];  };  int main()  {  //指针数组  MyClass \*parr = new MyClass[5];    //二级指针 | MyClass \*\*pparr = new MyClass\*[5]; //数组的每个元素都是一个指向MyClass的指针  //二维数组,数组每个元素都是MyClass  MyClass p2arr[5][5];  //指向数组的指针  MyClass (\*pp2arr)[5] = p2arr;  //强制转换将二级指针转换为指向数组指针  pp2arr = (MyClass (\*)[5])pparr;  //二位数主，每个数组的元素都是一个MyClass指针  MyClass \*p\_p2arr[5][5];  //声明指向数组的指针  MyClass \*(\*p\_pp2arr)[5] = p\_p2arr;  MyClass \*\*\*p\_pparr = (MyClass \*\*\*)p\_p2arr;  return 0;  } |

### 类型转换.cpp

|  |  |
| --- | --- |
| #include <iostream>  class Father  {  public:  int num;  void PrintName() {  std::cout << "this is father." << " num = " << num <<std::endl;  }  };  class Son : public Father  {  public:  int num;  void PrintName() {  std::cout << "this is father." << " num = " << num <<std::endl;  }  };  int main()  {  //指针没有初始化，就使用，这是危险的事  Father \*father;  Son \*son = reinterpret\_cast<Son \*> (father);  son->PrintName();  return 0;  }  int main03(int argc, char \*argv[])  {  //指针类型转换  int num = 5;  char \*p = reinterpret\_cast<char \*>(&num);  for(int i = 0; i < 4; i++) {  std::cout << static\_cast<int>(\*(p + i)) << " ";  } | std::cout <<"\n";  return 0;  }  int main02(int argc, char \*argv[])  {  const int num = 10;  int numarr[3] = {1,2,3};  const int \*p = numarr;  std::cout << \*p << " " << \*(p+1) << " "<< \*(p+2) <<std::endl;  //\*p = 10;  //\*(p+1) = 10; //报错，p为const类指针  int \*pnew = const\_cast<int \*>(p);  \*pnew = 10;  \*(pnew+1) = 11;  \*(pnew+2) = 12;  //去掉指针的const属性是成功的，但是对变量来说是不行的  //const变量是放在符号表中的，在编译的时候就确定了值，虽然在内存中修改  //变量的值，但是在运行的时候不是在内存中拿得，而是在寄存器取得  for(int i=0 ; i<3; i++) {  std::cout <<numarr[i] << std::endl;  }  //static\_cast<type>相当与c语言的中强制类转换  //std::cout << static\_cast<int>(100.0) << std::endl;  //去掉const属性  //int num01 = const\_cast<int> (num) ; //报错  return 0;  }  int main01(int argc, char \*argv[])  {  //static\_cast<type>相当与c语言的中强制类转换  std::cout << static\_cast<int>(100.0) << std::endl;    return 0;  } |

### virtualfunc.cpp

|  |  |
| --- | --- |
| /\*  \* 虚方法 虚析构函数可以解决父类指针指向子类delete发生的内存泄漏问题  \* 构造函数不能为虚析构函数  \* \*/  #include <iostream>  class Base  {  public:  int num;  void PrintNum() {  std::cout << "this is Base num = " << num << std::endl;  }  void Hi() {  std::cout << "Hi" << std::endl;  }  virtual void SayHello() {  std::cout << "virtual SayHello" << std::endl;  }  virtual ~Base() {  std::cout << "Base ～" << std::endl;  }  };  class Son : public Base  {  public:  void SayHello() {  std::cout << "this is Son SayHello" << std::endl;  }  virtual ~Son() {  std::cout << "~Son()" << std::endl;  }  };  class GreadSon : public Son  {  public:  int num;  void PrintNum() {  std::cout << "this is Greadson num = " << num << std::endl;  }  void GreadSonHi() {  std::cout << "Son Hi" << std::endl;  }  void SayHello() {  std::cout << "Greadson sayhello" << std::endl;  }  };  class SonSon : public GreadSon  {  public:  SonSon(int a, int b, int c)  {  this->a = a; | this->b = b;  this->c = c;  }  void SayHello() {  std::cout << "a = " << a << " b = " << b << " c = " << c << " " << std::endl;  }  void GetA() {  std::cout << " a = " << a << std::endl;  }  private:  int a;  int b;  int c;  };  int main()  {  Son son;  son.SayHello();  //指针父类调用父类的方法  //son.Base::SayHello();  //Base \*base = new Son;  //delete base;    //段异常错误 C语言类型的强制类型转换  //((GreadSon \*)base)->SayHello();  Base \*base(nullptr);  base = new GreadSon;  //base->SayHello();  base->PrintNum();  //用c++的强制类型转换则没有段错误  //(static\_cast<GreadSon \*>(base))->SayHello();    //虽然用把父类指针调用子类的特有的函数没有发生段异常错误，但是这是危险的事  //一旦这个方法中用到了子类的成员变量，就会可能发生down，但也能没有发生  //但这时危险的事  //(static\_cast<GreadSon \*>(base))->GreadSonHi();  //在ubuntu 14.04 LTS下测试没有down，但是在vs环境可能down  //(static\_cast<GreadSon \*>(base))->PrintNum();  //将父类指针转为子类指针  Base \*pbase = new Base;  GreadSon \*pson = static\_cast<GreadSon \*>(pbase);  //pson->PrintNum();  SonSon \*sonson = static\_cast<SonSon \*>(pbase);//new SonSon(1, 2, 3);  sonson->SayHello();  //在ubuntu 14.04 LTS 测试下没有问题，但是在vs下可能有问题存在  //sonson->GetA();  SonSon sonson02(1, 2, 3);  sonson02.GetA();  sonson02.Base::PrintNum();  return 0;  } |

### virtual\_class\_deep\_unstanderd.cpp

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  class Base {  public:  virtual void f() { cout << "Base::f" << endl;};  virtual void g() { cout << "Base::g" << endl;};  virtual void h() { cout << "Base::h" << endl;};  };  class Drived : public Base {  public:  virtual void f() { cout << "Drived::f" << endl;};  // virtual void g() { cout << "Drived::g" << endl;};  // virtual void h() { cout << "Drived::h" << endl;};  };  typedef void (\*Fun)();  int main(void)  {  Base b;  Fun pFun = NULL;  cout << "virtual table address:" << (int \*)(&b) << endl;  cout << "virtual table first function address:" << (int \*)\*(int \*)(&b) << endl;  pFun = (Fun)\*((int \*)\*(int \*)(&b) + 0);  pFun();  //attention: add 2 because in x64 OS, pointer sizese 8  pFun =(Fun)\*((int \*)\*(int \*)(&b) + 2);  pFun();  pFun =(Fun)\*((int \*)\*(int \*)(&b) + 4);  pFun();  Drived d;  pFun = (Fun)\*((int \*)\*(int \*)(&d) + 0);  pFun();  pFun = (Fun)\*((int \*)\*(int \*)(&d) + 2);  pFun();  pFun = (Fun)\*((int \*)\*(int \*)(&d) + 4);  pFun();    pFun = (Fun)\*((int \*)\*(int \*)(&d) + 6);  cout << pFun << endl;    //this also invoke drived f function.  (static\_cast<Base \*>(&d))->f();  return 0;  }  class Base1 {  public:  virtual void f() { cout << "Base1:f" << endl;};  virtual void g() { cout << "Base1:g" << endl;};  virtual void h() { cout << "Base1:h" << endl;};  };  class Base2 { | public:  virtual void f() { cout << "Base2:f" << endl;};  virtual void g() { cout << "Base2:g" << endl;};  virtual void h() { cout << "Base2:h" << endl;};  };  class Base3 {  public:  virtual void f() { cout << "Base3:f" << endl;};  virtual void g() { cout << "Base3:g" << endl;};  virtual void h() { cout << "Base3:h" << endl;};  };  class Derive : public Base1, public Base2, public Base3 {  public:  virtual void f() { cout << "Devive:f" << endl;};  //virtual void g() { cout << "Devive:g" << endl;};  };  int main02(void)  {  Fun pFun = NULL;  Derive d;  long \*\*pVtab = (long \*\*)&d;  //Base1's vtable  pFun = (Fun)pVtab[0][0];  pFun();  pFun = (Fun)pVtab[0][1];  pFun();  pFun = (Fun)pVtab[0][2];  pFun();  //the tail of vtable  pFun = (Fun)pVtab[0][3];  cout << pFun << endl;    //Base2's vtable  pFun = (Fun)pVtab[1][0];  pFun();  pFun = (Fun)pVtab[1][1];  pFun();  pFun = (Fun)pVtab[1][2];  pFun();    //Base3's vtable  pFun = (Fun)pVtab[2][0];  pFun();  pFun = (Fun)pVtab[2][1];  pFun();  pFun = (Fun)pVtab[2][2];  pFun();  pFun = (Fun)pVtab[2][3];  cout << pFun << endl;  return 0;  } |

### using.cpp

|  |  |
| --- | --- |
| \*  \* using 别名  \* \*/  #include <iostream>  namespace space //隔离模板，避免冲突  {  template<class T> using ptr = T\*; //模板的简写  }  int add(int a, int b)  {  return a+b;  } | typedef int(\*padd)(int a, int b);  using FUN = int (\*)(int a, int b); //利用using起别名  int main()  {  padd p = add;  std::cout << p(1, 2) << std::endl;  FUN fun = add;  std::cout << p(2, 2) << std::endl;  space::ptr<int> pint(new int(5));  std::cout << \*pint << std::endl;  return 0;  } |