第 一 章　　概述

1. 什么是对象？什么是面向对象方法？这种方法有哪些特点？

答：①**对象**是构成世界的一个独立单位，它具有自己的静态特征和动态特征。面向对象方法中的对象，是系统中用来描述客观事物的一个实体，它是用来构成系统的一个基本单位，由一组属性和一组行为构成。

②**面向对象的方法**将数据及对数据的操作方法放在一起，作为一个相互依存、不可分离的整体--对象。对同类型对象抽象出其共性，形成类。类中的大多数数据，只能用本类的方法进行处理。类通过一个简单的外部接口，与外界发生关系，对象与对象之间通过消息进行通讯。

2.什么叫做封装？

答：封装是面向对象方法的一个重要原则，就是把对象的属性和服务结合成一个独立的系统单位，并尽可能隐蔽对象的内部细节同时以特定权限访问类成员和方法。

第 二 章　　C++简单程序设计

1. 使用关键字 const 而不是#define 语句的好处有哪些？

答：const 定义的常量是有类型的，所以在使用它们时编译器可以查错；而且，这些变量在调试时仍然是可见的。

2. 在下面的枚举类型中，**Blue** 的值是多少？

enum COLOR { WHITE， BLACK = 100， RED， BLUE， GREEN = 300 };

答：Blue = 102

3. 在一个 **for** 循环中，可以初始化多个变量吗？如何实现？

答：在 for 循环设置条件的第一个";"前，用，分隔不同的赋值表达式。

for (x = 0， y = 10; x < 100; x++， y++) //不能像 C# 第一个;前 int x=0

4. 执行完下列语句后，**n** 的值为多少？

```
int n;
for (n = 0; n < 100; n++)
```

答： n 的值为 100

5. 什么叫做作用域？什么叫做局部变量？什么叫做全局变量？

答：作用域是一个标识符在程序正文中有效的区域。局部变量，一般来讲就是具有块作用域的变量；全局变量，就是具有文件作用域的变量。

6. 打印 **ASCII** 码为 **32~127** 的字符。

答：

```
#include <iostream.h>
int main()
{
for (int i = 32; i<128; i++)
cout << (char) i; //将 int 转换成 char 型即可输出数字对应 ASCII 码
return 0;
}
```

程序运行输出：

!"#$%G'()*+，./0123456789:;<>?@ABCDEFGHIJKLMNOP_QRSTUVWXYZ[\]^'abcdefghijklmnopqrstuvwxyz<|>~s

7.什么叫常量？什么叫变量？

答：所谓常量是指在程序运行的整个过程中其值始终不可改变的量，除了用**文字表示常量**外，**也可以为常量命名，这就是符号常量**；在程序的执行过程中其值可以变化的量称为变量，变量是需要用名字来标识的。

8. 变量有哪几种存储类型？

答：变量有以下几种存储类型：

1.auto 存储类型：采用堆栈方式分配内存空间，属于一时性存储，其存储空间可以被若干变量多次覆盖使用；

2.register 存储类型：存放在通用寄存器中；

3.extern 存储类型：在所有函数和程序段中都可引用；

4.static 存储类型：在内存中是以固定地址存放的，在整个程序运行期间都有效。

9.位操作 若 a = 1，b = 2，c = 3，下列各式的结果是什么？

1. a | b - c

2. a ^ b & -c

3. a & b | c

4. ! a | a

5. a >> 2

答:

1． -1 ？取-是什么意思?

2． 1 。 ^异或，位不同才为 1。1^2 = 0000 0001 ^ 0000 0010 = 0000 0011(3)

3-3 = 0???

3． 3 。1&2 = 0000 0001 & 0000 0010 = 0000 0011 （3）

3|3 = 0000 0011 | 0000 0011 = 0000 0011 （3）

4.！？？

5.0。0000 0010>>2，右移 2 位补 0（补码**正**数）=0000 0000（0）

10. 比较 **Break** 语句与 **Continue** 语句的不同用法。

答：Break 使程序从循环体和 switch 语句内跳出，继续执行逻辑上的下一条语句，不能用在别处；

continue 语句结束本次循环，接着开始判断决定是否继续执行下一次循环；

11. 定义一个表示时间的结构体，可以精确表示年、月、日、小时、分、秒；提示用户输入年、月、日、小时、分、秒的值，然后完整地显示出来。

答:

```
#include<iostream>
using namespace std ;
typedef struct
{
int year    ;    int month    ;int day; int hour ; int minute ; int second ;
```

```
} Time ;
void main()
{
Time *time = new Time() ;        //ATTENTION 这里是指针初始化或者 Time time;

cout << "请输入年：" ;    cin >> time->year ;

cout << "请输入月：";     cin >> time->month ;

cout << "请输入日：";     cin >> time->day ;

cout << "请输入时：";     cin >> time->hour ;

cout << "请输入分钟：" ;    cin >> time->minute ;

cout << "请输入秒：" ;       cin >> time->second ;}
```

12.定义枚举类型 **weekday**，**Sunday** 到 **Saturday** 七个元素在程序中定义 **weekday** 类型的变量，对其赋值，定义整型变量，看看能否对其赋 **weekday** 类型的值。

答:

```
#include <iostream.h>
enum weekday
{

Sunday，Monday，Tuesday, Wednesday,Thursday,Friday,Saturday

};
void main()
{
int i;
weekday d = Thursday;

cout << "d = " << d << endl; //①直接输出枚举类型是数字 4（从 0 计数）

i = d;                          //②枚举类型可以给整型赋值

cout << "i = " << i << endl; //输出  i = 4

d = (weekday)6;                //③整型类型可以强制转换为枚举类型

cout << "d = " << d << endl; //输出  d = 6
```

第 三 章　　函数

1.C++中的函数是？什么叫主调函数，什么叫被调函数，二者之间有什么关系？

答：C 和 C++语言中的子程序就体现为函数。调用其它函数的函数被称为主调函数，被其它函数调用的函数称为被调函数。一个函数很可能既调用别的函数又被另外的函数调用。

2.比较值调用和引用调用的相同点与不同点。

答：

值调用是指当发生函数调用时，给形参分配内存空间，并用实参来初始化形参。这一过程是参数值的单向传递过程，此后无论形参发生了怎样的改变，都不会影响到实参。

引用调用将引用作为形参，在执行主调函数中的调用语句时，系统自动用实参来初始化形参。这样形参就成为实参的一个别名，对形参的任何操作也就直接作用于实参。

3. 什么叫内联函数?它有哪些特点？

答：

概念：定义时使用关键字 inline 的函数叫做内联函数。

特点：①编译器在编译时在调用处用函数体进行替换,节省参数传递、控制转移等开销；

②内联函数体内不能有循环语句和 switch 语句；

③内联函数的定义必须出现在内联函数第一次被调用之前；

④对内联函数不能进行异常接口声明；

4.函数原型中的参数名与函数定义中的参数名以及函数调用中的参数名必须一致吗？

答：不必一致，所有的参数是根据位置和类型而不是名字来区分的。

5. 重载函数时通过什么来区分？

答： 重载的函数的函数名是相同的，但它们的参数的个数和数据类型不同。 返回类型不能区分!

6. 什么叫作嵌套调用？什么叫作递归调用？

答： 嵌套调用：如果函数 1 调用了函数 2，函数 2 再调用函数 3，便形成了函数的嵌套调用。递归调用：函数可以直接或间接地调用自身。

第 四 章　　类

1. 解释公有类型、私有类型、保护成员有些什么区别？

答:

①**公有类型成员**用 public 关键字声明，公有类型定义了类的外部接口；

②**私有类型的成员**用 private 关键字声明，只允许本类的函数成员来访问，而类外部的任何访问都是非法的。

③**保护类型的成员**用 protected 关键字声明，保护类型的性质和**私有类型的性质相似**，其差别在于继承和派生时**派生类**的成员函数**可以访问基类的保护成员**。

2.构造函数和析构函数有什么作用？

答:

构造函数的作用：就是在对象被创建时利用特定的值构造对象，完成的就是是一个从一般到具体的过程，构造函数在**对象创建时**由系统自动调用。

析构函数的作用：它是用来完成对象被删除前的一些清理工作，也就是专门作扫尾工作的。一般情况下，析构函数是在对象的**生存期即将结束的时**由系统自动调用的。

3. 数据成员可以为公有的吗？成员函数可以为私有的吗？

答: 可以，二者都是合法的。数据成员和成员函数都可以为公有或私有的。但数据成员最好定义为私有的。

4. 什么叫做拷贝构造函数？拷贝构造函数何时被调用？

答: 拷贝构造函数是一种特殊的构造函数，具有一般构造函数的所有特性。其**形参**是本

类的对象的引用，其**作用**是使用一个**已经存在的对象**，去**初始化一个新的同类的对象。**在以下三种情况下会被调用：

①在当用类的一个对象去初始化该类的另一个对象时；

②如果函数的形参是类对象，调用函数进行形参和实参结合时；

③如果函数的返回值是类对象，函数调用完成返回时。

5. 拷贝构造函数与赋值运算符(=)有何不同？

答：赋值运算符(=)作用于一个已存在的对象；

而拷贝构造函数会创建一个新的对象。

6. 设计一个用于人事管理的 People（人员）类。

要求：考虑到通用性，这里只抽象出所有类型人员都具有的属性：number（编号）、sex（性别）、birthday（出生日期）、id（身份证号）等等。其中"出生日期"定义为一个"日期"类内嵌**子对象**。用成员函数实现对人员信息的录入和显示。要求包括：构造函数和析构函数、拷贝构造函数、内联成员函数、带缺省形参值的成员函数、聚集。


```
#include <iostream>
using namespace std;  //ATTENTION1

class Date{
public:
    int year,month,day; //声明 public 方便对象访问，也可 private 用函数访问
    Date(){}
    Date(int y,int m,int d){
        year=y;month=m;day=d;
    }
};
enum SEX{MALE,FEMAL};//枚举
class Person{
private:
    int ID;
    SEX sex;
```

```
    Date birthday;
    char* IDCard;  //不用 string..
public:
    Person(int ID, SEX sex, int y, int m, int d, char* IDCard):birthday(y,m,d){  //ATTENTION 2: 同时也初始化 birthday，其他成员初始化就必须在{}中
        this->ID=ID;  //this 是指针用->
        this->sex=sex;
        this->IDCard = new char[100];
        strcpy(this->IDCard,IDCard);//ATTENTION ： 含指针成员的初始化
    }
    Person(Person& p){  //ATTENTION 3 ： 成员复制一个都不漏
        this->ID=p.ID;
        this->sex=p.sex;  //ATTENTION 4: 复制构造对象可以直接访问私有成员
        this->birthday=p.date; //类对象直接复制就好对吗？
        this->IDCard = new char[100];
        strcpy(IDCard,p.IDCard);//ATTENTION 5：含指针成员需深拷贝
                                //易错 1：这里定义数组 char IDCard[] 也需深拷贝
        //this->IDCard=p.IDCard;  //错误
    }
    ~Person(){  delete  IDCard}  //ATTENTION 5: 析构删指针
    //以下都是设置类属性的函数
    void setID(int ID){this->ID=ID;}
    void setSex(SEX sex){this->sex=sex;}
    void setDate(int y,int m,int d){
        birthday.year=y;
        birthday.month=m;
        birthday.day=d;
    }
    void setIDCard(char* IDCard){this->IDCard=IDCard;}
    void Show(){
        printf_s("ID: %d\nSex: %s\nBirthday: %d-%d-%d\nIDCard: %s\n",ID,0?"male":"femal",birthday.year,birthday.month,birthday.day,IDCard);
    }
};

int main(int argc, _TCHAR* argv[]){
    Person p(69,MALE,1994,10,31,"123456789");
    p.Show();
    system("pause");
    return 0;
}
```

7. 定义一个"数据类型" datatype 类，能处理包含字符型、整型、浮点型三种类型的数据，给出其构造函数。

```
#include <iostream.h>
class datatype{
enum //ATTENTION 1 :枚举成员记录当前是什么类型传入构造函数
     {  character,  integer,  floating_point
     } vartype;
union {  //ATTENTION 2:存储存入的对应类型值，只有最后一个被赋值成员生效
       char c;
       int i;
       float f;
};
public:
     datatype(char ch) { //构造函数 1：传入的是 char 类型
     vartype = character;
      c = ch;
}
      datatype(int i) {  //构造函数 2：传入的是 int 类型
      vartype = integer;
       this->i = i;
}
      datatype(float ff) { //构造函数 3：传入的是 float 类型
      vartype = floating_point;
      f = ff;
}  //class datetype
    void print();
};
void datatype::print() { //类外实现 print 函数
      switch (vartype) {
      case character:
      cout << "字符型: " << c << endl;  break;
      case integer:
      cout << "整型: " << i << endl;  break;
      case floating_point:
      cout << "浮点型: " << f << endl;  break;
}
}
void main() {
       datatype A('c'), B(12), C(1.44F);
       A.print();  //输出：字符型: c
       B.print();  //输出：整型: 12
       C.print();  //输出：浮点型: 1.44
}
```

8. 编写一个名为 CPU 的类，描述一个 CPU 的以下信息： 时钟频率，最大不会超过 3000MHZ = 3000*10^6 ；字长，可以 是 32 位或 64 位；核数，可以是单核，双核，或四核，是否支持超线程。各项信息要求使用位域来表示。

```cpp
#include<iostream>
#include<string>
using namespace std;
//ATTENTION 1 :和上不同 enum 定义在类外，方便类中使用位域
 //字长
 enum word{ a =32 , b = 64 };
 //核数
 enum keshu{one = 1 ,two = 2,four = 4};
 //是否支持超线程
 enum  es_no{yes ,no} ;

class intel_CPU{
  private :     // 2^29 = 536 870 912
    int tp : 29 ;  //ATTENTION 2: tp 是 int 类型但只用 29 为 bit 存储
    word  wd : 1  ;  //ATTENTION 3: 简单枚型底层是 int 类型
    keshu ks : 3 ; // ATTENTION 4: 应该是 3，需最大存储 4
    es_no en : 1 ;
  public :
   //构造函数
    intel_CPU(int tp=0,word wd=a,keshu ks=two,es_no en=no):
        tp(tp),wd(wd),ks(ks),en(en){
    };
    //析构函数
    ~intel_CPU();
     void show();
};
 void inline intel_CPU::show() {
     cout<<"时钟频率为: "<<ends;
     cout<<tp<<endl;
     cout<<"字长为【32 or 64】: ";
     cout<<wd<<endl;   // ATTENTION 5: 直接输出枚举类型，是数字
     switch (ks){
       case one :  cout<<"单核"<<endl; break;
       case two :  cout<<"双核"<<endl; break;
       case four:  cout<< "四核""<<endl; break;
       default :   cout<<"山寨机，没核" ; break;
    }
    cout<<"是否支持超线程模式: "<< (en==yes?"支持":"不支持")<<endl;
 }
```

第 五 章　　C++程序的基本结构

1. 什么叫做作用域？有哪几种类型的作用域？什么叫做可见性？

答:

①作用域: 讨论的是标识符的有效范围，作用域是一个**标识符**在程序正文中有效的区域。C++的作用域分为**函数原形**作用域、**块**作用域(局部作用域)、**类**作用域和**文件**作用域.

②可见性: 可见性是标识符是否可以引用的问题。

2. 什么叫做静态数据成员？它有何特点？

答: 类的静态数据成员是类的数据成员的一种特例，采用 static 关键字来声明。对于静态数据成员，每个类只要一个拷贝，**由所有该类的对象共同维护和使用**，这个共同维护、使用也就实现了同一类的不同对象之间的数据共享。

3.什么叫做静态函数成员？它有何特点？

答: 使用 static 关键字声明的函数成员是静态的，静态函数成员属于整个类，**同一个类的所有对象共同维护**，为这些对象所共享。静态函数成员具有以下两个方面的**好处**:

①静态成员函数**只能**直接访问**同一个类的静态数据成员**,可以保证不会对该类的其余数据成员造成负面影响;

②同一个类只维护一个静态函数成员的拷贝，**节约了系统的开销**，提高程序的运行效率。

③静态成员也可以声明是**静态**的。private　static　int　a;

4.定义一个 Cat 类，拥有静态数据成员 HowManyCats，记录 Cat 的个体数目；静态成员函数 GetHowMany（），存取 HowManyCats。

答:

```
#include <iostream.h>
class Cat
{
public:
    virtual ~Cat() { HowManyCats--; }//ATTENTION 1 :注意虚构函数时减少。

    static int GetHowMany() { return HowManyCats; }//创建对象时就++
private:
    static int HowManyCats;
};
int Cat::HowManyCats = 0;//ATTENTION 2：如何初始化静态函数成员为 0

void TelepathicFunction()//显示当前存在类对象个数
{
    cout << "There are " << Cat::GetHowMany() << " cats alive!\n";
}
int main()
{
    const int MaxCats = 5;
    Cat *CatHouse[MaxCats]; int i;
    for (i = 0; i<MaxCats; i++)
    {
        CatHouse[i] = new Cat(i);    TelepathicFunction();
    }
    for ( i = 0; i<MaxCats; i++)
    {
        delete CatHouse[i];         TelepathicFunction();
    }
    return 0;}
```

5. 什么叫做友元函数？什么叫做友元类？

答： ①**友元函数**是使用 friend 关键字声明的函数，在它的函数体中可以通过【该类对象.私有成员】访问该类私有成员。

②**友元类**是使用 friend 关键字声明的类，它的所有成员函数都是相应类的友元函数。 友元关系**不**具有**交换性**、**传递性**、**不能被继承**。

6.如果类 A 是类 B 的友元，类 B 是类 C 的友元，类 D 是类 A 的派生类，那么类 B 是类 A 的友元吗？类 C 是类 A 的友元吗？类 D 是类 B 的友元吗？

答： 类 B 不是类 A 的友元，友元关系不具有交换性；

类 C 不是类 A 的友元，友元关系不具有传递性；

类 D 不是类 B 的友元，友元关系不能被继承。

7. 在函数 fn1()中定义一个静态变量 n，fn1()中对 n 的值加 1，在主函数中，调用 fn1()十次，显示 n 的值。

答:

```
#include <iostream.h>
void fn1()
{
  //ATTENTION 1 :静态变量可以定义在函数里，且只定义一次，下次值还在!
    static int n = 0;n++;
    cout << "n 的值为" << n <<endl;
}
void main(){
    for(int i = 0; i < 10; i++)
    fn1(); //十次调用后  n = 1 0
    }
```

8. 定义类 X、Y、Z，函数 h(X*)，满足：类 X 有私有成员 i，Y 的成员函数 g(X*)是 X 的友元函数，实现对 X 的成员 i 加 1，类 Z 是类 X 的友元类，其成员函数 f(X*)实现对 X 的成员 i 加 5，函数 h(X*)是 X 的友元函数，实现对 X 的成员 i 加 10。在一个文件中定义和实现类，在另一个文件中实现 main()函数。

答:

```
#include "my_x_y_z.h"

// my_x_y_z.h 文件  //ATTENTION 0：这个文件包含三个类

#ifndef MY_X_Y_Z_H

class X;     //前向引用，X Y 互相引用

class Y {

   void g(X*); //ATTENTION 1 :由于要调用 X  私有成员 i，这里还没说明是友元

};
class X
{
private:       int i;
public:
      X(){i=0;}

      friend void h(X*); //ATTENTION 2：定义  本类函数成员 h(X*)是本类友元函数，传进

                                    来的对象指针 X*，就可以直接使用私有成员 i：X->i

      friend void Y::g(X*);    //定义 Y 类成员函数 g(x*)是本类友元函数

      friend class Z;       //定义 Z 类是本类友元

};
void h(X* x)    {    x->i =+10;    }
void Y::g(X* x) {    x->i ++;       }
class Z {
public:
void f(X* x) { x->i += 5; }
};
#endif // MY_X_Y_Z_H
```

9.定义 Boat 与 Car 两个类，二者都有 weight 属性，定义二者的一个友元函数 totalWeight()，计算二者的重量和。

答:

```
#include <iostream.h>

class Boat;   //错误 1:  注意前向引用

class Car
{
   private:
        int weight;
   public:
        Car(int j) {   weight = j;   }

        friend int totalWeight(Car &aCar,   Boat &aBoat);

};
class Boat
{
private:
      int weight;
public:
      Boat(int j ) {   weight = j;   }

      friend int totalWeight(Car &aCar,   Boat &aBoat); //ATTENTION 2 :能传引用指针最好!

};

int totalWeight(Car &aCar,   Boat &aBoat)

{
      return aCar.weight + aBoat.weight;
}
```


10. 如果在类模板的定义中有一个静态数据成员，则在程序运行中会产生多少个相应的静态变量？

答: 这个类模板的每一个实例类都会产生一个相应的静态变量。

※第 六 章　数组、指针与字符串

1. 数组 **A[10][5][15]**一共有多少个元素？

答： 10×5×15 = 750 个元素

2. 已知有一个数组名叫 **oneArray**，用一条语句求出其元素的个数。

答： **char** oneArray[10];

nArrayLength = sizeof(oneArray) / sizeof(oneArray[0]);

区分 nArrayLength = strlen(oneArray);

3. 用一条语句定义一个有 **5×3** 个元素的二维整型数组，并赋予 **1~15** 的初值。

答： **int theArray[5][3] = { 1，2，3，4，5，6，7，8，9，10，11，12，13，14，15 };**

或：int theArray[5][3] = { {1，2，3}， {4，5，6}， {7，8，9}， {10，11，12}，{13，14，15} };

4. 运算符*****和**&**的作用是什么？

答： *****：称为指针运算符，是一个一元操作符，表示**指针所指向的对象的值**；

&：称为取地址运算符，也是一个一元操作符，是用来得到一个对象的地址。

5. 什么叫做指针？指针中储存的地址和这个地址中的值有何区别？

答： **指针是一种数据类型**，具有指针类型的变量称为指针变量。指针变量存放的是另外一个对象的地址，这个地址中的值就是另一个**对象的内容**。

6. 定义一个整型指针，用 **new** 语句为其分配包含 **10** 个整型元素的地址空间。

答： int *pInteger = new int[10];

7. 在字符串”Hello，world!”中结束符是什么？

答： 是 NULL 字符 ？

8. 引用和指针有何区别？何时只能使用指针而不能使用引用？

答：引用是一个别名，不能为 NULL 值，不能被重新分配；

指针是一个存放地址的变量。当需要对变量重新赋以另外的地址或赋值为 NULL 时只能使用指针。

9. 给定 float 类型的指针 fp，写出显示 fp 所指向的值的输出流语句。

答： cout << "Value == " << *fp;

10. 程序中定义一个 double 类型变量的指针。分别显示指针占了多少字节和指针所指的变量占了多少字节。

答：

```
double *counter;
cout << "\nSize of pointer == "sizeof(counter);
cout << '\nSize of addressed value == "<<sizeof(*counter);
```

11. const int * p1 和 int * const p2 的区别是什么？

答：const int * p1 声明了一个**指向整型常量的指针** p1(类比常引用只读)，因此不能通过指针 p1 来改变它所指向的整型值；

int * const p2 声明了一个**指针型常量（**如数组名**）**，用于存放整型变量的地址，这个指针一旦初始化后，就不能被重新赋值了。

12. 定义一个整型变量 **a**，一个整型指针 **p**，一个引用 **r**，通过 **p** 把 **a** 的值改 **10**，通过 **r** 把 **a** 的值改为 **5**。

```
void main(){
   int a;
   int *p = &a;
   int &r = a;
   *p = 10;
   r = 5;
}
```

13. 下列程序有何问题，请改正；仔细体会使用指针时应避免出现的这个问题。

```
#include <iostream.h>
int Fn1();
int main()
{
int a = Fn1();
cout << "the value of a is: " << a;
return 0;
}
int Fn1()
{
int * p = new int (5);   //ATTENTION：还给它赋值了 5
return *p;
}
```


答： 此程序中给*p 分配的内存没有被释放掉。

改正： 主函数添加 delete a;

14. 声明一个参数为整型，返回值为长整型的函数指针；声明类 **A** 的一个成员函数指针，其参数为整型，返回值长整型。

答： long (* p_fn1)(int); //作参数也是这么写的

long (A::*p_fn2)(int);

15. 实现一个名为 SimpleCircle 的简单圆类，其数据成员 int *itsRadius 为一个指向其半径值的指针，设计对数据成员的各种操作。

```
#include <iostream.h>
class SimpleCircle
{
  public:
    SimpleCircle();
    SimpleCircle(int);
    SimpleCircle(const SimpleCircle &);
    ~SimpleCircle() {}
    void SetRadius(int);
    int GetRadius()const;
private:
    int *itsRadius;
};

SimpleCircle::SimpleCircle()  //默认构造函数
{
 //易错 0：有指针记得写默认构造函数，防止不分配内存
   itsRadius = new int(5); //如何给 int 型指针赋值
}

SimpleCircle::SimpleCircle(int radius)//带参数构造函数
{
//易错 1:*itsRadius = radius,指针使用一定要分配内存/指向其他已分配内存对象
   itsRadius = new int(radius);
}

SimpleCircle::SimpleCircle(const SimpleCircle & rhs)
{
   int val = rhs.GetRadius(); //如果指针 public 也可：*(rhs.Radius)
   itsRadius = new int(val);
}
int SimpleCircle::GetRadius() const
{
    return *itsRadius;
}
void main()
{
   SimpleCircle CircleOne, CircleTwo(9); //ATTENTION 1 ：调用带参构造函数!
}
```

16. 编写函数 reverse(char *s)的倒序递归程序，使字符串 s 倒序。

答:

```
#include <iostream.h>
#include <string.h>
/*
之前有想过写 reverse(int begin ,int end) ;但是获取不了对应元素，无法交换还是指针吧!
*/
void reverse(char *s, char *t)
{
   if (s >= t)    return;
   char c;
   c = *s;    *s = *t;*t = c; //交换位置
   reverse(++s, --t);
}
void reverse( char *s) //没办法题目要求..
{
   reverse(s, s + strlen(s) - 1);
}
```


17. 编写一个 3×3 矩阵转置的函数，在 main()函数中输入数据。

```
#include <iostream.h>
void BT (int matrix[3][3])
{
   int i, j, k;
/*          1 2 3
            4 5 6     j < i 精髓在于本来要遍历所有矩阵元素，变成只访问下三角元素，然
            7 8 9           后交换：  a10 a20 a21 --> 4   7   8
*/
   for(i=0; i<3; i++)
         for (j=0; j<i; j++) //交换值，即转置
         {
               k = matrix[i][j];      matrix[i][j] = matrix[j][i];      matrix[j][i] = k;
}
}
```

18. 编写一个矩阵转置的函数，矩阵的维数在程序中由用户输入。

答:

```
#include <iostream.h>

void move (int *matrix ,int n)    //一维数组实现多维数组

{
   int i, j, k;
   for(i=0; i<n; i++)
      for (j=0; j<i; j++)

         {    //注意这里如何获取元素

            k = *(matrix + i*n + j);

            *(matrix + i*n + j) = *(matrix + j*n + i);

            *(matrix + j*n + i) = k;

         }
}
```


19. 运行下面程序，观测执行结果，指出该程序是如何通过指针造成安全性隐患的，思考如何避免这种情况发生。

参考： https://blog.csdn.net/COCO56/article/details/80372261

第 七 章　　继承与派生

1. 比较类的三种继承方式 public 公有继承、protected 保护继承、private 私有继承之间的差别。

答：①公有继承，使得基类 public(公有)和 protected(保护)成员的访问属性在派生类中不变，而基类 private(私有)成员不可访问。

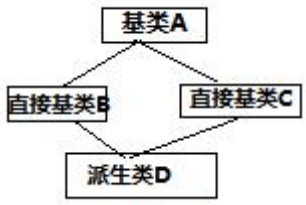
②私有继承，使得基类 public(公有)和 protected(保护)成员都以 private(私有)成员身份出现在派生类中，而基类 private(私有)成员不可访问。

③保护继承中，基类 public(公有)和 protected(保护)成员都以 protected(保护)成员身份出现在派生类中，而基类 private(私有)成员不可访问。

2. 派生类构造函数执行的次序是怎样的？

答： 派生类构造函数执行的一般次序为：调用基类构造函数；调用成员对象的构造函数（即初始化派生类对象新增成员，按声明顺序）；派生类的构造函数体中的内容。

3. 什么叫做虚基类？有何作用？



答：①定义： 当某**派生类**的部分或全部**直接基类**是从**间接基类(A)**派生而来，这些直接基类中，从上一级间接基类继承来的成员就拥有相同的名称，**派生类的对象**的这些同名成员在内存中同时拥有多个拷贝。我们可以用关键字 **virtual** 直接基类的共同基类设置为虚基类。

②作用：这时从不同的路径继承过来的该类成员在**派生类对象中**只拥有一个拷贝。

4. 定义一个 Shape 基类，在此基础上派生出 Rectangle 和 Circle，二者都有 GetArea()函数计算对象的面积。使用 Rectangle 类创建一个派生类 Square。

```
#include <iostream.h>
class Shape
{
   public:  //默认析构函数和默认构造函数可略，下同
//虚函数--使基类指针可以指向派生类对象：www.cnblogs.com/steven66/p/5013695.html
   virtual float GetArea() { return -1; }://ATTENTION 1：虚函数子类不必须实现
};
class Circle : public Shape
{
  public:
    Circle(float radius):itsRadius(radius){}
    float GetArea() { return 3.14 * itsRadius * itsRadius; }
  private:
    float itsRadius;
};
class Rectangle : public Shape
{
public:
    Rectangle(float len, float width): itsLength(len), itsWidth(width){};
    float GetArea() { return itsLength * itsWidth; }
private:
    float itsWidth;     float itsLength;
};
class Square : public Rectangle
{
public:
   Square(float len):Rectangle(len, len){}  //ATTENTION 2：委托构造函数
}
void main()
{  //why：用指针（引用也可）来测试基类指针指向派生类对象多态性？
   Shape * sp; //ANSWER:Shape sp = new Circle(5)；不会执向子对象，指针和引用多态
   sp = new Circle(5);
   cout << "The area of the Circle is " << sp->GetArea () << endl;
   delete sp;  //ATTENTION 3：只是删除对象而已，指针还在
   sp = new Rectangle(4, 6);
   cout << "The area of the Rectangle is " << sp->GetArea() << endl;
   delete sp;
   sp = new Square(5);
   cout << "The area of the Square is " << sp->GetArea() << endl;
   delete sp;
}
```

5. 定义一个 Document 类，有 name 成员变量，从 Document 派生出 Book 类，增加 PageCount 变量。

```
#include <iostream.h>
#include <string.h>
class Document
{
public:
    Document(){};
    Document( char *name );
    Document( const Document& D );
    ~Document();
    char *Name; // 易错 1：类中字符串还是定义动态指针比较好！
    void PrintNameOf(); // Print name.
};
Document::Document( char *name )  //ATTENTION 1 :有指针成员需构造函数初始化
{
    Name = new char[ strlen( name ) + 1 ]; //最后'\0'
    strcpy( Name, name );
};
Document::Document(const Document& D )//ATTENTION 2:指针成员需复制构造函数
{
    Name = new char[ strlen( name ) + 1 ]; //最后'\0'
    strcpy( Name, D.name );
};
Document::~Document()   //ATTENTION 3:指针成员需析构函数
{
      delete Name;
};
class Book : public Document
{
  public:
    Book( char *name, long pagecount );
    void PrintNameOf();
private:
    long PageCount;
};
Book::Book( char *name, long pagecount ):Document(name)
{
    PageCount = pagecount;//ATTENTION 4 :调用基类构造,派生类成员初始化写在{}
}
```

6. 定义基类 Base，有两个共有成员函数 fn1(),私有派生出 Derived 类，如果想在 Derived 类的对象中使用基类函数 fn1()，应怎么办？

```
class Base
{
   public:
        int fn1() const {     return 1;       }
};
class Derived : private Base
{
   public:
        int fn1() {     return Base::fn1();    }; //sao 操作，不明白还不如直接类名限定?
};
void main()
{
     Derived   a  ,   *p;
     a.fn1();     //或者：  a.Base::fn1();     p->Base::fn1();
}
```

7. 定义一个基类 BaseClass，从它派生出类 DerivedClass，BaseClass 有成员函数 fn1()，DerivedClass 也有成员函数 fn1()，在主程序中定义一个 DerivedClass 的对象，分别用 DerivedClass 的对象以及 BaseClass 和 DerivedClass 的指针来调用 fn1()，观察运行结果。


```
#include <iostream.h>
class BaseClass
{
  public:
    void fn1();  //非虚函数
};
void BaseClass::fn1()  {  cout << "调用基类的函数 fn1()" << endl;  }
class DerivedClass : public BaseClass
{
  public:
    void fn1();
};
void DerivedClass::fn1()  {  cout << "调用派生类的函数 fn1()" << endl;  }
void main()  {
   DerivedClass aDerivedClass;
   DerivedClass *pDerivedClass = &aDerivedClass; //指针一定要初始化
   BaseClass *pBaseClass = &aDerivedClass; //非虚函数父指针不动态绑定子对象
   aDerivedClass.fn1();   //调用派生类对象
   pBaseClass->fn1();    //ATTENTION 1：虽然被赋值子类对象，但还是调用基本对象
   pDerivedClass->fn1(); //调用派生类对象
}
```

8.组合和继承有什么共同点和差异？通过组合生成的类与被组合的类之间的逻辑关系是什么？继承呢？

答：组合和继承它们都使得已有对象成为新对象的一部分，从而达到代码复用的目的。

① 组合反映的是“有一个”（has-s）的关系，如类 B 含有类 A 对象，表示每一个 B 类型的对象都含“有一个”A 类型对象。A、B 对象是部分和整体的关系。

②继承反映的是”是一个“(is-a)的关系。如果类 A 是类 B 的公有基类，表示每一个 B 类型的对象都是一个 A 类型的对象，A、B 对象之间是特殊和一般的关系。

第 八 章　　多态性

1. 什么叫做多态性 ?在 C++中是如何实现多态的？

答： 多态是指同样的消息被不同类型的对象接收时导致完全不同的行为，是对类的特定成员函数的再抽象。

C++支持的多态有多种类型，**重载**(包括函数重载和运算符重载)和**虚函数**是其中主要的方式。

2. 什么叫做抽象类？抽象类有何作用？抽象类的派生类是否一定要给出纯虚函数的实现？

答：①定义：带有**纯虚**函数的类是抽象类。

②作用：通过它为一个类族建立一**公共的接口**，使它们能够更有效地发挥多态特性。

③实现：抽象类的派生类**并非一定**要给出纯虚函数的实现，如果派生类没有给出纯虚函数的实现，这个派生类仍然是一个抽象类。

3. 在 C++中，能否声明虚构造函数？为什么？能否声明虚析构函数？有何用途?

答： ①**不能**声明虚构造函数。

原因：多态是不同的对象对同一消息有不同的行为特性，虚函数作为运行过程中多态的基础，主要是**针对对象**的，而构造函数是在对象产生之前运行的。

②**可以**声明虚析构函数。

原因：析构函数的功能是在该类对象消亡之前进行一些必要的清理工作。析构函数设置为虚函数之后，在使用指针引用时可以**动态联编**，实现运行时的多态，保证使用**基类的指针**就能够调用适当的析构函数针对**不同的对象**进行清理工作。

4. 定义计数器 Counter 类，对其重载运算符 + 。

```
#include <iostream.h>
class Counter
{
public:
    Counter();
    Counter(int  initialValue);
    int  GetItsVal()const { return itsVal; }
    //非成员函数重载:
    //friend Counter operator+ (const Counter& ,const Counter& );
    Counter operator+ (const Counter &);
private:
    int  itsVal;
};
//构造函数
    Counter::Counter(int  initialValue):  itsVal(initialValue)  {  }
//委托构造函数
    Counter::Counter(): :Counter(0)  {  }
//1.以成员函数方式重载 +
   //为什么不返还引用？由于不能返回局部对象引用，则只能返回对象 A 或 B 的引用。
     //这样以后还会操作对象 A（B），并不是我们所期望的。（前置++就期望操作）
    Counter Counter::operator+ (const Counter & B)
    {
     return Counter(itsVal + B.GetItsVal());//易错 1：非友元不能访问私有
    }
/*
//2.以非成员方式重载 +
    Counter Counter::operator+ (const Counter & A,const Counter & B)
    {
     //易错 2：友元函数内所有对象可以访问私有成员，包括参数对象也可访问私有！
     return Counter(A.itsVal + B.itsVal );
    }
*/

int main()
{
    Counter varOne(2),  varTwo(4),  varThree;
    varThree = varOne + varTwo;
    cout << "varThree: " << varThree.GetItsVal() << endl;
    return 0;
}
//程序运行输出：    varThree: 6
```

5.定义一个 Shape 抽象类，在此基础上派生出 Rectangle 和 Circle，二者都有 GetArea()函数计算对象的面积，GetPerim()函数计算对象的周长。

```
class Shape
{
  public:
    Shape(){}
    virtual float GetArea() =0 ;//ATTENTION 1：声明纯虚函数才是抽象类!
};
class Circle : public Shape
{
  public:
    Circle(float radius):itsRadius(radius){}
    float GetArea() { return 3.14 * itsRadius * itsRadius; }
  private:
    float itsRadius;
};
class Rectangle : public Shape
{
  public:
  //易错 1：别忘构造，算我求你了
   Rectangle(float len,float width):itsLength(len),itsWidth(width){};
   virtual float GetArea() { return itsLength * itsWidth; }
  private:
   float itsWidth;
   float itsLength;
};
void main()
{
   Shape * sp;
   sp = new Circle(5);
   cout << "The area of the Circle is " << sp->GetArea () << endl;
   delete sp;
   sp = new Rectangle(4, 6);
   cout << "The area of the Rectangle is " << sp->GetArea() << endl;
   delete sp;
}
```

6. 对 Point 类重载++（自增）运算符（--（自减）同，此处代码略）。

```
#include <iostream.h>
class Point
{
   public:
     Point& operator++();
     Point operator++(int);
     Point(int x,int  y) ;
     Point() ;
   private:
     int _x, _y;
};
//构造函数
   Point(int x,int  y)：_x(x),_y(y){};
//委托构造函数：初始化为 0
   Point(int x,int  y)：point(0,0);
//前置++：point 对象 p 和 ++p 都自增了 xy
Point& Point::operator++()
{
    _x++;
    _y++;
   return *this;
}
//后置++ATTENTION 1：point 对象 p 自增了 xy ；++p 没有，它是 temp 对象复制。
Point Point::operator++(int)
{
   Point temp = *this;
   ++*this;
   return temp;
}
void main()
{
   Point A();  //调用委托构造函数，初始化 0,0
   A++;        //A++是 0,0 ，A 是 1,1
   ++A;        //A 和 A++都是 1,1
}
```

7. 写出下面程序的运行结果。

```
#include <iostream.h>
class BaseClass
{
   public:
      virtual void fn1();
      void fn2();  //ATTENTION 1 :fn2()不绑定
};
void BaseClass::fn1() {  cout << "调用基类的虚函数 fn1()" << endl;  }
void BaseClass::fn2() {  cout << "调用基类的非虚函数 fn2()" << endl; }
class DerivedClass : public BaseClass
{
   public:
     void fn1();
     void fn2();
};
void DerivedClass::fn1() { cout << "调用派生类的函数 fn1()" << endl; }
void DerivedClass::fn2() { cout << "调用派生类的函数 fn2()" << endl; }
int main()
{
   DerivedClass aDerivedClass;
   DerivedClass *pDerivedClass = &aDerivedClass;
   //问题：基类指针是动态绑定了派生类对象，那么就是相当使用派生对象吗？
   // 错。派生类对象有 virtual 关键字函数/成员才会动态绑定，其余依旧是基类对象的。
   BaseClass *pBaseClass = &aDerivedClass;
   pBaseClass->fn1(); //d f1
   pBaseClass->fn2(); //易错 1：基类指针不是相当派生类对象，调用派生类的 fn2()
   pDerivedClass->fn1(); //d f1
   pDerivedClass->fn2(); //d f2
   return 0;
}
```

程序运行输出：

调用派生类的函数 fn1()

调用基类的非虚函数 fn2()

调用派生类的函数 fn1()

调用派生类的函数 fn2()

8. 定义一个基类 BaseClass，从它派生出类 DerivedClass，BaseClass 中定义虚析构函数，在主程序中将一个 DerivedClass 的对象地址赋给一个 BaseClass 的指针，观察运行过程。

```
#include <iostream.h>
class BaseClass
 {
      public:
           virtual ~BaseClass()
           {
              cout << "~BaseClass()" << endl;
           }
};
class DerivedClass : public BaseClass
 {
      public:
           ~DerivedClass()
           {
              cout << "~DerivedClass()" << endl;
           }
};
void main()
{
      BaseClass* bp = new DerivedClass;
      delete bp;
}
```


答：唯一要注意的是，调用派生类的析构函数，会自动也调用基类的析构函数！

程序运行输出：

```
~DerivedClass()
~BaseClass()
```

9.定义 Point 类，有成员变量 X、Y，为其定义友元函数实现重载+，使得 int 类型对象可以和 point 类型对象相加。

```
#include <iostream.h>
class Point
{
  public:
    Point() { X = Y = 0; }
    Point( unsigned x, unsigned y ) { X = x; Y = y; }
    unsigned x() { return X; }
    unsigned y() { return Y; }
//ATTENTION 1 :int 类型对象不属于自定义，需重载非成员函数；且题中要求定义友元函
// 数就一定是重载非成员函数，因为友元函数一定非成员。
    friend Point operator+( Point& pt, int nOffset );
    friend Point operator+( int nOffset, Point& pt );
  private:
    unsigned X;
    unsigned Y;
};
//重载+: point 对象在前。为什么不返回引用？
//返回引用，由于不能返回局部对象引用，就只能返回 pt 引用。而我们也不不期望 pt 改变
Point operator+( Point& pt, int nOffset )
{
   Point ptTemp = pt;  //ATTENTION 2 :不能用*this,重载非成员没有当前操作对象
   ptTemp.X += nOffset;
   ptTemp.Y += nOffset;
   return ptTemp;
}
//重载+: point 对象在后。
Point operator+( int nOffset, Point& pt )
{
   Point ptTemp = pt;
   ptTemp.X += nOffset;
   ptTemp.Y += nOffset;
   return ptTemp;
}
void main()
{
  Point pt( 10, 10 );
  pt = pt + 5; // Point + int
  pt = 10 + pt; // int + Point
}
```

第 九 章　　异常处理

1. 什么叫做异常？什么叫做异常处理？

答： 当一个函数在执行的过程中出现了一些不平常的情况，或运行结果无法定义的情况，使得操作不得不被中断时，我们说出现了异常。异常通常是用 throw 关键字产生的一个对象，用来表明出现了一些意外的情况。

2. C++的异常处理机制有何优点？

答： C++的异常处理机制使得**异常的引发和处理不必在同一函数中**，这样底层的函数可以着重解决具体问题，而不必过多地考虑对异常的处理。上层调用者可以在适当的位置设计对不同类型异常的处理。

3. 举例 **throw** 、**try**、**catch** 语句的用法？

答： 1.throw 语句用来引发异常，用法为： throw 表达式;

例如： throw “异常发生！”；

2.try/ catch 语句

例如： try{ 语句 //可能会引发多种异常 }

catch（参数声明 1）{ 语句 //异常处理程序 }

4.练习使用 **try**、**catch** 语句，在程序中用 **new** 分配内存时，如果操作未成功，则用 **try** 语句触发一个字符型异常，用 **catch** 语句捕获此异常。

```
#include <iostream.h>
void main()
{
    char *buf;
try
{
    buf = new char[512];
    if( buf == 0 )      throw "内存分配失败!";
}
catch( char * str )
{
    cout << "有异常产生： " << str << endl;
}
```

}

5.定义一个异常类 CException，有成员函数 Reason()，用来显示异常的类型，定义函数 fn1()触发异常，在主函数的 try 模块中调用 fn1(),在 catch 模块中捕获异常，观察程序的执行流程。

```
#include <iostream.h>

class CException
{
public:
    const char *Reason() const { return "CException 类中的异常。"; }
};
//fn1:
void fn1()
{
   cout<< "在子函数中触发 CException 类异常" << endl;
   throw CException(); //有 trow 某类()!cacth 才能捕捉 CException 中发生异常!
}

//主函数
void main()
{
   cout << "进入主函数" << endl;
try
{
   cout << "在 try 模块中，调用子函数" << endl;
   fn1();
}
catch( CException E)//fn1 抛出 CException 类异常,才能捕捉 CException 中异常
{
   cout << "在 catch 模块中，捕获到 CException 类型异常："; 
   cout << E.Reason() << endl;
}
catch( char *str )     //捕捉其他异常
{
   cout << "捕获到其它类型异常：" << str << endl;
}
  cout << "回到主函数，异常已被处理" << endl;
}
```