C++程序设计期末复习

2014年考试题

一、简答题（本题满分30分，共三题）

1. 请简述C++程序设计语言的设计理念、演化历程（包括主要的贡献者），并阐明C和C++的关系

设计理念：效率、实用性优于艺术性严谨性、相信程序员

允许一个有用的特征比防止各种错误使用更重要

（一）演化历程：

Father of Simula67 :Kristen Nygaard

Father of OO programming:Ole-Johan Dahl

* Design PASCAL、Modula2、Oberon

*Niklaus Wirth

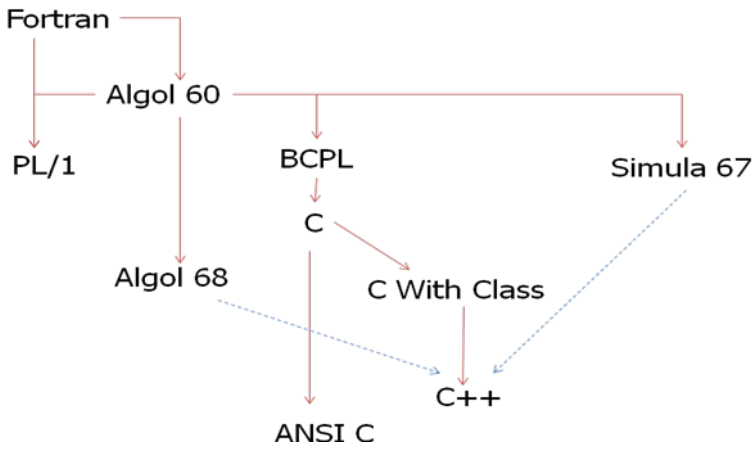
*Structural Programming: E.W.Dijkstr

C语言之父：Dennis Ritchie 、Ken Thompson

1980形成C with class:Bjarne Stroustrup

1983正式命名C++：Rick Mascitti

1994制定ANSI C++标准草案



（二）C与C++的关系：

（1）C++完全包含C 语言成分，支持 C 所支持的全部编程技巧（C的超集），C是建立C++的基础，同时C++还添加了OOP的完全支持；

（2）任何 C 程序都能被 C++ 用基本相同的方法编写，并具备相同的运行效率和空间；

（3）C++还引入了重载、内联函数、异常处理等功能，对 C 中过程化控制及其功能进行了扩充;

（4） 此外，C++还添加了 OOP 的完全支持。

（三） C 和 C++混合编程应该注意的问题。

（1）名变换:若要调用C语言库中的函数，要附加关键字“extern "C" ”；按照 C 语言方式编译和连接，限制C++编译器做 name mangling(名变换)，确保 C++和 C 编译器产生兼容的 obj 文件;

（2）静态初始化:C++静态的类对象和定义在全局的、命名空间中的或文件体中 的类对象的构造函数通常在 main 被执行前就被调用，只要可能，用 C++写 main()，即使要用 C 写 Main 也用 C++写;

（3）内存动态分配:new/delete 调用 C++的函数库，malloc/free 调用 C 的函数 库，二者要匹配，防止内存泄露;

（4） 数据结构兼容:将在两种语言间传递的东西限制在用 C 编译的数据结构的范 围内;这些结构的 C++版本可以包含非虚成员函数，不能有虚函数。

（5）因为C++是C的超集，且C是结构化编程语言，而C++支持面向对象编程语言，所以在混合编程时，不应当出现class等面向对象的关键字;

（6）C语言不支持函数重载。在C++中f (int, int) 与 f(int, double) 是不同的函数，都重载了函数f()；但是在C语言中却被认为是相同的函数。因为在编译时，C语言给这几个函数的命名为f_；而C++命名分别为f_int_int, f_int_double, f_，以表示区别；所以混合编程时应注意重载函数的问题；
（7）在c++中也允许在结构和联合中定义函数，他们也具有类的基本功能，与CLASS所不同的是：结构和联合的成员的默认访问控制为PUBLIC.

2. 影响表达式值的因素有哪些？请说明之（本题7分）
操作数、操作符和标点符号组成的序列，表示一个计算过程
优先级　　a+b*c　解决相邻两个运算符的运算顺序
结合性　　a+b-c
求值次序　　(a+b)*(a-b)　　解决不相邻的两个运算符的运算顺序 与编译系统有关
类型转换　　int x=10;　float y=2.0;　x*y
在求值次序上：操作符的副作用：改变操作数的值　（X+1）*(++X)

3. C++编译系统赋予一个空类，如：class Empty{ }，哪些成员函数？（本题8分）
Empty();
Empty(const Empty&);
~Empty();
Empty& operator=(const Empty&);
Empty *operator &();
const Empty* operator &() const;

二、请指出以下程序存在的问题（本题满分10分, 共2题）
1、
```
#include <IOSTREAM>
#include <stdlib.h>
#include <time.h>
using namespace std;

class poker{
public:
        unsigned int id;
        poker(){ id = rand() % 13 + 1; } // 产生1-13的随机数
};
void main() {
        srand(unsigned(time(NULL))); // 根据系统时间，设置随机种子值
        poker* p = new poker[5];
        int sum = 0;
        for(int i=0; i<5; i++, p++)
                sum += p->id;
        cout << "总点数为: " << sum << endl;
        delete p;
}
```
答案：由于在for循环中，移动了p指针的位置，导致原先p所申请的内存空间的首地址信息丢失，无法归还系统，因此delete p时程序会出现异常中止

2、
```
int* get_inputs() {
        int numbers[10];
        int i;
        for(i = 0; i < 10; i ++) {
                cin >> numbers[i];
        }
        return numbers;
}
```

```
int find_max(int* inputs, int size) {
        int i;
        int max = -1;
        for (i = 0; i < size; i ++ ) {
                if (max < inputs[i]) {
                        max = inputs[i];
                }
        }
        return max;
}

void main() {
        cout << "inputs" << endl;
        int* inputs = get_inputs();
        cout << "find the max" << endl;
        int max = find_max(inputs, 10);
        cout << "max: " << max << endl;
}
```

答案：get_inputs函数中，返回局部变量。在函数外部使用过程中会造成错误。

三、请给出以下程序运行结果（本题满分20分，共2题）

1、

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

class Vehicle{
public:
        virtual void run(int number = 10){
cout << "we do not know how to run\n";
        }

        virtual void stop(){
cout << “we do not know how to stop\n”;
}
        void announce(){
                cout << "this is a vehicle\n";
        }

void main(){
        Vehicle v1,*v2;
        Car c1;
        v1.run();
        c1.announce();

v2 = &c1;
        v2->run();
        v2->stop();
        v2->announce();
}
```

```
};
class Car:public Vehicle
{
public:
        void run(int number = 60){//子类改了也没用
cout << "driving at " << number << " km/h\n";
        }
        void stop(){
                cout <<“brake to stop \n”;
}
        void announce(){
                cout << "this is a car\n";
        }
};
```

答案：

```
we do not know how to run
this is a car
driving at 10km/h
brake to stop
this is a vehicle
```

2

```
class Error {
public:
        virtual void show(){ cout << "something is error"<<endl;}
};
```

```
class nameError:public Error {
public:
	void show() {
cout<<"name is error"<<endl;
}
};

class ageError:public Error {
public:
	void show() {
cout<<"age is error"<<endl;
}
};

class Person {
private:
	int age;
	char* name;
public:
	void setAge(int a) {
		ageError ag;
		if(a<0||a>100)
			throw ag;
		this->age=a;
}
	void setName(char* str){
nameError ne;
		if(str=="exit")
			throw ne;
		this->name=str;
}
};

void catcher(int command, Person p){
	try {
		switch(command){
		case 1:
			p.setAge(101);
			break;
		case 2:
			p.setName("exit");
			break;
		}
	}
	catch(nameError ner){
		ner.show();
	}
	catch(Error er) {
		er.show();
	}
	catch(ageError aer){
		aer.show();
	}
}

int main(void) {
	Person p;
	catcher(1, p);
	catcher(2, p);
	cout<<"program end"<<endl;
	return 0;
}
```

答案：

```
something is error
name is error
program end
```

四、程序填充（本题满分10分）

下列程序输出如下星状图形，请将程序空白部分补充完整：

```
*
* *
* * *
* * * *
```

```
int printEachLine(int j){
if (_____)
	return 1;
else{
```

```
_____
_____
return1;
}
}

int print(int i){
if (_____)
return 1;
else{
```

```
_____
_____
cout<<endl;
return1;
}
}
void main(){
print(4);
}
```

答案：

```
if (j==0)
printEachLine(j-1);
cout<<"* ";
if (i==0)
print(i-1);
printEachLine(i);
```

五、（编程题。本题满分30分。）

观察者模式：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

举个博客订阅的例子，当博主发表新内容的时候，即博客内容发生了改变，那些订阅的读者就会收到通知。博主与读者之间存在种一对多的依赖关系，**博客中可能有多个观察者**（即订阅者），当博客的内容发生变化时，通过notify成员函数通知所有的观察者，告诉他们博客的内容更新了。而观察者通过update成员函数获取博客的内容信息

请根据以下类图和部分类描述，用C++给出相关类的实现（Blog, BlogCSDN，Observer，ConsoleObserver）。

类名Blog：

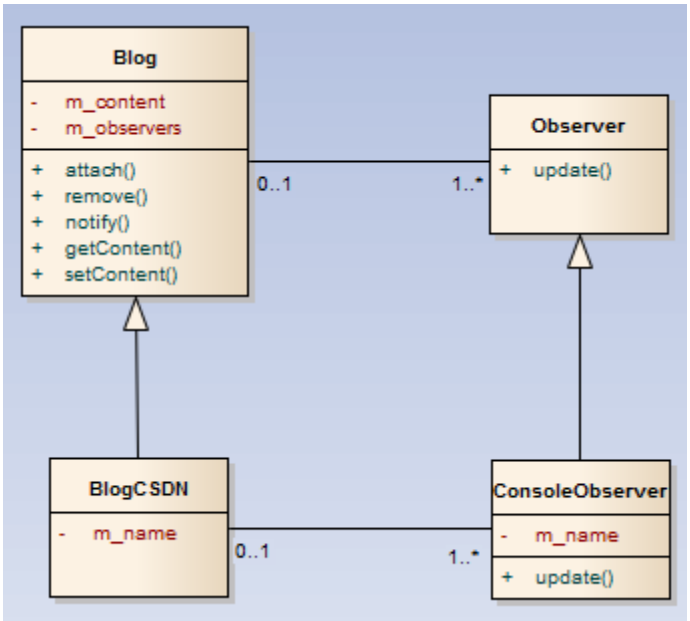
属性	描述
m_content	博客内容（长度< 1024个字符）
m_observers	多个观察者
方法	描述
attach	添加博客听众
remove	删除博客听众
notify	通知所有听众，获取当前博客内容
setContent	设置博客内容
getContent	获取博客内容

类名BlogCSDN:

属性	描述
m_name	博主名称（长度<16个字符）

类名ConsoleObserver：

属性	描述
m_name	观察者名称（长度< 128个字符）
方法	描述
update	获得观察的博客的更新内容，并在控制台显示



答案:

4个类写出来，并写对继承关系，以及成员变量初始化 **5分**

写出了1对多，即多个观察者 **7分**

attach、remove、notify、update **每个方法各4分**

getContent、setContent **每个方法各1分**

```
#include <iostream>
#include <list>

using namespace std;

//观察者
class Observer
{
public:
	Observer() {}
	virtual ~Observer() {}
	virtual void update() {}
};

//博客
class Blog
{
public:
	Blog() {}
	virtual ~Blog() {}
	void attach(Observer *observer) { m_observers.push_back(observer); }    //添加观察者
	void remove(Observer *observer) { m_observers.remove(observer); }      //移除观察者
```

```
        void notify() //通知观察者
        {
                list<Observer*>::iterator iter = m_observers.begin();
                for(; iter != m_observers.end(); iter++)
                        (*iter)->update();
        }
        void setContent(char* c) { strcpy(m_content, c); } //设置博客内容
        char* getContent() { return m_content; }    //获得博客内容
private:
        list<Observer* > m_observers; //观察者链表
protected:
        char m_content[1024]; //博客内容
};

//具体观察者
class ConsoleObserver : public Observer
{
private:
        char  m_name[128];
        Blog *m_blog;
public:
        ConsoleObserver(char* name,Blog *blog){
                strcpy(m_name, name);
                m_blog = blog;
        }
        ~ConsoleObserver() {}
        void update()  //获得更新内容并输出
        {
                string content = m_blog->getContent();
                cout<<"Blog content update : " << content << ". My name is : " << m_name << endl;
        }
};

//具体博客类
class BlogCSDN : public Blog
{
private:
        char m_name[16]; //博主名称
public:
        BlogCSDN(char* name){strcpy(m_name, name);}
        ~BlogCSDN() {}

};

//测试用例
int main()
{
   Blog *blog = new BlogCSDN("ZT");

        Observer *observer1 = new ConsoleObserver("GSX", blog);
        Observer *observer2 = new ConsoleObserver("HT", blog);
        Observer *observer3 = new ConsoleObserver("QYG", blog);
        Observer *observer4 = new ConsoleObserver("WLX", blog);

   blog->attach(observer1);
        blog->attach(observer2);
        blog->attach(observer3);
```

```
        blog->attach(observer4);

    blog->setContent("C++期末试卷");
    blog->notify();
        blog->remove(observer1);

    delete blog;
        delete observer1;
        delete observer2;
        delete observer3;
        delete observer4;
    return 0;
}
```

2003年考试题

一、 简述题（30分）

1、 面向对象程序设计的主要特点?

答：封装，继承，多态。通过消息传递来实现程序的运转。

2、 什么是静态绑定和动态绑定?

在C++中通过继承实现的多态是动态绑定，通过模板实现的多态是静态绑定。静态绑定发生在编译时刻，依据对象的静态类型进行，效率高，但灵活性差。 动态绑定发生在运行时刻，依据对象的实际类型动态进行，灵活性高，但效率低。 C++中默认为前期绑定，后期绑定需显式地指出。

3、 在C++中，重用一段代码，有哪两种方法？分别有什么特点?

答：内联函数；继承。

4、 结构化程序设计的基本控制结构有哪几种？分别流程图的形式表示。

答：顺序，循环，条件

5、 什么是抽象数据类型（ADT）？试以时间类型为例，简单描述之。

答：抽象数据类型是指一个数学模型以及定义在此数学模型上的一组操作，需要通过固有数据类型(高级编程语言中已实现的数据类型)来实现。抽象数据类型(ADT):用于指定逻辑特性而不指定实现细节的数据结构

二、 程序理解题（20分）

1．设有一个矩阵：

```
0    2    1
1    0    2
1    2    0
```

现把它放在一个二维数组a中，写出执行下面语句后a的值

```
for  (int  i=0;  i <=2;  i++)
   for  (int  j=0;  j <=2;  j++)
      a[i][j]  =  a[a[i][j]] [a[j][i]];
```

答：010 200 010

```
2     #include  <iostream  >
      using  namespace  std;
void  f(int  &x,int  y)
{   y  =  x  +  y;
    x  =  y  %  4;
    cout  <<  x  <<  y  <<  endl;
}
void  main()
{  int  x  =  4,  y  =  5;
```

```
   f(y,x);
   cout << x << y << endl;
   f(x,x);
   cout << x << y << endl;
}
```

答：09 05 08 04

```
3. #include <iostream.h>
class A{
public:
   A() { cout << "in A's constructor\n "; };
   ~A() { cout << "in A's destructor\n "; };
};
class B{
public:
   B() { cout << "in B's constructor\n "; };
   ~B() { cout << "in B's destructor\n "; };
};
class C: public B{
private:
   A a;
public:
  C(){ cout << "in C's constructor\n "; };
  ~C(){ cout << "in C's destructor\n "; };
};
void main()
{   A a;
    B *p=new C;
        delete p;
}
```

答：in A's constructor， in B's constructor ， in A's constructor

in C's constructor ， in B's destructor ， in A's destructor

三、 指出下面程序片段中错误（10分）

```
1.  const int x=0;
    int y = 2;
    const int *p1 = &x;
    int *const p2;
    int *p3;
    p1 = &y;
    *p1 = 10;
    p2 = &y;
    p3 = &x;
```

答：“int *const p2;”指针常量声明时就应该被初始化；所以，“p2 = &y;”也错

“*p1 = 10;”错误，常量指针里的内容不可修改；

“p3 = &x;”错误，不能将普通的int型指针转化为const int型指针；

```
2.  class A {
       int m1;
          static char m2;
     public:
          void f1() { m1 = 0; m2 = 1; };
          static void f2() { m1 = 2; m2 = 3; };
          void f3() const { m1 = 4; m2 = 5; };
    };
```

答：“static void f2() { m1 = 2; m2 = 3; };” 静态成员函数使用了非静态成员变量；

“void f3() const { m1 = 4; m2 = 5;};” 在const类型的函数中改变了类的非静态数据成员

四、下面的C++程序能正常结束吗？如果不能，请指出原因。（5分）

```
class A
{   int   i, j;
  public:
   A()  {  i=j=0;  }
};
class B
{   A  *p;
  public:
    B()   {   p  =  new  A;   }
   ~B()  {   delete  p;   }
};
void  f(B  x){  ......}
void  main()
{ B  b;
  f(b);
}
```

答：不能。在b调用函数f()时，直接传递了B b对象，在函数结束时，会自动清除函数中的局部变量和参数所占用的内存，所以对象B b被清除掉了。等到main函数结束后，系统又会自动结束所有变量的声明周期，去调用class B中的析构函数，在delete p时，由于p所指内容已经被清除过了，就会产生错误清除系统的内存，从而产生错误，是程序不能正常结束。将函数f中的参数改为（B &b）即可。

五、编程题（35分）

1.编写程序，使其能读入最多10个正整数，其中输入过程中一旦有零或付整数输入，则停止读取，然后反向输出已读入的正整数。其中，输入的部分要求以函数GetNums实现，反向输出的部分以ReverseWrite实现，并且要求使用递归。

2．参照以下表格，定义两个类CStudent（学生） 和CUnderGraduated（大学生），要求使用继承，即定义CUnderGraduated 为CStudent 的子类，同时使其能完成如下功能：

[1]可以这样定义数组：

```
CStudent  students[20];  CUnderGraduated   A[20];
```

[2]

```
CStudent   S(“Smith”,  0,  1002);
```

0代表女生

```
CUnderGraduated   S(“John”,   3201,   1,   50,  “Title:  Programming  Language  C++”);
```

1 代表男生

```
CUnderGraduated   S1  =  S;
```

[3] 在CUnderGraduated 中提供函数AddContent 用以增加该学生的论文内容

[4] 对于CUnderGraduated的对象S，S+=X (X为int型) 可以使S学生的学分数增加X，例如 S + =20，可以使S学生的学分数增加20

[5] 对于CUnderGraduated的对象S1和S2，可以用 S1 <S2 比较两个大学生学号的先后关系

[6] 提供全局函数，统一输出学生的信息（输出格式没有要求），要求使用多态性

学生 Name 最长16个字符 No 正整数 Sex Male 或Female Name 表示学生姓名 Sex 表示性别 No表示学号

大学生 Name 最长16个字符 No 正整数 Sex Male 或Female Credits 正整数 Thesis 不定长的一组字符 Name 表示学生姓名 Sex 表示性别 No表示学号 Credits 表示学生的学分数 Thesis 表示毕业论文

注意：题目要求不得使用系统类string

2008年考试题

1写出C和C++的关系，并说明应注意的问题。10%

2给给出影响表达式求值的因素。10%

3请举例说明C++多态的几种表现方式。10%

（一）多态的含义：

1) 某一论域中的一个元素可以有多种解释;

2) 相同的语言结构可以代表不同类型的实体;

3) 相同的语言结构可以对不同类型的实体进行操作;

4) 一个公共的消息集可以发送到不同种类的对象，从而得到不同的处理。

（二）C++多态的几种表现形式：

虚函数、函数重载、操作符重载、模板、继承、函数指针。

可分为四类:重载多态、强制多态、包含多态和参数多态。

4何为引用？其主要作用是什么？何时需要将一个函数的返回值类型定义为引用类型？如果随意将函数的返回值类型定义为引用类型会造成什么危害？10%

含义:为一块已有的内存空间取一个别名。

作用:避免耗费资源，提高访问效率。主要用于函数参数传递和动态变量命名。

用途:返回对象本身。

危害:如果函数返回值的类型是引用或指针类型，不应该把局部量或局部量的地址作为返回值。

5如何利用析构函数防止内存漏洞？举例说明。10%

对象消亡时，在系统收回他所占的存储空间之前，系统将自动调用析构函数。一 般情况下不需要定义析构函数，但是如果对象在创建后申请了一些资源并且没有 归还这些资源，则应定义析构函数来在对象消亡时归还对象申请的资源在对象创 建时，系统会为对象分配一块存储空间来存储对象的数据成员，但对于指针类型 的数据成员来说，系统只分配了存储该指针所需要的空间，而没有分配指针所指 向的空间，对象自己需要申请(作为资源)。同样，在对象消亡时，系统收回的 只是指针成员本身的存储空间，而指针所指向的空间需要对象自己归还(作为资源)。

举例：class String

```
{     char *str;
  public:
      String()    { str = NULL; }
      String(char *p)
    { str = new char[strlen(p)+1]; strcpy(str,p); }
      ~String()
      { delete []str; }
}
```

当String对象创建初始化后，会有char* str对象在堆中。如果没有析构函数，但对象会撤销，char* str仍在堆中，造成内存泄露。而用析构函数，可以在对象撤销时删除str，防止内存泄露。

6请说明C++设计类依附于什么原则将你所定义的成员函数定义为纯虚函数、虚函数或非虚函数？5%

纯虚函数：只给出函数声明而没有给出函数实现的虚成员函数。

只有函数接口会被继承

子类必须继承函数接口

（必须）提供实现代码

一般虚函数

函数的接口及缺省实现代码都会被继承

子类必须继承函数接口
可以继承缺省实现代码

非虚函数
函数的接口和其实现代码都会被继承
必须同时继承接口和实现代码

7请给出你认为最有价值的C++程序设计应该遵守的5条原则，并简明分析其意义所在。10%
Use const whenever possible:
Guard against potential ambiguity
Never treat arrays polymorphically
Make non-leaf classes abstract
Strive for exception-safe code
Use destructor to prevent resource leaks

8编写函数 int count_word(const char *text,const char*word)来统计一个英语文本（由参数text指向）中的某个单词（由word指向）出现的次数。例如：函数调用count——word （“the theater is showing the film Gone With The Wind ”,"the"）返回值为3．10%

```
// Author      :     yankai
bool beginWith(const char *text, const char* word) {
    while (*word != '\0') {
        if (*word != *text) {
            return false;
        }
        ++text;
        ++word;
    }
    return true;
}

int count_word(const char *text, const char* word) {
    int count = 0;
    while (*text != '\0') {
        if (beginWith(text, word))
            ++count;
        ++text;
    }
    return count;
}
```

9 定义一个时间类CTime，它表示时分秒，并能实现以下程序段所需的功能。25%

```
CTime t1;//t1表示的时间为0时0分0秒
CTime t2;//t2表示的时间为18时10分40秒
int s;cin??s;
t1=t2+s//把t1的时间设为t2表示的时间加S秒
cout??t1??endl;//输出时间格式时分秒
cout??t1-t2??endl;//输出时间差
```

```
//CTime.h
#ifndef CTIME_H_
#define CTIME_H_

#include ?iostream?
```

```
 using namespace std;

class CTime {
 private:
    long time;

public:
    CTime();
   CTime(long);
   CTime(long,long,long);
   CTime& operator+=(const CTime&);
   CTime& operator-=(const CTime&);
    friend CTime operator+(const CTime& cTime, const CTime& buffer);
    friend CTime operator-(const CTime& cTime, const CTime& buffer);
    friend std::ostream& operator??(std::ostream&, const CTime&);
};

#endif /* CTIME_H_ */


//CTime.cpp
 #include "CTime.h"
 #include ?iostream?

CTime::CTime() :
    time(0) {
 }

CTime::CTime(long l) :
    time(l) {
 }

CTime::CTime(long h, long m, long s) {
    m += 60 * h;
    s += 60 * m;
    time = s;
 }

CTime& CTime::operator+=(const CTime& buffer) {
    time += buffer.time;
    return *this;
 }

CTime& CTime::operator-=(const CTime& buffer) {
    time -= buffer.time;
    return *this;
 }

CTime operator+(const CTime& cTime, const CTime& buffer) {
    CTime newOne(cTime);
    newOne += buffer;
    return newOne;
 }

CTime operator-(const CTime& cTime, const CTime& buffer) {
    CTime newOne(cTime);
    newOne -= buffer;
    return newOne;
 }

std::ostream& operator??(std::ostream& out, const CTime& cTime) {
```

```
    long m = cTime.time / 60;
    long h = m / 60;
    m %= 60;
    long s = cTime.time % 60;
    out ?? h ?? ":" ?? m ?? ":" ?? s;
    return out;
}

int main(int argc, char **argv) {
    CTime t1;//t1表示的时间为0时0分0秒
    CTime t2(18, 10, 40);//t2表示的时间为18时10分40秒
    cout ?? t1 ?? endl;
    cout ?? t2 ?? endl;
    int s;
    cin ?? s;
    t1 = t2 + s;//把t1的时间设为t2表示的时间加S秒
    cout ?? t1 ?? endl;//输出时间格式时分秒
    cout ?? t1 - t2 ?? endl;//输出时间差
}
```

10.引用类型与指针类型相比，其优势在哪里?

（1）引用是采用直接访问形式，指针则采用间接访问形式——效率高;

（2）引用与被引用变量共享内存，而指针有自己的内存空间——内存占用少;

（3）在作为函数参数类型时，引用类型参数的实参是一个变量，而指针类型参数的实参是一个变量的地址——代码可读性好;

（4）引用类型一旦定义后不能改变，而指针变量定义后可以指向其他同类型的变 量——安全。

（5）大多数编译程序往往把引用类型作为指针类型来实现，它对使用者而言是透明的。

2010年考试题

1、请简述C++程序设计语言的设计理念、演化历程（包括主要的贡献者），并阐明C和C++的关系。

2、请简述C与C ++混合编程时要注意的问题。

3、（本题满分 10 分）inline函数的作用？随意使用所可能导致的问题？并请阐述合理使用的建议。

作用：（1）提高程序的可读性;

（2）提高程序的运行效率;

（3）可以弥补宏定义不能进行类型检查的缺陷。

导致的问题：（1）增大目标代码，因为在调用时，必须在调用该函数的每个文本文件中定义;

（2）病态换页;

（3）降低指令快取装置的命中率;

建议：（1）仅对使用频率较高的小段代码使用内联;

（2）将内联函数的定义放到头文件中;

（3）在内联函数中不能含有负复杂的结构控制语句，如switch、while等;

（4）递归函数不能用来做内联函数;

限制： （1）非递归;（2）由编译系统控制。

4、（本题满分 10 分）请给出C++语言中关键字const的几种使用方法，并给出示例?

const可以用来修饰数据类型，指针，函数，对象等;

（1）修饰数据类型时，表明该数据类型在程序运行的过程中，值不应当被改变；如const int a = 3，表示int型的数据 a 在程序运行开始到结束，其值一直是3，不能被改变。并且，在使用const修饰数据类型是，声明的时候就必须进行初始化；

（2）修饰指针时，有3种不同的用法：“指针常量，常量指针，常量指针常量”

1) 指针常量：const Type *p = (Type&)a;

Ø 指针可修改；

Ø 所指内容不可修改，因为所指的内容是常量；

2) 常量指针：Type const *p = (Type&)a;

Ø 指针不可修改，因为该指针是常量；

Ø 所指内容可以修改；

3) 常量指针常量：const Type const *p = (Type&)a;

Ø 指针和所指内容均不可修改，因为二者都是常量，并且在声明时就必须初始化；

（3）修饰函数时，有2种不同的用法：“Type f() const{}，Type f(const T & t){}”；

1) 常成员函数Type f() const{}：表明程序员告诉编译器不会修改常量值；

2) 拷贝构造函数中的常量参数Type f(const T & t){}：该用法长出现在拷贝构造函数中，其中const可有可无，主要目的是希望不要改变对象T的实例t中的任何内容；

（4）修饰对象时，主要是修饰对象中成员变量：如：

Class T{ ... ... const int a; ... ...};

因为在类的成员变量声明时，不应该初始化变量的值，而常量在运行时刻就已经拥有数值且不能被改变。所以，应当在构造函数中使用成员初始化表来对常量成员变量进行初始化。

5、（本题满分 15 分）什么是纯虚函数、虚函数和非虚函数？合理定义三种成员函数所应遵循的基本原则？请给出你认为合理定义的一个实例，并说明。

答：三者的定义：

非虚函数：即一般的成员函数；

虚函数：即在成员函数前加关键字virtual来标志；

纯虚函数：是指被表明为不具体实现的虚成员函数，形式为在虚函数后加“ = 0”。、

基本原则：

使用虚函数的限制

类的成员函数才可以是虚函数

静态成员函数不能是虚函数

内联成员函数不能是虚函数

构造函数不能是虚函数

析构函数可以（往往）是虚函数

注意事项：

不要定义与继承而来的非虚成员函数同名的成员函数

绝对不要重新定义继承而来的缺省参数值

三者的关系为：

纯虚函数

Ø 只有函数接口会被继承

Ø 子类必须继承函数接口

Ø （必须）提供实现代码

一般虚函数

Ø 函数的接口及缺省实现代码都会被继承

Ø 子类必须继承函数接口

Ø 可以继承缺省实现代码

非虚函数

Ø 函数的接口和其实现代码都会被继承

Ø 必须同时继承接口和实现代码

```
class Shape{
public:
Shape(){}
virtual ~Shape() = 0;                    //虚函数
virtual double area(){  return 0.0;  }          //纯虚函数
double getWeight(){  return  weight;  }      //非虚函数
private:
double area;
double weight;
};
```

6、（本题满分　20　分）

在C++程序中，可以利用析构函数防止资源泄露，请给出模板auto_ptr的基本定义、实现，及应用实例。

7、（本题满分　20　分）

请阐述在面向对象程序设计语言中引入构造函数机制的原因，并请给出控制一个类创建实例个数的手段（举例说明）。

答：原因：成员初始化的需要。

（1）类的封装性：由于访问权限控制，一部分数据是不能让外界访问的，通常不能直接赋值，所以初始化类的成员变量应该有类的成员函数来完成;

（2） 若使用普通成员函数，一方面需要显式地调用，使用不便，另一方面可能导致未调用初始化函数就使用对象，不安全。

（3）在构造对象实例时，希望由系统自动调用，而不是人为的去调用，同时根据类的唯一性和对象的多样性，则使用类名作为构造函数的名字;

示例：利用静态成员可以被所有该类的对象所共享的性质来控制该类创建实例的个数。

```
class singleton{
protected:
singleton(){ }
   singleton(const singleton &);
public:
static singleton * instance(){
    return m_num == 0? m_instance = new singleton: m_instance;
}
   static void destroy() { delete m_instance; m_instance = NULL; }
private:
static singleton * m_instance;
Static int m_num;
};
singleton * singleton ::m_instance= NULL;
```

补充读程序题

1．

```
class A {
  public:
    A(){
       a1=a2=0;
       cout<<"Default constructor called.\n";
    }
A(int i,int j);
~A()
{  cout<<"Destructor called.\n";  }
 void Print()
 {  cout<<"a1="<<a1<<','<<"a2="<<a2<<endl;  }
```

```
    private:
      int a1,a2;
  };
  A::A(int i,int j) {
     a1=i;
     a2=j;
     cout<<"Constructor called.\n";
  }
  void main()   {
     A a,b(5,8);
     a.Print();
     b.Print();
  }
```

答： Default constructor called.
Constructor called.
a1=0,a2=0
a1=5,a2=8
Destructor called.
Destructor called.

```
2.class B   {
    public:
      B()
      {  cout<<++b<<endl;  }
      ~B()
      {  cout<<b--<<endl;  }
       static int Getb()
       {  return b;  }
    private:
      static int b;
  };
  int B::b=10;
  void main()   {
     B b1,b2,b3;
     cout<<B::Getb()<<endl;
  }
```

答： 11
12
13
13
13
12
11

```
3. class Date   {
    public:
      Date(int y,int m,int d)
      {
         year=y;
         month=m;
         day=d;
      }
      friend void Print(Date &);
    private:
      int year,month,day;
  };
  void Print(Date &d)   {
     cout<<d.year<<'/'<<d.month<<'/'<<d.day<<endl;
  }
  void main()   {
      Date d1(2005,10,1),d2(2005,12,9);
      Print(d1);
      Print(d2);
  }
```

答： 2005/10/1
2005/12/9

```
4.    class C   {
    public:
      C(int i,int j)
      {  c1=i;c2=j;  }
      void Sum(C a,C b)       {
         c1=a.c1+b.c1;
         c2=a.c2+b.c2;
      }
      void Print()  {  cout<<"c1="<<c1<<','<<"c2="<<
c2<<endl;  }
    private:
      int c1,c2;
  };
  void main()   {
     C a(6,9);
     C b(a);
     C c(b);
     c.Sum(a,b);
     c.Print();
  }
```

答： c1=12,c2=18

5.返回数组和指针的区别（结构化部分）

```
char* GetString1(){
char p[] = "Hello World";
 return p;
}
char* GetString2(){
 char *p = "Hello World";
 return p;
}

void main(){
 cout<<"GetString1 returns:"<< GetString1()<<endl;
 cout<<"GetString2 returns:"<< GetString2()<<endl;
}
```

答案：输出两行，第一行GetString1 returns: 后面跟的是一串随机的内容，而第二行GetString2 returns: Hello World.

两个函数的区别在于GetString1中是一个数组，而GetString2中是一个指针。

当运行到GetString1时，p是一个数组，会开辟一块内存，并拷贝"Hello World"
初始化该数组。接着返回数组的首地址并退出该函数。由于p是GetString1内的一个局部变量，当运行到这个函数外面的时候，这个数组的内存会被释放掉。因此在_tmain函数里再去访问这个数组的内容时，结果是随机的。

当运行到GetString2时，p是一个指针，它指向的是字符串常量区的一个常量字
符串。该常量字符串是一个全局的，并不会因为退出函数GetString2而被释放掉。因此在_tmain中仍然根据GetString2返回的地址得到字符串"Hello World"。

6.缺省参数（面向对象部分）

```
class A{
public:
virtual void Fun(int number = 10)
  {
     cout << "A::Fun with number " << number;
  }
};

class B: public A{
public:
virtual void Fun(int number = 20)
  {
    cout << "B::Fun with number " << number;
  }
};

void main(){
B b;
  A &a = b;
  a.Fun();
}
```

答案：输出B::Fun with number 10

（1） 由于a是一个指向B实例的引用，因此在运行的时候会调用B::Fun。但缺省参数是在编译期决定的。在编译的时候，编译器只知道a是一个类型a的引用，具体指向什么类型在编译期是不能确定的，因此会按照A::Fun的声明把缺省参数number设为10。

（2） 这一题的关键在于理解确定缺省参数的值是在编译的时候，但确定引用、指针的虚函数调用哪个类型的函数是在运行的时候。

7.二维数组

```
void main() {
int m[3][3]={ {1}, {2}, {3} }, n[3][3]={ 1, 2, 3 };

cout << m[1][0]+n[0][0] << endl;
cout << m[0][1]+n[1][0] << endl;
 }
```

答案：输出两行，第一行：3；第二行：0

8.虚函数

```
class A{
public:
      virtual void fun1()
            {
                  cout<<"调用A::fun1()\n";
            }
      void fun2()
            {
                  fun1();
            }
};

class B:public A{
public:
      void fun1()
            {
                  cout<<"调用B::fun1()\n";
            }
};

void print(A& a)
{
            a.fun1();
}

void main()
{
            A a,*pa;
      B b;

      b.fun1();

      pa=&a;
            pa->fun1();

      pa=&b;
            pa->fun1();

      print(a);
      print(b);
}
```

答案：输出5行

调用B::fun1()

调用A::fun1()

调用B::fun1()

调用A::fun1()

调用B::fun1()

9.数组，字符串的比较

```
void main(){
char str1[] = "abc";
char str2[] = "abc";

const char str3[] = "abc";
const char str4[] = "abc";

const char *str5 = "abc";
const char *str6 = "abc";

char *str7 = "abc";
```

```
char *str8 = "abc";

cout << ( str1 == str2 ) << endl;
cout << ( str3 == str4 ) << endl;
cout << ( str5 == str6 ) << endl;
cout << ( str7 == str8 ) << endl;
}
```

答案：0 0 1 1

str1,str2,str3,str4是数组变量，它们有各自的内存空间；而str5,str6,str7,str8是指针，它们指向相同的常量区域。const的目的是混淆视听。

10.数组，指针

```
void main(){
int a[5]={1,2,3,4,5};
int *ptr=(int *)(&a+1);
cout << *(a+1) << ", " << *(ptr-1) << endl;
}
```

答案：2,5

(a+1)就是a[1]，(ptr-1)就是a[4], 执行结果是2, 5。&a+1不是首地址+1，系统会认为加一个a数组的偏移，是偏移了一个数组的大小(本例是5个int)。

11.成员变量初始化

```
class base{
private:
int m_i;
  int m_j;
public:
  base( int i ) : m_j(i),m_i(m_j) {}
  base() : m_j(0),m_i(m_j){}
  int get_i() {return m_i;}
  int get_j() {return m_j;}
};

void main (){
  base obj(98);
  cout << obj.get_i() <<endl<< obj.get_j() <<endl;
}
```

答案：输出结果第一个为随机数，第二个是98。

解析：本题想得到的结果是“98，98”。但是成员变量的声明是先 m_i ，然后是 m_j；初始化列表的初始化变量顺序是根据成员变量的声明顺序来执行的，因此，先初始化 m_i，但此时 m_j 还未初始化，m_i 会被赋予一个随机值。改变一下成员变量的声明顺序可以得到预想的结果。

12.变量值

```
void main(){
int x = 9999;
int countx = 0;
          while(x){
x = x & (x-1);
countx++;
}
cout << countx << endl;
}
```

答案：8

解析：countx是把x转化为二进制之后含有1的个数.

13.

```
class S    {
    public:
      S()        {  PC=0;  }
      S(S &s)        {
         PC=s.PC;
         for(int i=0;i<PC;i++)
            elems[i]=s.elems[i];
      }
      void Empty()        {  PC=0;  }
      int IsEmpty()        {  return PC==0;  }
      int IsMemberOf(int n);
      int Add(int n);
      void Print();
    private:
      int elems[100],PC;
  };
  int S::IsMemberOf(int n)    {
     for(int i=0;i<PC;i++)
        if(elems[i]==n)
           return 1;
     return 0;
  }
  int S::Add(int n)    {
     if(IsMemberOf(n))
        return 1;
     else if(PC==100)
        return 0;
     else        {
        elems[PC++]=n;
        return 1;
     }
  }
  void S::Print()    {
     cout<<'{';
     for(int i=0;i<PC-1;i++)
        cout<<elems[i]<<',';
     if(PC>0)
        cout<<elems[PC-1];
     cout<<'}'<<endl;
  }

  void main()    {
     S a;
     cout<<a.IsEmpty()<<endl;
     a.Print();
     S b;
     for(int i=1;i<=5;i++)
        b.Add(i);
     b.Print();
     cout<<b.IsMemberOf(3)<<endl;
     cout<<b.IsEmpty()<<endl;
     for(i=6;i<=10;i++)
        b.Add(i);
     S c(b);
```

```
    c.Print();
  }
```
答： 1
{}
{1,2,3,4,5}
1
0
{1,2,3,4,5,6,7,8,9,10}

14．
```
class A    {
  public:
    A(int i)
    { a=i; }
    A()       {
      a=0;
      cout<<"Default constructor called."<<a<<endl;

    }
    ~A()      { cout<<"Destructor called."<<a<<en
dl; }
    void Print()      { cout<<a<<endl; }
  private:
    int a;
  };
  void main()    {
    A a[4],*p;
    int n=1;
    p=a;
    for(int i=0;i<4;i++)
      a[i]=A(++n);
    for(i=0;i<4;i++)
      (p+i)->Print();
  }
```
答： Default constructor called. 0
Default constructor called. 0
Default constructor called. 0
Default constructor called. 0
Destructor called. 2
Destructor called. 3
Destructor called. 4
Destructor called. 5
2
3
4
5
Destructor called. 5
Destructor called. 4
Destructor called. 3
Destructor called. 2

15．
```
class B    {
  public:
    B(int i)      { b=i; }
    B()       {
      b=0;
      cout<<"Default constructor called."<<b<<end
l;
    }
    ~B()      { cout<<"Destructor called."<<b<<en
dl; }
    void Print()      { cout<<b<<endl; }
  private:
    int b;
  };
  void main()    {
    B *pb[4];
    int n=1;
    for(int i=0;i<4;i++)
      pb[i]=new B(n++);
    for(i=0;i<4;i++)
      pb[i]->Print();
    for(i=0;i<4;i++)
      delete *(pb+i);
  }
```
答： 1
2
3
4
Destructor called. 1
Destructor called. 2
Destructor called. 3
Destructor called. 4

16．
```
    class C    {
  public:
    C(int i)      { c=i; }
    C()       {
      c=0;
      cout<<"Default constructor called."<<c<<endl;

    }
    ~C()      { cout<<"Destructor called."<<c<<en
dl; }
    void Print()      { cout<<c<<endl; }
  private:
    int c;
  };
  void main()    {
    C *p;
    p=new C[4];
    int n=1;
    for(int i=0;i<4;i++)
      p[i]=C(n++);
    for(i=0;i<4;i++)
      p[i].Print();
    delete []p;
  }
```
答： Default constructor called. 0
Default constructor called. 0
Default constructor called. 0
Default constructor called. 0
Destructor called. 1
Destructor called. 2
Destructor called. 3
Destructor called. 4
1
2
3

```
4
Destructor called. 4
Destructor called. 3
Destructor called. 2
Destructor called. 1
```

17.
```
class D    {
  public:
    D()       {
      d1=d2=0;
      cout<<"Default constructor callrd.\n";
    }
    D(int i,int j)       {
      d1=i; d2=j;
      cout<<"Constructor called."<<"d1="<<d1<<','
<<"d2="<<d2<<endl;
    }
    ~D()       { cout<<"Destructor called."<<"d1="
<<d1<<','<<"d2="<<d2<<endl; }
    void Set(int i,int j)       { d1=i;d2=j; }
  private:
    int d1,d2;
  };
  void main()    {
    int n(10),m(20);
    D d[4]={D(5,7),D(3,6),D(7,9),D(1,4)};
    for(int i=0;i<4;i++)
      d[i].Set(n++,m++);
  }
```
答：
```
Constructor called. d1=5,d2=7
Constructor called. d1=3,d2=6
Constructor called. d1=7,d2=9
Constructor called. d1=1,d2=4
Destructor called. d1=13,d2=23
Destructor called. d1=12,d2=22
Destructor called. d1=11,d2=21
Destructor called. d1=10,d2=20
```

18.
```
class E    {
  public:
    E(int i,int j)       {
      e1=i; e2=j;
      cout<<"Constructor called."<<"e1="<<e1<<','
<<"e2="<<e2<<endl;
    }
    void FunE(E *e)       {
      e1=e->e1;
      e2=e->e2;
      cout<<" Constructor called."<<"e1="<<e1<<',
'<<"e2="<<e2<<endl;
    }
    void FunE(E &e)       {
      e1=e.e1;
      e2=e.e2;
      cout<<"In FunE(E &e)."<<"e1="<<e1<<','<<"
e2="<<e2<<endl;
    }
  private:
    int e1,e2;
  };
  void main()    {
    E a(5,6),b(3,4);
    a.FunE(&b);
    b.FunE(a);
  }
```
答：
```
Constructor called. e1=5,d2=6
Constructor called. e1=3,e2=4
In FunE(E &e). e1=3,e2=4
In FunE(E &e). e1=3,e2=4
```

19.
```
class F    {
  public:
    class G       {
    public:
      G()          {}
      G(int i)          { g=i; }
      int GetValue()          { return g; }
      void Print(F *p);
    private:
      int g;
    }myg;
    friend class G;
    F(int i,int j):myg(i)
    { f=j; }
  private:
    int f;
  };
  void F::G::Print(F *p){
    cout<<p->f<<endl;
  }
  void main()    {
    F::G g;
    F f(5,10);
    f.myg.Print(&f);
    g=f.myg;
    cout<<g.GetValue()<<endl;
  }
```
答：
```
10
5
```

20.
```
class A   {
 public:
  A(int i,int j)
  { a1=i;a2=j; }
  void Move(int x,int y)
  { a1+=x;a2+=y; }
  void Print()
  { cout<<'('<<a1<<','<<a2<<')'<<endl; }
 private:
  int a1,a2;
};
class B:private A   {
 public:
  B(int i,int j,int k,int l):A(i,j)
  { b1=k;b2=l; }
  void Print()
  { cout<<b1<<','<<b2<<endl; }
  void f()
  { A::Print(); }
  void fun()
```

```
    { Move(5,8); }
  private:
    int b1,b2;
};
void main()  {
    A a(11,12);
    a.Print();
    B b(31,32,33,34);
    b.fun();
    b.Print();
    b.f();
}
```

答： (11,12)
33.34
(36,40)

21.
```
class A  {
  public:
    void InitA(int i,int j)
    { a1=i;a2=j; }
    void Move(int x,int y)
    { a1+=x;a2+=y; }
    int Geta1()
    { return a1; }
    int Geta2()
    { return a2; }
  private:
    int a1,a2;
};
class B:public A  {
  public:
    void InitB(int i,int j,int k,int l)
    {
      InitA(i,j);
      b1=k;
      b2=l;
    }
    void Move(int x,int y)
    { b1+=x;b2+=y; }
    int Getb1()
    { return b1; }
    int Getb2()
    { return b2; }
  private:
    int b1,b2;
};
class C:public B
{
  public:
    void fun()
    { Move(10,15); }
};
void main()
{
    C c;
    c.InitB(11,12,13,14);
    c.fun();
    cout<<c.Geta1()<<','<<c.Geta2()<<','<<c.Getb1()
<<','<<c.Getb2()<<endl;
}
```

答： 11,12,23,29

22.
```
class A
{
  public:
    A(int i):a(i)
    { cout<<"A:constructor called.\n"; }
    ~A()
    { cout<<"A:Destructor called.\n"; }
    void Print()
    { cout<<a<<endl; }
    int Geta()
    { return a; }
  private:
    int a;
};
class B:public A
{
  public:
    B(int i=0,int j=0):A(i),a(j),b(i+j)
    {  cout<<"B:Constructor called.\n"; }
    ~B()
    { cout<<"B:Destructor called.\n"; }
    void Print()
    {
      A::Print();
      cout<<b<<','<<a.Geta()<<endl;
    }
  private:
    int b;
    A a;
};
void main()
{
    B b1(8),b2(12,15);
    b1.Print();
    b2.Print();
}
```

答： A:constructor called.
A:constructor called.
B:Constructor called.
A:constructor called.
A:constructor called.
B:Constructor called.
8
8,0
12
27,15
B:Destructor called.
A:Destructor called.
A:Destructor called.
B:Destructor called.
A:Destructor called.
A:Destructor called.

23.
```
class A
{
  public:
    A(int i)
    { cout<<"Constructor in A."<<i<<endl; }
```

```
    ~A()
    {  cout<<"Destructor in A.\n";  }
};
class B
{
 public:
   B(int i)
   {  cout<<"Constructor in B."<<i<<endl;  }
   ~B()
   {  cout<<"Destructor in B.\n";  }
};
class C
{
 public:
   C(int i)
   {  cout<<"Constructor in C."<<i<<endl;  }
   ~C()
   {  cout<<"Destructor in C.\n";  }
};
class D:public A,public B,public C
{
 public:
   D(int i,int j,int k,int l):A(i),B(j),C(k),a(l)
   {  cout<<"Constructor in D."<<l<<endl;  }
   ~D()
   {  cout<<"Destructor in D.\n"; }
 private:
   A a;
};
void main()
{
   D d(3,4,5,6);
}
```

答： Constructor in A. 3
Constructor in B. 4
Constructor in C. 5
Constructor in A. 6
Constructor in D. 6
Destructor in D.
Destructor in A.
Destructor in C.
Destructor in B.
Destructor in A.

24.
```
class Matrix    {
 public:
   Matrix(int r,int c)
   {
    row=r;
    col=c;
    elem=new double[row*col];
   }
   double & operator ()(int x,int y)
   {  return elem[x*col+y];  }
   ~Matrix()
   {  delete []elem;  }
   void print(int i)
   {  cout<<elem[i];  }
 private:
   double *elem;
   int row,col;
};
void main()    {
   Matrix m(3,4);
   for(int i=0;i<3;i++)
     for(int j=0;j<4;j++)
     m(i,j)=4*i+j;
   for(i=0;i<3;i++)
     for(int j=0;j<4;j++)
     {
       m.print(4*i+j);
       cout<<"  ";
     }
   cout<<endl;
}
```

答： 0 1 2 3 4 5 6 7 8 9 10 11

25.
```
class A    {
 public:
   virtual void fun()
   {  cout<<"A::fun() called.\n";  }
};
class B:public A
{
 void fun()
  {  cout<<"B::fun() called.\n";  }
};
void ffun(A *pa)
{
   pa->fun();
}
void main()    {
   A *pa=new A;
   ffun(pa);
   B *pb=new B;
   ffun(pb);
}
```

答： A::fun() called.
B::fun() called.

26.
```
class A    {
 public:
   A()
   {  ver='A';  }
   virtual void print()
   {  cout<<"The A version "<<ver<<endl;  }
 protected:
   char ver;
};
class B1:public A    {
 public:
   B1(int i)
   {  info=i;ver='B';  }
   void print()
   {  cout<<"The B1 info: "<<info<<" version "<<v
er<<endl;  }
 private:
   int info;
};
class B2:public A    {
```

```
    public:
      B2(int i)
      { info=i; }
      void print()
      { cout<<"The B2 info: "<<info<<" version "<<v
er<<endl; }
    private:
      int info;
  };
  class B3:public B1   {
    public:
      B3(int i):B1(i)
      { info=i;ver='C'; }
      void print()
      { cout<<"The B3 info: "<<info<<" version "<<v
er<<endl; }
    private:
      int info;
  };
  void print_info(A *pa)
  {
     pa->print();
  }
  void main()   {
     A a;
     B1 b1(14);
     B2 b2(88);
     B3 b3(65);
     print_info(&a);
     print_info(&b1);
     print_info(&b2);
     print_info(&b3);
  }
```

答： The A version A
The B1 info: 14 version B
The B2 info: 88 version A
The B3 info: 65 version C

27.

```
  class B   {
    public:
      virtual void fun1()
      { cout<<"B::fun1().\n"; }
      virtual void fun2()
      { cout<<"B::fun2().\n"; }
      void fun3()
      { cout<<"B::fun3().\n"; }
      void fun4()
      { cout<<"B::fun4().\n"; }
  };
  class D:public B   {
    public:
      void fun1()
      { cout<<"D::fun1().\n"; }
      void fun2()
      { cout<<"D::fun2().\n"; }
      void fun3()
      { cout<<"D::fun3().\n"; }
      void fun4()
      { cout<<"D::fun4().\n"; }
  };
  void main()   {
     B *pb;
     D d;
     pb=&d;
     pb->fun1();
     pb->fun2();
     pb->fun3();
     pb->fun4();
  }
```

答： D::fun1().
D::fun2().
B::fun3().
B::fun4().

28.

```
  class A   {
    public:
      A()
      { cout<<"In A cons.\n"; }
      virtual ~A()
      { cout<<"In A des.\n"; }
      virtual void f1()
      { cout<<"In A f1().\n"; }
      void f2()
      { f1(); }
  };
  class B:public A   {
    public:
      B()
      { f1();cout<<"In B cons.\n"; }
      ~B()
      { cout<<"In B des.\n"; }
  };
  class C:public B   {
    public:
      C()
      { cout<<"In C cons.\n"; }
      ~C()
      { cout<<"In C des.\n"; }
      void f1()
      { cout<<"In C f1().\n"; }
  };

  void main()   {
     A *pa=new C;
     pa->f2();
     delete pa;
  }
```

答： In A cons.
In A f1().
In B cons.
In C cons.
In C f1().
In C des.
In B des.
In A des.

29.

```
class A   {
    public:
      virtual void print()=0;
  };
```

```
class B:public A   {
  public:
    void print()
    {  cout<<"In B print().\n";  }
};
class C:public B   {
  public:
    void print()
    {  cout<<"In C print().\n";  }
};
void fun(A *pa){
    pa->print();
}
void main()  {
    A *pa;
    B b;
    C c;
    pa=&b;
    fun(pa);
    pa=&c;
    fun(pa);
}
```

答： In B print().
In C print().

补充简答题

1请说明new、delete、malloc、free分别在c++程序中的作用和使用场合，以及它们之间的不同之处
malloc/free是C/C++标准库函数，new/delete是C++运算符。他们都可以用于动态申请和释放内存。
对于内置类型数据而言，二者没有多大区别。malloc申请内存的时候要制定分配内存的字节数，而且不会做初始化；new申请的时候有默认的初始化，同时可以指定初始化；
对于类类型的对象而言，用malloc/free无法满足要求的。对象在创建的时候要自动执行构造函数，消亡之前要调用析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制之内，不能把执行构造函数和析构函数的任务强加给它，因此，C++还需要new/delete。

2．结构（struct）和联合(union)两者有何区别?
(1). 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。
(2). 对于联合的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。

3.如果虚函数是有效的，那为什么不把所有函数设为虚函数?
不行。首先，虚函数是有代价的，由于每个虚函数的对象都要维护一个虚函数表，因此在使用虚函数的时候都会产生一定的系统开销，这是没有必要的。

4.继承层次中，为什么基类析构函数是虚函数?
编译器总是根据类型来调用类成员函数。但是一个派生类的指针可以安全地转化为一个基类的指针。这样删除一个基类的指针的时候，C++不管这个指针指向一个基类对象还是一个派生类的对象，调用的都是基类的析构函数而不是派生类的。如果你依赖于派生类的析构函数的代码来释放资源，而没有重载析构函数，那么会有资源泄漏。

5.为什么构造函数不能为虚函数?
虚函数采用一种虚调用的方法。需调用是一种可以在只有部分信息的情况下工作的机制。如果创建一个对象，则需要知道对象的准确类型，因此构造函数不能为虚函数。

6. 结构化程序设计的缺点：1) 数据与操作分离，代码可读性差，难以理解; 2) 代码重用性差。
面向对象编程的优点：提高开发效率和软件质量: 1) 更高层次的抽象;2) 数据封装;3) 更好地模块化支持; 4) 软件复用; 5) 对需求变更有更好的适应性。

7.函数副作用的危害以及如何消除。
危害:1) 破坏了程序的可移植性; 2) 降低了程序的可读性。

消除:常量指针。

8.静态全局变量的作用： 1) 防止名冲突; 2) 值可靠。
静态全局函数的作用： 1) 限制文件外部代码对函数的使用; 2) 对函数定义的补充。

9. 后期绑定的实现方法：创建一个虚函数表，记录所有虚函数入口的地址。对象的内存空间中含有一个指针指向其虚函数表。

10. 多态的作用：1) 提高语言的灵活性; 2) 实现高层软件的复用。

11. new/delete 重载的意义：频繁调用系统的存储管理，影响效率。程序自身管理内存，提高效率。

12. 引入模板函数的原因。1) 宏实现的缺陷:a) 只能实现简单的功能; b) 没有类型检查;c) 重复计算。
2) 函数重载的缺陷:a) 需要定义的重载函数太多 b) 定义不全。
3) 函数指针的缺陷: a) 需要定义额外参数; b) 大量指针运算; c) 实现复杂; d) 可读性差。

13.引入异常处理机制的原因：发现异常之处与处理异常之处不一致；使用函数参数程序结构不清楚。

14.在类的多层次继承结构中，类之间哪些函数是按作用域规则处理的？哪些函数是按多态性规则处理的？试编程说明之。
一般成员函数是按作用域规则处理的，虚函数是按多态性规则中的动态联编处理的。下列程序中有一般成员函数和虚函数。

```
#include <iostream.h>
class A{
public:
void Print()
{  cout<<"In A.\n";  }
virtual void fun()
{  cout<<"virtual A.\n";  }
};
class B:public A{
public:
void Print()
{  cout<<"In B.\n";  }
virtual void fun()
{  cout<<"virtual B.\n";  }
};
void text(A &a){  a.fun();  }

void main(){
   A a;
B b;
a.Print();
b.Print();
text(b);
}
```

15.一个类中是否必须有用户定义的构造函数？如果用户没有定义构造函数，又如何对创建的对象初始化
答：一个类用户可以不定义构造函数，这时系统自动提供一个默认的构造函数，并可用该构造函数对创建的对象初始化。

16.拷贝构造函数具有几个参数？它有类型转换的作用吗?
答：拷贝构造函数具有一个参数，即为该类对象的引用。拷贝构造函数没有类型转换作用。

17.静态成员属于类的，是否每个对象都可以引用该静态成员？答：可以。

18.常对象可以引用非常成员函数吗？非常对象可以引用常成员函数吗？答：不可以。可以。

19.友元函数能否访问类中的保护成员？友元函数访问类中私有成员与成员函数访问私有成员的形式相同吗？答：能。相同。

20.对象指针可以指向一个有名对象，它可以指向一个无名对象吗？如何实现？
答：对象指针可以指向一个有名对象，也可以指向一个无名对象。例如，对象指针指向堆对象就是一个例子。假定已知类A，定义该类对象指针pa：
A *pa;
pa=new A(8);
pa就是一个指向类A的无名对象的指针。

21.对象数组和对象指针数组的区别在哪里？
答：对象数组的元素是同一个类的若干个对象，对象指针数组的元素是指向同一个类的若干个对象指针。

22.在一个类中定义了多个子对象，其构造函数调用子对象的构造函数的顺序取决于什么？
答：构造函数调用子对象的顺序取决于定义子对象的顺序，而与构造函数的成员初始化列表中给出的子对象速设无关。

23.在继承关系中，派生类中包含基类所有成员，基类是否也包含派生类的部分成员？答：不包含。

24.构造函数不能继承，派生类的构造函数中是否应包含直接基类的构造函数和所有间接基类的构造函数签：派生类的构造函数中只包含直接基类的构造函数

25.派生类的析构函数中不包含直接基类的析构函数，对吗？答：不对。

26.派生类的对象可以给基类对象赋值吗？答：一般情况下不可以，只有在公有继承的情况下可以。

27.多重继承的二义性可以避免吗？答：可以避免，通常使用类名限定。

28.运算符重载使用成员函数方法和友元函数方法是否都可以？并且是没有区别的吗？
答： 两种方都法可以。有区别，参数个数不同。

29.运算符重载实际上通过函数来重新定义运算符的功能，运算符重载的功能直接通过函数调用是否可以
答： 可以。

30.多态性中对函数的选择从时间上来区分有哪两种方式？
答： 一种是在编译时选定函数，称静态联编方式，另一种是在运行时选定函数，称动态联编方式。

31.有虚函数是否就一定是动态联编？非虚函数是否就一定是静态联编？答： 不一定。一定是。

32.在多层次的继承结构中，基类与派生类中存在着虚函数，这时调用虚函数就一定实现动态联编吗？
答：不一定。

补充设计题

```
1.char* get_SO2_String_1() {
        char gas[] = "SO2";
```

```
	return gas;
}

char* get_SO2_String_2() {
	char *gas = "SO2";
	return gas;
}

char* get_NO2_String_1() {
	char gas[] = "NO2";
	return gas;
}

char* get_NO2_String_2() {
	char *gas = "NO2";
	return gas;
}

void print_air_condition(int m[][2], int day, intgas_index) {
	cout<< "\t" << day + 1;
	cout<< "\t" << (gas_index == 0 ? get_SO2_String_1() : get_NO2_String_1());
	cout<< "\t" << m[day][gas_index] <<endl;
}

void print_special_day(int m[][2], int day, intgas_index) {
	cout<< "\t" << day + 1;
	cout<< "\t" << (gas_index == 0 ? get_SO2_String_2() : get_NO2_String_2());
	cout<< "\t" << *(m[day] + gas_index) <<endl;
}

int main() {
	cout<< "\tDay\t" << "Gas\t" << "Number" <<endl;
	int SO2_NO2[7][2] = {{95}, {102,133}, 163, 94, 89, 76, 133, 54, {69}, 76};
	print_air_condition(SO2_NO2, 0, 1);
	print_air_condition(SO2_NO2, 3, 0);
	print_air_condition(SO2_NO2, 5, 1);

	print_special_day(SO2_NO2, 2, 1);
	print_special_day(SO2_NO2, 4, 0);

return 0;
}
```

2.
```
int get_int(){
int i;
if(!(cin>>i))
throw Invalid_Input("Was expecting an integer");
return i;
}
```

利用上述信息，实现Invalid_Input类，并写出main函数。要求main函数中，利用try/catch结构和get_int()函数，获取控制台输入的整数，并能catch非整数输入异常,输出异常信息。

答案

```
class Invalid_Input {
public :
Invalid_Input (char * in_ptr) : msg_ptr (in_ptr) { }
const char * const msg_ptr;
};
```

```
void main()
{
try{
        get_int();
}
catch (Invalid_Input& except){
cout << “Invalid input - ” << except.msg_ptr << endl;
}
}
```

3.（1）第一部分

银行ATM随处可见，使人们的生活更便捷。ATM系统提供的主要功能包括登录、查询余额、存款、取款。ATM系统提供一系列的操作界面供用户使用这些功能，并将信息存入数据库。

请你给出ATM系统的设计类图，要求写清各个类的属性及方法，并说明每个类的作用。

（2）第二部分

我们的生活中有太多的银行，可是他们各自的ATM一般只对自己的用户提供最大的方便，跨行操作会收取手续费。

联合公司致力于提供一个跨银行的平台，方便人们的生活。他们最近与三家大银行——小工银行（ICBank）、小农银行（ABank）、小建银行（CBank）联合推出了一款新型ATM，在这台ATM上，人们可以选择进入上述三家银行其中一家，之后的操作就仿佛在这家银行自有的ATM上进行一样。

该ATM系统提供的功能有登录、查询余额、存款、取款，这三家银行对这些功能的实现有各自不同的方法，对应有不同的界面以及数据库。

请你给出该ATM系统的设计类图，要求写清各个类的属性及方法，并说明每个类的作用。

4．按下列要求编程：

（1）定义一个描述矩形的类Rectangle，包括的数据成员有宽（width）和长（length）；（2）计算矩形周长；（3）计算矩形面积；（4）改变矩形大小。

```
#include <iostream.h>
class Rectangle
{
 public:
   Rectangle(int a,int b)
   {  width=a; length=b; }
   int Area()
   {  return width*length;  }
   int Periment()
{  return 2*(width+length);  }
   void Changesize(int a,int b)
{  width=a;  length=b;  }
   void Print();
  private:
   int width,length;
};
void Rectangle::Print()
{
   cout<<"AREA="<<Area()<<endl;
cout<<"PERIMENT="<<Periment()<<endl;
}

void main()
{
   Rectangle r(5,8);
   r.Print();
r.Changesize(3,9);
r.Print();
}
```

5.编程实现一个简单的计算器，从键盘上输入两个浮点数，计算出它们的加、减、乘、除运算的结果。

```
#include <iostream.h>
class ASMD
{
 public:
   ASMD(double a,double b)
   {  x=a; y=b; }
   void Addition()
   {  cout<<x+y<<endl;  }
   void Subtration()
   {  cout<<x-y<<endl;  }
   void Multiply()
   {  cout<<x*y<<endl;  }
   void Divison()
   {  cout<<x/y<<endl;  }
   void Print();
  private:
   double x,y;
};
void ASMD::Print()
{
   Addition();
Subtration();
Multiply();
```

```
Divison();
   }

   void main()
   {
      ASMD a(40,8);
      a.Print();
   }
```

6.编一个关于求多个某门功课总分和平均分的程序。具体要求如下：（1）每个学生信息包括姓名和某门功课成绩。（2）假设5个学生。（3）使用静态成员计算5个学生的总成绩和平均分。

```
   #include <iostream.h>
   #include <string.h>
   class Student
   {
    public:
       Student(char s[],int gr)
       {  strcpy(name,s); grade=gr; totalize+=gr; }
       static void Print();
     private:
       char name[10];
 int grade;
 static int totalize;
   };
int Student::totalize=0;
   void Student::Print()
   {
      cout<<"总成绩为 "<<totalize<<",平均成绩为 "<<totalize/5<<endl;
   }

   void main()
   {
      Student s1("ma",85),s2("Wang",96),s3("Li",82),s4("lu",78),s5("zhang",80);
Student::Print();
   }
```

7.按下列要求实现一个栈类的操作，该类名为Stack，包括如下操作：

（1）压栈操作：Push()；（2）弹栈操作：Pop();

（3）获取栈顶元素：Peer();

（4）判栈空操作：IsEmpty();（5）判栈满操作：IsPull()。

设栈最多可存放50个整数，成员用数组表示。编写一个程序，定义一个栈类的对象数组来验证该类操作。

答：栈的正常操作程序如下：

```
   #include <iostream.h>
   #include <stdlib.h>
   class Stack
   {
    public:
       Stack(int i);
       ~Stack()
 {  delete [] sta;  }
       void Push(int i);
 int Pop();
 void IsUpll()
 {
    if(tos==length)
 {
    cout<<"Stack is fill.\n";
    return;
 }
 }
 int IsEmpty()
 {
    if(tos==0)
 {
     cout<<"Stack underflow.\n";
    return 0;
 }
 }
     private:
       int *sta;;
    int tos,length;
   };
Stack::Stack(int i){
sta=new int[i];
if(!sta){
cout<<"Can't allocate stack.\n";
abort();
}
tos=0;
length=i;
}
void Stack::Push(int i){
sta[tos]=i;
tos++;
}
int Stack::Pop(){
tos--;
return sta[tos];
}
   void main()  {
Stack a(50);
a.Push(1);
a.Push(2);
……
cout<<a.Pop()<<',';
cout<<a.Pop()<<',';
……
cout<<a.Pop()<<','<<endl;
   }
```

判断栈的空、满操作，请读者修改上述程序。

8.按下列要求实现一个有关学生成绩的操作，该类名为Student。

（1）每个学生的信息包含有姓名（字符数组）和成绩（int型）。
（2）共有5个学生，用对象数组表示。（3）计算出5个学生中的最高分，并输出姓名及分数。

```
#include <iostream.h>
#include <string.h>
class Student
{
 public:
   Student(char s[],int gr)
   { strcpy(name,s); grade=gr; }
   friend void fun();
 private:
   char name[10];
 int grade;
};
Student ss[5]={Student("马力",85),Student("王欣
",96),Student("李明",82),
Student("赵亮",78),Student("张京",80)};
void fun(){
int k=0;
for(int i=0;i<5;i++)
if(ss[i].grade>ss[0].grade)
 k=i;
cout<<"最高分的学生姓名和成绩如
下: \n"<<ss[k].name<<','<<ss[k].grade<<endl;

}
 void main() {
fun();
 }
```

9.按如下要求编程验证子对象的有关操作。
（1）定义两个类A和类B。（2）在类B中有两个类A的对象one，two。
验证如下事实：
（1）在类B的构造函数中应该包含对两个类A的子对象的初始化项，被放在成员初始化列表中。
（2）在类B的默认构造函数中隐含着子对象的初始化项。
（3）在类B的析构函数中也隐含着子对象的析构函数。
（4）调用子对象构造函数的顺序。

```
#include <iostream.h>
class A
{
 public:
A()
{ cout<<"In A0.\n"; }
A(int i)
{ a=i; cout<<"In A1."<<a<<endl; }
~A()
{ cout<<"In A2."<<a<<endl; }
   int a;
};
class B
  {
   public:
B()
{ cout<<"In B0.\n"; }
B(int i,int j,int k):two(j),one(k)
{ b=i; cout<<"In B1.\n"; }
void Print()
{
cout<<b<<','<<one.a<<','<<two.a<<endl;
}
~B()
{ cout<<"In B2.\n"; }
   private:
    int b;
A one,two;
};

 void main()
 {
cout<<"构造函数的调用情况：\n";
static B bb0;
    B bb(1,2,3);
 cout<<"输出对象bb的数据成员值：\n";
    bb.Print();
cout<<"析构函数的调用情况：\n";
 }
```

10.设计一个程序，一行是信息，下一行画线，所画的线与信息行同长。例如，

```
#include <iostream.h>
#include <string.h>
class Line
{
public:
Line(int i)
{ length=i; }
void Show()
{
for(int i=0;i<length;i++)
cout<<'_';
cout<<endl;
}
private:
int length;
};
class Message {
public:
Message(char *ptr) {
msg=new char[strlen(ptr)+1];
strcpy(msg,ptr);
}
~Message() { delete []msg; }
void Show() {
int i=0;
cout<<msg<<endl;
}
```

```
private:
char *msg;
};
class MsgLine:public Line,public Message {
 public:
MsgLine(char *ptr):Message(ptr),Line(strlen(ptr)) { }

void Show() {
Message::Show();
Line::Show();
}
};

   void main()  {
MsgLine string1("C++");
string1.Show();
MsgLine string2("Progrmming");
string2.Show();
   }
```

11. 编程求圆、圆内接正方形和圈外切正方形的面积和周长。要求使用抽象类。

答： 编程如下：

```
   #include <iostream.h>
   const double PI=3.1415;
   class Shape
   {
public:
Shape(double i)
{ r=i; }
virtual void Area()=0;
virtual void Perimeter()=0;
protected:
double r;
};
class Circle:public Shape
{
public:
Circle(double i):Shape(i)
{ }
void Area()
{ cout<<"圆的面积是 "<<PI*r*r<<endl; }
void Perimeter()
{ cout<<"圆的周长是 "<<2*PI*r<<endl; }
};
class In_Square:public Shape
{
 public:
In_Square(double i):Shape(i)
{ }
void Area()
{ cout<<"圆内接正方形的面积
是 "<<2*r*r<<endl; }
void Perimeter()
{ cout<<"圆内接正方形的周长
是 "<<4*1.414*r<<endl; }
};
class Ex_Square:public Shape
{
 public:
Ex_Square(double i):Shape(i)
{ }
void Area()
{ cout<<"圆外切正方形的面积
是 "<<4*r*r<<endl; }
void Perimeter()
{ cout<<"圆外切正方形的周长是 "<<8*r<<endl; }
};

   void main()
   {
Shape *ps;
ps=new Circle(8);
ps->Area();
ps->Perimeter();
ps=new In_Square(8);
ps->Area();
ps->Perimeter();
ps=new Ex_Square(8);
ps->Area();
ps->Perimeter();
      delete ps;
   }
```