

第一章习题

- 1, C++程序文件扩展名是: .cpp, 由于 C++语言中保留了 main 函数, 所以称为混合型语言, C 语言是典型的面向过程语言, C++ 兼容 C 语言, 所以也可以使用 C++编制面向过程程序。
- 2, C++程序中两种注释方式: //注释一行 和 /* 注释多行 */
- 3, cout, cin 用于 C++程序中的输入输出, 使用时必须添加包含命令: #include "istream.h", 与 cout<<endl;等效的语句可以写作 (多种形式): cout<<'\\n'; cout<<'\\n'
- 4, int x,y,z; cin>>x>>y>>z;
用于向变量输入数据, 输入时三个数据间可以使用 空格, tab, 回车 分隔。
- 5, 若 int x; double y,z; 则利用 cin>>x>>y>>z;输入 12.23.34 则 x, y, z 的值分别是: 12, 0.23, 0.34
- 6, 使用 cout 和 cin 进行数据的输入输出时, 若包含头文件命令为#include <iostream>, 则需要同时使用语句: using namespace std;
- 7, 若程序中要使用字符串对象则要包含头文件 #include <string.h>
- 8, 使用数学函数则要包含头文件 #include <cmath.h>
- 9, 使用泛型算法中的 sort 方法并按逆序排列某数组内容则要添加头文件: functional.h
- 10, int z (2); 语句的作用是: 定义整型变量 z 并初始化值为 2, 与之等价的语句可写作: int z=2;
- 11, void 是定义声明函数函数时使用的类型, void 定义的函数表示该函数 无 返回值。此类函数中仍可以使用 return 语句, 使用时须注意: return 语句后不能有表达式, 函数声明时给出函数类型,函数名,函数参数,其中无需给出 参数名
- 12, const int k=2; 则表示 k 是 整型常量值为 2, main 函数中 k=11; 语句是否可执行? 不可执行。使用此方式在函数中定义时必须注意: const 限定的变量定义时必须初始化, 若在类的数据成员定义时加 const 限定则必须注意的是: const 限定的数据成员必须在构造函数的初始化列表中初始化
- 13, 如下语句是否正确, 说明原因。
#define PI 3.1415q2b 正确, PI 表示 3.1415q2b
Const double PI=3.1415q2b; 错误, PI 是 double 型数据
- 14, 函数重载时, 若要求重载的函数参数类型与原函数类型一致, 则参数 个数 不能相同, 若要求重载函数参数个数与原函数一致, 则参数 类型 不能一致。
重载函数的条件是 重载函数的参数个数与参数类型不能与员函数完全相同。
- 15, C++中的常量表示: -32768, 0, 123L, 0L, 0A, 0xA, 0110, 0108, 0.2f, 123.4, 3.2F, '0101', '\\101', NULL, 正确的常量形式有哪些?
- 16, new 语句的作用是 动态分配内存, int *p; p=new int [10]; 表示 动态分配 10 个整型内存, 并用 p 指向这些内存单元,
int *p; p=new int (10); 作用是 动态分配一个整型内存, 并初始化值为 10。分配一块存放整型数据 10 的内存单元, 并让一个指针 p 指向该单元的语句是: int *p=new int (10);
- 17, 数据类型 &别名 =对象名; 是定义引用的格式。
引用的实质就是: 1, 引用实际上就是变量的 别名, 使用引用和直接使用变量一样。引用不占用 内存。若有 int x=1; int & r=x; 则&x 表示 变量 x 的地址, &r 表示 变量 x 的地址。语句 int x=1; int & r; r=x; 是否正确并说明原因。 错误, 引用必须在定义时初始化

18, 有数组 `int a[10]` 若要定义一个 `r` 为 `a` 的引用语句为: `typedef int array[10]; array &r=a;`

19, 若下列语句在某函数体内定义:

`const int *p;` 是否正确, 并说明原因: 正确

`int * const p;` 是否正确, 并说明原因: 错误

`const int * const p;` 是否正确, 并说明原因: 错误

20, 泛型算法是 C++ 标准模板库提供的一组数组操作的方法, 分别是: `copy`, `reverse`, `reverse_copy`, `sort`, `find` 等, 这些方法要使用必须包含头文件: `algorithm.h`, `sort` 方法对数组排序时, 默认为 升序 排序, 若要排序与之相反, 可用 `greater` 操作符, 此时要包含头文件: `functional.h`, 使用泛型算法的 `find` 方法时也需要包含该头文件。

21, 若有数组 `int a[10]`, `b[5]`, 并已存入数据, 则利用 (输出时数据间用*号分隔)

`copy` 输出 `a` 数组的 `a[1]` 到 `a[5]` 元素: `copy(a,a+5,ostream_iterator<int>(cout,"*"));`

逆序输出 `a` 数组 `a[1]` 到 `a[5]` 元素: `reverse_copy(a,a+5,ostream_iterator<int>(cout,"*"));`

对 `a` 数组 `a[1]` 到 `a[5]` 元素升序排序: `sort(a,a+5);`

对 `a` 数组 `a[1]` 到 `a[5]` 元素降序排序: `sort(a,a+5,greater<int>());`

将 `a` 数组的 `a[1]` 到 `a[5]` 元素复制到 `b` 数组中: `copy(a,a+5,b)`

22, 进行输入输出格式控制时需要用到 C++ 中的格式操控符, 请说明如下符号含义: `dec`, `oct`, `hex`, `endl`, `setw`, `setfill`, `setiosflags`, `ios_base::right`。记忆课本上表 1.8。

`int a=22,b=0101;`

```
double c=1; cout<<setfill('*'); cout<<oct<<a<<endl;
cout<<b<<endl; cout<<dec<<b<<endl; cout<<setw(5)<<b<<endl;
cout<<setw(15)<<setiosflags(ios_base::showpoint)<<b<<endl;
cout<<setw(15)<<setprecision(12)<<c<<endl;
cout<<resetiosflags(ios_base::showpos)<<b<<endl;
```

请写出输出结果:

```
26
101
65
***65
*****65
**1.000000000000
65
```

22, 给出程序运行结果

```
#include <iostream>
#include <algorithm>
#include <functional>
using namespace std;
main()
{ double a[]={1.1,4.4,3.3,2.2},b[4];
  copy(a,a+4,ostream_iterator<double>(cout," ")); cout<<endl;
  reverse_copy(a,a+4,ostream_iterator<double>(cout," ")); cout<<endl;
```

```

copy(a,a+4,b);
copy(b,b+4,ostream_iterator<double>(cout, " "));    cout<<endl;
sort(a,a+4);
copy(a,a+4,ostream_iterator<double>(cout, " "));    cout<<endl;
reverse_copy(a,a+4,b);
copy(b,b+4,ostream_iterator<double>(cout, " "));    cout<<endl;
sort(b,b+4,greater<double>());
copy(b,b+4,ostream_iterator<double>(cout, " "));
}

```

```

1.1 4.4 3.3 2.2
2.2 3.3 4.4 1.1
1.1 4.4 3.3 2.2
1.1 2.2 3.3 4.4
4.4 3.3 2.2 1.1
4.4 3.3 2.2 1.1

```

23, 若输入数据为 10 10 10 回车, 给出程序运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{   int a,b, c;
    cin>>dec>>a;    cin>>oct>>b;    cin>>hex>>c;
    cout<<endl;    cout<<a<<endl;    cout<<b<<endl;    cout<<c<<endl;
    cout<<endl;
    cout<<oct;    cout<<a<<endl;    cout<<b<<endl;    cout<<c<<endl;
    cout<<endl;
    cout<<hex;    cout<<a<<endl;    cout<<b<<endl;    cout<<c<<endl;
    cout<<endl;
}

```

```

10 10 10

10
8
16

12
10
20

a
8
10

```

24, 给出程序运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    int a=11,b=12, c=13;
    cout<<setw(5);    cout<<setfill('*');
    cout<<setiosflags(ios_base::right)<<a<<endl;
    cout<<b<<endl;    cout<<c<<endl;
    cout<<resetiosflags(ios_base::right)<<a<<endl;
    cout<<setw(5);    cout<<setfill('*');
    cout<<setiosflags(ios_base::left)<<a<<endl;
    cout<<b<<endl;    cout<<c<<endl;
    cout<<endl;
}

```

```

****11
12
13
11
11****
12
13

```

25. 给出程序运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    int a=11,b= -12, c=13;
    cout<<setw(5);    cout<<setfill('*');
    cout<<setiosflags(ios_base::right) <<setiosflags(ios_base::showpoint)<<a<<endl;
    cout<<b<<endl;    cout<<c<<endl;
    cout<<resetiosflags(ios_base::right);
    cout<<setw(5);    cout<<setfill('*');
    cout<<setiosflags(ios_base::left)<<a<<endl;
    cout<<b<<endl;    cout<<c<<endl;
    cout<<endl;
}

```

```

***11
-12
13
11***
-12
13

```

26, 给出程序运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    double x=1.123,y=2,z=123.3333333;
    cout<<setw(5);    cout<<setfill('*');
    cout<<setiosflags(ios_base::right) <<setprecision(3)<<x<<endl;
    cout<<y<<endl;    cout<<z<<endl;
    cout<<resetiosflags(ios_base::right);
    cout<<setw(20);    cout<<setfill('*');
    cout<<setiosflags(ios_base::left)<<x<<endl;
    cout<<y<<endl;    cout<<z<<endl;
    cout<<endl;
}

```

```

*1.12
2
123
1.12*****
2
123

```

27, 给出程序运行结果

```

#include <iostream>
#include <iomanip>
using namespace std;
void main()
{
    int a=22,b=0101;    double c=1;
    cout<<setfill('*');    cout<<oct<<a<<endl;    cout<<b<<endl;
    cout<<dec<<b<<endl;    cout<<setw(5)<<b<<endl;
    cout<<setw(15)<<setiosflags(ios_base::showpoint) <<b<<endl;
    cout<<setw(15)<<setprecision(12)<<c<<endl;
    cout<<resetiosflags(ios_base::showpos) <<b<<endl;
}

```

```

26
101
65
***65
*****65
***1.000000000000
65

```

28, 给出程序运行结果

```

#include "iostream"
using namespace std;
typedef myarray[5];
void main()
{
    myarray x={1,2,3,4,5}; myarray &a=x;    myarray &r=a;
    for(int i=0;i<5;i++)    cout << x[i] ;
    for(i=0;i<5;i++)    cout << a[i] ;
    for(i=0;i<5;i++)    cout << r[i] ;
}
123451234512345

```

第二章习题

- 1, 结构体定义使用关键字 struct , 结构体内可定义数据成员和 成员函数 , 结构体内的成员可以使用 private , public 限定, 未加限定时默认为 public 。若有结构体名为 stru , 且其成员 x 限定为 private , y 成员限定为 public , z 成员未加限定, 则由 stru 定义的结构体对象 s1 可否在 main 函数中通过如下形式使用 x , y , z 成员: s1.x=1; s1.y=2; s1.z=3; 请说明原因: s1.x 不可使用, 因为 x 是私有成员
- 2, 结构体内定义的与结构体同名的函数称为 构造 函数, 定义结构体对象时系统会自动调用该函数, 在定义结构体对象时, 若要初始化对象如: point p (1, 2); 形式, 则要求结构体 point 中必须定义了 有 2 个参数的构造函数
- 3, 将结构体定义中的 struct 修改为 class , 则结构体定义变为类的定义, 类定义中若成员未加 private , public 限时, 默认为 private 。
- 4, 面向对象程序设计中, 可以将一组密切相关的函数统一封装在一个 对象 中, 从而可以合理而有效的避免全局变量的使用。结构化程序设计使用的是 功能 抽象, 面向对象程序设计不仅使用 功能抽象 而且能进行 数据 抽象, 对象实际就是 功能抽象 和 数据抽象 的统一。
- 5, 面向对象程序设计方法不是以函数过程和数据结构为中心的, 而是以 对象 代表求解问题的中心环节, 它追求的是 现实问题空间 与 软件系统解空间 的近似和直接模拟。
- 6, 面向对象程序设计的特点是 对象 、 抽象和类 、 封装 、 继承 、 多态 。
- 7, 对象名用来标识一个具体对象, 用 数据 表示对象的属性, 一个属性就是描述对象 静态 特征的一个数据项。 操作 是描述对象的动态特征的一个 函数 序列, 也称为 方法 或 服务 。数据称为数据成员, 函数称为成员函数。C++中 对象 是系统中用来描述客观事物的一个实体, 使构成系统的基本单位。一个对象有一组 属性 和对这组属性进行操作的 成员函数 构成。
- 8, 类由 类名 、 一组属性 和 一组操作 三部分构成。类的属性只是性质的说明, 对象 的属性才是具体的数据。类是具有相同的 属性 和 操作 的一组对象的集合, 他为属于该类的全部对象提供了统一的抽象描述, 其内部包括 属性 和 操作 两个主要部分。这两部分也是对象分类的依据。类的作用是 定义对象 。C++中将对象称为类的一个 实例 。所谓“一个类的所有对象具有相同的属性”是指 属性的个数 、 名称 和 数据类型 相同, 各个对象的属性值则可以互不相同。
- 9, 按照面向对象的封装原则, 一个对象的属性和操作是紧密结合的, 对象的属性只能由 这个对象的操作 来存取。对象的操作分为 内部 操作(private 限定的成员函数)和 外部 操作 (public 限定的成员函数)。
- 10, 继承 是一个类可以获得另一个类的特征的机制, 继承支持 层次 概念。
- 11, 不同的对象可以调用相同名字的函数, 但可能导致完全不同的行为的现象称为 多态性 。
- 12, string 类成员函数有 size 、 substr 、 swap 、 find 、 begin 、 end 。在程序中使用 string 类定义对象必须包含头文件: string.h 。定义 string 类对象的格式: 定义 string 对象 s1 , 并初始化值为 “hello”, 请写出语句 (两种形式): string s1="hello" 、 string s1("hello")
- 13, 若有 string 类对象 s1 , s2= "123" , s3= "456" , s1=s2+s3; cout <<s1;输出结果为: 123456
- 14, 请写出程序运行结果:

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
void main()
{
    string str1("hello,");
    string str2="everyone!";
    cout<<str1[0]<<str1[5]<<"    " <<str1<<endl;  cout<<str2[0]<<str2[8]<<"    " <<str2<<endl;
    cout<<"length of the "<< str1<<" is "<< str1.size()<<endl;
    string str3;
    str3=str2.substr(3,3);  cout<<str3<<endl;
}
```

```
h,    hello,
e!    everyone!
length of the hello, is 6
ryo
```

15, getline 可以使用 cin 读入一行字符串给 string 类的对象

string s1; getline(cin,s1,'\n'); 若从键盘输入 “hello”，则 cout<<s1;输出结果为: hello

16, 如下程序的输出结果为: 1,4,4,25

```
string s="4-25,2002";  int i=s.find("-",0);  string s1=s.substr(0,i);  int j=s.find(",",0);
string s2=s.substr(i+1,j-i-1);  cout<<i<<endl<<j<<endl<<s1<<endl<<s2<<endl;
```

```
1
4
4
25
```

17, complex 是 C++标准类库提供的定义复数的类，使用时要包含头文件 complex.h，complex 定义对象时需要两个初始值：实部和虚部，并且 complex 是一个模板类，请给出实部值为 1，虚部值为 2 的复数 n1 的定义：complex<int>n1(1,2);，请写出输出语句：cout<<n1.real()<<"+"<<n1.imag()<<"i"得到结果 “1+2i” 形式

18, 有定义: string s1="hello",s2="every";

请写出将 s1 内的字符串反转的语句（两种形式）:

reverse(s1.begin(),s1.end()) 或 reverse(&s1[0],&s1[0]+5)。

请写出将 s1 复制给 s2 的语句（两种形式）:

copy(s1.begin(),s1.end(),s2.begin()) 或 copy(&s1[0],&s1[0]+5,&s2[0])。

请写出利用 copy 输出 s1 的语句（两种形式）:

Copy(s1.begin(),s1.end(),ostream_iterator<char>(cout)) 或 _____。

请写出对 s1 进行升序排序的语句（两种形式）:

sort(s1.begin(),s1.end()) 或 _____。

请写出对 s1 进行降序排序的语句（两种形式）:

Sort(s1.begin(),s1.end(),greater<char>()) 或 _____。

请写出将 s1 逆向复制给 s2 的语句（两种形式）:

Reverse_copy(s1.begin(),s1.end(),s2.begin()) 或 _____。

请写出交换 s1 和 s2 内容的语句（两种形式）：

s1.swap(s2) 或 s2.swap(s1)。

19，给出程序运行结果

void main()

```
{  string s1="hello",s2="every";  sort(s1.begin(),s1.end(),greater<char>());  
  copy(s1.begin(),s1.end(),s2.begin());  sort(s2.begin(),&s2[3]);  
  cout<<s1<<endl; cout<<s2<<endl;  
  s2.swap(s1);  reverse(&s2[3],&s2[5]);  cout<<s2<<endl;  
}
```

```
o11he  
11ohe  
o11eh
```

20，使用 string 定义对象时，string s= 'ok'；如此定义是否正确？

第三章习题 函数和函数模板

备注：※标示的题目选自《高等教育自学考试考前密押试卷》

1. C++中函数的参数传递有两种方式： 传值 和 传引用
2. 只有 引用 作函数形参时，才是传地址方式，是 双向 传递
3. 默认参数必须放在参数序列的 后部，函数调用时，如果一个默认参数需要指名一个特定值，则 其前 的所有参数都必须赋值
4. 使用关键字 inline 修饰的函数，称为内联函数。特点：代码一般短小，不能含有循环语句，switch 语句
5. 定义函数模版的关键字是 template，C++还专门定义一个仅仅用在模版中的关键字 typename，可以代替 class
6. 函数声明 `int &f1(float, int);` 的含义是：定义函数 f1，含有两个形参，一个为单精度实型，另一个为基本整型，返回一个整型的引用
7. 函数 `compute` 的返回值是一个单精度实型引用，含有三个参数，前两个为整型，第三个为单精度实型，则声明此函数的原型为：`float &compute (int, int, float);`
8. 函数 `f2` 返回一个指向整型数据的指针，含两个参数，一个是字符型，另一个为 `int` 类型的引用，声明此函数的原型为：`int *f2 (char, int &);`
9. 已定义函数 `int min (int, int)` 功能为求取两个整数中的最小值，则欲求出整数 4, 8, 1 中的最小值，调用方式错误的是 (D)
 - A. `min (4, min (8, 1));`
 - B. `min (8, min (4, 1));`
 - C. `min (1, min (8, 4));`
 - D. `min (4, 8, 1);`
10. 下列关于函数参数默认值的描述中，正确的是 (B)
 - A. 函数参数的默认值只能设置一个
 - B. 若一个函数含有多个参数，其中一个参数设置成默认值后，其后所有参数都必须设置默认值
 - C. 若一个函数含有多个参数，则设置默认参数时可以不连续设置默认值
 - D. C++语言中函数都必须设有默认值
11. 已知函数 `f` 的原型是 `void f(int *a, long & b);`，变量 `v1`、`v2` 的定义是：`int v1; long v2;` 下列调用语句中正确的是 (D)
 - A) `f(v1, &v2);`
 - B) `f(v1, v2);`
 - C) `f(&v1, &v2);`
 - D) `f(&v1, v2);`
12. 下面是一个模板声明的开始部分：
`template<typename T> double` 由此可知 (D)
 - A) 这可能是一个函数模板的声明
 - B) 这可能是一个类模板的声明

C) 这既可能是一个函数模板的声明，也可能是一个类模板的声明

D) 这肯定是一个错误的模板声明

13. 下面函数声明错误的是 (D)

A. void fun(int x, int y=0); B. int fl(float i=0, char c=' a');

C. float *f(float, float); D. double abc(int a=0, int b);

14. 找出下面程序段中的错误

```
int fun( int x, const int y)
{ int z ;
  y++ ;          //形参 y 使用 const 修饰，只能被使用不能改变
  z=x+y ;
  return z ;
}
```

15. 给出下面程序的输出结果

```
#include<iostream>
using namespace std;
template<class T>
T add(T x, T y)
{ return x+y; }
void main() {
  cout<<add(1, 7)<< " \t" <<add(2.0, 5.0)<<endl;
}
```

8 7

※16. 一个函数功能不太复杂，但要求被频繁调用，选用 (B)

A. 重载函数 B. 内联函数 C. 递归函数 D. 嵌套函数

※17. 下列有关重载函数的说法正确的是 (C)

A. 重载函数必须具有不同的返回值类型 B. 重载函数参数个数必须相同

C. 重载函数必须有不同的形参列表 D. 重载函数名可以不同

※18. 下列给字符数组进行初始化，正确的是 (D)

A. char s1[]= ' \n' ; B. char s2[3]= "xyz" ;

C. char s3[][3]={ 'a' , 'x' , 'y' }; D. char s4[2,4]={ "xyz" , "mnp" };

※19. 对于 int *pa[5]; 的描述，正确的是 (D)

- A. pa 是一个指向数组的指针，所指向的数组是 5 个 int 型元素
- B. pa 是一个指向某个数组中第 5 个元素的指针，该元素是 int 型变量
- C. pa[5]表示某个数组的第 5 个元素的值
- D. pa 是一个具有 5 个元素的指针数组，每个元素是一个 int 型指针

※20. 下列 for 循环的循环体执行次数为 (D)

```
for(int i(0), j(10); i=j=4; i++, j--)
```

- A. 0
- B. 1
- C. 4
- D. 无限

※21. 在 C++中，编写一个内联函数 Fun，使用 int 类型的参数，求其平方并返回，返回值也为 int 类型，下列定义正确的是 (C)

- A. int Fun (int x) { return x*x ; }
- B. int inline Fun (int x) { return x*x ; }
- C. inline int Fun (int x) { return x*x ; }
- D. int Fun (int x) { inline return x*x ; }

※22. 若有以下定义，则说法错误的是 (D)

```
int a=100, *p=&a;
```

- A. 声明变量 p，其中*表示 p 是一个指针变量
- B. 变量 p 经初始化，获得变量 a 的地址
- C. 变量 p 只可以指向一个整型变量
- D. 变量 p 的值为 100

※23. 以下关于函数模板叙述正确的是 (C)

- A. 函数模板也是一个具体类型的函数
- B. 函数模板的类型参数与函数的参数是同一个概念
- C. 通过使用不同的类型参数，函数模板可以生成不同类型的函数
- D. 用函数模板定义的函数没有类型

※24. C++是通过引用运算符 & 来定义一个引用的。

※25. C++语言中如果调用函数时，需要改变实参或者返回多个值，应该采取引用方式。

※26. 有 int a=1, b=2; 则表达式 a+++-b 的值是 2

※27. 设函数 sum 是由函数模板实现的，并且 sum (3, 6) 和 sum (4.6, 8) 都是正确的函数调用，则函数模板具有 2 个类型参数

※28. 将执行对象的引用作为函数的形参，形参是对象的引用，实参是 对象名。

※29. 下面的类定义中有一处错误，请用下划线标出并给出修改意见

```
(1) #include<iostream.h>

class A
{private:
    int x,y;
```

```
public:
```

void fun(int i, int j) //函数调用只提供一个参数，fun 函数有两个形参，所以应定义一个默认参数，可以修改为： void fun (int i, int j=0)

```
{x=i; y=j; }
void show( )
{cout<<x<<" "<<y<<endl; }
};
void main( )
{ A a1;
  a1. fun(2);
  a1. show( );
}
```

(2) 实现数值，字符串的交换

```
#include<iostream>
#include<string>
using namespace std;
template <class T>
void swap(T &a, T &b)
{T temp;
temp=a; a=b; b=temp;
}
void main( )
{int a=5, b=9;
char s1[]="hello", s2[]="hi";// swap(s1,s2) 交换的是地址, 应该改为:
char *s1= "Hello", *s2= "hi"
swap(a, b);
swap(s1,s2);
cout<<"a="<<a<<", b="<<b<<endl;
cout<<"s1="<<s1<<", s2="<<s2<<endl;
}
```

※30. 完成程序题

(1) 程序实现大写字母转换成小写字母

```
#include<iostream.h>
void main( )
{ char a;
第三章第 4 页
```

```

    int i=32;
cin>>a;
if(a<=' Z' && a>=' A' ) a=a+i;
cout<<a<<endl;
}

```

(2) 完成下面类中成员函数的定义

```

#include<iostream.h>
template<class T>
class f
{ private:
    T x,y,s;
public:
    f(T a=0, T b=0) { x=a; y=b; }
    void sum( )
    { s=x+y; }
    T gets( );
};

template<class T>
T f:: gets( )
{return s;}

void main( )
{ f <float> a(1.5, 3.8);
a.sum( );
cout<<a.gets( )<<endl;
}

```

※30. 程序设计题（10 分）

求 n (n=3) 个学生的最高分和最低分及姓名，已有 student 类声明和 main 函数，完成 student 类的实现部分

```

#include<iostream.h>
#include<string.h>
class student
{ char name[10];
int deg;
public:

```

```

Student(char na[ ]= " ", int d=0);
char * getname( );
friend int compare(student &s1, student &s2);
int getdeg( );
};



---


void main( )
{
student st[]={ student( "王强" , 74), student( "李刚" , 68), student ( "张旭" , 84) };
int i=0, min=0, max=0;
for(i=1; i<3; i++)
{if(compare(st[max], st[i])==1) max=i;
 if(compare(st[min], st[i])==1) min=i;
}
cout<< "最高分: " <<st[max].getdeg( )<< "姓名: " <<st[max].getname( )<<endl;
cout<< "最低分: " <<(*st+min).getdeg( )<< "姓名: " <<st[max].getname( )<<endl;
}

```

各成员函数定义如下:

```

student::student(char na[], int d)
{strcpy(name, na);
deg=d;
}

char * student::getname() {return name;}

int compare(student &s1, student &s2)
{ if(s1.deg>s2.deg) return 1;
else if(s1.deg==s2.deg)return 0;
else return -1;
}

int student::getdeg()
{return deg;}

```

第四章习题 类和对象

- 假定 AB 为一个类，则执行“AB a[10];”语句时，系统自动调用该类的构造函数的次数为 10 。
- 对一个类中的数据成员的初始化可以通过构造函数中的 函数体内赋值语句 实现，也可以通过构造函数中的 初始化列表 实现。
- 在下列函数原型中，可以作为类 AA 构造函数的是（ D ）
A) void AA(int); B) int AA(); C) AA(int)const; D) AA(int);
- 有如下类定义：

```
class MyClass{  
    Int value;  
public:  
    MyClass(int n): value (n) {}  
    int gerValue()const{ return value;}  
};
```

则类 MyClass 的构造函数的个数是 A
A) 1 个 B) 2 个 C) 3 个 D) 4 个
- 在类的对象被创建的时候，构造函数会被自动调用。
- 在类中，静态成员为类的所有对象所共享。
- 在面向对象的程序设计中，将数据和处理数据的操作封装成一个整体就定义了一种事物的类型，称作“类”。类是一种抽象的概念，属于该类的一个实例叫做“对象”。
- 一个类的构造函数 D。
A. 可以有不同的返回类型 B. 只能返回整型
C. 只能返回 void 型 D. 没有任何返回类型
- 拷贝构造函数用于（ C ）。
A. 创建新对象 B. 返回对象 C. 将已存在的对象赋值给新对象 D. 已存在的对象互相赋值。
- 有以下的类定义

```
class MyClass  
{public:  
    MyClass ( ) {cout<<1; }  
};
```

则执行语句 MyClass a, b[2], *p[2]; 后，程序的输出结果是（ D ）

A. 11 B. 111 C. 1111 D. 11111

- 有如下类声明：

```
class Foo { int bar; };
```


则 Foo 类的成员 bar 是(C)

A) 公有数据成员 B) 公有成员函数 C) 私有数据成员 D) 私有成员函数

12. 下列关于 this 指针的叙述中, 正确的是(D)

A) 任何与类相关的函数都有 this 指针

B) 类的成员函数都有 this 指针

C) 类的友元函数都有 this 指针

D) 类的非静态成员函数才有 this 指针

※13. 下列特征中, C 和 C++共有的是(B)

A. 继承 B. 函数定义不能嵌套 C. 封装 D. 多态性

※14.类的析构函数是对一个对象进行哪种操作时自动调用的(B)

A. 建立 B. 撤销 C. 赋值 D. 引用

※15.假定 AA 为一个类, a() 为该类公有的函数成员, x 为该类的一个对象, 在访问 x 对象中函数成员 a() 的格式为(B)

A. x.a B. x.a() C. x—>a D. (*x).a()

※16.关于对象概念的描述中, 说法错误的是(A)

A. 对象就是 C 语言中的结构变量 B. 对象代表着正在创建的系统中的—个实体

C. 对象是类的一个变量 D. 对象之间的信息传递是通过消息进行的

※17.关于 new 运算符的下列描述中, 错误的是(D)

A. 它可以用来动态创建对象和对象数组

B. 使用它创建的对象或对象数组可以使用运算符 delete 删除

C. 使用它创建对象时要调用构造函数

D.使用它创建对象数组时必须指定初始值

※18. 已知: p 是一个指向类 A 数据成员 m 的指针, A1 是类 A 的一个对象。如果要给 m 赋值为 5, 正确的是(C)

A. A1.p=5; B. A1—>p=5; C. A1.*p=5; D. *A1.p=5;

※19. 定义析构函数时, 说法正确的是(C)

A. 其名与类名完全相同 B. 返回类型是 void 类型

C. 无形参, 也不可重载 D. 函数体必须有 delete 语句

※20.在 C++语言中, 数据封装要解决的问题是(D)

A. 数据的规范化 B. 便于数据转换 C. 避免数据丢失 D. 防止不同模块间数据的非法访问

※21.假定一个类的构造函数为 A(int aa, int bb){a=aa++; b=a*++bb; }, 则执行 A x(4, 5); 语句后, x.a 和 x.b 的值分别为(C)

A. 4 和 5 B. 4 和 20 C. 4 和 24 D. 20 和 5

※22..在 C++中, 访问一个对象的成员所用的成员运算符是。。

※23.如果要把类 B 的成员函数 void fun()说明为类 A 的友元函数,则应在类 A 中加入语句 friend void fun();

※24. 下面代码中有一处错误, 请用下划线标出并给出修改意见

(1) 输出最小值

```
#include<iostream.h>
```

```
class Test
```

```
{ int a, b;
```

```
int getmin()
```

```
{return (a<b? a: b); } //getmin()函数的访问权限为 private, 在主函数中不能通过对象名直接访问, 可以放到 public 权限后定义
```

```
public:
```

```
int c;
```

```
void setValue(int x1, int x2, int x3)
```

```
{a=x1; b=x2; c=x3; }
```

```
int GetMin( );
```

```
};
```

```
int Test::GetMin( )
```

```
{ int d=getmin();
```

```
return (d=d<c? d:c);
```

```
}
```

```
void main()
```

```
{Test t1;
```

```
t1.setValue(34,6,2);
```

```
cout<<t1.getmin()<<endl;
```

```
}
```

※25. 完成程序题 (下面是一个三角形三边, 输出其面积 C++程序, 在下划线处填上正确的语句)

```
#include<iostream.h>
```

```
#include<math.h>
```

```
void area( )
```

```
{double a,b,c;
```

```
cout<<"input a b c:";
```

```
cin>>a>>b>>c;
```

```
if(a+b>c && a+c>b && c+b>a)
```

```
{ double l=(a+b+c)/2;
```

S=sqrt((s-a)*(s-b)*(s-c));

```
cout<<"the area is:"<<s<<endl;
}
else cout<<"error"<<endl;
}
void main( )
{ area( ); }
```

※26. 读程序，分析输出结果

```
#include<iostream.h>
class AAA{
    int A,B;
public:
    AAA(int i,int j)
    {A=i,B=j;
    cout<<"C\n";
    }
    ~AAA(){cout<<"D\n";}
void print( )           印刷错误改为 void print(); };
{
    void AAA::print( )
    {cout<<A<<" "<<B<<"\n";}
void main()
{
    AAA *a1,*a2;
    a1=new AAA(1,2)
    a2=new AAA(5,6)
    a1->print();
    a2->print();
    delete a1;
    delete a2;
}
```

C

C

1,2

5,6
D
D

第五章习题

- 1, 当初始化 `const` 成员和引用成员时, 必须在构造函数的 初始化列表 进行初始化
- 2, 类 A 的对象做类 B 的数据成员, 定义类 B 的对象时, 先调用 A 类 构造函数, 再调用 B 类 构造函数
- 3, 定义静态成员使用的关键字是 static
- 4, 静态数据成员初始化的语句格式: 类型 类名:: 成员名=表达式;
- 5, 静态成员可以直接通过 类名:: 成员名 形式使用 公有 静态成员, 静态成员是 类 的成员, 不是 对象 成员, 没有建立对象前, 静态 成员就已经存在, 静态成员函数不能直接访问 非静态 成员, 静态成员函数不能说明为 虚 函数, 静态对象的生存周期是 从定义开始一直到程序结束, 静态对象在程序运行过程中调用 1 次构造函数, 1 次析构函数
- 6, 类的友元分为: 普通函数 做友元, 类的成员函数 做友元, 类 做友元, 类的友元中可以使用类的所有成员, 但需通过 对象, 指针 和 引用 来使用类的成员。声明友元函数需使用关键字 friend, 友元需在 类体 中声明, 友元关系是 单向的, 即当说明 A 是 B 的友元, B 是 C 的友元, A 却不是 C 的友元, 若 A 是 B 的友元, B 不一定是 A 的友元。
- 7, 静态 `Const` 整型数据成员 X 初始化语句: const 类型 类名:: 成员名=表达式;, `Const` 对象可以调用 const 成员函数, 不能调用 非 const 成员函数, `const` 成员函数中 不能 调用非 `const` 成员函数。定义 `const` 函数时, `const` 写在 函数首部 和 函数体 之间。
- 8, 改错题, 找出程序中的错误, 并说明错误原因。

```
#include <iostream>
using namespace std;
class base{    int x;    int a;
public:    base(int i,int j):x(i),a(j) {cout<<"构造"<<x<<","<<a<<endl;};
        void show()    {cout<<x++<<","<<a++<<endl;}
        ~base(){cout<<"析构"<<x<<","<<a<<endl;}
};
void main()
{    const base a1(104,118),a2(119,140);
    a1.show(); 将 show 函数的定义修改为 const 函数
    a2.show();    }
```

9, 请写出程序运行结果

```
#include <iostream>
using namespace std;
class base{    int x;    int a;
public:    base(int i,int j):x(i),a(j) {cout<<"构造"<<x<<","<<a<<endl;};
        void show()    {cout<<x++<<","<<a++<<endl;}
        void show() const {cout<<x<<","<<a<<endl;}
        ~base(){cout<<"析构"<<x<<","<<a<<endl;}
};
```

```
void main()
{
    const base a1(104,118),a2(119,140);
    a1.show();
    a2.show();
}
```

```
构造104,118
构造119,140
104,118
119,140
析构119,140
析构104,118
```

10，请写出程序运行结果

```
#include <iostream>
using namespace std;
class test{
    int num; double f;
public:
    test(int a){num=a;}
    test(int a,double b){num=a;f=b;}
    int getnum(){return num;}
    double getf(){return f;}
};
void main()
{
    test one[2]={test(2),test(4)};
    test two[2]={test(1,3.2),test(5,9.5)};
    for(int i=0;i<2;i++)    cout<<"one["<<i<<"]="<<one[i].getnum()<<endl;
    for(i=0;i<2;i++)    cout<<"two["<<i<<"]="<<two[i].getnum()<<","<< two[i].getf()<<endl;
}
```

```
one[0]=2
one[1]=4
two[0]=1,3.2
two[1]=5,9.5
```

11，若 main 函数修改为如下形式，请写出程序输出结果

```
void main()
{
    test two[2]={test(1,3.2),test(5,9.5)};    test *p;    p=two;
    for(int i=0;i<2;i++,p++)
        cout<<"two["<<i<<"]="<<p->getnum()<<","<< p->getf()<<endl;
}
```

```
two[0]=1,3.2
two[1]=5,9.5
```

12，请改正 main 函数中的错误（test 类仍然使用上题中的定义），改正后给出运行结果

```
void main()
{
    test *one[2]={new test(2),new test(4)};
    test *two[2]={new test(1,3.2),new test(5,9.5)};
    for(int i=0;i<2;i++)
```

```

        cout<<"two["<<i<<"]="<<two.getnum()<<","    // two[i]->getnum()
        << two.getf()<<endl;    // two[i]->getf()
    }
two[0]=1,3.2
two[1]=5,9.5
void main()
{
    test *one[2]={new test(2),new test(4)};
    test *two[2]={new test(1,3.2),new test(5,9.5)};
    for(int i=0;i<2;i++)
        cout<<"two["<<i<<"]="<<two[i].getnum()<<","    // two[i]->getnum()
        << two[i].getf()<<endl;    // two[i]->getf()
}
two[0]=1,3.2
two[1]=5,9.5

```

13, 请写出程序运行结果

```

#include <iostream>
using namespace std;
class test{    int val;
public:    test(int a){val=a;}
        int value(int a){return val+a;}
};
void main()
{
    int(test::*p)(int);    p=test::value;
    test obj(10);    cout<<obj.value(15)<<endl;    cout<<(obj.*p)(15)<<endl;
    test *p1;    p1=&obj;    cout<<p1->value(15)<<endl;    cout<<(p1->*p)(15)<<endl;
}

```

25
25
25
25

14, 改正程序中的错误

```

#include <cmath>
#include <iostream>
using namespace std;
class test{int x,y;    const double PI;    int &r;
public:    test(int a,int b) {PI=3.14;r=x;x=a;y=b;}
        void
        {cout<<"x="<<x<<endl<<"y="<<y<<endl<<"PI="<<PI<<endl<<"r="<<r<<endl;}
};

```

show()

```
void main(){ test t(1,2);    t.show();    test t1;    t1.show(); }
//修改构造函数为: test(int a=0,int b=0):PI(3.14),r(x) {x=a;y=b;}
```

```
x=1
y=2
PI=3.14
r=1
x=0
y=0
PI=3.14
r=0
```

15, 改正程序中的错误, 改正后给出程序输出结果。

```
#include <cmath>
#include <iostream>
using namespace std;
class test{    int x,y;    static int count;
public:    test(int a=0,int b=0){x=a;y=b;count++;cout<<"构造"<<x<<","<<y<<endl;}
        void show(){cout<<"当前共有"<<count<<"个对象"<<endl;} //前面加上 static
        ~test(){count--;cout<<"析构"<<x<<","<<y<<endl;}
};
count=0;//修改为: int test::count=0;
void main(){
    test::show();    test t(1,2);    test tx[2];
    for(int i=0;i<2;i++)
    {    static test ty;
        test tz;
        ty.show();    }
    test::show();
    t.show();
}
```



```

当前共有0个对象
构造1.2
构造0.0
构造0.0
构造0.0
构造0.0
当前共有5个对象
析构0.0
构造0.0
当前共有5个对象
析构0.0
当前共有4个对象
当前共有4个对象
析构0.0
析构0.0
析构1.2
析构0.0

```

16, 改错

```

#include <iostream>
#include <cmath>
using namespace std;
class point{    double x,y;
public:    point(double xi,double yi){x=xi; y=yi; }
        friend double dist(point & p1, point & p2);
};
double dist(point &p1,point &p2)
{    double dx=p1.x-p2.x;    double dy=p1.y-p2.y;    return sqrt(dx*dx+dy*dy);}
void main(){
    point p1(1.1,2.2),p2(3.3,4.4);
    cout<<"distance p1 to p2 : " << p.dist(p1,p2)<<endl;    //dist 是友元函数 dist(p1,p2)
}

```

17, 改错

```

#include <iostream>
#include <cmath>
using namespace std;
//此处加上对 classTwo 的声明 class classTwo;
class classOne
{    int x;
public:    classOne(int a ){x=a;}
        int getx(){return x;}
        void fun( classTwo & t);
};

```

```

class classTwo
{
    int y;
public:
    classTwo(int b ){y=b;}
    int gety(){return y;}
    void classOne::fun( classTwo & t); //友元函数声明:   friend void classOne::fun( classTwo & t);
};

void classOne::fun( classTwo & t){ t.y=x;   }

void main()
{
    classOne obj1(1);  classTwo obj2(2);
    cout<< obj1.getx()<<endl;   cout<< obj2.gety()<<endl;
    fun(obj2);   cout<<"调用 fun 函数之后: "<<endl;   //成员函数作友元调用: obj1.fun(obj2);
    cout<< obj1.getx()<<endl;   cout<< obj2.gety()<<endl;
}

```

```

1
2
调用fun函数之后:
1
1

```

18, 改错

```

#include <iostream>
using namespace std;

class base{
    int x=0;//不能初始化   const int a;   static const int b;   const int &r;
public:
    base(int i,int j):x(i),r(x){a=j;}; //a 必须在初始化列表中初始化
    void show(){cout<<x<<" "<<a<<" "<<b<<" "<<r<<endl;}//show 应该是 const 函数
    void f() const   {r=r+1;}//f 函数体内不能有修改语句
};

const int base::b=25;

void main()
{
    const double x=10;   base a1(104,118); base const a2(119,140);
    a1.show();   a2.show();   a2.f(x);//f 函数是无参函数
}

```

19, 写出程序运行结果

```

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

class dog{
    static int dogs;   string name;
public:
    dog(string a)   {name=a; dogs++;}
    static int getdogs(){return dogs;}
    ~dog(){dogs--;}
};

```

```

int dog::dogs=0;
void main(){
    dog d1("Tom"),d2("Jerry");  cout<<"现在有"<<dog::getdogs()<<"只 dog"<<endl;
    dog *p;  p=new dog("snoopy");  delete p;
    cout<<"现在有"<<dog::getdogs()<<"只 dog"<<endl;
}

```

现在有2只dog
 现在有2只dog

20，写出程序运行结果

```

#include <iostream>
using namespace std;
class complex{    double real;    double image;
public:    complex(double a,double b) {real=a; image=b;cout<<"构造"<<real<<","<<image<<endl;}
        complex(complex &t)    {real=t.real; image=t.image;cout<<"复制构造"<<endl;}
        complex(){cout<<"无参构造"<<endl;}
        friend complex add(complex a,complex b);
        void show(){cout<<real<<"+ "<<image<<"i"<<endl;}
};
complex add(complex a,complex b){
    complex temp(a.real +b.real,a.image +b.image);
    return temp;
}
void main(){
    complex a(1,2),b(3,4);    complex c;    c=add(a,b);    c.show ();    }

```

构造1,2
 构造3,4
 无参构造
 复制构造
 复制构造
 构造4,6
 复制构造
 4+6i

第六章习题

- 1, 在一个已有类的基础上定义产生一个新的类的过程称为派生, 已有的类称为基类, 新的类称为派生类, 类的继承是指派生类继承了基类的数据成员和成员函数, 继承常用来表示关系, 当从现有类中派生出新类时, 派生类有如下三种变化: 增加了新成员, 重定义了已有成员函数, 改变了基类成员的访问权限。
- 2, C++中有两种继承: 单一继承和多重继承。
- 3, 派生类对象产生时, 先调用基类的构造函数, 然后调用派生类构造函数。派生类的对象析构时, 先调用派生类析构函数, 在调用基类析构函数。
派生类继承了基类的所有成员, 但构造函数和析构函数不能被继承。
- 4, 类中的保护成员具有私有成员和公有成员的双重角色: 对于派生类成员函数来说它是公有成员, 可以被访问, 而对于其他函数而言它是私有成员, 不能被访问。
- 5, 公有派生类中不可以(可以/不可以)访问基类的私有成员, 公有派生类中可以(可以/不可以)访问基类的公有成员, 公有派生类中可以(可以/不可以)访问基类的保护成员。
- 6, 基类的私有成员在公有派生类中变为不可访问成员, 基类的公有成员在公有派生类中变为公有成员, 基类的保护成员在公有派生类中变为保护成员
- 7, 赋值兼容规则: 派生类对象可以赋值给基类对象, 基类指针可以指向派生类对象, 派生类对象可以初始化基类引用。
- 8, 当基类指针 p 指向了派生类对象 a 后, p 所调用的成员函数是基类的成员函数。当基类的引用 r 引用了派生类对象 a 后, r 所调用的成员函数是基类的成员函数。
- 9, 基类的私有成员在私有派生类中变为不可访问成员, 基类的公有成员在私有派生类中变为私有成员, 基类的保护成员在私有派生类中变为私有成员
- 10, 基类的私有成员在保护派生类中变为不可访问成员, 基类的公有成员在保护派生类中变为保护成员, 基类的保护成员在保护派生类中变为私有成员
- 11, 如果使用一个表达式的含义能解释为可以访问多个基类中的成员, 则这种对基类成员的访问是不确定的, 称这种访问具有二义性。
- 12, 当派生类中从多个基类中继承得到同名函数时, 在派生类中使用这些函数时, 须使用成员名限定, 否则会出现二义性, 在类体外使用这些函数时, 也需要进行类名限定。
- 13, 派生类中定义了函数 f, 同时可能从基类中继承了同名函数 f, 此时派生类的对象能正确调用函数 f, 是由 C++的作用域规则保证的。

14, 写出程序运行结果

```
#include <iostream>
#include <string>
using namespace std;
class human{
protected: string name;  int age;  string sex;
public:  human(string a,int b,string c) {name=a;  age=b;  sex=c;  cout<<" 人  "<<name<<"  诞生了 "
"<<endl;}
    ~human(){ cout<<" 人  "<<name<<"  over 了 " <<endl; }
```

```

        void set(string a,int b,string c)    {name=a; age=b; sex=c;}
        void show()
        { cout<<"my name is "<<name<<endl <<"my age is "<<age<<endl<<"my sex is
"<<sex<<endl; }
};
class student :public human{
    int num; string myclass;
public:    student(string a,int b,string c,int d,string e):human(a,b,c)
        {num=d;myclass=e;cout<<"学生 "<<name<<" 诞生了"<<endl;}
    ~student(){cout<<"学生 "<<name<<" over "<<endl;}
    void show()
    { cout<<"my name is "<<name<<endl<<"my age is "<<age<<endl
        <<"my sex is "<<sex<<endl<<"my num is "<<num<<endl
        <<"my class is "<<myclass<<endl;}
};
void main()
{    student s1("李斯",20,"女",1001,"07 信 1"); s1.show();
    human h1("张三",20,"男"); h1.show();    }

```

```

人 李斯 诞生了
学生 李斯 诞生了
my name is 李斯
my age is 20
my sex is 女
my num is 1001
my class is 07信1
人 张三 诞生了
my name is 张三
my age is 20
my sex is 男
人 张三 over 了
学生 李斯 over
人 李斯 over 了

```

15, 改错

```

#include <iostream>
#include <string>
using namespace std;
class base{    int x1;
protected:    int y1;
public:    base(int a,int b){x1=a; y1=b;}
        void fun1(){cout<<x1<<" "<<y1<<endl;}
};

```

```

class derived :protected base{ int x2;
protected:    int y2;
public:       derived(int a,int b,int c,int d):base(a,b) {x2=c; y2=d;}
              void fun2(){cout<<x1<<" "<<y1<<endl<<x2<<" "<<y2<<endl;} //x1 不可访问
};
void main()
{   base b(1,2);   derived d(3,4,5,6);
    cout<<b.x1<<endl;    cout<<b.y1<<endl;    //x1, y1 不可访问
    b.fun1();
    cout<<d.x1<<endl; cout<<d.y1<<endl; cout<<d.y2<<endl; cout<<d.x2<<endl;//不可访问
    d.fun1(); //不可访问    d.fun2();
}

```

16, 改正程序中的错误

```

#include <iostream>
#include <string>
using namespace std;
class base{
private:   int x1;
protected:int y1;
public:    base(int a,int b){x1=a; y1=b;}
          void fun1(){cout<<x1<<" "<<y1<<endl;}
};
class derived :private base{
private:   int x2;
protected:int y2;
public:    derived(int a,int b,int c,int d):base(a,b) {x2=c; y2=d;}
          void fun2(){cout<<x1<<" "<<y1<<endl<<x2<<" "<<y2<<endl;} x1 不可访问
};
void main()
{   base b(1,2);   derived d(3,4,5,6);   b.fun1 ();    d.fun1 ();不可访问    d.fun2 ();}

```

17, 改正程序中的错误, 然后写成程序运行结果

```

#include <iostream>
#include <string>
using namespace std;
class A{ int x1;
protected:int y1;
public:    A(int a,int b) {x1=a; y1=b; cout<<"构造 A"<<endl;}
          void fun(){cout<<x1<<" "<<y1<<endl;}
};

```

```

class B { int x2;
protected: int y2;
public:    B(int a,int b)  {x2=a; y2=b; cout<<"构造 B"<<endl;}
        void fun(){cout<<x2<<" "<<y2<<endl;}
};
class C:public A,public B{ int x3;
protected: int y3;
public:    C(int a, int b ,int c ,int d ,int e ,int f):A(a,b),B(c,d){x3=e; y3=f; cout<<"构造 C"<<endl;}
        void Cfun(){A::fun(); B::fun();cout<<x3<<" "<<y3<<endl;}
};
void main()
{    C c1(1,2,3,4,5,6);
    c1.Cfun();
    c1.fun();//调用继承自 A 的 fun 函数 c1.A:: fun();
    c1.fun();//调用继承自 B 的 fun 函数 c1.B::fun();
}

```

```

构造A
构造B
构造C
1 2
3 4
5 6
1 2
3 4

```

第七章习题

1, 请填全下面类模板的定义:

```
template <class T>  
class TAnyTemp{  
    T x,y;  
public:  
    TAnyTemp(T a, T b):x(a),y(b){ }  
    T getx(){ return x;}  
    T gety();  
};  
template <class T> TAnyTemp <T> :: gety(){ return y; }
```

2, 请利用上面定义的类模板定义一个模板类的对象 a, 指定数据类型为 int, TAnyTemp<int> a;

3, 下面是派生类模板的定义, 请补充完整

```
#include <iostream>  
using namespace std;  
class point{    int x,y;  
public:    point(int a,int b){x=a; y=b;}  
          void display(){cout<<x<<" "<<y<<endl;}  
};  
template <typename T>  
class line:public point{  
    T x2,y2;  
public:  
    line(int a,int b,T c,T d): point(a,b) {x2=c; y2=d; }  
    void display() {point::display(); cout<<x2<<" "<<y2<<endl;}  
};
```

4, 下面是派生类模板的定义, 请补充完整

```
#include <iostream>  
using namespace std;  
template <class T>  
class point{    T x,y;  
public:    point(T a,T b){x=a; y=b;}  
          void display();  
};  
template <class T>  
void point<T> ::display(){cout<<x<<" "<<y<<endl;}  
template <typename T>  
class line:public point<T> {
```



```

T x2,y2;
public: line(int a,int b,T c,T d): point<T>(a,b) {x2=c; y2=d; }
    void display() {point<T>::display(); cout<<x2<<" "<<y2<<endl;}
};

```

```

void main()
{
    point <int> a(3,8);
    Line <int> b(4,5,6,7);
}

```

5, C++中要使用向量, 须包含头文件 vector.h , 定义向量有 5 种形式, 请按要求给出向量的定义语句。(1) 定义一个存放 10 个 int 数据的向量 v。 vector<int> v(10);

(2) 定义一个存放 10 个字母'a'的向量 v。 vector<char> v(10,'a');

(3) 定义一个同上一个向量完全相同的向量 v2。 vector<char> v2(v);

(4) 假设有数组 int a[10]={1}; 定义一个向量 v 使得其数据与 a 数组相同。 vector<int>v(a,a+10);

(5) 定义一个整型向量 v 不给出长度和数据。 vector<int> v;

6, 假设有向量 v, 则 v.begin()表示 向量中第一个元素的地址, v.end()表示 向量中最后一个元素之后的地址, v.rbegin()表示 逆向指针, 表示最后一个元素地址, v.rend()表示 逆向指针, 表示第一个元素之前的地址。

利用 begin()函数和循环语句正向输出向量 v 中的数据

for(vector<int>::iterator p=v.begin(); p<v.end(); p++) cout<<*p;

利用 rbegin()函数和循环语句正向输出向量 v 中的数据

for(vector<int>::reverse_iterator p=v.rbegin(); p<v.rend(); p++) cout<<*p;

利用 rend()函数和循环语句正向输出向量 v 中的数据

for(vector<int>::reverse_iterator p=v.rend()-1; p>=v.begin(); p--) cout<<*p;

利用 end()函数和循环语句正向输出向量 v 中的数据

for(vector<int>::iterator p=v.end()-1; p>=v.begin(); p--) cout<<*p;

7, vector<int> v(10,0);请定义一个正向泛型指针 p 指向向量 v,

vector<int>::iterator p=v.begin();

定义一个逆向泛型指针 p 指向向量 v, vector<int>::reverse_iterator p=v.rbegin();

8, int y[10]={0,1,2,3,4,5,6,7,8,9}; vector<int> v1(y,y+10);

请写出语句, 利用泛型算法正向输出向量 v1 中的数据, 数据间用空格分隔

copy(v1.begin(),v1.end(),ostream_iterator<int>(cout," "));

请写出语句, 利用泛型算法逆向输出向量 v1 中的数据, 数据间用空格分隔

reverse_copy(v1.begin(),v1.end(),ostream_iterator<int>(cout," "))

请写出语句, 利用泛型算法对向量 v1 中的数据进行降序排序

sort(v1.begin(),v1.end(),greater<int>());

9, 请写出程序运行结果

```
#include <iostream>
```

```
#include <vector>
```

```

#include <functional>
#include <algorithm>
#include <complex>
#include <string>
using namespace std;
class human{
    string name;  int age;  string sex;
public:  human(string a,int b,string c) {name=a;  age=b;  sex=c;  cout<<" 人  "<<name<<"  诞生了
"<<endl;}
    ~human(){ cout<<" 人  "<<name<<"  over  了  "<<endl; }
    void set(string a,int b,string c)      {name=a; age=b; sex=c;}

    void show(){ cout<<"my name is "<<name<<endl <<"my age is "<<age<<endl
                <<"my sex is "<<sex<<endl; }
};
void main()
{   int i;
    complex<int> x[3]={complex<int>(1,2),complex<int>(3,4)};
    vector<int>   a(3,  2 );
    vector<int *> b(3);
    vector<human>  c(3, human("张三",20,"男") );
    vector<human*> d(3 );
    vector< complex<int>*> e(3);
    cout<<"向量 c 中有:  "<<c.size()<<"个对象"<<endl;
    cout<<"利用数组形式输出向量元素(正向)"<<endl;
    for( i=0;i<=2;i++)      c[i].show();
    cout<<endl;
    e[0]=&x[0];    e[1]=&x[1];    e[2]=&x[2];
    cout<<"向量 e 中有:  "<<c.size()<<"个对象"<<endl;
    cout<<"利用数组形式输出向量元素(正向)"<<endl;
    for( i=0;i<=2;i++)      cout<<e[i]->real()<<" "<<e[i]->imag() <<endl;
    cout<<endl;
    int y[10]={0,1,2,3,4,5,6,7,8,9};
    vector<int> v1(y,y+10);
    copy(v1.begin(),v1.end(),ostream_iterator<int>(cout," "));
    cout<<endl;
    reverse_copy(v1.begin(),v1.end(),ostream_iterator<int>(cout," "));
    cout<<endl;
    sort(v1.begin(),v1.end(),greater<int>());
}

```

```

copy(v1.begin(),v1.end(),ostream_iterator<int>(cout," "));
cout<<endl;
cout<<"v2 向量: "<<endl;
vector <int> v2(10);
copy(v1.begin(),v1.end(),v2.begin());
copy(v2.begin(),v2.end(),ostream_iterator<int>(cout," "));
cout<<endl;
reverse_copy(v1.begin(),v1.end(),v2.begin());
copy(v2.begin(),v2.end(),ostream_iterator<int>(cout," "));
cout<<endl;
if(*find(v1.begin(),v1.end(),8)!=8)
    cout<<"向量中不存在相关数据"<<endl;
else
    cout<<"向量中存在相关数据"<<endl;
}

```

```

my sex is 男
my name is 张三
my age is 20
my sex is 男
my name is 张三
my age is 20
my sex is 男

```

向量e中有：3个对象
利用数组形式输出向量元素<正向>

```

1,2
3,4
0,0

0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
v2向量:
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
向量中存在相关数据
人 张三 over 了
人 张三 over 了
人 张三 over 了

```

10、请写出向量对应的成员函数

- (1) 返回当前向量中已存放对象的个数 size()
- (2) 返回向量可容纳最多对象的个数 max_size()
- (3) 返回无需再次分配内存就能容纳的对象个数 capacity()
- (4) 判断当前向量是否为空 empty()

- (5) 在向量尾部插入一个对象 push_back()
- (6) 在向量指定位置 it 插入一个对象 insert(iterator it, const T &)
- (7) 在向量指定位置 it 插入 n 个对象 insert(iterator it, n, const T &)
- (8) 删除向量中最后一个对象 pop_back()
- (9) 删除向量中 it 位置的向量 erase(iterator it)
- (10) 删除向量中所有对象 clear()

11, 找出下面类模板定义中的错误, 并改正

```
template<class Type>
class TAnyTemp{
    T x,y;    改错: T→ Type
public:
    TAnyTemp(T a, T b):x(a),y(b){ }    改错: T→ Type
    int getx(){ return x;}    改错: int → Type
    int gety(){ return y;}    改错: int → Type
};
```

12, 找出下面类模板定义中的错误, 并改正

```
#include <iostream>
using namespace std;
template<class T>
class max4{    T a,b,c,d;
    max(T x1, T x2) {return (x1>x2)?x1:x2;}
public:
    max4(T, T ,T ,T );
    T max(void);
};
max4:: max4(T x1, T x2, T x3, T x4):a(x1),b(x2),c(x3),d(x4){ }
max4:: max(void){return max(max(a,b),max(c,d));}
函数定义前加上: template<class T>
```

第八章习题

- 1, 多态性包含编译时的多态性和运行时多态性两大类
- 2, 通过指向对象的指针和对象的引用调用成员函数时: 所调用成员函数为指针或对象所属类的成员函数
- 3, 存在继承和派生时, 如果派生类和基类中均定义了同名函数, 则通过指向派生类对象的基类指针或通过引用了派生类对象的基类引用调用成员函数时, 系统会调用派生类的成员函数, 若派生类中未定义同名函数时, 调用基类的成员函数(前提是该函数为虚函数)
- 4, 定义成员函数时, 函数首部前加关键字virtual则定义的成员函数为虚函数
若基类的成员函数 f 定义为虚函数, 则派生类中与基类的同名的成员函数 f (函数类型、参数也相同) 自动为虚函数
- 5, 虚函数的调用规则: 根据当前对象, 优先调用对象本身的虚成员函数
- 6, 若将基类成员函数 f 定义为虚函数, 并且基类指针指向了派生类对象, 则通过基类指针调用 f 函数时, 若派生类中未定义同名函数 f , 则基类指针调用的 f 函数是基类中的 f 函数,
若派生类中定义了同名函数 f , 则基类指针调用的 f 函数是派生类中的 f 函数。
- 7, 虚函数是类的成员函数, 且不能是静态成员函数, 构造函数不能定义为虚函数, 析构函数可以定义为虚函数
- 8, 假设基类 A 中定义了虚函数 $f()$, 在派生类 B 中重定义了函数 $f()$, 有 B 类的对象 b 调用函数 $f()$ 时, 调用的是派生类的函数 $f()$, 或者使用 $b.A::f();$ 此时调用的 $f()$ 为基类的 $f()$ 。
- 9, 产生运行时多态的条件(动态联编): 1, 类之间的继承关系满足赋值兼容规则。2, 重定义了同名虚函数。3, 根据赋值兼容规则使用指针或引用
- 10, 利用指向派生类的基类指针调用虚函数时调用的是派生类的成员函数, 利用引用了派生类对象的基类引用调用虚函数时调用的是派生类的成员函数。用基类引用(基类指针)作函数形参参数时, 函数调用时用派生类对象(或派生类对象的地址)做实参时, 在函数体内所调用的虚成员函数是派生类的成员函数。(此处假定基类和派生类中都定义了该函数)
- 11, 在构造函数和析构函数中调用虚函数时采用静态联编, 即: 当前对象所调用虚函数是本身或基类中定义的函数, 不是派生类的函数。而且优先调用自身的虚函数。
- 12, 若已知 A 派生 B , B 派生 C 。若 C 类对象调用的成员函数为继承自 B 类的函数 $f1()$, 且 $f1()$ 函数体内又有成员函数 $f2()$ 调用时, 若该函数 $f2()$ 是虚函数且在 C 类中有定义时, 优先调用 C 类的 $f2()$ 函数
- 13, 若已知 A 派生 B , B 派生 C 。若 C 类对象调用的成员函数为继承自 B 类的函数 $f1()$, 且 $f1()$ 函数体内又有成员函数 $f2()$ 调用时, 若该函数 $f2()$ 不是虚函数, 则优先调用 C 类的 $f2()$, 若 C 类中没有 $f2()$ 定义, 则调用 B 类的 $f2()$ 。
- 14, 请给出程序运行结果

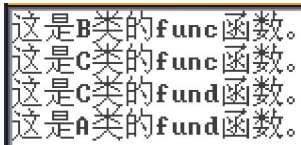
```
#include <iostream>
using namespace std;
class A{
public:  A(){}

```

```

        virtual void func(){cout<<"这是 A 类的 func 函数。"<<endl;}
        ~A(){}
        virtual void fund(){cout<<"这是 A 类的 fund 函数。"<<endl;}
};
class B:public A{
public:    B(){func();}
        void func(){cout<<"这是 B 类的 func 函数。"<<endl;}
        void fun(){cout<<"这是 B 类的 fun 函数。";func();}
        ~B(){fund();}
};
class C:public B{
public:    C(){}
        void func(){cout<<"这是 C 类的 func 函数。"<<endl;}
        ~C(){fund();}
        void fund(){cout<<"这是 C 类的 fund 函数。"<<endl;}
};
void main()
{
    C c;
    c.func();
}

```



这是B类的func函数。
 这是C类的func函数。
 这是C类的fund函数。
 这是A类的fund函数。

15, 请给出程序运行结果

```

#include <iostream>
using namespace std;
class A{
public:    A(){}
        virtual void func(){cout<<"这是 A 类的 func 函数。"<<endl;}
        ~A(){}
        virtual void fund(){cout<<"这是 A 类的 fund 函数。"<<endl;}
};
class B:public A{
public:    B(){func();}
        void fun(){cout<<"这是 B 类的 fun 函数。";func();}
        ~B(){fund();}
};
class C:public B{

```

```

public:  C(){
        void func(){cout<<"这是 C 类的 func 函数。"<<endl;}
        ~C(){fund();}
        void fund(){cout<<"这是 C 类的 fund 函数。"<<endl;}
};
void main()
{   C c;
    c.func();
}

```

```

这是A类的func函数。
这是C类的func函数。
这是C类的fund函数。
这是A类的fund函数。

```

16, 将上例中的 main 函数修改为如下形式, 请给出程序运行结果

```

void main()
{   C c;
    c.fund();
}

```

```

这是A类的func函数。
这是C类的fund函数。
这是C类的fund函数。
这是A类的fund函数。

```

17, 在构造函数中调用虚函数采用 静态连编, 即他们所调用的虚函数是自己类或基类中定义的虚函数, 但不是任何派生类中重新定义的虚函数。

18, 目前 C++ 标准不支持虚构造函数, 但可以定义析构为虚函数, 若基类的析构函数定义虚函数后, 其 派生类 类的析构函数自动为虚函数。

19, 请写出纯虚函数的定义形式:

```

class 类名 {
    virtual 函数类型 函数名 (参数列表) =0 _____;
}

```

20, 一个类可以定义多个纯虚函数, 具有纯虚函数的类称为 抽象类, 这样的类只能作为基类来派生新的类, 不能 定义抽象类的对象。从一个抽象类派生的类必须提供 纯虚函数 的实现代码, 或在该派生类中仍将他定义为纯虚函数。这些未给出纯虚函数实现代码的类仍称为 抽象类, 若某个派生类中给出了纯虚函数的代码则该派生类不再是 抽象类 类。

21, 由同一个基类派生出一系列的派生类, 则将这些类总称为 类族。

22, void (A::*pf) (void); 表示定义一个函数指针, 请说明该指针 pf 能指向什么样的函数
指向 A 类的无参无返回值的函数

23, 请改正下面程序中的错误。两种方式, 一种通过修改类体不修改 main 函数, 一种通过修改 main 函数不修改类体。修改后请写出程序运行结果。

```

#include <iostream>
#include <cmath>
using namespace std;
class A{
public: virtual void f(){cout<<"A 的函数 f"<<endl;}
};
class B{
public: virtual void f(){cout<<"B 的函数 f"<<endl;}
};
class C:public A,public B{    };
void main()
{   A *pa;   B *pb;   C *pc,c;
    pa=&c;   pb=&c;   pc=&c;
    pa->f();   pb->f(); pc->f();
}

```

将类 C 定义为: class C:public A,public B{ public:void f(){cout<<"C 的函数 f"<<endl;} };

C的函数f
C的函数f
C的函数f

将 main 函数修改为: pa->A::f(); pb->B::f(); pc->A::f();

A的函数f
B的函数f
A的函数f

24, 请给出程序运行结果。

```

#include <iostream>
#include <cmath>
using namespace std;
class A{
public: virtual void f1(){cout<<"A 的函数 f1"<<endl;}
        virtual void f2(){cout<<"A 的函数 f2"<<endl;}
        virtual void f3(){cout<<"A 的函数 f3"<<endl;}
};
class B:public A{
public: void f1(){cout<<"B 的函数 f1"<<endl;}
        void f2(){cout<<"B 的函数 f2"<<endl;}
        void f3(){cout<<"B 的函数 f3"<<endl;}
};
void main()
{   A *p1;      A  a;      B  b;

```



```

void (A::*pf1)(void);
pf1=A::f1;
p1=&a;      (p1—>*pf1)();
p1=&b;      (p1—>*pf1)();
}

```

A的函数f1
B的函数f1

25, 请给出程序运行结果

```

class base{    int x;
public:   base(int a){x=a;cout<<"base..."<<x<<endl;}
        base(base &t){x=t.x;cout<<"base copy.."<<endl;}
        ~base(){cout<<"~base"<<x<<endl;}
};

class derived:public base{    int y;
public:   derived(int a,int b):base(a){y=b;cout<<"derived..."<<y<<endl;}
        derived(derived &t):base(t){y=t.y;cout<<"derived copy..."<<y<<endl;}
        ~derived(){cout<<"~derived..."<<y<<endl;}
};

void main()
{
    base *p1=new derived(52,54);
    base a(*p1);
    delete p1;
    derived *p2=new derived(52,54);
    derived d(*p2);
    delete p2;
}

```

```

base...52
derived...54
base copy..
~base52
base...52
derived...54
base copy..
derived copy...54
~derived...54
~base52
~derived...54
~base52
~base52

```

第九章习题 运算符重载及流类库

1. 下列 (BD) 运算符不能重载.
A. + B. * C. → D. ::
2. 友元函数不能重载的运算符为 (ABD) .
A. = B. → C. * D. []
3. 下列有关运算符重载的叙述中, 正确的是(A)
A) 运算符重载是多态性的一种表现
B) C++中可以通过运算符重载创造新的运算符
C) C++中所有运算符都可以作为非成员函数重载
D) 重载运算符时可以改变基结合性
4. 若已知 `string str;`, 有语句 `cin>>str;` 当输入为:
This is a C++program
时, `str` 所得结果是 (B)
A. This is a C++program B. This
C. This is D. This is a C
5. 已知在一个类体中包含如下函数原型: `VOLUME operator-(VOLUME)const;`, 下列关于这个函数的叙述中, 错误的是(B)
A) 这是运算符-的重载运算符函数
B) 这个函数所重载的运算符是一个一元运算符
C) 这是一个成员函数
D) 这个函数不改变类的任何数据成员的值
6. 在表达式 `x+y*z` 中, `+` 是作为成员函数重载的运算符, `*` 是作为非成员函数重载的运算符。
下列叙述中正确的是(C)
A) `operator+` 有两个参数, `operator*` 有两个参数
B) `operator+` 有两个参数, `operator*` 有一个参数
C) `operator+` 有一个参数, `operator*` 有两个参数
D) `operator+` 有一个参数, `operator*` 有一个参数
7. 下面关于 C++流的叙述中, 正确的是(ACD)
A) `cin` 是一个输入流对象
B) 可以用 `ifstream` 定义一个输出流对象
C) 执行语句序列 `char *y="PQMN"; cout<<y;` 将输出字符串 PQMN
D) 执行语句序列 `char x[80]; cin>>x;` 时, 若键入
Happy new year

则 x 中的字符串是"Happy"

8. 下列有关 C++流的叙述中, 错误的是(A)

A) C++操作符 setw 设置的输出宽度永久有效

B) C++操作符 endl 可以实现输出的回车换行

C) 处理文件 I/O 时, 要包含头文件 fstream

D) 进行输入操作时, eof()函数用于检测是否到达文件尾

※9. 在 C++中, 使用流进行输入输出, 其中用于屏幕输入的是(A)

A. cin B. cerr C. cout D. clog

※10. 下列说法中正确的是(B)

A. 所有的运算符都能被重载

B. 运算符被重载时, 它们的优先级和结合性不会改变

C. 当需要时, 我们可以自定义一个运算符进行重载

D. 每个运算符都可以被重载成成员函数和友元函数

※11. 执行语句序列 ofstream outf("SALARY.DAT"); if (...) cout<<"成功"; else cout<<"失败"; 后, 如文件打开成功, 显示"成功", 否则显示"失败", 由此可知, 上面 if 语句的条件表达式是(D)

A. ! outf 或者 outf.fail() B. ! outf 或者 outf.good()

C. outf 或者 outf.fail() D. outf 或者 outf.good()

※12.在 C++中要创建一个文件输入流对象 fin, 同时该对象打开文件"Test.txt"用于输入, 则正确的声明语句是 ifstream fin("Test.txt");

※13. C++语言中支持的两种多态性分别是编译时的多态性和 运行时 的多态性

※14.语句序列

ifstream infile;

infile.open("data.dat"); 的功能可用一个语句实现, 该语句是 ifstream infile("data.dat");

※15.下面的类定义中有一处错误, 请用下划线标出所在行并给出修改意见(5分)

#include<iostream.h>

class fract

{ private:

int den;

int num;

public:

fract(int d=0, int n=1): den(d), num(n){ }

friend fract &operator +=(fract &, fract &);

void show() {cout<<den<<'/'<<num; }

```

};

friend fract & operator +=(fract &f1, fract & f2) //函数定义时不需要再加表示友元的关键字 friend
{ f1.den=f1.den* f2.num+f1.num*f2.den;
  f1.num*=f2.num;
return f1;
}

void main( )
{ fract fr(3,4);
  fr+=fract(5,7);
fr.show( );
cout<<endl;
}

```

16. 编写一个复数类，要求用成员函数重载 " + "，友元函数重载 <<，已给出类的定义及主函数，请编写重载+运算符和<<运算符的函数实现（10 分）

```

#include<iostream.h>

class complex
{ private:
    int real;
    int imag;
public:
    complex( int r=0, int i=0){ real=r;  imag=i ;}
    complex  operator +( complex &b);
    friend  ostream & operator<< (ostream &output,  complex &a);
};

void main( )
{
complex  x (1, 1),  y (2, 3),  z;
z=x+y;
cout<<z<<endl;
}

```

备注：※标示的题目选自《高等教育自学考试考前密押试卷》

第九章习题

1, 运算符重载本质上是__函数__重载, 运算符重载两种方式: 类运算符重载, 友元运算符重载, 有些运算符不能进行重载: ., ::, *, ?:, sizeof # 不是运算符所以也不能重载。有些运算符只能重载为类运算符: ==, _(), [], ->。运算符重载时不能改变运算符原来的优先级, 不能自己定义新的运算符。==运算符外, 其他重载的运算符可继承。

2, 下面程序中要执行 A 类对象的加法运算, 需要对+运算符进行重载, 请将程序补充完整。

```
class A{ int x;
public:   A(int a=0){x=a;cout<<"构造函数调用: "<<x<<endl;}
        void show(){cout<<"x 值是: "<<x<<endl;}
        friend A operator +( A t1, A t2 )
        {   A t(t1.x+t2.x);    return t;}
};
void main()
{   A a1(1),a2(2),a3;
    a3=operator+(a1,a2); //此语句可写作: a3=a1+a2;
    a3.show();
}
```

3, 下面程序中要执行 A 类对象的加法运算, 需要对+运算符进行重载, 请将程序补充完整。

```
#include <iostream.h>
class A{ int x;
public:   A(int a=0){x=a;cout<<"构造函数调用: "<<x<<endl;}
        void show(){cout<<"x 值是: "<<x<<endl;}
        A operator _+( A t1 )
        {   A tx(x+t.x);    return tx;  }
};
void main()
{   A a1(1),a2(2),a3;
    a3=a1.operator+(a2); //此语句可写作: a3=a1+a2;
    a3.show();
}
```

3, 如下程序中实现 mystring 类对象的相互赋值时, 需要对=运算符进行重载。(重载为类运算符)

```
class mystring{ char *str;
public:   mystring(char *s) {   str=new char[strlen(s)+1];    strcpy(str,s);    }
        int size(){ return strlen(str); }
        void show(){ cout<<str<<endl; }
        ~mystring(){ delete str; }
        mystring &operator = (mystring s );
}
```

```
};
    mystring & mystring::operator= ( mystring s)
{
    if( &s==this ) return *this;
    delete str;
    str=new char[ strlen(s.str)+1 ];
    strcpy( str, s.str );
    return *this; }

void main()
{
    mystring s1("hello");    mystring s2("everyone");    mystring s3("hi");
    char *s4="study hard";
    s1=s3; //此句可以等价写作: s1.operator=(s3); (注意: 此处不能使 s4 赋值给 s1
    s1.show ();
    cout.operator << (s1.size());//请说明此语句含义 此处是利用 cout 输出 s1 对象中字符的个
数
}
```

4, 下面程序中要实现利用<<输出 student 类的对象, 并将<<运算符重载为友元运算符。

```
class student { int num;
                string name;
public:
    student(int a,string b)    { num=a; name=b; }
    friend ostream & operator<<( ostream & os , student &t
    {
        cout <<"学号: "<<t.num<<endl;
        cout <<"姓名: "<<t.name<<endl;
        return cout; }
};

void main(){
    student t1(1001,"张三");student t2(1002,"李四");
    operator<<(cout,t1);
    cout<<t1;    //请写出等价形式的语句: operator<<( cout,t1);
    operator<<(cout,t1).operator<<(endl);//请理解此语句含义: 输出 t1 然后换行
}
```

```
学号: 1001
姓名: 张三
学号: 1001
姓名: 张三
学号: 1001
姓名: 张三
```

4, 下面程序中要实现利用<<输出 student 类的对象, 并将<<运算符重载为类运算符。

```
class student { int num;
                string name;
public:
    student(int a,string b)    { num=a; name=b; }
```

```

        ostream & operator<<( ostream & os )
        {
            os <<"学号: "<<num<<endl;
            os <<"姓名: "<<name<<endl;
            return os; }
};

```

```

void main(){
    student t1(1001,"张三");
    student t2(1002,"李四");
    t1.operator<<(cout);
    t1.operator <<(t2.operator<<(cout));
}

```

```

学号: 1001
姓名: 张三
学号: 1002
姓名: 李四
学号: 1001
姓名: 张三

```

5, 自增++运算符重载实例。请写出程序运行结果。

```

class number{ int num;
public:
    number(int i){num=i;}
    int operator++() { num=num+1; return num; } 后置自加
    int operator++(__int) {int i=num; num=num+1; return i; } 前置自加
    void print(){ cout<<"数据成员 num 的值是: "<<num<<endl;}
};

```

```

void main()
{
    number n(10); int i;
    i=++n;//请写出此语句的等价调用形式: i=n.operator++();
    cout<<"i 值为: "<<i<<endl; n.print();
    i=n++;//请写出此语句的等价调用形式: i=n.operator++(0);
    cout<<"i 值为: "<<i<<endl; n.print();
}

```

```

i值为: 11
数据成员num的值是: 11
i值为: 11
数据成员num的值是: 12

```

6, 本题属于第八章内容。请写出程序运行结果

```

#include <iostream>
using namespace std;
class base{ int m;
public:
    base(int a){m=a;}
}

```

```

    int getm(){return m;}
    virtual void show(){cout<<getm()<<endl;}
};
class derived :public base{    int n;
public:    derived(int a,int b):base(a){n=b;}
        void show(){cout<<getm()<<" "<<n<<endl;}
};
void print(base *p) {p->show();}
void main(){    base A(58);    derived B(18,28);    print(&A);    print(&B);}
58
18,28

```

若将 **print** 函数和 **main** 函数修改为如下程序，请写出输出结果

```

void print(base p) {p.show();}
void main(){    base A(58);    derived B(18,28);    print(A);    print(B);}
58
18

```

若将 **print** 函数和 **main** 函数修改为如下程序，请写出输出结果

```

void print(base &p) {p.show();}
void main(){    base A(58);    derived B(18,28);    print(A);    print(B);}
58
18,28

```

7，补充完整下面程序，并写出程序运行结果。

```

#include <iostream.h>
class complex{ double real,imag;
public:    complex(double r=0,double i=0){ real=r; imag=i; }
        friend _____ complex operator +(____ complex a, ____ complex b ____ )
        { double r=a.real +b.real;    double i=a.imag+ b.imag; return complex(r,i);}
        void show(){ cout<<real<<"+"<<imag<<"i"<<endl; }
};
void main()
{    complex c1(5,3),c2;
    c2=c1+7;    //写出与此语句等价的调用形式:  c2=operator +(c1, 7);
    c2.show();
    c2=7+c1;    //写出与此语句等价的调用形式:  c2=operator +(7, c1);
    c2.show();
    12+3i
    //请写出程序运行结果:  12+3i
}

```

7，补充完整下面程序，并写出程序运行结果。

```

#include <iostream.h>

```



```

class complex{
    double real,imag;
public:
    complex(double r=0,double i=0)    { real=r; imag=i; }
    complex operator +( complex a )
    { double r=a.real +real;    double i=a.imag+ imag;    return complex(r,i);    }
    void show(){ cout<<real<<"+"<<imag<<"i"<<endl; }
};

void main()
{
    complex c1(5,3),c2(1,2),c3;
    c3=c1.operator+(7); //请写出与词句等价的调用形式: c3=c1+7;
    c3.show();
    c3=c2.operator+(c1); //请写出与词句等价的调用形式: c3=c2+c1;
    c3.show();
    //请写出程序运行结果: 12+3i
    6+5i
}

```

8, C++将与输入输出有关的操作定义为一个类体系, 存放在系统类库中, 此类体系叫做流类, 提供此类体系的系统类库叫做流类库, ios 类是 istream 类和 ostream 类的虚基类, 用来提供对流进行格式化 i/o 操作和错误处理的成员函数。用关键字virtual 可将公共基类说明为虚基类。从 ios 类公有派生的 istream 和 ostream 两个类分别提供对流进行提取操作和插入操作的成员函数。C++中预定义了 4 个流, 分别是: cin、cout、cerr、clog。这 4 个流所连接的具体设备是: 标准输入设备、标准输出设备、标准错误输出设备 (非缓冲)、标准错误输出设备 (缓冲)。