**Part I. Choose A, B, C or D and write your choices in the table. (30 points)**

1. 下面语句正确的是 ( ).  
   (A) for(int n, n<10, n++); **(B)** for( ; ; );(C) for( ); (D) for(int n; n<10; n++);
2. Among the following 4 expressions, ( ) is suitable for the blank in the assignment statement a= .  
   　　  
   **(A)** new char[strlen(aa)+1] (B) char[strlen(aa)+1]  
   (C) char[strlen(aa)] (D) new char[sizeof(aa)-1]

class A   
{ private: char \*a;  
 public: A():a(0){}

A(char \*aa)  
 { a= ;  
 strcpy(a,aa); }

~A() { delete [] a;} };

1. 面向对象思想中，对象间的“IS-A”关系在C++语言中由（ ）描述  
   **A：父类和子类** B：对象作为类成员  
   C：指向对象的指针作为类成员 D：静态数据成员
2. 下面选项中不属于面向对象语言程序基本特征的是（ ）  
   A：继承性 B：多态性 **C：相似性** D：封装性
3. 下列关于C++语言类成员函数的叙述中，正确的是（ ）  
   A：每个函数至少要具有一个参数 B：每个函数都必须返回一个值  
   **C：函数在被调用之前必须先声明** D：函数不能自己调用自己
4. C++中，关于类的成员函数，下述说法正确的是 ( ).  
   (A) 成员函数至少有一个参数 (B) 成员函数必须有返回值   
   (C) 所有成员函数都能够重载 **(D) 所有成员函数都可以直接调用**
5. C++ 提供重载函数的目的是 ( ).  
   (A) 共享函数的实现 (B) 减小函数使用的存储空间   
   (C) 提高运行效率 **(D) 方便编程和提高程序可读性**
6. C++ uses ( ) to implement dynamic polymorphism  
   (A) constructor (B) overload function (C) deconstructor **(D) virtual function**
7. 下列说法正确的是 ( ).  
   (A) 缺省构造函数可以重载 **(B) 所有构造函数的名字相同**  
   (C) 拷贝构造函数不能重载 (D) 缺省拷贝构造函数可以重载
8. About object, ( ) is incorrect among the following statements.  
   **(A) It cannot be assigned to another one.** (B) It can be used as a function argument.   
   (C) It can be assigned to an array element (D) It can be used as a data member of another class.
9. 下面所列 函数Func( ) 的4个定义中，正确的是 ( ).  
   (A) **(B)**   
   (C) (D)

int& func(int& m)

{ int &r=m;

return r; }

int& func(int& m)

{ m\*=2;

return m; }

int& func(int m)

{ int &r=m;

return r; }

int& func(int m)

{ m\*=2;

return m; }

1. Among the following programs, the function of ( ) is to exchange the value of the two actual arguments.  
   (A) (B)   
   (C) **(D)**

void fun(int &x, int &y)  
{ x=y; y=x; }

void fun(int x, int y)  
{ int t=x; x=y; y=t; }

void fun(int \*px, int \*py)  
{ int \*pt=px; px=py; py=pt; }

void fun(int &x, int &y)  
{ x=x+y; y=x-y; x=x-y; }

1. Among the following 4 member functions of class MyClass, ( ) is a copy constructor  
   **(A)** MyClass(MyClass& Obj){…} (B) MyClass(MyClass c1, MyClass c2){…}  
   (C) MyClass(MyClass\* Obj){…} (D) MyClass(void){…}
2. Among the following 4 possible results of running the following program, ( ) is correct.  
   　　  
   (A) No output displayed due to failing to pass compilation (B) 0, 0, 2, 3, 1  
   (C) 0, x, 2, 3, 2 (where, x: a random digit) **(D) 0, 0, 2, 3, 2**

class MyClass  
{ public: int m\_nN;  
 MyClass(int n=0) { m\_nN=n; }   
};

void main()  
{ cout<<MyClass(0).m\_nN<<", ";

MyClass c1; cout<<c1.m\_nN<<", ";

MyClass c2=MyClass(2); cout<<c2.m\_nN<<", ";

c1=MyClass(3); cout<<c1.m\_nN<<", ";

MyClass &rC=MyClass();  
 rC=c2; c2=c1; cout<<rC.m\_nN<<endl;  
}

1. 下方所列的4个屏幕显示中哪一个是下面程序的输出.  
   　　  
   (A) (B) (C) (D)

#include "iostream"  
using namespace std;

class A   
{ public: void a() { cout<<" A::a()"<<endl; } };

class B : virtual public A  
{ public: void a()  
 { cout<<" B::a()"<<endl; } };

class C: virtual public A, public B  
{ public: void g(){ a(); }; };

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

**A::a()**

**A::a()  
B::a()**

**B::a()**

**B::a()  
A::a()**

1. \_\_\_\_\_\_\_\_ identifies a constant pointer to changeable double data.

**(A)** const double& value **(B)** double& const value

**(C)** const double\* value  **(D)** double\* const value

1. \_\_\_\_\_\_\_\_ will be printed on the screen after running.

void main() { int i = 100; int &x = i; cout << &x << "\_\_" << x ; }

1. Compiler error (B) Program will crash (C) MemoryAddress\_\_100 (D) 100\_100
2. When creating an object (holding float data) of template Tstack， is right.

(A) Tstack ( ) obj; (B) Tstack (float) obj;

(C) Tstack < float > obj; (D) Tstack < > obj;

1. Among the following 4 member functions of class MyClass, \_\_\_\_\_\_\_\_ is an interface of copy-constructor

(A) MyClass(const MyClass& Obj); (B) MyClass(const MyClass\* obj);

(C) MyClass(const MyClass Obj); (D) MyClass();

1. \_\_\_\_\_\_\_\_ does not belong to the features of object oriented thinking.

(A) Inheritance (B) Polymorphism (C) overloading (D) Encapsulation

1. About the two functions ① and ②, \_\_\_\_\_\_\_\_ is right.

class Example

{ int member;

public:

void set(int a) const { member = a; cout << member; } ①

friend void show(const Example& E) const { cout << E.member; } ②

};

(A) None (B) ①② (C) ① (D) ②

1. C++ uses \_\_\_\_\_\_\_\_ to implement dynamic polymorphism in running time.

(A) overload function (B) virtual function (C) constructor (D) destructor

1. If “+=” is declared as a member function, “obj1 += obj2” will be compiled as \_\_\_\_\_\_\_\_

(A) obj1.operator += (obj2) (B) obj2.operator += (obj1)

(C) operator += (obj1, obj2) (D) operator+= (obj2, obj1)

1. The initialized-list of a constructor of a class can NOT be used to initialize \_\_\_\_\_\_\_\_.

(A) const data members (B) reference data members

(C) static data members (D) member objects

1. \_\_\_\_\_\_\_ can NOT be a member of the class Example.

(A) Example E; (B) Example \*p ; (C) Example & ref; (D) string str;

1. \_\_\_\_\_\_\_\_ is right.

(A) A friend function must be defined in the class

(B) A friend function is a member function of the class.

(C) A friend function must be defined in the public part of the class.

(D) A friend function can access the private members of the class.

1. Consider the following codes carefully, \_\_\_\_\_\_\_\_\_\_\_ is right.

class Biology

{ public:

Biology( ) { }  
 virtual void Move( ) = 0;  
};

class Kangaroo : public Biology  
{ public:  
 void Move( ) { }  
 void Jumping( ) { }  
};

(A) Biology & pb = new Kangaroo; (B) Biology\* pb = new Kangaroo;

(C) Biology & pb = new Biology; (D) Biology\* pb = new Biology;

1. If *S1* is an object of the defined class ***Sample***, the statement “***Sample* *S2 = S1*;** ” will call \_\_\_\_\_\_\_\_ automatically.

(A) constructor without arguments (B) constructor with arguments

(C) operator= (D) copy-constructor

1. After executing the following codes, \_\_\_\_\_\_\_\_is the right result.

#include <iostream>  
using namespace std;  
class Base  
{ public:  
 void play() {cout << "A::play, " ;}  
};

class Derived : public Base  
{ public:  
 void play() {cout <<"B::play, " ;}  
};

void tune( Base& obj ) { obj.play(); }

void main()

{ Derived flute1;  
tune(flute1);   
Base flute2;  
tune(flute2);  
}

(A) A::play, B::play, (B) A::play, A::play, (C) B::play, B::play, (D) B::play, A::play,

1. The output is \_\_\_\_\_\_\_ after the following program executed.

|  |  |
| --- | --- |
| #include <iostream >  using namespace std;  class A  { int i;  public:  A(){ cout<<'A'; }  virtual void f() { cout<<"FA";};  };  class B: virtual public A  { int i;  public:  B(){ cout<<'B'; }  void f() { cout<<"FB";};  }; | class C: virtual public A  { int i;  public:  C(){ cout<<'C'; }  void f() { cout<<"FC";};  };  class D: public B, public C  { int i;  public:  D(){ cout<<'D'; }  void f() { cout<<"FD";};  };  void main( )  { D\* p = new D( ); A\* q = p; p->f(); delete p; } |

(A) ABACDFD (B) ABACDFA (C) ABCDFD (D) ABCDFA

**Part II. Fill blanks or write out the outputs. (40 points)**

1. (10 points) Write the outputs.

#include <iostream>

using namespace std;

class Compute

{ private:

int value;

public:

Compute(int a = 0) { value = a; cout << "constructor is called." << endl; }

Compute(const Compute& C) { value = C.value; cout << "copy-constructor is called." << endl; }

Compute& operator=(const Compute& C)

{ value = C.value; cout << "assignment is called." << endl; return \*this; }

operator int() { cout << "operator int() is called." << endl; return value; }

~Compute() { cout << "destructor is called. value = " << value << endl; }

friend const Compute operator-(const Compute &C1, const Compute &C2);

};

const Compute operator-(const Compute &C1, const Compute &C2)

{

cout << "operator-() is called." << endl;

return Compute (C1.value - C2.value);

}

void main()

{ Compute m(5), n=m;  
 m = m - n;  
 int result = m;  
 cout << "result = " << result << endl;

}

2. (10 points) **Fill the blanks and write the outputs**.

#include <iostream>

using namespace std;

class Myclass

{

private:

const int length;

static int num;

public:

Myclass (int r = 0) : ① (r)

{ num++;

cout << "length = " << length << ", num = " << num << endl;

}

int GetLength() { return num \* length; }

~ Myclass ()

{ num--;

cout << "length = " << length << ", num = " << num << endl;

}

};

② = 3;

void main()

{ Myclass S1(3), &S2 = S1, \*S3 = new Myclass[2];

cout << "The length is " << S1.GetLength() << endl;

delete[ ] S3;

cout << "The length is " << S1.GetLength() << endl;

}

3. (10 points) Write the outputs.

# include <iostream>

using namespace std;

class Instrument

{ public:

virtual void play() const { cout << "Instrument::play" << endl;}

virtual char\* what() const {return "Instrument"; }

};

class Wind : public Instrument

{ public:

void play() const { cout << "Wind::play" << endl; }

char\* what() const { return "Wind"; }

};

class Brass : public Wind

{ public:

char\* what() const { return "Brass"; }

};

void tune(Instrument& i) { i.play( );}

void main()

{ Wind flute;

Brass horn;

tune(flute);

tune(horn);

}

4. (10 points) Complete ① and ②, and write the outputs.

#include <iostream>

using namespace std;

template<typename T>

class Stack

{

public:

Stack(): top(0) { }

void push(const T& value); // push an element to stack

T pop(); // Get an element at the top of stack

private:

T stack[10];

int top;

};

① { stack[top++] = value; } // push an element to stack

② { return stack[--top]; } // Get an element at the top of stack

void main( )

{

Stack<int> is;

for(int i = 0; i < 5; i++) is.push(i \* 2);

for(int k = 0; k < 5; k++) cout << is.pop() << ",";

cout << endl;

Stack<double> ds;

for( int i = 0; i < 5; i++) ds.push(i \* 0.5);

for( int k = 0; k < 5; k++) cout << ds.pop() <<",";

}

1. (10 points) 写出下面程序的输出结果

#include "iostream"

using namespace std;

class CRectangle

{ public:

int area() { return 4; }

};

class CSquare : virtual public CRectangle

{ public:

int area() { return 8; }

};

class CDiamond : virtual public CSquare

{ public:

int area() { return 16; };

};

void main(void)

{ CSquare sq;

CDiamond dia;

cout<<sq.area()<<endl;

CRectangle &rc=sq;

rc=sq;

cout<<rc.area()<<endl;

rc=dia;

cout<<rc.area()<<endl;

}

The outputs are as followings:

8

4

4

2. (10 points) 填空并写出下面程序的输出.

#include "iostream"  
using namespace std;

class MyClass

{ private: double m\_dData;  
 public:  
MyClass(double d=0)  
{ cout<<"An object is created by "<<d<<endl;  
  m\_dData=d; }

operator double ( )  
{ cout<<"Type-cast of double() is called."<<endl;  
  return m\_dData; }

double GetData() { return m\_dData; }

friend MyClass operator - (double d, MyClass m);

friend ;

};

MyClass operator - (double d, MyClass m)  
{ cout<<"The friend operator \'-\' is called."<<endl;  
 return MyClass(d-m.m\_dData); }

MyClass operator + (double d, MyClass m)  
{ cout<<"The operator \'+\' is called."<<endl;  
 return MyClass(d-m.GetData()); }

int main(int argc, char\* argv[])  
{ MyClass m(3);  
 m = m - 2;  
 cout<<"The data member of m is "<<m.m\_dData<<endl;  
 return 0;  
}

the blank is int main(int argc, char\* argv[])

the outputs are as followings:

An object is created by 3

Type-cast of double() is called

An object is created by 1

The data member of m is 1

3. (10 points) 填空并写出下面程序的输出.

#include "iostream"  
using namespace std;

class Embedded  
{ private: int e;

public:  
 Embedded(int n) { e=n; }  
 int GetEmbedded() { return e; } // Display the data member value  
};

class Base  
{ private: int b; Embedded em;

public:  
 Base(int i=0) : em(100\*i) { b=i; }

friend ① ;

friend int main(int argc, char\* argv[]);  
};

class Drv1 : virtual public Base  
{ private: int d1;

public: Drv1(int i, int j) { d1=j; }  
};

class Drv2 : virtual public Base  
{ private: int d2;

public: Drv2(int i, int j) { d2=j; }  
};

class Drv : virtual public Drv1, virtual public Drv2  
{ private: int d;

public:  
 Drv(int i, int j, int k, int l) : Base(i), Drv1(i, j), Drv2(i, k)  
 { d=l; }

Embedded& GetEmbedded() { return Base::em; }

};

int main(int argc, char\* argv[])  
{ Drv obj(1, 2, 3, 4);  
 obj.Drv1::b=11;  
 obj.Drv2::b=21;

cout<<obj. ② <<endl;  
 // This wants to display the data member value of the  
 // embedded objct Base::em

cout<<obj.Drv1::b<<endl;  
 cout<<obj.Drv2::b<<endl;  
 return 0;  
}

The blank ① is *class Drv*

The blank ② is GetEmbedded().GetEmbedded()

The outputs are as followings:

100

21

21

4. (10 points) 写出下面程序的输出.

#include "iostream"  
using namespace std;

class MyComplex   
{ private:  
 double m\_dR;  
 double m\_dI;

public:  
 template <class T>  
 MyComplex(T r=0, T i=0)  
 { m\_dR=r; m\_dI=i;  
 cout<<"constructor"<<endl; }

operator double ()  
 { cout<<"type cast"<<endl;  
 return m\_dR; }

friend ostream& operator << (ostream& os, MyComplex c);  
};

ostream& operator << (ostream& os, MyComplex c)  
{ os << c.m\_dR << " + i" <<c.m\_dI <<endl;  
 return os; }

void main()

{ MyComplex c1(1.0, 2.0);

c1=c1+1;

cout<<c1;

}

* 1. **constructor**
  2. **type cast**
  3. **constructor**
  4. **2 + i0**

**PartIII. Programming. (30 points)**

1. (15 points) The classes in the following program are:

**Point** : describes a point on the plane.

**Circle** : describes a circle at a point of type **Point**.

**Rect** : describes a rectangle at a point of type **Point**. （rectangle：矩形）

**Triangle** : describes a triangle at a point of type **Point**. （triangle：三角形）

**Cylinder** : the cylinder with a bottom of type **Circle**, **Rect** or **Triangle**.   
It locates at a point of the plane. (cylinder: 柱体)

Please complete the functions or statements that are not completed in the program.

#include "iostream"  
using namespace std;

class Point  
{ private: double m\_dX;  
 double m\_dY;

public: (1)   
 // Above blank is a constructor with default argument values  
};

class Bottom  
{ protected: double m\_BttmLen;  
 double m\_Height;  
 Point m\_Pos;

public:  
 Bottom(double x=0, double y=0, double l=0, double h=0)  
 : m\_Pos(x, y) { m\_BttmLen=l; m\_Height=h; }  
 virtual double Area() (2) ; // Here is a pure function  
 virtual double Volume() (3) ; // Here is a pure function  
};

class Circle : (4) Bottom  
{ private: double m\_dRadius;  
 public:  
 Circle(double x=0, double y=0, double r=0) { m\_dRadius=r; }  
 virtual double Area() { (5) } // Returns the area  
 virtual double Volume() { return 0; }  
};

class Rect : virtual public Bottom  
{ public:  
 Rect(double x, double y, double BttmLen, double Height)  
 : Bottom(x, y, BttmLen, Height) { }

virtual double Area() { (6) ; } // Returns the area  
 virtual double Volume() { return 0; }  
};

class Triangle : virtual public Bottom  
{ public:  
 Triangle(double x, double y, double l, double h)  
 : Bottom(x, y, l, h) { }

virtual double Area() { return 0.5\*m\_BttmLen\*m\_Height; }  
 virtual double Volume() { return 0; }  
};

template < (7) >  
class Cylinder : public virtual BTYPE  
{ private: double m\_bHeight;

public:  
 Cylinder(double x, double y, double BL, double bH, double h)  
 : (8) (x, y, BL, bH), Bottom(x, y, BL, bH) { m\_bHeight=h; }

Cylinder(double x, double y, double h, double r)  
 : Circle(x, y, r), Bottom(x, y, 0, 0) { m\_bHeight=h; }

double Volume() { (9) }  
};

int main(int argc, char\* argv[])  
{ Bottom \*pBttm;  
 (10) Clndr1(1,2,3, 4); // Clndr1 is a cylinder with a circular bottom

(11) Clndr2(1,2,3,4,5); // Clndr2 is a cylinder with a triangular bottom

(12) Clndr3(1,2,3,4,5); // Clndr3 is a cylinder with a rectangular bottom

(13) ;   
 cout<<pBttm->Area()<<endl; // prints the area of Clndr1

(14) ;   
 cout<<pBttm->Area()<<endl; // prints the area of Clndr2

(15) ;   
 cout<<pBttm->Area()<<endl; // prints the area of Clndr3

return 0;  
}

**答案：**

**(1) Point(double x=0, double y=0) { m\_dX=x; m\_dY=y; }**

**(2) =0**

**(3) =0**

**(4) virtual public**

**(5) return 3.14\*m\_dRadius\*m\_dRadius**

**(6) return m\_BttmLen\*m\_Height**

**(7) class BTYPE**

**(8) BTYPE**

**(9) return Area()\*m\_bHeight**

**(10) Cylinder<Circle>**

**(11) Cylinder<Triangle>**

**(12) Cylinder<Rect>**

**(13) pBttm=&Clndr1**

**(14) pBttm=&Clndr2**

**(15) pBttm=&Clndr3**

2. (15 points) Use class template to implement a simple and universal stack, named **MyStack**, which contains items of type **TYPE** and the maximum number of items can reach up to **MaxItemNum**. The detailed requirements are as follows.

**⑴** In MyStack, the following member functions are defined:

**int Push(MyStack e)** : Inserts an element of type TYPE at the end of the controlled array.  
It returns the position of the end item

**MyStack GetTop()** : Return the top item which locates at the end of the controlled array.

**MyStack Pop()** : Removes and return the top item at the end of the controlled array, which must be non-empty.

**void CleanUp()** : Delete all item in the MyStack.

**booI sEmpty()** : The member function returns true for an empty controlled sequence.

**and so on** : Other necessary functions such as some constructors and so on.

**⑵** Use your MyStack to invert a string defined by yourself. "abcdefg", for example, will be inverted as "gfedcba"

**参考答案：**

#include "iostream"  
using namespace std;

template <class TYPE, int MaxItemNum>  
class MyStack   
{ private: TYPE Data[MaxItemNum];  
 int Top;

public:  
 MyStack() { memset(Data,'\0', MaxItemNum\*sizeof(TYPE)); Top=0; }

int Push(TYPE e) { Top+=1; Data[Top]=e; return Top; }

TYPE Pop()  
 { TYPE e=Data[Top];  
 if(Top>=1) { Top-=1; return e; }  
 else { cout<<"No item in the MyStack."<<endl; exit(1); }  
 }

int GetItemNum() { return Top; }

void CleanUp() { memset(Data,'\0', MaxItemNum\*sezeof(TYPE)); Top=0; }

bool IsEmpty() { return Top<=0; }  
 TYPE GetTop() { return Data[Top];} };

int main(int argc, char\* argv[])

{ MyStack <char, 100> s;

char str[]="abcdefg";

int len=strlen(str);

for(int i=0; i<len; i++) s.Push(str[i]);

for(int i=0; i<len; i++) str[i]=s.Pop();

cout<<str<<endl;

return 0; }

1. (15 points) Define necessary member functions for the class Myclass in order to allow users to use it as in the main( ).

class Myclass

{ int \*p;

public:

Myclass (int i) { p = new int( i ); }

// Define necessary member functions in the following

…...

};

void main()

{ Myclass obj1(5);

Myclass obj2(obj1);

Myclass obj3(6), obj4(10);

obj4 = obj3;

cout << obj1 << endl;

}

2. (15 points) Define three classes, ***Shape*, *Point*** and ***Circle:***

(1) ***Shape*** is an abstract class with two pure virtual functions: ***Area()*** and ***Perimeter()***.

(2) ***Point*** has data members: ***X*** and ***Y***,as coordinates of a point.

(3) ***Circle*** is inherited from ***Shape***, with data members: ***Radius*(半径)**, and ***P*(圆心)** of class ***Point***, and

1. member function ***Area()*** to get the area of a circle;
2. member function ***Perimeter()*** to get the perimeter(周长) of a circle.

(4) If necessary, you can add other members for the three classes.

(5) A programmer can use ***Shape***, ***Point*** and ***Circle*** in main() as follows:

void main()

{ Point A(5.8, 4.1);

Shape \*pS = new Circle(A, 3); // 3 is radius of a circle

cout << pS->Area() << endl; // show the area of the circle

cout << pS->Perimeter() << endl; // show the perimeter of the circle

delete pS;

}