C/C++ Basic Skill Examination

Part I Multiple Choice Questions (only one answer is valid)

1. **Which one of the following name is valid? ( )**

<A> **int** double = 3.1415926; <B> **signed** **char** \_;

<C> **bool** non-Determin; <D> **float** static = 3.14f;

1. **Which one is not a key word in C/C++ programming language? ( )**

<A> extern <B> volatile

<C> until <D> explicit

1. **Which one of the following variable’s definitions is illegal? ( )**

<A> **int** iCount, i, j = 0;

<B> **unsigned** **char** szArray[128] = {0};

<C> **extern** **std**::**string** name;

<D> **int** a, **int** \*pRef = &a, \*pDef;

1. **Which is wrong about the following variable’s initialization? ( )**

<A> **int** iArr[100] = 0;

<B> **const** **int** iVal = 128;

**int** **const** \* **const** p = &iVal;

<C> **std**::**string** szName(“Hello, world!”);

<D> **void** (\*p)(int, float) = 0;

1. **In C++, what is the difference between a struct and class? ( )**

<A> Structs are more effective than classes in C++.

<B> Classes use less memory since it is newer technology.

<C> You cannot implement functions when you use a struct.

<D> The default access specifier, members of a class are private, members of a struct are public.

1. **Which kind of loops is invalid in C/C++ programming language? ( )**

<A>

**do** {

statement;

}**while**(condition);

<B>

**for** (initialization; condition; increase)

{

statement;

}

<C>

**while** (condition)

{

statement;

}

<D>

**loop** (start point) **to** (end point)

statement;

**end** **loop**

1. **Given the following declarations, determine which call is illegal in C++. ( )**

**double** calc(**double**);

**int** count(**const** **string** &, **char**);

**int** sum(**vector**<**int**>::**iterator**, **vector**<**int**>::**iterator**, **int**);

**vector**<**int**> vec(10);

<A> calc(2.12, 34.5); <B> count(“abcda”, ‘a’);

<C> calc(12.6); <D> sum(vec.**begin**(), vec.**end**(), 3);

1. **Which class’ definition is invalid about the following definitions? ( )**

<A>

**class** **singleton**

{

**public**:

**static** **singleton** \* Instance(){**return** &\_instance;}

**private**:

**static** **singleton** \_instance;

};

<B>

**class** **singleton**

{

**public**:

**singleton**& Instance(){**return** \_instance;}

**private**:

**int** \_RefCount = 0;

**singleton** \_instance;

};

<C>

**class** **screen**

{

**public**:

**unsigned** **short** GetX(){**return** \_XCoordinate;}

**unsigned** **short** GetY(){**return** \_YCoordinate;}

**private**:

**unsigned** **short** \_XCoordinate;

**unsigned** **short** \_YCoordinate;

};

<D>

**class** person

{

**pubic**:

**std**::**string**& Name() {**return** \_name;}

**unsigned** **char** Age(){**return** \_age;}

**private**:

**std**::**string** \_name;

**unsigned** **char** \_age;

};

1. **Which one of the following functions can’t swap two values: ( )**

<A> **void** swap(**int** \*ix, **int** \*iy) {

**int** \*p = ix;

ix = iy;

iy = p;

}

<B> **void** swap(**int** \*a, **int** \*b) {

\*a ^= \*b;

\*b ^= \*a;

\*a ^= \*b;

}

<C> **void** swap(**int** &a, **int** &b) {

**int** tmp = a;

a = b;

b = tmp;

}

<D> **void** swap(**int** \*&a, **int** \*&b) {

**int** \*tmp = a;

a = b;

b = tmp;

}

1. **Which ones of the following template definitions are invalid: ( )**

<A> **template** <**class** T, U, **class** V>

**void** Function(T, U, V);

<B> **template** <**class** T1, **typename** T2, **class** T3>

T1 Function(T2, T3);

<C> **template** <**class** myT, **class** myT>

**void** Function(myT, myT);

<D> **template** <**class** T>

T Function(T, **unsigned** **int** \*);

1. **Which of the following statement is true: ( )**

<A> A class must provide at least one constructor.

<B> static data members can be declared in class body, and it must be defined (exactly once) outside the class body.

<C> A default constructor is a constructor with no parameters for its parameter list.

<D> If a class does not define a default constructor, the compiler generates one automatically, initializing each data member to the default value of its associated type.

1. **What is the greatest value in using virtual functions within C++? ( )**

<A> They prevent the base class function from ever being called.

<B> They allow you to provide unique behavior from derived classes.

<C> Since they are “virtual” they sometimes don’t really exist.

<D> Database technologies.

1. **Given the following program:**

**int** \* p = NULL;

**int** a[] = {1, 2, 3};

p = a;

\*(p+1) = ( )

A: 1 B: 2 C: 3 D: unknow

1. **Given the definition:**

#**define** SQURT(a) a\*a

**int** a = 4;

SQURT(a+1) = ( )

A: 9 B: 20 C: 18 D: 17

Part II Fill the blank

1. **Write the output of program**

**class** **Base** {

**public:**

**Base**(int i): mem(0) { }

**protected**:

**int** mem;

};

**class** **Derived** : **public** **Base** {

**public**:

**Derived**(**int** i): mem(i),Base(i){ }

**int** GetMem() { **return** mem; }

**protected**:

**int** mem;

};

**int** main()

{

**Derived** d(42);

**std**::**cout** << d.GetMem() << **std**::**endl**;

**return** 0;

}

Output:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **If program running on 32-bits platform, please write the answer according to requirement.**

**class** Display

{

**public**:

**virtual** **void** print() **const** = 0;

};

**class** LCDDisplay : **public** Display

{

**public**:

**virtual** **void** print() **const** {**std**::**cout** << “Hello, LCD!” << **std**::**endl**;}

**private**:

**long** \_Size;

};

**sizeof**(LCDDisplay)> = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Read the following program, and answer the question.**

**class** Base {

**public**:

Base()

{

**std**::**cout** << “Constructor Base.” << **std**::**endl**;

}

**virtual** ~Base()

{

**std**::**cout** << “Destructor Base.” << **std**::**endl**;

}

**virtual** **void** output() **const**

{

**std**::**cout** << “Hello, Base.” << **std**::**endl**;

}

};

**clase** Derive: public Base {

**public**:

Derive()

{

**std**::**cout** << “Constructor Derive.” << **std**::**endl**;

}

**virtual** ~Derive()

{

**std**::**cout** << “Destructor Derive.” << **std**::**endl**;

}

**virtual** **void** output() **const**

{

**std**::**cout** << “Hello, Derive.” << **std**::**endl**;

}

};

**int** main()

{

Derive b;

b.output();

**return** 0;

}

Please write down output of program run:

1. **Assuming a = 0, please write down the output of program**

**switch(a)**

**{**

**default: printf(“d”);**

**case 0: printf(“a”);**

**case 1: printf(“b”);**

**}**

Output:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Define a function to determine whether the system is big-endian**

Function declaration: bool isBigEndian ();

Part III Program Correction

1. **Please correct the following program**

1 **class** person

2 {

3 **public**:

4 person():\_waveTimes(0){}

5 **virtual** ~person(){}

6 **static** **int** getSalary(){ **return** \_mySalary;}

7 **int** waveHand() **const**

8 {

9 **return** ++\_waveTimes;

10 }

11 **void** DoAction(behavior curBehavior){ /\* to do action\*/ };

12 **void** Smile(**const** int teethNumber) **const**

13 {

14 **if** (teethNumber < 0)

15 {

16 teethNumber = 0;

17 DoAction(eNone);

18 }

19 **else** (teethNumber < 4)

20 {

21 DoAction(eSimle);

22 }

23 **else** (teethNumber > 4)

24 {

25 DoAction(eLaughing);

26 }

27 **else**

28 {

29 DoAction(eCry);

30 }

31 }

32 **private**:

33 **enum** behavior {

34  eSmile,

35 eLaughing

36 eCry,

37 eNone

38 };

39 **static const int** \_fingerNumber = 10;

40 **static int** \_teethNumber = 24;

41 **int** \_mySalary;

42 **int** \_waveTimes;

43 behavior curBehavior;

44 };

Please write down error line number and why. (Chinese are acceptable for reason)

Line:\_\_\_\_\_\_\_ reason:

Line:\_\_\_\_\_\_\_ reason:

Line:\_\_\_\_\_\_\_ reason:

Line:\_\_\_\_\_\_\_ reason:

Line:\_\_\_\_\_\_\_ reason:

Line:\_\_\_\_\_\_\_reason:

Part V Write down the program

1. **Some concepts explain:**

**Design Pattern**

A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context.

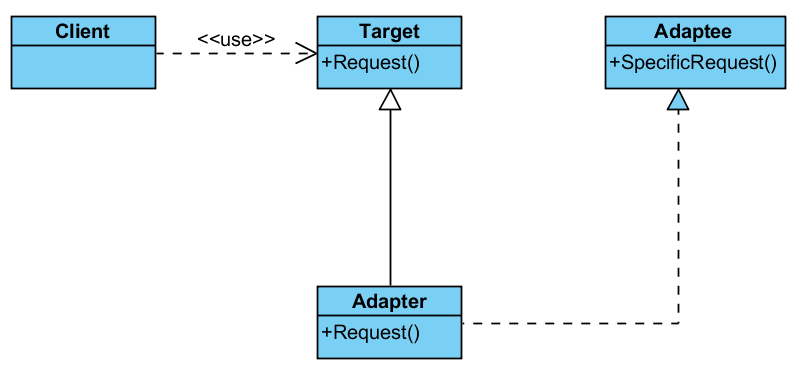
**Adapter Pattern**

One of 23 kinds of Design Patterns in GOF’s <<Design Patterns>>. Its intent is convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

--------------------------<<Design Patterns>> of GOF

Use the Adapter pattern when

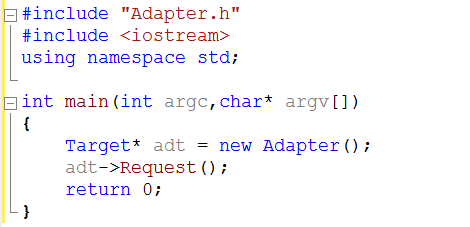
* you want to use an existing class, and its interface does not match the one you need.
* you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.



**Problem description:**

**Now, we had an interface Request in Target, It output “Target::Request”, and there is another interface exist in current system, It is SpecificRequest in Adptee. It ouput “Adaptee::Specific Request”;**

**Current client code as follows:**

****

**Please use adapter pattern implement Request interface, make it output “Adaptee::Specific Request”**

**Implement all of source code make client code can run correctly.**

Your implementation code:

Part IV Optional Question

1. **Why do you think C++ wasn't named ++C? (Chinese are acceptable)**
2. **What are your comments regarding this test? (Chinese are acceptable)**
   1. The stupid question\_\_\_\_\_, Reason:
   2. The best question\_\_\_\_\_\_\_, Reason:
   3. The most interested question\_\_\_\_\_\_\_\_\_\_, Reason:
   4. I have found some problems in question\_\_\_\_\_, Reason:
   5. I want to have face to face talk regarding question\_\_\_\_\_\_
   6. Other comments: