**Derived Class:**

**Writer:** Sayak Haldar

**1. Where is the derived class is derived from?**

a) derived

b) base

c) both derived & base

d) None of the mentioned

**Answer: b**

Explanation: Because derived inherits functions and variables from base.

(Now, there is multilevel inheritence. Suppose, Class B inherits A, class C inherits B. Now, For C, B acts as a base class)

#include<cstdio>

#include<iostream>

using namespace std;

class GrandParent

{

public:

void display()

{

cout<<"Grandparent's display function called"<<endl;

}

};

class Parent: public GrandParent

{

public:

void display()

{

cout<<"Parent's display function called"<<endl;

}

};

class Child: public Parent

{

};

int main()

{

Child child\_obj;

child\_obj.display();

return 0;

}

For instance, this will print:

**Parent's display function called**

Because, Child class does not have it’s own display function.But, it inherits from Parent class. (Parent class overrides the definition of display() function of GrandParent.)

**Now, what does that mean?**

Parent is acting as Base class for Child class.

**2. Pick out the correct statement.**

a) A derived class’s constructor cannot explicitly invokes its base class’s constructor

b) A derived class’s destructor cannot invoke its base class’s destructor

c) A derived class’s destructor can invoke its base class’s destructor

d) None of the mentioned

Answer: b

Explanation: Destructors are automatically invoked when a object goes out of scope or when a dynamically allocated object is deleted. Inheritance does not change this behavior. This is the reason a derived destructor cannot invoke its base class destructor.

**3. Which of the following can derived class inherit?**

a) members

b) functions

c) both members & functions

d) none of the mentioned

**Answer: c**

**4. What is the output of this program?**

#include <iostream>

using namespace std;

class A

{

public:

A(int n )

{

cout << n;

}

};

class B: public A

{

public:

B(int n, double d)

: A(n)

{

cout << d;

}

};

class C: public B

{

public:

C(int n, double d, char ch)

: B(n, d)

{

cout <<ch;

}

};

int main()

{

C c(5, 4.3, 'R');

return 0;

}

a) 54.3R

b) R4.35

c) 4.3R5

d) None of the mentioned

**Answer: a**

Explanation: In this program, We are passing the value and manipulating by using the derived class.

The answer is depended upon the constructor calling order.

**5. What is the output of this program?**

#include <iostream>

using namespace std;

class BaseClass

{

protected:

int i;

public:

BaseClass(int x)

{

i = x;

}

~BaseClass()

{

}

};

class DerivedClass: public BaseClass

{

int j;

public:

DerivedClass(int x, int y): BaseClass(y)

{

j = x;

}

~DerivedClass()

{

}

void show()

{

cout << i << " " << j << endl;

}

};

int main()

{

DerivedClass ob(3, 4);

ob.show();

return 0;

}

a) 3 4

b) 4 3

c) 4

d) 3

**Answer: b**

Explanation: In this program, We are passing the values and assigning it to i and j and we are printing it.

4 3

Again, It is depended upon the constructor calling order.

When a derived class is created. First, the base class’s constructor is invoked, after that the derived class’s constructor is invoked.

**6. What is the output of this program?**

#include <iostream>

using namespace std;

class Base

{

public:

int m;

Base(int n=0)

: m(n)

{

cout << "Base" << endl;

}

};

class Derived: public Base

{

public:

double d;

Derived(double de = 0.0)

: d(de)

{

cout << "Derived" << endl;

}

};

int main()

{

cout << "Instantiating Base" << endl;

Base cBase;

cout << "Instantiating Derived" << endl;

Derived cDerived;

return 0;

}

a) Instantiating Base

Base

Instantiating Derived

Base

Derived

b) Instantiating Base

Instantiating Derived

Base

Derived

c) Instantiating Base

Base

Instantiating Derived

Base

d) None of the mentioned

**Answer: a**

Explanation: In this program, We are printing the execution order of the program.

**Another problem related to constructor calling order.**

Instantiating Base

Base

Instantiating Derived

Base

Derived

**7. What is the output of this program?**

#include <iostream>

using namespace std;

class Parent

{

public:

Parent (void)

{

cout << "Parent()\n";

}

Parent (int i)

{

cout << "Parent("<< i << ")\n";

};

Parent (void)

{

cout << "~Parent()\n";

};

};

class Child1 : public Parent { };

class Child2 : public Parent

{

public:

Child2 (void)

{

cout << "Child2()\n";

}

Child2 (int i) : Parent (i)

{

cout << "Child2(" << i << ")\n";

}

~Child2 (void)

{

cout << "~Child2()\n";

}

};

int main (void)

{

Child1 a;

Child2 b;

Child2 c(42);

return 0;

}

a) Parent()

Parent()

Child2()

Parent(42)

Child2(42)

~Child2()

~Parent()

~Child2()

~Parent()

~Parent()

b) error

c) runtime error

d) none of the mentioned

**Answer: b**

**Explanation:** Compile time error.

Because of the following:

The function/constructor definition of Parent(void) has the same function signature as the constructor starting from line 6. Now, function overloading does not work like that.

**8. What is the output of this program?**

#include<iostream>

using namespace std;

class X

{

int m;

public:

X() : m(10)

{

}

X(int mm): m(mm)

{

}

int getm()

{

return m;

}

};

class Y : public X

{

int n;

public:

Y(int nn) : n(nn) {}

int getn() { return n; }

};

int main()

{

Y yobj( 100 );

cout << yobj.getm() << " " << yobj.getn() << endl;

}

a) 10 100

b) 100 10

c) 10 10

d) 100 100

**Answer: a**

Explanation: In this program, We are passing the value and getting the result by derived class.

10 100

Now, it is a nice conceptual question.

**9. Which operator is used to declare the destructor?**

a) #

b) ~

c) @

d) $

**Answer: b**

**10. Which constructor will initialize the base class data member?**

a) derived class

b) base class

c) class

d) none of the mentioned

**Answer: b**

Explanation: Because it is having the proper data set to initialize, Otherwise it will throw a error. (Compilation error. Not runtime error)

(Generally, In derived class constructor initialized base class members by invoking base class constructor through initializer list.

)

And if the base class only have parametric constructor, it becomes necessary to have initializer list with the derived class constructor.

**(**

**Q.Is there any way I can just call it later instead of calling it on the initializer list?**

No, you cannot. The base class constructor must be called in the initializer list, and it must be called first.

This is from stackoverflow.

**)**

**Now, what do I mean when I said “**And if the base class only have parametric constructor, it becomes necessary to have initializer list with the derived class constructor.”

Check the following program:

#include<cstdio>

class Base

{

private:

int m;

public:

Base(int m)

{

this->m=m;

}

};

class Derived: public Base

{

private:

int n;

public:

Derived()

{

n=0;

}

Derived(int n)

{

this->n=n;

}

};

int main()

{

Derived derived\_obj(12);

return 0;

}

It will give lots of compilation error.

**classKnowledge3.cpp: In constructor ‘Derived::Derived()’:**

**classKnowledge3.cpp:18: error: no matching function for call to ‘Base::Base()’**

**classKnowledge3.cpp:7: note: candidates are: Base::Base(int)**

**classKnowledge3.cpp:3: note: Base::Base(const Base&)**

**classKnowledge3.cpp: In constructor ‘Derived::Derived(int)’:**

**classKnowledge3.cpp:22: error: no matching function for call to ‘Base::Base()’**

**classKnowledge3.cpp:7: note: candidates are: Base::Base(int)**

**classKnowledge3.cpp:3: note: Base::Base(const Base&)**

Now, If base class has a default constructor it wont give error.

#include<cstdio>

class Base

{

private:

int m;

public:

Base()

{

m=0;

}

Base(int m)

{

this->m=m;

}

};

class Derived: public Base

{

private:

int n;

public:

Derived()

{

n=0;

}

Derived(int n)

{

this->n=n;

}

};

int main()

{

Derived derived\_obj(12);

return 0;

}

However, the error could be fixed in a different way. Having initializer list. Which invokes the parametric constructor of the Base class.

#include<cstdio>

class Base

{

private:

int m;

public:

Base(int m)

{

this->m=m;

}

};

class Derived: public Base

{

private:

int n;

public:

Derived():Base(0)

{

n=0;

}

Derived(int n,int m):Base(m)

{

this->n=n;

}

};

int main()

{

Derived derived\_obj(12,6);

return 0;

}