8 Write C++ Programs and incorporating various forms of Inheritance

a) Single Inheritance

```
 #include<iostream>

using namespace std;
class A
{
	public:
		A()
		{
			cout<<"This is super class A"<<endl;
		}
};
class B : public A
{
	public:
		B()
		{
			cout<<"This is sub class B"<<endl;
		}
};
int main()
{
	B obj;
}
```

Output :

```
This is super class A
This is sub class B
```

b) Multiple Inheritance

In multiple inheritance there might be ambiguity problem in sub class.

For example if super class A contains a variable *x* and another super class B also contains a variable *x*, then in sub class C, when *x* is referred, compiler will be confused whether it belongs to class A or class B. To avoid this ambiguity, we use the scope resolution operator.

```
 #include<iostream>

using namespace std;
class A
{
	public:
```

```
                int x;
                A(int x)
                {
                        this->x = x;
                }
                void show()
                {
                        cout<<"A's x = "<<x<<endl;
                }
};
class B
{
        public:
                int x;
                B(int x)
                {
                        this->x = x;
                }
                void show()
                {
                        cout<<"B's x = "<<x<<endl;
                }
};
class C : public A, public B  //multiple inheritance
{
        public:
                C(int a, int b) : A(a), B(b)
                {
                        A::show();
                        B::show();
                }
};
int main()
{
        C objC(10, 20);
}
```

Output:

```
A's x = 10
B's x = 20
```

c) Hierarchical Inheritance

```
   #include<iostream>

using namespace std;
```

```
class A
{
        public:
                A()
                {
                        cout<<"Super class A's constructor"<<endl;
                }
};
class B : public A
{
        public:
                B()
                {
                        cout<<"Sub class B's constructor"<<endl;
                }
};
class C : public A
{
        public:
                C()
                {
                        cout<<"Sub class C's constructor"<<endl;
                }
};
int main()
{
        C objC;
}
```

Output:

```
Super class A's constructor
Sub class C's constructor
```

d) Multi-Level Inheritance

```
  #include<iostream>

using namespace std;
class A
{
        public:
                A()
                {
                        cout<<"This is super class A"<<endl;
                }
};
```

```
class B : public A
{
        public:
                B()
                {
                        cout<<"This is sub class B of super class A"<<endl;
                }
};
class C : public B
{
        public:
                C()
                {
                        cout<<"This is sub class C of super class B"<<endl;
                }
};
int main()
{
        C obj;
}
```

Output:

```
This is super class A
This is sub class B of super class A
This is sub class C of super class B
```

e) Hybrid Inheritance

```
    #include<iostream>

using namespace std;
class A
{
        public:
                A()
                {
                        cout<<"Super class A's constructor"<<endl;
                }
};
class B : public A
{
        public:
                B()
                {
                        cout<<"Sub class B's constructor"<<endl;
                }
};
class C : public A
{
        public:
                C()
                {
                        cout<<"Sub class C's constructor"<<endl;
                }
};
class D: public B
{
        public:
                D()
                {
                        cout<<"Sub class D's constructor"<<endl;
                }
};
int main()
{
        D objD;
}
```

Output:

```
Super class A's constructor
Sub class B's constructor
Sub class D's constructor
```

f) Multipath Inheritance and virtual base class solution for the diamond problem:

```
  #include<iostream>

using namespace std;
class A
{
        protected:
                int x;
        public:
                A()
                {
                        cout<<"Super class A's constructor"<<endl;
                }
                void read()
                {
                        cout<<"Enter value of x: ";
                        cin>>x;
                }
                void show()
                {
                        cout<<"x = "<<x;
                }
};
class B : virtual public A
{
        public:
                B()
                {
                        cout<<"Sub class B's constructor"<<endl;
                }
};
class C : virtual public A
{
        public:
                C()
                {
                        cout<<"Sub class C's constructor"<<endl;
                }
};
class D: public B, public C
{
        public:
                D()
                {
                        cout<<"Sub class D's constructor"<<endl;
                }
};
int main()
{
        D objD;
        objD.read();
        objD.show();
        return 0;
}
```

Output:

```
Super class A's constructor
Sub class B's constructor
Sub class C's constructor
Sub class D's constructor
Enter value of x: 20
x = 20
```

9(a) C++ program to illustrate the order of execution of constructors and destructors in inheritance.

```
#include <iostream>
using namespace std;
class A
{
        public:
                A()
                {
                        cout<<"A's Constructor"<<endl;
                }
                ~A()
                {
                        cout<<"A's Destructor"<<endl;
                }
};
class B : A
{
        public:
                B()
                {
                        cout<<"B's Constructor"<<endl;
                }
                ~B()
                {
                        cout<<"B's Destructor"<<endl;
                }
};
class C : B
{
        public:
                C()
                {
                        cout<<"C's Constructor"<<endl;
                }
                ~C()
                {
                        cout<<"C's Destructor"<<endl;
                }
};
int main()
{
        C c;
        return 0;
}
```

OutPut:

```
A's Constructor
B's Constructor
C's Constructor
C's Destructor
B's Destructor
A's Destructor
```

9(b) C++ program to show how constructors are invoked in derived class

```
#include <iostream>
using namespace std;
class A
{
        protected:
                int x;
        public:
                A(int p)
                {
                        x = p;
                }
};
class B : A
{
        private:
                int y;
        public:
                B(int p, int q) : A(p)
                {
                        y = q;
                }
                void display()
                {
                        cout<<"x = "<<x<<endl;
                        cout<<"y = "<<y;
                }
};
int main()
{
        B b(10, 20);
        b.display();
        return 0;
}
```

Output:

```
x = 10
y = 20
```

9(c) C++ program to illustrate runtime polymorphism

```
#include <iostream>
using namespace std;
class Animal
{
    public:
        virtual void sound() = 0;
        virtual void move() = 0;
};
class Dog : public Animal
{
    public:
        void sound()
        {
            cout<<"Bow wow wow"<<endl;
        }
        void move()
        {
            cout<<"Dog is moving"<<endl;
        }
};
class Cat : public Animal
{
    public:
        void sound()
        {
            cout<<"Meow meow meow"<<endl;
        }
        void move()
        {
            cout<<"Cat is moving"<<endl;
        }
};
int main()
{
    Animal *a;
    a = new Dog();
    a->sound(); //run-time polymorphism
    a = new Cat();
    a->sound(); //run-time polymorphism
    return 0;
}
```

Output:

```
Bow wow wow
Meow meow meow
```

10(a) C++ program to illustrate template class

```
#include <iostream>
using namespace std;
template<class T>
class Swapper
{
	private:
		T x;
		T y;
	public:
		Swapper(T x, T y)
		{
			this->x = x;
			this->y = y;
		}
		void swap()
		{
			T temp = x;
			x = y;
			y = temp;
		}
		void display()
		{
			cout<<"After swap x = "<<x<<", y = "<<y<<endl;
		}
};
int main()
{
	Swapper<int> s1(2, 4);
	s1.swap();
	s1.display();
	Swapper<double> s2(4.2, 6.9);
	s2.swap();
	s2.display();
	return 0;
}
```

Output:

```
After swap x = 4, y = 2
After swap x = 6.9, y = 4.2
```

10 (b) C++ program to illustrate template class with multiple parameters.

```
#include <iostream>
using namespace std;
template<class T1, class T2>
class Adder
{
        private:
                T1 x;
                T2 y;
        public:
                Adder(T1 x, T2 y)
                {
                        this->x = x;
                        this->y = y;
                }
                void add()
                {
                        cout<<"Sum is: "<<(x+y)<<endl;
                }
};
int main()
{
        Adder<int,int> a1(3, 5);
        a1.add();
        Adder<int,double> a2(2, 5.3);
        a2.add();
        return 0;
}
```

Output:

```
Sum is: 8
Sum is: 7.3
```

10(C) C++ program to illustrate member function template

```
#include <iostream>
using namespace std;
class Adder
{
        public:
                template<class T1, class T2>
                void add(T1 x, T2 y)
                {
                        cout<<"Sum is: "<<(x+y)<<endl;
                }
};
int main()
{
        Adder a1;
        a1.add(4, 2);
        Adder a2;
        a2.add(3, 4.7);
        return 0;
}
```

Output:

```
Sum is: 6
Sum is: 7.7
```

11(a) C++ program for handling divide by zero exception

```
#include <iostream>
using namespace std;
int main()
{
        int a, b;
        cout<<"Enter two integer values: ";
        cin>>a>>b;
        try
        {
                if(b == 0)
                {
                        throw b;
                }
                else
                {
                        cout<<(a/b);
                }
        }
        catch(int)
        {
                cout<<"Second value cannot be zero";
        }
        return 0;
}
```

Output:

```
Enter two integer values: 4 0
Second value cannot be zero
```

11(b) C++ program to rethrow an exception

```
#include <iostream>

using namespace std;
int main()
{
	try
	{
		int a, b;
		cout<<"Enter two integer values: ";
		cin>>a>>b;
		try
		{
			if(b == 0)
			{
				throw b;
			}
			else
			{
				cout<<(a/b);
			}
		}
		catch(...)
		{
			throw; //rethrowing the exception
		}
	}
	catch(int)
	{
		cout<<"Second value cannot be zero";
	}
	return 0;
}
```