1. 15.5:
2. The printout is:

**Parent’s no-arg constructor is invoked**

1. The default constructor of Child tries to invoke the default constructor of Parent, but the default constructor in Parent is not defined.
2. 15.8:

No. Because the destructor in derived class will automatically invoke the destructor in base class. There is no need to implement the destructor in the derived class.

1. 15.12:

**invoke f from Parent**

**invoke f from Parent**

**invoke f from Child**

**invoke f from Parent**

**invoke f from Parent**

**invoke f from Parent**

**invoke f from Child**

**invoke f from Parent**

**invoke f from Parent**

**invoke f from Parent**

**invoke f from Child**

**invoke f from Child**

1. 15.14:

NO. To enable dynamic binding for a function, the variable also needs to be passed by reference or passed as a pointer in the virtual function.

1. 15.20:

Yes, it complies and runs. The printout is:

**invoke f from B**

**invoke m from D**

1. 16.6:

10:

**Enter a temperature:** 10

**Start of try block ...**

**End of try bolck ...**

**Continue ...**

60:

**Enter a temperature:** 60

**Start of try block ...**

**End of try bolck ...**

**Continue ...**

120:

**Enter a temperature:** 120

**Start of try block ...**

**Exceptional temperature**

**It is too hot**

**Continue ...**

1. 16.8:

You cannot throw multiple exceptions in one throw statement.

You can have multiple catch blocks in a try-catch block.

1. 16.9:

* statement3 will not be executed.
* If the exception is not caught, statement4 will not be executed.
* If the exception is caught in the catch block, statement4 will be executed.

1. 12.8:
2. swap(v1, d2); is wrong because v1 and d2 are variables of different types.
3. swap(1, 2); is wrong because 1 and 2 are values, not variables. To invoke this swap function, you have to use variables.
4. 12.11:

The code in this problem is wrong because the following codes in red is missing:

**template<typename T>**

**class Foo**

**{**

**public:**

**Foo();**

**T f1(T value);**

**T f2();**

**};**

**template<typename T>**

**Foo<T>::Foo()**

**{**

**...**

**}**

**template<typename T>**

**T Stack<T>::f1(T value)**

**{**

**...**

**}**

**template<typename T>**

**T Stack<T>::f2()**

**{**

**...**

**};**