CECS 282 EXAMINATION 2
Fall 2019 70 points

Part I – [15 points] True/False and multiple choice . 1 point for each correct answer.
Indicate whether the sentence or statement is T (true) or F(false) for questions 1 to 8. Write A for true and B for false.

__T__ 1. C++ allows you to pass an object of a derived class to a formal parameter of the base class type.

__T__ 2. In compile-time binding, the necessary code to call a specific function is generated by the compiler.

__F__ 3. The binding of virtual functions occurs at compile time.

__T__ 4. An object of the base class type cannot be passed to a formal parameter of the derived class type.

5. _____ members of a base class are never accessible to a derived class.
   a. Public
   b. Private
   c. Protected
   d. a, b, and c
   e. None of these

   ANS: B

6. The _____ constructor is called before the _____ constructor.
   a. base, derived
   b. derived, base
   c. public, private
   d. private, public
   e. None of these

   ANS: A

7. The _____ destructor is called before the _____ destructor.
   a. base, derived
   b. derived, base
   c. public, private
   d. private, public
   e. None of these

   ANS: B

8. When member functions behave differently, depending upon which object performed the call, this is an example of _____.
   a. chaos theory
   b. virtual insubordination
   c. polymorphism
   d. encapsulation
   e. None of these

ANS: C

9. Virtual function is _____ class function which expected to be redefined in _____ class, so that when reference is made to derived class object using pointer then we can call virtual function to execute _____ class definition version.

a) Base, derived, derived
b) Derived, Derived, Derived
c) Base, derived, base
d) Base, base, derived

Answer: a

Explanation: The functions which may give rise to ambiguity due to inheritance, can be declared virtual. So that whenever derived class object is referred using pointer or reference to the base class methods, we can still call the derived class methods using virtual function. Hence this differentiates those methods from each other.

10. Virtual functions are mainly used to achieve _____

a) Compile time polymorphism
b) Interpreter polymorphism
c) Runtime polymorphism
d) Functions code polymorphism

Answer: c

Explanation: It is used to achieve runtime polymorphism. The functions which are inherited and overridden, so at runtime the correct function is executed. The correct function call is made from the intended class.

11. The virtual functions must be declared and defined in _____ class and overridden in _____ class.

a) Base, base
b) Derived, derived
c) Derived, base
d) Base, derived

Answer: d

Explanation: The virtual functions must be declared and defined in base class. The functions can be redefined in derived class. If redefined in derived class then it overrides the base class function definition.

12. Downcasting is _____

a) Always safe
b) Never safe
c) Safe sometimes
d) Safe, depending on code

Answer: b

Explanation: The downcasting concept is made for exception cases. When there is a need to represent an entity in the form which is not suitable for it. Representing a base type in derived type is not right but can be done for special cases.

13. What should be used for safe downcast?

a) Static cast
b) Dynamic cast
c) Manual cast
d) Implicit cast

Answer: b

Explanation: The dynamic cast can be done using the operator dynamic_cast. This converts one type to another type in safe way.

14. If two classes are defined “Parent” and “Child” then which is the correct type upcast syntax in C++?

a) Parent *p=child;
b) Parent *p=*child;
c) Parent *p=&child;
d) Parent *p=Child();

Answer: c

Explanation: The syntax must contain the base class name first. So that the parent class object pointer can be declared. Then the object is assigned with the derived class object with & symbol. & symbol is added to get the address of derived class object.

15. Which casting among the following is allowed for the code given below?

```
class A
{
public :int a;
}
class B:public A
{
 int b;
}
main()
{
B b=new A();  //casting 1
A a=new B();  //casting 2
}
```

a) Casting 1
b) Casting 2
c) casting 1 and casting 2
d) casting 1 nor casting 2

Answer: b

Explanation: The casting 2 is correct. The objects casting must be done from derived class object to a parent class object. That is, the object of the superclass can be made an object of subclass only. Vice versa is not possible.

Part II – Written coding and coding analysis

1. [14 points] Book is a class with cover and pageLength defined as protected data members. Textbook is a is derived class from Book with the *subjectMatter* as a private data member. Each class has the member function *describe* along with its constructor.

Note:
Enumerated types are types that are defined with a set of custom identifiers, known as enumerators, as possible values.
For example, a new type of variable called colors_t could be defined to store colors with the following declaration: enum colors_t {black, blue, green, cyan, red, purple, yellow, white};
For example, once the colors_t enumerated type is declared, the following expressions will be valid:

```
colors_t mycolor;
```

mycolor = blue;
if (mycolor == green) mycolor = red;

Below is the class Book.

```
 enum CoverType {Hardcover, Softcover};
class Book

{ protected:
   CoverType cover;
   int pageLength;
public:
       Book(CoverType ct, int pglen): cover(ct), pageLength(pglen)
       { }
       virtual void describe()
       {      cout<<"A "<< pageLength<< " pages ";
           if (cover == Hardcover)
                    cout<<"hard covered book."<<endl;
              else
                   cout <<" soft covered book."<<endl;
       }
};
```

a. [8 points] Implement the class Textbook from the class book as public inheritance. You can define the constructor and functions in the class. The method describe displays the Book output defined in the method describe in the class Book, and the subject matter.

```
class Textbook: public Book
{ private:
   string subjectMatter;
 public:
          Textbook(CoverType ct, int pglen, string subject) : Book(ct, pglen)
          {subjectMatter = subject;}
           void describe()
          {Book::describe();
           cout<<subjectMatter<<endl;
          }
};
```

b. [6 points] Write a main function to display the following output:

A 450 pages soft covered book.
A 760 pages hard covered book.
Python Programming

Note: Python Programming is a subject matter.

```
int main()
{ Book myDictionary(Softcover, 450);
  Textbook courseBook(Hardcover, 760, "Java Programming");
  myDictionary.describe();
  courseBook.describe();
 }
```

2. [3 points] What is the output from the executing of the program below?

```
 #include <iostream>
using namespace std;

class Base
{
public:
	Base() {
		cout << "Base Constructor \n" ;
	}
	~Base() {
		cout << "Base Destructor \n" ;
	}
};


class Derived : public Base
{
public:
	Derived(string s):str(s) {
		cout << "Derived Constructor \n" ;
	}

	~Derived() {
		cout << "Derived Destructor \n" ;
	}
private:
	string str;
};

int main()
{
	Base *pB = new Derived("derived");
	delete pB;
}
```

Base Constructor
Derived Constructor
Base Destructor

3. [3 points] What do you need to modify the code in the program so the output will be displayed as it's shown below? You are not allowed to modify the main function, or the bodies of constructors or destructors.

Output of the program

Base Constructor
Derived Constructor
Derived Destructor
Base Destructor

```
#include <iostream>
using namespace std;

class Base
{
public:
	Base() {
		cout << "Base Constructor \n" ;
	}
	virtual ~Base() {
		cout << "Base Destructor \n" ;
	}
};

class Derived : public Base
{
public:
	Derived(string s):str(s) {
		cout << "Derived Constructor \n" ;
	}
	~Derived() {
		cout << "Derived Destructor \n" ;
	}
private:
	string str;
};


int main()
{
	Base *pB = new Derived("derived");
	delete pB;
}
```

4. [4 points] What is the output from the executing of the program below?

```
#include<iostream>
using namespace std;
class A
{ private:
   int a;
public:
   A()
   {}
       A(int n)
   {a = n;}
   void put()
   {cout<<a<<endl;}
};
class B : public A
{private:
    int b;


 public:
        B()
        {}
       B(int n, int m): A(n)
       {b=m;}

       void put()
       { A::put();
         cout<<b<<endl;
       }
};

int main()
{B b(5,2); //a <- 5, b<- 2
 A a(4); //a <- 4
 a = b;
 a.put();/
}
```

5

5. [3 points] What is the output from the executing of the program below? If the program has any errors, please indicate them.

```
#include<iostream>
using namespace std;
class A
{ private:
   int a;
public:
   A()
   {}
       A(int n)
   {a = n;}
   void put()
```

```
   {cout<<a<<endl;}
};
class B : public A
{private:
    int b;
 public:
        B()
        {}
       B(int n, int m): A(n)
       {b=m;}
       void put()
       { A::put();
         cout<<b<<endl;
       }
};

int main()
{A a(5);
 B b;
 b = a; //ERROR
 b.put();
}
```

6. [3 points] Given the program below.

```
#include<iostream>
using namespace std;

class A
{
public:
   int a;
};
class B : virtual public A
{ public:
        int b;
};
class C : virtual public A
{ public:
        int c;
};
class D : public B, public C
{ public:
        int d;
};
int main()
{
       D objD;
       objD.b = 2;
       objD.c = 3;
       objD.d = 4;
       objD.a = 6;
       cout<<objD.b +objD.c + objD.d+objD.a<<endl;
}
```

Will the program produce the following output? **YES**
15

If your answer is no, then describe the problem.

7. [7 points] When dealing with polymorphism, you'll often encounter cases where you have a pointer to a base class, but you want to access some information that exists only in a derived class.

Consider the following (slightly contrived) program:

```
#include <iostream>
#include <string>

class Base
{
protected:
int m_value;

public:
Base(int value)
: m_value(value)
{
}
virtual ~Base() {}
};

class Derived : public Base
{
protected:
std::string m_name;

public:
Derived(int value, std::string name)
: Base(value), m_name(name)
{
}

const std::string& getName() { return m_name; }
};
```

```
Base* getObject(bool bReturnDerived)
{
if (bReturnDerived)
return new Derived(1, "Apple");
else
return new Base(2);
}

int main()
{
Base *b = getObject(true);

// how do we print the Derived object's name here, having only a Base pointer?

delete b;

return 0;
}
```

Derived *d = dynamic_cast<Derived*>(b); // use dynamic cast to convert Base pointer into Derived pointer

if (d) // make sure d is non-null
std::cout << "The name of the Derived is: " << d->getName() << '\n';

8. [8 points] Given the classes below:

```
class A
{
public:
   virtual void display() =0;
   virtual int getValue()
   {
        return 0;
   }

};

class B : public A
```

```
{
public:
       int y;
      B(int b): A() {
          y = b;
      }
      void display()
      {cout <<"B"<<endl;
      }
   int getValue()
   {
         return y;
   }
};

class C : public B
{
public:
       int z;
      C(int a, int b): B(a) {
          z = b;
      }
      ~C() {
           cout << "Derived Destructor \n" ;
      }
      int getValue()
   {
         return z;
   }
      void display()
      {cout <<"C"<<endl;
      }
};
```

a. [4 points] What is the output from the running of the program below?

```
int main()
{

      A *ptr[4];

      ptr[0] = new B(3); //y <- 3
      ptr[1] = new B(4); //y <- 4
      ptr[2] = new C(3,7); //y <- 3, z <- 7
      ptr[3] = new C(2,5); //y <- 2, z <-5
```

```
      int sum=0;
      for (int i = 0; i< 4; i++)
            if( typeid(*ptr[i]) == typeid(C))
              sum += ptr[i]->getValue();
      cout<<sum<<endl;/// 7 + 5
      //delete ptr [];
}
```

Output:

12

b. [4 points] What is the output from the executing of the program below?

```
int main()
{

      A *ptr[4];

      ptr[0] = new B(3); //y <- 3
      ptr[1] = new B(4); //y <- 4
      ptr[2] = new C(3,7); //y <- 3, z <- 7
      ptr[3] = new C(2,5); //y <- 2, z <-5
      int sum=0;
      for (int i = 0; i< 4; i++)
            if(dynamic_cast<B *>(ptr[i]))
                        sum += ptr[i]->getValue();
      cout<<sum<<endl; //2 + 3 + 7 + 5

}
```

Output

19