

**FINAL EXAM SOLUTION - BESE15**  
**OBJECT ORIENTED PROGRAMMING PARADIGM**

**Question 1:**

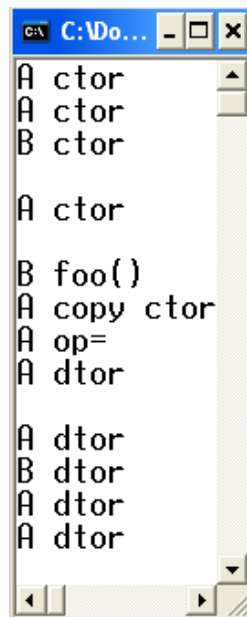
Please see the scanned pages in the attached zip file.

**Question 2:**

**(a)**

The derived class over-rides the base class function foo(). The derived class version of foo() takes no argument. So when an object of derived class calls foo() with one argument, it does not compile.

**(b) output:**



```
C:\Do...
A ctor
A ctor
B ctor

A ctor

B foo()
A copy ctor
A op=
A dtor

A dtor
B dtor
A dtor
A dtor
```

**(c) output:**

2  
3  
8  
6  
15

**(d) Output**

Answer is: 4

**Question 3:**

The function `get_i()` returns a pointer to a local variable which is destroyed as the program returns from the function back to the main program. The problem can be solved by using dynamic memory within the body of the function `get_i()` as given below:

```
int* get_i()
{
    int *i = new int;
    *i = 2;
    return i;
}
```

**Question 4:**

```
void main()
```

```
{
```

```
    A objA; //cannot create object of an abstract class A which is  
    abstract due to the pure virtual function output()
```

```
    B objB; //B is also abstract since it is derived from A and it does  
    not over-ride the pure virtual function output(), and so an object  
    of class B cannot be created
```

```
    C objC;
```

```
    D objD;
```

```
    objC.setX(2); //protected member function of class C not accessible  
    in main
```

```
    cout<<objC.getX(); //protected member function of class C not  
    accessible in main
```

```
    objD.setX(1); //class D inherits from class C so setX() remains  
    protected and therefore is not accessible in main
```

```
    objD.f(3);
```

```
}
```

PTO

Question 5:

```
class customer
{
    char* name;
    double mins;
    double rate;
public:
    customer():name(0),mins(0.0),rate(0.0){}
    customer(char* c_name,double c_mins,double c_rate):
                                                mins(c_mins),rate(c_rate)
    {
        name = new char [strlen(c_name)+1];
        strcpy(name,c_name);
    }
    ~customer()
    {
        delete [] name;
    }
    virtual double compute_bill()
    {
        return mins*rate;
    }
    char* getName()
    {
        return name;
    }
};

class pre:public customer
{
public:
    pre(){}
    pre(char* c_name,double c_mins,double c_rate):
                                                customer(c_name,c_mins,c_rate){}
};

class post:public customer
{
    double lrent;
public:
    post():lrent(0.0){}
    post(char* c_name,double c_mins,double c_rate, double rent):
                                                customer(c_name,c_mins,c_rate),lrent(rent){}
    double compute_bill()
    {
        double a = customer::compute_bill();
        return a+lrent;
    }
};
```

```

    }
};

void main()
{
    customer* array[3];
    array[0] = new pre("OJ",400.0,2.5);
    array[1] = new post("SK",510.0,1.5,150);
    array[2] = new post("RS",544.0,1.25,250);
    for(int i=0 ; i<3 ; i++)
    {
        cout<<"Customer"<<array[i]->getName()<<"owes Rs."
            <<array[i]->compute_bill()<<endl;
    }
    for(int i=0 ; i<3 ; i++)
        delete array[i];
    getch();
}

```

PTO

Question 6:

```
int* copy(int a[],int a_size,int b[], int b_size)
{
    int *p = new int[a_size+b_size];
    int i=0;
    for(;i<a_size;i++)
        p[i] = a[i];
    for(int j=i; j<(a_size+b_size);j++)
        p[j] = b[j-i];
    for(int k=0;k<5;k++)
        cout<<p[k]<<'\\t'<<endl;
    return p;
}

void main()
{
    int a[2] = {1,2};
    int b[3] = {3 , 4, 5};
    int* p = copy(a,2,b,3);
    for(int i=0;i<5;i++)
        cout<<p[i]<<'\\t';
    getch();
}
```

PTO

Question 7:

```
class vehicle
{
    char* reg_no;
    long mileage;
public:
    vehicle(char*,long);
    vehicle(const vehicle&);
    virtual void show();
};
//end of vehicle.h

//Car.h
class car : public vehicle
{
    string owner;
public:
    car(char*, long , string );
    car(const car&);
    void show();
};
//end of car.h

//vehicle.cpp
vehicle::vehicle(char* no = "no number" ,long miles =0):
                                                    mileage(miles)
{
    reg_no = new char[strlen(no)+1];
    strcpy(reg_no,no);
}

vehicle::vehicle(const vehicle& v)
{
    reg_no = new char[strlen(v.reg_no)+1];
    strcpy(reg_no,v.reg_no);
    mileage = v.mileage;
}
void vehicle::show()
{
    cout<<"registration no is:"<<reg_no<<endl;
    cout<<"mileage is:"<<mileage<<endl;
}
//end of vehicle.cpp

//car.cpp
car::car(char* no = "no car number" ,long miles = 0, string
car_owner = 0):vehicle(no,miles),owner(car_owner) {}
```

```

car::car(const car& c) : vehicle(c)
{
    owner = c.owner;
}
void car::show()
{
    vehicle::show();
    cout<<"owner is:"<<owner<<endl;
}
//end of car.cpp

//global functions
vehicle* allocate(int i)
{
    vehicle* v;
    if(i==0)
    {
        v = new vehicle("IDH-1234",10);
    }
    else
        v = new car("SV-2341",20000,"ABC");
    return v;
}

void main()
{
    vehicle* v = allocate(1);
    v->show();
    getch();
}

```

**Part 5:** This can be done in the following way:

```

    vehicle& v = *(allocate(1));
    v.show();

```

**Part 6:** The Vehicle class can be made abstract by making the show() function in the Vehicle class pure virtual by using the following statement

```

    virtual void show() = 0;

```