Answer-book

i) Univ. Roll No. (in figures) 2819105 (inwords) _____.
ii) Name of the student Kashish Jain iii) Class/Semester 3rd sem, Sec B.
iv) Name of the Paper: OOPS v) Code of Paper: PC - CS 203A.
vi) Total No. of Pages written by candidate: 16 vii) Sign. of the Student Kashish
viii) Date of Exam :- 3rd Nov. 2020

Sessional

Ans: Object oriented programming is a type of programming which uses objects and classes its functioning. Object oriented programming based on real life entities like Inheritance, Polymorphism, Encapsulation, data hiding etc. It aims at binding together data and function work on these data sets into single entity to restrict their usage.
Basic concepts of object oriented programming are:-

1.) Classes
2.) Objects
3.) Encapsulation
4.) Inheritance
5.) polymorphism
6.) Abstraction

Classes - A class defines properties and behaviour for objects represented by abstraction. Class is a data type

Roll No. (in figures) 2819105 Code of Paper. PC - CS 203A

that has its own members ie. data members and member functions. It is a blueprint for an object in object oriented programming language. It is basic building block of object oriented programming in C++. Members of class are accessed in programming language by creating an instance of class.

Objects - objects are basic run time entities in an object oriented system they may represent any person, place, or any item that program has to handle. Objects are instance of classes, which holds data variable declared in class.

Syntax: class_name object name;

Syntax of class - class class_name {
data_type data_name;
return_type method_name
(parameters);
}

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC-CS 203A

Encapsulation - Wrapping up of data and information under single unit. In object oriented programming, Encapsulation is defined as binding together data & functions that manipulates them. Encapsulation also lead to data abstraction or hiding.

Inheritance - capability of class to derive properties and characteristics from another class. Inheritance is one of the most important feature in object oriented programming.

Sub - class - class that inherits properties from another class called derived class.

super class - class whose properties are inherited by sub - class called base class.

Syntax -

```
class subclass-name : access-mode
                      base-class
{
    // body of subclass
};
```

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC-CS 203A

polymorphism - It is ability of message displayed in more than one form. A real life example of polymorphism, a person at same time can have different characteristics like a man at same time is a father, a husband, an employee. So same person possess different behaviour in different situations. It includes operator overloading and function overloading.

Abstraction - Data abstraction or data hiding is hiding data and showing any relevant data to final user. It is an important part of object oriented programming.

Ans 1 (b) Static data member - static data members are class members that are declared using static keyword. There is only one copy of static data member in class, even if there are many class objects. This is because all objects share static data member. static data

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC-CS 203A

member always initialized to zero when first class object is created.

Syntax of static data member

static data-type data-member-name;

In above, static keyword is used. Data-type is C++ data type such as int, float, etc. The data-member-name is name provided to data memeber

Example:

```
# include <iostream>
# include <string.h>

using namespace std;
 class student {
     private :
 int roll no.;
  char name[10];
  int marks;
     public :
static int object count;
```

Rashmi
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC-CS 203 A

```
student ( ) {
      object count ++;
  }

void getdata ( ) {
  cout << " Enter roll no. : " << endl;
      cin >> roll no;
  cout << " Enter name : " << endl;
      cin >> name;
  cout << " Enter marks : " << endl;
      cin >>  marks;
  }

void putdata ( ) {
  cout << " Roll no. = " << rollno. << endl;
  cout << " Name = " << name << endl;
  Cout << " Marks = " << Marks << endl;
      cout << endl;
    }
};
 int student :: object count = 0;
    int main (void) {
    student S1;
  S1. getdata ( );
  S1. putdata ( );
student S2;
```

Kashish

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC-CS203A

```
    S2. getdata();
    S2. putdata();
Student S3;

    S3. getdata();
    S3. putdata();
cout << "Total objects created = " << student
                                  :: object count << endl

    return 0;
}
```

OUTPUT:

```
Enter roll no. : 1
Enter name : Mark
Enter marks : 78
 Roll no. = 1
  Name = Mark
  Marks = 78

Enter roll no. : 2
Enter name : Nancy
 Enter marks : 55
   Roll no. = 2
   Name = Nancy
    Marks = 55
```

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. _____

Enter roll no. : 3
EnterName : Susan
Enter marks = 90
roll no. = 3
Name : Susan
Marks = 90
Total objects created = 3.

In above program, class student has 3 data members denoting student roll no, name & Marks. object count data member is static data member that contains no. of objects created of class student. Student() is a constructor that increments object count each time a new class object is created.

Ans 1 (c) Friend function - Friend function is defined outside that class scope but it has right to access all private and protected members of class. We declare friend function using the friend keyword inside body of class.

```
class class name{
.....
friend return Type function name();
.....
}
```

Kashish

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

Merits of friend function

1.) While defining friend function, there is no need to use scope resolution operator as friend keyword.
2.) Friend can be defined anywhere in program similiar to normal function.
3.) Function may be friend of more than one class.

Demerits of friend function

1.) Friend function is declared in class, but it is not member function of class. To access private data members of class it is necessary to create objects.
2.) When function is friend of more than one class than forward declaration of class is needed.

Ans 1 (d) C++ constructor call order will be from top to down that is from base class to derived class in C++. while destructor call order will be in reverse order ie it will be from derived class to base class.

Kashish

Sign. of the Candidate

Roll No. (in figures) 2819105. Code of Paper.

Here is example of constructor destructor call order for multi-level inheritance, in which device class is base class & Mobile class is derived class

When we create object Android class, order of invocation of constructor & destructor will be as below:

Constructor : device
Constructor : Mobile
Constructor : Android
Destructor : Android
Destructor : mobile
Destructor : device

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

// base class
class device {

public:
device() { cout << " constructor : device \n" ;}
~ device() { cout << " Destructor : device \n" ;}
};
```

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

```
// derived class
class mobile : public device{

  public :

  mobile () { cout << " Constructor : Mobile \n";}
~ mobile () { cout << " Destructor : Mobile \n" ;}
  };

// derived class
class android : public mobile {

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

int main()
{
 android- android ; // create object that
                       will regd call constructor

     return 0;
 }
```

Kashvi

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

Ans 3: @. Private access specifier - It is used while creating class, then public and protected data members of base class become private member of derived class and private member of base class remains private. In this case members of base class can be used only within derived class and cannot be accessed through object of derived class whereas they can be accessed by creating function in derived class.

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

class base
  {
    private:
  int x;

  protected:
    int y;

  public:
    int z;
```

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

```
base () // constructor to initialize data
                members
    {
        x = 1;
        y = 2;
        z = 3;
    }
};

class derive : private base
    {
// y and z becomes private members
    public:
    void showdata ()
    {
cout << " x is not accessible " << endl;
cout << " value of y is " << y << endl;
cout << " " value of z is " << z << endl;
    }
};

int main()
{
derive a; // object of derived class
   a. showdata ();
// a.x = 1; not valid : private member can't
                be accessed outside class
```

Output

x is not accessible
value of y is 2
value of z is 3.

Sign. of the Candidate

Kashyap

Roll No. (in figures) 2819105 Code of Paper.

```
// a.y = 2; not valid ; y is now private
                member of derive class
// a.z = 3; not valid ; z also now private
                member of derived class.

}
```

Protected access specifier - It is similar to private access modifier in sense that it can't be accessed outside its' class unless with help of friend class.

```
# include < bits / stdc++ .h >
using namespace std;
class Parent {
  Protected :
  int id - protected;
};
class child : public parent {
 public :
void set id ( int id) {
 id - protected = id;
}
void display id () {
   cout << " id is << id - protected << /n";
      }
};
```

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

```
int main() {
cried oject ();
 obj /. set id (81);
obj /. display id ();
  return 0;
  }
```

Public access specifier - All class members declared under public specifier will be available to everyone. Data members & member fns declared as public can be accessed by other classes & function too.

```
# include <iostream>
 using namespace std;
   class circle {
      public :
  double radius;
double  compute - area. () {
 return 3.14 * radius * radius;
   }
};

int main() {
  circle obj;
 obj . radius = 5.5;
```

Kashish
Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper.

```
cout << " Radius : " << obj.radius << endl;
cout << " Area : " << obj.compute_area();
return 0;
}
```

Ans 3 (b) In multiple inheritance, there may be possibility that class may inherit member fns with same name from 2 or more base classes & derived class may not have functions with same name as those of its base class. If object of derived class need to access one of the same named fn. of base classes then it result in ambiguity as it is not clear to compiler which base class member fn it should be invoked. Ambiguity simply means state when compiler is confused.

Syntax

```
class a {
private :
------
------
public :
void abc () { }
```

Kashish

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. _____

```
.....
.....
};

class b{
  private :
    ==
  public :
  void abc () { }
    ==
};
class c : public a, public b {
    private :
      ==
    public :
      ==
};
void main () {
 c. obj ;
obj. abc () ; // error
}
```

In above example, 2 base classes a & b have fn of same name abc() which are inherited to derived class c. when obj

Kashish

Sign. of the Candidate

Roll No. (in figures) 2819105 Code of Paper. PC - CS 203A.

of class is created & call fn abc() then compiler is confused which base's class fn is called by compiler.

Sol^{n} of ambiguity - Ambiguity can be resolved by using scope ~~reso~~ resolution operator to specify class in which member function lies as-

```
obj.a :: abc();
```

statement invoke function name abc() which lies in base class a.

Kashish

Sign. of the Candidate