Date __/__/__ Name: Rahul Kumar Mehta

Choose the correct option

1. d. Stacks

2. c. Compiler error in line "Derived *dp = new Base;"

3. a. Inaccessible.

4. a. The member of times destructor is called depends on number of object created.

5. a. True.

Short answer type question.

1. Syntax to use new operator: To allocate memory of any data type the syntax is

$\boxed{\text{pointer-variable = new datatype;}}$

delete operator.
Since it is programmer's responsibility to deallocate dynamically allocated memory, programmers are provided delete operator.

Syntax.
```
// Release memory pointed by pointer-variable
delete pointer-variable;
```

Date ___/___/___

Example.

```
#include <iostream>
using namespace std;
int main()
{
// pointer initialization to null
      int *p = Null;
// Request memory for the variable.
   // using new operator.
     p=new(nothrow) int;
     if (!p)
cout<<"allocation of memory failed\n";
  else
  {
// Store value at allocated address
     *p = 29;
cout<<"value of p:"<<*p << endl;
  }
// Request block of memory
// Using new operator
     Float *r = new Float (75.25);
cout<<"Value of r:"<< *r << endl;
     // Request block of memory of size n
 int n=5;
 int *q = new(nothrow) int [n];
 if (!q)
cout<<"allocation of memory failed\n";
 else
```

Date ___/___/______

```
{
  for (int i = 0; i<n; i++)
    q[i] = i + 1;
cout<< "Value store in block of memory:";
    for (int i = 0; i<n; i++)
~~cout (int~~ ~~cout<<q<n;~~
cout<< q[i] << " ";
  }.
// Freed the allocated memory.
    delete p;
    delete r;
// Freed the block of allocated memory
      delete [] q;
      return 0;
}.
```

2. A constructor is a member function of a class which initializes objects of a class. In c++ constructor is automatically called when object (instance of class) create. It is special member function of the class. The main purpose of the class construction in c++ programming is to construct an object of the class.

<u>Default</u> constructor is the constructor which doesn't take any argument. It has no parameter.

Date ___ /___ /______

For example.
```
 clas cube.
{
  public;
  int side;
  Cube()
  {
     side=10;
  }
 };
  int main()
{
  cube c;
  cout<<c.side;
}.
```

Parameterized Contructors
There are the contructors with parameter using the constructor you can provide different value to data member of different object, by passing the appropriate value as argument.
For example.

```
Clas cube
{
  public;
  int side;
  Cube(int x)
```

Date ___/___/_____

```
{
  side=x;
}
 };
int main()
{
  cube c1(10);
  Cube c2 (20);
  Cube c3 (30);
  cout<<c1.side;
  cout<<c2.side;
  cout<<c3.side;
}.
```

Copy Constructors
There are Special type of constructor which take an object as argument and is used to copy value of data member of one object into other object. ~~We will study copy constructor in~~

```
#include<iostream>
using namespace std;
class A
{
  public;
  int x;
  A(int a)
{
```

```
x=a;
}
A(A &i)
{
x=i.x;
}
};
 int main()
{
 A a1(20);
 A a2(a1);
cout<<a2.x;
 return 0;
 }.
```

3.	Procedural Oriented Programming.	Object Oriented Programming.
*	In procedural programming, program is divided into small parts called function.	In object oriented programming program is divided into small parts called objects.
*	Procedural programming follows top down approach.	Object oriented programming follows botton up approach.
*	There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public protected etc.

Date ___/___/_____ Saathi

*	Adding new data and function is not easy	Adding new data and function is easy.
*	Procedural programming does not have any proper way for hiding data so it is less secure.	Object oriented programming provides data hiding so it is more secure.
*	Procedural programming is based on unreal world.	Object oriented programming is based on real world.
*	Ex:- C FORTRAN Pascal, Basic etc.	Ex:- C++, Java, Python, C# etc.

Long answer type question.

A Type of Polymorphism.
C++ support two type of polymorphism

Compile time polymorphism
You invoke the overloaded function by matching the number and type of argument The information is present during compile-time This means the c++ compiler will select the right function at compile time.

Compile time polymorphism is achieved through function overloading and operator overloading.

Date ___ /___ /______

Function Overloading.

Function overloading occurs when we have many function with similar names but different arguments. The arguments may differ in terms of number or type.

Example.

```
#include<iostream>
using namespace std;
void test(int i)
{
cout<<"The int i s"<<i<<endl;
}
void test(double f)
{
cout<<" The float is" <<f<<end;
}
void test(char const *ch)
{
cout<<" The char *is " <<ch<<endl;
}
int main()
{
    test(5);
    test (5.5);
    test ("Five");
    return 0;
```

Date ___/___/______

Operator Overloading
In operator overloading we define a new meaning for a c++ operator. It also change how the operator work. for example we can define the + operator to concatenate two string we know it as the addition operator for adding ~~numb~~ numerical value after our definition when placed between integers it will add them when place between strings it will concatenate them.
Example.

```
#include<iostream>
using namespace std;
Class Complex Num
{
 private:
     int real, over;
 public:
     complex Num(int rl=0, int ov=0)
 {
     real = rl;
     Over = ov;
 }
 complex Num operator +(complex Num const &obj)
 {
  complex Num result;
```

Date ___ /___ /______

```
result.real = real + obj.real;
result.over = over + obj.over;
return result;
}
Void print ()
{
cout<<real<<"+i"<<over<<endl;
}
};
int main()
{
    ComplexNum c1(10,2), c2(3,7);
    complex Num c3 = c1 + c2;
    c3. print();
}.
```

Runtime Polymorphism
This happens when an object's method is invoked/called during runtime rather than during compile time Runtime polymorphism is achieved through function overriding the function to be called/invoked is established during runtime

Function Overriding.
Function overriding occurs when a function of the base class is given a new definition in a derived class. At that time, we can say

Date ___ /___ /______

the bar function has been overriding.

Example:

```
#include<iostream>
using namespace std;
 class mammal
{
   public:
     void eat()
{
cout<<"Mammals eat...";
}
 };
 class cow: public mammal {
  public:
      void eat()
{
cout<<"cows eat grass...";
}
 };
  int main (void) {
      cow c = cow();
      c.eat();
      return 0;
}.
```

Date ___/___/______

Virtual Function
A virtual function is another way of implementing run-time polymorphism in c++. It is a special function defined in a based class and redefined in the derived class To declare a virtual ~~by~~. keyword. The keyword should precede the declaration of the function in the base class.

If virtual function class is inherited the virtual class redefines the virtual function to suit its needs for example.

```
#include<iostream>
using namespace std;
class classA
{
 public:
 Virtual void show()
{
cout<<"The show() function in base class
 invoked....."<<endl;
}
 };
Class classB : publicc classA
{
   public:
```

Date ___/___/_____

```
void show()
{
 t<<"The show function in derived class
Invoke d...";
 }
 };
 int main():
{
   class A *a;
   Class B  b;
   a = &b;
   a->show();
 }.
```

B.
```
//C++ program to sort an array
//with 0, 1, and 2 in a single pass
#include<bits/stdc++.h>
using namespace std;

// Function to sort the input array,
// the array is assumed
// to have values in {0,1,2}

void sort12(int a[], int arr_size)
{
  int lo = 0
  int hi = arr_size-1;
  int mid = 0;
```

Date ___ /___ /______

```
// Iterate till all the element
// are sorted
 while (mid<=hi)
 {
 Switch(a[mid])
 {
// If the element is 0
 case 0;
  Swap(a[lo++], a[mid++]);
  break;
// If the element is 1
 case 1:
   mid++;
   break;
// If the element is 2
  case 2;
   Swap(a[mid], a[hi--]);
   break;
 }
 }
 }
// Function to print array arr[]
void print Array (int arr[], int arr_size)
{
//Iterate and print every element
  for (int i=0; i<arr_size; i++)
    cout<<arr[i]<<" ";
 }
```

Date ___ / ___ / _____

```
// Drive code
 int main()
{
   int arr[] = {1,1,2,2,0,0,21,2};
 int n = sizeof(arr) / sizeof(arr[0]);
  sort012(arr, n);
cout<<"array after segregation";
  print array(arr, n);

   return 0;
}.
```

c.
```
Class Member
{
String name;
int age;
String number;
String address;
int salary;

public void printSalary() {
{
System.out.println(salary);
}
  }
Class Employee extends Member{
{
 String Specialization;
}
```

Saathi

Date ___ / ___ / ____

```
Class Manager extends member
{
 String department;
}
Class Any
{
 public static void main (String[]args
{
 Employee e=new employee();
  e.name = "xyz";
  e.age = 28;
  e. number = "986**** *07;
  e. address = "xyzxyz";
  e. salary = 1281;
  e. specialization = "xyzxyz";

 Manager m=new manager();
  m.name ="xyz"
  m. age = 28;
  m. number = "986*** **07";
  m. address = "xyzxyz";
  m. salary = 1281;
  m. specialization ="xyzxyz";
  }
  }.
```