Assignment -2.

=> Choose the correct option

1. Local variables are stored in an area called
Ans. Stack.

2. Choose the correct option 20.
Ans. compiler error. In line " Derived *dp = new Base ; "

3- When the Inheritance is private, the private methods in base class are ____ in the derived class.
Ans. Inaccessible.

4. Which of the following is true?
Ans. The number of times destructor is called depends on number of objects created.

5. State true or false
type conversion is auto matic whereas type casting is explicit.
Ans. True.

=> short answer type questions

Ans 1) (P) The new operator :-

The new operator requests for the memory allocation on heap. If the sufficient memory is available, it initialises the memory to the pointer variable and returns its address.

syntax :- pointer_variable = new datatype ;

code :-

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

int main()
{
    int *ptr1 = NULL;
    ptr1 = new int;
    float *ptr2 = new float (223.324);
    int *ptr3 = new int [20];
    *ptr1 = 28;
    cout << " value of pointer variable 1 " << *ptr1 << endl;
    cout << " value of pointer variable 2 " << *ptr2 << endl;
    if (!ptr3)
```

```
cout<< " allocation of memory failed ";
else
{
    for (int P= 10 ;  P< 15  ;  i++)
    ptr3[i] = i+1 ;
    cout << " value of store in block memory ";
    for (int P= 10;  P< 15 ;  i++)
    cout << ptr3[i] << " ";
}
return 0;
}
```

(iii) The delete operator :-

The delete operator is used to de allocate the memory. User has privilage is to deallocate the created pointer variable by this deleted operator.

code :-

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

int main ()
{
    int * ptr1 = NULL;
    ptr1 = new int;
    float * ptr2 = new float (299.121);
```

```
int *ptr3 = new int (28);
*ptr1 = 28;
cout<< " value of pointer variable 1 : " << *ptr1 << endl;
cout<< " value of pointer variable 2 : " << *ptr 2 << end;
if ( ! ptr 3)
cout << " allocation of memory failed " ;
else
{
    for (int i= 10;  i< 15  ; i++)
    ptr 3[i]  = i+1 ;
    cout << " value of store in block of memory " ;
    for (int i=10 ;  i<15;  i++)
    cout << ptr3[i] << " " ;
}
delete ptr1;
delete ptr2;
delete [] ptr3;
return 0;
}
```

Ans 2) Constructors :- A constructor is a special type of member function of a class which initialises object of a class. In C++ constructor is automatically called when object is created.

Types of constructors :-

1. Default constructors & default constructors is the constructor which dosent take any argument. It has no argument.

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

class construct
{
  public:
    int a, b;
    construct()
    {
      a = 10;
      b = 20;
    }
};

int main()
{
  construct c;
  cout << "a: " << c.a << endl << "b: " << c.b;
  return 1;
}
```

2. Parameterised constructor :- It is possible to pass arguments to constructors. These arguments help initialise an object when it is created.

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

class point
{
    private:
    int x, y;
    public:
    point (int x1, int y1)
    {
        x = x1;
        y = y1;
    }
    int getX ()
    {   return x;
    }
    int getY ()
    {   return y;
    }
};

int main ()
{
```

```
    point p1(10,15);
    cout<< "p1.x =" << p1.getx()<<", p1.y =" << p1.gety();

    return 0;
}
```

3. Copy constructor & A copy constructor is a member function which initialises an object using another object of the same class.

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

class point
{
  private:
    double x, y;

  public:
    point (double px, double py)
    {
        x=px, y=py;
    }
};

int main (void)
{
```

```
point a (10);
point b = point (5,6);
}
```

Ans 3) Procedural Programming :-
It can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure. Procedures also called as routines or simply the steps need to be carried out.

Object oriented programming :-
It can be defined as a programming model which is based upon the concept of objects. Objects contain data in the form of attributes and code in the form of methods. In object oriented programming, computer programs are designed using the concept of objects that interact with real world.

Differences b/w procedural programming and object oriented programming.

Procedural programming	object-oriented programming
→ Here, program is divided into small parts called functions.	Here, program is divided into small parts called objects.
→ Procedural programming follows top-down approach.	It follows bottom-up approach.
→ There is no access specifier here.	Here we have access-specifiers like private, public, protected.
→ Adding the new data is not easy.	Adding new data is very easy.
→ It does not have way to hide data thus its less secure.	It provided data hiding thus its more secure.
→ Here overloading is not there.	Here we have the function of overloading.
→ examples → C, FORTRAN, Basic. etc.	examples → C++, Java, python, C# etc

Long Answer Type Question

Ans A) Poly morphism can be defined as the ability to take more than one form. It is mainly divided into two types.

(i) Compile Time poly morphism.
(ii) Run Time poly morphism.

(i) Compile time polymorphism or this type is polymorp can be achieved by function overloading or operator overloading.

example :-

function overloading -

```
#include <bits/stdc++.h>
using namespace std;
class geeksomen
{
    public:
    void func(int x)
    {
        cout<< " value of x is " << x <<endl;
    }
```

```
void func (double z)
{
    cout << " value of x is " << x << endl;
}

void func (int x, int y)
{
    cout << " value of x and y is " << x << " " << y << endl;
}
};

int main ()
{
    geeks omen obj1;
    obj1. func (7);
    obj1. func (9.132);
    obj 1. func (85, 64);
    return 0;
}
```

) Runtime polymorphism or this type of poly morphism is achieved by function overriding

Function overriding occurs when derived class has a definition for one of the member functions of base class.

```
#include <bits/stdc++.h>
using namespace std;

class base
{   public:
    virtual void print()
    {
        cout<<" print base class"<<endl;
    }
    void show()
    {  cout<<" show base class"<<endl;
    }
};

class derived : public base
{
   public:
    void print()
    {  cout<<" print derived class"<<endl;
    }
    void show()
    {  cout<<" show derived class"<<endl;
    }
};

int main()
{
```

```
base *bptr;
derived d;
bptr = &d;

bptr -> print();
bptr -> show();

return 0;
}
```

Ans 8) `#include <bits/stdc++.h>`

```
using namespace std;

void sort( int a[], int arr_size)
{
    int low = 0;
    int high = arr_size - 1;
    int mid = 0;

    while (mid <= high)
    {
        switch (a[mid])
        {
        case 0: swap (a[low++], a[mid++]);
                break;

        case 1: mid++;
```

```
            break;

            case 2:   swap ( a[mid], a[high --]);
            break;
        }
    }
}

void printarray ( int arr[], int arr_size)
{
    for (int i=0 ; i< arr_size ; i++)
     cout << arr[i] << " ";
}

int main()
{
    int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
    int n= sizeof (arr) / sizeof (arr[0]);

    sort (arr, n) ;
    cout << " array after sorting ";
    printarray (arr, n) ;
    return 0;
}
```

Ans C)
```
# include < bits/stdc++.h>
# include < string >
using namespace std;

class member
{
    char name[20], address[60];
    double number;
    int age;

public:
int salary;
  void input()
  {
     cout << " Name : " << endl;
     cin.getline (name, 20);
     cout << " Age : " << endl;
     cin >> age;
     cout << " mobile number : " << endl;
     cin >> number;
     cout << " Address : " << endl;
     cin.getline (address, 60);
     cout << " salary : " << endl;
     cin >> salary;
}
```

```
void display ()
{
    cout<<"Name: " << name << endl;
    cout << " Age: " << age << endl;
    cout << " Phone number: " << number << endl;
    cout << " Address: " << address << endl;
    cout << " salary: " << salary << endl;
}
};

class employee : public member
{
  char specialization[20], department[30];
  public:
  void input ()
{
  cout << " \n Enter the employee details \n";
  member :: input ();
  cout << " specialization: " << endl;
  cin.getline (specialization, 20);
  cout << " \n department: " << endl;
  cin.getline (department, 20);
}

void display ()
{
  cout << " \n displaying employee details \n";
```

```
member :: display ();
cout << " specialization : " << specialization << endl;
cout << " department : " << department << endl;
}

void print salary()
{
cout << " \n salary of the member is : " << salary <<e
}
};

class manager : public member
{
    char specialization[20], department[20];
public:
void input ()
{
cout << " \n enter manager details " << endl;
member :: input ();
cout << " specialization : " << endl;
cin.getline ( specialization , 20);
cout << " department : " << endl;
cin.getline ( department , 20);
}
```

```
void display()
{
    cout << "\n displaying manager details \n";
    member :: display();
    cout << " specialisation : " << specialisation << endl;
    cout << " department : " << department << endl;
}

void printsalary()
{
    cout << " \n salary of the member is : " << salary << en
}
};

int main()
{
    employee e;
    manager m;
    e.input();
    m.input();
    e.display();
    e.printsalary();
    m.display();
    m.printsalary();
}
```