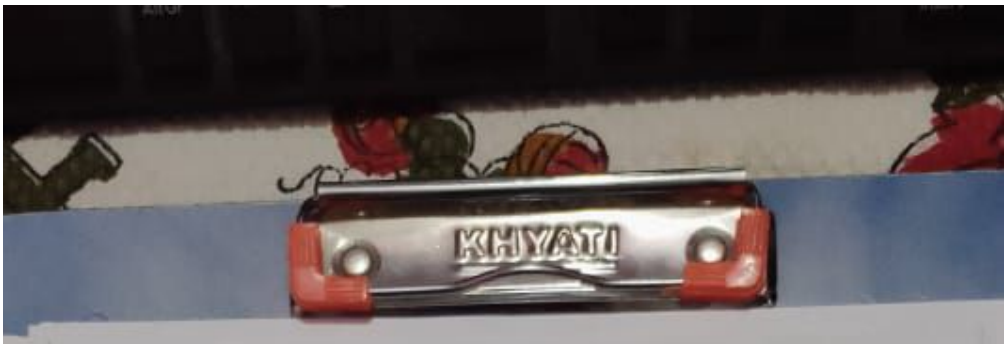



Assignment-2

MCQ

Ans1 → d) stack

Ans2 → c) Compiler Error in line "Derived *dp = new Base;"

Ans 3 → a) Inaccessible

Ans 4 → a) The number of times destructor is called depends on number of objects created.

Ans 5 → a) True

Short Answer Type

Ans1 → The new operator allocates memory for a variable or any other entity on a heap.

Syntax :- pointer_variable = new datatype;

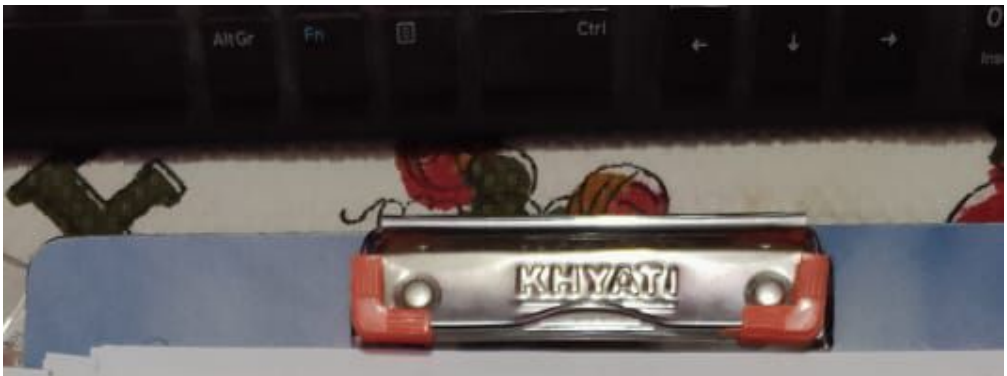
Example - int *ptr = NULL;
ptr = new int();

In this example, we declared pointer variable ptr to integer and initialized it to null. Then using new operator we allocate memory to ptr variable. If memory is available on the heap 2nd line will be successful. If no memory available it throws exception.

The memory allocated dynamically using the new operator has to be freed explicitly by programmer. For this purpose, we are provided with the delete operator.

Syntax : delete pointer-variable;

To free the above memory allocated to ptr variable : delete ptr;




Ans2→ A constructor is a member function of a class which initializes objects of a class.

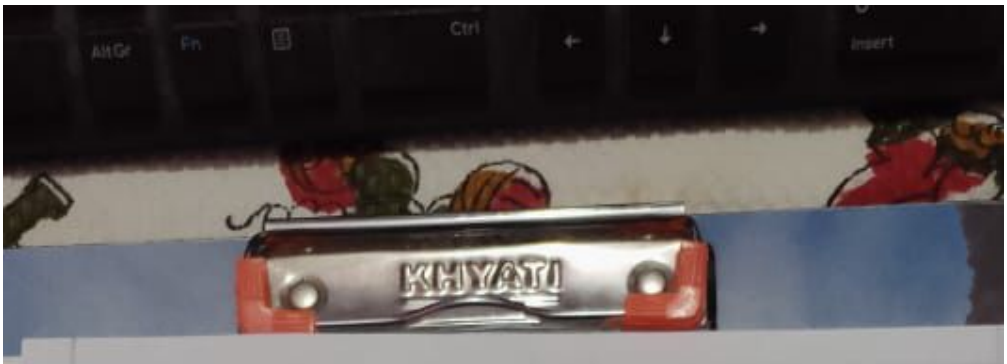
The constructor is used to automatically initialize your object. An object should never exist without being initialized. This is the task of the constructor, initialized the object just when the object is being created.

Types of Constructors:

Default constructor: Default constructor is the constructor which doesn't take any argument. It has no parameter.

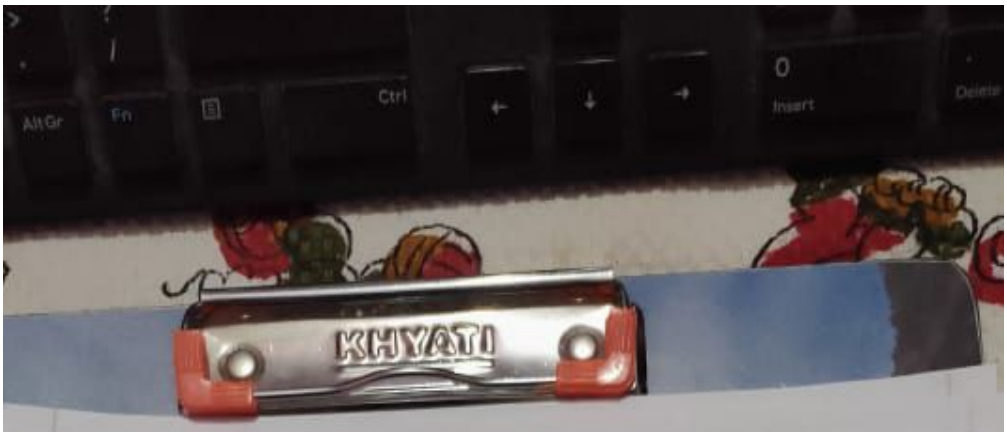
Parameterized constructor: These are the constructors with parameter. Using this you can provide different values to data members of different objects, by passing appropriate values as argument.

Copy constructor: These are special type of constructors which takes an object as argument, and is used to copy values of data members of one object into other objects.




Ans3 →

| Procedural Oriented Programming | Object Oriented Programming |
|---|---|
| * In procedural, program is divided into small parts called functions. | *In this, program is divided into small parts called objects. |
| * It follows top down approach. | * It follows bottom down approach |
| * There is no access specifier. | *It has access specifiers like private, public, protected etc. |
| * Adding new data and function is not easy. | *Adding new data & function is easy. |
| * Less secure | * More secure |
| * Overloading is not possible. | * Overloading is possible. |
| * In this function is more important than data | *In this data is more important than function. |
| * Example: C, pascal etc. | * Example : C++, java, python etc. |




Long answer type questions.

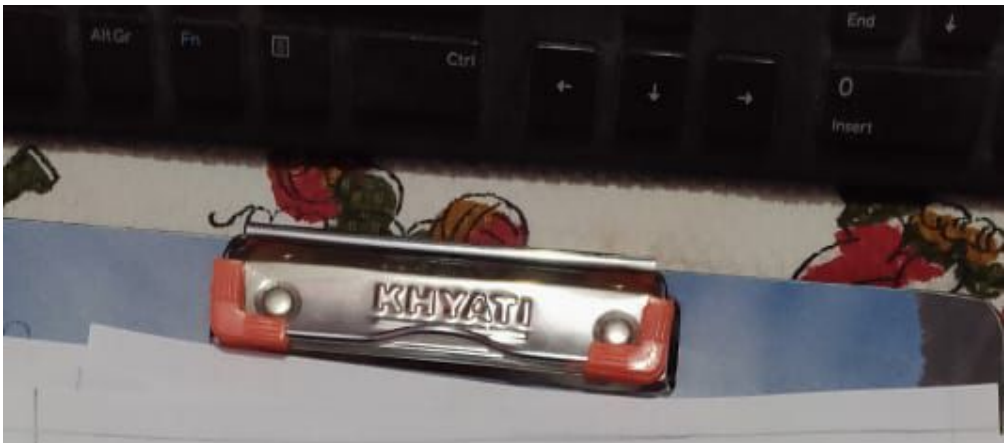
Ans1 → We have two types of polymorphism:

i) Compile time polymorphism

```
#include<iostream>
using namespace std;
class Add{
public:
    int sum(int num1, int num2){
        return num1+num2+num3;
    }
};
int main(){
    Add obj;
    cout<<"Output:"<<obj.sum(10,20)<<endl;
    cout<<"Output:"<<obj.sum(11,22,33);
    return 0;
}
```

Output : 30, 66

In this example, we have two functions with same name but different number of arguments. Based on how many parameters we pass during function call determines which function is to be called, this is why it is considered as an example of polymorphism because in different conditions the output is different. Since, the call is determined during compile time thats why it is called compile time polymorphism.

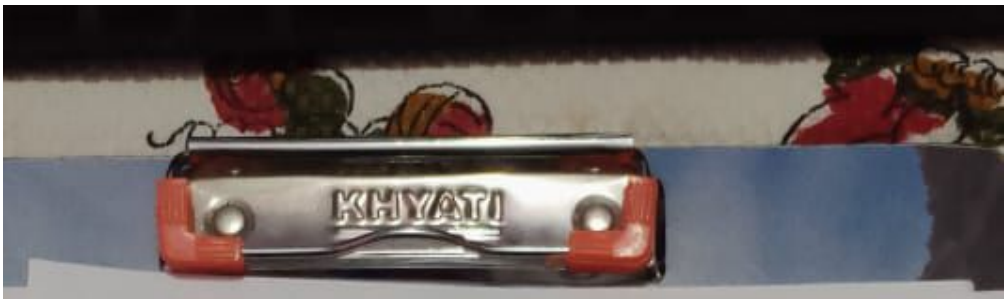



ii) Runtime polymorphism :

Function overriding is an example of Runtime - polymorphism.

```
#include<iostream>
using namespace std;
class A {
  public :
    void disp(){
        cout << " Super class function " << endl;
    }
};
  class B : public A{
  public :
  void disp(){
      cout << " Sub class function ";
    }
};
int main(){
    A obj;
    obj.disp();
    B obj2;
    obj2.disp();
    return 0;
}
```

Output : Super class function / Sub class function.



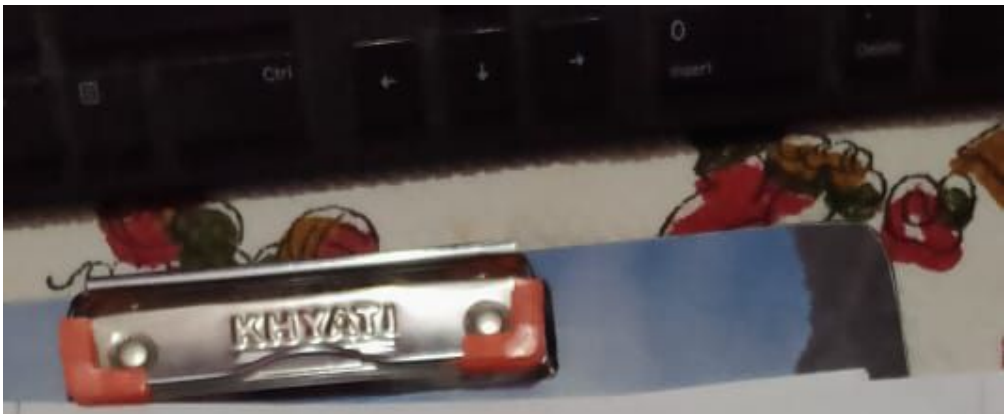

In case of function overriding we have two definitions of the same function, one is parent class and one is child class. The call to function is determined at runtime to decide which definition of the function is to be called, thats the reason it is called runtime polymorphism.

Ans 2 →

```
#include <bits/stdc++.h>
using namespace std;
void sort012 (int a[], int arr_size) {
  int lo = 0;
  int hi = arr_size -1;
  int mid = 0;
while (mid <= hi) {
    switch (a[mid]) {

    case 0:
     swap (a[lo++], a[mid++]);
     break;
    case 1:
      mid++;
      break;
    case 2:
      swap (a[mid], a[hi--];
      break;
    }
  }
}
```

Next →




```
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]);
  Sort012(arr,n);
  cout<<" array ";
  printArray(arr,n);
  return 0;
}
```

Output:

Array

0 0 0 0 0 1 1 1 1 1 2 2