Q1.

a).Is it mandatory to have constructors in a class?Give Reasons.

No it's not mandatory to have constructors in a class because if we do not supply a constructor compiler automatically provides one default constructor.

b)With suitable programs discuss various OOP features.

Data abstraction

```
#include<iostream.h>

class entity
{
   private:
   int data;
   public:
   entity(){}
   int& getdata( ) { return data; }
};


int main()
{
  return 0;
}
```

Polymorphism

```
#include<iostream.h>

class entity
{
private:
int data;
public:
void data(int )
};

int main()
{
 return 0;
}
```

Inheritance

```
#include<iostream.h>

class base
{
private:
public:
base()
{
}
};

class child : public base
{
private:
public:
child()
{
}
};

int main()
{
return 0;
}
```

Q2.

a).How pointers can be used to execute a function.

Functions can be executed using a function pointer.
Syntax of declaring a function pointer.

```
<return type>   (* <function pointer name> )(int,int,...) ;
```

Eg.

```
void (*fptr)(int,int);
```

Assigning address of function to a function pointer

```
<function pointer name> =  &<function name>;
```

Eg.

```
fptr = &sum;
```

Calling function using function pointer..

```
<function pointer name>(<arguments>);
```

Eg.

```
fptr(10,10);
```

b).Write a program to demonstrate function overriding.

```
#include<iostream.h>

class base
{
private:
public:
base(){}
void function(){ std::cout<<"Base class function"<<std::endl; }
};

class derived : public base
{
private:
public:
base(){}
void function(){ std::cout<<"derived class function"<<std::endl; } //overridden function
};

int main()
{
derived d;
d.function();
std::cin.ignore();
return 0;
}
```

Q3.

a)List what operators cannot be overloaded?

1. Conditional operator ?:
2. Scope resolution operator ::
3. Dot operator .
4. Pointer member operator .*
5. Sizeof operator
6. Typeid operator

b)Write a program to eliminate ambiguity during inheritance

```
#include<iostream.h>

class base
{
 base()
{
 std::cout<<"Base class constructor "<<std::endl;
}

virtual void function()  // virtual keyword resolves ambiguity when a function is overridden in child class
{
 std::cout<<”Base class function”<<std::endl;
}
};

class child : public base
{
 child(): base()
{
 std::cout<<”Base class constructor “<<std::endl;
}

void function()
{
std::cout<<”child class function”<<std::endl;
}

};

int main()
{
base* ptr;
child obj;
ptr  = &obj;
ptr->function(); // child class function will be executed if the virtual keyword was not used base class function would have been called.
return 0;
}
```

Q4.WAP to demonstrate conversion of user defined to user defined and user defined to basic data types. Name this feature of OOP.

This feature is called operator overloading.

```
#include<iostream>

class meter
{
private:
unsigned int data;
public:
```

```
meter(unsigned int data) : data(data)
{
}

void print()
{
 std::cout<<data<<"m"<<std::endl;
}

unsigned int Data()
{
    return data;
}
};

class kilometer
{
private:
float data;
public:
kilometer(float data) : data(data)
{
}

void print()
{
 std::cout<<data<<"km"<<std::endl;
}


kilometer(meter m)
{
    data  = m.Data()/1000.0;
}

operator float()
{
    return float(data);
}
};

int main()
{
```

```
meter m(1500);
kilometer km(0.0);
km = m;
km.print();
float fkm = km;
std::cout<<fkm<<" kilometer to float"<<std::endl;
return 0;
}
```

Q5.

a)WAP to sort array elements using pointer

```
#include<iostream.h>

void sort(int* A,int N)
{
int i, j;
    for (i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-i-1; j++)
        {
             if (arr[j] > arr[j+1])
             {
               temp  = A[i];
              A[i] = A[i+1];
              A[i+1] = temp;
             }
        }
     }
 }

int main()
{
int N=0;
std::cout<<”Enter number of elements to enter”<<std::endl;
std::cin>>N;
int* A = new int[N];
std::cout<<”Enter the array elements”<<std::endl;
for(int i=0;i<N;i++)
{
 std::cin>>A[i];
}
sort(A,N);
return 0;
}
```

b).List advantages and disadvantages of pointer.

<u>Advantages</u>

pointers are fast to execute.
pointer helps in creating data structures.
pointers have versatile use cases.
pointers provide direct access to memory.

<u>Disadvantages</u>

a pointer can crash a program that is used incorrectly.
pointers can lead to memory leaks.
pointer needs to be handled explicitly.
pointers can cause exceptions.

Q6.

a).Comment on access specifier during inheritance.

Access specifier within a class specifies access of data to outside of the class.
Access specifier during inheritance specifies access of base class in child class.

public inheritance mode:
private of base class becomes inaccessible in child class.
public of base class becomes public in child class.
protected of base class becomes protected in child class.

private inheritance mode
private of base class becomes inaccessible to child class.
pubic of base class becomes private of child class.
protected of base class becomes private of child class.

protected inheritance mode.
private of base class becomes inaccessible to child class.
public of base class becomes protected of child class.
protected of base class becomes protected of child class.

b).WAP to demonstrate visibility /accessibility of constructors of base class to child class.Comment on order of execution of constructors and destructors.

```
#include<iostream.h>

class base
{
public:
```

```
base()
{
   std::cout<<”Base class constructor”<<std::endl;
}
};

class child : public base
{
 child() :
base()
{
std::cout<<”Child class constructor”<<std::endl;

}
};

int main()
{
child c;
return 0;
}
```

As base class is inherited in public mode so we have access to the constructor of base class in child class.
Same for happens if we inherit in protected mode.
But in private mode the inheritance constructor of base class is inaccessible to the child class.

First constructor of the base class is called then the child class constructor is called when creating the object of the child class.
When destroying object of child class destructor of child class is called first then destructor of base class is called,