POINTERS

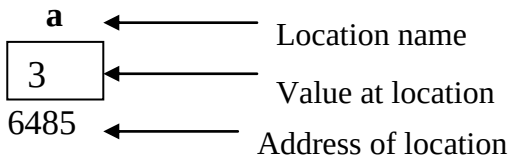
Objectives:

- Declare Pointer & Pointer arithmetic.
- Pointer to Arrays, string & Object.
- "this" pointer concept.

5.1 Concepts of Pointer: Pointer declaration, Pointer operator, address operator, Pointer arithmetic.
5.2 Pointer to Array: Searching, Insertion, deletion
5.3 Pointer to String: Searching, finding length, comparisons, concatenation, reverse
5.4 Pointer to Object: Pointer to Object, this pointer, Pointer to derived class.

5.1 Concepts of Pointer: Pointer declaration, Pointer operator, address operator, Pointer arithmetic.

- "A pointer is a variable that holds an address of another variable."
- Consider the following figure.



- We can print the address of variable a by using & operator (address operator)

```
int main()
{
 int a =3;
 cout << " address of a is : " <<  &a ;
}
```

OUTPUT
Address of a is 6485

DECLARING AND INITIALIZING POINTER VARIABLE:

Ques : How to initialize a pointer explain with a suitable example.(May – 2003)

- Pointer is a variable, which stores an address of another variable.
- Pointer must be declared with its proper type and its name is preceded by *.

Syntax for declaring pointer**:** **Datatype * variable-name;**
Example : int * ptr;

- Where ptr is a pointer which can be used to store the address of an integer.
- Data type of pointer must be same that of the data type of a variable whose address it is going to store.
- After declaring a pointer variable, it must be initialized with the address of another variable.
    - To initialize a pointer variable use following syntax: **Pointer-variable = & ordinary-variable;**

```
Example :    int *ptr;          // ptr is a pointer
             int a = 10;        // a is ordinary variable holding an integer
            ptr = & a;      //initializing pointer with the address of a
```

ACCESSING VALUE OF A VARIABLE USING POINTER:

- To access the value of a variable using pointer, we use * in the expression.
- Consider the following example where a is a variable of type int that holds value 10 and p is a pointer that stores the address of variable a. In short pointer p a points to variable a;

```
int a =10;
int *p;
p =&a;
```

We can print the value of variable a using pointer using following expression.

```
cout << *p;
```

POINTER OPERATORS:

There are two special pointer operators,

1) & is called address of operator :

This returns the address of(its operand.) a variable. Where address is the internal computer's memory location of variable.

```
cout << &a ; //prints  address of variable a
```

2) * operator which is called " value at address " operator.

The "*" is the indirection operator and it is the complement of &. It is the unary operator that returns the value of the variable located at the address specified by its operand.

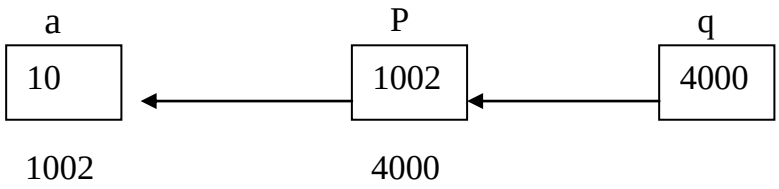
SAMPLE PROGRAM :

```
# include <iostream.h>
void main()
{
 int a = 3;          // ordinary variable
 int *p;  // pointer variable
 p = & a;            // p stores the address of  a
 cout << "Adress of  a = " << &a << endl;
 cout << " Address of  a = " << p <<endl;
 cout << "Value of  a = "<< a <<endl;
 cout << "value of a = " << *p <<endl;
}
```

OUTPUT
Address of a = 1003
Address o a = 1003
Value of a = 3
Value of a =3

POINTER TO POINTER:

A pointer can store the address of another pointer. In other words a pointer can point to another pointer.



In the above example pointer p points to variable a and pointer q points to pointer p.

```
int a = 10;
int * p;            // pointer to ordinary variable
int  **q;                   // pointer  to pointer
p = & a;                    // p points to a
q = &p ;                    // q points to p.
cout << a ;                 // prints value of a - 10
cout << *p;                 // prints value of a using pointer p- 10
cout << **q ;               //prints value of a using pointer q - 10
cout << &a;                 // prints address of a -1002
cout <<  p;                 //prints contents of p -1002
cout << q;                  // prints contents of q - 4000
cout << *q ;                // prints value of p using pointer q – 1002 =*4000=1002
```

- Here a is a variable of type int and p is pointer to variable a and q is pointer to p.
  - If we want to access value of variable a using pointer ,p then use the following expression, ***p** ;
  - If we want to access value of variable a using pointer q , then use the following expression, ****q ;**

POINTER ARITHMETIC:

The following operation can be performed on a pointer.

1) Addition of a number to a pointer.

```
int a =7;
int * p = & a;
p = p+2;
```

2) Subtrcation of a number from a pointer

```
int a =7;
int * p = & a;
p = p - 2;
```

2) Subtraction of two pointers.

```
int a [4] = { 1,2,3,4};
int * p = & a [0];      // p points to the first element of array
int *q = & a [3];       // q points to the last element of array
cout << q –p;
```

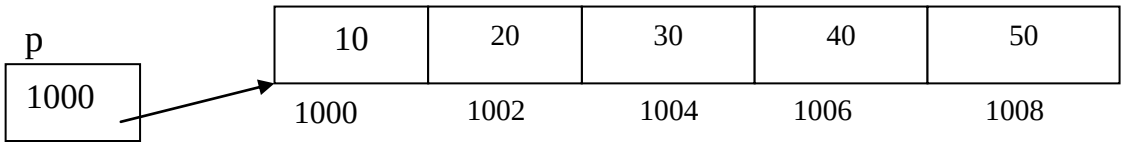
Do not attempt the following operations on pointer.

1) Addition of two pointers.
2) Multiplying a pointer with a number.
3) Dividing a pointer by a number.

Advantages of pointers
- Can return more than one value from function.
-Increasing the programming performance, using pointers we can access the variables faster.
-In case of arrays, we can decide the size of the array at runtime by allocating the necessary space Or dynamic memory management
-Data structure handling (Linked list, Stack, queue)

5.2 Pointer to Array: Searching, Insertion, deletion
POINTERS AND ARRAYS:

- Array is a collection of variables similar data types. Array elements are always stored in contiguous memory locations.
- A pointer for an array points to the base address of an array.
- A pointer is initialized to the memory address of first element placed inside an array.
- We can use this base address for performing arithmetic operations on an array.
- The way pointer can point to a variable, it can point to the entire array.
- The data type of a pointer should be same as that of the data type of array.
- The following figure shows array of 5 integers and their memory locations.



int *p = & arr[0];
int *p = arr

As shown in the diagram , the pointer variable p holds an address of the 0^{th} element in the array. The address of the 0^{th} element is called base address of the array.

```
void main ()
{ int myarray [5] = { 10,20,30,40,50};    // declaring and initializing array
 int *p;  // pointer p
 p =  &myarray [0] ;       // store base address of array in pointer.
}
```

NOTE : Array name itself represents the base address of the array.
That is to store the base address of array in a pointer, just assign the name of array to the pointer.
Thus following two statements have the same effect. Both store the base address of myarray in to pointer p.
1) int *p = myarray;
2) int *p = &myarray [0];

Accessing array elements using pointer.
The following program displays elements of array using pointer.

```
void main ()
{        int myarray [5] = { 10, 20, 30,40,50};      // declaring and initializing array
          int *p;  // pointer p
          p =  &myarray [0] ;       // store base address of array in pointer.
          for ( int i =0 ; i<=4 ; i++)
          {
          cout <<  *p;
          p++;
          }
}
```

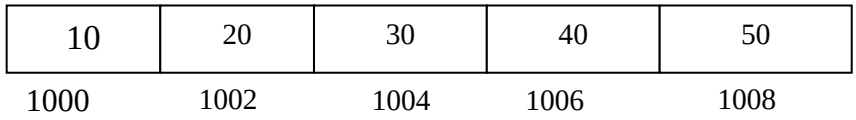
OUTPUT:
10 20 30 40 50

Another way of accessing array elements using pointer.

```
Void main()
{       int myarray [5] = { 10, 20, 30,40,50};        // declaring and initializing array
          int *p;  // pointer p
          p =  &myarray [0] ;       // store base address of array in pointer.
          for ( int i =0 ; i<=4 ; i++)
          {  cout <<  *  ( p + i ) ;
          }
}
```

Incrementing a Pointer

- A pointer when incremented always points to an immediately next location of its type.
- That is if pointer is of type integer then it is always incremented by 2.
  If it's a float then it is incremented by 4 and so on..



To understand ,consider the following program that prints address of all elements in array.

```
void main ()
{
 int arr [5] = { 10, 20, 30,40,50};          // declaring and initializing array
 int *p; // pointer p
 p =  & arr [0] ; // store base address of array in pointer.
 for ( int i =0 ; i<=4 ; i++)
{
 cout << *p <<" : " <<  p <<endl;
 p++;
 }
}
```

OUTPUT

```
10 1000
20 1002
30 1004
40 1006
```

- Notice that when pointer is incremented (p++) it is incremented by 2 every time.

PASSING AN ENTIRE ARRAY TO THE FUNCTION

```
Void main ()
{   void display ( int * , int );      // function prototype
 int a[5] = {1,2,3,4,5};
display ( a , 5 );
}
void display ( int *p, int n)
 {
   for (int i =0 ; i<=4 ; i++)
  {
        cout << *p;
        p++;
  }
}
```

NOTE : If num [ ] is an array of integers, then remember the following

1. *Name of array i.e. num* represent the base address of array.i.e. num is equivalent to &num[0]
2. Expression * num and * (num + 0) are same and both give the value stored at base address (value of 0^{th} element) of array.
3. The following expressions are same
   num [I] *(num + I) * (I + num)
4. The compiler never allows us to change the base address of array.
   The following expressions will report error " Illegal use of pointer"

```
num++;
num = num +2;
```

Ques : If intarray is an array of integers, why the expression intarry++ is illegal ? justify

Here intarray is an array of integers.

- We know that name of array represents the base address of the array.
- If we write,

```
intarray ++;
```

We are trying to change the base address of array.

- The compiler would never allow changing the base address this is because all that known to the compiler about an array is its base address.
- If the base address is lost, the compiler will never be able to access the array.

- That is why when ever we try to alter the base address , compiler throws an error " Illegal use of pointer".
- If you want to manipulate the array, create another pointer and store the base address of array in it.

```
int intarray [5];
int * p = intarray;
```

- Now using pointer p you can manipulate intarray. The statement p++ will not generate error.

Ques : differentiate variable and pointer variable: (specimen paper)

Variable	Pointer variable
The c++ ordinary variable is a place holder where we can store values of different type such as integer, floating –point , character etc.	Pointer is a special variable that stores the address of another variable.
Declaring variable: Syntax : Data-type variable-name; Example: int a ;	Declaring pointer variable: Syntax : Data-type *variable-name; Example: int *p; Pointer declaration is always preceded by *
Initializing variable; Syntax : Datatype variable –name = value; Example : int a = 10;	Initializing pointer variable; Syntax : Datatype * pointer-variable = & variable; Example: int *p = &a; Pointer is always assigned the address of another variable.
Accessing value of variable: Cout << a;	Accessing value of variable using pointer. Cout << *p; Where pointer p points to variable a;
Accessing the memory location using variable is slower than accessing using pointer.	Accessing the memory location using pointer is faster than accessing using a variable.

PROGRAMS BASED ON POINTER TO ARRAY

❖ **Program to search an element in the array using pointer: (Searching)**

```
# include <iostream.h>
void main()
{
 int a[10], n, m, i;
 int *p;
 p =a;
 cout << "Enter how many elements?";
 cin >> n;
 cout << endl << "Enter " << n << "elements;
 for (i=0 ; i<n ;i++ )
 {
        cin >> *( p + i);
 }
 cout << endl << "Enter element you  want to search";
 cin >> m;
 for ( i=0 ; i<n; i++)
 {
    if ( *p == m )
        {
        cout << "Position of element is "<< i+1;
         break;
        }
    else
        p++;
 }
  if ( I == n )
 cout<<"Element  not found";
}
```

OUTPUT

```
Enter how many elements? 4
Enter 4 elements
10 23 4 56
Enter element you want to search 23
```

Position of element is 2

❖ **Program to find out maximum number in array.**

```
# include <iostream.h>
void main()
{ int a[10], n,  i;
 int *p;
 p = a;
 cout << "Enter how many elements?";
 cin >> n;
 cout << endl << "Enter " << n << "elements;
 for (i=0 ; i<n ;i++ )
 {  cin >> *( p + i);  }
 int max = a [0];
 for ( i=1 ; i < n ;i++)
{
            if ( * p > max)
           {        max = *p;
                    p++;
           }
            else
                    p++;
 }
          cout << max;
}
```

❖ **Program to insert an element into array using pointer. (Insertion)**

```
# include <iostream.h>
# include<stdlib.h>
void main()
{ int a[10],n,m,i,pos,*p;
 p=a;     // store a base address of array in p
 cout<<"Enter how many elements?";
 cin >> n;
 cout << "Enter" << n << "elements";
 for (i=0;i<n;i++)
 {
          cin >> *( p + i);
 }
 cout<<"Enter element  u want to insert";
 cin >> m;
 cout<<"Enter position";
 cin >> pos;
  for ( p = p + n , i= n - 1; i >= pos-1 ; i--, p--)
 {
           *p=*(p - 1);
          }
  *p=m;
 for (i=0; i <= n; i++)
 cout<< * (p + i );
 }
```

OUTPUT

```
enter how many elements?5
12 13 14 16 17
Enter element u want to insert 15
Enter position4
12 13 14 15 16 17
```

❖ **Program to reverse an array using pointer (reversing)**

```
# include <iostream.h>
# include <conio.h>
        void main()
        {           int I;
         int a[10],n;
        int *p ,*q;
        p = a ; // p points to first element of array
        cout<<"Enter how many elements? ";
        cin>>n;
        cout << "Enter" << n << "elements";
         for (I=0;I<n;I++)
         {
                  cin >> *( p + I);
         }
        q= a + n - 1;      // q points to the last element of array
          for (I=0; I<=n/2;I++)
          {   t=*p;
            *p=*q;
            *q=t;
             p++;
             q--;
          }
         cout <<"\nThe reverse array is :"<< endl;
         for (I=0;I<n;I++)
         cout<<endl<<*(p+I);
        }
        OUTPUT
        Enter how many elements? 5
        Enter 5 elements:
        12 34 56 7 4
        The reverse array is :
        4    7 56 34 12
```

❖ **Program to sort an array .(sorting)**

```
        # include <iostream.h>
        # include <conio.h>
        # include <stdio.h>
        void main()
        {           int a[10],n ,I,t;
         cout<<"\nEnter how many elements?";
         cin>>n;
        cout << "Enter" << n << "elements";
         for (i=0;i<n;i++)
            cin >> *(p + i);
          for (i=1; i<n;i++)         // number of iterations
           {
             for (j=0;j<=n-i; j++)  // number of comparisons in each iteration
             {
               if (*(p+j) > *(p+j+1))
               {
                    t= *(p+j);
                    *(p+j) = *(p+j+1);
                    *(p+j+1)=t;
               }
             }
           }
         cout <<"\nThe sorted array is :"<< endl;
```

```
 for (i=0;i<n;i++)
 cout<< *(p+i);
 }
```

OUTPUT

Enter how many elements?4
12 4 7 22
The sorted array is :
4 7 12 22

5.3 POINTER TO STRING:

- String is nothing but an array of characters.
- Pointer to string is a pointer to character data type which points to the base address of the string. Using this pointer ,we can access individual characters in string.

Declaration of pointer to string:
Syntax : data - type *ptr;
Example : char *ptr;

- To initialize pointer to the base address of the string ,assign the name of the string to the pointer.

```
Char  city[ ] = "mumbai";    // city is a character array containing "mumbai"
Char *p;     // pointer to string
P = city;    // assign base address of city array to pointer p
```

PROGRAM BASED ON POINTER TO STRING:

Program to find length of a string using pointer.

```
# include <iostream.h>
void main()
{
  char s[40];// declare character array
  char *p;              // declare pointer to string
  p = s;                // initialize pointer to the base address of the string
  int length =0 ;
  cout<<"Enter string  : " ;
  cin>> s;
 while ( *p != '\0')
{
     length++;
     p++;
}
cout << "Length  of string = " << length;
}
```

OUTPUT

Enter string : vidyalankar
Length of string = 10

❖ **Program to reverse a string using pointer to string.**

```
 # include <iostream.h
# include < string.h>>
void main()
{ char s[40];          // declare a character array
 char *p , *q;         // declare two pointers
 cout<<"Enter string  : " ;
 cin>> s;
 len = strlen (s);     // find the length of the string.
 p = s;        // initialize pointer p to the first character of  array s
q = p +s-1;  // initialize q to point to the last character of array s
//Now swap first and last character , then second and second last character and // so on till len /2
 char t;
for (int i =0 ; i < len /2 ; i++)
{
     t = *p;
     *p =*q ;
```

```
      *q =t;
      p++;
      q - -;
}
cout << “After Reversing  the string is : “ <<s;
}
```

OUTPUT

```
Enter string : dipali
After reversing the string is : ilapid
```

❖ **Program to compare to strings.**

```
# include<iostream>
int cmp(char *,char *);          // function prototype
void main( )
{
  char s1[40], s2[40];
  cout<<"Enter string1  : " ;
  cin>> s1;
  cout<<"Enter string2  : ";
  cin>>s2;
  int i=cmp(s1 , s2);
  if( i <0)
  cout<<"string1 is less than string2";
  else if (i>0)
  cout<<"string1 is greater than string2";
  else
  cout<<"eqaual";
 }
int  cmp (char *s1,char *s2)
{
   while(*s1 == *s2)
  {
     if(*s1=='\0')
          return(0);
    s1++;
    s2++;
  }
  return(*s1- *s2);
 }
```

Output

```
Enter string1  : dipali
Enter string2  : dip
string1 is greater than string2
```

❖ **Program to copy one string to another using pointer to string (May –2003)**

```
# include <iostream.h>
void copy(char *,char *);      // function prototype
void main()
{   char s[40],t[40];
  cout<<"Enter source string ";
  cin >> s;
  cout<<"Enter target string";
  cin >> t;
  copy ( t , s);
  cout<<"\nAfter copy Target string is:";
  cout << t;
 }
void copy(char * t, char *s)
{
  while(*s != '\0')
  {
   *t=*s;
    t++;
```

```
   s++;
  }
  *t='\0';
 }
```

Enter source string : dipali
Enter target string: sada
After copy Target string is: dipali

❖ **Program to concat(join) two strings using pointer to string. (specimen paper)**

```
# include <iostream.h>
void cat (char *,char *);        // function prototype
void main()
{   char s1[40], s2[40];
  cout<<"Enter string1  : " ;
  cin>> s1;
  cout<<"Enter string2  : ";
  cin>>s2;
  cat ( s1,s2);
 cout << "After concatenation  : "<< s1 <<endl;
 cout << "After concatenation  : "<< s2 <<endl;
}
void  cat ( char *s1 , char * s2)
{
     while ( *s1 != '\0')
               s1++;                // take the pointer s1 at the end of first string.
     while (*s2 != '\0')
     {         *s1 = *s2;          // copy s2 char by char to s1
               s1++;
              s2++;
     }
     *s1 = '\0';              // after concatenation append null
}
```

OUTPUT:
Enter string1 : dipali
Enter string 2 : sadavarti
After concatenation s1 = dipalisadavarti
After concatenation s2 = sadavarti

❖ **Program to count number of vowels in a given string.**

```
# include <iostream.h>
void main()
{  char s[40];          // declare character array
  char *p;              // declare pointer to string
  p = s;                // initialize pointer to the base address of the string
 int count = 0;
  cout<<"Enter string  : " ;
  cin>> s;
  while ( *p != '\0')
{
     if ( *p == 'a' || *p == 'e' || *p == 'i' || *p == 'o' || *p == 'u')
     count++;
     p++;
}
cout << "No. of vowels = " << count;
}
```

Enter string : education
No. of vowels = 5

Program to search a character in a string using pointer to string

```
# include <iostream.h>
void main()
{  char s[40];          // declare character array
  char *p;              // declare pointer to string
  p = s;                // initialize pointer to the base address of the string
```

```
 cout<<"Enter string  : " ;
 cin>> s;
char ch;
int pos = 0;
cout << “Enter character u want to search : “;
cin >> ch;
 while ( *p != ‘\0’)
{
     if ( *p == ch)
     {
          cout << “ character found at position << pos +1;
          break;
     }
     pos ++;
     p++;
}
if ( *p == ‘\0’)
     cout << “character not found;

}
Enter string : education
Enter  character u want to search : c
Character found at position : 4
```

Program: To find whether the string is palindrome using pointer to string.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
char str[20],str1[20],*ptr,*ptr1;
int t=0;
clrscr();
cout<<"\n Enter a string : ";
cin>>str;
strcpy(str1,str);
strrev(str);
ptr=str1;
ptr1=str;
while(*ptr!='\0' && *ptr1 !='\0')
{
if(*ptr!=*ptr1)
{
t=1;
break;
}
ptr++;
ptr1++;
}
if(t==0)
cout << "\nString is palindrome.";
else
cout << "\nString is not palindrome.";
getch();
}
```

POINTER TO OBJECT:

- A pointer can point to an object created by a class.
- Consider the following example :**product x;**

Where ‘product’ is a class and x is an object of product class.

We can define pointer ptr of type product, as shown below**. product *ptr.**

Now this ptr can be used to store an address of object of class product. Pointer to objects are useful in creating objects at run time.

Example:

```
class product
{
private: int code;float price;
public:void getdata(void)
        {
        cout << "enter code";
        cin>>code;
        cout << "enter price";
        cin>>price;
        }
        void display(void)
        {
        cout << code <<price;
        }
};
void main()
{
product p1;             // create object of product
product *ptr;           // create pointer of type product
ptr = &p1;              // ptr points to object p1
p1.getdata ();          // Invoking getdata()  using object
ptr -> getdata ();      // Invoking getdata() using pointer to obejct
p1.display();
ptr ->display();
}
```

Here -> is called member access operator.
It requires a pointer to object on LHS and member function on RHS.
NOTE : The following statements are same

1) p -> getadat();
2) (*p) **.** getdata();

CREATING OBJECTS USING POINTER / NEW OPERATOR

We can create the object using new operator. The new operator is used to create an object at run time.
Syntax :

```
Datatype *pointer-var = new datatype;
```

Example :

```
product  *p = new product;
```

The new operator allocates required memory space for the object and returns the address of the allocated memory to the pointer variable on LHS (i.e. p)
We can also create an array of objects using pointer.
Syntax :

```
Datatype  *pointer-var = new datatype [ size];
product *p = new product [5];
```

This statement allocates memory space array of 5 objects of type product and return the base address of the array to the pointer p;
Using new operator we can create an array of integers also.
Thus to create an array of 10 integers write,

```
int  *p = new int [10];
```

PROGRAM ILLUSTRATING USE OF POINTER TO OBJECT:

```
# include <iostream.h>
class product
{       int code;
        float price;
public:void getdata(void)
        {
        cout << "enter code";
        cin>>code;
        cout << "enter price";
        cin>>price;
        }
        void display(void)
        {
```

```
        cout << code <<price;
        }
};
void main()
{ int n;
cout <<"enter how many products " ;
cin >> n;
product   *p = new product[n ];  // create array of 5 products using pointer to object.
for (int I=0; I<=n ; I++)
{
   p -> getdata ();
   p-> display();
   p++;
}
}
```

This Pointer :

Ques : Explain the concept of this pointer (Dec-2003 ,may-2005)

- C++ uses a unique keyword called 'this' to represent an object that invokes a member function.
- This unique pointer is automatically passed to a member function when it is invoked.
- 'this' is a pointer that always points to the object for which the member function was called.
- For example, the function call A.max() will set the pointer 'this' to the address of the object A. Next time suppose we call B.max() , the pointer 'this ' will store address of object B.

Consider the following example:

```
Class sample
{       int a;
public :void setdata (int x)
        {       this ->a = x;
        }
};
```

The private variable a can be directly used inside a member function like **a =123;**

We can also use the following statement to access private variable a **this -> a =123;**

```
class example
{       private : int m;
        public:void setdata( int a)
        {       m = a ; // one way to set data
                this -> m = a; // another way to set data using this pointer
        }
        void showdata (void)
        {       cout << m;      // one way to display data
                cout << this ->m;       // another way to display data using this pointer
        }
};
```

Practical use of this pointer:

1) **this** pointer is normally used when we overload the operator using member function
2) **this** pointer is used to return object by reference.

- We can pass object to the function and return object from the function either by value or by reference. When entire object is passed / returned by value, a copy of that object is created. This leads to considerable wastage of memory.
- But when the object is passed/ returned by reference, no new copy is created, only reference to that object is passed to the function, this saves memory.
- We can return object by reference using this pointer as shown below.

  Return *this;

- The above statement returns the object by reference.

Returning object by reference using this pointer

```
# include <iostrem.h>
Class circle
{ private : int rad;
            float x , y;
public:circle (){ }         // default constructor
        circle ( int rr ,float xx , float yy)
```

```
        {       rad = rr ;
                x =xx ;
                y=yy ;
        }
        circle& operator = ( circle &c)     // overloading = operator to copy data //members of one object to another.
        {       this -> rad = c.rad;
                this ->x = c.x;
                this->y = c.y;
               return *this;                // returning object by reference
        }
        void showdata()
        {
                cout << rad << x <<y;
        }
};
void main()
{
circle c1,c2;
c1.setdata(3,5,6);
c2 =c1;
c1.shwdata();
c2.showdata();
}
```

POINTER TO DERIVED CLASSSES:

- We can use pointer not only to the base objects but also to the object of derived class.
- "Pointers to object of base class are type compatible with pointers to objects of derived class". This means the base class pointer can point to the object of derived class.
- There fore a single pointer variable (base class pointer) can be used to point to the object of different derived classes.
- For e.g. If B is a base class and D is a derived class form B, then a pointer declared as a pointer to B can also be pointer to D.

Consider the following declarations

```
        B *ptr;  // pointer to base class
        B b;            // create base class object
        D d;            // create derived class object
        ptr = & b;      // ptr points to base class object
```

We can make ptr to point to derived class object also.

```
        Ptr = &d;       // will work.
```

- This is valid because, d is an object of class D that is derived form class B.
- However there is a problem using ptr (pointer to base class) to access the public members of the derived class.
- Using base class pointer ,we can access only those members of derived class, which are inherited from base class. The members that originally belong to derived class cannot be accessed.
- Moreover, in case a derived class member function overrides the base class member function (that is, same function exists in both base and derived class) , and if you invoke this member function using ptr (pointer to base class) then always base class function will be invoked.
- This is because the compiler ignores contents of the pointer and chooses a member function that matches its type. Since ptr is of type base class it will always invoke the base class member function.

```
# incluide <iostream.h>
Class base
{  public:void display ()
{       cout << "In base class "<< endl;
}
};
Class derived1: public base
{  public:void display ()
{       cout << "In derived1 class " << endl;
}
};
void main()
{       base *ptr;      // pointer to base class
```

```
        base b1;          // create object of base class
        ptr = &b1;        // ptr points to base class object b1
        b1.display () ;   // invokes base  class display()
        derived d1;       // create object of derived class
        ptr = & d1;       // ptr now points to derived class object d1 . will work
        ptr -> display ();          // Invokes base class display();
}
```

OUTPUT

In base class

In base class

- Always base class function is called irrespective of whose address is stored in ptr. Compiler checks the type of a ptr and ignores contents of ptr.

```
#include <iostream>
using namespace std;
class emp
{
 private :
 int emp_id;
  float salary;
public:
void getdata(void)       // Inline function
     {
               cout <<"\n ENTER EMP ID "<< endl;
   cin>>emp_id ;
   cout <<"\n ENTER SALARY"<<endl;
   cin >> salary;
            }
          void showdata(void)       // Inline function
          { cout <<"\n EMP ID =  "<< emp_id<<endl;
 cout <<"\n SALARY ="<< salary << endl;
         }
};   // END OF CLASS
int main()
{

  emp e1,*e;;
  e=&e1;// creating 2 objects of class emp
 e->getdata();   // get data for object e1
e->showdata(); // display data   object e1
return (0);
}
```