13
CHAPTER

POINTER IN C++

Inside This Chapter

13.1. Declaration of Pointer
13.2. Accessing the Variable Pointed
13.3. Pointer to Void
13.4. Pointer Arithmetic
13.5. Pointer and Array
13.6. Pointer and String
13.7. Array of Pointer
13.8. Pointer to Pointer
13.9. Pointer to Functions
13.10. Pointer as an Argument to Function
13.11. Pointer and Structure
13.12. Pointer and Class
13.13. Memory Management through Pointer
13.14. Dynamic Array
13.15. Memory Management Operator and Function
13.16. Member Dereferencing Operator
13.17. Dynamic Memory

Pointer is a variable which stores the address of another data type. The pointer is powerfull technique by which we can access data by indirect reference.

Pointer is also stored in memory but holds the address of another variable. If p is pointer which holds the address of variable x then this is known as p point to x.

13.1. DECLARATION OF POINTER

The general format of the pointer declaration is.

```
data - type * pointer - variable;
```

for example → int * p;

means the p is pointer variable which can store the address of int type data i.e. p point to integer type data.

pointer–variable is any identifier and data-type is any data-type of C++, like int, float etc.

When a variable name is preceeded by asterisk (*) then that is pointer - variable.

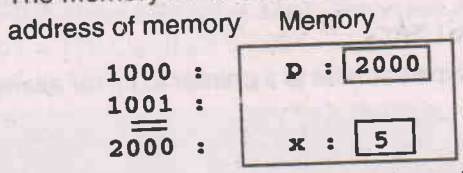
When we declare pointer variable that can store the address of data-type which is written infront of that. Now next point is to assign the address of a variable to pointer variable as follows :

```
int * p; //declaration of pointer
int x; //declaration of variable
p = &x;
x = 5;
```

The statement p = &x; assigns the address of x to p. The & is a unary operator that returns the memory address of its operands.

The * is known as **pointer operator** and & is **address operator** both has same pecedence as the other unary operator such as ++, – –, etc.

The memory for above four statement is as follows :



Note that pointer stores the address. Therefore its contents are unsigned integer type.

13.2. ACCESSING THE VARIABLE POINTED

The variable which is pointed by pointer variable can be accessed as follows :

```
int * p;
int x;
x = 20;
p = &x;
cout<<*p;
```

The statement cout << * p; print the value 20 because *p means value at the address p. Similarly

```
*p = *p + 10;
```

is same as —

```
x = x + 10;
```

Note that if we write the following statement —

```
cout << p;
```

this prints the address of x. Similarly the statement —

```
cout << &x;
```

also prints the address of x.

Example 13.1

```
# include <iostream.h>
# include <conio.h>
 void main ( )
 { int *p;
   int x;
   clrscr ( );
   cout <<"Enter a value :";
   cin >>x;
p = &x; //assign address of x to p
*p = * p + 10; // same as x = x+10
cout << "Now x is :" <<x<<endl;
cout << "Value of x through p:"<<*p<<endl;
cout <<"The address of x is "<<&x<<endl;
cout <<"The address of x is"<<p<<endl;
getch ( );
  }
```

The sample output of above program is :

```
Enter a value : 25
Now x is : 35
Value of x through p : 35
The address of x is 0 × 8f51fff4
The address of x is 0 × 8f51fff4
```

Note that may be the address of x is printed different from o × 8f51fff4 in your computer.

13.3. POINTER TO VOID

VoID pointer

Note that we can not assign the address of a float type variable to a pointer to int for example :

```
float y;
int *p;
p = &y;  //illegal statement
```

Similarly see the following :

```
float * p;
int x;
p = &x; //illegal statement
```

That means if variable type and pointer to type is same then only we can assign the address of variable to pointer variable. And if both are different type then we can not assign the address of variable to pointer variable but this is also possible in C++ by declaring pointer variable as a void as follows :

```
void *p;
```

The above pointer is called pointer to void type. That means pointer p point any type of data int, float etc. for example :

```
void *p;
int x;
float y;
p = & x; //ok
p = & y; //ok
```

13.4. POINTER ARITHMETIC

We know that a pointer stores the address of memory location but we can perform some arithmetic with pointer.

1. Increment :

If p is a pointer type then ++p; or p++; is possible with pointer. Note that ++p or p++ not add 1 to p but add size of that data type with p which is pointed by pointer. for example —

```
int x, *p;
p= &x;
p++;
```

1000+ 1*2

If p is 1000 after the statement p =&x then after p++ the p will be 1002 because the size of (x) is 2 byte.

The difference between p++, ++p is same as post fix and prefix increment with basic type. for example :

```
int x, *p1, *p2, *p3;
p1 = &x;
p2 = p1++;
p3 = ++p1;
```

If location of x is 1000 then after execution of p = &x; the value 1000 is assigned to p;
After statement p2 = p1++; the value p1 is assigned to p2 first then p1 is incremented. i.e.
p2 = 1000, and p1 = 1002 after above statement because size of X is 2 byte.
After statement p3 = ++p1; the value of p1 is incremented first then assigned to p3 i.e.
p1 = 1004, and p3 = 1004
p1 is 1004 because p1 is 1002 after statement p2 = p1++;

2. Decrement :

If p is a pointer type then p– – of – – p is possible. p– – or – –p not decrement I but the size of data type which is pointed by p.

p – – is postfix decrement and – – p is prefix decrement. The difference is same like p++ and ++p.

For example:

```
int x. *p1, *p2, *p3;
p1 = &x;
p2 = p1 - -;
p3 = - - p1;
```

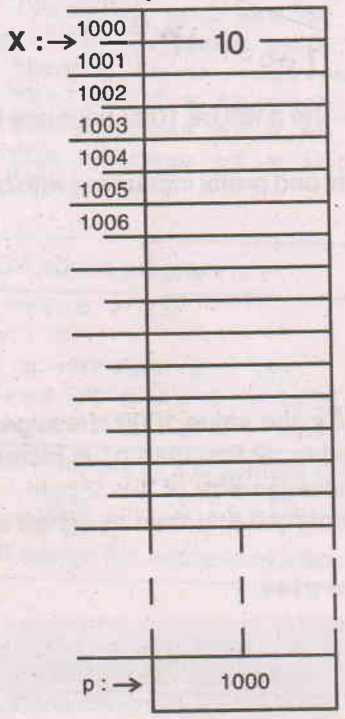
If address of x is 1000 then after statement p1 = &x; p1 contain 1000.
after statement p2 = p1 – –; p2 is 1000 and p1 is 998 (p1 – size of x)
after statement p3 = – –p1; p1 is 996 and p3 is also 996.

3. We can add integer value with pointer :

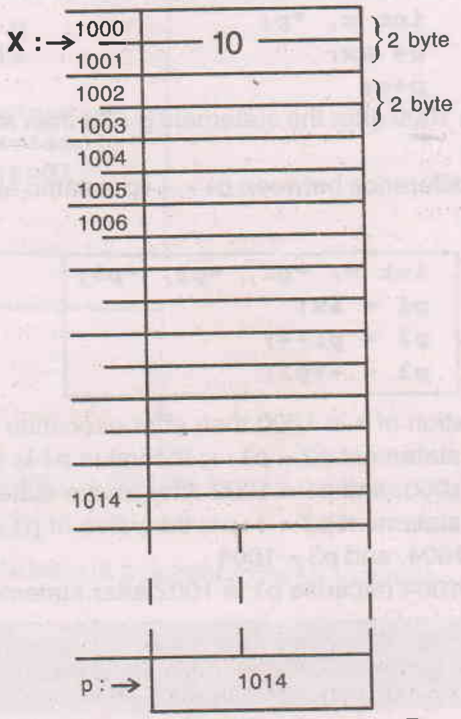
We can add any integer value to pointer like follow :
p2 = p1 + 6; where p2 and p1 both are pointer type, point to same type of data.
after p2 = p1 + 6; p2 to point to the sixth element of pointer type beyond the element pointed to by p1.
See the following statement

```
int *p, x;
x = 10;
p = &x;
p = p + 7;
```

if address of x is 1000 then p is 1000 after statement p = &x.
after statement p = p + 7; now p is → 1000 + 14 = 1014 because one integer type element take 2 byte. The memory for above is :



After statement p = &x;



After statement p = p + 7

4. We can subtract integer value from pointer :

We can subtract an integer number from pointer type like addition. for example p = p-9; (where p is a pointer type) after above statements now p points to an element which is ninth element back from an elements which currently points to. For example :

```
float x, *p;
p = &x;
p = p - 3;
```

2000 - 3*4

After statement p = &x; the address of x is assigned to p if address of x is 2000 then p is assigned to value 2000.

After p = p–3; the value of p is 1088. (i.e. 2000 – 3* 4) because float type data takes 4 byte.

Note : Only above arithmetic operators are allowed with pointer type other than above are not allowed. i.e. following are not allowed with pointer type :

1. To multiply or divide
2. To add or substract type float or double to pointers.
3. To operate the bitwise shift etc.

13.5. POINTER AND ARRAY

There is a close association between pointer and array. Array name is pointer to itself. For example :

```
int a[5], *p;
p = & a[o];
```

after statement p = &a [o]; p point the array i.e. the p contains the address of a[o], (first element of array)

p++;

after this statement p points to a[1] element.

Example 13.2

```
# include <iostream.h>
# include < conio.h>
  void main ( )
   { int a[5] = {5,6,7,8,9}
     int *p;
       p = &a [o];
     Clrscr ( );
   for (int i = 0; i < =4; i ++)
     cout << * (p+i) << endl;
       getch ( );
                 }
```

The output of above program is :

```
5
6
7
8
9
```

because the statement cout << * (p+i) << endl; execute 5 times from i = 0 to 4.

When i = 0 then * (p+i) is * (p+o) means * p therefore this print the value which store at location p. Similarly * (p+1) means the value of second element, because p+1 is pointer to the second element of array. Similarly other elements are printed.

We know that array name is also pointer to that so the above program can be written as follows also :

Example 13.3

```
# include <iostream.h>
# include <conio.h>
  void main ( )
        {
int a [5] = {5,6,7,8,9}
 Clrscr ( );
for (int i = 0; i ++; i < = 4)
cout << * (a+i) << endl;
     getch ( );
           }
```

The output of above program is same as for example 13.2.

13.5.1. Pointer and Multidimensional Array

We know that the name of array is point to first element of array i.e. 0th element. In two dimensional array the array name point to [o] [o] element. In three dimensional array the array name point to [o] [o] [o] element etc.

In two mensional array the second element is [o] [1].

If we want to evaluate the [1] [2] element through pointer then the expression for that is —

* (*(p+1) + 2); where p is pointer to array (i.e. p contains the address of [o] [o] element) or p is name of array.

Example 13.4

```
# include <iostream.h>
# include <conio.h>
void main ( )
 { int a [2] [3] = { {5,6,7},
                     {8,9,10}};
   Clrscr ( );
   for (int i = o; i < = 1; i++)
 { for (int j = o; j < = 2; j++)
cout <<*(*(a+i)+j) <<'\t';
       cout<<endl;}
   getch ( ); }
```

The output of above program is

```
5    6    7
8    9    10
```

13.6. POINTER AND STRING

We know that strings are array of char. Therefore pointer notation can be applied to the characters in strings, just as it can to the elements of any array. But in case of string if we initialized the string at the time of declaration then the string name is constant. Therefore we cannot perform pointer arithmetic operation with that see the following example :

Example 1

```
# include <iostream.h>
# include <conio.h>
 void main ( )
    { char str [ ] = "Rajesh Pandey";
      Clrscr ( );
     cout << str; // display the string
      str ++ //illegal
      getch ( );
        }
```

Above program give an error because str is constant we can not write statement like str ++. But if we declare the string by pointer then we can perform arithmetic operation. See the following example :

Example 2 :

```
# include <iostream.h>
# include < conio.h>
void main ( )
{ char * str1 = " Rajesh Pandey";
  Clrscr ( );
  cout << str1 << endl;
  str1 ++; //ok
  cout << str1 << endl;
  getch ( );
       }
```

The above program is ok the output of above program is :

```
Rajesh Pandey
ajesh Pandey
```

str1 is a pointer type variable therfore we can perform arithmetic operation str1 ++, point the string from second character.

But in example1 the str is an array name, that is a pointer to string. which contains a address which is constant. i.e. we can say that that is a pointer constant.

13.6.1. String with Pointer Notation as a Function Arguments

When we represent a string using pointer that is known as string with pointer notation. as in the above example 2. i.e.

```
char *s;
```

s is a pointer which point to char type data but this also uses for string.

Example 13.5

```
# include <iostream.h>
# include <conio.h>
void main ( )
{ void display (char * str)
  char s [ ] = "Archana";
  Clrscr ( );
  display (s);
  getch ( );
      }
void display (char * str)
  {
     while (*str)
       cout << * str ++ << endl;
             }
```

Archana 10 10 A

The output of above program is :

```
A
r
c
h
a
n
a
```

In above program s is pointer constant but this is passed to function where str is pointer variable. within the body of function the loop

```
while (*str)
  cout << * str ++ << endl;
```

terminates when * str is o i.e. upto the end of string.
* str ++ first print constants pointed by str then incremented.

Example 13.6
Copy of one string by using pointer.

Ans.

```
# include <iostream.h>
# include < conio.h >
void main ( )
  { void copy (char *, char *);
    clrscr ( );
    char *s1 = "PANDEY";
    char s2 [80];
    copy (s2, s1);
    cout << s2 << endl;
    getch ( );
           }
void copy (char *d, char *s)
      { while (*s)
            * d ++ = * s ++;
          * d = '\o'; }
```

Output of above program is :

```
PANDEY
```

Pandey
Pauzey

In above program the statement

```
*d ++ = * s ++;
```

means first the value pointed by s is assigned to *d then both d and s increment, because the increment is postfix increment.

13.7. ARRAY OF POINTER

We know that array is homogeneous collection of data. This is also possible that an array element can be pointer type that array is known as array of pointer. The array of pointer is declared as follows :

```
data - type * array - name [size];
```

char arty[u];

for example

```
int * a[10]
```

makes an array of 10 element. Each element is a pointer type which points to int type data.

```
char *a [10];
```

makes on array of pointer to strings; Because char *s; makes an pointer to string, therfore an array which each element points to string is a string type.

Example 13.7

```
# include <iostream.h>
# include <conio.h>
 void main ( )
   { char *a [3] = { "SUNIL", "KUMAR", "PANDEY"};
     Clrscr ( );
    for (int i = 0, i < = 2; i++)
cout << a [i] << endl;
          getch ( );
              }
```

char a[2][3]

The output of above program is :

```
SUNIL
KUMAR
PANDEY
```

13.8. POINTER TO POINTER

An array of pointer is conceptually same as pointer to pointer type. The pointer to pointer type is declared as follows :

```
data-type ** pointer - variable - name;
```

for example —

```
int **p;
```

where p is a pointer which holds the address of the another pointer.

Example 13.8

```
  # include < iostream.h>
  # include < conio.h>
  void main ( )
     { int data;
       int * p1;
       int **p2;
       data = 15;
       cout << "data = "<< data << endl;
       p1 = & data;
       p2 = & p1;
cout <<"data through p1="<<*p1<<endl;
cout <<"data through p2 = " <<**p2<<endl;
       getch ( );}
```

The output of above program is :

```
data = 15
data through p1 = 15
data through p2 = 15
```

13.9. POINTER TO FUNCTIONS

The general syntax of a pointer to a function is :

```
return-type (*pointer-name) (list of parameter);
```

For example —

```
void (*p) (float, int);
```

In above declaration a pointer to a function returns void and takes the two formal arguments float and int types.

After declaring of a pointer to a function the address of the function must be assigned to a pointer like follow :

```
p = & function - name;
```

where p is pointer variable.

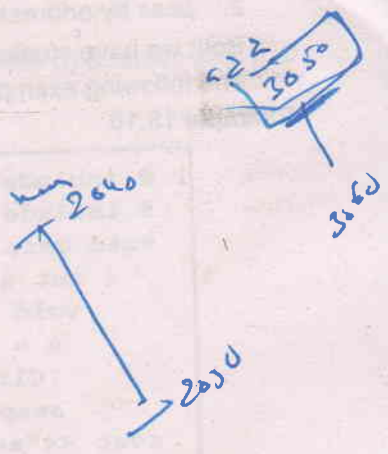
Through pointer to function the function is called as follows :

```
A = (*p) (x,y);
```

where x and y are actual parameters and A is variable in which the function return value. p is a pointer name which points to function. See the following example :

Example 13.9

```
        # include <iostream.h>
        # include <conio.h>
        void main ( )
           { float add (float, float);
        float a, b, s;
      float (*p) (float, float);
        p = & add;
          a = 20.5;
          b = 10.3;
        s = (*p) (a,b);
      cout <<"s = " << s << endl;
        getch ( );
                 }
   float add (float a1, float b1)
      { int sum;
          sum = a1 + b1;
          return (sum);
            }
```



The output of above program is :

```
s = 30.8
```

In above program the statement :

```
float (*p) (float, float);
```

declares a pointer p which is point to function.
The statement →

```
p = & add;
```

assign address of function add to p. i.e. now pointer p point to function add.
The statement →

```
s = (*p) (a,b);
```

calls the function which is pointed by p (i.e. add) and then returned value of function is assigned to s.

Note : Note that if in pointer to function declaration we skip the bracket in which pointer name is written then C++ compiler interprets it as a function which returns a pointer type value. For example if in above example we write —

```
float *p (float, float);
```

this statement means p is a function which returns pointer and the pointer points to float type data i.e. function return a pointer to float.

13.10. POINTER AS AN ARGUMENT TO FUNCTION

If pointer is passed as an argument to a function that is known as pass by address or pass by pointer. This concept is also called call by reference. Actually call by reference concept is interpreted in C++ by two way –

1. pass by reference
2. pass by address

Both we have studied in chapter name function.

The following example interpreted the pass by address concept.

Example 13.10

```
# include <iostream.h>
# include <conio.h>
void main ( )
   { int a, b;
     void swap (int *, int *)
      a = 10; b = 20;
       Clrscr ( );
      swap (&a, &b)
cout <<"a="<<a<<"b="<<b<<endl;
      getch ( );
                    }
     swap (int *a1, int *b1)
        { int t;
t = *a1;
* a1 = * b1;
* b1 = t;
      }
```

The output of above program is

```
a = 20, b = 10
```

If we change the header of function is as follows :

swap (int & a1, int & b1)

then this is called pass by reference see the following example.

Example 13.11

```
   # include <iostream.h>
   # include <conio.h>
   void main ( )
     { int a, b;
       void swap (int &, int &);
         a = 10; b = 20;
          clrscr ( );
           swap (a, b);
cout <<"a=" <<a<<"b = "<< b << endl;
getch ( );
    }
void swap (int & a1, int &b1)
     {int t;
        t = a1;
        a1 = b1;
        b1 = t;}
```

The output of above program is same as of example 13.11 as follows :

```
a = 20, b = 10
```

If we pass the simple variable i.e. not use the pass by reference or pass by address then that is called call by value or pass by value. See the following example :

Example 13.12

```
# include < iostream.h>
# include <conio.h>
void main ( )
  { int a, b;
  void swap (int, int);
   a = 10; b = 20;
    clrscr ( );
    swap (a, b);
cout << "a=" <<a<<"b = "<<b<<endl;
          getch ( ); }
void swap (int a1, int b1)
       { int t;
         t = a1;
         a1 = b1;
         b1 = t; }
```

The output of above program is :

```
a = 10, b = 20
```

Note : By above example we can say that when we pass arguments as a call by value then any change in formal parameter does not effect on the actual parameter but if we pass arguments as a pass by address or pass by reference then any change in formal parameter effect on the actual parameter also.

13.11. POINTER AND STRUCTURE

A pointer to structure is defined as follows :

```
structur - name * pointer - variable;
```

After this the address of any structure variable is assigned to pointer variable as follows :

```
pointer - variable = & structure - variable
```

for example :

```
1. struct A { int a; float b;}
       A a1;
       A *p;
       p = & a1
```

But note that a1 must be A type structure because p is point to A type structure for example :

```
1. struct A {int a; float b;};
  struct B { char a, float b; };
```

```
A *p;
B a1;
p = & a1; //wrong
```

The statement p = & a1; is wrong because a1 is B type structure while p points to A type structure.

The next point is through pointer how we can access the structure variable. See the following :

p → mamber of structure;

for example

```
    1. struct A {int x;
            float y; };
    A a1;
    A *p;
    p = & a1;        // now p points to a1 structure // variable.
    p → x = 10;   // equal to a1.x = 10
    p → y = 20; // equal to a1.y = 20
```

If we pass structure variable to function that is call by value but if we pass address of structure that is pass by address see the following example.

Example 13.13

```
# include <string.h>
# include <iostream.h>
# include <conio.h>
 struct employee
     { char name [20];
       int age;
     long int salary; };
void main ( )
 { void modify (employee *);
   void output (employee);
   clrscr ( );
employee a = {"Rajesh", 26, 8000};
        modify (& a);
        output (a);
        getch ( )
          }
void modify (employee *a1)
      { strcpy (a1-> name, "Pandey");
        a1 → age = 30;
         a1 → salary = 19000;
                  }
void output (employee a2)
      { cout << "Name =" << a2.name << endl;
        cout << " Age = " << a2.age << endl;
        cout <<" Salary =" << a2.salary << endl;
                  }
```