Unit-4

# POINTERS:

Definition:

It is a variable which holds the address of another variable

```
int a = 50; (value)
        ↓
    (2000)(address)
```

→ the above statement instructs the system to find a location for 'a' and keep this value (50) in that location

→ So during the execution of program the value of a is associated with name 'a' and with the address 2000. So we can access the value of a by its name and as well as with address

→ Since the address 2000 again a number, simply we can assign to other variable.

| varible | value | address |
|---|---|---|
| a | 50 | 2000 |
| p | 2000 | 5000 |

By this we can access the value of a by using the value of 'p'. Since it is the address of 'a' and which is pointed to by 'p', and hence gets the name as pointer.

* Pointer operators

There are two operators 1) * (Asterisk)
2) & (Ampersand)

* Declaration of a pointer variable:

syntax: Datatype *pointer variable name;

eg: int *p;

In the above statement int refers to the datatype of the pointer variable but not the value of the pointer, and it is pointed to another variable 'a'

NOTE: & → returns variable address,
* → returns value at the address of a variable

eg int a=10;
int *p;

initialization: The process of assigning the address of the variable to the pointer variable is called pointer initialization.

Syntax: pointername = &variablename;

eg int a=10; (1000)
int *p;
clrscr();
p=&a;

p = &a (1000)
*p = a (10)

Accessing a variable through its pointer :- To access the value of the variable, indirection operator (*) is used.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a=5;
 int *p;
 clrscr();
 p=&a
 printf("\n a = %d", a)
 printf("\n address of a = %u", &a);
 printf("\n value of a = %d", *p);
 printf("\n value of p = %u", p);
 getch();
}
```

'*' can be treated as value at address.

If we are assigning the address of another variable to the pointer variable at that time both variables must declare the same datatype.

i.e., p = &qty;
n = *p;
Then n = qty

a [5] 1000
P [1000] 3000

*p → value at p.
→ value at (&a)
→ value of a

output: 5
1000
5
1000

Advantages of pointers:

1) Pointers are used to reduce the length and complexity of the program.
2) Increases the execution speed thus reduces the execution time.
3) Supports dynamic memory management
4) Pointers provides the manipulation of data structures like stack, queue, linked list etc.
5) Used effectively in handling arrays
6) function can written multiple values via the function arguments (using pointers)

* Valid and invalid operations on Pointers :

(legal) (illegal)

Arithmetic operations using pointers :

-> if $P_1$ and $P_2$ are two pointer variables pointed to two integers. then the valid or legal operations on $P_1$ and $P_2$

valid

* $P1 + *P2$
* $P1 - *P2$
* $P_1 * *P_2$
* $P1 / *P_2$ (there should be space between / and *. otherwise it is taken as comment beginning of comment)

-> we can add an integer to pointer variable and can subtract an integer from pointer variable

eg: $P_1 + 2$
$P1 - 6$
$P1 - P2$ ($P_1$ and $P_2$ are pointed to the elements of same array then it returns the difference between 2 elements)
$P1++$
$P2--$
$Sum += *P1$
$P_1 > P_2$

$P1 == P2$ } relational operators are used while checking strings
$P1 != P2$ }

invalid or illegal operations

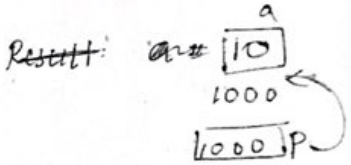
P1 + P2
P1 / P2
P1 * P2
P1 / 5

⇒ arithmetic operations on pointers

program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10;
    int *p, *q;
    clrscr();
    p = &a;
    q = p+2;
    printf("\na = %d", a);
    printf("\np = %u", p);
    printf("\nq = %u", q);
    getch();
}
```

Result a = 10
p = 1000
q = 1004




p+2 = 1000 + 2*2
= 1004

(when an integer is added to the pointer variable the integer value is increased with its scale factor)

*Scale factor: the length of the datatype which is associated it with variable

Eg program

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a=12, b=4, x, y, z;
  int *p1, *p2;
  clrscr();
  p1 = &a;
  p2 = &b;
  x = *p1 * *p2 - 6;
  y = 4* -*p2/*p1 +10;
  printf("\n p1 = %u", p1);
  printf("\n p2 = %u", p2);
  printf("\n a = %d, b = %d", a, b);
  printf("\n x = %d, y = %d", x, y);
  *p2 = *p2 + 3;
  *p1 = *p2 - 5;
  z = *p1 * *p2 - 6;
  printf("\n a = %d, b = %d, z = %d", a, b, z);
  getch();
}
```

Result: x = 12 * 4 - 6 = 42

y = 9

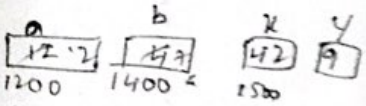
p1 = 1200

p2 = 1400

a = 12, b = 4

x = 42, y = 9

a = 2, b = 7, z = 8




*p1 = 12

*p2 = 4

**4* -4/12 +10 (from left to right)

4*(-4)

-16/12 +10

-1 +10 = 9

*p2 = 4+3 = 7

*p1 = 7-5 = 2

z = 2*7-6 = 8

## *Pointer to Pointer

It is a variable which holds the address of another pointer

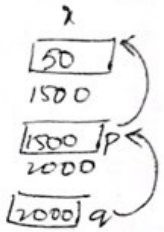
syntax: datatype ** variablename;

eg: int **p;

eg: int x = 50;
int *P, **q;

⇒initialization

eg: int x=50;
int *p, **q;
clrscr();
p = &x;
q = &p;




**q
value @(value @(q))
value @(*p)
value @(1500)
= 50

Programe

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10;
    int *p;
    int **q;
    clrscr();
    p = &a;
    q = &p;
    Printf("a = %d", a);
    printf("a value through pointer = %d", *p);
    printf("a value through pointer to pointer = %d", **q);
    getch();
}
```

Result a = 10
a value through pointer = 10
a value through pointer to pointer = 10

* Pointers and functions (parameter passing mechanism)

→ There are two ways to pass the ~~arguments~~ parameters as arguments from calling function to called function

(1) Call by value (value passed)/(pass by value)

(2) Call by reference (address passed)/(pass by reference)

1) Call by value: The process of passing the variable from calling function to called function is called call by value.

According to callby value mechanism the modification done in the function definition (called function) doesn't effect the arguments of the calling function.

eg:

```
#include<stdio.h>
#include<conio.h>
void swap(int,int);
void main()
{
   int a,b;
   clrscr();
   printf("enter the values of a & b=");
   scanf("%d%d",&a,&b);
   printf("before swapping a=%d, b=%d",a,b);
   swap(a,b);
   printf("after swapping in main function a=%d,
                                  b=%d",a,b);
   getch();
}
void swap(int a,int b)
{
        a=(a+b)-(b=a);
printf("after swapping a=%d, b=%d", a,b);
}
```

## 2) Call by reference :

The process of passing the variables addresses from calling function to called function is called as call by reference.

According to call by ~~value~~ reference machanism the modification is done in the function definition (called function), it must effect the arguments of the calling function

program

```
#include <stdio.h>
#include <conio.h>
void swap(int*, int*);
void main()
{
    int a,b;
    clrscr();
    printf("enter the values of a&b = ");
    scanf("%d %d", &a, &b);
    printf("before swapping a=%d, b=%d", a, b);
    swap(&a, &b);
    printf("after swapping a=%d, b=%d", a, b);
    getch();
}
void swap(int *a, int *b)
{
        *a = (*a + *b) - (*b = *a);
}
```

a&b : 5, 6

a = 5, b = 6

a = 6 b = 5

```
*temp;
*temp = *a; = 5
*a = *b = 6
*a = 6
*b = *temp = 5
```

* Pointers and arrays

→ When an array is declared compiler allocates base address and sufficient amount of storage to contain the elements in the array in contiguous memory locations

→ pointer and one dimensional array

eg)

```
int a[5] = {10, 20, 30, 40, 50};
```

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |

1200 1202 1204 1206 1208

+2 +2 +2 +2 → scale factor

eg) `float a[5] = {1.0, 2.1, 3.2, 4.3, 5.4};`

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 1.0 | 2.1 | 3.2 | 4.3 | 5.4 |

1200 1204 1208 1212 1216

+4 +4 +4 +4 → scale factor

`int *p;` — Base address is the address of the starting element

`p = &a[0];`

`p+1 = &a[1]`

→ using the base address the remaining element address can be calculated by the scale factor

→ Scale factor is the length or size of the data which is associated with the array

eg: `int a[5] = {10, 20, 30, 40, 50};`

```
int *p;
p = &a[0];
p+1 = &a[1];
```

p+2 = &a[2];
p+3 = &a[3];
:
p+i = &a[i];

p+3 = (p + 3 * S.F)
p+3 = (p+3 * 2)
p+3 = (p+6)
= (1200+6)
= 1206

scale factor of the datatype

**&a[i] = p + (i * S.f)**

To get the value of a

*p = value at(p) ⇒ value @ (1200) (&a[0])
⇒ value @ (1200) = 10.

*(p+i) ⇒ *(p+1*S.f)
= val @ (1200+2)
= val @ (1202)
= 20

***(p+i) = a[i]**

example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a[5], i;
  int *p;
  clrscr();
  printf("\n enter the array elements");
  for(i=0; i<5; i++)
  {
    scanf("%d", &a[i]);
  }
  p = &a[0];
  printf("\n the array elements are ");
  for(i=0; i<5; i++)
  {
    printf("%d\t", *(p+i));
  }
  getch();
}
```

Result:

input enter array elements

1
2
3
4
5

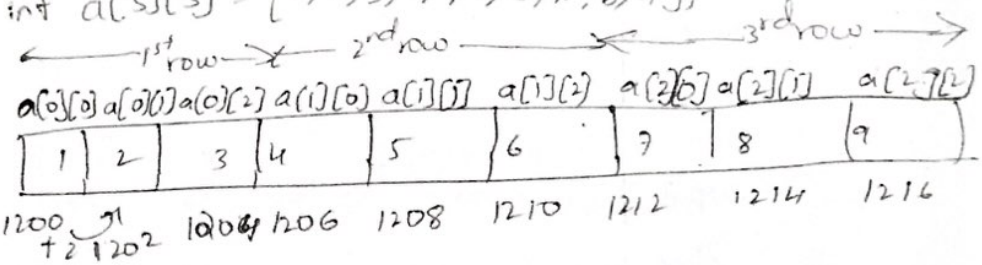
output the array elements are

1 2 3 4 5

## → 2 Dimensional array with pointer

example

```
int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```




```
p = &a[0][0];
```

To get the value

```
*(p+0)
=*(p+(0*sf))
=*(1200+0)
=1
```

```
=>*(p+5)
=>*(p+5*sf)
=>*(p+5*2)
=>*(p+10)
=>*(1200+10)
=>*(1210)
=6
```

```
a[i][j] = *((p+((i*cl size)+j)*S.F.));
```

```
a[2][0] = *(p+(2*3)+0)
        =*(p+6*S.F)
        =*(1200+12)
        =*(1212)
        =7
```

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a[3][3], i, j;
  int *p;
  clrscr();
  printf("\n enter the array elements");
  for(i=0;i<3;i++)
  {
    for(j=0;j<3;j++)
    {
      scanf("%d", &a[i][j]);
    }
  }
    p = &a[0][0];
    printf("\n array elements are");
    for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        printf("%d\t", *(p+(i*3)+j));
      }
      printf("\n");
    }
    getch();
}
```

## * Pointers to functions

'C' language allows a function reffered by the pointer variable.

### Declaration

syntax: Returntype *pointername(list of arg);

eg: int *fact(int);

### function call.

syntax: pointername(list of arg);

Swap(&a, &b);

### called function

Syntax: Returntype *pointername(list of arg)

eg: int *swap(int *a, int *b)

Q) write a 'c' program to find out the largest number of two numbers using pointer to function concept

```
eg: void main()
    {
     int *largest(int *, int *);
     int *p,a,b;
     printf("\n enter the value of a,b);
     scanf("%d %d", &a,&b);
     p=largest(&a, &b);
     printf("\n largest of 2 numbers is %d",*p);
     getch();
    }
```

```
int *largest(int *a, int *b)
{
    if(*a > *b)
      return a;
    else
      return b;
}
```

input a=10
b=20

output: largest value

Note:

when the called function receives the address of a and it will decide the largest number and returns the address of its location and that is assigned to 'p'

In the above program the address of b is return and assigned to p so the output will be the value of b i.e 20

## *void pointer (or) Pointer to void (or) generic pointer

It is a pointer variable which holds the address of any datatype variable or can point to any variable datatype variable.

Declaration

Syntax: ~~datatype~~ void *voidpointername;

eg: ~~int~~ void *vp;

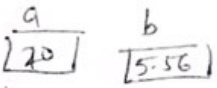
```
eg: int a=20;          float b=5.56;
    ~~int~~ void *vp;      ~~int~~ void *vp;
    vp=&a;             vp=&b;
```

Accessing

*(typecast *) void pointer);

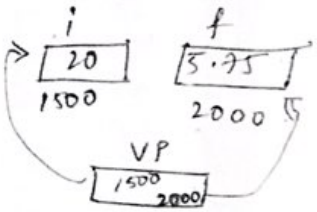
*((int *) VP);

② *((float *) VP);




Program:

```
void main()
{
    int i=20; float f=5.75;
    void * VP;
    clrscr();
    VP = &i;
    printf("\n value of i = %d", *(int *)VP);
    VP = &f;
    printf("\n value of f = %f", *(float *)VP);
    getch();
}
```



* Array of pointers

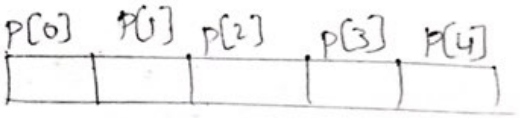
→ It is collection of address(or) collection of pointers

Declaration

datatype *pointername[size];

eg: int *p[5];

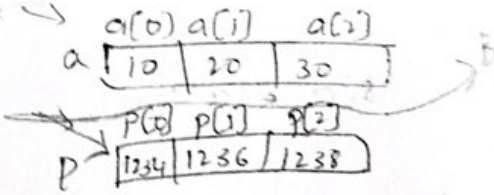
→ It represents an array of pointers that can hold 5 integer element addresses




→ initialization.

'&' is used for initialization

```
eg: int a[3] = {10, 20, 30};
    int *p[3], i;
    for (i=0; i<3; i++)      (or) for(i=0; i<3; i++)
      p[i] = &a[i];                p[i] = a+i;
```




→ Accessing.

Indirection operator(*) is used for accessing

eg: for(i=0; i<3; i++).
    printf("%d", *p[i]);

Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
```

```
int a[3]={10,20,30};
int *p[3],i;
for(i=0;i<3;i++)
p[i]=&a[i];
printf("elements of the array are");
for(i=0; i<3;i++)
   printf("%d\t",*p[i]);
getch();
}
```

## ~~Arrays of strings:~~

## ~~*~~ Arrays of pointers: (to strings)

It is an array whose elements are pointers to the base address of the string.

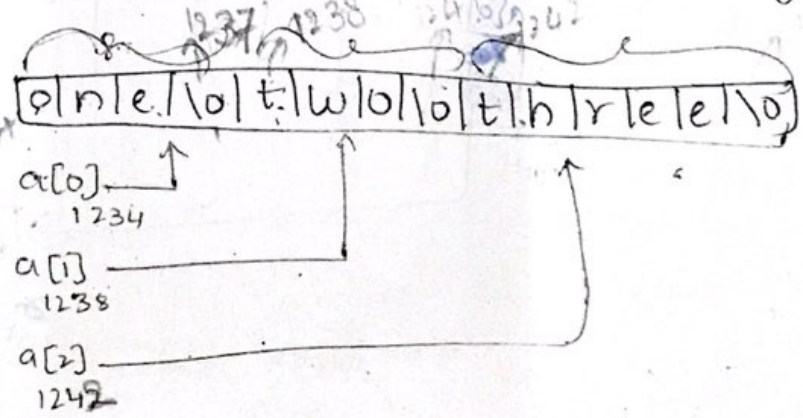
It is declared and initialized as follow:

```
char *a[] = { "one", "two", "three" };
```

Here, a[0] is a pointer to the base address of the string "one"

a[1] is a pointer to the base address of the string "two"

a[2] is a pointer to the base address of the string "three"




Array of pointers

Advantages

Unlike the two dimensional array of characters In (array of strings), in array of pointers to strings there is no fixed memory size for storing The strings occupy only as many bytes as required hence, there is no wastage of space.

program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   char *a[5]={"one","two","three","four","five"};
   int i;
   clrscr();
   printf("the strings are");
   for(i=0;i<5;i++)
     printf("%s",a[i]);
   getch();
}
```