Pointers:

→ A pointer is a variable that can hold the address of other variable, structure & function that are used in program.

→ It contains only the memory location of the variable rather than its content.

→ Pointers can be used to point to the variable such as variable of any basic data type, an array, a function, a structure and union.

Advantages:

* To point to different data structure
* manipulation of data at different memory location is easier
* To achieve ~~clarat~~ clarity & simplicity.
* More compact & efficient coding.
* To return multiple values via functions
* Dynamic memory allocation.

Operators Used with Pointers

There are two basic operators used with pointers

1) The address operator : & (ampersand)
2) The indirection operator : * (asterisk)
   (Value)

Pointer Declaration

→ When a pointer to any variable (of any type) is declared the compiler will allocate memory only for pointer, but not for type to which it points.

→ Both the pointer variable & the variable to which pointer points to, must be declared.

→ A pointer variable must be assigned the address of the declared variable before it is used in program.

Syntax

```
ptr = & ivar
```

where

ptr = a pointer variable
ivar = a variable

here ptr holds the address of variable ivar not value of ivar.

example

```
int x, y, *px, *py;
px = &x;
py = &y;
```

Program To print address & value of any variable

```
#include <stdio.h>
void main()
{
    int ivar=486;
    int *iptr;
    iptr= &ivar;
    printf("Address of ivar = %d \n", iptr);
    printf("The content of ivar = %d \n", *iptr);
}
```

Output

```
Address of ivar = 4056
The content of ivar = 486
```

Program to accept integer & floating value & print their value using pointer

```
#include <stdio.h>
void main()
{
    int x, *iptr;
    float y, *fptr;
    printf("Enter the value of x & y\n");
    scanf("%d %f", &x, &y);
    iptr = &x;
    fptr = &y;
    printf("The address of x = %d & its value is %d\n", iptr, *iptr);
    printf("The address of y = %d & its value is %f\n", fptr, *fptr);
}
```

output

```
Enter the value of x & y
25
66.5
The address of x = 4056 & its value is 25
The address of y = 4058 & its value is 66.500000
```

Pointer Dereferencing

→ The process of referencing the content of variable by pointers is called pointer dereferencing.

→ Steps involved in dereferencing

(a) Address of variable whose value is to be referenced is assigned to a pointer

(b) Then, this pointer points to the value.

Program to accept character & print is along with its address & value pointed by pointer

```
#include <stdio.h>
void main()
{
  char alp, *cptr;
  printf("Enter a character\n");
  alp = getchar();
  cptr = &alp;
printf("The character is %c\n", alp);
printf("Address of alphabet is %d\n", cptr);
  printf("cptr points to %c\n", *cptr);
}
```

Operations on Pointers

1) Assignment

(a) A pointer variable can be assigned address of other variable

eg
```
int *ptr, var;
ptr = &var;
```

(b) If two pointer variables are pointing to the object of same data type they can be assigned

eg
```
int *ptr1, *ptr2;
ptr1 = ptr2;
```

(c) A pointer variable can be assigned a NULL

eg
```
int *ptr;
ptr = Null;
```

2) Addition & Substraction

(a) An integer value can be added to & substracted from a pointer variable

```
int *ptrx;
ptrx = (ptrx+2);
ptrx = (ptrx-1);
```

int *ptr1 *ptr2) *ptr3 ;

ptr3 = ptr2 - ptr1;

Comparison

int *ptr1, *ptr2;

then, (ptr1 == ptr2)

(ptr1 != ptr2)

(ptr1 < ptr2)

(ptr1 > ptr2)

Pointers & Function

→ Like other variables pointers can also be passed to the function.

Call by Reference

→ In this mechanism, by which pointers can be passed as arguments to the function

When the pointers are passed as argument, then following two points must be considered

1) In the calling program, the function is invoked with the function name & address of actual parameter enclosed within parenthesis

ie function_name (&var1, &var2 - - - &varn)

2) In the called program, parameter list, each and every formal parameter (pointer) must be preceded by value operator (*)

ie datatype function_name (*var1, *var2 - - *varn)

Program to show call by reference

```
# include <stdio.h>
Void main()
{ int num1, num2;
  int inter_change (int *n1, int*2);
  printf ("Enter two numbers \n");
  scanf ("%d %d", &num1, &num2);
  printf ("num1 = %d and num2 = %d (before) \n", num1, num2);
inter_change (&num1, &num2); // function call
  printf ("num1 = %d & num2 = %d (after) \n", num1, num2);
}
  int inter_change (int *n1, int *n2)
{ int temp;
    temp = *n1;
    *n1 = *n2;
  *n2 = temp;
}
```

Pointers & Array

The name of an array itself designate some memory location & this location in the memory is the address of the very first element of the array. The address of first element of array can be written as:

& array[0] or simply array

address of second element

& array[1] & so on

Program to find sum of statically declared 5 elements using pointers & function

```
#include <stdio.h>
void main()
{
  int array[5] = {200, 400, 600, 800, 1000};
  int addnum (int *ptr);
   int sum;
    sum = addnum (array);
    printf ("Sum of all array elements = %d\n", sum);
}
int addnum (int *ptr)
{
  int total=0, i;
  for (i=0; i<5; i++)
     total += *(ptr + i);
  return (total);
}
```