Pointer:

Pointers are variables that contain the memory address for another variable. A pointer is defined like a normal variable, but with an asterisk before the variable name. The type-specifier determines what kind of variable the pointer points to but does not affect the actual pointer.

Pointers concept is introduced in the C language to support the Different Data Structures which is the main feature of the C Language.

Pointer Declaration:

Syntax:

type \*P\_Variable;

In the Above declaration the P\_Variable is a pointer variable which stores the address of the variable with specified type.

The difference b/w normal and pointer declaration is, we include a \* before the pointer variable.

Eg:

int \*i1;

float \*f1;

Pointer Initialization:

We use two pointer operators named, \* (pointer), and & (Address of) operators.

int var=10;

int \*i1;

The address operator causes the memory address for a variable to be returned. It is written with an ampersand (&) sign before the variable name.

You need to remember that the pointer variable of the specific type can only point to(stores the address of the) variable of the same type.

For example in the above declaration, the i1 pointer of the type int can only point the variable of the type int.

i1 = &var;

the above statement is the initialization of the pointer i1 with the address of the var.

address

var 10 --- 2000

i1 2000 --- 3020

If you use a pointer operator \* we can represents the value in address of pointer.

printf("%d is the value in the address %p",\*i1,i1);

Pointer to arrays:

Array variable itself holds the address of the first element. so we can use the array variable to initialize the pointer to point an array.

eg:

int iarray[20],\*ip;

ip = iarray;

if 'ip' pointing to the array then it may represents the elements in the array by specifying the index as following,

\*(ip + index)

pointers to strings:

pointers that are pointing to the character array itself represents the string.

Pointers to string makes the programmer easy to work with the strings.

eg:

char str[10],\*cp;

cp = str;

scanf("%s",cp);

You can directly make the string operation to the character pointer,

char \*cp;

gets(cp);

puts(cp);

the above statements will read and print the string using the character pointer.

in order to represents the character in the string we use,

\*(cp + index) as like we used in the int arrays.

Arrays passing to functions:

If you want to pass an array to the function argument we can use the pointer to that array to represent it.

eg:

main()

{

int arr[10],i;

printf("enter elements:\n");

for(i=0;i<10;i++)

scanf("%d",arr[i]);

sortarray(arr,10);

getch();

}

sortarray(int \*iparray,int size)

{

sorting procedure...

}

Pointers to the Functions:

Functions can also be represented with a pointer. A function pointer is defined in the same way as a function prototype, but the function name is replaced by the pointer name prefixed with an asterisk and encapsulated with parenthesis. Such as:

int (\*fptr)(int, char);

fptr=some\_function;

To call this function:

(\*ftpr)(3,'A');

This is equivalent to:

some\_function(3,'A');