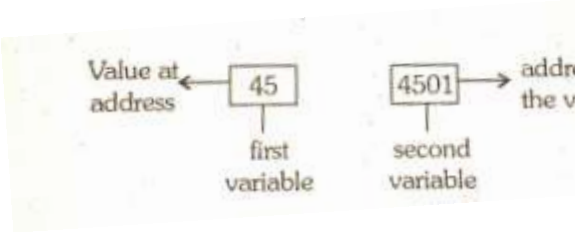
Pointers

One of the most important and powerful features in C language is pointer.

"A pointer is a variable, it may contain the memory address of the another variable." pointer can have any name that is legal for other variable. It is declared in the same manner like other variables. It is always denoted by * operator.

A pointer is a variable whose value is, also an address. Each variable has two attributes; address and value. A variable can take any value specified by its data type.



• A pointer to an integer is a variable that can store the address of that integer.

• Second the value contained by a pointer must be an address which indicates the location of another variable in the memory.

So pointer is called as address variable.

Features of pointers:

a) Pointers are efficient in handling data and associated with array.

b) Pointers are used for saving memory space.

c) Pointers reduce length and complexity of the program.

d) Pointer helps to make letter use of the available memory.

e) Since the pointer data manipulation is done with address, the execution time is-faster.

f) Two-dimensional and multidimensional array representation is easy in pointers.

g) Pointers allows for references to function, this may facilitate passing of function as arguments to other functions.

Pointer declaration:

Syntax data-type * pointer name ;

Example:

int *a

char *b

float *c

int *a - means 'a' contains the address of variable, which is integer variable.

char *b - means 'b' contains the address of variable, which is character variable.

float *c - means 'c' contain the address of variable which is float variable.

Accessing variable through pointers:

Example:

int *p

x = 15 ;

p = & x

Variable	Value	Address
x	15	4001
p	4001	4005

Program to access through variable pointer:

```
# include < stdio.h>
main ( )
{
int a = 22, *a ;
float b = 2.25 , *b ;
a = & a ; b = & b ;
printf ( " \ n value of a = % d", * a) ;
printf ( " \ n value of b = % d", * b) ;
}
```

Output:

Value of a = 22

Value of b = 2.25

}

Null pointer:

A pointer is said to be a null pointer when its right value is 0. A null pointer can never point to a valid data. For checking a pointer, it it is assigned to 0, then it is a null pointer and is not valid.

Example:

int *a ;

int *b ;

b = a = 0

Hence b and a become null pointers after the integer value of 0 is assigned to them.

Pointer to Pointer:

Pointer is a variable that contains the address of the another variable. Similarly another pointer variable can store the address of this pointer variable. So, we can say, this is a pointer to pointer variable.

Pointer Expressions:

Pointer variables can be used in expressions.

Example:

c = *a + *b = ⇒ c = (*a) + (*b) ;

s = 10* - *a / *b ⇒ s = 10* (-(*a))/(*b) ;

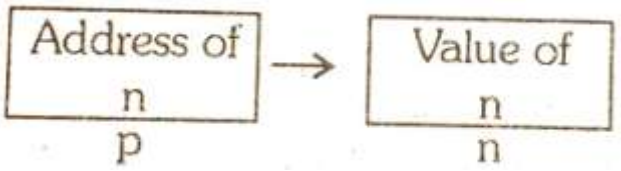
Initializing pointer variable:

The process of assigning the address of a variable to a pointer variable is known as initialization. The location of the variable in memory depends on system memory. This can be achieved in ‘c’ through an ampersand (&) sign. The ampersand is an address operator, which is used to access the address of a variable and assign it to a pointer to initialize it. The programs with uninitialized pointers will produce errorneous results. It is therefore important to initialize pointer variables carefully before they are used in the program. We can use the assignment operator to initialize the pointer variable.

Example:

p = & n

where ‘p’ contains the address of variable ‘n’.



Pointers and Functions:

The pointer can be used as arguments in functions. The arguments or parameters to the function are passed in two ways.

i) call by value

ii) call by reference

Call by value:

The process of passing the actual value of variables is known as “call by value". When a function is called in program, the values to the arguments in the function are taken by the calling program, the value taken can be used inside the function. Alteration to the value is not accepted inside the function in calling program but change is locally available in the function.

Call by Reference:

The process of calling a function using pointers to pass the addresses of variables is known as call by reference. A function can be declared with pointers as its arguments. Such function are called by calling program with the address of a variable as argument from it.

The address of the variables are altered to the pointers and any changes made to the

value inside the function is automatically carried out in the location. This change is accepted by the calling program.

Pointers and Arrays:

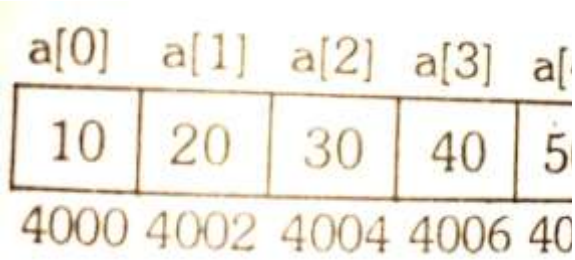
Array is a collection of similar data type elements stored under common name. When we declare an array the consecutive memory locations are located to the array of elements.

The elements of an array can be efficiently accessed by using pointers.

Example:

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

a[5] means the array 'a' has 5 elements and of integer data type.



The base address of the array start with 0th element of the array. The array is an integer type, the integer will have 2 bytes and hence the address element is incremented by 2.

Example:

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

int * b ;

b = a

'b' is a pointer variable which holds the base address the array 'a' i.e. b*= &a[0]

When pointer variable is incremented by 1 then the related base address will be incremented by address +2 because pointer is of type integer. Hence the next address element increase by 2 till the size of an array.

Pointers with Multi - dimensional array:

A multidimenstional array can also be represented with an equivalent pointer notation. A two dimensional array can be considered as a collection of one - dimensional arrays.

Syntax:

data-type (*pointer variable) [Expression 2].

If 'a' is a two dimensional integer array having 10 rows and 20 columns, 'a' can be declared as

pointer ← int (*a) [20] ; rather than

array ← int a[10][20] ;

Array of pointers:

The array elements can be accessed by two methods.

- Standard array notation
- Pointer arithmetic

Array rotation is a form of relative addressing. The compiler treats the subscripts as a relative offset from the beginning of the array. When a pointer variable is set to the start of an array and is incremented to find the next element, absolute addressing is being implemented. Therefore, using pointers have the following advantages.

- Accessing arrays through pointers is marginally faster than using array notation.
- Pointers give better control over array processing.
- Pointers allows u$ to perform certain functions that are extremely difficult with array notation.

Pointers and Strings:

i) Character and Pointer:

A character pointer is a pointer to the character. It can be declared as

char * pointer - character ;

A character pointer can also be used to access character arrays and string constants, in the same way as accessing numeric arrays using their respective pointer variable.
A character type pointer variable can be assigned an entire string as part of a variable declarations whereas a numerical pointer variable cannot be initialized in the same manner as a numerical array.

ii) Strings and Pointers:
A string can conveniently be represented by either using a one - dimensional character array as a character pointer.
A string constant is accessed by a pointer to its first element.
Char pointers message ;
The statement pointer message = "I an a student" allocates memory and stores the character string and then assigns to pointer message a pointer containing the starting address of the character array.

Pointers & Structures:
Pointer pointing to a structure is known as structure pointers,

```
struct
{
member 1 ;
member 2 ;
} variable, * ptr ;
```

Here the variable represents structure type variable and *ptr represents the name of the pointer.

Programs:
1. Program to interchange the values stored in two variable using function with pointers.
Answer:

```
# include < stdio.h >
main ( )
{
void interchange (int *, int * ) ;
int a = 39, b = 77 ;
printf ( " \ n Before Interchange: a = %d,
b = %d , n", a, b) ;
interchange (& a, & b) ;
printf ("After interchange: a = %d,
b = %d \ n", a, b) ;
}
Void interchange (int 'a, int *b)
{
int t ;
t = * a ;
*a = *b ;
* b = t ;
}
```

Before interchange: a = 39, b = 77
After interchange: a = 77, b = 39

Dynamic memory allocation in 'C':
Dynamic memory allocation means, a program can obtain its memory while it is running. It allows us to allocate additional memory space or to release unwanted space at the time of program execution (runtime).
Pointers support the dynamic memory allocations in Q language. The C language provides 4 library functions known as 'Memory management function' which can be used for allocating and releasing memory during execution.

Function	Meaning
Malloc ( )	used to allocate blocks of memory in required size of bytes.
Free ()	used to release previously allocated memory space.
calloc ( )	used to allocate

	memory space for an array of elements.
realloc ()	used to modify the size of the previously allocated memory space

Static Memory Allocation:

The static memory is allocated during compilation time. When a variable is declared, based on their type, compiler allocates space in memory, these variables can be indirectly accessed with the help of pointer variable by assigning the address of the variable to the pointer variable. This way of assigning the pointer variable is called static memory allocation.

Example:

int m1, m2, *p = &m2 ;

* p = 4

When the statement is encountered the compiler allocates space in memory for ml, m2 and p. The value of ml is unknown and p holds the address of m2 and indirectly the value 4 is assigned to m2.

Program to find the sum of two numbers using pointers

```
# include < stdio.h >
# include < conio.h >
main ( )
{
int a, b, c ;
int *a, *b ;
printf ( "\ n Enter two integer value" ) ;
scanf ("%.d %d", &a, &b) ;
*a = &a ;
*b = &b ;
c = *a + *b ;
printf ( " \ n sum of two integer is = %d ", c)
;
}
```

Output: Enter two integer value: 20 30

Sum of two integer is = 50