POINTERS :-

Introduction :

* A pointer is also derived datatype in C-language.
* Every variable will have the memory locations.
* To access the address of a variable, we use '&' operator.
* In General, the computers, memory is a sequential collection of storage cells (set of bytes) & are represented through 'address'.
* The address, depends on the compiler or machine ie, randomly generated.

→ Definition of Pointer :

"A Pointer is a memory variable that stores a memory address of another variable".

* A pointer type variable is denoted by '*' operator.
* A pointer variable follows same rules of Identifiers.

~~For~~ Syntax : datatype *pointervariable;

Eg : int *a;
char *ch;
float *f;
} → Declaration of "Pointers".

Advantages :

1. Pointers reduces the memory space.
2. Using pointers, the execution time will be fast.

Scanned by CamScanner

Scanned by CamScanner

because pointer variables have direct access to memory locations & if value is changed, it automatically gets updated in the address mentioned.

3. It is fast to access using pointers ie, Efficient coz, once the program / variables use is over, we can release the memory allocated during execution ie, by using Dynamic Memory allocations.

4. Pointers can be used to return multiple values at a time from a function via function arguments.

5. It is also efficient for using data structures like structures, linked Lists, Queues, Stacks & etc...

i, Declaration of pointer variables:

* The general form to declare pointer variable is shown below:

syntax: datatype *pointervariable;

Eg: ~~int~~ char *a;
int *b;
float *f;

* The symbol '*' (asterisk) tells that the 'pointervariable' can be any name given by user which holds only the address of other variable, which is of its data type.
* Pointervariable will also have a memory locations.

Scanned by CamScanner

* In the first example, 'a' is an character pointer which tells the compiler to hold the address of other character variable.

* Here, '*' is the Indirection operator & also called as Dereference Operator. It, when a pointer is dereferenced, the value at that address stored by the pointer is retrieved.

* Normal variables can access directly their own values ie, a=5, where as a pointer provides indirect access to the values of the other variable.

* Compiler will easily understand; whenever user entered '*' as an multiplication operator or a pointer type.

* The '%u' format specifier is used to display the address of a variable

(ii) Accessing of a pointer variable:

Eg: int a=5, *p;

p=&a;

printf("%d", *p);

int a=5, *p;
p=&a;
printf("%d", *p);

This statement is used refer 'p' to 'a' variable by holding the address of 'a'. We can't do anything with the pointer until we assign the address of a variable.

* Accessing of a pointer variable is done using '*' variable

Scanned by CamScanner

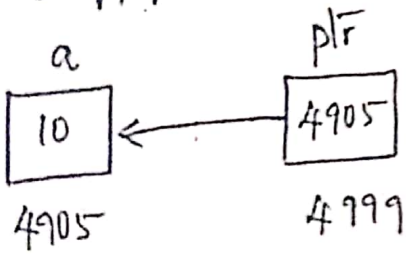
Scanned by CamScanner

iii, Initialization :

* The process of assigning the address of a variable to a pointer variable is known as Initialization.
* By using the address operator(&), we can initialize the pointer with the following statement.

```
ptr = &a;
```

* After initializing, 'ptr' points to 'a'.



* Here, 10 — value of a = a
4905 — Address of a = &a
4905 — value of ptr = ptr
4999 — Address of ptr = &ptr
10 — Value of *ptr = *ptr.

// Program for demonstrating the 'Pointers.'

```
#include<stdio.h>
void main()
{
    int a,*ptr;
    a=10;
    ptr=&a;
    printf("Value of a = %d\n",a);
```

Scanned by CamScanner

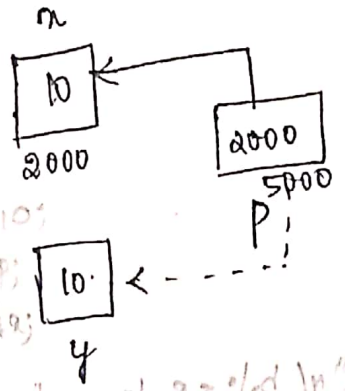
Scanned by CamScanner

```
printf("value of a using ptr = %d\n", *ptr);
printf("Address of a = %d\n", &a);
printf("Address of a using ptr = %d\n", ptr);
}
```

Output: Value of a = 10
Value of a using ptr = 10
Address of a = 4905
Address of a using ptr = 4905

// Program for accessing the pointer variables:

```
#include<stdio.h>
void main()
{
    int x, y, *p;
    x = 10;
    p = &x;
    y = *p;
    printf("value of x = %d\n", x);
    printf("value of *p = %d\n", *p);
    printf("value of y = %d\n", y);
}
```



Output: Value of x = 10
Value of *p = 10
Value of y = 10

Scanned by CamScanner

Scanned by CamScanner

// Program to show pointers size of any data type

```
#include<stdio.h>
void main()
{
    printf("Size of char pointer = %d\n", sizeof(char*));
    printf("Size of int pointer = %d\n", sizeof(int*));
    printf("Size of float pointer = %d\n", sizeof(float*));
}
```

output: Size of char pointer = 4
Size of int pointer = 4
Size of float pointer = 4

Note: 1. All int, char, float pointers will be of same size.

2. If 32-bit system, pointer will occupy 4 bytes, & in 64-bit system, pointer will occupy 8 bytes because the memory size/location depends on the address width allocated by computer.

3. `printf("%d", *ptr);` can also be written as:

`printf("%d", *(&a));`

// Program that use two pointer variables to access two different values.

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

Scanned by CamScanner

Scanned by CamScanner

```
int a, b, *p, *q;
a = 10;
b = 5;
p = &a;
q = &b;
printf("The value of b = %d\n", *p);
printf("value of b = %d\n", *q);
}
```

Output: The value of a = 10
value of b = 5

→ Operations on Pointers:

* Pointer is a standard data type. It just takes the data type of the variable to which it is pointing..
* The following two operations are permitted on pointers:
  i) Pointer arithmetic
  ii) Pointer comparison.

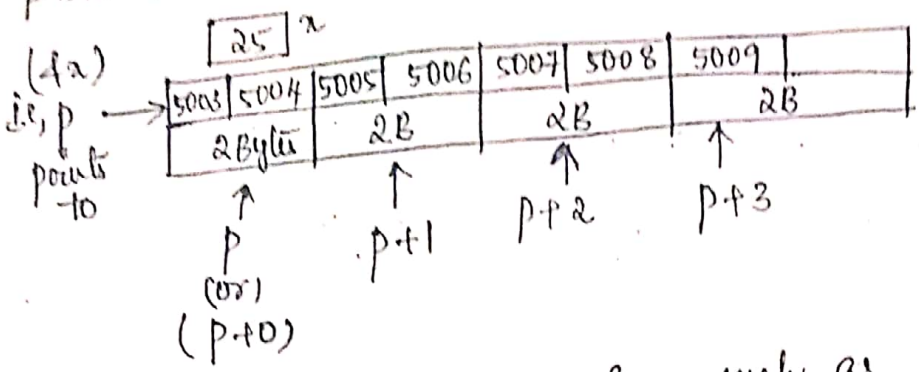
i) Pointer Arithmetic:

* We can only perform addition & substraction on pointers, which is called as "Addres arithmetic.
* Eg: If the statement p = p + 3 is executed in the following case: int x = 25, *p;
  p = &x;
  p = p + 3;

Scanned by CamScanner

Scanned by CamScanner

then it gives a garbage value. because, if p is assigned with address of 'x' ie, p = 5003 & if p = p+3 means p will be 5009.



* We can write few expressions such as

++p, *p+10, *p*10+x, ++*p, --*p, p+K, p-K, --p

where K is an integer / constant

Eg: (i) ++p // Increments the address of 'p' depends on datatype

(ii) --p // decrement address of 'p' depends on datatype

(iii) ++*p // Increment the value of x by 1

(iv) --*p // Decrement the value of x by 1

(v) *p+10 // adding 10 to x

(vi) *p*10+x // same as x*10+x

(vii) p+K // The address of x + K, constant

Note: Invalid cases are multiplication, division (ie, $p_1 * p_2$ or p_1 / p_2). But $(*p_1)*(*p_2)$ is valid
(by address) (by values).

Scanned by CamScanner

Scanned by CamScanner

//write a program to add 2 no's through variables & their pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{ int a, b, *p, *q;
  printf("Enter 2 nos:");
  scanf("%d%d", &a, &b);
  p=&a; q=&b;
  printf("sum of a+b vars=%d",
          a+b);
  printf("\nsum of a+b using
      pointers = %d", *p+*q);
}
```

Op: Enter 2 nos: 2 5
Sum of a+b vars = 7
Sum of a+b using pointers = 7

*Write a program to show the effect of increment & decrement with pointer variables */

```
#include<stdio.h>
void main()
{
  int a, *p;
  puts("Enter a number:");
  scanf("%d", &a);
  p=&a;
  printf("Address = %u\n", p);
  printf("Address = %u\n", p++);
  printf("Address = %u\n", ++p);
  printf("Address = %u\n", --p);
  printf("Address = %u\n", p--);
  printf("Address = %u\n", p);
}
```

Output: Enter a number: 4
Address = 4062
Address = 4062
Address = 4066
Address = 4064
Address = 4064
Address = 4062

/* Write a program to perform different arithmetic operations using pointers */

```
#include<stdio.h>
void main()
{
  int x=25, y=10, *p, *q;
  p=&x;
  q=&y;
  printf("Addition = %d", *p+*q);
  printf("Subtraction = %d\n", *p-*q);
  printf("Division = %d\n", *p/ *q);
  printf("Multiplication = %d\n", *p**q);
  printf("Modulo Div = %d\n", *p%*q);
}
```

Output:
Addition = 35
Subtraction = 15
Division = 2
Multiplication = 250
Modulo Div = 5

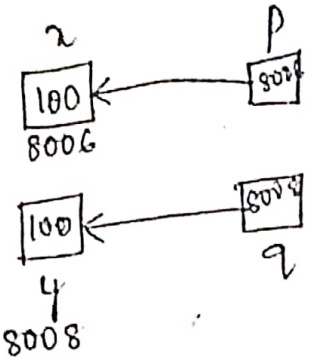
Scanned by CamScanner

Scanned by CamScanner

ii) Pointer comparisons

* Two pointers can be compared using relational expressions. (iff these two pointers are of same data type)

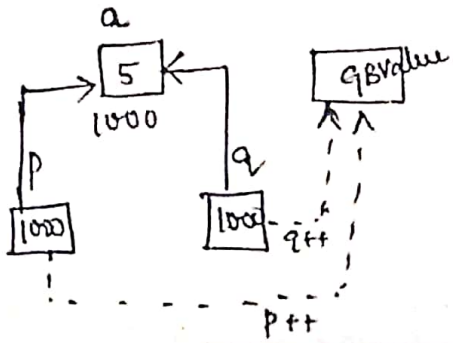
Eg:
```
int x, y, *p, *q;
x = 100; y = 100;
p = &x;
q = &y;
if(*p == *q)
    printf("Pointer comparison is accepted");
else
    printf("Comparison is not accepted");
```



Note: if(p == q) means it compares the addresses of x & y. which will not be same

// Write a program to show the effect of increment & decrement on pointers variables & compare the addresses & display of it

```
#include<stdio.h>
void main()
{
    int a=5, *p, *q;
    p = &a;
    q = &a;
    p++; q++;
    if(p==q)
        printf("Both addresses are same");
    else
        printf("Both addresses are not same");
}
```



Scanned by CamScanner

Scanned by CamScanner

Output: Both addresses are same.

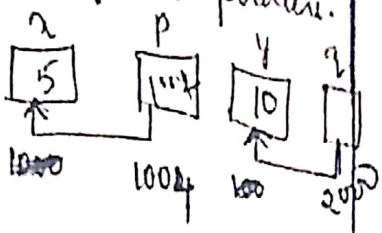
Note:

1. We can compare addres or values of two pointers.

Eg: int x=5, *p, y=10, *q;

p = &x;

q = &y;



p > q // Expression evaluates to false. because x variable is stored before y

q > p // True because 'y' is stored after 'x'

p+1 == q // True

p == q // False

p != q // True

Program, Examples:

/* Program to illustrate the use of indirection operator '*' to acess the updated value pointed by a pointer */

```
#include<stdio.h>
void main()
{
    int x=25, *p;
    p=&x;
    printf("value = %d\n", *p);
    *p = *p + 10;
    printf("value of x = %d\n
      value of x using p = %d", *p, *p);
}
```

output:

value = 25

value of x = 35

value of x using p = 35

Scanned by CamScanner

Scanned by CamScanner

Eg: // Program to compute the factorial of a given no using pointers.

```
#include<stdio.h>
void main()
{   int n, *p, i=1;
    long int f=1;
    //p=&n;
    printf("Enter the no:");
    scanf("%d",&n);
    for(p=&n; *p>=0; *p--)
    {
        f=*p*f;
    }
    printf("factorial = %ld",f);
}
```

Output:

```
Enter the no:4
Factorial = 24
```

→ Pointers and Arrays:

1-D array:

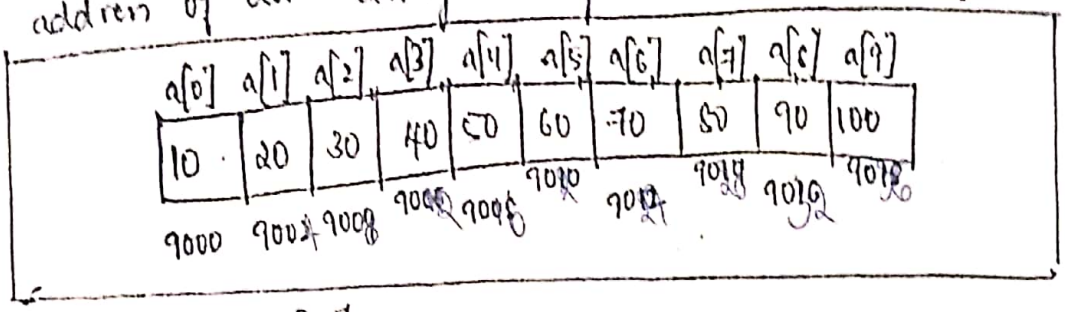
* An Array is a collection of similar data elements refered by unique name
* Array name by itself is an address or pointer. Here, it points to the address of the first element of an array. (ie, 0th element).
* The set of array elements with their addresses can be used / referred using array name itself.
* These elements are always stored in contiguous memory locations.
* Any operations by arrays can also be done with pointers.

Scanned by CamScanner

Scanned by CamScanner

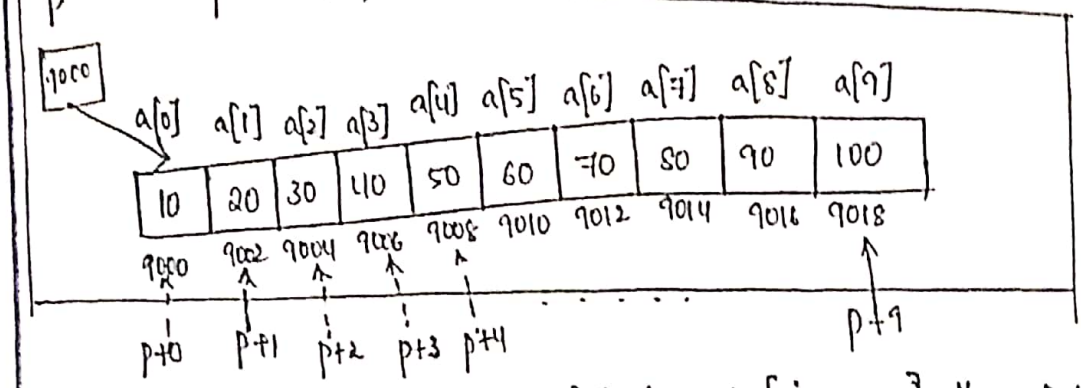
Examples: ① int a[10];

* Here array takes 10 memory allocations, with 2 bytes for each element in contigous manner. So the base address of an array is represented as below:



② int a[10], *p;

p = &a[0];



Here p is pointing to a[0] element [i.e, p+0] then p+1 points to the next element.

* Eg: 1) If p points to a[5], then p+1 points to a[6] or p-1 points to a[4].

* 2) p = &a[0]; (or) p = a;

Here *p refers to a[0] element = 10.

*(p+1) refers to a[1] element = 20

*(p+2) refers to a[2] element = 30 & so on.

* 3) a[0] is similar to *(a+0)

p = &a[0]; can access either by *(a+0) or *p

Scanned by CamScanner

Scanned by CamScanner

// Program which reads array elements & display using pointers.

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

Output: Enter 5 numbers: 5 10 15 20 25
The array elements are: 5 10 15 20 25

(or)

```
void main()
{
    int a[5], i;
    printf("Enter ele:");
    for(i=0; i<5; i++)
        scanf("%d", a+i);
    printf("The array elements are");
    for(i=0; i<5; i++)
        printf("%d\t", *(a+i));
}
```

(or)

```
void main()
{
    int a[5]={5,10,15,20,25}, *p, i; p=&a[0];
    for(i=0; i<5; i++, p++)   (or)   for(; *p!=0; p++)
        printf("%d\t", *p);
}
```

Output: 5 10 15 20 25

// Program to display sum of array elements using pointers.

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

Scanned by CamScanner

Scanned by CamScanner

→ Pointers & Arrays:

* Array name itself is an address or a pointer.
* It always points to the address of the first element (i.e, 0th element of an array).
* The elements of array can be displayed by using array name itself.
* Here, array elements are always stored in contigous memory locations.

// Program to display elements of an array (array itself as an array).

```
#include<stdio.h>
void main()
{
    int a[5] = {1,2,3,4,5}, i;
    printf("The array elements are:\n");
    for(i=0; i<5; i++)
        printf("%d\t", *(a+i));
}
```

Output: The array elements are:

1 2 3 4 5

* Here, when array itself declared / used as an array, &a[i] can be represented as (a+i) which means the based address of i^{th} element of an array.
* To consider the value of i^{th} element, a[i] which can be represented as *(a+i). ~~which~~ ~~is~~ from an array.

Scanned by CamScanner
Scanned by CamScanner

(or)

```
#include<stdio.h>
void main()
{
    int a[10], i;
    printf("Enter 5 elements:");
    for(i=0; i<5; i++)
        scanf("%d", (a+i));
    printf("The array elements are:\n");
    for(i=0; i<5; i++)
        printf("%d\t", *(a+i));
}
```

Output: Enter 5 elements: 11 15 23 25 31

The array elements are:

11 15 23 25 31

* To access the array values, different notations can be considered using pointer.

i.e., a[i]
*(a+i)
*(i+a)
i[a]

} Notation of i^{th} element of an array.

* To represent address, the notation of i^{th} element address are (a+i) (or) &a[i] (or) &i[a]

// Program to display the sum of n array elements. Use array name itself as a pointer.

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

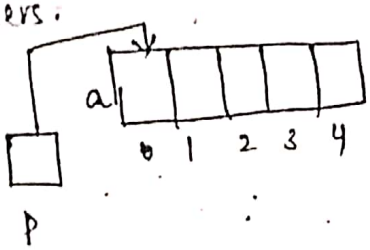
Scanned by CamScanner

Scanned by CamScanner

```
int a[] = {1, 2, 3, 4, 5}, i, sum = 0;
for (i = 0; i<5 ; i++)
        sum = sum + *(a+i);
printf(" Sum of 5 elements = %d\n", sum);
}
```

Output: Sum of 5 elements = 15

// Program to read elements of an array and access them through a pointers.




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

Output: Enter 5 elements : 13 52 62 89 10

The array elements are :

13 52 62 89 10

(on

Scanned by CamScanner

Scanned by CamScanner

```
#include<stdio.h>
void main()
{   int a[5]={1,2,3,4,5},*p,i;
    p=&a[0];
    printf("Array elements are:\n");
    for(i=0; i<5; i++, p++)
        printf("%5d",*p);
}
```

Output: Array elements are:

1 2 3 4 5

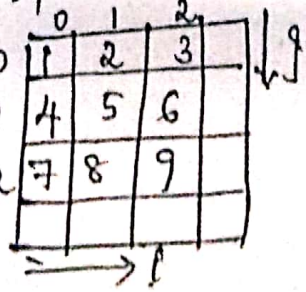
* Here, the pointer 'p' is pointing to each element address for each iteration
* To display all elements in reverse order:

```
p=&a[4];
for(i=0; i<5; i++, p--)
    printf("%5d",*p);
```

Pointers and Two-Dimensional arrays:

* A Matrix is also represented as 2D-elements of an array.
* We create a two dimensional integer array 'a' having 3 rows & 3 columns.

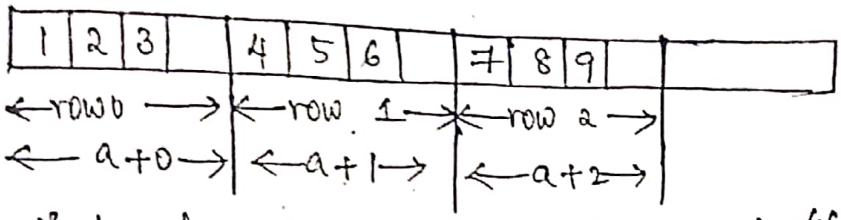
ie, int a[3][4]={{1,2,3},{4,5,6},{7,8,9}};



Scanned by CamScanner

Scanned by CamScanner

* In the above image, based on this array, the compiler will allocate the memory in row-wise.



* Here, first element of array 'a' = 1 ie, *(*(a+0)+0)
* If address of 1st element of an array is 1000, then row 0 will have 1000, 1004, 1008, 1012. & row 1 will have 1016, 1020, 1024, 1028, & row 2 will have 1032, 1036, 1040, 1044.

Declaring array itself as a pointer:

```
#include<stdio.h>
void main()
{  int a[5][5], i, j;
   printf("Enter matrix elements:");
   for(i=0; i<3; i++)
       for(j=0; j<3; j++)
           scanf("%d", &a[i][j]);
   printf("The matrix elements are:\n");
   for(i=0; i<3; i++)
   {  for(j=0; j<3; j++)
      {  printf("%d\t", *(*(a+i)+j));
      }
      printf("\n");
   }
}
```

Scanned by CamScanner

Scanned by CamScanner

Array of pointers :

* Array of pointers is nothing but a collection of addresses.
* Here we store address of variables for which we have to declare an array as a pointer.
* When we have 4 integer pointers. If creating seperate pointer variables is done, accessing is not done coz of accessing. So we can create one single integer array of pointers 'ptr' variable that will point at the four variables.

Eg: int *ptr[4];

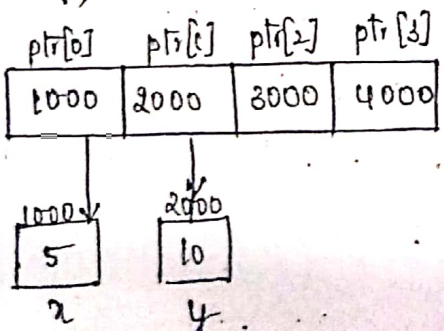
i, Assigning address to array of pointers :

* We get the address of a variable using the address of '&' operator & then save that address in the array of pointers.

Eg: int *ptr[4], x = 5, y = 10;

ptr[0] = &x;

ptr[1] = &y;



* Here ptr[0] holds the address of 'x' & ptr[1] holds the address of y variables & each pointer will have 4 bytes ie, 8000 to ptr[0] & 8004 to ptr[1].

Scanned by CamScanner

Scanned by CamScanner

* To access the value of the variables via array of pointers, we have to use the value at the address of '*' operator.

// Program to demonstrate the array of pointers

```
#include<stdio.h>
void main()
{
    int x = 5, y = 10, z = 15, *p[5];
    p[0] = &x;
    p[1] = &y;
    p[2] = &z;
    printf("x = %d\n", *p[0]);
    printf("y = %d\n", *p[1]);
    printf("z = %d\n", *p[2]);
}
```

Output:

x = 5
y = 10
z = 15

→ Pointer to structure:

* The Pointer is a variable that holds the address of another variable, which can be of any type
* We can define a pointer to structure also.

Scanned by CamScanner

Scanned by CamScanner

* Here it stores only the address of starting member variable. Thus such pointers are called as "structure Pointers".

Eg:
```
struct Book
{
    char name[20];
    int pages;
};
struct Book *b;
```

* '*b' variable is a pointer variable points to structure Book.

* To access the structure members which are of pointer structure variable, can be done using '→' operator. ie,

b → name
b → pages.

// Program that illustrates the use of pointer to structure

```
#include<stdio.h>
void main()
{
    struct Book
    {
        char name[30];
        int pages;
    };
    struct Book b1={"C and DS", 856};
    struct Book *b;
```

Scanned by CamScanner

Scanned by CamScanner

```
    b = &b1;
    printf("Book title = %s\t Pages = %d\n", b1.name,
                                    b1.pages);
    printf("Book title = %s\t Pages = %d\n",
                    b->name, b->pages);
}
```

Output:

Book title = C and DS Pages = 856

Book title = C and DS Pages = 856.

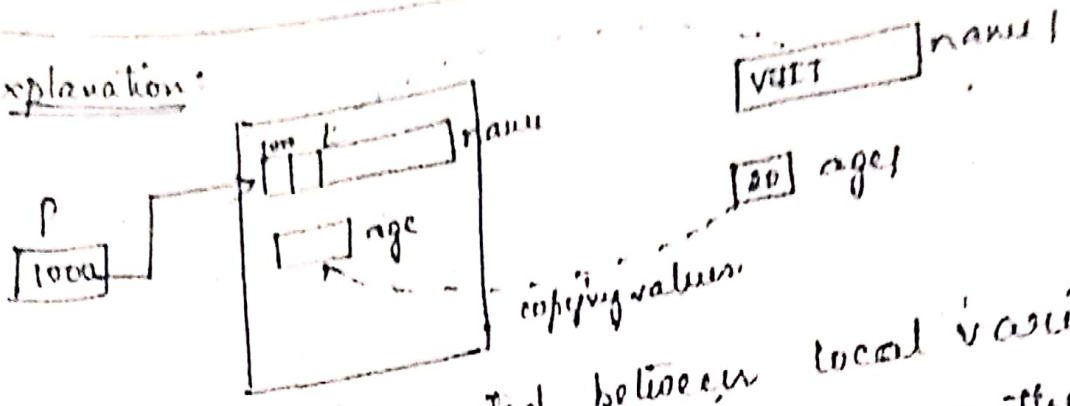
Program to declare pointer as member of structure & display the person details.

```
#include<stdio.h>
struct Person
{
    char *name;
    int *age;
};
void main()
{
    struct Person *p;
    char name[30] = "VIIT";
    int age = 20;
    p->name = name;
    p->age = &age;
    printf("Name = %s\n Age = %d\n", p->name,
                                    p->age);
}
```

Scanned by CamScanner

Scanned by CamScanner

Explanation:



* Here reference is created between local variables to pointer to structure members. & to access the structure member values, we use '->' arrow operator.

iii) Declaring structure members as pointers:

* Here, pointer is not declared to a structure variable & pointer is declared only to structure members. Here we use 'dot'(.) operator to access / display the structure members.

#Program to declare pointer as members of structure & display the contents of the structure without using -> (arrow) operator. */

```
#include<stdio.h>
#include<string.h>
void main()
{
    struct Person
    {
        char *name;
        int *age;
        float *height;
    };
    struct Person p;
```

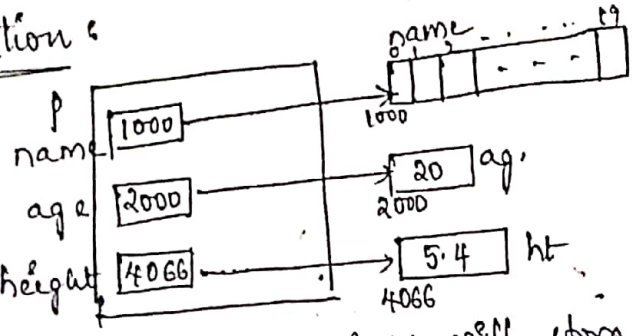
Scanned by CamScanner

Scanned by CamScanner

```
char nm[20] = "Anjali";
int ag = 20;
float ht = 5.4;
strcpy(p.name, nm);
p.age = ag;
p.height = ht;
printf("Name = %s\n", p.name);
printf("Age = %d\n", *p.age);
printf("Height = %f\n", *p.height);
}
```

Output:
Name = Anjali
Age = 20
Height = 5.400000

Explanation:



* Here p.name, p.age, p.height will store the address of name-nm, ag, ht variables because name, age, & height are of pointer variables.
* When the structure members are of pointer type & if structure variable is not a pointer type, they can be accessed using '.' operator { i.e, *p.name or p.name *p.age *p.height

Scanned by CamScanner

Scanned by CamScanner

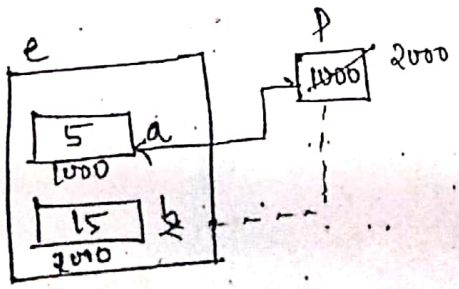
Note :// Program to display the contents of the structure using ordinary pointer.

```
#include<stdio.h>
struct Example
{
    int a;
    int b;
};
void main()
{
    struct Example e={5,15};
    int *p;
    p=&e.a;
    printf("\n a=%d\n", *p);
    p=&e.b;
    printf(" b=%d\n", *p);
}
```

output :

a = 5
b = 15

Explanation :



* Here, 'p' is a pointer variable used to access the structure members first to a & then to b. ie, p=&e.a; p=&e.b; holding the address of a & b variables.

Scanned by CamScanner

Scanned by CamScanner

→ Pointers and strings:

* To allocate & initialize a string, we declared an array of characters as follows:

Eg: char str[] = "Engineering";

* The same thing, we can declare a pointer type character as follows:

Eg: char *str = "Engineering";

* Both the declarations are equivalent. Here the compiler allocates enough space to hold the string along with '\0' (null character) as termination.

* A pointer pointing to 'str' contains the base address of the character array & complete array can be accessed using pointer. i.e, *str & for reading only 'str' variable is used.

// Program to illustrate the use of pointers with strings.

```
#include<stdio.h>
void main()
{
    char *name;
    printf("Enter name:");
    gets(name);
    printf("The entered name = %s\n", name);
}
```

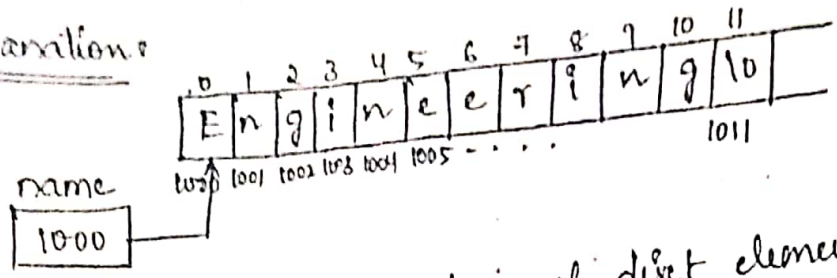
o/p:
Enter name: karthik
The entered name = karthik

output: No output coz it leads to error i.e, crash may occur
~~Enter name: karthik~~
~~The entered name = karthik~~ ('Ubuntu')

Scanned by CamScanner

Scanned by CamScanner

Explanation:



* Here name holds the address of first element 'E'. when we try to access we get address. so it wont work (or)

```
#include<stdio.h>
void main()
{
    char name[20], *p;
    printf("Enter your name:");
    gets(name);
    p= name;
    printf("The entered name = ");
    while(*p != '\0')
    {
        printf("%c", *p);
        p++;
    }
}
```

Output: Enter your name: karthik
The entered name = karthik

* Here 'p' holds the base address of first element (k) in the array 'name' ie, name[0] is stored in 'p'. Then the value is printed using printf() statement. If 'p++' is used, the pointer goes to the next character of the string ie, 'a' from karthik.

Scanned by CamScanner

Scanned by CamScanner

* When it encounters '\0' (null) character, the while loop gets terminated.

(or)

```
#include<stdio.h>
void main()
{
    char *p="hello";
    printf("%c", *p);
}
```

Output: hello

Here we should not prefer '*p' to access the value we have to use 'p'. If we try to use '*p' it leads to error, i.e, crash.

(or)

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    char *p;
    p=(char*) malloc(10);
    printf("Enter your name:");
    scanf("%s", a);
    printf("The entered name=%s\n", a);
}
```

Output: Enter your name: Welcome
The Entered name = Welcome

Scanned by CamScanner

Scanned by CamScanner