Structure: structure is a composition of different variable and different datatype grouped under same name

→ Ideally structure is the collection of different variable under same name.

→ Basically structure is used for storing the compleated data

→ A structure is a convienient way of grouping several pieces related information together.

```
typedef struct {
  char name[69];
  char course[118];
  int age;
  int year;
}
  student;
```

→ Name given to a structure is tag.

→ Structure member may be of different datatype including user defined datatype also

```
struct student
{
int rollno;
char name[20];
float marks;
};
```

```
void main()
{
struct student S1 = {1, "Om", 90.9};
struct student S2 = {2, "Amy", 98};
printf("%d", S1.rollno);
```

Signature

Initialization & accessing of structure

* Structure declaration

①
```
Struct tag Name
{
Data type member1;
Datatype member2;
:
Datatype membern;
}
```

②
```
Struct tag Name
{
 struct element1;
 Struct element2;
 :
 Strect element3n;
}
```

③
```
Struct
{
Datatype member1;
Datatype member2;
:
Datatype member n;
}
```

* Struct keyword is used to declare structure
* members of structure are enclosed with in opening and closing braces.
* Declaration of structure reserve no space, it is nothing but template / map / shape of structure.
* Memory is created very first time when the variable is created or instance

Immediatly after structure Template.

①
```
Struct data
{
int date;
char ~~moth~~ month [20];
int year;
} today
```
* today → variable.

② Declare variable using Struct Keyword.
```
Struct date
{
int date;
char month [20];
int year;
};
struct date today.
```
* date → tagname
* struct date today → Variable

Signature_____

③ Declaration of multiple structure variable.

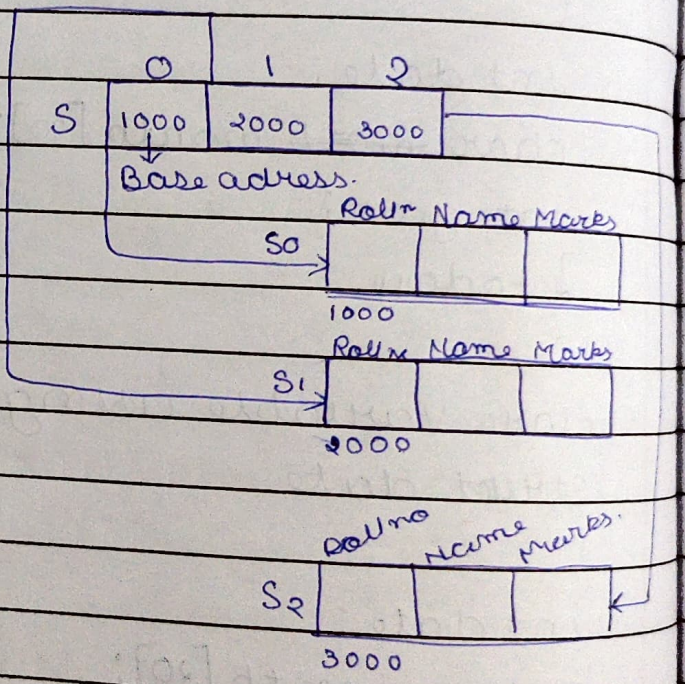
```
struct book
{
int page;
char name[20];
int year;
} book1, book2, book3;
```

Array of structure

when database of any element is used in huge amount we prefer array of structure.

* To declare

```
S[0].Rollno;
S[0].Name;
S[0].Marks.
S[1].Rollno;
S[1].Name;
S[1].Marks;
S[2].Rollno;
S[2].Name;
S[2].Marks;
```




```
Struct Student
{
int Roll no;
char Name [20];
float Marks;
};
Void main ()
```

Signature

```
{
struct student S[3];
int i;
for(i=0; i<3; i++)
scanf("%d %s %f", &S[i].Rollno, &S[i].Name,
    &S[i].marks);
}
```

* WAP to create a structure having members Empid, name and salary. Store 20 Employee details only those whose salary less then 10,000

Ans
```
#include<stdio.h>
 struct Emp
 {
 int Empid;
 char name[20];
 float Salary;
 };
 int main()
 {
 struct Emp[20];
 int i;
 printf("Enter Empid, name & salary of 20 people:");
 for(i=0; i<20; i++){
 scanf("%d %f %s", &Emp[i].Empid, &Emp[i].name,
  &Emp[i] salary);
 printf("Employee whose salary less then 10,000
 are");
```

Signature

```
for (i=0; i<20; i++)
{
if (Emp[i].Salary <= 10000)
{
printf ("%d and %S", Emp[i].Empid, Emp[i].name);
}
}
return 0;
}
```

Array of Structure

- An array structure in C can be defined as the collection of multiple structures variable where each variable contains information about diffrent enitities.

- The array of structure in C are used to store information about multiple entities of different datatype.

- The array of structure is also known as the collection of structures.

Q.) Program for accepting and printing details of 10 students.

```
#include <stdio.h>
struct student
{
int Sno;
char Sname[30];
float marks;
};
main()
{
Struct Student S[10];
int i;
for(i=0; i<10; i++)
{
 printf("Enter the detail of 10 students: %d", i+1);
 Scanf("%d %S %f", &S[i].Sno, &S[i].Sname, &
 S[i].marks);
}
for(i=0; i<10; i++)
 {
 printf("The details of student %d are ", i+1);
 printf("Sl no. = %d" S[i].Sno);
 printf("Name = %S", S[i].Sname);
 printf("Marks: %f", S[i].marks);
}
}
```

Pointers

- Pointers are some kind of special variable which contains the adress of any other variable.

- Pointers are derived datatype.

Declaration of pointer

Datatype * pointer name.

i) int *p
ii) float *p
iii) char *p
iv) Double *p.

it means here pointer p is containing the adress of a variable whose datatype is integer.

Here P = 1.1
P is containing the adress of P whose datatype is float

Initialisation of pointer

P = &a

There are two ways to acess the member of the structure with the help of a structure pointer.

i) with the help of astrick (*)
ii) with the help of arrow operator (→)

Q) write a C program to demonstrate structure pointer

⇒
```
#include <stdio.h>
#include <string.h>
// Creating struct student
~~stu~~ struct student
{
 int roll_no;
 char name[30]
 char branch[40];
 int batch;
};
// variable of structure with pointer defined.
struct student S,*ptr;
int main()
{
 ptr = &S;
// Taking inputs
printf("Enter rollno of student: ");
scanf("%d", &ptr->Rollno);
printf("Enter name of student: ");
scanf("%s", &ptr->name);
printf("Enter branch of student: ");
scanf("%c", &ptr->branch);
printf("Enter batch of student: ");
scanf("%d", &ptr->batch");
```

```
// Displaying detail of student
printf("\n Student details\n");
printf("Rollno : %d\n", ptr -> rollno);
printf("Name : %s\n", ptr -> name);
printf("Branch : %s\n", ptr -> Branch);
printf("Batch : %d\n", ptr -> Batch);
}
```

Structure with function

We can apply a structure as an argument to a function.

* What is a function?

-> A function is itself a block of code ~~with~~ which can solve complex task / calculation.

-> A function performs calcution on the data provided to it is called parameter or argument.

Q) WAP to Create a structure having members Emp ID, Name, Salary, store 10 employee details and then using UDF display only those employees whose salary >= 10,000.

->
```
#include <stdio.h>
struct emp.
{
int empid;
char name[40];
float Salary;
};
void display (struct emp [10]);
int main()
{
struct emp a[10];
int i;
```

Signature

```
printf("\n Enter EmpID, name & Salary for
 10 people:");
for (i=0; i<10; i++);
 {
scanf("%d %s %f", &a[i].EmpID, &a[i].Name
&a[i].Salary);
}
display (a);
return 0;
}
void display (structure emp x[10])
{
printf("Employee whose salary whose salary
less than 10000 are:");
for (i=0; i<=9; i++)
 {
 if (x[i] <=10.Salary <=10.000)
 printf("%d %s %f", x[i].EmpID, x[i].
Name, [redacted]);
 }
}
[redacted]
```

Pointers to array

The compiler alocates continuous memory location for all the element of the array

The base adress = First element adress (index = 0) of the array.

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

Element	a[0]	a[1]	a[2]	a[3]	a[4]
Value	10	20	30	40	50
address	1000	1004	1008	1012	1016

If P is declared as integer pointer then an array a can be pointed by the following assignment.

Every value of a is accessed by using P++ to move one element to another element when a pointer is incremented its value is increased by the size of the datatype that it points to.

Signature

Pointer to array (1-D)

This length is called scale factor.

Relationship between 'p' & 'a'

p = &a[0] = 1000
p+1 = &a[1] = 1004.
p+2 = &a[2] = 1008
p+3 = &a[3] = 1012
p+4 = &a[4] = 1016

- Address of an element is calculated using its index and scale factor of the data type.
- Adress of a[3] = base address + (3 * Scale factor)
  = 1000 + (3*4)
  = 1000 + 12
  = 1012.
- Pointers can use to acess array element ~~including~~ indexing.
- Pointer (P+3) gives the value of a[3].

```
#includ < stdio . h>
main ( ){
int a [5];
int * p, i;
printf ("Enter 5 element: ");
for (i = 0; i<5; i++)
Scanf ("%.d", & a[i]);
printf ("Elements of the array are: ");
for (i = 0; i < 5; i++)
printf ("%.d", *(p+i));
}.
```

	[0]	[1]	[2]	[3]	[4]
x =	50	25	11	66	18

Here x = name of array & it represent the base address.

To get the address of any simple variable we use address of operator it will give the memory location of a where it is stored

To get the value of next element of the array we have to increment of the pointer i.e. p++

Signature

Q.) Display Array of element and their address using pointer.

A)

```
#include <stdio.h>
void main()
{
 int x[5] = {10, 20, 30, 6, 3}
int i;
int *p;
p = &x;
for(i=0; i<5; i++)
 {
  printf("%d", *(p+i));
 }
}
```

2D - Array & Pointer

```
#include <Stdio.h>
int main()
{
int p[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
printf("%p", p);
printf("%p", *p);
printf("%d", **p);
printf("%d", **(p+1));
printf("%d", *(*p+1));
printf("%d", **(p+2));
return 0;
}
```

Signature

Memory Allocation:

-> Static memory Allocation (SMA)
-> Dynamic memory Allocation (DMA)

Static Memory Allocation

when the memory is allocated at the begning of a program using declaration statement then it is called static memory allocation

```
int x, y, z;
char stack[100];
float *p, *q;
struct emp a[10];
```

Dynamic Memory Allocation:

The process of alocating memory at the time of the exicutation or at the run time is called Dynamic memory allocation.

-> Two type of problem may occure in static memory allocation. If no of values to be stored is less then the size of memory there would be wastage of memory.

-> If we would want to store more values by increase in size during the exicution on assigned size then it fails.

Signature_____

→ Allocation and release of memory space can be done with the help of some library function called dynamic memory allocation function

→ These library function are called as dynamic memory allocation function

→ These library function prototype are found in the header file alloc.h where it has define.

→ Function take memory from memory area is heap and release when not required.

→ Pointer has imp. role in the dynamic memory allocation to allocate memory.

→ It has 4 functions
1. Malloc()
2. Calloc()
3. Realloc()
4. free()

① Malloc:-

→ It returns the pointer to the first ~~bit~~ byte and allocates memory and it return type is void which can be typecast such as.

int* p(datatype*) mallo (size);

→ If memory location is successful it returns the address of the memory chunk that was allocated and it returns null on unsuccessful and form, the above declaration of pointer of type (data type) and size in ~~bit~~ byte.

→ And Datatype pointer use to typecast the pointer returned by malloc() and this type casting is necessary since malloc() by default returns the pointer to void.

int * p = (int *) malloc (10);

So from the above pointer p allocated to contigious memory space address of first byte and it is stored in the variable.

→ we also use the size of operator to specify size ~~of~~ such as.

* p (int*) malloc (5* size of int)

Here S is the no of data more over it returns if no sufficient memory is available we should always check the malloc return such as if

```
if(p == null)
printf("not sufficient memory");
```

M T W T F S S
Page No.:
Date:

② Calloc( )

→ Similar to malloc but only difference is that calloc function use to allocate multiple block of memory.

→ Two arguement are there
  ⓘ First argument specifies the no. of block.
  ⓘⓘ Second argument specifies the size of each block.

```
int *p = (int*) calloc (5,2);
int *p = (int*) calloc (5, size of (int));
```

Another difference between malloc and calloc is by default malloc contains garbage values where as ~~allocated~~ memory allocated by calloc is initialized by zero but this initialization is not reliable.

③ Realloc( )

The function ReAlloc used to change the size of memory block and it alter the size of the memory block without losing the old data is called Reallocation of memory.

→ It takes the arguemenent such as.

```
int *ptr = (int*) malloc (size)
int *p = (int*) ~~no~~ realloc (ptr new size);
```

Signature

→ The new size allocated may be larger or smaller. If the new size is larger then the old size then the old data is not lost. the newly allocated byte are un initialised.

→ If the old adress is not sufficient then starting address contained in pointer may be changed and this reallocation function moves content of old block.

(4) Free

→ Function free is used to release splace allocated dynamically the memory released by free is made available to heap again it can be used for further ~~hotte~~ purpose.

Syntax

```
void (* ptr)
    or
free (p)
```

when the program is terminated memory released by automatically by the operating System even we don't free memory it doesn't give error.

→ Thus lead to memory leak we can't free the memory those which doesn't allocated.