★ **Structures** :- We can write our own data types called User defined data types.

int, float, char → Primitive / fundamental data types

Here we use keyword struct and it is a collection of heterogeneous elements.

```
Struct NAME
{ Properties
};
```

```
Struct HEIGHT
{
    int f;
    int I;
};
int main()
{
    Struct HEIGHT Prince, Sahsham, diff;
    Prince.f = 5;
    Prince.I = 6;
    printf("Enter Heigh of Sahsham");
    scanf("%d %d", &Sahsham.f, &Sahsham.I);
    int df = abs((Sahsham.f * 12 + Sahsham.I) - (Prince.f * 12 + Prince.I));
    diff.f = df / 12;
    diff.I = df % 12;
    printf("\n difference = %d & %d", diff.f, diff.I);
}
```

Creation

Properties / Attributes

Input

Processing

output

diff	
f	1
I	3

Sahsham	
f	4
I	3

Prince	
f	5
I	6

Home Work

Q- Write a note on Structures.

A- A Structure is a user defined data type in c/c++. A Structure creates a data type that can be used to group items of possibly different types into single type.

Creation :- 'struct' keyword is used to create a structure

Eg :-
```
struct address
{
    char name[50];
    char street[100];
    char city[50];
    char state[20];
    int pin;
};
```

Declaration :- A structure variable can either be declared with structure declaration or as a separate declaration like basic types

// A variable declaration with structure Declaration.
```
struct Point
{ int x,y;
} P1; // The variable P1 is declared with 'Point'.
```

// A variable declaration like basic data types
```
struct Point
{ int x,y;
};
```

Q- Difference b/w Arrays and Structures.

A -

	Structure	Array
1) Definition	Structure can be defined as a data structure used as Container which can hold variables of different types.	Array is a type of data structure used as Container which can hold variables of same type and do not support multiple data type variables.
2) Performance	Structure due to use defined data type become slow in Performance as access and searching of element is slower in structure as compared to arrays.	In case of Array access and searching of element is faster and hence better in Performance
3) Instantiation	Structure object can be created after declaration later in the program.	In case of Array we can't create its object after declaration.
4) Pointer	Structure do not have concept of pointer internally.	In case of Array it internally implements pointer which always points to first element of Array.

Q- Write a program for difference of time b/w two clocks.

A -

```
Struct TIME
{
    int min;
    int Hour;
    int Sec;
};

int main ( )
{
    Struct TIME  Clock1, clock2, diff;

    printf (" Enter Time in clock 1 : \n");
    Scanf (" %d %d %d", & Hour, & min, & Sec);

    printf (" Enter Time in clock 2 : \n");
```

```
    Scanf (" %d %d %d", & clock2. Hour, & clock2. min, & clock2. Sec);

int df = abs( ((clock1. Hour * 60 * 60 + clock1. min * 60 + clock1. Sec) -
    (clock2. Hour * 60 * 60 + clock2. min * 60 + clock2. Sec)) );

    diff. Hour = df / (60 * 60);
    diff. min = [illegible] (df % (60 * 60)) / 60;
    diff. Sec = (df % (60 * 60)) % 60;

    printf (" Difference = %d_ %d_ %d", diff. Hour, diff. min,
                                        diff. Sec);
}
```

Difference	
min	19
Sec	40
Hr	1

= 4780 Sec

clock 2	
min	20
Sec	40
Hr	1

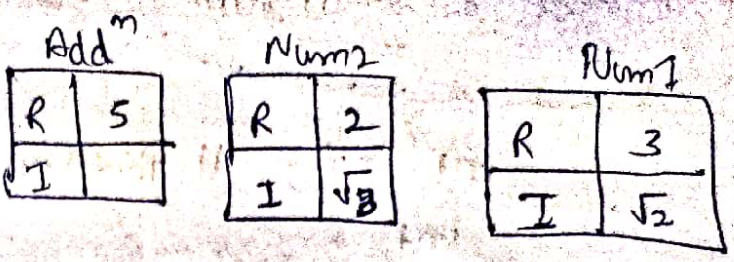
= 4840 Sec

clock 1	
min	40
Sec	20
Hr	2

= 9620 sec

Q- Write a program for sum of two Complex No.'s.

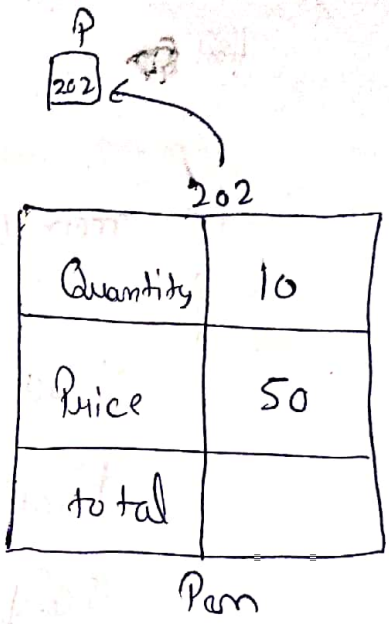
A-




```
Struct Sum
{
    int R;
float I;
}

int main()
{
    Struct Sum Num1, Num2;
    printf("Enter Num1: \n");
    Scanf("%d %f", &Num1.R, &Num1.I);
    printf("Enter Num2: \n");
    Scanf("%d %f", &Num2.R, &Num2.I);
    int Real = Num1.R + Num2.R;
float Imaginary = Num1.I + Num2.I;
    printf("%d _ %f", Real, Imaginary);
}
```

★ Pointer to Structure :-

```
int *P; // Pointer to integer
int x;
P = &x;
float y;
P = &y; X
```




```
Struct PRODUCT
  {
    int Quantity;
    float Price, total;
  };
int main()
 {
    Struct PRODUCT *P; // Declaration of pointer to structure
    Struct PRODUCT Pam;
      P = & Pam; // Storing address
  printf ("Enter price and Quantity: \n");
  scanf ("%d %f", & Pam.Quantity, & Pam.Price);
(*P).total = (*P).Quantity * (*P).Price;
                or
  P -> total = P -> Quantity * P -> Price;
  printf ("\n total = %f", P -> total);
 }
```

Home Work

Q- Write a note on pointer to structure.

A- We know, Pointers are variables that store the addresses of the same type of variable i.e. an int pointer can store an address of an integer, a char pointer can store an address of a char, and similarly for all other data types, fundamental or user-defined.

We can Access a structure member using pointers, of type structure, in the following ways.

(i) Using the arrow operator :- If the members of the structure are public then you can directly access them using the arrow operator (->). If they are private then you can define methods for accessing the values and use pointers to access the methods. The arrow operator can be used to access structure variables as well as method.

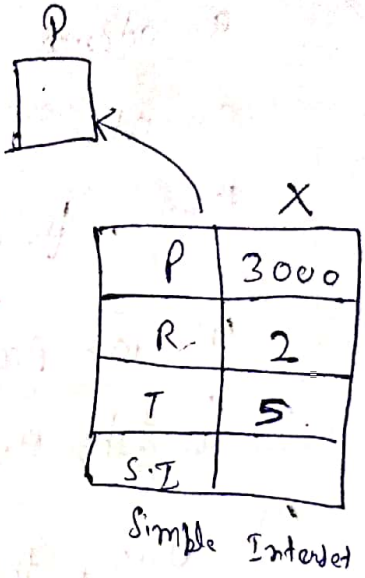
Syntax :- Pointer-Name -> member-Name;

(ii) Using the Defrencing operator :- you can also access structure elements using the defrencing operator on the pointer, which is using an asterisk to defarence the pointer and then using the dot operator to specify structure element.

Syntax :- (* Pointer_Name). Member Name;

Q- Write a program for Simple interest using pointer to structure.

A-




```
Struct Simpleinterest
{
    int P;
    int R;
    int T;
    float S.I;
};
int main()
{
    Struct Simpleinterest *A;
    Struct Simpleinterest X;
        A = &X;
    printf("Enter principal value, rate of interest, time : \n");
    Scanf("%d %d %d", &X.P, &X.R, &X.T);
(*A).S.I = (*A).P * (*A).R * (*A).T / 100;
    printf("Simple Interest : %f", (*A).SI);
}
```

Q- Write a note on Self-Referential Structure.

A- Self referential structures are those structures that have one or more pointers which point to the same type of structure as their member. Or we can say that structure pointing to same type of structure are self-referential in nature.

Eg:-

```
Struct Prince
{
    int A1;
    char s2[];
    struct Prince * B;
};

int main ( )
{
    Struct Prince x;
    return 0;
}
```

Here, we see that 'B' is a pointer to a structure type 'Prince'. Hence the structure 'Prince' is a self-referential structure with 'B' as referencing pointer.

Types :- ① Self Referential structure with Single Link

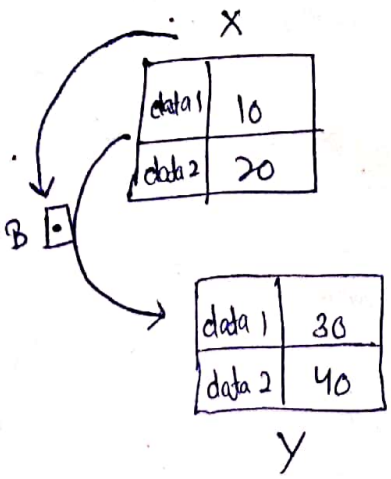
② Self Referential Structure with multiple Link.

① Self - referential Structure With Single link :-

These structures can have only one self-pointer as their member.

Eg :-

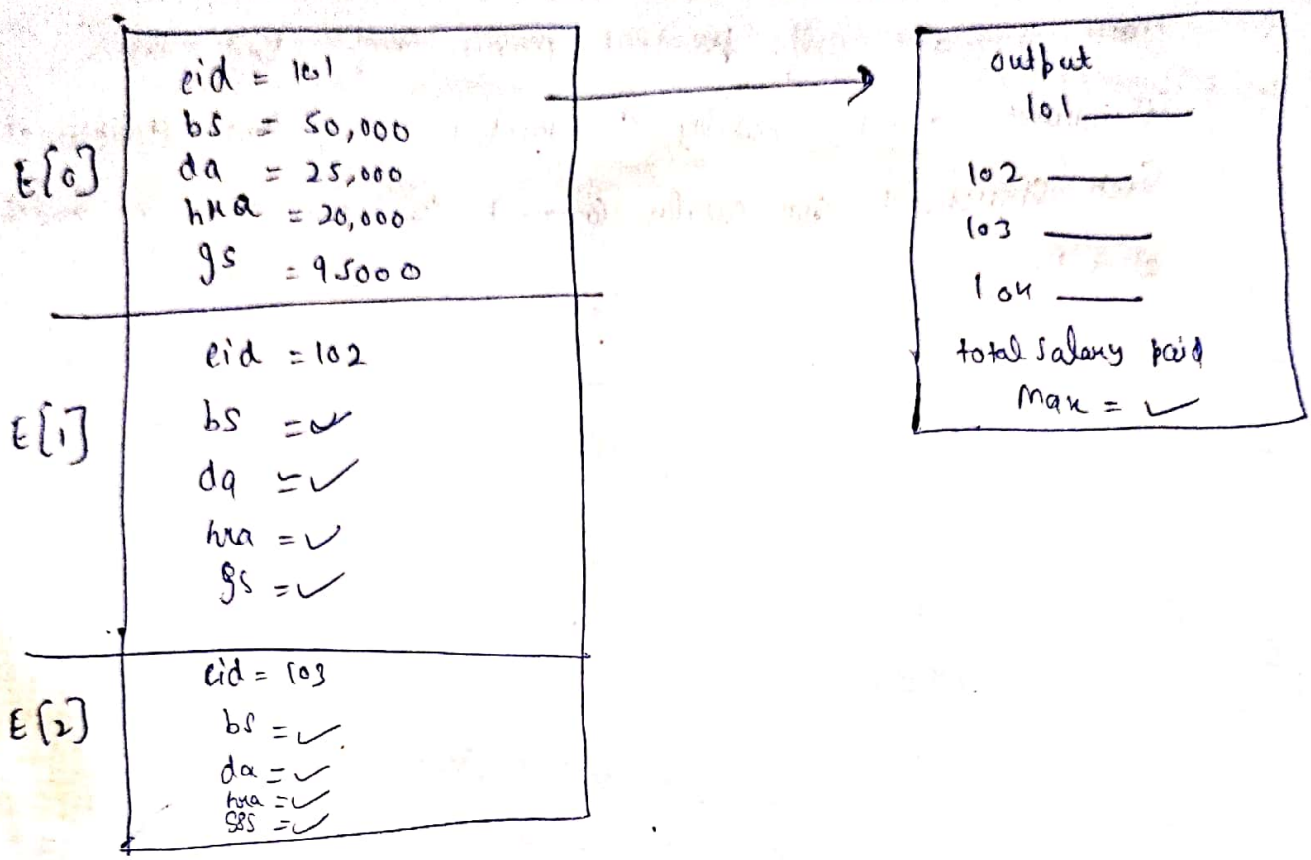
```
Struct Prince
{
    int data1;
    char data2;
    struct Prince * B;
};

int main()
{
    struct Prince x;
        x.data1 = 10;
        x.data2 = 20;
    struct Prince y;
        y.dat1 = 30;
        y.data2 = 40;
    //--Linking x and y---
        x.B = &y;
    printf ("%d", x.B -> data1);
    printf ("\n%d", x.B -> data2);
}
```



(2) Self-referential Structures with Multiple Links :-

Self-referential structures with multiple links can have more than one self-pointer. Many complicated data structures can be easily constructed using these structures. Such structures can easily connect to more than one nodes at a time.

* **Array of structures** :- Used to store and process data of multiple entities of similar type.




```
Struct EMPLOYEE
  {
     int eid;
     float bs, da, hra, gs;
  };
int main ( )
  {
     Struct EMPLOYEE A,P,C --100;
     int N, i ; float sum = 0;
     printf (" Enter No. of Employes : \n");
     Scanf (" %d", &N);
```

```
struct EMPLOYEE E[N]; // creation of array of structure
for (i=0; i<N; i++)
{
    printf ("Enter empid and Basic salary of %d: \n", i+1);
    scanf ("%d %f", &E[i].eid, &E[i].bs);
    E[i].da = E[i].bs * 50/100;
    E[i].hra = E[i].bs * 40/100;
    E[i].gs = E[i].bs + E[i].da + E[i].hra;
}
printf ("\n output: \n");
float max = E[0].gs;
for (i=0; i<N; i++)
{
printf ("\n Empid = %d, da = %f, hra = %f, Gross = %f",
        E[i].eid, E[i].da, E[i].hra, E[i].gs);
    sum = sum + E[i].gs;
    if (max < E[i].gs)
        max = E[i].gs;
}
printf ("\n Maximum salary = %f", max);
printf ("\n total salary = %f", sum);
}
```

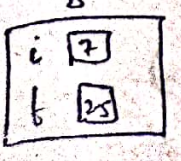
Q- Diff. b/w Structures and Union.

STRUCTURES

- The size of struct depends upon the sum of size of all member variables
- All member variables occupy different memory space for storing the values.
- So different values can be stored at a time.
- Syntax :- struct NAME
  {
  ——
  ——
  };

Eg :- struct MI x → 8 bytes(4+4)
{
int i;
int f;
} s; // s is global variable

int main()
{
s.i = 7;
s.f = 2.5;
printf("%d %f", s.i, s.f);
}

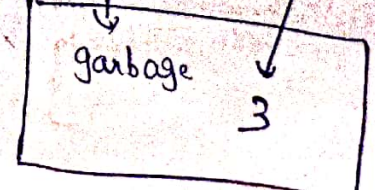


UNIONS

- The size of union depends upon the size of member variable having largest size among others.
- All member variables occupy same space.
- So one value can be stored at a time, which is stored at last.

Eg :- union MI x → 4 bytes
{ int i;
float f;
} u;

int main()
{ u.f = 8.9;
u.i = 3;
printf("\n u.f = %f, u.i = %d",
u.f, u.i);



Home Work

Q- Write a program for N students using array of structures for max. marks, total, min. Percentage, Average Percentage

A-

```
Struct Student
{
    int Roll No., class;
    float marks, total, Percentage;
};

int main ()
{
    int n, i
    printf (" Enter No. of students : \n");
    scanf ("%d", &n);
    Struct Student S [n];
    for (i = 0 ; i < n ; i++)
    {
        printf (" Enter Roll No.[%d] & marks[%d] : \n", i, i);
        scanf (" %d %f ", & s[i].Roll No, & s[i].marks);
    }
    float max = s[0].marks;
```

```
Struct Student
{
    int Roll No.;
    float marks P, marks H, marks C, total, Percentage;
}

int main ()
{
    int N, i ; float HPercentage = 0;
    printf (" Enter value of no. of students : \n");
    scanf (" %d", & N);
    struct Student S[N];
    for (i=0 ; i < N; i++)
    {
        printf (" Enter Roll No. , marks in maths , marks in
                Physics & marks in chemistry : \n");
        scanf (" %d %f %f %f", & S[i]. Roll No., & S[i]. marks P,
                & S[i]. marks H, & S[i]. marks C");
    S[i]. total = S[i]. marks P + S[i]. marks H + S[i]. marks C;
    S[i]. Percentage = S[i] total / 300 * 100 ;
    }
float max = S[0]. total
float min = S[0]. Percentage
```

```
for (i=0 ; i < N; i++)
{
  printf (" total marks : %f\n", s[i].total);
    if ( max < s[i].total)
        max = s[i]. total ;
    if ( min > s[i]. percentage)
        min = s[i]. percentage
  }
  printf (" maximum marks = %f \n", max);
  printf (" minimum percentage = %f \n", min);
for (i=0 ; i < N; i++)
{
  [redacted]
    H Percentage = H Percentage + s [i]. Percentage ;
}
  printf (" Average percentage : %f ", H Percentage /N);
}
```