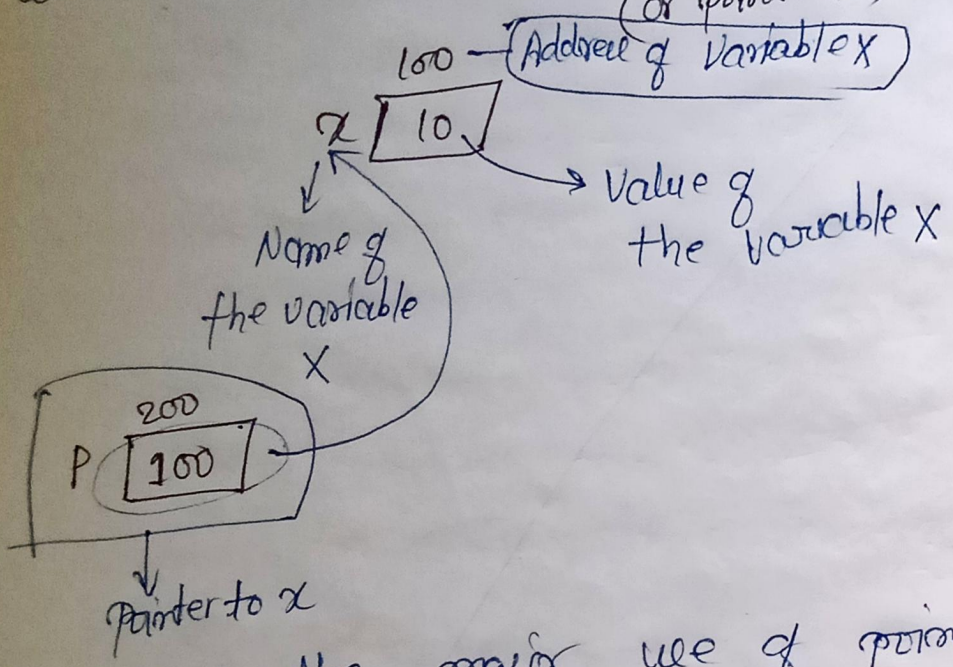
Unit-5

~~Pointer~~

Pointers:-

Pointer in c is defined as a variable which stores the address of some another variable (or pointer too)



* One of the major use of pointers is in allocating memory dynamically or run time

Declaration of pointer:-

The syntax of a pointer declaration is as given below.

data-type *ptr_name

The type of value to which pointer will point

Asterisk operator which signifies that it is a pointer variable

user define name given to pointer variable

Ex: Consider a pointer p which is pointer some Integer value.

```
int *p;
```

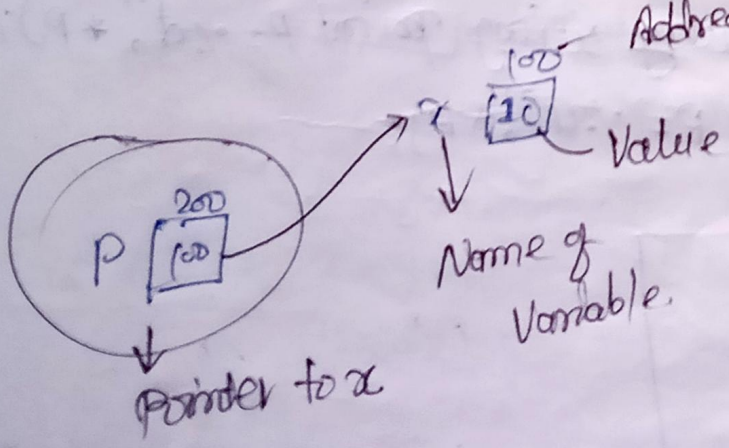
Initialization of pointer:-

Any pointer variable can be Initialized using Address of operator (&) And syntax is-

```
Ptr_name = & Variable_name;
```

Ex.

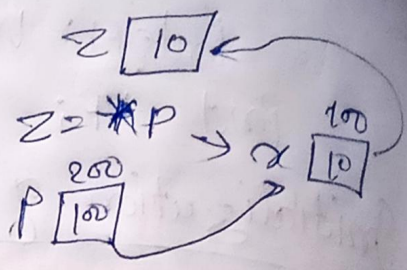
```
int x = 0;
int *p;
p = &x; // Initialization of pointer p.
```



Accessing the value of some variable using it's pointer:-

The value of some variable 'x' can be Accessed using it's pointer 'p' as follows with the help of Asterisk operator (*).

Eg.
```
void main()
{
    int x=10, z;
    int *p; //Pointer declaration
    p = &x; //Initialization of pointer p
    z = *p; // z=10
```



↓

The '*' is read as "Value stored at"

```
    printf("Value of z = %d", z);
    getch();
}
printf("Value of x = %d", x);
printf("Address of x = %u", &x);
printf("Address of x or value of pointer p = %u", p);
printf("Value of x using pointer p = %d", *p);
printf("Address of p = %u", &p);
```

Output :-

```
Value of z = 10
Value of x = 10
Address of x = 100
Address of x or value of pointer p = 100
Value of x using pointer p = 10
Address of p = 200
```

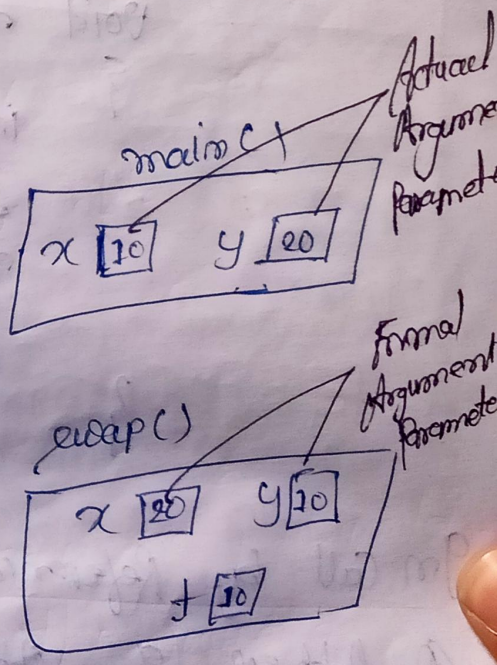
Call by value and Call by reference:-

In call by value parameter passing technique whenever a function is called the value of the arguments (Actual Arguments) are passed to the respective formal Arguments.

Therefore the change is made in the values of formal Argument. inside the called function are not reflected in the actual Argument in caller function.

Consider the following program to swapping the value of two variable x & y.

```
// Called function
void swap (int x, int y)
{   int t;
    t = x;
    x = y;
    y = t;
}
// Caller function
void main ()
{   int x=10, y=20;
    swap (x,y);
    printf ("x=%d, y=%d", x,y)
    getch();
}
```

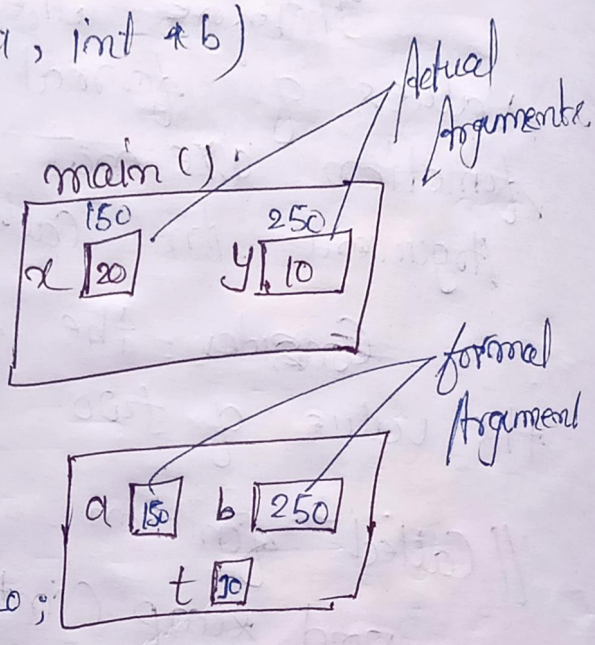


But in main function changes only not come. therefore value are printed.

Here the values of formal Argument a & b of y are swap but the value of Actual Arguments x & y remain same.

Call By Reference :-

```
Void swap (int *a, int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
Void main()
{
    int x=10, y=20;
    swap (&x, &y);
    printf ("x=%d & y=%d", x, y);
    getch();
}
```



In Call by Reference parameter passing technique

(i) Address of Actual Argument are passed to the formal Arguments (which are pointer)

(ii) Therefore the change in made using this pointers inside the Called function.

causes the change is at the address pointed by them (address of Actual Argument).

Consider the following example where the Address of Actual Arguments are passed to the formal Arguments a & b which are nothing but pointers.

Files :-

```
#include<stdio.h>
```

fscanf - formated scanf
fprintf - formated printf

{ stdio — standard Input output

Console - blank screen

① Create point to file :- File *fp

EOF - End of file.

② open the file to work with :- fp = fopen("files.txt", "w")

③ Read/write operations :-

fscanf(fp, "%d %f", &id, &sal);

Mode :-

(i) Read mode = r
(ii) write mode = w
(iii) Append mode = a

fgetc(fp) — read one character at a time.
fputc(fp) - print a character

④ Close file after use :- fClose(fp)

① File is a data structure sometime also called data types) that can be use to store any data in it parmanently.

② In programming a file is handal using some file pointer of type 'File' (which is a data structure defined as files in standard library)

③ To use any file for reading writting for any other purpose usually following step are perform in sequence —

① Creat a pointer to the file:— File *fp

② open the file to work with:- for this to open a file the standard function name 'fopen' is used as follows.

```
fp = fopen("filename", "mode")
```

↓ Name of the file to be open

↘ specifies the mode

if the file is succesfully open itle pointer is return other wise null is return— there are usually three mode that can be use

(i) Read mode :- 'r' - if file is open ~~successfull~~ unable to open then it returns null.

(ii) write mode :- 'w'

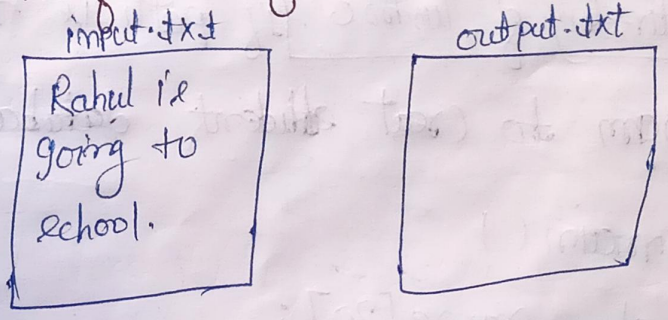
(iii) Append mode :- 'a'

if

(ii) If specified is not existing it creats a new file with that name.

(iii) ~~Repeee~~ If we ceemt to add Contains after the last written Contain then append mode is use. if the specified name file is not found it create a new file with that file.

/* Program to Copy Contex of one file to another file */




```
void main()
{
    FILE *fp1, *fp2;
    char ch;
    fp1 = fopen("input.txt", "r");
    fp2 = fopen("output.txt", "w");
    if(fp1==Null || fp2==Null)
        printf("There is an error in opening the file");
```

```
else
{   //logic to copy
    while((ch=fgetc(fp1))!=eof)
    {
        fputc(ch,fp2);
    }
    fclose(fp1);
    fclose(fp2);
}
getch();
}
```

If char is read to Console then use 'getchar' at the place 'fgetc'

If reading is done in Console then use 'putc' of printf at place of 'fputc'

file will save in bin directory of Turbo C. if path is not given

```
/* Program to creat student database */
void main()
{   char name[20];
    FILE *fp;
    int id;
    float marks;
    fp=fopen("stud.txt","w");
    if(fp==NULL)
        printf("Error in opening the file");
    else
    {   fprintf(fp,"Id Name marks \n");
```

stud.txt

Id

```
for(i=0; i<5; i++)
{
    scanf("%d", &id);
    gets(name);
    scanf("%f", &marks);
    fprintf(fp, "%d %s %f \n", id, name, marks);
}
fclose(fp);
}
getch();
}
```

Credit Debit

~~Bank Reservation~~

~~Name Consumer Id~~

Dynamic Memory Allocation :-

↳ Allocation of memory at runtime i.e during the execution of program.

(1) Advantage of this is, that memory can be allocated any time during the execution of the program as per the requirement which is not possible in static memory allocation.

(2) Unlike array it saves the wastage of memory because if allocated memory not required it

may be released.

④ Dynamic memory allocation can be handle by standard library functions of memory management given as below -

malloc()
calloc()
realloc()
free()
} - <stdlib.h>

① malloc() :- It is used to allocated requested bytes of memory during the program and return the pointer (void) to the first byte of allocated memory. (memory no any data type)

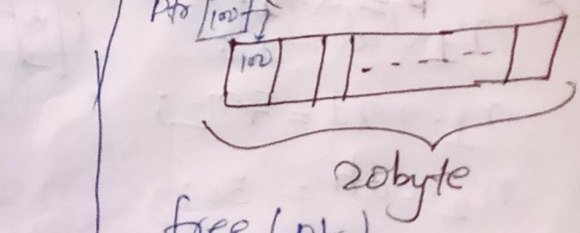
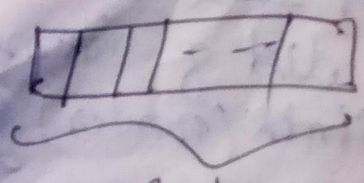
Syntax :
(cast_type *) malloc (byte_size)

int *
float *
char *
} malloc(10)

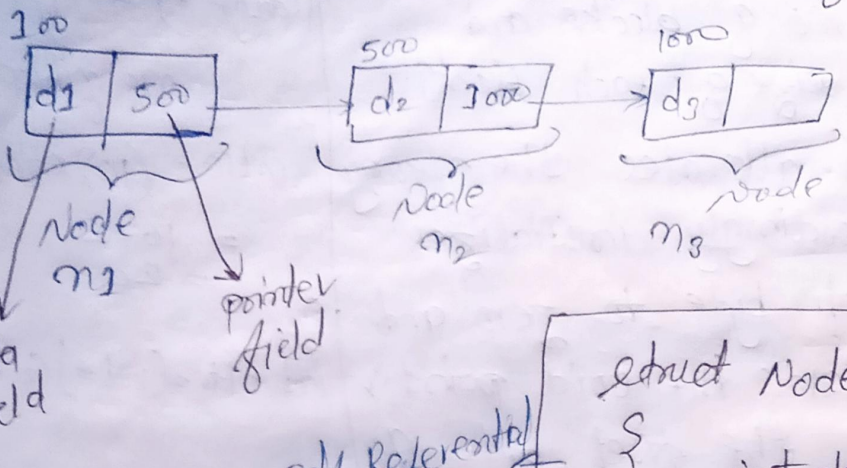
If the required size of memory is not available then it return 'NULL'.

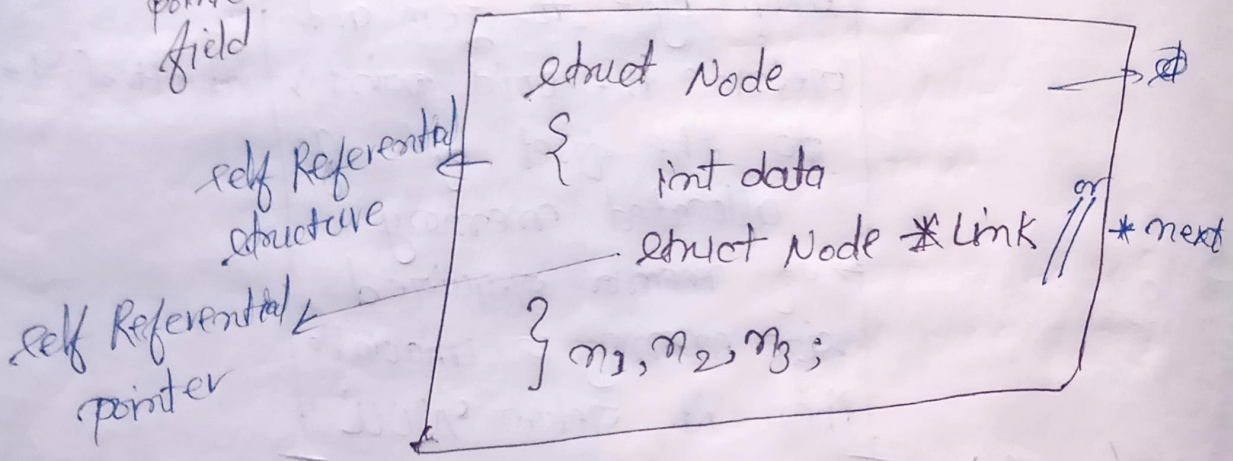
eg- malloc (10 * sizeof (int))

(10 * 2) = 20 byte

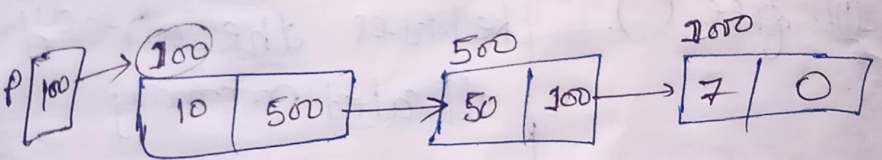
	Description	syntax
(ii) calloc( )	It takes two argument 1- no of blocks and 2- size of each block. and allocates the memory accordingly, initializes each byte to zero and returns the void pointer to the first byte of allocated memory ⇒ If memory required size memory is unavailable then it return 'NULL'	(Castype *) calloc (n, elem size) no of block eg calloc (10, sizeof(int)) = 10 * 2 = 20 eg int * Ptr = (int *) calloc(
(iii) free( )	Release the already allocated memory	free (Ptr_memory)  free (Ptr)
(iv) realloc( )	realloc function is use to resize the memory already allocated dynamically. It is used when we want to increase or decrease the size already allocated	realloc (Ptrname, new size) eg- realloc (Ptr, 30) 

Self Referential structure (Notion of linked list)






```
struct Node *p;
p = &n1;
n1.data = 10;
n1.next = &n2;
n2.data = 50;
n2.next = &n3;
n3.data = 7;
n3.next = NULL;
```



A link list is a data structure that maintains the list of items that are logically connected with link (pointer). In singly link list every node contains two information inside it — Data field :- ~~that~~ it stores the actual value of data items

Pointer fields:- It stores the address of next Node or (data items).

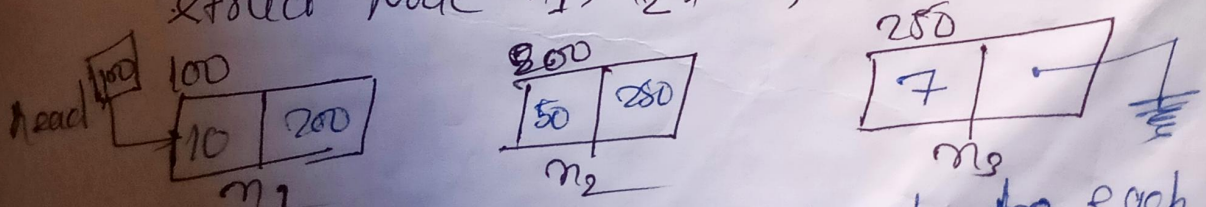
To creat a singly link list the structure of Node can be defined as follows.

```
struct Node
{
    int data
    struct Node *next
} n1, n2, n3;
```

Here next is a pointer which points to a structure of kind itself Hence it is called self referential pointer. moreover, the structure which contains such referential pointer is called self referential ~~structure pointer~~ structure.

Creation of link list:- Consider that we want to creat a link list to store three data items then we need to creat three Nodes as follows.

```
struct Node n1, n2, n3;
```



the value can be assigned to each and every Node as follow.

struct Node n_1, n_2, n_3

struct Node *head, *new

head = &n_1;

n_1.data = 10;

n_1.next = &n_2;

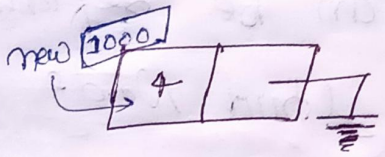
n_2.data = 50;

n_2.next = &n_3;

n_3.data = 7;

n_3.next = NULL;

new = (struct Node *) malloc(sizeof(struct Node



new → data = 4

new → next = NULL

n_3.next = new;

new is pointer so we used new → data at place of (n_3.data)

head → next → next → data = 7

Enumerated Data types :-

① It is use define data type in C.

② It used to give names to integer Constant.

③ It makes the program more readable and understable.

Syntax :-

enum enum_name { Constant 1, Constant 2 ... Constant N }

enum → Keyword

enum_name → User defined name given to enumeration

Constant 1, Constant 2 ... Constant N → Integer Constant Names.

Ex. enum status { off=0, on=1 }

```
enum status switch;
switch = on;
printf("%d", switch);   // 1
```

Ex. enum week { Mon, Tue, Wed, Thus, Fri, Sat, Sun }
0 1 2 3 4 5 6

{ we have not assigned any value then Compiler assigned value 0 to ... }

Creation of Enumeration Variable :-

Syntax :- enum enum_name var_name;

Ex. enum week day;

⑥ Defining Macros:-

```
#define size 10
Void main()
{
    int [size], i;
    // Reading array
    for(i=0; i<= size-1; i++)
        scanf("%d", &a[i])
    // Display array
    for(i=0; i<= size-1, i++)
        printf("%d", a[i]);
```

Preprocesser directives are Commands to the Compiler to perform certain task before the process of compilation start. These Comand are start from symbol (#) following are main preprocesser directives —

① File Inclusion:-

This preprocesser directives is used to include the contains of predefine system header files or user header files.

ex. # include <stdio.h> // To include built in header files.

include "my header.h" // To include user define header files.

② Defining Macros:-

Macros are basically symbollic names corresponding to some Constant or expression of function

before the Compilation start the value corresponding to this macro is substituted throughout in the program.

Ex.
```
# define PI 3.14
void main()
{
    float r,a;
    scanf("%f",&r)
    a = PI * r * r;
    printf("Area = %f", a);
    getch();
}
```

```
# define square(x)  ((x) * (x))
void main()
{
    int z=10, y;
    y = square(z);
    printf("y=%d", y);
    getch();
}
```

3) Conditional Compilation :-

```
#ifdef PI
    #define PI 3.14
#endif
```

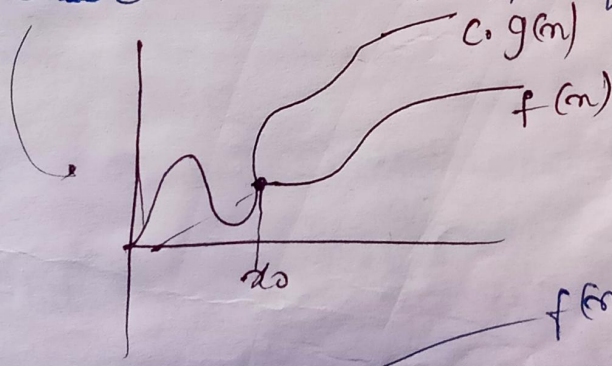
#ifdef
#ifndef

#if expression
#elif
#endif

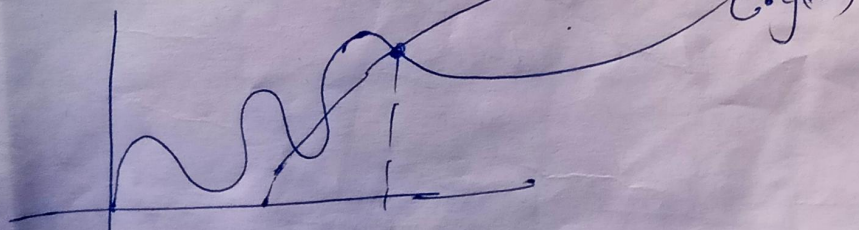
Notion of Complexity:-

Complexity is a way to asses and compair algorithms for a problem with respect to several parameter. These parameter may be time, space, power consumption. Basically following are some notations that are used to meaure the complexity of any Algorithm.—

① Big Oh (O):- It is define for a function $f(n)$ we say that $f(n)$ is order bigoh of $g(n)$ $\{f(n) = O(g(n))$ if there exist some +ve constant c and $n(0)$ such that $c \geq 0$ $\forall n \geq n(0)$ if $f(n) \leq c \cdot g(n)$

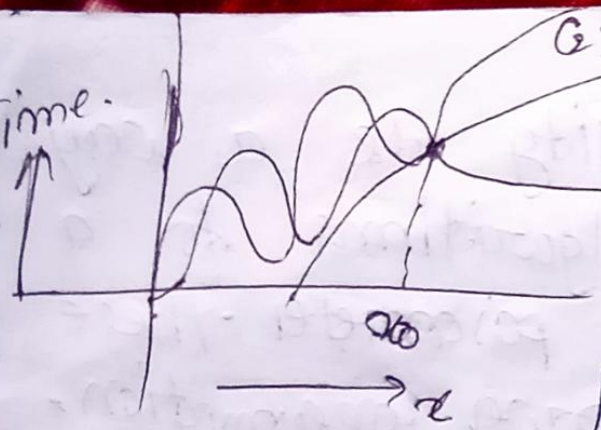


② if



$f(n) = \Theta(g(n))$

(iii)



$$C_1 g(n) \leq f(n) \leq C_2 g(n)$$

Theta (θ)

Asymptotically Tight Bound of $f(x)$

$$f(n) = \theta(g(n))$$

* structures & unions :-

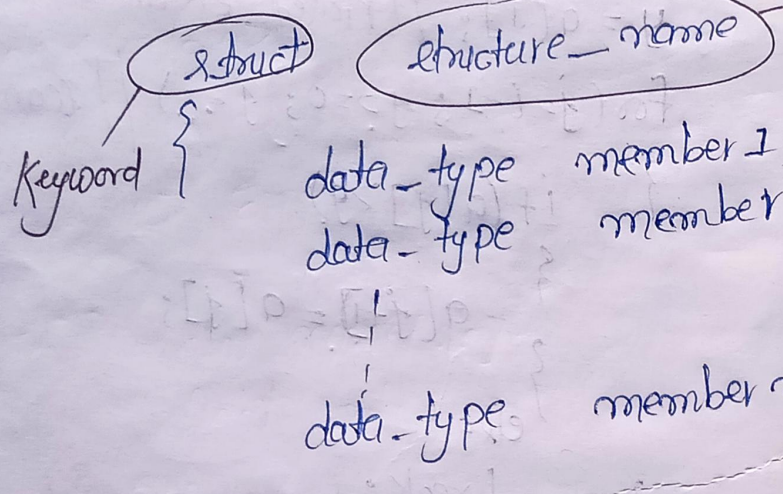
In programming, a structure can be considered as A user defined datatype i.e —

1. Capable of grouping data ty item of different types in
2. Usually groups logically related records items a single unit.

Declaring a structure :-

The syntax of declaring A structure is as following -

syntax :-



For example Consider a structure storing to record the employee id (integer), Name (character array or string), and salary of employee (floating value) will be declared as.

```
struct Employee
{
    int id;
    char name[20];
    float sal;
};
```

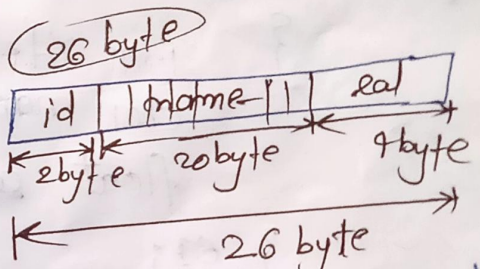
declaration of structure Variables :-

A structure variable can be declared in following two ways. (Any where in a function locally or Globally)

1st method :- ~~struct Em~~

A variable e1 consider to a employee will be declared as below.

```
struct Employee e1;
```



As soon as structure variable e1 of employee structure is declared a memory is allocated for e1 of total 26 byte as shown in the diagram.

2nd Method :- (with declaration)

structure variable can be created with structure declaration also. The example is given below.

```
struct Employee
{   int id;
    char name[20];
    float sal;
} e1, e2, ---en;
```

Accessing of members using structure variable.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    struct Emp
    {   int id;
        char name[20];
        float sal;
    } e1;
    void main()
    {   struct Emp e2;
        printf("Enter id of emp1");
        scanf("%d", &e1.id);
        printf("Enter the name of emp1");
        scanf("%s", e1.name);
        printf("Enter sallary of emp1");
        scanf("%f", &e1.sal);
        printf("Enter the Details of emp2");
        scanf("%d", &e2.id);
```

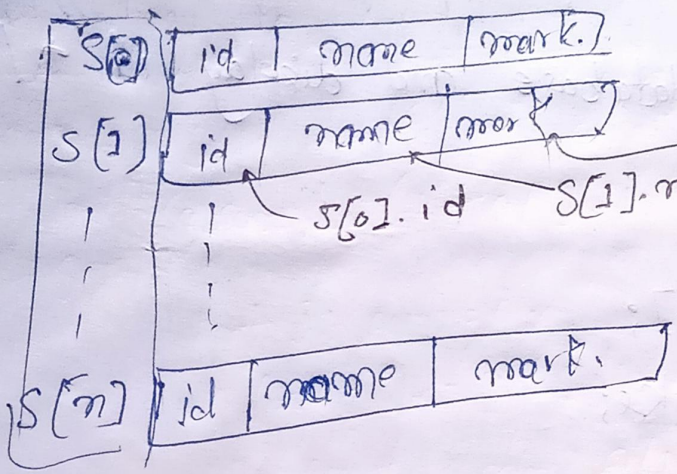
```
gets(e2.name);
scanf("%f", &e2.sal);
printf("\n\n Details of Employee 1");
printf("\n id = %d", e1.id);
printf("\n name = %s", e1.name);
printf("\n sallary = %f", e1.sal);
printf("\n Details of Employee 2");
printf("\n id = %d", e2.id);
printf("\n name = %s", e2.name);
printf("\n sallary = %f", e2.sal);
getch();
}
```

for using gets:
before using gets function we have to clear buffers by using function fflush(stdin);

Array of structure :-

* WAP to creat student database and display it */

```
struct stud
{ int id;
  char name[20];
  float marks;
};
void main()
{ struct stud s[10];
  int i;
  // Input student details
  for(i=0; i<=9; i++)
  {
    scanf("%d", &s[i].id);
    scanf("%s", s[i].name);
    scanf("%f", &s[i].marks);
  }
  // Display the student details
  for(i=0; i<=9; i++)
  {
    printf("id = %d", s[i].id);
    printf("name = %s", s[i].name);
    printf("marks = %f", s[i].mark);
  }
getch();
```



Union:- It is also a user defined datatype.

Similar with structure

① Capable of grouping different types data items in a single unit

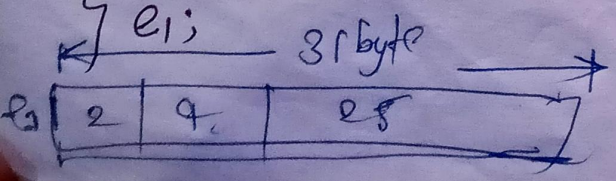
Dissimilar with structure

② The union variable is allocated a common memory equal to the memory size required by the maximum sized data member.

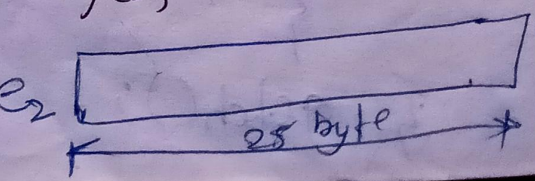
③ All members share the same common memory but one at a time

④ Union are used to efficiently use the memory.

```
struct emp
{
    int id;
    float sal;
    char name[25];
} e1;
```




```
union emp
{
    int id;
    float sal;
    char name[25];
} e2;
```



/* WAP to read Name, id & sallary and display it
Union emp by using union */

```
{ char name[20];
  int id;
  float sal;
} e2;
void main()
{
    printf("Enter name of employee");
    gets(e2.name);
    Puts(e2.name);
    printf("Enter id of employee");
    scanf("%d", &e2.id);
    printf("sallary = %d", e2.id);
    printf("Enter sal of employee");
    scanf("%f", &(e2.sal));
    printf("sallary = %f", e2.sal);
    getch();
}
```

Passing Array to function :-

```
/* Example program to pass 1D Array to function that calculate
                    sum of its elements */
int calcSumArr (int a[], int n) // n for size of array
{
    int i, sum=0;
    for (i=0; i<=n-1; i++)
    {
        sum= sum+ a[i];
    }
    return sum;
}
void main()
{
    int a[100], n, i
    printf("Enter size of array");
    scanf("%d", &n);
    for (i=0; i<=n-1; i++)
    {
        scanf("%d", &a[i]);
    }
    s= sum(a);
    s = calcSumArr(a, n); // function calling and passing the Array along with the size of Array
    printf("sum of element = %d", s);
    getch();
} // End of main function
```

Complex Number :-

```
/* Program to add two input Complex Numbers */
struct comp
{
    float real;
    float img;
};
void main()
{
    struct comp n1, n2, n3;
    printf("Enter 1st ele Complex number");
    &Conf("%f %f", &n1.real, &n1.img);
    printf("Enter 2nd Complex number");
    &Conf("%f %f", &n2.real, &n2.img);
    // logic to sum Complex numbers
    n3.real = n1.real + n2.real;
    n3.img = n1.img + n2.img;
    //output Complex number
    printf("%f.%f", n3.real, n3.img);
    getch();
} // End of main function
```