Patterns

①

```
* * * *
* * * *
* * * *
* * * *
```

```
# include <stdio.h>
void main()
{ int i,j;
 for(i=1; i<=3; i++)  } → Rows
{ for(j=1; j<=4; j++) → Columns
{ printf("*"); }  Ye loop 'i' baar yani ki '3' baar chalega
}
printf("\n"); → 'j' baar yani ki '4' baar print Hone ke baad line change Hogi.
}
}
```

②

```
*
* *
* * *
* * * *
```

```
# include <stdio.h>
void main() {
 int i,j,r;
 printf("Enter the No. of rows");
 scanf("%d",&r);
 for(i=1; i<=r; i++) → Rows;
  { for(j=1; j<=i; j++) → Iiss No. of Rows utne star uss row pe print Karvane ke liye
   { printf("*");
   }
   printf("\n");
   }
 }
```

③

```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
# include <stdio.h>
void main() {
 int i,j,r;
 printf("Enter the No. of rows");
 scanf("%d",&r);
 for(i=1; i<=r; i++) → Rows;
  { for(j=1; j<=r-i+1; j++) {
     printf("%d",j);
  }
  printf("\n");
  }
 }
```

for Example :-

① r = 5
j <= 5-1+1
j <= 5
1 2 3 4 5

② j <= 5-2+1
j <= 4
1 2 3 4

④ Floyds Triangle

```
1
2  3
4  5  6
7  8  9  10
```

```
# include <stdio.h>
void main () {
  int i, j, r, val = 1;
  printf ("Enter the No. of rows");
  scanf ("%d", &r);
  for (i=1, i<=r; i++){
    for (j=1; j<=i; j++) {
      print ("%d", val);
      val++;
    }
    printf("\n");
  }
}
```

Rows Select Karne ke liye

Jis No ki row hai utne No. of element print Karaenga. (value ++).

⑤ Padding

```
# include <stdio.h>
int i, j, r, c;
 char b, p;
printf ("border");
 scanf ("%c", &b);
printf ("padding");
 scanf ("%c", &p);
 printf ("No. of rows and Column");
  scanf("%d %d", &r, &c);
 for (i=1; i<=r; i++)
 {  for (j=1; j<=c; j++)
 {   if (i==1 || i==r || j==1 || j==c)
      print f ("%c", b);
     else
      print f ("%c", p);
   }
    printf ("\n");
  }
}
```

Ye kehte Ha ki first and last, row and column ko 'b' se bhaar do. Baaki sab 'p' se.

if b=0, p=1, r=4, c=0

```
0 0 0 0
0 1 1 0
0 1 1 0
0 0 0 0
```

Arrays → can store multiple values.

(a) Syntax 1D Array → `int arr[10];` (datatype, Name, size)

(b) " 2D " → `int arr[4][4]` (rows, column)

(c) " 3D " → `int arr[10][10][10]`

LOGIC FOR CODES

① Odd / Even

```
odd[i] = 2*i + 1;
even[i] = odd[i] - 1;
```

i=0 → odd[0] = 1, even[0] = 0

i=1 → odd[1] = 3, even[1] = 2

② sum / max / min

sum → `total = total + sum[i];`

min → `if (sum[i] < min) min = sum[i];` | `min = sum[0];`

isko '1' se start karna hoga kyunki i=0 used hai isme

max → `if (sum[i] > max) max = sum[i];` | `max = sum[0];`

③ search element

```
if (x == array[i]) {
    printf("found");
    break;
}
```

isme loop lagana hoga jitne size ka array hai. (for Here it is 10)

so, `for (i=0; i<10; i++)`

```
if (i == 10) {
    printf("Not found");
}
```

④ delete element

<-- phele search karna -->

```
if (count == 1)
for (j=i; j<10; j++)
{
    array[j] = &array[j+1];
}
```

(space bharne ke liye)

maan lo element '4th' loc pe mila so, i=4

j=i, ye saare left wale elements ko move kardega toward right

Yani ith wala delete ho gaya.

Last the extra element

```
size--;
break;
```

isse remove ho jayega.

(i == size) ↓

<-- Not found wali cond^n -->

(5) Insert Element

No of elements

← – – define array, usme Jaha store karna Hai / elements daalo, index value use "loc" me save kar do – – →

```
for (i = loc + 1 ; i < n+1 ; i++)
{ a[i] = a[i-1] ; }
a[loc] = x ;
```

Jaha enter karna Hai uske next index se start Hoga loop.

insertion index ke baad wale elements unse pehle wale se replace kiya. (basically elements Age shift karke space banaya).

```
printf (" Array after insertion ");
for (i=0; i<n+1; i++)
  print ("%d", a[i]);
}
```

2D Array / Matrices

LOGIC FOR CODES

(1) form a 2D Array

```
# include <stdio.h>
void main () {
int x[3][4], i, j;
printf ("Enter array elements : \n");
for (i=0; i<3; i++) {
  for (j=0; j<4; j++) {
    scanf ("%d", &x[i][j];
    }
  }
printf ("Array elements are : \n");
  for (i=0; i<3; i++) {
    for (j=0; j<4; j++) {
      printf ("%d", x[i][j]);
      }
    printf ("\n");
  }
```

Input lenewala (store karega bas)

x[0][1] x[0][2] x[0][3] x[0][4] and so on.....

display lenewala (sirf print karega)

② Addition of Matrix

```
<--- decleration ---->
printf("Enter array 1 elements : \n");
 for(i=0; i<m; i++){
     for(j=0; j<n; j++){
       scanf("%d", &x[i][j]);
      }
     }
<--- same for array 2 ---->
 printf(" After Addition: \n");
   for(i=0; i<m; i++){
    for(j=0; j<n; j++){
      z[i][j]= x[i][j] + y[i][j];
       printf("%d", z[i][j]);
       }
       printf("\n");
       }
      }
```

③ Transpose of matrix

```
<--- deloration ---->
<--- create and display era ---->
     for(i=0; i<m; i++){
       for(j=0; j<n; j++){
          tx[i][j] = x[j][i];
         }
       }

<--- display transpose ---->
        matrix
```

Makes transpore matrix.

④ Multiplication of matrix

<- - - declaration - - ->

```
if (n1 != m2) {        → Row of matrix 2
        printf(" invalid");
    }
```

column of matrix 1

<- - - Take values for both matrix - - ->

```
printf("matrix after multiplication");
  for (i=0; i<m1; i++){
    for (j=0; j<n2; j++){
        sum=0
      for (k=0; k<n1; k++){
        sum = sum + x[i][k] * y[k][j];
      }
      z[i][j] = sum;
     printf("%d", z[i][j]);
    }
    printf("\n");
  }
```

resets for every elements (sum=0)

to run a loop for every element of that column. (for k loop)

STRINGS

1. gets () → store / takes value from the user
2. puts () → helps to print a string with newline character.
3. strlen(str) → length of the string
4. strrev → reverse the string
5. strcmp (str1, str2) → compare two string (returns diff b/w ASCII values if non similar element is found).
6. strcpy (str1, str2) → copy string
7. strcat (str1, str2) → adds two strings
8. strupr (str) → capitilize
7. strlwr (str) → small
9. strcmpi (str1, str2) → if str1 = str2 ⇒ 0; if str1 > str2 ⇒ +ve; if str1 < str2 ⇒ -ve
10. strncpy (str1, str2, n); → copy till this index
11. strncat (str1, str2; n); add till this index of str2.

FUNCTIONS

↳ group of statements that performs a specific task and is relatively independant of the remaining code.

User-Defined function

1. defined by user according to the task want to perform.
2. to use these functions, they should be declared, defined and called.
3. You have to right a logic to run these functions.
4. function name can be changed by the user.
5. part of C language.
6. use of this increases program complexity.

Library function

1. pre defined in a header file (can be used as per the task)
2. can be used by including in header file.
   Eg → <string.h> strcpy (str1, str2)
3. No logic required.
4. Name can't be changed as they are pre-defined
5. part of C Header file
6. makes it simple and easy to program design.

function decleration

```
int sum ( int a , int b)
int sum ( int, int)
```

with parameter name
without " "
parameter datatype

```
int sum ( int a , int b) ;
```

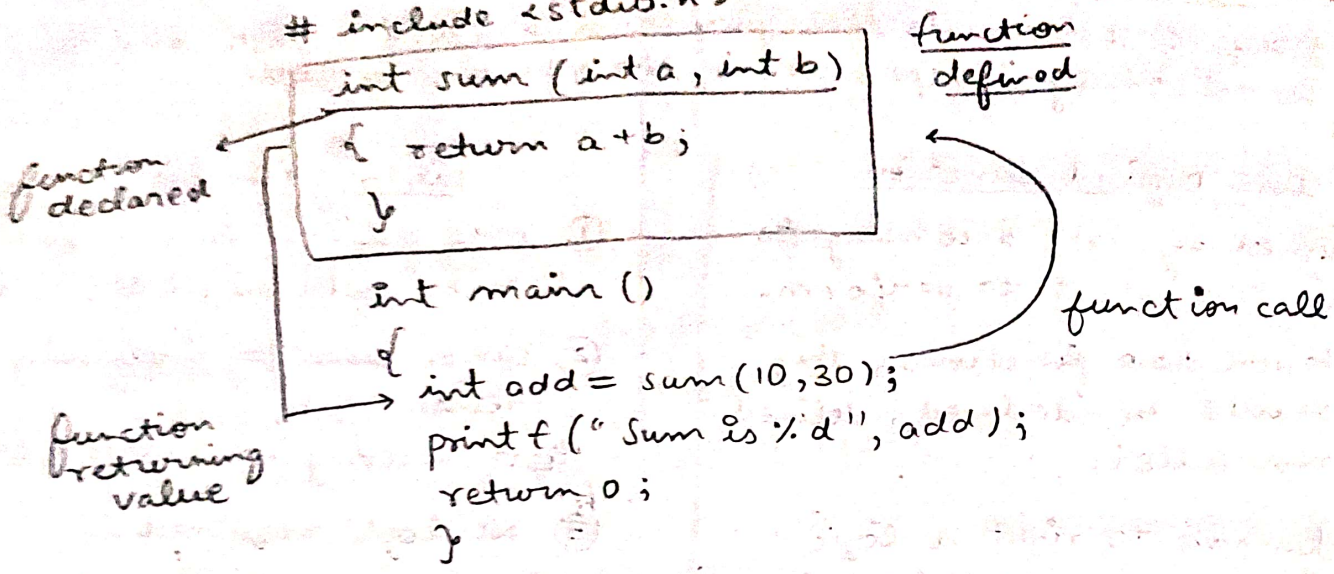
- int → return Datatype
- sum → function Name
- a, b → parameter Name.
- ; → ending statement Semi-colon.

function Defination

```
int heading (void)
{
    #statements
    return 0;
}
```

- int → Return type
- heading → function Name
- (void) → Parameter (Argument)

Function Call and its working



* factorial using function

```
#include <stdio.h>
int fact (int a);
void main ()
{ int a,f;
printf ("Enter a value");
scanf ("%d", &a);
  f = fact (a);
printf ("%d", f);
}
int fact (int a)
{ int i, f = 1;
for (i=a; i=>1; i--); {
  f = f * i; }
  return f;
}
```

function call → Normal code but this is the use of function.

function define

statements

* factorial using Recursion

```
# include <stdio.h>
  int fact (int);
  void main()
  { int a, f;
    printf ("Enter a Number");
    scanf ("%d", &a);
      f = fact(a);  ------> function call
    printf ("%d",f);
    }
int fact (int x);  ------> function define
{ if (x==1)
  return (1);   } condition to stop recursion
  else
   return x * fact(x-1); } recursion call.
}
```

statements

statements ke andar hi func use kara usse function ka.

* Ackermann Using Recursion

```
# include <stdio.h>
  int ackermann (int m, int n);
  int main () {
    int x, y;
    printf ("Enter two No. : ");
    scanf (" %d %d", &x, &y);
    if (x>=0 && y>=0) {
      printf (" Ackermann function for %d and %d gives
          %d\n", x, y, ackermann (x,y));
      }
    else {
    printf( "Both No. must be non-negetive.\n");
    }
    return 0;
  }
int ackermann (int m, int n) {
 if (m==0)
   return n+1;
 else if (m>0 && n==0)
   return ackermann(m-1,1);
```

```
else if (m>0 && n>0)
    return ackermann (m-1, ackermann (m,n-1));
}
```

* **Sum of digits using Recursion**

```
# include <stdio.h>
  int sum_of_digits (int);
  void main()
  { int n;
  printf ("Sum of digits of 3481 is %d", sum_of_digits
                                          (3481));
  printf ("Enter a Number");

  scanf("%d", &n);
  printf("Sum of digits of %d is %d", n, sum_of_digits
                                          (n));

  int sum_of_digits (int n)
  { if (n==0)
    return 0;
    else
   return (n%10)+ sum_of_digits (n/10);
    }
  }
```

Arguement → a value in the form of variable, constant OR expression which can be passed to the function.

Formal Arguement

① variable OR expression mentioned in function defination.
scope of formal arguement is limited to function defination.

② datatype of formal arguement must be mentioned while defining the function.

Actual Arguement

① variable OR const. OR exp. that is contained in function call, that replaces formal ones.

② not required to mention datatype of actual arguement.

```
void add(int a, int b) {          → formal arguements [datatype mentioned]
    int sum = a+b;
    printf("sum : %d \n", sum);
}
```
← function defination

```
int main() {
    int x = 5;
    int y = 10;
    add(x,y);        → function call.
                     → actual arguement (replaces formal ones).
    return 0;
}
```

Return Statement
↳ returns the control to the calling function after the execution of the function.

LOCAL AND GLOBAL VARIABLE

```
# include <stdio.h>
void display();
void main()
{  int x=10;        → local
   display();       → only accesible till here.
}
Void display()
{  x++;
   printf("%d", x);
}
```

```
# include <stdio.h>
int x;          → GLOBAL (can be used any where)
void display();
void main()
{  x = 10;
   display();
}
void display()
{  x++;          // 11
   printf("%d", x);
}
```
x = 11

Pointers

↳ pointer is that which stores memory address of another variable.

* declaration ⇒
```
int * ptr;
char * ptr1;
float * ptr2;
```

Types of pointer

① wild pointer

↳ uninitialized pointer

Ex →
```
int *ptr;
*ptr = 9;
```
} wild pointer ↳ incorrect way

```
int * ptr;
int num = 9;
*ptr = &num
```
<-- correct way -->

② NULL pointer

↳ `float * ptr = NULL;`

③ void pointer

↳ can access or manipulate any kind of datatype by using void pointer. `void * ptr;`

* Accessing array elements using pointer :-

```
# include <stdio.h>
  void main()
{ int x[5] = {0,-10,8,15,33};
  printf("value stored at *x[0] %d \n", *x);   → 0
  printf("value stored at *(x+2) %d \n", *(x+2));   → 8
```

* chain of pointers :-

```
# include <stdio.h>
  void main()
{ int x = 10, *p1, **p2;
  p1 = &x;
  p2 = &p1;
  printf("p2 = %u  *p2 = %u  **p2 = %d \n", p2, *p2, **p2);
```

gives → `*(*p2)` → `*(p1)` = x

* pointer to array of

```
# include <stdio.h>
int main () {
int x[5]= {10,20,30,40,50};
int *ptr[5];     // Array of pointers
 int i;
for (i=0; i<5; i++) {
     ptr[i] = &x[i]; }
   }
for (i=0; i<5; i++) {
printf ("ptr[%d]= %p , *ptr[%d]= %d\n", i , ptr[i] , i , *ptr[i] ;
}
 return 0;
}
```

pointer array me 'x' ki values store kar di

%d → i
%p → pointer location
%d → i
location me jo value stored Hai wo.

function Call

Call by value

```
# include <stdio.h>
void swap (int ,int);
void main ()
{ int a=10 , b=20;
  swap (a,b);
 printf(" After Swapping
a=%d  b=%d ", a, b) ;
}
void swap(int x, int y)
 { x = x+y;
   y = x-y;
   x = x-y;
 }
```

① pass value of variable while calling it.
② value of each variable in calling function is copied into dummy variable.
③ actual and formal arguement are created at the different location.

Call by Reference

```
# include <stdio.h>
void swap (int *x, int *y)
void main ()
{ int a=10 , b=20;
 swap (&a, &b);
 printf (" After swaping
 a=%d  b=%d , a , b);
}
void swap (int *x, int *y)
{ *x = *x + *y ;   30
  *y = *x - *y ;   10
  *x = *x - *y ;   20
}
```

① pass address of variable to the funcn while calling.
② address of actual variable is copied into dummy variable
③ actual and formal arguement are created at the same memory location.

Structures

↳ used to store a collection of different types of data under a single variable name.

Array is collection of Homogeneous [same data type] element

↳ Heterogeneous

Example :- Student data

```
#include <stdio.h>
struct student {
    char name[50];
    int roll_no;
    char section[10];
};
int main() {
    struct student s1 = {"Akshit", 111111, "CSE"};
    struct student s2 = {"Homie", 222222, "CSE"};
    struct student s3;
    printf("Enter name of student 3:");
    scanf("%s", s3.name);
    printf("Enter roll no. of student 3:");
    scanf("%d", &s3.roll_no);
    printf("Enter section of student 3:");
    scanf("%s", &s3.section);
    printf("\n Student 1: \n Name: %s \n
           Roll No: %d \n Section: %s \n", s1.name,
           s1.roll_no, s1.section);
    <--- student 2 --->
    <--- student 3 --->
    return 0;
}
```

* Array of structure

```
#include <stdio.h>
struct Student {
char name [50];
 int roll_no;
char section[10];
};
void main () {
  struct Student std[5];
for (int i=0; i<5; i++){
   printf ("Enter data for student %d : \n", i+1);
 printf ("Enter a name : ");
 scanf ("%s", std[i].name);
 printf ("Enter a roll Number : ");
  scanf ("%d", &std[i].roll_no);
 <--- section --->
printf ("\n Student Details : \n");
 for (int j=0; j<5; j++){
    printf ("Student %d :", j+1);
    printf ("Name : %s :", std[j].name);
                ↑ roll No.
                | section
                ↓
}
}
```

to access all array element

to display all array element

File Handling

fopen(), fclose(), fputc(), fgetc() character, fgets(), fputs()

fgetc() → Read a character from a line

fgets() → Reads a string from a file

fputs() → write a string

fprintf() → write formatted data to a file

fscanf() → Reads formatted data of a file

* WAP to write array elements to the file.

```
#include <stdio.h>
void main()
{ int x[10];
  FILE * file
  file = fopen("b.txt","w");
  if ( file == NULL) {
  print("Cannot Open the file");
   exit(1);
  }
  for (int i=0; i<10; i++)
  { printf("Enter a Number");
    Scanf("%d", &x[i]);
    fprintf( file, "%d\n", x[i]);
  }
  fclose();
}
```

Common part.

data to write

first select file

assign data.

* LOGIC to read content of file

```
<-- common part -->
declare char ch;
ch = fgetc (file);
while (ch != EOF)
 { print ("%c", ch);
   ch = fgetc(fp);
 }
fclose (file);
}
```

FILE is a structure defined in header file <stdio.h>.
It offers many func^n s (predefined) like fputc(), fgetc(), etc.

* Copy content from one file to another

<-- common part Lekin isme 2 file kholne dono ka NULL check karna padega - - ->

```
ch = fgetc (fp1);
while (ch != EOF)
{ fprintf (fp2, "%c", ch);
        OR
  fputc (ch, fp2);
}
fclose (fp1);
fclose (fp2);
}
```

* WAP to count No. of character and words in a file

<-- common part - - -> chcount = 0, wordcount = 0;

```
ch = fgetc (file)
while (ch != EOF)
{ count ch++;
  if (ch == ' ' || ch == '\n' || ch == '\t' || ch == ',' ||
      ch == ';' || ch == ':' || ch == '.')
  wordCount++;
  ch = fgetc (file);
}
if (ch count > 0 && wordcount == 0) {
wordCount = 1; }
else
  printf ("Empty file");
fclose (file);
if (chcount > 0) {
  printf ("Number of characters : %d \n", ch count);
  printf ("Number of words : %d \n", wordcount);
}
```

agar character Hai file me

word count karte waqt agar ye aayya to matlab 9k word complete

but word count Nhi Hua, Yani koi delimiter Nhi aayya so. Usko 9k word man Lenge Hum.

Dynamic Memory Allocation

static memory Allocation
- memory allocation while declaring an array of fix size.
- size of allocated No. can't be changed during program.

Dynamic Allocation
- allocating memory at the time of execution. ~~is called dynamic~~
- the allocation and release of memory can be done with the help of some library functions, declared in <alloc.h> and <stdlib.h>
- pointers play an important role here.

① Malloc function
- used to allocate memory dynamically
- The arguement size passed to the function specifies No. of bytes to be allocated.
  - * On success returns a pointer to the first byte of allocated memory.
  - * and NULL if the allocation fails.
- does not initialize memory allocated during execution. The memory allocated carries garbage values.

② Calloc function (Contiguous Allocation)
- used to allocate multiple blocks of memory of given size.

difference b/w calloc and malloc

① calloc takes 2 arguement (No. of block, size of block); malloc take 1 ''

② calloc initializes to zero while malloc contains garbage value

③ calloc → multiple '' '' ''
malloc → single block of memory

calloc → [complex data structures.]
malloc → simple data structure (used for)

③ free function

↳ the dynamically allocated memory is not automatically released. To free this so that it can be used again we use free function. (returns back to heap memory.

④ Realloc function

↳ used to increase OR decrease the memory allocated by malloc function OR calloc function.

↳ alters the size of memory block without loosing the stored data.

↳ takes 2 arguement

↙ pointer to the block

↘ New size