UNIT-3,4 P.P.S.

Deepak Sir **FUNCTIONS IN C** 36

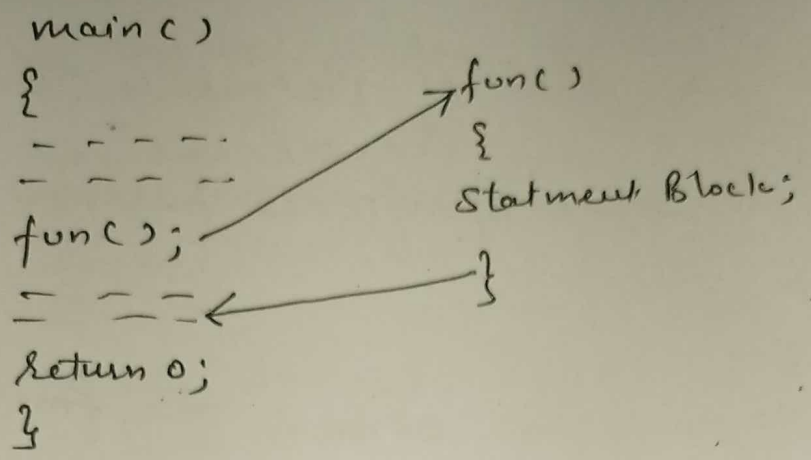
→ A function is a set of statements that take inputs, do some specific computation & produce output

→ Every 'c' program has atleast one function, which is 'main()'.

Why do we need functions :-

1) function helps to reducing code redundancy.
2) functions make code modular means consider a big file having many lines of codes. It becomes really simple to read & use the code if the code is divided into functions.
3) Understanding, coding & Testing multiple separate functions are far easier than doing the same for one huge function.
4) functions provide abstraction. for example, we can use library functions without worrying about their internal working.

Function Interface :-



→ From above we can see that main() calls the function named fun().

→ So, main() is known as Calling function & fun() is called Called function.

→ While using function we may use following terms :

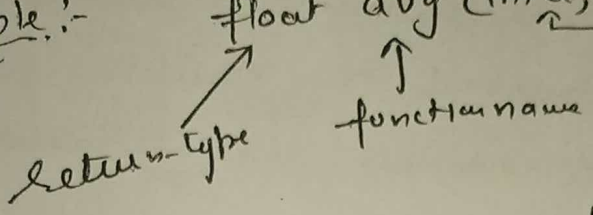
1) Function Declaration / Prototype :-

→ Before using the function, compiler must know about the number of parameter & type of parameter that the function expect to receive & the data type of the value that it will return to calling program.

→ Therefore function declaration enable the compiler to check on the arguments used while calling that function.

→ Syntax :- Return type functionname (datatype Variable1, - - -);

Example :- float avg (int a, int b);



→ After declaration of every function, there should be a Semicolon. If Semicolon is missing then compiler will generate error.

2. Function Definition :-

→ when function is defined, space is allocated for that function in the memory.

A function definition have Two part :

- function header
- function body

Syntax :- return-type function_name (data-type variable1, - -)
{
- - -
statements
- - -
}

Example :-
```
float avg (int a, int b)
{
int c;
c = a+b
}
```

The number, of argument & the order of arguments in function header must be same as that given in the function declaration.

→ The function Call statement invokes the function.
→ When a function is invoked the compiler jumps to the Called function to execute the statement that are the part of that function.
→ Once the Called function is executed, the program is passed back to the Calling function.

Syntax :- function_name (Variable 1, Variable 2 - - - -.);

example :- result (10);

or

```
int a=10;
result(a);
```

4 Return statement :- The return statement to terminate the execution of function & control to the calling function.

Syntax :- return <expression>;

Example :-

```
int sum (int a, int b)
{
int c;
c= a+b;
return c;
}
```

Program to add two integers using function.

```
#include <stdio.h>
// FUNCTION DECLARATION
int sum(int, int);
int main()
{
int a, b, sum;
scanf("%d%d", &a, &b);
sum = total(a, b); // FUNCTION CALLING
printf("%d", sum);
}
// FUNCTION DEFINATION
int total(int x, int y)
{
int z;
z = x+y;
return z;
}
```

output:- 20 , 30

sum = 50

Difference between actual argument and formal argument

Actual Arguments:- The arguments that are passed in a function call are called actual arguments.

→ They arguments defined in the calling function.

→ Syntax:- functionname (Variable1, Variable2 ...); //calling fonctn.
(Variable1, Variable2 ...) ← Actual argument

Example:- Avg(int a, int b);
or
Avg (a, b) ;
a, b ← actual arguments

2. **formal Arguments:-** They arguments are defined in the function declaration.

→ The scope of formal arguments is local to the function defination in which they are used.

→ They belong to called function.

Syntax :- functionname (datatype Variable1, ----)
↑ formal Arguments
{
statements;
}.

Example:-

```
main()
{
sum(int a, int b);
}
void sum(int x, int y)
{
int z;
z = x+y;
}
```

formal arguments.

M. Imp.

Parameter passing to the function

→ There are two ways in which arguments can be passed to the called function.

1) Call by Value :- In this value of the variable passed by the calling function to the called function.

2) Call by reference :- In this address of the Variable are passed by the calling function to the called function.

Difference between Call by Value & Call by Ref

Call by value

1) In Call by value, copy of actual argument is passed to formal arguments of the Called function.

2) Any changes made to the formal argument in the Called function have no effect on the value of actual argument in the calling function.

3) WAP to Swap two numbers using Call by value :-

```
#include<stdio.h>
Void swap(int, int);
Void main()
{
int a, b;
a=10, b=20;
swap(a, b)
Printf("%d %d", a, b);
}
Void swap(int a, int b)
{
int temp;
temp = a;
a = b;
b = temp;
}
```

output a=10, b=20

Call by refrence

1) The adress of actual argument is passed to formal arguments of the Called function

or

2) By accessing the address of actual argument, we can alter them within from the Called function.

3) WAP to Swap two numbers using Call by reference.

```
#include<stdio.h>
Void swap(int*, int*)
Void main()
{
int a, b;
a=10, b=20;
swap(&a, &b);
Printf("%d %d", a, b);
}
Void swap(int *a, int *b)
{
int temp;
temp = *a;
*a = *b;
*b = temp;
}
```

output : a=20, b=10

Library function	User defined function
1. These functions are predefined in C language	1. These fun are created by users.
2. Library function needs header files.	2. User define function not need header files.
3. Library function are defined in header file	3. User define function are defined at the end main() program.
4. Example of library function are printf(), scanf(), clrscr() etc.	4. Example of user def function are avg(a,b

RECURSION :-

→ It is a process in which function calls itself again & again untill certain condition is satisfied.

→ A very simple example of recursion is as:

```
main()
{
Printf("Hello");
main();
}
```

Type of Recursion :-

1) Direct Recursion :- A function is said to be directly recursive if it explicitly calls itself.

Example :-
```
int func(int n)
  {
    if (n==0)
      return n;
    return (fun(n-1));
  }.
```

2) Indirect Recursion :- A function is said to be indirect recursive if it contains a

Call to another function which ultimately calls it.

Example:-

```
int fun1(int n)
{
    if (n==0)
     return n;
  return fun2(n);
}
int fun2(int x)
{
  return fun1(x-1);
}
```

WAP to evaluate factorial of integer using recursion

```
fact(int n)
{
    int result;
    if(n==1)
      return(1);
    else
      result = n * fact(n-1);
      return(result);
}
```

Advantage of Recursion:-

1) This Recursive function are written with less number of statment.
2) Recursion is usefull for branching process.
3) Recursion is powerfull in mathematical defination

Disadvantage of Recursion.-

1) It consume large space becaus recursive functin save in Stack
2) Recusion require good programing skill.
3) It is not more efficient in term of speed of execution.

UNIT-3

Array :- An Array is a collection of similar data elements.

→ The elements have the same data type.

→ The elements of the array are stored in consecutive memory location & referenced by index.

→ Some of examples of arrays are:

- List of students in a class
- List of customers
- List of employees in Company.

Declaration of Array :-

→ The array must be declare before being used.

→ Declaring of array must specify three things:

- Data type – Kind of value it store, for Exap int, char, float, double
- name – to identify the array
- Size – Maximum value that array can hold.

Syntax :- Data_type name[size];

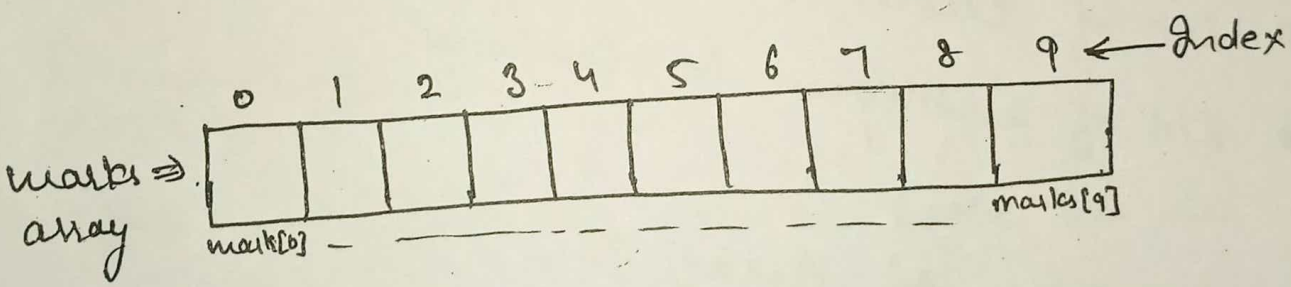
for Example :- int marks[10];

→ The above statement declares a marks variable to be an array containing 10 elements.

→ In 'C', array index start from Zero.

→ This means the first element will be stored in marks[0], second element in marks[1] & so on.

→ The last element will be stored at marks[9].



Accessing elements of the array

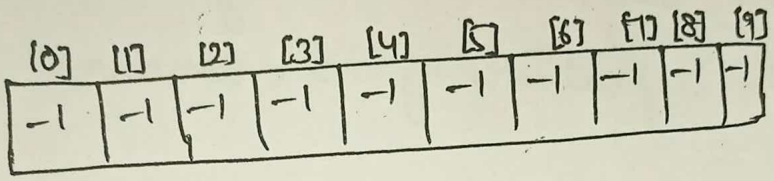
→ To access all the elements of the array, you must use a loop.

→ The first element of the array marks[10] can be accessed by writing, mark[0].

→ Now to process all the elements of array, we

use loops as:

```
int i, marks[10];
for(i=0; i<10; i++)
{
 marks[i] = -1;
}
```



→ The above code set the value -1 to each elem. of array.

→

STORING VALUES IN ARRAYS

→ To store the value in the array, there are three ways -

1. initialize the array element
2. Input value for the element
3. Assign value to the element.

① Intialization of array:-

→ The array can be intialized at the time of declaration.

→ When an array is initialized, we need to provide a value for every element in the array.

Syntax :- type array_name[size] = {list of value};

→ The value are written with curly brackets & every value is seprated by a comma.

for Example :- int marks[5] = {90, 82, 78, 95, 88};

marks[0]	90
marks[1]	82
marks[2]	78
marks[3]	95
marks[4]	88

Inputting Value :-

→ The array can be filled by inputting value from the keybord by using for loop as:

```
int i, marks[10];
for( i=0; i<10; i++)
{
Scanf("%d", a[i]);
}
```

3. Assigning Value:-

→ we can assign value to individual element of array by using assignment operator as

```
marks[3] = 100;
```

→ Here, 100 is assigned to the fourth element of the array which is specified as marks[3].

Example of Array

WAP to read & display 'n' number using Array

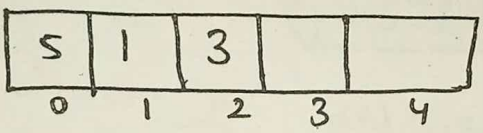
```
#include <stdio.h>
#include <conio.h>
main()
{
int i, n, arr[20];
Scanf("%d", &n);
for(i=0; i<n; i++)
{
scanf("%d", &arr[i]);
}
for(i=0; i<n; i++)
{
printf("%d", arr[i]);
}
getch();
}
```

Operation on Array :-

(1) Traversing :- It means accessing each & every element of array atleast once.

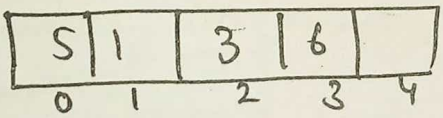
(2) Insertion :- It means adding new element in an already existing array.

For example :- int a[5] = {5, 1, 3};



(Before insert)

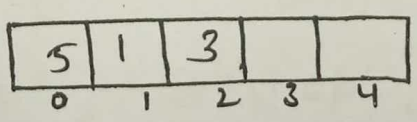
Now insert '6' at end of array.



(after insert)

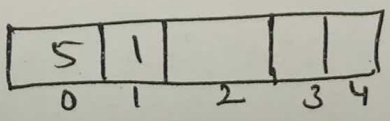
3 Deletion :- It means removing a data element from an already existing array.

for example :- int a[5] = {5, 1, 3}



(Before Delet)

Now Delet '3'

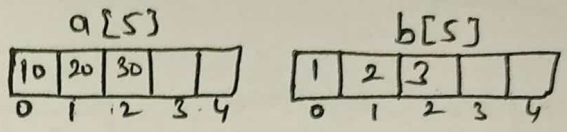


(After Delet '3')

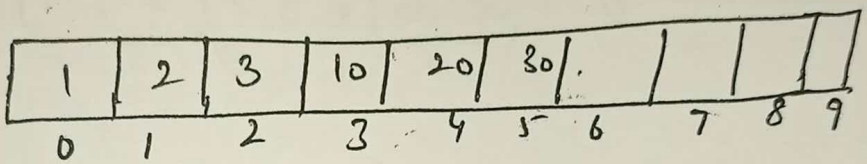
4 Merging :- It means Combine two array into a single array.

for Example:- int a[5] = {10, 20, 30};
int a b[5] = {1, 2, 3, 4};

Now after Merging :-

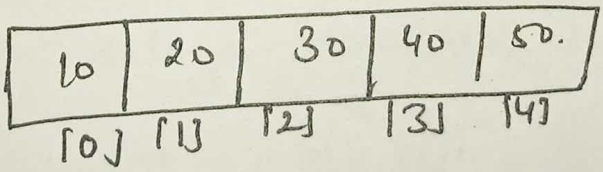


c[10] =



5 Searching :- It means find whether a particular value is present in the array or not.

for Example:- int a[5] = {10, 20, 30, 40, 50};



Now Search '30'

Printf("Search is succesfull");

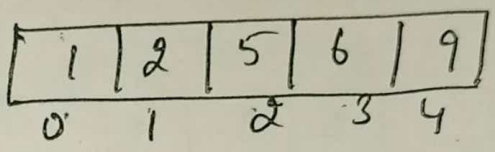
→ Two type of Search we can use as:

- Linear Search
- Binary Search.

6 Sorting :- It means arrang the unsorted array element in either ascending or decending order.

For Example:- int a[5] = { 5, 9, 2, 6, 1 };

After Sorting :-



Answer.

TYPE OF SORTING.

TYPES OF ARRAYS

→ In 'c' Language, arrays are classified into 2 types:

1) one-dimensional array
2) two-dimensional array.

1) 1-Dimensional Array :- This array is used to store list of values of same data type.

Declaration :- datatype arrayname [size];

Example :- int marks[10];

Initialization :- datatype name[size] = {value1, value2, ...};

Example :- int marks[10] = {2, 3, 4};

Accessing 1-D array :- name[index];

```
int i,a[5];
for(i=1 ; i<=5 ; i++)
{
printf("%d", a[i]);
}
```

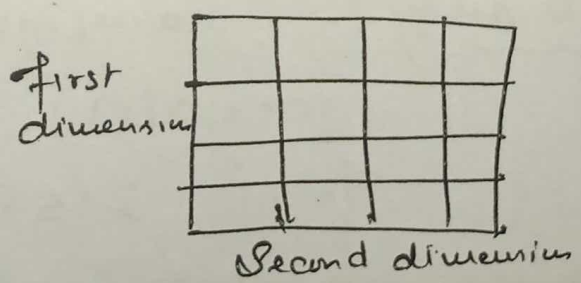
Example program for 1-D array:-

```
#include <stdio.h>
#include <conio.h>
main()
{
int i, a[5] = {10,20,30,40,50};
for(i=0; i<5 ; i++)
{
printf("%d", a[i]);
}
getch()
}
```

2. 2-D array :- An array of array is known as 2-D array.

→ The 2-D array in 'C' language is also known as matrix.

→ A 2-D array is specified using two subscript where one subscript denotes rows & other denotes column.



Declaration of two-Dimensional Array:-

Syntax: datatype arrayname [rowsize] [columnsize];

→ A 2-Dimensional m x n array is an array that contains m x n data element & each element is accessed using two subscript, i & j where i<=m & j<=n

→ for example:- int marks[3][5];

Row \ Colum	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	marks[0][0]	marks[0][1]	marks[0][2]	marks[0][3]	marks[0][4]
Row 1	marks[1][0]	marks[1][1]	marks[1][2]	marks[1][3]	marks[1][4]
Row 2	marks[2][0]	marks[2][1]	marks[2][2]	marks[2][3]	marks[2][4]

Intialization of 2-D array:-

→ 2-D array is intialized same as 1-D array as:

int marks[2][3] = {90, 87, 78, 68, 62, 71};

→ In order to initialized 2-D array to zero:

int marks[2][3] = {0};

Accessing 2-D array:-

→ In order to access the 2-D array, we for loop as:

```
for(i=0; i<2; i++)
    for(j=0; j<2; j++)
        scanf("%d", a[i][j]);
```

Example of 2-D array

Program to print elements of 2-D array :-

```
#include <stdio.h>
# includ <conio.h>
main()
{
int i,j ;
int a[2][2] = {12, 34, 56, 32} ;
for(i=0 ; i<2 ; i++)
   {
   for(j=0 ; j<2 ; j++)
      {
      printf ("%d", a[i][j]);
      }
   }
getch();
}
```

PASSING ARRAYS TO FUNCTIONS

→ Like a Simple Variable, we can also pass the values of an array to a function.

→ We can pass array as one-dimensional to a called function by simple pass the name of array without any subscripts and the size of array as argument

→ for Example:-

```
Largest(a,n)
```

Here 'a' is the name of array & 'n' is the size of array pass to called function as largest().

→ The declaration of the formal arguments array is define as:

```
float array[ ];
```

Rules to Pass an Array to a function

1) The function must be called by passing only the name of array.
2) In the function definition, the formal parameter must be an array type. The size of the array does not need to be specified.
3) The function prototype must show that the argument is an array.

Example :-

```
#include <stdio.h>
void display (int n);
int main()
{
int i;
int marks[] = {55, 60, 70, 80};
for(i=0; i<=6; i++)
{
display (marks[i]);
}
return 0;
}
void display (int m)
{
printf ("%d", m);
}
```

(Passing Array value)

```
#include <stdio.h>
void update (int [], int);
main()
{
int i;
int add[]={1,2,3,4,5};
update(add,5);
for(i=0; i<5; i++)
{
printf("%d", add[i]);
}
}
void update(int b[], int z)
{
int i;
for(i=0; i<z; i++)
{
b[i] = b[i]+5;
}
```

(Passing Entire Array)

RETURN ARRAY FROM FUNCTION

→ 'C' programing does not allow to return an entire array as an argument to a function.

→ There are three methodes to return one-dimensional array from a function.

1. using dynamic allocated array
2. using static array

(3) using structure

(1) using Dynamically Allocated Array :-

→ Dynamic allocated memory remains their untill we delet it using delete or free().

→ we can create a dynamically allocated array & we can delete it once we come out of function.

→ Example :-

```
#include <stdio.h>
int* fun( int *arr)
{
  arr[0] = 10;
  arr[1] = 20;
  return arr;
}
int main()
{
   int arr [100];
   int* ptr= fun(arr);
   Printf ("%d,%d", ptr[0], ptr[1]);
   return 0;
}.
```

output :- 10, 20

2 using static array :-

→ Lifetime of static Variable is throughout the program So we can always create a local static array

Example:-

```
#include<stdio.h>
int* fun()
{
    static int arr[100];
    arr[0] = 10;
    arr[1] = 20;
    return arr;
}
int main()
{
    int* ptr = fun();
    printf("%d %d", ptr[0], ptr[1]);
    return 0;
}
```

output: 10 20

3. Using structure:-

→ We can wrap array in structure & return the instance of the structure.

→ The reason for this is, array members of structure are deeply copied.

→ Example:-

```
#include <stdio.h>
struct anwrap
{
  int an[100];
  };
struct anwrap fun()
{
      struct anwrap x;
      x.an[0] = 10;
      x.an[1] = 20;
    return x;
  }
int main()
{
struct anwrap x = fun();
printf("%d %d", x.an[0], x.an[1]);
return 0;
}
```

output :- 10 20.