call by value & call by reference,

DATE: / 20
PAGE No

function

- * function is a block of statements that performs a specific task.
- used to perform same task more than once.
- Types of functions:

- Predefined (standard Library function):
  These are the functions which are already have a defination in header files, so we just call them whenever there is a need to use them. ex: puts(), gets(), printf(), scanf().

- User defined function
  The function that we create in a program are known as user defined functions.

* Need of functions:
- To improve the readability
- To Improve the reusability
- Debugging would be easier
- Reduces the size of the code

void

DATE : / 20
PAGE No

* To creat the user defined f^n :
  func. declaration
  " Defination
  " calling

* Function Declaration:
- A function declaration tells the compiler about function name, return type, data type of parameters and how to call the function

- Syntax:
  return_type function_name (parameter list);

- Ex:
  int sum (int, int);

* function Defination:
- A function defination consist of a function name, body, return type and parameters list.

- The return statement terminates the execution of a function and returns a value to the calling function.

DATE : / 20
PAGE No

- Syntax:

```
return_type function_name (Parameters)
{
    Body of the function
}
```

- Ex:

```
int Sum(int a, int b)
{
    return (a+b);
}
```

Ex:

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

```
void sum (int a, int b)
{
    int c;
    c=a+b;
    printf("%d", c)
}
```

* Function Calling :-
When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and return the program control back to the main program.

DATE : / 20
PAGE No

- Syntax:

Var_name = function_name (arguments);

- Ex

result = sum (a, b);

Ex/ void main()
{
int a = 10, b = 20, res;
res = sum (a, b);
Printf ("%d", res);
}

Q. Arithmetic operation with functions!

- int sum (int a, int b)
{
int sum
Sum = a+b;
return (Sum);
}

- int sub (int a, int b)
{
int sub
Sub = a-b;
return (sub);
}

int mul (int a, int b)
{
int mul
mull = a*b;
return (mul);
}

- int div (int a, int b)
{
int div
div = a/b;
return (div);
}

DATE: / 20
PAGE No

```
int sum (int, int);
int sub (int, int);
int mult (int, int);
int div (int, int);
int sum (int a, int b)
  {
     return (a+b);
  }

int sub (int a, int b)
  {
     return (a-b);
  }

int mult (int a, int b)
  {
     return (a*b);
  }

int div (int, int b)
  {
     return (a/b);
  }
```

DATE : / 20
PAGE No

* Call by value :-

When we pass the actual parameters while calling a function then this is known as function call by value.

In this case the value of actual parameters are copied to formal parameters. Thus operation perform on the formal parameters don't reflect in the actual parameters

Ex:
```
#include <stdio.h>
#include <conio.h>
void swap (int, int);
void swap (int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
void main()
{
    clrscr();
    int x, y;
    printf("Enter the two numbers. \n");
    scanf("%d %d", &x, &y);
    swap(x, y)
    printf("the value of x and y after swapping are \n");
    printf("x = %d \n y = %d", x, y);
    getch();
}
```

Output:
Enter the two no.
1
2
the value of x and y after swapping are
x = 1
y = 2

DATE : / 20
PAGE No

* Call by reference :-
When we call a function by passing the addresses of actual parameters then this way of calling the function is known as call by reference.

In call by reference, the operations performed on formal parameters, affects the value of actual parameters because all the operation performed on the stored in the address of actual parameters.

Ex:-
```
#include <stdio.h>
#include <conio.h>
void swap (int *, int *);
void swap (int *a ,int *b)
 {
     int temp;
      temp = *a;
      *a = *b;
      *b = temp;
     }
 void main()
    {
        clrscr();
       int x,y;
```

DATE : / 20
PAGE No

```
Printf("Enter the two numbers \n");
Scanf("%d %d", &x, &y);
Swap(&x, &y);
Printf("the value of x and y after
        swapping are \n");
Printf("x = %d \n y = %d", x, y);
getch();
}
```

Output: enter the two number.

1

2

the value of x and y after swapping are

x = 2

y = 1

27/3/19

DATE: / 20
PAGE No

Recursion:

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.

- Two main components.

1. **Base case:**
   Returns a value without making any subsequent Recursive calls.

   (where we stop)

2. **Reduction step:**
   Reduce the value of input at each step.

Ex: factorial

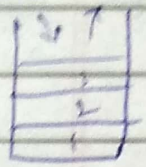
```
int fact (int n)
{
    if (n<=1)                    // base case
        return 1;

    else
        return n*fact(n-1);      // Reduction step.
}
```

DATE : / 20
PAGE No

* int fact(int);
void main()
{
int f;
f = fact(4);
printf("%d", f);
}

LIFO



fact(4)
↳ return 4 * fact(3)
↓
return 3 * fact(2)
↓
return 2 * fact(1)
↓
return 1.

Imp

* Fibonacci series :-

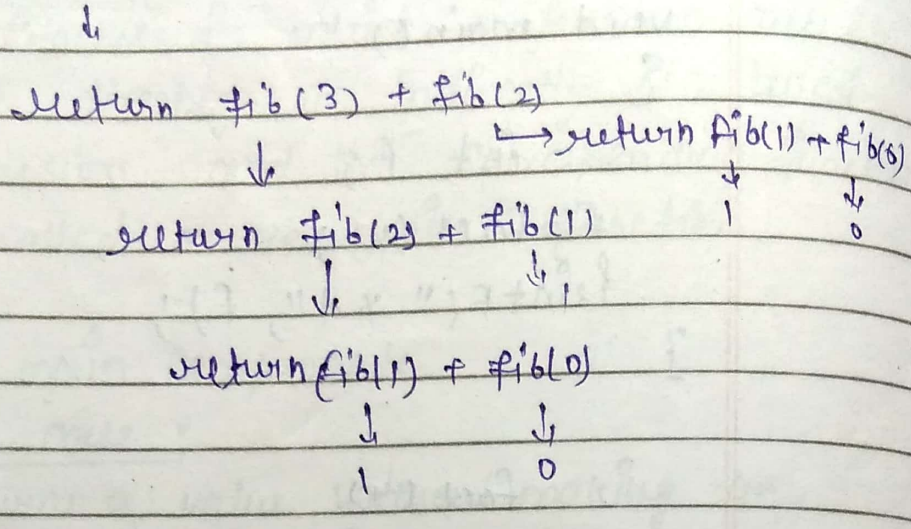
int fibonacci (int n)
{
if (n == 0 || n == 1) // base case
return n;

else
return fibonacci(n-1) + fibonacci(n-2);
// reduction step.
}

where n = nth term

DATE: / 20
PAGE No

Ex: fib(4)



Ackermann function:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \quad \text{(Base case)} \\ A(m-1, 1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0 \quad \text{(Reduction step)} \end{cases}$$

```
{
    if (m==0)
        return n+1;
    else
    else {
        if (m>0 && n==0)
            return A(m-1, 1);
        else
            return A(m-1, A(m, n-1));
    }
```

DATE : / 20
PAGE No

```
{
    if (m==0)
        return n+1;

    else if ( m>0 && n==0)
        return A (m-1, 1);

    else if (m>0 && n>0 )

        return( A (m-1, A(m, n-1)) );
    else
    {
    }
}
{
    if (m==0)                    // base case
        return n+1;

    else if (m>0 && n==0)
        return acker (m-1, 1);

    else
        return acker(m-1, acker(m, n-1));
}
```

DATE : / 20
PAGE No

Q. write a Program to sum digits of a no. without using loops.

Ans

```
int sum(int n)
{
    if (n<=1)
        return n;

    else
        return n + sum(n-1);
}
```

Loop:

```
while (n>0)
{
sum = sum + n%10;
n = n/10;
}
```

Recursion:

```
int sum(int n)
{
    if (n<10)
        return n;
    else
    {
        d = n%10;
    return d + sum(n/10)
```

DATE : / 20
PAGE No

28/3/19

* Divide - and - Conquer :

- A divide-and conquer algorithm has three parts.

1. Divide the problem into a number of sub-problems that are smaller instances of the same problem.

2. Conquer the sub-problems by solving them recursively. If they are small enough, solve the sub-problems as base cases.

3. Combine the solutions to the sub-problems into the solution for the original problem.

1. Quick sort :

Algorithm

```
Void quick_sort(int A[], int start, int end)
{
    if (start < end)
    {
        // stores the position of pivot element
        int Piv_Pos = Partition (A, start, end);

        // Sorts the left side of pivot.
        quick_sort (A, start, Piv_Pos - 1);

        // Sort the right side of pivot.
        quick_sort (A, Piv_Pos + 1, end);
    }
}
```

DATE: / 20
PAGE No

```
Pnt Partition (int A[], int start, int end)
{
   int i = start + 1;
   int Piv = A[start]; // make the first element as
                          Pivot element.
for (int j = start + 1; j <= end; j++)
{
   /* reg rearrange the array by putting
   elements which are less than Pivot on
one side and which are greater that
   on other. */

   if (A[j] < Piv)
    {
        Swap(A[i], A[j]);
          i++;
    }
}
Swap (A[start], A[i-1]);

   // Put the Pivot element in its proper place

return i-1;

   // return the position of the Pivot
}.
```

In Exam.
algorithm
or
example

DATE : / 20
PAGE No

⇒ [30] [40] 10 70 60 50 [20]
P l r

40 > 30 stop
20 < 30 stop
Swap.

[30] 20 10 [70] 60 50 40
P
l → l r ← r
r ← r ←
stop

10 20 [30] 70 60 50 40
pivot

⇒

30	40	10	70	60	50	20
0	1	2	3	4	5	6