JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8251 – C PROGRAMMING

(Regulation 2017)

Year/Semester: I/II CSE

2020 – 2021

Prepared by

Mr.H.Shine

Assistant Professor/CSE

UNIT - III - FUNCTIONS AND POINTERS :-

- Introduction to functions: Function Prototype, function definition, function call, Built-in functions (string functions, math functions) - Recursion - Example program: Computation of sine series, scientific Calculator using built-in functions, Binary Search operators - pointer arithmetic - Arrays and pointers - Array of pointers - Example program: Sorting of names - Parameter passing: Pass by value, Pass by reference - Example program: swapping of two numbers and changing the value of a variable using pass by reference.

Introduction to functions:-

✓ C - function is a self-contained block of statements that can be executed repeatedly whenever we need it.

Benefits of using function in C:-

✓ The function provides modularity.

✓ The function provides reusuable code.

✓ In large programs, debugging and editing tasks is easy with the use of functions.

✓ The Program can be modularized into smaller parts.

✓ Separate function independently can be developed according to the needs.

Types of functions:-

✓ Built in function:-

✓ These functions are provid-ed by the system and stored in the library, therefore it is also called library Functions.

ex:- scanf(), printf(), strcpy, strlwr, strcmp, strlen, strcat etc.

✓ To use these functions, you just need to include the appropriate C header files.

✓ User defined Functions:-

✓ These functions are defined by the user at the time of writing the program.

⇒ Parts of Functions:-

1. Function declaration [Proto types]
2. Function Definition
3. Function Call.

1) Function Prototype [Function Declarations]

Syntax:-

```
datatype functionName (Parameter list)
```

Example:-

```
int addition();
```

2) Function Definition:-

Syntax:-

```
return Type functionName(Function arguments)
{
body of the function
}
```

Example:-

```
int addition()
{

}
```

3) Calling a function in C:-

Syntax:-

```
function Name(Function arguments)
```

Example:

✓ Program to illustrate addition of two Numbers using user defined Functions.

```
#include <stdio.h>
/* function declaration */
int addition();
```

```
int main()
 }
 /* local variables definition */
  int answer;
 /* calling a function to get addition value */
 answer = addition();
 printf("The addition of 2 number : %d \n",
                              answer);
 return 0;
 }
/* function returning the addition of 2 Nos */
 int addition()
  {
  int num1 =10, num2 = 5;
  return num1+num2;
  }
```

Output:-

The addition of 2 numbers: 15

Built-in Functions (String functions, math function)

Math Functions:-

✓ C Programming allows us to Perform mathematical operations through the functions defined in <math.h> header file.

✓ The <math.h> header file contains various methods for performing mathematical operations such as

- ✓ sqrt()
- ✓ pow()
- ✓ ceil()
- ✓ floor() etc.

✓ There are various methods in math.h header file. The commonly used functions of math.h header file are given below.

S.NO	Function	Description
1.	ceil(number)	Rounds up the given number. It returns the integer value which is greater than or equal to given number.
2.	floor(number)	Rounds down the given number. It returns the integer value which is less than or equal to given numbers.
3.	sqrt(number)	Returns the square root of given number.
4.	pow(base, exponent)	Returns the power of given number
5.	abs(number)	Returns the absolute value of given numb

Example Programs:-

```
#include <stdio.h>
#include <math.h>
int main()
```

```
    printf ("\n %f ", ceil (3.6));
    printf ("\n %f", ceil (3.3));
    printf ("\n %f", floor (3.6));
    printf ("\n %f", sqrt (16));
    printf ("\n %f", pow (2,4));
    printf ("\n %d", abs (-12));
    return 0;
}
```

Output: 4.000000
4.000000
3.000000
4.000000
16.000000
12.

String Functions:-

✓ There are many important string functions defined in "string.h" library.

S.NO	Function	Description
1)	strlen(strname)	Returns the length of string name.
2.	strcpy(destination, source)	Copies the contents of source string to destination string.
3.	strcat(first string, second string)	Concatenates or joins first string with second string. The result of the string is stored in first string.
4.	strcmp(first string, second string)	Compares the first string with second string. If both strings are same it returns 0.

S.NO	Functions	Description.
5.	Strrev (string)	Returns reverse string
6.	strlwr (string)	Returns string characters in lowercase.
7.	strupr (string)	Returns string characters in uppercase.

Example Program:-

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1="hello", str2="welcome", str3

printf("Length of string %d", strlen(str1));
printf("join string:% s", strcat(str1, str2));
strcpy (str3, str2);
printf("Copy string: %d ", str3);
printf("Reverse string : %s", strrev(str1));
printf("lower case : %s", strlwr(str1));
printf("Upper Case : %s", strupr (str1));

getch();
}
```

Output:-

```
Length of String  5
join string : hellowelcome
Copy string : Welcome
Reverse string : olleh
lower case : hello
UpperCase : HELLO.
```

Recursion:-

✓ A function that calls itself is known as a recursive function.

Direct & indirect Recursion:-

Direct recursion:-

✓ A function is directly recursive if it calls itself.

```
A()
{
------
------
A(); // call to itself
------
}
```

Indirect recursion:-

✓ Function calls another function, which in turn calls the original function.

```
A()
{
-------
-------
B();
--------
}
B()
{
------
------
A(); // function B calls A
-------
}
```

Consider the calcalation of 6! (6 factorial)

i.e $6! = 6*5*4*3*2*1$

$6! = 6*5!$

$6! = 6*(6-1)!$

$$n! = n*(n-1)!$$

Example Program:

```
#include<stdio.h>
#include<conio.h>
void main()  → int fact (int);
{
printf("
int n;
printf("Enter the number:\n");
scanf("%d",&n);
```

```
    f= fact (num);
    printf("\n factorial of %d is %d", num, f);
    getch();
}
int fact (int n)
 {
  if (n==1)
     return 1;
  else
     return (n * fact(n-1));
 }
```

Output:-

Enter a number : 5

Factorial of 5 is 120

Example Program:- [Computation of sine series]:-

```
#include<stdio.h>
#include<conio.h>
void main()
 {
 int i,n;
 float x, sum, t;
 printf ("Enter the value for x: ");
 scanf ("%f",&x);
```

```
printf("Enter the value for n:");
scanf("%d",&n);
x=x*3.14159/180;
t=x;
sum=x;
for(i=1;i<=n;i++)
 {
 t=(t*(-1)*x*x)/(2*i*(2*i+1));
  sum=sum+t;
 }
printf("The value of Sin %f=%.4f",x,
                                sum);
getch();
}
```

Output:-

```
Enter the value for x: 45
Enter the value for n: 5
The value of Sin 0.7853980 = 0.7071
```

Scientific Calculator using Buit-in Functions:

Program:-

```
#include <stdio.h>
#include <math.h>
int main()
 {
```

```
int choice, i, a, b;
float x, y, result;
do
 {
printf("\n Select your option:\n");
printf("1. Addition\n 2. Subtraction\n
        3. Multiplication\n 4. Division\n");
printf("5. Square root\n 6. x^y\n
        7. x^2\n 8. x^3\n");
 printf("Enter your choice:\n");
  scanf("%d", &choice);
  if (choice == 0)
     exit(0);
  switch (choice)
     {
   Case 1:
       printf("Enter x:");
        scanf("%f", &x);
        printf("\n Enter y:");
         scanf("%f", &y);
         result = x+y;
         printf("\n Result: %f", result);
         break;
```

```
case 2:
    printf("Enter x:");
    scanf("%f",&x);
    printf("\n Enter y:");
    scanf("%f",&y);
    result = x-y;
    printf("\n Result :%f", result);
    break;
case 3:
    printf("Enter x:");
    scanf("%f",&x);
    printf("Enter y:");
    scanf("%f",&y);
    result = x*y;
    printf("\n Result: %f", result);
    break;
case 4:
    printf("Enter x:");
    scanf("%f",&x);
    printf("Enter y:");
    scanf("%f",&y);
    result = x/y;
    printf("\n Result:%f", result);
    break;
case 5:
    printf("Enter x:");
    scanf("%f",&x);
```

```
        printf("Enter y:");
        scanf("%f", &y);
        result = sqrt(x);
        printf("\n Result = %f", result);
        break;
Case 6:
        printf("Enter x:");
        scanf("%f", &x);
        printf("Enter y:");
        scanf("%f", &y);
        result = pow(x,y);
        printf("\n Result : %f", result);
        break;
Case 7:
        printf("Enter x:");
        scanf("%f", &x);
        result = pow(x, 2);
        printf("Result : %f", result);
        break;
Case 8:
        printf("Enter x:");
        scanf("%f", &x);
        result = pow(x, 3);
        printf("Result : %f", result);
```

```
    }
  }while(choice);
  getch();
}
```

Binary search using recursive functions:-

Program:-

```
#include <stdio.h>
 int recursivebinary(int arr[], int low,
                     int high, int element)
  {
 int middle;
  if (low > high)
    {
    return -1;
   }
  middle = (low+high)/2;
  if (element > arr[middle]).
    {
    recursivebinary(arr, middle+1, high,
                                element);
    }
 else if (element < arr[middle]).
   {
   recursivebinary(arr, low, middle-1,
                                element);

   }
 else
   {
   return middle;
   }
}
```

```
int main()
{
int count, element, limit, arr[50], position;
printf("Enter the no. of elements \t");
scanf("%d", &limit);
printf("\n Enter %d elements in array :\n",
                                    limit);
for(count=0; count < limit; couht++)
{
scanf("%d", &arr[count]);
}
printf("\n Enter Element to search:\t");
Scanf("%d", &element);
position = recursivebinary(arr, 0, limit-1,
                                    element);
if(position == -1)
{
printf("\n Element %d Not Found \n", element);
}
else
{
printf("\n Element %d Found \n", element);
}
return 0;
}
```

Output:-

Enter the no. of elements : 5

Enter 5 elements

1 2 3 4 5

Enter Element to search: 3

Element 3 is found.

2M (X) **Pointers:-**

✓ A pointer is a variable that stores the address of a variable or a function.

Advantages:-

1. pointers Save memory space.
2. Faster Execution
3. Memory is accessed efficiently.

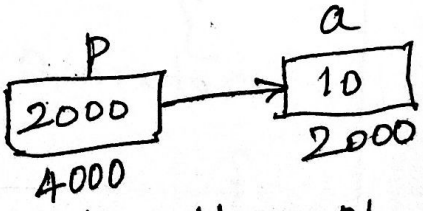
Declaration:-

datatype *pointername

Example:-

```
int *p; // p is an pointer to an int
float *fb; // fp is a pointer to a float

int a=10;
int *p = &a;
```



✓ p is an integer & holds the address of variable a.

Pointer to pointer:-

✓A pointer that holds the address of another point variable is known as a pointer to pointer.

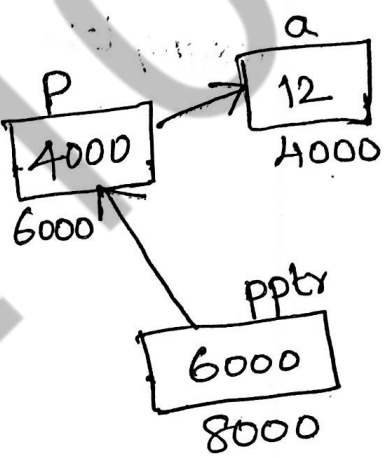
Example:-

```
int **p;
```

✓P is a pointer to a pointer to an integer.

```
int a=12;
int *p=&a;
int **pptr=&p;
```

∴ so **pptr = 12



Operations on pointers:-

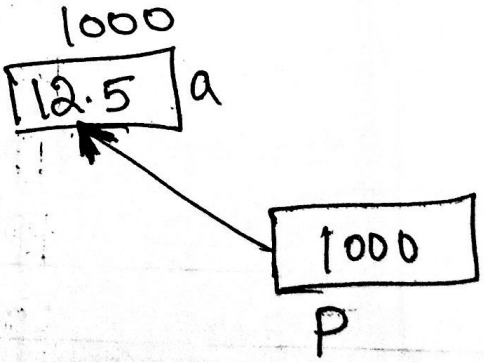
1: Referencing operation:-

✓A pointer variable is made to refer to an object. Reference operator (&) is used for this.

✓Reference operator is also known as address of (&) operator.

example:-

```
float a=12.5;
float *p;
p=&a;
```



2. De-referencing a Pointer:-

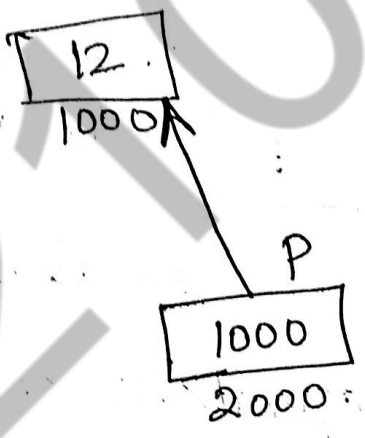
✓ The object referenced by a pointer can be indirectly accessed by de-referencing the pointer.

✓ De-referencing operator (*) is used for this. This operator is also known as indirection operator or value at operator.

example:-

```
int b;
int a=12;
int *P;
p=&a;
b= *P;
```

∴ So b=12



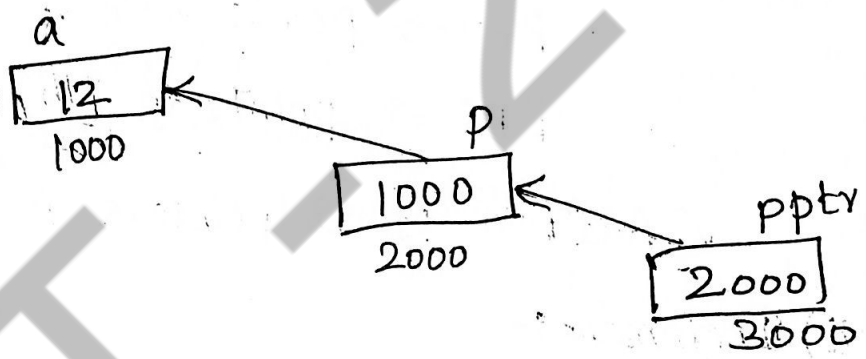
Example Program:-

```
#include<stdio.h>
void main()
{
int a=12;
int *P;
int **pptr;
p=&a;
pptr=&P;
printf("a value=%d", a);
printf("value by de-referencing is
%d\n" *P);
```

```
    printf("Value by de-referencing pptr is %d\n",
                                        **pptr);
    printf("Value of p is %u\n", p);
    printf("value of pptr is %u \n", pptr);
}
```

output:-

a value= 12
value by de-referencing p is 12
value by de-referencing pptr is 12
Value of p is 1000
value of pptr is 2000.



Initialization:-

i) Pointer can be assigned or initialized with the address of an object.

eg:-

```
int a=10;
int *p=&a;
```

2) A pointer to one type cannot be initialized with the address of other type object.

eg:

```
int a=10;
float *p;
p=&a; //not possible
```

(Because p is a float pointer, so it can't point int data)

Pointers Arithmatic :-

✓ Arithmatic operations on pointer variables are also possible.

eg:- Addition, Increment, Subtraction, decrement.

i) Addition:-

✓ An addition of int type can be added to an expression of pointer type. The result is pointer type (or). A pointer and an int can be added.

eg:- if p is a pointer to an object then

P+1 ⇒ points to next object.

P+i ⇒ points to i^{th} object after P.

✓ Addition of 2 pointers is not allowed.

ii) Increment:-

✓ Increment operator can be applied to an operand of pointer type.

iii) Decrement:-

✓ Decrement operator can be applied to an operand of pointer type.

iv) Subtraction:-

✓ A pointer and an int can be Subtracted

✓ 2 pointers can also be Substracted.

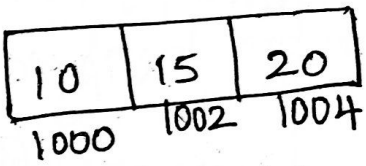
Pointers and Arrays:-

✓ In c-Language pointers and arrays are so closely related.

i) In array name itself is an address or pointer. It points to the address of first element (0th element) of an array.

Example:-

```
#include <stdio.h>
void main()
{
int a[3] = {10,15,20};
printf("First element of array is at %u\n",a);
printf("2nd element of array is at %u\n",a+1);
printf("3rd element of array is at %u\n",a+2);
}
```



Output:-

First element of array is at 1000
2nd element of array is at 1002
3rd element of array is at 1004.

ii) Any operation that involves array subscripting is done by using pointers in c-language

eg: $E_1[E_2] \Rightarrow *(E1+E2)$

Example:-

```
#Include <stdio.h>
 void main()
  {
  int a[3] = {10,15,20};
  printf("Elements are %d %d %d \n", a[0], a[1],
                                        a[2]);
 printf("Elements are %d %d %d \n", *(a+0), *
                                (a+1), *(a+2));
  }
```

Output:-

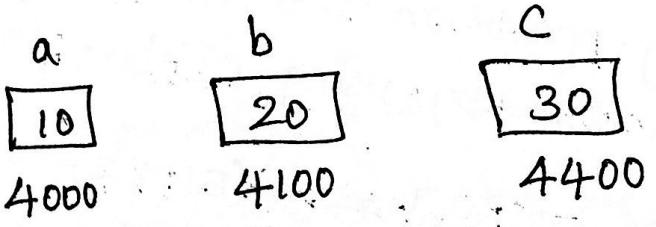
Elements are 10 15 20

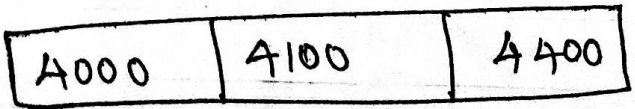
Elements are 10 15 20

Array Pointers:-

✓Array pointers is a collection of addresses. Pointers in an array must be the same type.

```
int a=10, b=20, c=30
 int *b[3] = {&a, &b, &c};
```





Parameter Passing:-

Function with input and outputs:-

✓ More than one value can be indirectly returned to the calling function by making use of pointers.

eg: Program

Call by reference program

Pass by value & Pass by reference:-

✓ Argument passing methods in 'C' are

1. Pass by value
2. Pass by reference

i) Pass by value:-

✓ In this method the values of actual arguments are copied to formal arguments.

✓ Any change made in the formal arguments does not affect the actual arguments.

✓ Once control return backs to the calling function the formal parameters are destroyed.

Eg: Program:- [swapping]:-

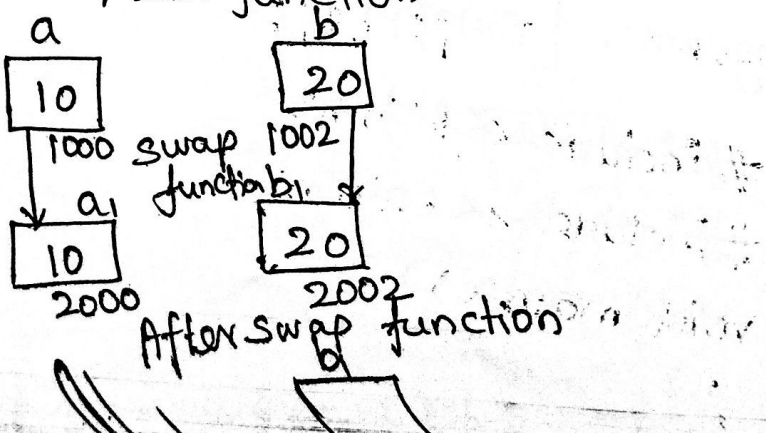
```
#include <stdio.h>
#include <conio.h>
void main()
{
int a,b;
```

```
void swap(int, int);
 a=10;
 b=20;
 printf("\n Before swapping a=%d and
                        b=%d", a,b);
 swap(a,b);
 printf("\n After swapping a=%d and
                        b=%d", a,b);
  getch();
  }
void swap (int a1, int b1)
  {
    int temp;
    temp = a1;
    a1 = b1;
    b1 = temp;
  }
```

Output:-

Before swapping a=10 and b=20

After swapping a=10 and b=20



ii) Pass by reference:

✓ In this method, the address of the actual arguments are passed to formal argument.

✓ Thus formal arguments points to the actual arguments.

✓ so changes made in the arguments are permanent.

Example Program:- [swapping]

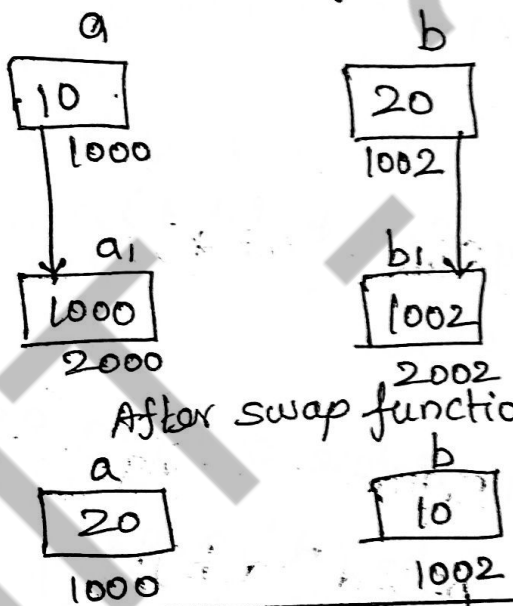
```
#include <stdio.h>
#include <conio.h>
void main()
 {
 int a,b;
 void swap (int *, int *);
 a=10;
 b=20;
 printf("\n Before swapping a=%d and
                b=%d", a,b);
 swap (&a,&b);
 printf("\n After swapping a=%d and
                b=%d", a,b);
 getch();
}
void swap( int *a1, int *b1)
 {
```

```
    int t;
    t = *a1;
    *a1 = *b1;
    *b1 = t;
}
```

output:-

Before swapping a=10 and b=20

After swapping a=20 and b=10



Pass by Valve	Pass by Reference
✓ values of the actual arguments are passed to the Final arguments.	✓ Address of the actual arguments are passed to the final arguments.
✓ Different Memory Locations are occupied.	✓ Same memory Locations are occupied.
✓ changes does not affect the actual value.	✓ changes to the value affect the orginal data.