20CS1101 –PROGRAMMING FOR PROBLEM SOLVING

(Common to all branches)

UNIT – V

POINTERS: Fundamentals, Declaration and initialization of Pointers, Arithmetic Operations, Pointers and Arrays.

FUNCTIONS: Definition, Function Prototypes, Types of functions, Call by Value and Call by Reference, Recursion.

→POINTERS

5.1 Definition: A pointer is memory variable that stores a memory address. Pointer can have any name that is legal for other variable and it is declared in the same fashion like other variable but it is always denoted by "* "operator.

5.2 Advantages of pointers

1. Pointers save the memory space.
2. Pointers are used to increase the speed of execution.
3. Pointers are used to reduce the length and complexity of a program.
4. Pointers are useful for representing two dimensional and multi dimensional arrays.
5. The memory is accessed efficiently with the pointers.

5.3 Operators used with pointers

a) Indirection operator or dereference operator (*): It is used in two distinct ways with pointers

1. To declare a pointer variable in the declaration section, before the identifier * is used. Then that variable is converted into pointer variable.
2. To access the value stored in a particular memory location * operator is used. That is why it is also called dereference operator.

b) Address operator (&): It is used to find the address of a variable. If we put & before any variable name, we get the address of that memory location.

→5.4 Pointer declaration:
Like all variables, a pointer variable should also be declared.

Syntax: Data type * variable name;

Example: int *ptr;

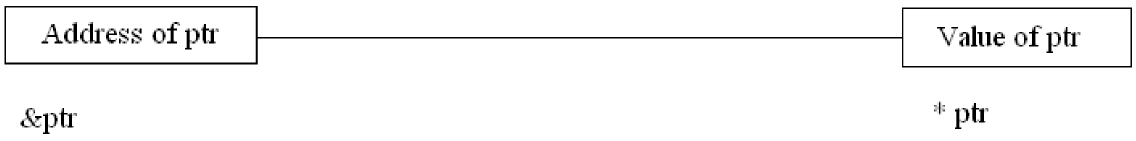


Fig: Representation of a pointer variable

Example: Write a c program for accessing a variable through a pointer

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=5,b=10,*ptr;
ptr=&a;
printf("before values a=%d b=%d",a,b);
b=*ptr;
printf("after values a=%d b=%d",a,b);
printf("value of ptr%d",ptr);
printf("address of a is %d", &a);
printf("value of *ptr is%d",*ptr);
getch();
}
```

```
Output: Before values a=5 b=10
        After values a=5 b=5
        Value of ptr is 4040
        Address of a is 4040
        Value of *ptr is 5
```

→5.5 Initialization of pointers:

A pointer is similar to any other variable except that is holds only memory address, and therefore, it needs to be initialized with a valid address before it can be used.

Pointers should be initialized either when they are declared or in an assignment statement. A pointer may be initialized to NULL or with the address of some other variable, which is already defined.

Example: ptr= & i is called as pointer initialization.

→5.6 Pointer arithmetic:

Arithmetic operations on pointer variables are also possible. ++, -- , prefix and postfix operations can be performed with the help of the pointers. The effect of these operations are shown in the below given table.

Data type	Initial address	Operation		Address after operations		Required bytes
int i=2	4046	++	--	4048	4044	2
char c='x'	4053	++	--	4054	4052	1
float f=2.2	4058	++	--	4062	4054	4
long l=2	4060	++	--	4064	4056	4

Note: ++ to be incremented, but not by 1. - - to be decremented, not by 1.

Example: write a c program to display the address of the variable

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n;
printf("Enter the n value");
scanf("%d",&n);
printf("value of num=%d", num);
printf("address of num=%d",&num);
getch();
}
```

Output: n=20

```
value of num=20
Address of num=4066
```

Example: Write a c program to find the sum of two values using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, sum, *ptr1,*ptr2;
ptr1=&a;
ptr2=&b;
printf("enter the two numbers");
scanf("%d %d",&a,&b);
sum=*ptr1 +*ptr2;
printf("sum=%d", sum);
getch();
}
```

Example: Write a c program to find the biggest of two values using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, *big *ptr1,*ptr2;
ptr1=&a;
ptr2=&b;
printf("enter the two numbers");
scanf("%d %d",&a,&b);
if(*ptr1>*ptr2)
big=ptr1;
else
big=ptr2;
printf("biggest number is=%d", *big);
getch();
}
```

→5.7 Pointers and arrays:

In C, when an array is declared, the array name denotes the address of the 0th element in that array. The address of the 0th element is also called base address or starting address. Therefore, if we refer the name of the array we get the starting address of the array.

So array name itself is an address or pointer. The elements of the array together with their address can be displayed by using array name itself since array elements are stored in contiguous memory locations.

For example, an array is initialized as bellow

int a[4]= {3,7,9,4};

Now, we can access the value in the second location by two ways.

By array notation: a[1]

By pointer notation: *(a+1)

Similarly, we can get address of the second location by two ways.

By array notation: &a[1]

By pointer notation: (a+1)

Example: Write a C program to find the biggest number among given list of elements by using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,a[20], *ptr;
clrscr();
printf("enter n value");
scanf("%d",&n);
for(i=0;i<n;i++)
scanf("%d",&a[i]);
*ptr=a[0];
for(i=0;i<n;i++)
if(a[i]>*ptr)
*ptr=a[i];
printf("maximum number=%d",*ptr);
getch();
}
```

Example: Write a C program to display element name, value at that location and address of that location.

```
#include<stdio.h>
void main()
{
int  x[]={1,2,3,4};
int i;
printf("element.no,  elem    address\n");
for(i=0;i<4;i++)
 {
   printf("x[%d] %d %d", i, *(a+i), (a+i));
 }
getch();
}
```

```
Output:
Element. No         element             Address
X[0]                   1                  4056
X[1]                   2                  4058
X[2]                   3                  4060
X[3]                   4                  4062
```

→5.8 Dynamic memory allocation: Memory allocations mean allocating sufficient memory space to all the variables in the program. Memory space can be allocated in two ways. They are

1. Static memory allocation
2. Dynamic memory allocation

1. Static memory allocation: To allocating the memory at the time of compilation, then it is said to be a static memory allocation.

2. Dynamic memory allocation: To allocating the memory at the time of execution, then it is said to be dynamic memory allocation.

In C language there are 4 ways to allocating the memory

1.The malloc( ) function   2.the calloc( ) function   3.The realloc( ) function
4.The free( ) function

1. The malloc ( ) function: This is used to allocate a contiguous block of memory in bytes.

Syntax:

Pointer variable=(cast-type *) malloc(size);

Example: x=(int *)malloc(20);

x=(int   *)malloc(10 *sizeof(int)); on execution of malloc, 10 times the size of an int (ie., 10*2=20 bytes) is allocated and the starting address of the first byte is assigned to pointer x of type int.

Where pointer-variable is a valid C variable already defined. Cast –type is the type of the pointer returned by malloc ( ) such as int, char…etc. size is the required size of memory in byte.

Note: If the memory allocation is success it returns the starting address else it returns the NULL.

Example: Write a C program to perform sum of array elements by using malloc( ) function

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *a,i,n,sum=0;
printf("enter the size of array");
scanf("%d",&n);
a=(int *)malloc(n*size(int));
if(a!=NULL)
{
  printf("enter the elements");
  for(i=0;i<n;i++)
   {
     scanf("%d",&a[i]);
     sum=sum+a[i];
   }
printf("sum of given elements are %d",sum);
}
else
printf("memory can not be allocated");
getch();
}
```

```
Output: Enter the size of the array    5
             Enter the elements    1    2    3    4    5
            Sum of given elements are    15
```

2. The calloc ( ) function: This function is used to allocate multiple blocks of contiguous memory in bytes. All the blocks are of same size.

Syntax: pointer-variable =(cast-type *) calloc(n, size);

Example: x=(int *) calloc(5,10);

Where pointer-variable is a valid C variable already defined. Cast-type is the type of the pointer returned by calloc ( ) such as int, char ….etc. 'n' is the number of blocks and size is the required size of memory in bytes.

Note: If the memory allocation is success it returns the starting address else it returns NULL.

3. The realloc ( ) function: The realloc () function is used to increase or decrease the size of memory previously allocated by using malloc/calloc () function.

Syntax: new-pointer=realloc (old-pointer, new-size);

Where new-pointer is valid C variable previously defined. Old-pointer is the pointer variable used in malloc( ) or calloc( ) function. New-size is the size of the new memory needed.

Example: y= (int *)malloc(50);

X=realloc(y, 30);

The first statement allocated memory space of size 50 bytes and returns the starting address of the memory through the pointer x. the second statement reallocates (decreases) the already allocated space to 30 bytes.

4. The free ( ) function: The free ( ) function is used to free (release or de allocate) the block of unused or already used memory.

Syntax: free (pointer-variable);

Where pointer-variable is a pointer to the memory block which has been already created by malloc ( ) or calloc ( ) function.

Example: x= (int *) malloc (50);

free (x);

The first statement allocated memory space of 50 bytes and returns the starting address of the allocated memory through a pointer variable x. The second statement frees the allocated memory.

→5.9FUNCTIONS: Definition, Function Prototypes, Types of functions, Call by Value and Call by Reference, Recursion.

Introduction: The C language supports two types of functions

1. Library functions or predefined functions
2. User defined functions

1. Library functions or predefined functions: The library functions are pre-defined set of functions. Their task is limited. A user cannot understand the internal working of these functions. The user can only use the functions but cannot change or modify them.

Example: sqrt(81) gives result 9. Here the user need not worry about its source code, but the result should be provided by the function.

2. User defined functions: The user defined functions are totally different. The functions defined by the user according to his/her requirements are called user defined functions. The user can modify the function according to the requirement. The user certainly under stands the internal working of the function.

Definition: A function is a self-contained block is called a function.

[Or]

A subprogram of one or more statements that performs a special task when called.

Use of functions: The use of functions offers flexibility in the design development and implementation of the program to solve complex problems.

Advantages of functions:

1. Modular programming.
2. Reduction in the amount of work and development time.
3. Program and function debugging is easier.
4. Division of work is simplified due to the use of divide and conquers principle.
5. Reduction in the size of the program due to code reusability.
6. Logical clarity of the programming will be clear.
7. A library with user defined functions can be created.

→5.10 Variables: There are two kinds of variables.

1. Local variables   2.Global variables

Local variables	Global variables
1. It is declaration inside the function. 2.It is declared by auto int a=10 or int a = 10 (auto is default scope) 3. If it is not initialized garbage value is stored. EX: int a; a is garbage value. 4. It is created when the function starts execution and lost when the function terminates 5. It is visible in only one function. 6. It can be accessed in only one function that is the function where it is declared. 7. Data sharing is not possible that is data of local variable can be accessed by only one function 8. Parameters passing is required for local variable that is local variable of one variable 9. If value of local variable is modified in one function changes are not visible in another functions	1.It is declaration outside the function 2. It is declared by int a=10 3. If it is not initialized zero is stored EX: int a; a = 0. 4. It is created before program execution starts and lost when program terminates 5. It is visible throughout the program. 6. It can be accessed in more than one function. 7. Data sharing is possible that is multiple functions can access the same global variable. 8. Parameters passing is not required for global variable since global variable is visible throughout the program. 9. If value of global variable is modified in one function changes are visible in rest of the program.

Example: Write a c program on local variables

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int b=10, c=5;
  clrscr();
printf("\n In main function b=%d c=%d", b,c);
fun( );
getch();
}
```

```
fun( )
{
  int b=20, c=10;
  printf("\n In function b=%d c=%d", b,c);
}
```

Output: In main function b=10 c=5

```
             In function b=20 c=10
```

Example: Write a c program on global variables

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

int b=10, c=5;
void main()
{
  clrscr();
printf("\n In main function b=%d c=%d", b,c);
fun( );
b++;
c--;
Printf("\n again in main() b=%d c=%d", b,c);
getch();
}

fun( )
{
  b++;
 c--;
printf("\n In function b=%d c=%d", b,c);
}
```

Output: In main() b=10 c=5

```
              In fun( ) b=11 c=4
             Again in main( ) b=12  c=3
```

→5.11 Function components: Every function has the following elements associated with it

1. Function declaration and function prototype
2. Function parameters
3. Function definition
4. Return statements
5. Function call

1. Function declaration and function prototype: Function is declared as per format given bellow

```
function-name(arguments/parameter list)
{
 local variable declaration;
 statement1;
 statement2;
 return(value);
}

void main()
{
 ........
abc(x,y,z);              Function call or calling or function declaration
.........
.........          Actual arguments
}

abc(l,k,j)               Function definition or called function
{
..........         Formal arguments
..........
return();            return value

}
```

2. Function parameters: The parameters specified in the function call (calling) are known as actual parameters and those specified in the function definition (called) are known as formal parameters.

```
void main()
{
 int x=1, y=2 ,z;
 z=add(x,y);            Function call(calling)
printf("z=%d", z);
getch();
}

add(a,b)           Function definition (called)
{
  return(a+b);
}
```

3. Function definition: The function itself is referred to as function definition. The first line of the function definition is known as function declaratory and is followed by the function body.

```
add(a,b)           Function definition (called)
{
  return(a+b);
}
```

4. Return statements: Functions can be grouped into two categories

1. Functions that do not have a return value(void)

2.Functions that have a return value.

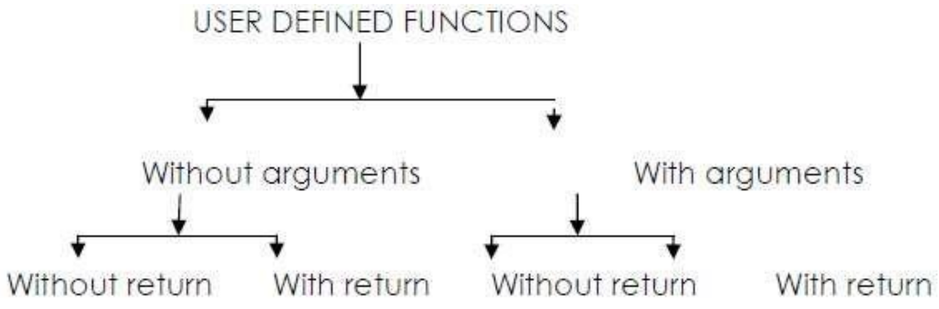
```
void main()                The main() function should not return anything.
{
.............
............
getch();
}


int  main()            The main() function should return a integer type of value.
{
.............
............
return(0);
getch();
}
```

5. Function call: A function call is specified by the function name followed by the arguments enclosed in parenthesis and terminated by a semicolon. The return type is not mentioned in the function call.

Example: z=add(x, y);   Function call(calling)

→5.12 Types of functions: Depending upon the argument present, return value sends the result back to the calling function based on this the functions are divided into four types.



Example:

void sum( )	function without arguments and without return value
int sum( )	function without arguments and with return value
void sum(int ,int)	function with arguments and without return value
int sum(int,int)	function with arguments and with return value

Example1: Write a C program on without arguments and return values

```
#include<stdio.h>
#include<conio.h>
void main()
{
void message();
}
void message()
{
printf("Have a nice day");
}
```

Output: Have a nice day

Explanation: - This program contains a user defined function named message (). It requires no arguments and returns nothing. It displays only a message when called.

Example2: Write a C program on with arguments without return values

```
#include<stdio.h>
#include<conio.h>
void main()
{
int j=0;
void sqr();
for(j=1;j<=5;j++)
sqr(j);
}
void sqr(int k)
{
printf("\n %d",k*k);
}
```

Output: 1 4 9 16 25

Explanation: Here the main () function passes one argument per call to the function sqr(). The function sqr() collects this arguments and prints its square. The function sqr() is void.

Example 3: Write a C program on with arguments and with return values

```
#include<stdio.h>
#include<conio.h>
void main()
{
int date(int, int,int);
int d,m,y,t;
clrscr();
printf("enter date dd/mm/yy");
scanf("%d%d%d",&d,&m,&y);
t=date(d,m,y);
printf("\n Tomorrow=%d/%d/%d",d,m,y);
return 0;
}
date(int x, int y, int z)
{
printf("\n Today=%d/%d/%d",x,y,z);
return(++x);
}
```

output: enter date dd/mm/yy 12 12 12

```
Today = 12/12/12
Tomorrow= 13/12/12
```

Explanation: In the above program three values date, month, and year are passed to functions date (). The function displays the date. The function date () returns the next date. The next date is printed in function main (). Here, function date () receives arguments and returns the values.

Example 4: Write a c program on without arguments and without returns values

```
#include<stdio.h>
#include<conio.h>
main()
{
int sum(),a,s;
clrscr();
s=sum();
printf("sum=%d", s);
return 0;
}

sum()
{
int x,y,z;
printf("\n Enter the three values");
scanf("%d %d %d", &x,&y,&z);
return(x+y+z);
}
```

Output: enter the three values   3 5 4

```
             Sum=12
```

→5.13 Parameter passing mechanism: There are two parameter passing mechanisms

1. Call by value. Or passing by value
2. Call by reference or passing by address or call by address.

1. Call by value:

Passing arguments by value means, the contents of the arguments in the calling function are not changed, even if they are changed in the called function. This is because the content of the variable is copied to the formal parameter of the function definition, thus preventing the contents of the argument in the calling function.

2. Call by reference:

Call by reference means sending the address of variables as arguments to the Function. When addresses are sent , the changes occurred in the called function can also effect in the calling function.

Example 1: Write a C program to perform parameter passing mechanism by using call by value.

```
#include<stdio.h>
#include<conio.h>
void add(int);
void main()
{
  int x=5;
printf("Before x=%d",x);
add(x);
printf("After x=%d",x);
getch();
}
void add(int y)
{
 y=y+5;
printf("In called function x=%d",y);
}
```

Output:

```
    Before x=5.
    In called function x=10.
    After x=5.
```

Advantages: 1. Expression can be passed as arguments.

2. Un wanted changes to the variables in the calling function can be avoided.

Disadvantages: Information cannot be passed back from the calling function to the called function through arguments.

Example 2: Write a C program to perform parameter passing mechanism by using call by reference.

```
#include<stdio.h>
#include<conio.h>
void add(int *);
void main()
{
  int x=5;
printf("Before x=%d",x);
add(&x);
printf("After x=%d",x);
getch();
}
void add(int  *y)
{
 *y=*y+5;
printf("In called function x=%d", *y);
}
```

Output:

```
        Before x=5.
        In called function x=10.
        After x=10.
```

→5.14 Recursion: A function which calls itself until a certain condition is reached is called recursive function. The recursion can be

1. Direct recursion   2.Indirect recursion.

1. Directive recursion: The direct recursion function calls itself till the condition is true.

2. Indirect recursion: In indirect recursion a function calls another function then the called function calls the calling function.

Example1: Write a C program to find the factorial of a given number using recursion

```
#include<stdio.h>
#include<conio.h>
long int fact(int);
void main()
{
 long int f;
 int n;
 clrscr();
printf("Enter the number");
scanf("%d",&n);
f=fact(n);
printf("The factorial of a given number is %ld", f);
getch();
}

long int fact(int x)
{
  long int m;
  if(x==1|| x==0)
  return(1);
  else
   {
     m=x*fact(x-1);
     return(m);
   }
}
```

Output: Enter the number 5

```
Enter the number  5
             The factorial of a given number is 120
```

Example 2: Write a C program to find the Fibonacci series using recursion

```
#include<stdio.h>
#include<conio.h>
int fib(int);
void main()
{
 int i, n;
 clrscr();
printf("Enter the number");
scanf("%d",&n);
printf("The fibnocci series is");
for(i=1;i<=n;i++)
printf("%d", fib(i));
getch();
}
```

```
int fib(int x)
{
  if(x==1|| x==2)
  return(x-1);
  else
   return(fib(x-1)+fib(x-2));

}
```

Output: Enter the number 5

```
            The Fibonacci series is 0 1 1 2 3
```

Difference between Iterative statements and Recursion

Iterative statements	Recursion
1. Function calls some other functions.	1. Function calls some itself.
2. while loop, do while loop or for loop is necessary in iterative function.	2. if statement is necessary in recursive Function
3. Fast in execution since one function call is enough to get the result.	3. Slow in execution since several function calls are involved to get the result, as number of function increases execution will slow down.
4. Function call leads to value Eg: fact(4) = 24.	4. Function call leads to another function call Eg: fact(4) = 4 xfact(3)
5. We require mathematical steps or procedure to write an iterative function.	5. We require a formula to write a recursive function.
6. Stacks are not used during execution	6. Stacks are used during execution of recursive functions.