PROGRAMMING FOR PROBLEM SOLVING -- PPS

SUBJECT CODE- BTPS-101-18

PPS

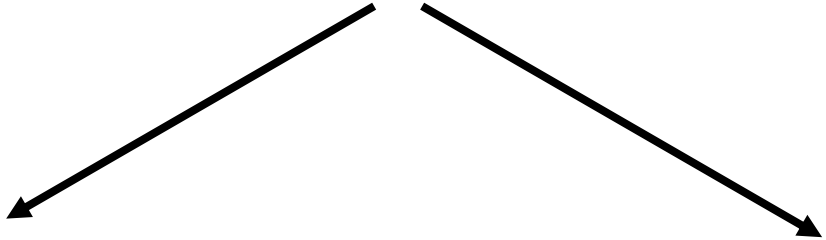
UNIT-5

FUNCTIONS

Session Objectives

- Explain User Defined Functions
- Explain Type of Functions
- Discuss category of Functions
- Declaration & Prototypes
- Storage class in “C”
- Calling the Function

FUNCTIONS



Built-In Functions

- scanf()
- printf()
- getc()
- putc()

User-Defined Functions

Series of instructions that are to be executed

USER DEFINED FUNCTION :

SYNTAX :

```
returndatatype functionname(arguments)
{
Body of the function
statements;
return;
}
```

call the function from main() :

syntax :

```
functionname (arguments );
```

```
#include <stdio.h>
main()
{
        .
        .
        address()
        .
        .
}

address()
{
        .
        .
        .
}
```

For Example

```
#include<stdio.h>
void hai()    //definition
{
printf(" Welcome to functions\n");
printf("Good Morning\n");
}
void main()
{
clrscr();
printf("Main, Welcome to functions\n");
hai();       //calling
printf("Bye");
getch();
}
```

Category of Functions

(Based on Return values and passing Arguments)

- **NO ARGUMENT NO RETURN VALUES**
- **ARGUMENT BUT NO RETURN VALUES**
- **NO ARGUMENT WITH RETURN VALUES**
- **WITH ARGUMENT WITH RETURN VALUES**

```
/*  To perform Addition of two numbers */
/* NO ARGUMENT NO RETURN VALUES */
#include<stdio.h>
void  add();
void main()
{
add();
printf(“main ends here");
add();
}
```

```
void add()
{
int a,b,c;
printf("Enter two numbers\n");
scanf("%d%d",&a,&b);
c=a+b;
printf("The sum is %d\n",c);
}
```

```
/*  To perform Addition of two numbers */
/* WITH ARGUMENT BUT NO RETURN VALUES*/
#include<stdio.h>
void  add(int,int);
void main()
{
int x,y;
printf("Enter two number");
scanf("\t\t%d %d",&x,&y);
add(x,y);    /* Actual Arguments */
}
void add(int a,int b)    /* Formal Arguments */
{
int c=a+b;
printf("\t\tThe C Value is %d",c);
}
```

return Statement

- The return statement is used to return from a function.
- It causes execution to return to the point at which the call to the function was made.
- The return statement can have a value with it, which it returns to the program.

```
/*  To perform Addition of two numbers
 Without Argument  and With Return values */
#include<stdio.h>
#include<conio.h>
int add();     //declaration
void  main()
{
int c;
c=add();      /* Return Variable - c */
printf("The sum of two numbers is %d",c);
}
int add()
{
int a,b,c;
printf("Enter two Numbers=");
scanf("%d %d",&a,&b);
c=a+b;
return(c);
}
```

```
/*  To perform Addition of two numbers
 With Argument  and With Return values */
#include<stdio.h>
int add(int,int);     //Function prototype declaration
void  main()
{
int c;
printf("Enter two Numbers=");
scanf("%d %d",&a,&b);
c=add(a,b);  /* Actual Arguments */
printf("The sum of two numbers is %d",c);
}
int add(int x,int y)  /* Formal arguments */
{
int c;
c=x+y;
return(c);
}
```

STORAGE CLASS

Every ‘C’ variable has a characteristic called its Storage Class.

All variables have datatype and storage classes

Based On

- ✓ **Keyword**
- ✓ **Where it is Declared**
- ✓ **Storage Area**
- ✓ **Default Initial value**
- ✓ **Lifetime of a variable**

Four different storage classes in "c"

1. Local or Auto or Internal variable
2. External or Global variable
3. Static variable
4. Register Variable

Local Variable

Auto variable are always declared within a function and they are local to the function in which they are declared. Hence they are also named as local variables

Keyword : auto

Declaration : Inside the function

Storage Area : Stack

Initial Value : Garbage value (At the time of compilation compiler assigns any value)

Lifetime : Upto that function only

Example :

auto int x; (or) int x;

```
#include<stdio.h>
void function1();
void function2();
void main()
{
int m=1000;
function2();
printf("%d\n",m);
}

void function1()
{
int m=10;
printf("%d\n",m);
}

void function2()
{
int m=100;
function1();
printf("%d\n",m);
}
```

Global Variable

A variable which can be access with in a function and outside the main function. These variables are also named as Global variables or External variables

Keyword : extern

Declaration : Outside of the main() function

Storage Area : CPU–memory

Initial Value : zero

Lifetime : Upto the entire program

Example :

```
int x;
main()
{
}
```

(or)

```
main()
{  extern int x;
}
```

```
#include<stdio.h>
int k;
void function1();
void function2();
void function3();
void main()
{
k=20;
function1();
function2();
function3();
}
void function1()  {
k=k+10;
printf("%d\n",k);  }
void function2()
{
k=k+1000;
printf("%d\n",k);
}
void function3()
{
k=k+10;
printf("%d\n",k);  }
```

Static Variable

This variable static is constant and the value is continued in all the steps.

Keyword : static

Declaration : Inside the function

Storage Area : CPU – memory

Initial Value : Zero

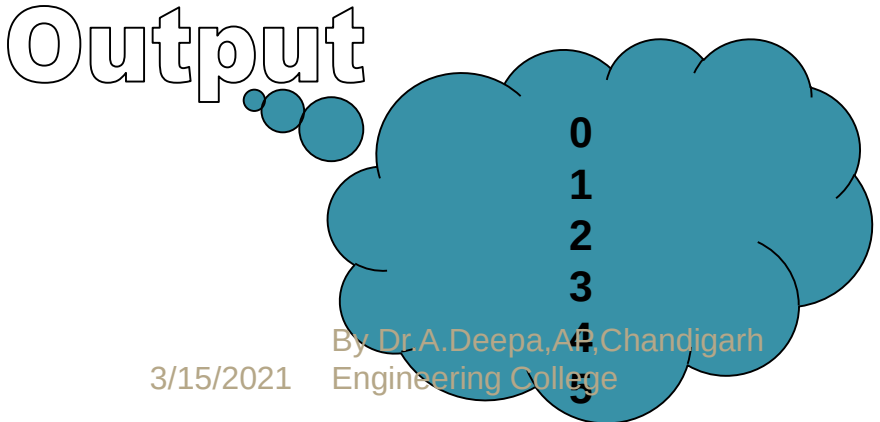
Lifetime : The value of the variable persists between different function calls.

Example :

```
static int x;
```

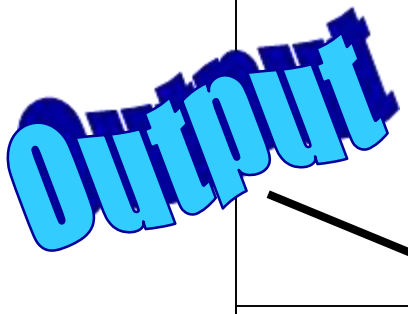
```
/* To print the value of x */
#include<stdio.h>
void stat();
void main()
{
int i;
for(i=1;i<=5;i++)
stat();       //calling
}
void stat()       //definition
{
static int x=0;
printf("x=%d\n",x);
x=x+1;
}
```

Output



```
/* Example 2  */
#include<stdio.h>
#include<conio.h>
void incre();  /* Function prototype
declaration */
void main()
{
clrscr();
incre();
incre();
incre();
getch();
}

void incre()
{
static char x=65;
printf("\n The character stored in x
is %c",x++);
}
```




```
The character Stored in x is A

The character Stored in x is B

The character Stored in x is C
```

Register Variable

These variables are stored in CPU registers and hence they can be accessed faster than the one which is stored in memory.

Keyword : register

Declaration : Inside the function

Storage Area : CPU - Register

Initial Value : Garbage value(At the time of compilation compiler assigns any value)

Lifetime : Upto that function only

Example : register int x;

Note :

register double x; register float y;

Registers are usually a 16bit therefore it cannot hold a float or double data type value which require 52 & 64 bytes respectively for storing a value. But the compiler would treat as automatic variables

```
#include<stdio.h>
#include<conio.h>
void main()
{
register int x;
clrscr();
printf("\n The value is %d",x);
getch();
}
```

Output

-899
(Garbage Value)

Passing Parameters

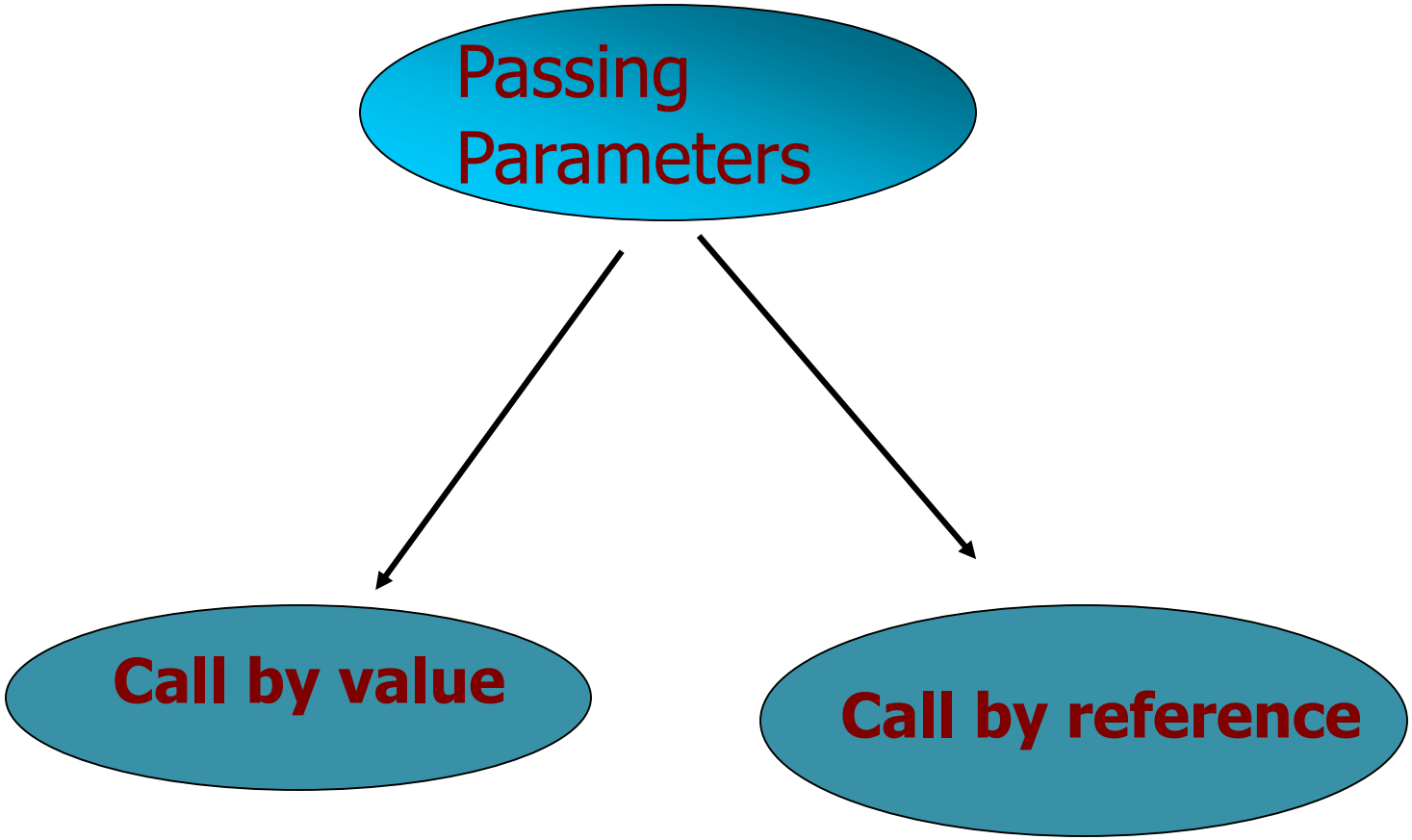
passing values to a Function

Functions communicate with each other by passing arguments.

It can be passed in Two Ways

1.Call By Value

2. Call by Reference


Passing Parameters
Call by value
Call by reference

CALL BY VALUE

- ✓ **The values are passed through temporary variables. Any manipulation to be done only on these temporary variables.**
- ✓ **The called function does not access the actual memory location of the original variable and therefore cannot change its value.**

In C, Call by Value is Default

```
/* CALL BY VALUE EXAMPLE*/
#include<stdio.h>
void  add(int,int);
void main()
{
int x,y;
printf("Enter two number");
scanf("\t\t%d %d",&x,&y);
add(x,y);
}
void add(int a,int b)
{
int c=a+b;
printf("\t\tThe C Value is %d",c);
}
```

OUTPUT

```
Enter two number =   60
20
The C Value is 80
```

Call By Reference

- ✓ **The function is allowed access the actual memory location(Address) of the argument (original variable) and therefore can change the value of the arguments of the calling routine have to be changed.**

In C, Pointers are used to made this.

```
/*CALL BY REFERENCE as well as to swap 2 numbers
 using pointers */
#include<stdio.h>
void swap(int *,int *);
void main()
{
int a,b;
printf("\nEnter the numbers to swap");
scanf("%d %d",&a,&b);
printf("The values before swapping :");
printf("\n%d %d",a,b);
swap(&a,&b);
}
```

Continue….

```
void swap(int *e,int *f)
{
int *temp;
*temp=*e;
*e=*f;
*f=*temp;
printf("\n The swapped values are %d %d",*e,*f);
}
```

OUTPUT

```
Enter the  numbers to swap =   5
6
The values before swapping : 5  6
The swapped values are  :   6    5
```

NOTE :

& → Address of

*** → Content of**

HINTS

- Functions are easier to write and understand
- The arguments are seperated by commas
- The body of the function may consist of one or many statements
- It cannot be defined within another function
- Function prototype is a function declaration that specifies the data types of the arguments
- Calling one function from within another is said to be nesting of Function calls
- main() returns an integer which is generally the operating system

Session Summary

- A function is a self contained program segment (block of statements) that performs some specific well defined task.
- Three steps in using a function are defining a function, prviding a prototype and calling the function.
- Return statement is used to return the information from the function to the calling portion of the program
- Scope of a variable is defined as the region over which the variable is visible or valid.

EXERCISES

1. **Write a program to sort the numbers in ascending order using functions?**
2. **Write a program to calculate X^n using functions?**
3. **Write a program to check whether the year is leap year or not using functions?**
4. **Write a program to find the square of first N Numbers and to calculate its sum?**
5. **Write a program to swap two numbers using functions?**