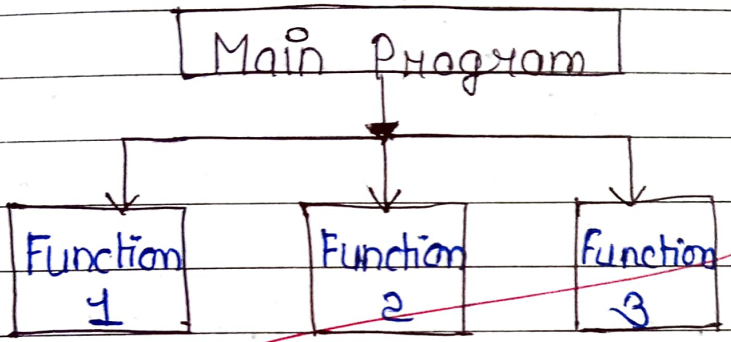
Chapter - 6
User defined functions

1. Modularity :- Longer programs are divided into multiple functions, to solve above stated problems. This concept is called as modularity. The main benefit of modularity is parallel development. [The concept can be understood by a practical example]



1. Function :- A function is a set of statements that performs a particular task, whenever called. The function is an independent program that can be called by other functions. Any large C programs can be divided into two or more smaller function. Creating a function is similar to hiring a person and giving him a particular work to perform. The function performs a particular action whenever called. A function is a set of statements written in particular language that performs a particular task, whenever called.

3.4 Types of functions:-

Functions can be broadly categorized into two categories

1.4 Inbuilt or System defined or library functions
2.4 User defined functions.

1.4 Inbuilt functions:- These are the functions whose definition is already defined and stored in respective header files of C library. These functions can be easily incorporated into programs by including the respective header files. Examples of this type of functions are gets() puts(), clrscr(), getch(), printf(), scanf() etc

2.4 User defined functions:- The functions which are created by the user to perform the desired task are called User defined functions.

7 Advantages of User defined functions:-

1.4 No Redundancy/Repitation:- Complex programs generally contain same piece of code at more than one location in a program. When we create a function of repeated code, the redundancy of code is reduced, size of the program file is reduced and it becomes easy to design and debug.

2. Universal use :- Some functionality may be needed in more than one programs. There are some common tasks in programs of similar type. When we create a function and make it a part of library, the function is universally available to every program.

3. Modularity :- is a process of dividing a big problems into multiple smaller problems. Modularity is the main benefit provided by functions.

4. Teamwork :- When we divide a problem into a number of functions, the coding can be done in parallel by many people. This reduces the development time and increases the spirit of teamwork in developers.

5. Declaring a function :- Declaring a User defined function in C language is very easy. Following is the syntax of declaring a function in C language.

```
return_type function_name(parameter list) {
    body of the function
}
```

Following is the list of elements of a function.

1. Return type :- A function may return a value. The return-type is the data

type of the value the function returns. Some function performs the desired operations without returning a value. In this case the return type is the keyword void.

2. Function name :- This is the actual name of the function. The function name and the parameter list together constitute the function signature.

3. Parameters :- A Parameter list contains variables that carry information from the main program to function

4. Function body :- The function body contains a collection of statements that define what the function does.

Example program :-

```
int max(int num1, int num2) {
int result;
if (num1 > num2)
  result = num1;
else
  result = num2;
return result;
}
```

6.] Function Declarations :- A function declaration tells the Compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has following parts

return_type function_name(parameter List);

Example for above statement :-

int max(int num1, int num2);

7.] Function Call :- A function call is a special statement, which is written in calling function at a point where the functionality of function is needed function body consists of the actual statements that will be executed by the function. The function from which the call a function is made is called calling function and the function that has been called is called "Called function".

8.] Types of user defined functions :- C allows us to define user defined functions in various way. Th-ese are

1.] function with no argument and no return type
2.] function with no arguments and a return type

3.) Functions with arguments and with no return type
4.) Functions with arguments and with return type

4.) Function with no arguments and no return type :- These are the functions which do not take any data item from calling function in the form of parameters and do not return anything to calling function after execution. As these function do not contain any parameters, a pair of empty braces follow the name of function

Program :-

```
//check from two number that which is bigger
#include <stdio.h>
#include <conio.h>
void greatnum();
int main()
{
  greatnum();
  return 0;
}
void greatnum()
{
  int i,j;
  printf("Enter 2 numbers that you want to compare...");
  scanf("%d %d",&i,&j);
  if(i>j){
    printf("The greater number is: %d",i);
}
```

```
    else {
        printf("The greater number is : %d", j);
    }
}
```

2.) Function with no arguments and a return value :- This type of function does not take any value from calling function. After execution, they return some value to main function. The data type of returned value determines the return type of function.

Program :-

```
#include<stdio.h>
#include<conio.h>
int greatnum();
int main()
{
    int result;
    result = greatnum();
    printf("The greater number is %d", result);
    return 0;
}
int greatnum()
{
int i, j, greaternum;
printf("Enter 2 numbers that you want to compare.");
scanf("%d %d", &i, &j);
if (i > j) {
    greaternum = i;
```

```
}
else {
greaternum=j;
}
return greaternum;
}
```

3.4 Function with arguments and no return type :-

The function of this type take some value from the calling function and don't return any value from called function from calling function. The values which are passed with the function call are called as parameters or arguments

Program :-

```
#include<stdio.h>
#include<conio.h>
void greatnum(int a,int b);
int main()
{
 int i,j;
 printf("Enter 2 numbers that you want to compare...");
 scanf("%d%d",&i,&j);
 greatnum(i,j);
 return 0;
}
void greatnum(int x,int y)
{
 if(x>y){
```

```
        printf("The greater number is %d", x);
    }
    else {
        printf("The greater number is - %d", y);
    }
}
```

4. Function with arguments and return value :-

In this type of user defined functions, the values are passed from calling function to the called function in the form of parameters. The result is calculated in the function. The result is passed back from called function to calling function by using return statement

Program :-

```
#include<stdio.h>
#include<conio.h>
int greatnum(int a, int b);
int main()
{
int i, j, result;
printf("Enter 2 numbers that you want to compare ...");
scanf("%d %d", &i, &j);
result = greatnum(i, j);
printf("The greatnumbes is %d", result);
return 0;
}
int greatnum(int x, int y)
```

```
{
  if (x>y) {
    return x;
  }
  else {
    return y;
  }
}
```

9.) Actual parameters :- The variables used to pass information from calling function to the called function are called Actual parameters

10.) Formal parameters :- The variables created in the function body to hold the values of actual parameters are called formal parameters

11.) Passing Parameters to function :- As we know that parameters are passed from calling function. We can call a function in two different ways

1.) Call by value method
2.) Call by reference method

1.) Call by value method :- In Call by Value method a copy of actual argument created and passed to formal arguments in the function definition. Thus for each and argument, A copy of value is created and

passed to formal argument. That is why called "Call by Value method".

2. Call by reference method :- In this method of calling a function, the address of actual arguments is passed, instead of creating the copy of actual argument. The called function makes changes directly to actual parameters and no copies of parameters are created. This process increases the efficiency of programs.

Program :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int sum(int *, int *);
 int a,b, ans;
 printf("\n Enter 2 numbers");
 scanf("%d %d",&a,&b);
 ans = sum(&a,&b);
 printf("\n The result after addition is %d",ans);
 getch();
}
 int sum(int *j1, int *j2)
 {
 int result;
 result = *j1 + *j2;
 return(result);
}
```

12.) Recursion :- Recursion is a technique that involves a function calling itself from its body. Recursive function is a function that calls itself from its own body. The function keeps on calling itself till a particular condition holds true

Program to Calculate factorial of a Number Using recursion :-

```
#include<stdio.h>
#include<conio.h>
int fact(int);
int main()
{
 int n,f;
 printf("Enter number whose factorial you want to calculate");
 scanf("%d",&n);
 F=fact(n);
 printf("factorial= %d",f);
}
int fact(int n)
{
 if(n==0)
 {
  return 0;
 }
 else if(n==1)
 {
  return 1;
 }
 else
```

```
{
    return n*fact(n-1);
  }
}
```

Advantages :-

1. Recursion makes the program code compact
2. For some complicated problems, recursion can lead to solutions that are much clearer and easier to write and modify.
3. Recursion is very simple and suitable to be used for data structures like stacks, queues, trees etc.
4. In case of recursion system takes care of internal stack.
5. Recursion is useful for the problem in which there is some repetition.

Limitations :-

1. Recursion is slower in terms of speed and execution time.
2. Recursion takes more memory space as variables

13. Local Variables :- Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. [All the variables a, b, c are local to main() function

Example :-

```
#include<stdio.h>
#include<conio.h>
Void main(){
  {
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf("Value of a = %d, b = %d and c = %d \n", a, b, c);
  }
```

14. Global Variables :- Global variables are defined outside a function, usually on top of the program. Global Variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program. A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Example :-

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

```
int g;
Void main()
{
 int a,b;
 a = 10;
 b = 20;
 g = a+b;
 printf("Value of a = %d, b = %d and g = %d \n", a,b,g);
}
```

Storage Classes in C :- Storage classes in C are used to determine the life-time, visibility, memory location, and intial value of a variable. There are four types of storage classes in C

1. Automatic
2. External
3. Static
4. Register

	Storage classes	Storage Place	Default Value	Scope	Lifetime
1.	Auto	RAM	Garbage Value	local	within function
2.	Extern	RAM	Zero	Global	Till end of main program maybe declared anywhere in program
3.	Static	RAM	Zero	Local	Till end of main program, Retains value between multiple functions Call

4.	Register	Register	Garbage Value	Local	within the function

1. Automatic :- Automatic variables are Allocated memory automatically at runtime. The Visibility of automatic Variables is limited to the block in which they are defined. The scope of the automatic Variables is limited to the block in which they are defined. The automatic Variables are intialized to garbage by default. The memory assigned to automatic Variables gets freed upon exiting from the block. The keyword used for defining automatic variables is auto. Every local variable is automatic in C by default

Example :-

```
#include <stdio.h>
#include <conio.h>
Void main()
{
 int a;
 char b;
 float c;
 printf(" %od %c %f ",a,b,c);
}
```

2. Static :- The Variables defined as static specifer can hold their value between the multiple function calls. Static local variables are visible only to the function or the block in which they are defined. A same static Variable can be declared

many times but can be assigned at only one time
Default intial value of static integral variable is
0 otherwise null. The visibility of the static glo-
-bal variable is limited to file in which it has
declared. The keyword used to define static Variab
is static.

Syntax :-
Example :- Static data-type variable_name;

```
#include<stdio.h>
#include<conio.h>
Static char c;
Static int i;
Static float f;
Static char s[100];
Void main()
{
  printf("%d%d%f%s",c,i,f);
}
```

3.) Register :- The variable defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in CPU. We can not dereference the register variables ie we cannot use & operator for register variables. The access time of register variables is faster than automatic variables. The intial default value of register local variable is 0. The "register keyword is used for variable which should be stored in CPU register. We can store pointers into register. [A register can store address of Variables. Static variable can not be

stored into the register since we can not use more than one storage specifier for same variables

Syntax :-
register data_type variable_name

Example :-
```
#include<stdio.h>
#include<conio.h>
int main()
{
 register int a;
 printf("%d",a);
}
```

4.) External :- The external storage class is used to tell the Compiler that the variable defined as extern is declared with an external linkage elsewhere in program. The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in program the default initial value of external integral type is 0 otherwise Null. We can only initialize the extern variable globally. We can not initialize external variable within any block or method. An external variable can be declared many times but can be initialized at only one. If a variable is declared as external then Compiler searches for that variable to be initialized somewhere in program which may be extern or static. If it is not, then Compiler will show an error.

Example :-
```
#include<stdio.h>
```

```
#include<conio.h>
void main()
{
 extern int a;
 printf("%d ",a);
}
```

It will print error

Example:-

```
#include<stdio.h>
int a;
int main()
{
  extern int a;
  printf("%d ",a);
}
```

Syntax:-

extern data_type variable_name

Example:-

```
#include<stdio.h>
#include<conio.h>
Extern int gl=10;
Void main()
{
  void f1();
  printf("\n the value of gl is = %d",gl);
  f1();
  printf("\n the value of gl after calling f1() is %d",gl);
  getch();
}
void f1()
{
  gl=gl+5;
}
```