Module 4

Functions and Storage Classes

FUNCTIONS

Ques 1) Define functions with example. Explain how to define a function in C language.

Ans: Function

A function is a group of statements that together perform a task. Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

A function is a block of code that has a name and it has a property that it is reusable i.e. it can be executed from as many different points in a C program as required.

Defining Function

In C language, one can define the function using the following syntax:

Syntax:

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

The function definition consists of two sections – the first section is header, and the other section is function body. Different parts of a function are as follows:

1) **Return Type:** A function generally returns a value. The return_type of the function shows the data type of a value that is to be returned. Some functions successfully perform the calculation and do not return a value. In such conditions, the **void** return_type is used.

2) **Function Name:** Function name shows the actual name given to a function. The function name alongwith arguments list collectively constructs the function signature.

3) **Parameters:** Parameters look like a placeholder. One can pass the value to the parameter whenever function is invoked. Such value is known as actual parameter or argument. The argument list contains the data type, order and number of arguments. A function may contain either zero or more parameters. Parameters are optional within the function.

4) **Function Body:** A group of statements are included in the body of a function. These statements define the things that are to be done by function.

For example, the max() function consists of various source code. This function takes two arguments num1 and num2 of int type and returns the maximum of two values:

```
/* Function Returning the Max between Two Numbers
*/
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if(num1>num2)
      result = num1;
   else
      result = num2;
 return result;
}
```

Ques 2) What is function prototype? Explain with example.

Or

Explain how to declare a function in C language with example.

Or

What are the rules for declaring a function?

Ans: Function Prototype/Function Declaration

A function prototype is a function declaration that specifies the data types of its arguments in the parameter list. The compiler uses the information in a function prototype to ensure that the corresponding function definition and all corresponding function declarations and calls within the scope of the prototype contain the correct number of arguments or parameters, and that each argument or parameter is of the correct data type.

The function prototype or prototype of the function is referred as function declaration statement. Function declaration does not contain code; it only consists of header part of the function. The header includes the following three parts:

Syntax:

```
return_type Function_name(argument list);
```

1) **Return Type:** It shows the return type of function. If the function does not contain any arguments, then the return type is void else one has to write the return type of function. If there is more than one argument then each should be separated by comma (,).

2) **Function Name:** It is the name of function and is given by programmer similar to variable declaration.
3) **Argument List:** The naming of arguments present in the argument list is not necessary. Argument list is only used to provide information to compiler regarding the number, data type and order. It is necessary that arguments present in function declaration, and the function definition should not have same name. The compiler checks its data types if there is mismatch then it returns an error.

At the end of function declarations, semicolon should be added. The function declaration provides the quick reference. Thus, function used in the program can access them very easily.

For example,

```
int largest(int a, int b, int c);
```

Explanation: The above definition of function provides the information to the compiler that function largest () have return type int. The function consists of three arguments and every argument is of integer type. The naming of argument is not important.

Rules for Declaring a Function

1) If there is two or more parameter in the list then each should be separated by commas.
2) The parameter names in prototype declaration and function definition should not be similar.
3) The type of function must be similar to type of parameters.
4) In the function declaration, the names given to parameters are optional.
5) The list is written as void if no formal parameters are present in the function.
6) If function returns integer type then return type is optional.
7) When no value is returned then the return type will be void.
8) If the function definition type and function declaration type are not similar, then compiler shows an error.

Ques 3) What are actual and formal arguments? Discuss with example.

Or

What do you mean by parameters in functions?

Or

Differentiate actual and formal parameters with example.

Or

What are the rules regarding the relationship between formal arguments and Actual arguments. Support your answer with example.

Ans: Function Arguments/Parameter

A parameter is a special kind of variable, used in a subroutine to refer to one of the pieces of data provided as input to the functions. These pieces of data are called arguments.

In C language, there are two types of function arguments:

1) **Actual Arguments:** If any portion of a program calls the function then the parameters present in the calling portion of program is known as actual arguments.

For example,

```
main()
{
        int x, y;
        _______
        _______
        output(x, y); /* x and y are the actual
arguments */
}
```

2) **Formal Arguments:** The parameters written in the function definition are known as formal parameters. It is also known as dummy arguments or parametric variables.

For example,

```
main()
{
        int x, y;
        _______
        _______
        output(x, y);
}

output(a,b) */ formal or dummy arguments */
int a, b;
{
      /* body of the function */
}
```

The name of formal arguments present in called function may or may not be similar to calling function but the data types should be same.

For example, the following function declaration is invalid:

```
main()
{
        int x, y;
        char s1, s2
        _______
        _______
        function(x, y, s1, s2);
}
function (x, y, s1, s2)/* data type mismatch */
char x, y;
int s1, s2;
{
        /* body of a function */ }
```

Rules Regarding the Relationship between Formal Arguments and Actual Arguments

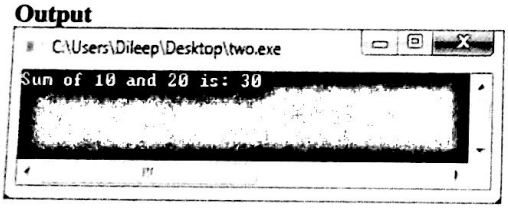
1) The arguments of calling functions are actual arguments. The arguments of called function are formal arguments.
2) The number of actual and formal arguments and their data types should match.

3) The data type of formal argument may be declared in the next line which follows the function declaration statement.
4) The formal arguments used in defining a function may be scalars or arrays. Each formal argument type must be specified.
5) The actual arguments in the calling function must agree in number, order and type with the formal arguments in the function declaration.

For example, let consider the following program:

```
/* demonstrating relationship between actual and formal arguments */
#include<stdio.h>
#include<conio.h>
int addTwoInts(int, int); /* Prototype */

int main()                    /* Main function */
{    int n1 = 10, n2 = 20, sum;
     /* n1 and n2 are actual arguments. They are the source of data. Caller program supplies the data to called function in form of actual arguments. */
      sum = addTwoInts(n1, n2);     /* function call */
      printf("Sum of %d and %d is: %d \n", n1, n2, sum);
      getch();
}
 /* a and b are formal parameters. They receive the values from actual arguments when this function is called. */

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

Output



Explanation: In above piece of code, the variables n1 and n2 are called actual arguments, whereas the variables a and b are called formal arguments. When values are passed to a called function the values present in actual arguments are copied to the formal arguments. In case of above program the values of n1 and n2 are 10 and 20 respectively.

The main() would call addTwoInts with n1 and n2 as its actual arguments, and addTwoInts will send back the computed sum to the main(). Conclusively, when a function is called all actual arguments (those supplied by the caller) are evaluated and each formal argument is initialized with its corresponding actual argument.

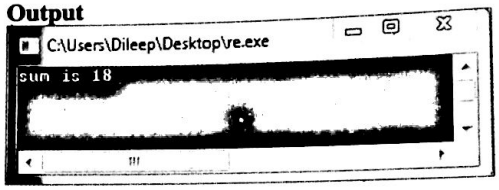
Difference between Actual and Formal Parameters

The major difference between actual and formal arguments is that actual arguments are the source of information; calling programs pass actual arguments to called functions. The called functions access the information using corresponding formal arguments.

For example, let consider the following program:

```
#include<stdio.h>
#include<conio.h>
void sum(int i, int j, int k);

/* calling function */
int main()
{    int a = 5;
     // actual arguments
     sum(3, 2 * a, a);
     getch();
     return 0;
}

/* called function */
/* formal arguments*/
void sum(int i, int j, int k)
{
     int s;
     s = i + j + k;
     printf("sum is %d", s);
}
```

Output



Explanation: Here 3, 2*a, a are actual arguments and i, j, k are formal arguments.

Ques 4) What do you mean by parameter passing mechanism? What are the different parameters passing methods?

Or

Write a program to implement call by Reference.

Or

Discuss the two important methods of parameter passing with examples.

Or

What is difference between call by value and call by reference?

Ans: Parameter Passing

In order to pass data to a function as arguments, a particular mechanism is used. The individual arguments are known as actual arguments. A name is given to function after which arguments list are enclosed within parentheses.

The arguments specified in the function definition are called as formal arguments. The formal arguments must be similar to actual arguments in terms of type, order and number. The constants, variables, arrays or expressions can be used as actual arguments.

Parameters Passing Methods

1) **Call by Value:** In call by value arguments passing method, operation is performed on the formal

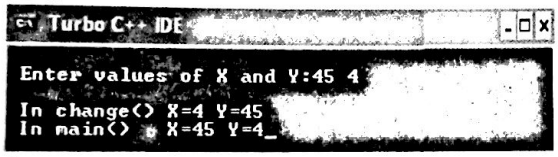
arguments whenever the values of actual arguments are passed to formal arguments. If some changes happened in the formal arguments, then it will not affect the actual arguments as it is a photocopy of actual arguments.

If this method is used to call a function, it does not change the actual arguments. The changes done in formal argument will exist only in the block of called function. Whenever the control returns back to calling function, the changes done will disappear.

Program: /* To Send Values by Call by Value*/

```
#include<stdio.h>
#include<conio.h>
void main()
{    int x, y, change(int, int);
     clrscr();
     printf("\n Enter values of X and Y:");
     scanf("%d%d",&x,&y);
     change(x,y);
     printf("\n In main()  X=%d Y=%d",x,y);
     getch();
}
change(int a, int b)
{    int k;
     k=a;
     a=b;
     b=k;
     printf("\n In change() X=%d Y=%d",a,b);
}
```

Output



Explanation: In this program, the 'x' and 'y' are the actual argument whose values are passed to called change() function. These values are received by the formal arguments 'a' and 'b'. As it is the program of swapping, hence, the values of variable 'a' is assigned to temporarily variable 'k', 'b' is assigned to variable 'a' and finally 'k' is assigned to variable 'b'. These values are printed in the change() function. When the control returns back to calling function the changes happens disappears. Again in the main() function, the control prints the actual values of 'x' and 'y' which is taken from the keyboard. In this method, formal arguments work as the duplicate of actual arguments. The change happens exist only for some time duration.

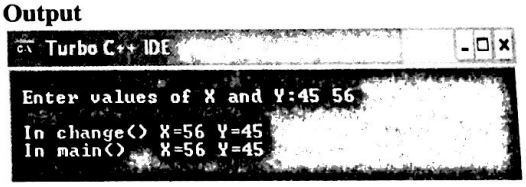
2) **Call by Reference (Address):** In this method of argument passing, the address of actual arguments is passed to function. Thus, addresses of actual arguments are considered when the function is called.

When the control comes in called function, the address of actual arguments is replaced with address of formal arguments and the statements of function get executed. The formal arguments are pointer type and it should be similar to actual arguments type. This method play major role when the array or structures are used in function and one needs to return more than one value to calling function. Return statement is used to return a value from called function. Extra values of pointer type are transferred to calling function through formal argument.

Program: /* To send a value by reference to the user-defined function.*/

```
#include<stdio.h>
#include<conio.h>
void main(){
     int x, y, change(int*, int*);
     clrscr();
     printf("\n Enter values of X and Y:");
     scanf("%d%d", &x, &y);
     change(&x, &y);
     printf("\n In main()  X=%d Y=%d", x, y);
     getch(); }
change(int*a, int*b)
{
     int *k;
     *k=*a;
     *a=*b;
     *b=*k;
     printf("\n In change() X=%d Y=%d", *a,*b);
     return; }
```

Output



Explanation: In this program, initially the formal arguments are taken into consideration and their addresses are passed to called function change(). The pointer takes the address and indicates the actual argument. In this way, the function will work on actual argument using pointer. Thus, permanent changes will happen in the values. There is no need to use return statement in this method.

Differences between Call by Value and Call by Reference

Table 4.1 shows the some basic difference between call by value and call by reference.

Table 4.1: Differences between Call by Value and Call by Reference

	Call by Value	Call by Reference
1)	The formal parameters locally exist within the block of function.	Formal parameter is photocopy of actual parameter.
2)	Formal parameters are initialized with the help of the value of actual parameter.	Actual and formal parameter both refer to same memory cell.

3)	If there is a change in the formal parameter then it is not passed to calling function.	As formal parameter point the actual parameter, hence, changing the formal parameter will also change the actual parameter.
4)	C and Java limits themselves to use Call by Value method.	C++ and Pascal declare that a parameter is a reference.

Ques 5) Explain how to call a function in C language.
Or
Explain the how to access a function in C language. Support your answer with suitable program.

Ans: Function Call
In order to call and access any function, one has to specify the name of function and list of parameters surrounded by the parentheses.

The function call can be located either in simple expression, or can take the form of operand in a complex expression.

If a function call exists within an expression, then it will be assessed initially next the parentheses will be executed. Thus, a function call is a postfix expression. The operator used (.) shows the high level of precedence.

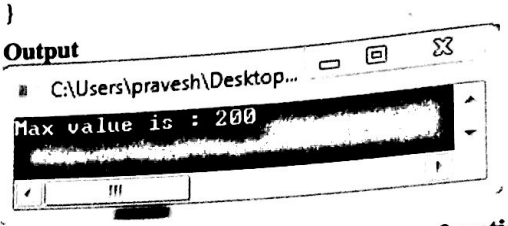
The function name works as operand, while the parentheses set (.) works as operator in function call. The actual parameters are enclosed in the parentheses. These parameters should be equal to formal parameters in terms of order, data types and number. The parameters within parentheses are separated by commas.

Note:
1) In case the formal parameters are less than the actual parameters, then the additional actual arguments will be rejected.
2) In case the formal parameters are more than the actual parameters, then the unmatched formal arguments will take the garbage values.
3) If there is mismatch between data types of parameters, then the garbage values will be taken.

Program 1: /*Function and Function Calls*/

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

/* function declaration */
int max(int num1, int num2);

main()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get maximum value */
   ret = max(a, b);
   printf("Max value is : %d\n", ret);
   getch();
}

/* function returning the max between two numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;
   if(num1>num2)
      result = num1;
   else
      result = num2;
   return result;
}
```

Output



Ques 6) What are the different library functions? List out the all functions with its purpose.

Ans: Library/Standard Functions
Functions which are already defined in C language compiler are known as library functions. Functions such as scanf and printf are the examples of library functions.

The C language consists a library of standard functions. These libraries include a source code and re-locatable code. Re-locatable code is able to run anywhere as it is compiled into an intermediate code. Intermediate code is linked with the compiled main program. Hence, the re-locatable code generates the final object code program.

Standard functions consist of the following commonly used functions:
1) Functions to perform mathematical calculation such as addition, subtraction, finding square of number, etc.
2) Functions to perform string operations such as concatenation of the strings, comparison between two strings, etc.
3) Functions to read and write characters.
4) Functions to open and close files.
5) Functions to perform operations on the characters, such as changing lowercase letters to uppercase letters and *vice versa*.

Some common library functions of C language are shown in **table 4.2**:

Table 4.2: Library Functions of C

Function	Type	Purpose	Include File
abs(i)	int	Returns the absolute value of variable i	stdlib.h
exit(u)	void	Terminates the program.	stdlib.h
fclose(f)	int	Closes file f. If the file is successfully closed, returns 0.	stdio.h
fgete(f)	int	Enters a single character from file f.	stdio.h

fgets(s, i, f)	char	Enter string s, containing i characters, from file f.	stdio.h
fopen(s1, s2)	FILE*	Opens a file named s1 of s2 type. Returns a pointer to a file.	stdio.h
fprintf(f,...)	int	Sends data items to file f.	stdio.h
fputc(c, f)	int	Sends a single character to file f.	stdio.h
fputs(s, f)	int	Sends string s to file f.	stdio.h
fread(s, i1, i2, f)	int	Inputs i2 data items each of size i1 bytes, from file f to string s.	stdio.h
free(p)	void	Frees a block of allocated memory whose starting address is indicated by p.	Malloc.h or stdlib.h
fscanf(f,...)	int	Inputs data items from file f.	stdio.h
fseek(f, l, i)	int	Moves the pointer for file f at distance l bytes from location i (i may represent pointer position or end-of-file).	stdio.h
pow(d1, d2)	double	Returns d1 raised to the power d2.	math.h
printf(...)	int	Sends data items to the standard output device.	stdio.h
putchar(c)	int	Sends a single character to the output device.	stdio.h
puts(s)	int	Sends the string s to the standard output device.	stdio.h
scanf(....)	int	Allows input of data items from the standard output device.	stdio.h
strcmp(s1, s2)	int	Compares two strings. Returns 0 if s1 = s2, –ve value if s1 < s2 and +ve if s1 > s2.	string.h
strcpy(s1, s2)	char*	Copies string s2 to string s1.	string.h
strlen(s)	int	Returns number of character in string s.	string.h
toascii(c)	int	Converts value of argument to ASCII.	ctype.h
tolower(c)	int	Converts value of argument to lower-case.	ctype.h or stdlib.h
toupper(c)	int	Converts letter to upper-case.	ctype.h or stdlib.h

In the main portion of the program, some certain information needs to be included for using the standard library function. Special files store such types of information. One can access such files in the program using the following preprocessor statement:

#include<filename>

The special files are known as "header" file. It is provided to program with the compiler. stdio.h, string.h are some examples of header files where 'h' indicates header file. Header file must be included before main function, i.e., at the top of the program.

Ques 7) What do you mean by user-defined function?

Or

What is the need for user-defined functions? Explain with example.

Or

What are the advantages of user-defined functions?

Ans: User-Defined Functions

When the user creates a function according to his needs, then these functions are known as user-defined functions. In this function, various types of constructs such as looping, jumping and decision making statements are being used.

User-defined functions are different from the built-in functions because they are created by user when he/she writes the program, while library function is pre-defined function.

One can use the user-defined function as library functions.

Need for User-Defined Functions

There are many situations when the operations or calculations are repeated several times in a program. **For example,** using some common lines in a program many times or factorial of a number. In such situations, the code is to be copied and pasted to points where it is needed within the program. For this reason, user-defined functions are used. Thus, user can save time and memory spaces by using the user-defined functions.

Advantages of User-Defined Function

1) The program will be easier to understand, maintain and debug.
2) Reusable codes that can be used in other programs
3) A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

Ques 8) What is the difference between user function and library function in C?

Ans: Difference between User Defined Function & Library Function in C

User Defined Functions	Library Function
User-defined functions are those functions which are defined by the user at the time of writing program.	Library functions are those functions which are defined by C library.
User-defined functions are part of the program which compiles runtime.	Library functions are part of header file (such as MATH.h) which is called runtime.
In user-defined functions, the name of function id decided by user	In library functions, it is given by developers.
In user-defined function, name of function can be changed any time.	In library function, name of function cannot be changed.
For example, fibo, mergeme	**For example,** printf(), scanf(), strcat()

Ques 9) Describe the various functions on the basis of argument passing and return value with example.

Or

How values can be passed as an argument to the function?

Ans: Categories of Functions

A function, depending on whether arguments are present or not and whether a value is returned or not, may belong one of the following ways:

1) **No Arguments and No Return Values:** It is the simplest way of writing a user-defined function in C. There is no data communication between a calling portion of a program and a called function block. The function is invoked by a calling environment by not feeding any formal arguments and the function also does not return back anything to the caller.

For example, let consider the following program:

```
main(){
    int x, y;
    message(); /* function is invoked */}
message(){
    /* body of the function */}
```

2) **Arguments but No Return Values:** This type is passing some formal arguments to a function but the function does not return back any value to the caller, it is a one way data communication between a calling portion of the program and the function block.

For example, let consider the following program:

```
main()
{
    int x, y;
    power(x, y); /* function declaration */
}
power(x, y)
int x, y;
{
/* body of the function */
/* no values will be transferred back to the caller */
}
```

3) **Arguments with Return Values:** This type function is passing some formal arguments to a function from calling the portion of the program and the computed values, if any, is transferred back to the caller. Data are communicated between calling the portion and a function block.

For example, let consider the following program:

```
main()
{
        int x, y, temp;
        char ch;
        temp = output(x, y, ch);
}
output(a, b, s)
int a, b;
char s;
{
        int value;
        /* body of the function */
        return(something);
}
```

4) **No Arguments but Return a Value:** Sometimes one may need to design functions that may not take any arguments but returns a value to the calling function.

For example, let consider the following program:

```
int get_number(void);
main{
    int m = get_number();
    printf("%d", m);
}
int get_number(void)
{
    int number;
    scanf("%d", &number);
    return(number);
}
```

5) **Functions that Return Multiple Values:** If one wants to get more information from a function then he/she can achieve this in C using the arguments not only to receive information but also to send back information to the calling function. The arguments that are used to "send-out" information are called output parameters.

The mechanism of sending back information through arguments is achieved using what are known as the address operator (&) and indirection operator (*).

For example, let consider the following program:

```
void mathoperation(int x, int y, int *s, int *d);
main() { int x = 20, y = 10, s, d;
    mathoperation(x, y, &s, &d);
    printf("s=%d\n d=%d\n", s, d); }
void mathoperation(int a, int b, int *sum, int *diff)
{   *sum = a + b;
    *diff = a – b;
}
```

The actual arguments x and y are input arguments, s and d are output arguments. In the function call, while we pass the actual values of x and y to the function, we pass the addresses of locations where the values of s and d are stored in the memory.

Ques 10) What is recursion? Explain recursive function by taking suitable example.

Or

Write the recursive and non-recursive programs to find the factorial of a number.

Ans: Recursive Function and Recursion

A function calls itself several times until the desired objective is obtained, thus, C language support recursion.

In recursion, a problem is divided into smaller sub-problems. These sub-problems can be solved by calling a function repeatedly. In this way, the original problem can be solved.

As several algorithms are recursive in nature, it is considered as effective programming tool. Recursion is a process in which the function calls itself till the desired objective is not to be achieved.

A process in which a new activity depends on the previous result of calculation, then it is known as recursion. Many problems which are repetitive in nature can be written in recursive form. Two conditions must be fulfilled for solving a problem using recursion method:

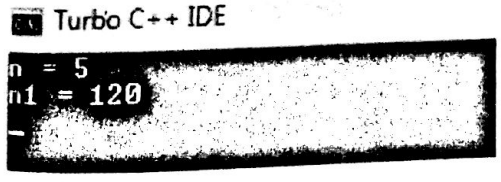
1) The problem must be represented in a recursive form, and
2) There must be a stopping condition in order to find the final result of the problem.

For example, let consider that one has to find out the factorial of a positive number. This problem can be written as n! = 1 × 2 × 3 ×...× n, where n indicates the positive integer. This problem can also be rewritten as n! = n × (n – 1)!. This statement is known as recursive statement. Here, the calculation of factorial n depends on the last result which works as input to next result. Let assume that value of (n – 1)! is known. The value of 1! = 1 is known by definition. The final expression shows the stopping point of recursion.

Program: Recursive Program to Find Factorial of a Number

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

long int factorial(int n);      /* function prototype */
main()
{    int n;
     long int factorial(int n);
     /* read in the integer quantity */
     clrscr();
     printf("n = ");
     scanf("%d", &n);
     /* calculate and display the factorial */
     printf("n1 = %1d\n",  factorial(n));}
long int factorial(int n)       /*calculate the factorial */
{
if(n<=1 )
          return(1);
    else
          return (n * factorial (n–1));
}
```

Output



Explanation: In the above program, the value of n is entered via keyboard. The main function reads the value of n and next calls the factorial() function. The function calls itself several times recursively. The value of actual argument is reduced by one unit for each call. When the value of n becomes equal to 1 then the recursive calls terminate.

The if-else statement of the factorial () function also contain termination condition which will work when the value of n<=1.

The factorial function will be retrieved repeatedly several times, once in main() function and (n – 1) times within itself. This activity is not known to a person. He/she only sees the final result as follows:

n = 10
n! = 3628800

Whenever the program starts executing, the function calls does not work immediately. They are pushed into a stack until the termination condition is not obtained. When they are popped from the stack, the function calls execute in reverse order. The function calls will be executed in the following manner:

n! = n × (n – 1)!
(n – 1)! = (n – 1) × (n – 2)!
(n – 2)! = (n – 2) × (n – 3)!
(n – 3)! = (n – 3) × (n – 4)!
...
...
...
2! = 2 × 1!

The actual values will be returned in the following reverse manner:

1! = 1
2! = 2 × 1! = 2 × 1 = 2
3! = 3 × 2! = 3 × 2 = 6
4! = 4 × 3! = 4 × 6 = 24
5! = 5 × 4! = 5 × 24 = 120
...
...
...
n! = n × (n – 1)! = ···

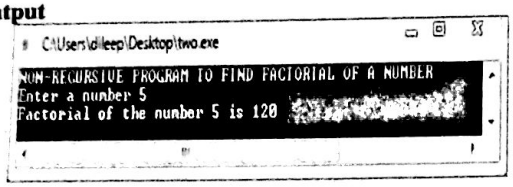
This reverse order execution shows features of all functions which are recursive in nature.

Program: Non-Recursive Program to Find Factorial of a Number

```
#include<stdio.h>
void main()
{    int n,f;
     printf("NON-RECURSIVE PROGRAM TO FIND
     FACTORIAL OF A NUMBER");
     printf("\nEnter a number");
     scanf("%d",&n);
     f=factorial(n);
     printf("Factorial of the number %d is %d",n,f);
     getch();
}
int factorial(int num)
```

```
{
    int fact=1,i;
    for(i=1;i<=num;i++)
    {
        fact=(fact*i);
    }return(fact);}
```

Output



Ques 11) Explain passing array to functions with example.

Ans: Passing Array to Functions

Array can be passed as an argument to a function. Whenever anyone needs to pass a list of elements as argument to the function, it is preferred to do so using an array.

While passing arrays to the argument, the name of the array is passed as an argument (i.e., starting address of memory area is passed as argument).

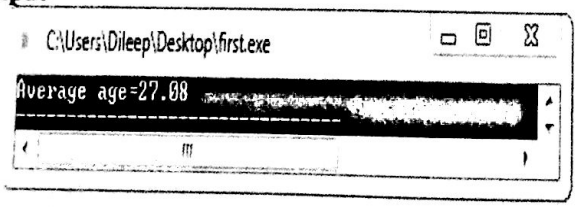
Syntax:

```
int function_name(int arr[]);
```

Program: Illustrating Passing Array to Functions

```
#include <stdio.h>
float average(float a[]);
int main()
{
   float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
   avg=average(c);    /* Only name of array is passed as
argument. */
   printf("Average age=%.2f",avg);
   return 0;
  }
float average(float a[])
{
   int i;
   float avg, sum=0.0;
   for(i=0;i<6;++i)
  {
    sum+=a[i];
   }
   avg =(sum/6);
   return avg;
}
```

Output



Explanation: The above C program is used to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

Ques 12) Explain the concept of returning an array from the function with the help of suitable program.

Ans: Returning Array from Function

C programming language does not allow to return an entire array as an argument to a function. However, one can return a pointer to an array by specifying the array's name without an index.

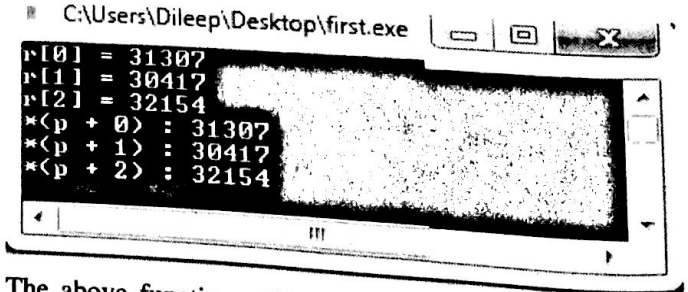
For example, if anyone wants to return an array from a function, he/she would have to declare a function returning a pointer as given below:

```
int * myFunction()
{
    ....
    ....
    ....
}
```

Program: Illustrating Returning of an Array from Function

```
#include <stdio.h>
/* function to generate and return random numbers */
int * getRandom( )
{
 static int  r[10];
 int i;
 srand( (unsigned)time( NULL ) );
 for ( i = 0; i < 3; ++i){
   r[i] = rand();
   printf( "r[%d] = %d\n", i, r[i]);

 } return r;
}
/* main function to call above defined function */
int main ()
{  /* a pointer to an int */
  int *p;
  int i;

  p = getRandom();
  for ( i = 0; i < 3; i++ )
  {
    printf( "*(p + %d) : %d\n", i, *(p + i));
  } return 0; }
```

Output



The above function will generate three random numbers and return them using an array.

STORAGE CLASSES

Ques 13) Define the storage class in 'C'.
Or
What is storage class? Describe automatic, register static and external with neat syntax.
Or
What are storage classes? Explain all its types in brief.
Or
What is storage class in C? Explain its importance. Show the usage of each storage class by making a suitable example in C.

Ans: Storage Class

A storage class defines the scope (visibility) and life time of variables and/or functions within a C Program.

A storage class helps in defining the life-time and scope of variables within the program. They come before the type that they wants to modified.

Imporatnce of Storage Class

1) It specifies the area or scope or part of the program within which it is accessible. The visibility of a variable may be either local or global.
2) Lifetime of a variable defines at what duration or time period a variable is accessible. In other words, it is difference between time when a variable is declared and when the variable is destroyed.
3) As name suggests, it provides place or location where the content of the variable is stored. In general, the variable is stored either inside RAM or internal registers of the CPU.

Types of Storage Classes

There are four types of storage class:

1) **Automatic Storage Class:** The **auto** storage class is the default storage class for all local variables. A variable defined within a function or block with **auto** specifier belongs to automatic storage class. All variables defined within a function or block by default belong to automatic storage class if no storage class is mentioned. Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block. By default they are assigned **garbage value** by the compiler.

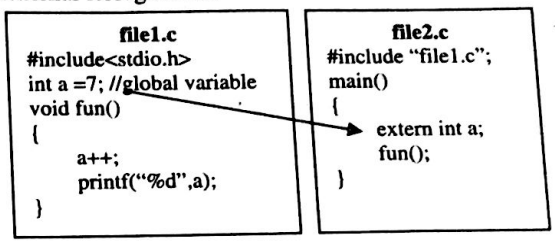
Example: Illustrating Auto Storage Class

```
#include<stdio.h>
void main()
{       int i;
        auto int j;
        printf("\ni=%d \nj=%d",i, j);
}
```

Explanation: In the above program, there are two variables in which one is declared as auto. The printf() statement only uses the auto variable as it is local within the function.

2) **External Storage Class:** The extern storage class is used to give a reference of a global variable that is visible to all the program files. When anyone uses 'extern' the variable cannot be initialised as all it does is point the variable name at a storage location that has been previously defined. External variable can be accessed by any function. They are also known as **global variables**. Variables declared outside every function are external variables.

For example, the following **figure** shows concept of external storage class:



The global variable from one file can be used in other using extern keyword.

3) **Static Storage Class:** The **static** storage class instructs the compiler to keep a local variable in existence during the lifetime of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls. The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared. In C programming, when **static** is used on a class data member, it causes only one copy of that member to be shared by all objects of its class.

For example, the following program shows the static storage class:

```
#include<stdio.h>
void test();                    //Function declaration
main()
{
  test();
  test();
  test();
}
void test()
{
     static int a = 0;          //Static variable
     a = a+1;
     printf("%d\t", a);
}
```

Output

1 2 3

4) **Register Storage Class:** The register storage class is used to define local variables that should be stored in

a register instead of RAM. Register variable has faster access than normal variable. Frequently used variables are kept in register. Only few variables can be placed inside register.

For example, the following program shows the register storage class:

```
main
{   /*Register Variable Declaration*/
    register int miles;}
```

Ques 14) What do you mean by scope of local and global variables? Explain with example.

Or

Using an example, explain local and external variables and also give their scope.

Ans: Local Variables

A variable declared inside a function is visible to that function only. Such variable is known as local variable. These variables are not visible to other function.

Scope of Local Variable

Local variables are executed when any function calls it and destroyed whenever the control goes outside the function. The statement defined within the function can use the local variables only.

Global Variables

Variables which are declared in the main() function and visible to all other functions are known as **external** or **global variables**. If the local and global variables have same name then the function in which local variable is declared, execute the local variable only and it does not affect the global variable.

Generally, global variables are declared at the starting of the program and outside the function. The value of global variable remains same throughout the program. Any function defined within the program can access the global variable.

If the global variable is declared then it can be used throughout the entire program.

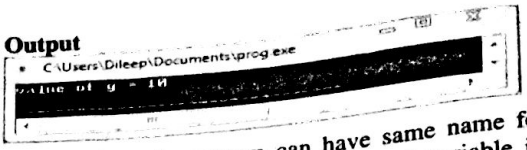
Scope of Global Variable

The scope of a global variable is the file in which it is declared. It can extend the scope of the global variable to other files by using an 'extern' declaration.

Program: Illustrating Local and Global Variables

```
#include <stdio.h>
/* global variable declaration */
int g = 20;
int main ()
{
    /* local variable declaration */
    int g = 10;
    printf ("value of g = %d\n",  g);
    return 0;
}
```

Output



Explanation: A program can have same name for local and global variables but value of local variable inside a function will take preference as shown in program above.

Ques 15) What do you mean by scope and life of different classes of a variable?

Ans: Scope and Life of Storage Classes

1) **Scope:** Scope of a variable specifies that area or part of the program within which it is active i.e., that portion of the program within which it is recognised or within which it is accessible. The scope of a variable may be either local or global:
   i) **Local scope** means the scope of variable is restricted within the module in which variables are declared.
   ii) **Global scope** means that the variable is accessible to all modules of the program i.e., they are active till the program is not terminated.These variables are declared outside all functions including the main() function.
2) **Life:** With lifetime, we mean for what time period a variable is accessible or active for a particular module. Actually lifetime of a variable is the time difference between the time when it is created and when it is destroyed.

Table 4.3 shows the scope and life of different classes of a variable:

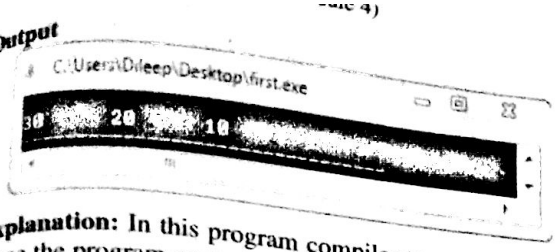
Table 4.3: Scope and Life of Variables

Information \ Storage Class	Automatic	Static	Register	External
Storage	Memory	Memory	CPU Register	Memory
Default Initial Value	Garbage	Zero	Garbage	Zero
Scope	Local to block	Local to block	Local to block	Global
Life	Till control remains within the block	Value of variable persists between function calls	Till the control remains within the block	Till the completion of the program
Keyword	Auto	Static	Register	Extern

Program: Illustrating the Scope and Life of the Variable

```
#include <stdio.h>
void main() {
 auto int a=10; {
  auto int a=20; {
   auto int a=30;
   printf("\n%d",a); }
  printf("\t%d",a);
 }
 printf("\t%d",a); }
```

Output



Explanation: In this program compiler treats three a variables totally different since they are defined in different blocks. Once the program control comes out of the innermost block the variable a with value 30 is lost, and hence the variable a in the second printf() statement refers to variable a with value 20. Similarly, when program control comes out of the next innermost block, the third printf() statement refers to variable a with value 10.

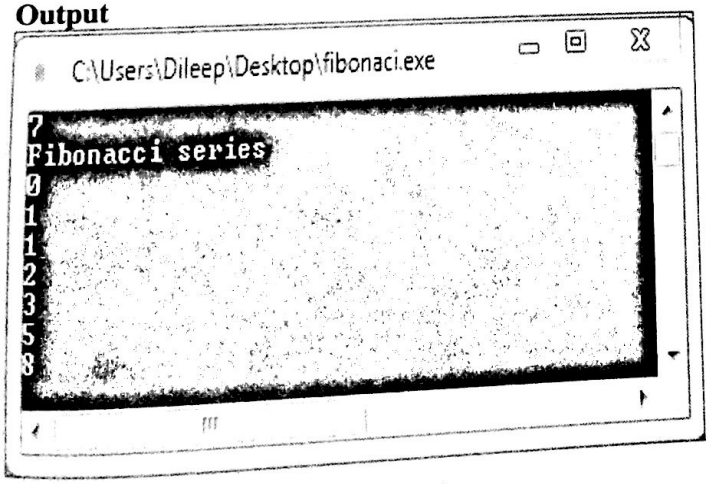
EXAMPLE PROGRAMS

Ques 16) Write a program to generate first N Fibonacci series using recursion.

Ans: Program Illustrating the Generation of First N Fibonacci Series Using Recursion

```
#include<stdio.h>
#include<conio.h>
int Fibonacci(int);
main()
{
     int n, i = 0, c;
     scanf("%d",&n);
     printf("Fibonacci series\n");
     for ( c = 1 ; c <= n ; c++ )
     {
          printf("%d\n", Fibonacci(i));
          i++;
     }
     getch();
     return 0;
}

int Fibonacci(int n)
{
     if ( n == 0 )
          return 0;
     else if ( n == 1 )
          return 1;
     else
          return (Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Output

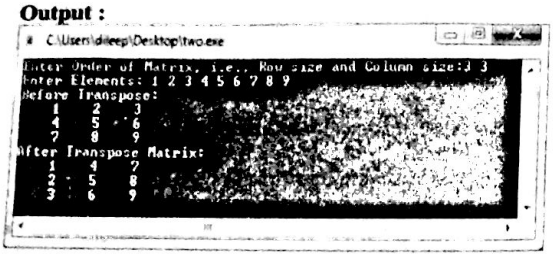


Ques 17) Using suitable functions and their parameters. Write a program to find transpose of a matrix of order specified by the user.

Ans: C Program to Find Transpose of Matrix using Functions

```
#include<stdio.h>
#include<conio.h>
void read_mat(int,int,int[10][10]);
void print_mat(int,int,int[10][10]);
void transpose_mat(int,int,int[10][10]);
main()
{  int a[10][10],b[10][10],i, j, m, n;
   printf("Enter Order of Matrix, i.e., Row size and
   Column size:");
   scanf("%d %d", &m, &n);
   read_mat(m,n,a);
   printf("Before Transpose:\n");
   print_mat(m,n,a);
   printf("After Transpose Matrix:\n");
   transpose_mat(m,n,a);
   getch();
 } void read_mat(int x,int y,int c[10][10])
 {  int i,j;
    printf("Enter Elements:\t");
    for(i=0;i<x;i++)
    for(j=0;j<y;j++)
    scanf("%d",&c[i][j]);
 } void print_mat(int x,int y,int c[10][10])
 {
    int i,j;
    for(i=0;i<x;i++)
    {
      for(j=0;j<y;j++)
      printf("%5d",c[i][j]);
      printf("\n");
    }}

 void transpose_mat(int x,int y,int a[10][10])
 {  int i, j, b[10][10];
    for(i=0;i<x;i++)
    for(j=0;j<y;j++)
    b[j][i]=a[i][j];
    print_mat(x,y,b);
 }
```

Output :

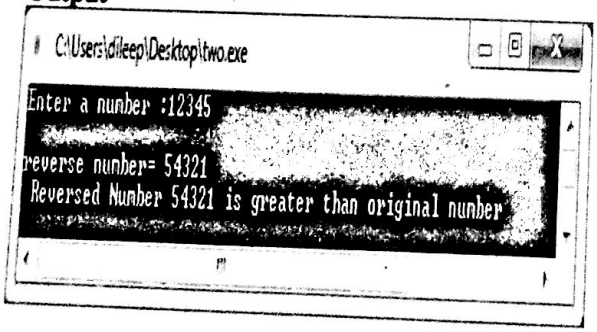


Ques 18) A five digit number is entered through keyboard. Write a function to reverse the number, and check which one is larger original or reversed number.

Ans: Program to Reverse Five Digit Number and Checking Which is Greater

```
#include<stdio.h>
int reversedigit(int);
int main()
{
    int a,count;
    printf("Enter a number :");
    scanf("%d",&a);
    int b=reversedigit(a);
    printf("\nreverse number= %d",b);
    if(a>b)
    {
        printf("\n Original Number %d is greater than
        reversed number",a);
    }
    else
    {
        printf("\n Reversed Number %d is greater than
        original number",b);
    }
    getch();
}
reversedigit(int x)
{
    int y=0;
    while(x)
    {
        y=y*10+x%10;
        x=x/10;
    }
    return y;
}
```

Output

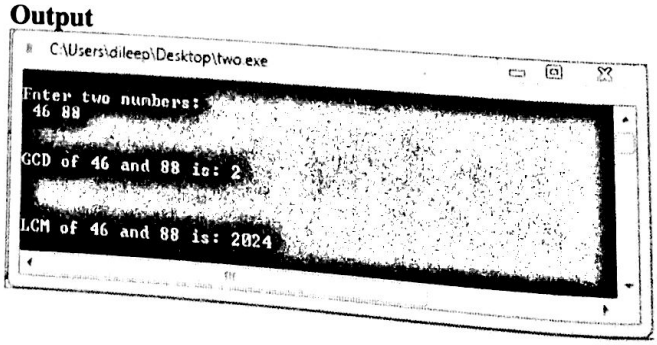


Ques 19) Write a recursive C program to calculate GCD and LCM of two numbers.

Ans: Recursive C Program to Calculate GCD and LCM of Two Numbers

```
#include "stdio.h"
int find_gcd(int,int);
int find_lcm(int,int);
int main()
{
    int num1,num2,gcd,lcm;
    printf("\nEnter two numbers:\n ");
    scanf("%d %d",&num1,&num2);
    gcd=find_gcd(num1,num2);
    printf("\n\nGCD of %d and %d is:
    %d\n\n",num1,num2,gcd);
    if(num1>num2)
        lcm = find_lcm(num1,num2);
    else
        lcm = find_lcm(num2,num1);
        printf("\n\nLCM of %d and %d is:
        %d\n\n",num1,num2,lcm);
    return 0;
}

int find_gcd(int n1,int n2)
{
    while(n1!=n2)
    {
    if(n1>n2)
        return find_gcd(n1-n2,n2);
    else
        return find_gcd(n1,n2-n1);
    }
    return n1;
}
int find_lcm(int n1,int n2)
{
    static int temp = 1;
    if(temp % n2 == 0 && temp % n1 == 0)
    return temp;
    temp++;
    find_lcm(n1,n2);
    return temp;
}
```

Output



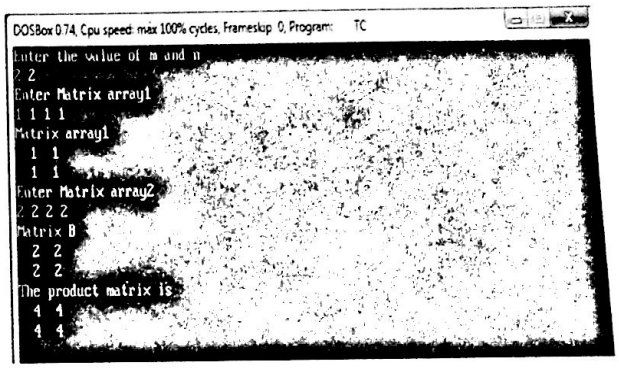
Ques 20) Write a C program to accept two matrices and display their product using functions for reading, calculating product and displaying.

Ans: C Program for Matrix Multiplication

```
#include<stdio.h>
#include<conio.h>
#define MAXROWS 10
#define MAXCOLS 10

void read(int arr[][MAXCOLS], int m, int n);
void display(int arr[][MAXCOLS], int m, int n);
void cal_product (int array1[][MAXCOLS], int
array2[][MAXCOLS],
int array3[][MAXCOLS], int m, int n);

void main()
{
   int array1[MAXROWS][MAXCOLS],
   array2[MAXROWS][MAXCOLS],
   array3[MAXROWS][MAXCOLS];
   int m, n;
   clrscr();
   printf("Enter the value of m and n \n");
   scanf("%d %d", &m, &n);
   printf("Enter Matrix array1 \n");
   read(array1, m, n);
   printf("Matrix array1 \n");
   display(array1, m, n);
   printf("Enter Matrix array2 \n");
   read(array2, m, n);
   printf("Matrix B \n");
   display(array2, m, n);
   cal_product(array1, array2, array3, m, n);
   printf("The product matrix is \n");
   display(array3, m, n);
   getch(); }

/* Input Matrix array1 */
void read(int arr[][MAXCOLS], int m, int n)
{
  int i, j;
  for (i = 0; i < m; i++) {
     for (j = 0; j < n; j++)
     {
        scanf("%d", &arr[i][j]);
     }
  }
}
void display(int arr[][MAXCOLS], int m, int n)
{
  int i, j;
  for (i = 0; i < m; i++)
  {
     for (j = 0; j < n; j++)
     {
        printf("%3d", arr[i][j]);
     }
     printf("\n");
  }}
```

```
/* Multiplication of matrices */
void cal_product(int array1[][MAXCOLS], int
array2[][MAXCOLS],
int array3[][MAXCOLS], int m, int n)
{
  int i, j, k;
  for (i = 0; i < m; i++)
  {
     for (j = 0; j < n; j++)
     {
        array3[i][j] = 0;
        for (k = 0; k < n; k++)
        {
           array3[i][j] = array3[i][j] + array1[i][k] *
           array2[k][j];
        }
     }
  }
}
```

Output



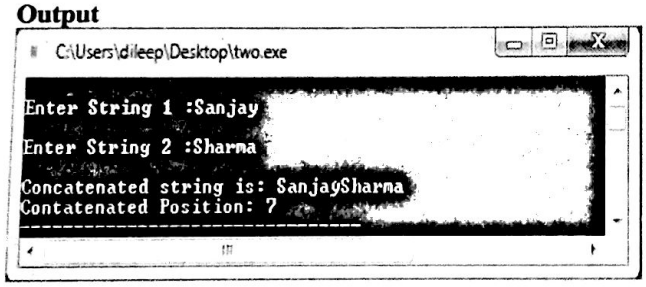
Ques 21) Write C program for joining two strings and to find the position of second string in the first without using library functions.

Ans: C Program for Joining Two Strings and Finding the Positions of Second String in First

```
#include<stdio.h>
#include<string.h>
void concat(char[], char[]);
int search(char[], char[]);
int main()
{ char s1[50], s2[30];
  int loc;
  printf("\nEnter String 1 :");
  gets(s1);
  printf("\nEnter String 2 :");
  gets(s2);

  concat(s1, s2);
  printf("\nConcatenated string is: %s", s1);
  loc = search(s1, s2);
  printf("\nContatenated Position: %d", loc+1);
  return (0);
}
void concat(char s1[], char s2[]) {
  int i, j;
  i = strlen(s1);
```

```
    for (j = 0; s2[j] != '\0'; i++, j++) {
    s1[i] = s2[j];
    }
    s1[i] = '\0';
}
int search(char src[], char str[]) {
    int i, j, firstOcc;
    i = 0, j = 0;
    while (src[i] != '\0')
    {
        while (src[i] != str[0] && src[i] != '\0')
            i++;
        if (src[i] == '\0')
            return (-1);
        firstOcc = i;
        while (src[i] == str[j] && src[i] != '\0' && str[j] != '\0') {
            i++;
            j++;
        }
        if (str[j] == '\0')
            return (firstOcc);
        if (src[i] == '\0')
            return (-1);

        i = firstOcc + 1;
        j = 0;
    }}
```

Output

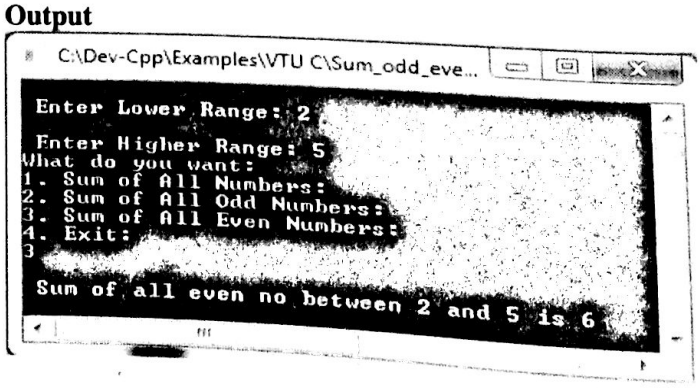


Ques 22) Write a program to find the Sum of All numbers from n to m using function.

Ans: Program to Find the Sum of All numbers from n to m

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int m,n,c;
    up:
    printf("\n Enter Lower Range: ");
    scanf("%d", &n);
    printf("\n Enter Higher Range: ");
    scanf("%d", &m);
    if(n >= m)
    {
        printf("\nImproper Input\n");
        goto up;
    }
    printf("What do you want:\n");
    printf("1. Sum of All Numbers:\n");
    printf("2. Sum of All Odd Numbers:\n");
    printf("3. Sum of All Even Numbers:\n");
    printf("4. Exit:\n");
    scanf("%d", &c);

    switch(c){
        case 1: All_sum(n,m);
                break;
        case 2: Odd_sum(n,m);
                break;
        case 3 : Even_sum(n,m);
                break;
        case 4 : exit(0); }
    getch();
    return 0; }

/*Sum of All Numbers*/
All_sum(n,m)
{
    int i,sum = 0;
    for(i = n;i <= m ; i++)
    {
        sum = sum + i;
    }
    printf("\n Sum of all no between %d and %d is %d", n,
    m, sum);
}

/*Sum of All Odd Numbers*/
Odd_sum(n, m)
{
    int x3=n,x4=m, sum=0;
    while(x3 <= x4)
    {
        if(x3 % 2 != 0)
            sum = sum + x3;
            x3++;
    }
    printf("\n Sum of all odd no between %d and %d is %d",
    n, m, sum);
}

/*Sum of All Even Numbers*/
Even_sum(n,m)
{
    int x1=n,x2=m, sum=0;
    while(x1 <= x2){
        if(x1 % 2 == 0)
            sum = sum + x1;
            x1++; }
    printf("\n Sum of all even no between %d and %d is
    %d", n, m, sum);
}
```

Output

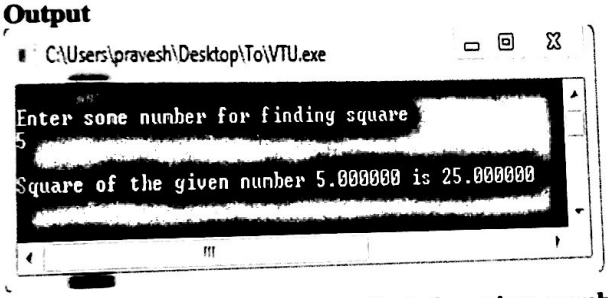


Ques 23) Write a program to calculate the square of a number using function.

Ans: Program for the Calcuation of Square of a Value

```
#include<stdio.h>
#include<conio.h>
// function prototype, also called function declaration
float square ( float x );
main( )
{
    float m, n ;
    printf ( "\nEnter some number for finding square \n");
    scanf ( "%f", &m ) ;
    // function call
    n = square (m);
    printf ( "\nSquare of the given number %f is %f",m, n
    );
    getch(); }

/* function definition*/
float square ( float x )
{
    float p ;
    p = x * x ;
    return ( p ) ; }
```

Output

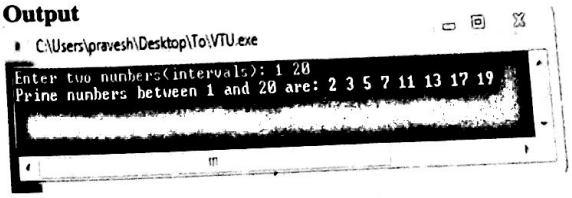


Ques 24) Write a program to find the prime numbers between two intervals using user-defined functions.

Ans: Program for Finding Prime Numbers between Two Intervals

```
#include<stdio.h>
#include<conio.h>
int check_prime(int num);
main()
{
    int n1,n2,i,flag;
    printf("Enter two numbers(intervals): ");
    scanf("%d %d",&n1, &n2);
    printf("Prime numbers between %d and %d are: ", n1,
    n2);
    for(i=n1+1;i<n2;++i)
    {
        flag=check_prime(i);
        if(flag==0)
        printf("%d ",i); }
    getch();
}
int check_prime(int num)
{
    int j,flag=0;
    for(j=2;j<=num/2;++j)
    {
        if(num%j==0)
        {
            flag=1;
            break;
        }
    }
    return flag;
}
```

Output



Ques 25) Write a program in C to swap two numbers using function.

Ans: C program to Swap Two Numbers Using swap() Function

```
#include<stdio.h>
#include<conio.h>
int swap(int , int);                    // Declaration of function
main( )
{
   int a = 10, b = 20 ;                 // call by value
   swap(a,b);                           // a and b are actual parameters
   printf ( "\na = %d b = %d", a, b ) ;
   getch();
}
int swap( int x, int y )                // x and y are formal parameters
{
   int t ;
   t = x ;
   x = y ;
   y = t ;
   printf ( "\nx = %d y = %d", x, y ) ;
}
```

Output