MIT-WPU

MIT-WORLD PEACE UNIVERSITY

F. Y. B. Tech

Trimester: I/II/III Subject: Programming and Problem Solving

Name: Krishnaraj Thadesar **Division:** 9

Roll No.: 109054 **Batch:** I3

Experiment No.: 6

Name of the Experiment: Factorial using Recursion

Performed on: 10rd February 2022

Submitted on: 17st January 2022

AIM: Write an algorithm and draw a flowchart for a C program to find the factorial of a number using recursion.

OBJECTIVE:

1. To learn and understand recursion in C
2. To learn about factorials.

PLATFORM: *Arch Linux 64 Bit*

THEORY:

Functions

C has no procedures, only functions. A function is a self-contained block of statements that perform a specified task. C provides functions which are again similar most languages. One difference is that C regards main() as function. Their definitions can't be nested but all except main can be called recursively.

Syntax

The general form of a function definition is:

```
<return type><function name> ( <formal argument list> )
        {       <local variables>
                .....
                <body>
                ......
        }
```

Example

```
int mean(int x, int y)
        {
                inttmp;
                tmp = (x + y)/2;
                returntmp;
        }
```

Function Execution

```
int mean(int x, int y)
```

This line begins the function definition. It tells us the type of the return value, the name of the function, and a list of arguments used by the function. The arguments and their types are enclosed in brackets, each pair separated by commas.

The body of the function is bounded by a set of curly brackets. Any variables declared here will be treated as local unless specifically declared as static or extern types.

```
return(ret_val);
```

On reaching a return statement, control of the program returns to the calling function. The bracketed value is the value which is returned from the function. If the final closing curly bracket is reached before any return value, then the function will return automatically, any return value will then be meaningless.

The example **function can be called** by a line in another function which looks like this

```
result = mean(a,b);
```

This calls the function mean assigning the return value to variable result.

Void Functions

If you do not want to return a value you must use the return type void and miss out the return statement:

```
void mean(int x, int y)
{
        inttmp;
        tmp = (x + y)/2;
        printf("Mean = %d",tmp);
}
```

We must have ( ) even for no parameters unlike some languages.

Function Prototyping

Before you use a function C must have ***knowledge*** about the type it returns and the parameter types the function expects. A functions must be defined before it is used (called). It is good practice to prototype all functions at the start of the program.

Scope of Function Variables

Only a limited amount of information is available within each function. Variables declared within the calling function can't be accessed unless they are passed to the called function as arguments. A function can be accessed through the use of global variables.

Local variables are declared within a function. They are created anew each time the function is called, and destroyed on return from the function. Values passed to the function as arguments can also be treated like local variables (Pass by value).

Static variables are slightly different, they don't die on return from the function. Instead their last value is retained, and it becomes available when the function is called again.

Global variables don't die on return from a function. Their value is retained, and is available to any other function which accesses them.

Recursive Functions

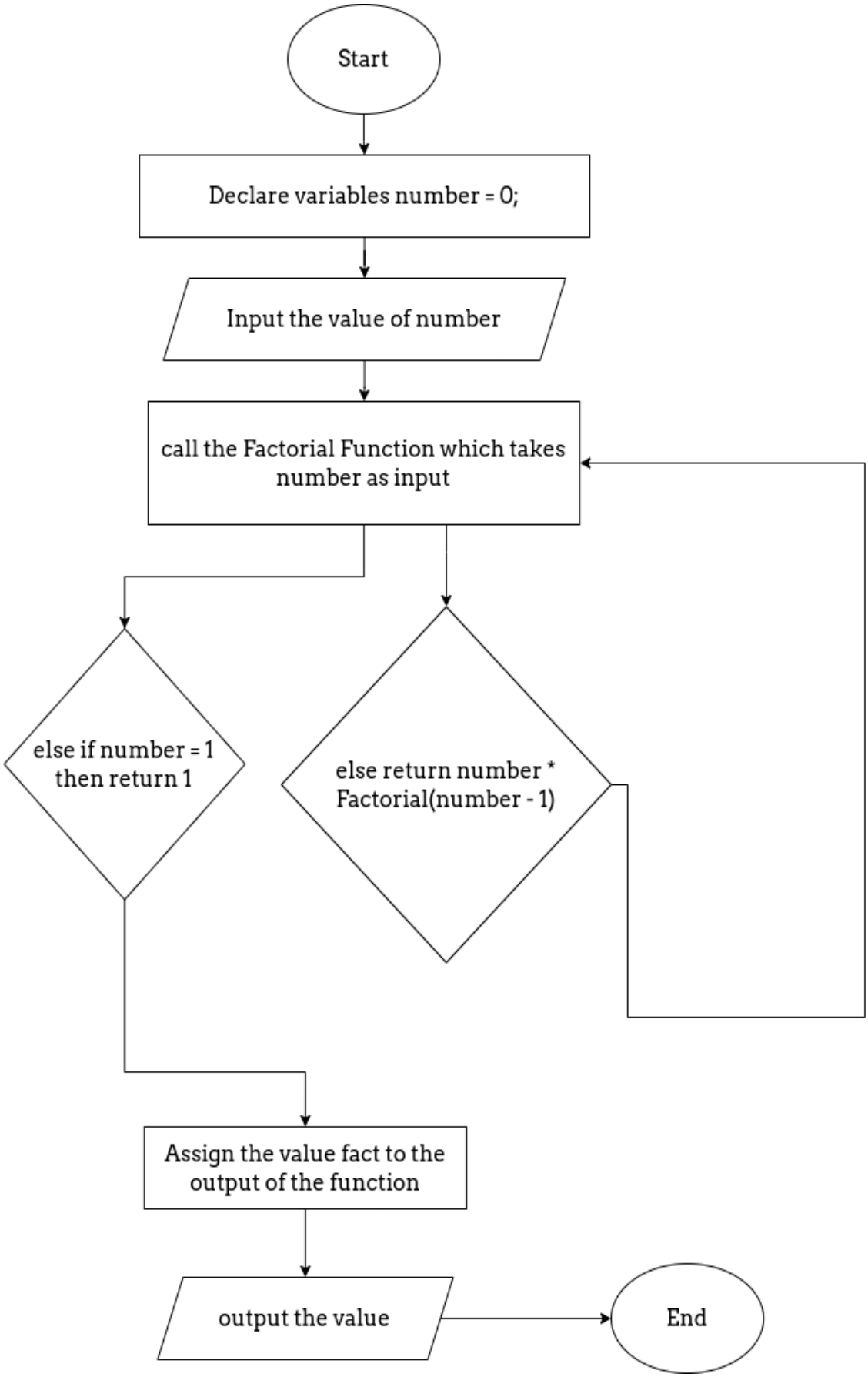
A recursive function is one which calls itself. This is another complicated idea which you are unlikely to meet frequently. We shall provide some examples to illustrate recursive functions. Recursive functions are useful in evaluating certain types of mathematical function. You may also encounter certain dynamic data structures such as linked lists or binary trees. Recursion is a very useful way of creating and accessing these structures.
Here is a recursive version of the power function.

```
double power(double val, unsigned pow)
        {
                if(pow == 0) /* pow(x, 0) returns 1 */
                return(1.0);
                else
                return(power(val, pow - 1) * val);
        }
```

ALGORITHM:

Step 1: Start
Step 2: Declare a variable number = 0
Step 3: Input the value of the variable
Step 4: Call the Factorial function passing number as the arguement
Step 5: If value of number is 1 return 1
Step 6: If value of number is not 1, then return number * Factorial(number -1)
Step 7:Assign the value of returned by factorial function to another variable fact.
Step 6: Output value of fact.
Step 7: End

Flowchart:


Start
Declare variables number = 0;
Input the value of number
call the Factorial Function which takes number as input
else if number = 1 then return 1
else return number * Factorial(number - 1)
Assign the value fact to the output of the function
output the value
End

CODE:

```c
// Write a C program to compute the factorial of a given number using recursion.

#include<stdio.h>

double factorial(int number)
{
    if(number == 1)
        return 1;
    return number * factorial(number - 1);
}


int main()
{
    int a = 0;
    printf("Enter the number whose factorial you want: ");
    scanf("%d", &a);
    printf("The factorial of the number is: %.1lf", factorial(a));
    return 0;
}
```

OUTPUT

Addition

```
Enter the number whose factorial you want: 5
The factorial of the number is: 120.0
```

CONCLUSION:

Recursion and Factorials were understood in detail.

FAQ

Q1. Explain the Proces of calculating factorial of a number using example.

4 factorial, that is, 4! can be written as: 4! = 4×3×2×1 = 24.

Q2. How to declare the function?

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 the following parts –

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows –

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

```
int max(int, int);
```

Q3. What are the ways of calculating factorial in C?

1. Using Loop

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

```
1.int main()
2.{
3. int i,fact=1,number;
4. printf("Enter a number: ");
5.  scanf("%d",&number);
6.    for(i=1;i<=number;i++){
7.     fact=fact*i;
8.  }
9.  printf("Factorial of %d is: %d",number,fact);
10.return 0;
11.}
```

2. Using Recursion like shown in the Code in this assignment.

```
long factorial(int n)
1.{
2.  if (n == 0)
3.    return 1;
4.  else
5.    return(n * factorial(n-1));
6.}
7.
8.void main()
9.{
10.  int number;
11.  long fact;
```

```
12. printf("Enter a number: ");
13. scanf("%d", &number);
14.
15. fact = factorial(number);
16. printf("Factorial of %d is %ld\n", number, fact);
17. return 0;
18.}
```

Q4. What is recursive function?

Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
  recursion(); /* function calls itself */
}
int main() {
  recursion();
}
```

Q5. When does the recursion (recursive call to a function) come to an end?

When a condition inside the recursive function is met, and the function returns a value without calling itself, or it terminates itself.

Q6. Implement the factorial program using other method

Using Loop

```
#include<stdio.h>

12.int main()
13.{
14. int i,fact=1,number;
15. printf("Enter a number: ");
16. scanf("%d",&number);
17.   for(i=1;i<=number;i++){
18.     fact=fact*i;
19. }
20. printf("Factorial of %d is: %d",number,fact);
21.return 0;
22.}
```