

JUNE 24 , 2021

PPS- ASSIGNMENT

WEEK 1

Prepared BY :
49_D1_CSE- Vidhi Bhatt

QUE-1

WRITE A PROGRAM IN C TO FIND THE FACTORIAL OF A NUMBER USING RECURSION. EXPLAIN YOUR IMPLEMENTATION IN DETAIL.

Recursion is a process in which a function calls itself. Here, first we have included the header file then we have declared a function named factorial with data type int. The actual implementation of the program starts from main. Next we define a variable n, and then we ask the user to enter a positive integer whose factorial is to be found. This value is stored at the address of n. In the next statement the control goes to the factorial function which we have defined below. Basically, factorial of a number n is the product of all positive integers less than or equal to n. Here the factorial of entered no. is calculated and the result is then passed to print statement which displays the final answer. If $n=1$, 1 is returned as output. If $n \neq 1$, the no. goes through the else part till the value of n becomes 1.

The screenshot displays the Code::Blocks IDE with a C program for calculating factorial using recursion. The code is as follows:

```

1 #include<stdio.h>
2 int factorial(int n);
3 int main()
4 {
5     int n;
6     printf("Que = Find factorial of a given number using Recursion: \n \n");
7     printf("Enter a positive integer: ");
8     scanf("%d",&n);
9     printf("Factorial of %d = %ld \n", n, factorial(n));
10    return 0;
11 }
12
13 factorial(int n)
14 {
15     int fact;
16     if(n==1)
17         return(1);
18     else
19         fact = n*factorial(n-1);
20     return(fact);
21 }
22

```

The execution output shows the program running successfully. It prompts the user to enter a positive integer, and when 5 is entered, it outputs the factorial of 5, which is 120. The process returned 0 (0x0) and the execution time was 1.613 s.

Build messages:

```

=== Build file: "no target" in "no project" (compiler: unknown) ===
warning: return type defaults to 'int' [-Wimplicit-int]
=== Build finished: 0 error(s), 1 warning(s) (0 minute(s), 0 second(s))

```

QUE-2

WRITE A PROGRAM IN C TO PRINT EVEN OR ODD NUMBERS IN GIVEN RANGE USING RECURSION. EXPLAIN YOUR IMPLEMENTATION IN DETAIL.

First we declare a function named EvenOdd with two arguments start and stop. Then in the main we introduce a variable named upperLimit. Now we print the que, then print that the range starts from 1. Then we ask the user to enter a number, this value is stored in variable upperLimit. Suppose that user entered 10. This means our range is 1 to 10. Next we print Even nos. in the range. The control is shifted to the function defined below. Here we have EvenOdd(2,10). As $2 > 10$ is false, 2 is printed and the value of start is incremented by 2. Now we have EvenOdd(4,10). This process goes on till value of start becomes greater than 10. For the next print statement we have EvenOdd(1,10). As $1 > 10$ is false, 1 is printed and the value of start is incremented by 2. Now we have EvenOdd(3,10). This process goes on till value of start becomes greater than 10 and hence the desired output is obtained. It is to be noted that odd nos are of form $2n+1$ and even nos are of $2n$ form. When the range starts from one, the first odd and even number is 1 and 2 respectively. Hence the functions EvenOdd(1,upperLimit) and EvenOdd(2,upperLimit) are used.

```

1 #include<stdio.h>
2
3 void EvenOdd(int start, int stop);
4 int main()
5 {
6     int upperLimit;
7     printf("Que = Print Even and odd numbers in a given range using Recursion:\n\n");
8
9     printf("Range starts from: 1\n");
10    printf("Range ends at: ");
11    scanf("%d", &upperLimit);
12
13    printf("Even Numbers from 1 to %d are: ", upperLimit);
14    EvenOdd(2, upperLimit);
15    printf("\n");
16    printf("Odd Numbers from 1 to %d are: ", upperLimit);
17    EvenOdd(1, upperLimit);
18
19    return 0;
20 }
21
22 void EvenOdd(int start, int stop)
23 {
24     if(start > stop)
25         return;
26
27     printf("%d, ", start);
28     EvenOdd(start+ 2, stop);
29 }
30

```

Output:

```

Que = Print Even and odd numbers in a given range using Recursion:

Range starts from: 1
Range ends at: 10
Even Numbers from 1 to 10 are: 2, 4, 6, 8, 10,
Odd Numbers from 1 to 10 are: 1, 3, 5, 7, 9,
Process returned 0 (0x0)   execution time : 1.694 s
Press any key to continue.

```

Build Log:

```

--- Build file: "no target" in "no project" (compiler: unknown) ---
--- Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s))

```

QUE - 3

WRITE A PROGRAM IN C TO DISPLAY N TERMS OF NATURAL NUMBER AND THEIR SUM. EXPLAIN YOUR IMPLEMENTATION IN DETAIL.

```

1 #include<stdio.h>
2
3 int main()
4 {
5     printf(" Que = Write a program to display n terms of natural number and their sum \n\n");
6
7     int i, j, sum = 0;
8     printf(" Enter a number: ");
9     scanf("%d", &j);
10
11     printf(" The first %d natural numbers are : \n", j );
12     for( i=1 ; i<=j ; i++)
13     {
14         printf(" %d ", i);
15         sum = sum + i;
16     }
17     printf("\n \n");
18     printf(" Sum of first %d natural numbers is = %d \n", j, sum);
19     return 0;
20 }

```

Output:

```

Que = Write a program to display n terms of natural number and their sum

Enter a number: 8
The first 8 natural numbers are :
1 2 3 4 5 6 7 8

Sum of first 8 natural numbers is = 36

Process returned 0 (0x0)   execution time : 1.724 s
Press any key to continue.

```

First three variables i , j and sum are taken. The value of sum is initialized as 0. Next we ask the user to enter a number, we store the entered value in j . Suppose that the user entered 8. We therefore print the first 8 natural numbers. We need to display the first n natural nos and the sum of these natural numbers.

To do so we take the help of a for loop. Here $i=1, j=8$ as $i < j$ we print i a, i.e. 1 and add the value of i to sum . The value of i is now incremented by 1. This for loop goes on till i becomes = to j . After this happens we jump out of the loop and print the value of sum . And hence the desired output is obtained.

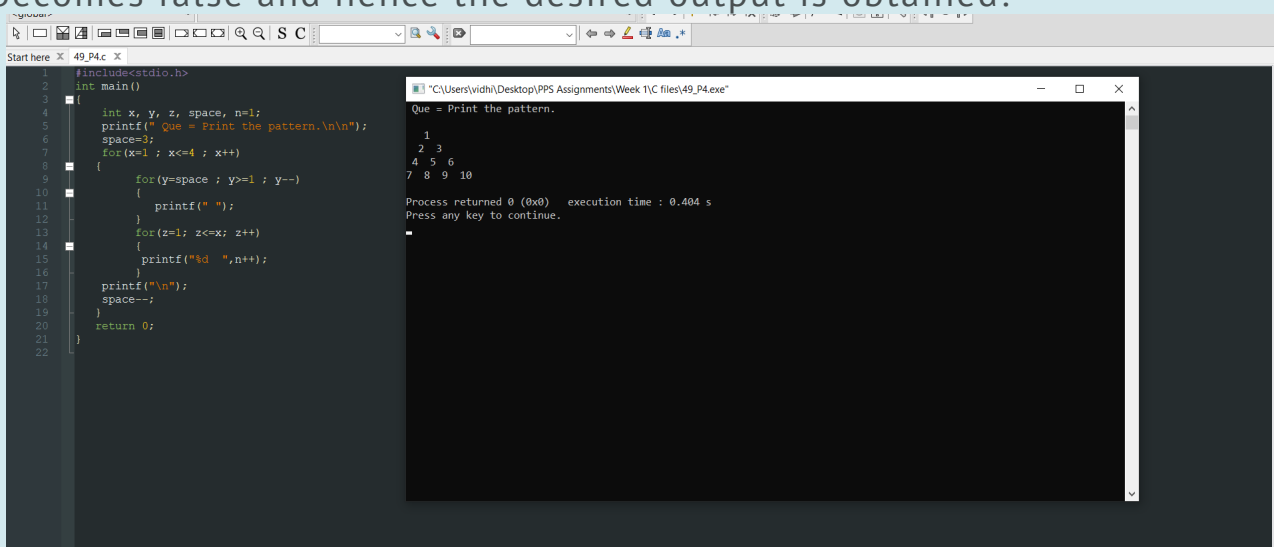
QUE - 4

04

WRITE A PROGRAM IN C TO MAKE SUCH A PATTERN LIKE A PYRAMID WITH NUMBERS INCREASED BY 1. EXPLAIN YOUR IMPLEMENTATION IN DETAIL

```
1
2 3
4 5 6
7 8 9 10
```

To print any program with patterns we must ask ourselves three questions. How many lines do we need to print? How many characters? What to print? To get the desired output here we use a for loop. Here we take 5 variables: x,y,z, space and num.If we observe there are 3 spaces before 1 hence space is initialized as 3.Here x denotes rows.We now enter the for loop.As x=1,1<4, we move to the inner loop.Here y=3 as y>=1 is true, a space is printed and the value of y is decremented by 1.Again y=2,y>=1 is true the loop is executed again this goes on till y>=1 becomes false.Then we enter the next for loop.As the condition is true n is printed and then n is increased by 1.Now the condition becomes false as 2<=1 is not true we jump out of the loop, go to next line and value of space is decreased by 1.Now space becomes=2 and the above procedure repeats till the condition of first for loop becomes false and hence the desired output is obtained!



The screenshot shows a C program in a code editor and its execution output in a separate window. The code defines a function `main` that prints a pyramid pattern of numbers. It uses nested loops: an outer loop for rows (x) and an inner loop for spaces (y) and numbers (z). The pattern is printed row by row, with spaces decreasing and numbers increasing by 1 in each row. The execution window shows the output of the program, which matches the pattern shown in the text above. The output is:

```
Que = Print the pattern.
1
2 3
4 5 6
7 8 9 10
```

Below the output, the execution details are shown: "Process returned 0 (0x0) execution time : 0.404 s" and "Press any key to continue."

QUE - 5

WRITE A PROGRAM IN C TO DISPLAY THE N TERMS OF HARMONIC SERIES AND THEIR SUM.

$1 + 1/2 + 1/3 + 1/4 + 1/5 \dots 1/N$ TERMS.

EXPLAIN YOUR IMPLEMENTATION IN DETAIL.

Two variables i and j are taken with datatype `int` and one variable `sum` is taken with datatype `float`. We ask the user to enter a number and we store it in j . Let the number entered by the user be 5. As per the question we need to display the first 5 terms and their sum in harmonic series for this eg. To do so we take the help of a for loop.

In for loop, $i=1, j=5$, as $i < j$ we print i and add value of i in `sum`. Now the value of i is incremented by 1. Which means that now $i=2$ and we enter the loop again. This goes on till $i=5$ after which we jump out of the loop and print the value of `sum`. It is to be noted that as per the formula of harmonic series the sum may not be a perfect integer value always hence we have used the datatype `float` for `sum`.

The screenshot shows a C program in a code editor and its execution output in a terminal window.

Code Editor (C Program):

```

1  #include<stdio.h>
2  void main()
3  {
4      printf("Que = Display the n terms of harmonic series and their sum \n\n");
5
6      int i, j;
7      float sum = 0.0;
8      printf("Enter a number: ");
9      scanf("%d", &j);
10
11     printf("The first %d terms are : \n", j);
12
13     for( i=1; i<=j; i++)
14     {
15         printf("1/%d + ", i);
16         sum = sum + 1.0/i;
17     }
18
19     printf("\n\n");
20     printf("Sum of first %d terms is = %f \n", j, sum);
21     return 0;
22 }

```

Terminal Output:

```

C:\Users\vidhi\Desktop\PPS Assignments\Week 1\49_P5.exe
Que = Display the n terms of harmonic series and their sum

Enter a number: 5
The first 5 terms are :
1/1 + 1/2 + 1/3 + 1/4 + 1/5 +

Sum of first 5 terms is = 2.283334

Process returned 37 (0x25)   execution time : 1.671 s
Press any key to continue.

```

The terminal output shows the program's execution for $n=5$, displaying the first 5 terms of the harmonic series and their sum as 2.283334.

QUE - 6

WRITE A PROGRAM IN C TO PRINT NTH TERM FIBONACCI SERIES USING RECURSION AND ITERATIVE APPROACH. WHICH ONE IS MORE EFFICIENT? EXPLAIN IN DETAIL WITH AN APPROPRIATE EXAMPLE.

Fibonacci series is the series in which each number is the sum of the two preceding numbers. Here we have declared a function named fibo. Then we ask the user to enter a number. We need to show the value of that term in Fibonacci series.

#Recursive Approach:

If the no. is -ve we print that it is not possible to print the nth term, else we call the fibo function and print the required value. Let the value entered by user be 3. As 3 is +ve we enter the else part and then fibo function is called. We now have fibo(3). As $3 \neq 0$ or 1 , the statement $\text{fibo}(n-1) + \text{fibo}(n-2)$ is executed. In our eg it will be $\text{fibo}(3) = \text{fibo}(2) + \text{fibo}(1)$. Here to solve fibo(2), fibo is again called. $\text{fibo}(2) = \text{fibo}(1) + \text{fibo}(0)$. The answer of this is $1 + 0 = 1$. Now the ans of $\text{fibo}(3) = \text{fibo}(2) + \text{fibo}(1)$ which is equal to $1 + 1 = 2$. Now this value is the ans of fibo(3). As answer = fibo(3) we get the final result printed on the screen as 2 in our example.

The screenshot shows a C program in a code editor (49_P6.c) and its execution output in a terminal window. The program implements two methods to calculate the nth term of the Fibonacci series: a recursive approach and an iterative approach. The recursive approach is commented out, and the iterative approach is active. The user enters 3 as the nth term, and the program outputs 2.

```

1 #include <stdio.h>
2 int fibo(int);
3 int main()
4 {
5     int n;
6     int answer;
7     // using recursive approach
8     printf("Display the nth term of fibonacci series\n\n");
9     printf("Enter the nth number in fibonacci series: ");
10    scanf("%d", &n);
11    if (n < 0)
12        (printf("Fibonacci of negative number is not possible.\n"));
13    else
14    {
15        answer = fibo(n);
16        printf("The number in fibonacci series is %d\n", answer);
17    }
18    return 0;
19 }
20 // using iterative approach
21 (int n, a = 0, b = 1, c = 0, i;
22 printf("Enter the nth term: ");
23 scanf("%d", &n);
24 if (n == 0 || n == 1)
25     printf("The number in fibonacci series is %d", n);
26 else
27 {
28     c = a + b;
29     for (i = 3; i <= n; ++i)
30         (a = b; b = c; c = a + b);
31     printf("The number in fibonacci series is %d", b);
32 }
33 int fibo(int n)
34 {
35     if (n == 0 || n == 1)
36         return n;
37     return (fibo(n-1) + fibo(n-2));
38 }

```

Execution Output:

```

"C:\Users\vidhi\Desktop\PPS Assignments\Week 1\49_P6.exe"
Display the nth term of fibonacci series
Enter the nth number in fibonacci series: 3
The number in fibonacci series is 2
Process returned 0 (0x0)   execution time : 1.559 s
Press any key to continue.

```

continued on next page...

QUE - 6 CONTINUED...

#Iterative approach:

Iteration simply means something which happens again and again. Here we have taken 5 variables n, a, b, c and i. C denotes the current term, b denotes the previous term and a denotes the pre previous term. Say for example the fibo series is 1,1,2,3,5... The current term is 3 which means that $c=3$, $b=2$, $a=1$. Current term c is always the sum of its previous two terms i.e. a and b. In our program we ask the user to enter a number. Our goal is to display that particular term. In our program we first ask the user to enter a value. Let us assume that the value entered by the user is 3. This value is stored in n. Now we move to the next statements. As $3 \neq 0$ or 1, we enter the else part. Here we have used a for loop. We have initialised the value of i here as 3. In our example as $i=n$, the following block of statements are executed and the final result is printed.

Recursion vs Iteration:

In the process of Recursion, a function is called again and again whereas iteration is implemented using loops. It becomes easier to write the code using recursion. Also the size of the code is reduced in recursion; but as the function is called again and again its execution takes longer time. Hence whenever time and space complexity are involved / concerned, Iteration is preferred over Recursion.

The background is a light teal color with a delicate line-art illustration of various flowers and leaves. The flowers are stylized with simple outlines and some internal detailing. A large, solid white circle is centered on the page, serving as a backdrop for the text.

THANK
YOU
