

Unit #: 2

Unit Name: Counting, Sorting and Searching

Topic: Recursion

Course objectives:

The objective(s) of this course is to make students

CObj1: Acquire knowledge on how to solve relevant and logical problems using computing machine

CObj2: Map algorithmic solutions to relevant features of C programming language constructs

CObj3: Gain knowledge about C constructs and it's associated eco-system

CObj4: Appreciate and gain knowledge about the issues with C Standards and it's respective behaviors

CObj5: Get insights about testing and debugging C Programs

Course outcomes:

At the end of the course, the student will be able to

CO1: Understand and apply algorithmic solutions to counting problems using appropriate C Constructs

CO2: Understand, analyse and apply text processing and string manipulation methods using C Arrays, Pointers and functions

CO3: Understand prioritized scheduling and implement the same using C structures

CO4: Understand and apply sorting techniques using advanced C constructs

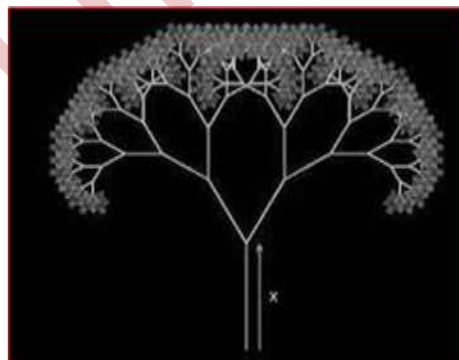
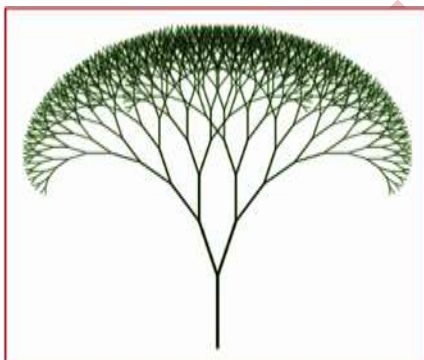
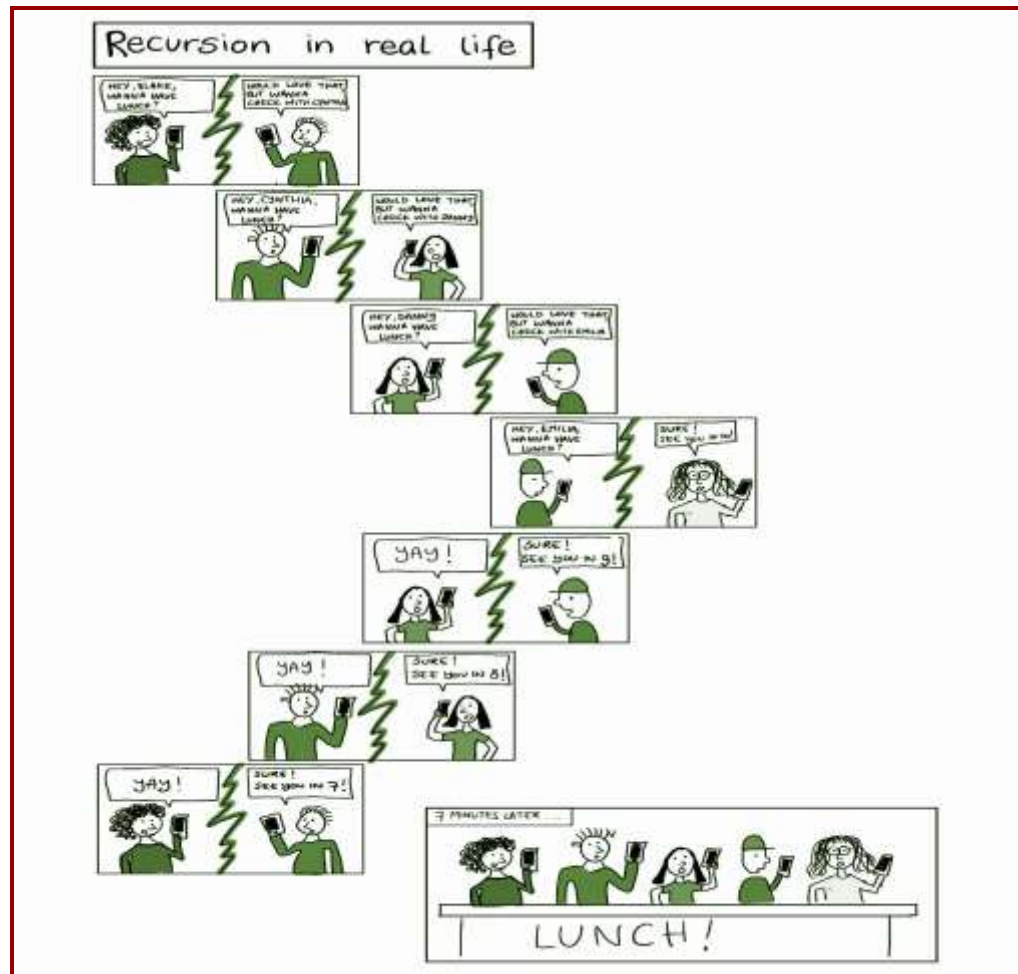
CO5: Understand and evaluate portable programming techniques using preprocessor directives and conditional compilation of C Programs

Team – PSWC,

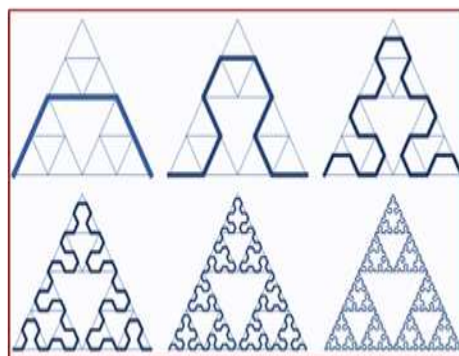
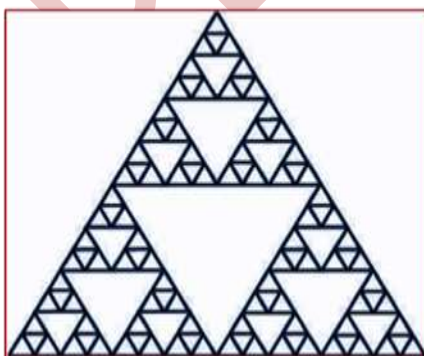
Jan - May, 2022

Dept. of CSE,

PES University



Fractals



The Sierpinski Triangle

Recursion

A function may call itself directly or indirectly. **The function calling itself with the termination/stop/base condition is known as recursion.** Recursion is used to solve various problems by dividing it into smaller problems. We can **opt if and recursion instead of looping statements.** In recursion, we try to **express the solution to a problem in terms of the problem itself, but of a smaller size.**

Coding Example_1:

```
int main()
{
    printf("Hello everyone\n");
    main();
    return 0;
}
```

Execution starts from main() function. Hello everyone gets printed. Then call to main() function. Again, Hello everyone gets printed and these repeats. We have not defined any condition for the program to exit, results in Infinite Recursion. In order to **prevent infinite recursive calls, we need to define proper base condition in a recursive function.**

Let us write Iterative and Recursive functions to find the Factorial of a given number.

Coding Example_2:

```
int fact(int n);
int main()
{
    int n = 6;
    printf("factorial of %d is %d\n",fact(n));
    return 0;
}
```

Iterative Implementation:

```
int fact(int n)
{
    int result = 1;
    int i;
```

```
for (i = 1; i<=n; i++)  
{  
    result = result * i;  
}  
return result;  
}
```

Recursive Implementation:

Logic: If n is 0 or n is 1, result is 1 Else, result is $n*(n-1)!$

```
int fact(int n)  
{  
    if (n==0)  
        return 1;  
    else  
    {  
        return n*fact(n-1);  
    }  
}
```

Pictorial representation: When n is 5

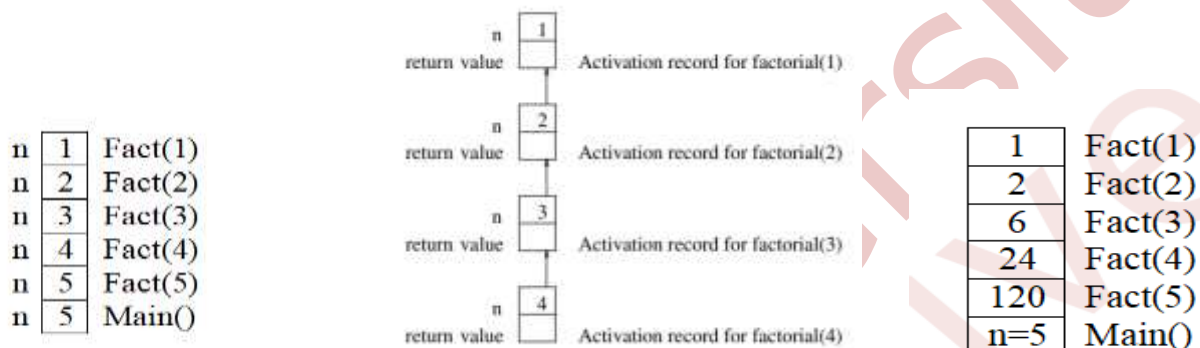
```
=factorial(5)  
=5*factorial(4)  
=5*4*factorial(3)  
=5*4*3*factorial(2)  
=5*4*3*2*factorial(1)  
=5*4*3*2*1*factorial(0)  
=120
```

Output: When n is 6

```
factorial of 6 is 720  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Activation Record

- Each function call generates an instance of that function containing memory for each parameter, memory for each local variable and memory for return values. This chunk of memory is called as an **activation record**.
- To support recursive function calls, the run-time system treats memory as a stack of activation record.
- For example computing the factorial (n) requires allocation of 'n' activation records on the stack.



Let us try to solve below examples to understand recursive calls better.

Coding Example_3: What this code does? Write activation frame for each call and then decide

```
#include<stdio.h>
int what1(int n)
{
    if(n==0) return 0;
    else return (n%2)+10*what1(n/2);
}
int main()
{
    printf("%d",what1(8));
    return 0;
}
```

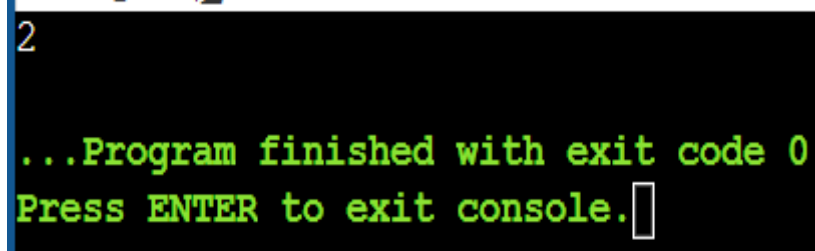
Output: This code prints the binary representation of a given number

```
1000
...Program finished with exit code 0
Press ENTER to exit console.
```

Coding Example_4: What this code does?

```
#include<stdio.h>
int what2_v1(int n);
int what2_v2(int n);
int main()
{
    int n = 10;
    printf("%d",what2_v1(n));
    //printf("%d",what2_v2(n));
    return 0;
}
int what2_v2(int n)
{
    if(!n) return 0;
    else if (!(n%2)) return what2_v2(n/2);
    else
        return 1+what2_v2(n/2);
}
int what2_v1(int n)
{
    if(n==0) return 0;
    else return (n&1)+ what2_v1(n>>1);
}
```

Output: Finds the number of 1s in the binary representation of the number



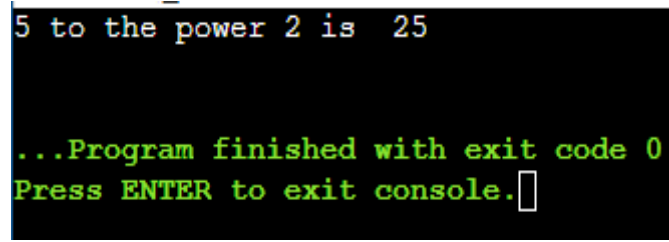
```
2
...Program finished with exit code 0
Press ENTER to exit console.
```

Coding Example_4: *What this code does?*

```
#include <stdio.h>
int what3_v1(int,int);
int main()
{   int a=5,b=2;
    printf("%d to the power %d is  %d\n",a,b,what3_v1(a,b));
    //printf("%d to the power %d is  %d\n",a,b,what3_v2(a,b));
    return 0;
}
int what3_v1(int b,int p)
{   int result=1;
    if(p==0)
        return result;
    else
        result=b*(what3_v1(b,p-1));
}
int what3_v2(int b, int p)
{
    if(!p)
        return 1;
    else if(!(p % 2))
        return what3_v2(b*b, p/2);
    else
```

```
return b*what3_v2(b*b,p/2);  
}
```

Output: Finds a to the power b



```
5 to the power 2 is 25  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Coding Example_5: Last example for today!!

```
#include <stdio.h>  
  
int what4(int);  
  
int main()  
{  
    int a=5;  
    printf("the sum of numbers upto %d is %d\n",a,what4(a));  
    return 0;  
}  
  
int what4(int n)  
{  
    if (n>0)  
        return n + what4(n - 1);  
    else if (n==0)  
        return 0;  
    else  
        return -1;  
}
```


Output: If the given number is n, Finds $n+(n-1)+(n-2)+ \dots +0$ If n is -ve, returns -1.

```
the sum of numbers upto 5 is 15  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Few problems for you to Code:

- Recursive function to reverse a string
- Recursive function to print from 1 to n in reverse order
- Find the addition, subtraction and multiplication of two numbers using recursion. Write separate recursive functions to perform these.
- Find all combinations of words formed from Mobile Keypad.
- Find the given string is palindrome or not using Recursion.
- Find all permutations of a given string using Recursion