Recursion

Department of Computer Science,
National University of Computer & Emerging Sciences,
Islamabad Campus

Introduction to Recursion

- A **recursive function** is one that **calls itself**.

```
void Message(void)
{
      cout << "This is a recursive function.\n";
      Message();
}
```

The above **function displays** the **string** "This is a recursive function.\n", and then **calls itself**.

Can you see a problem with this function?

Recursion

- The **function** is **like an infinite loop** because **there is no code to stop it** from **repeating**.

- **Like a loop**, a **recursive function must have some algorithm to control** the **number of times it repeats**.

Recursion : Using Control Condition

```
void Message(int times)
{
        if (times > 0)
        {
                cout << "This is a recursive function.\n";
                Message(times - 1);
        }
        return;
}
```

The function contains an **`if/else` statement** that **controls** the **repetition**. For example, if we call the function:

```
Message(5);
```

The **argument, 5**, will cause the function **to call itself 5 times**.

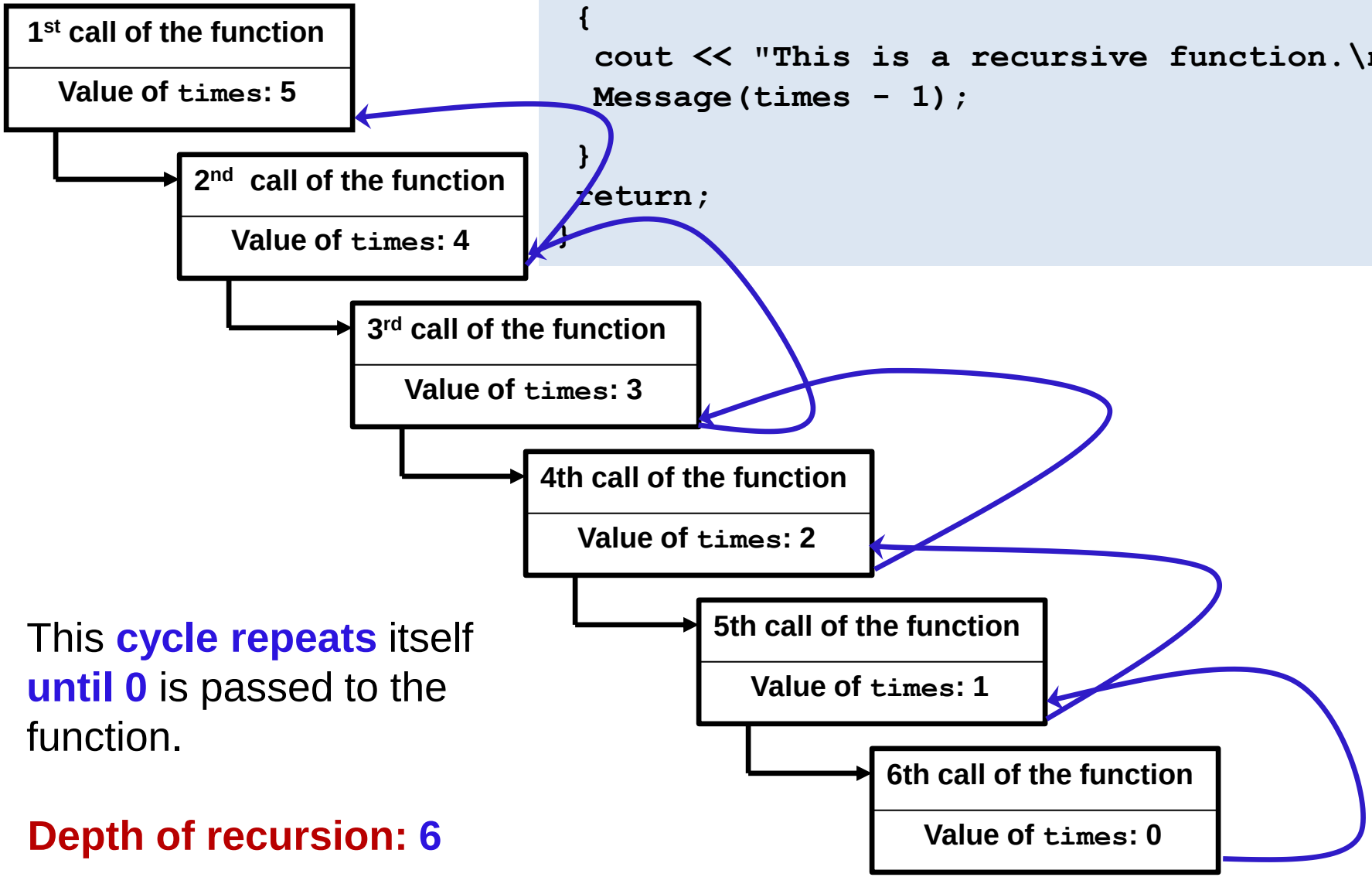
Recursion : Using Control Condition

```
void Message(int times)
{
        if (times > 0) // Base case
        {
                cout << "This is a recursive function.\n";
                Message(times - 1);
        }
        return;
}
```

- With **each** **recursive call**, the **parameter controlling the recursion** should **move closer** to the **base case**

- Eventually, the **parameter reaches** the **base case** and the chain of **recursive calls terminates**

```
void Message(int times)
{
 if (times > 0)
 {
  cout << "This is a recursive function.\n";
  Message(times - 1);

 }
 return;
}
```



This **cycle repeats** itself **until 0** is passed to the function.

Depth of recursion: 6

Program Output

```
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
```

What Happens When Called?

- Each time a **recursive function** is **called**, a **new copy** of the **function runs**, with **new instances of parameters** and **local variables** being **created**

- As **each copy finishes executing**, it **returns** to the copy of the **function** that **called it**

- When the **initial copy finishes executing**, it returns to the **part of the program** that **made the initial call** to the **function**

Types of Recursion

- **Direct recursion**
  - a **function calls itself**

- **Indirect recursion**
  - **function A calls function B**, and function B **calls function A**. Or,

  - **function A calls function B**, which calls …, which calls **function A**

Recursive Function

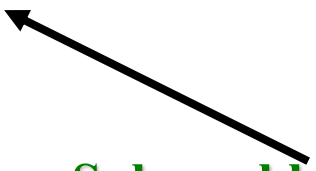
```
#include <iostream>
using namespace std;

void message(int);
int main() {
   message(5);
   return 0;
}

//*********************************************************
// Definition of function message. If the value in times is
// greater than 0, the message is displayed and the functior
// is recursively called with the argument times - 1.
//*********************************************************
void message(int times)
{  cout << "message called with " << times
        << " in times.\n";
   if (times > 0)
   {
      cout << "This is a recursive function.\n";
      message(times - 1);
   }

   cout << "message returning with " << times;
   cout << " in times.\n";
   return;
}
```

```
message called with 5 in times.
This is a recursive function.
message called with 4 in times.
This is a recursive function.
message called with 3 in times.
This is a recursive function.
message called with 2 in times.
This is a recursive function.
message called with 1 in times.
This is a recursive function.
message called with 0 in times.
message returning with 0 in times.
message returning with 1 in times.
message returning with 2 in times.
message returning with 3 in times.
message returning with 4 in times.
message returning with 5 in times.
```

Recursion

To build *all* recursive functions:

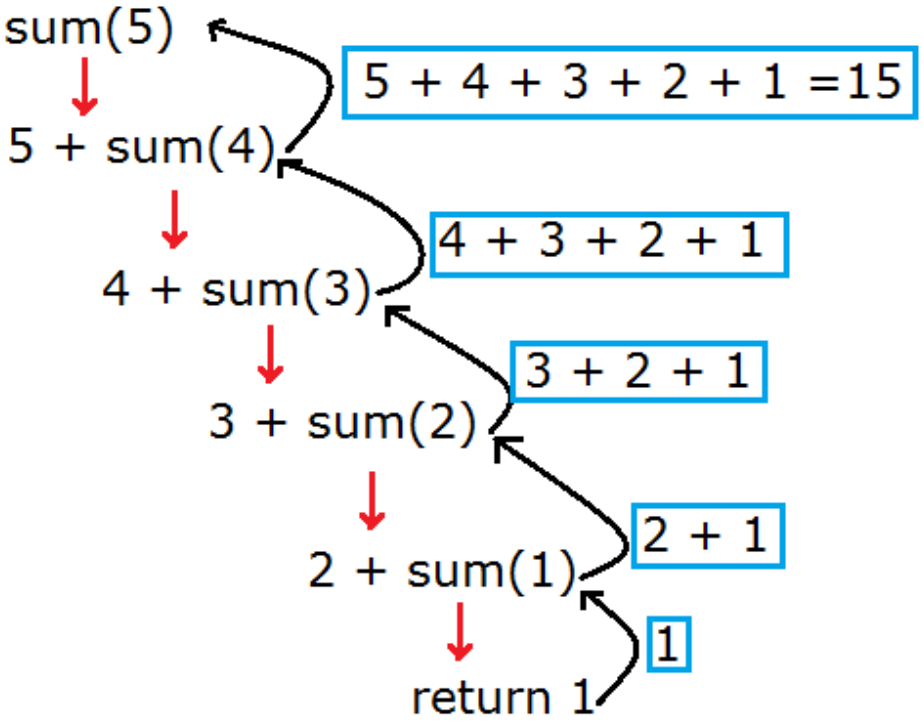
1. Define the base case(s)
2. Define the recursive case(s)
   a) Divide the problem into smaller sub-problems
   b) Solve the sub-problems
   c) Combine results to get answer



Sub-problems solved as a recursive call to the same function

Creating a Sum Function

- sum(10) = 10+9+…2+1 = 55

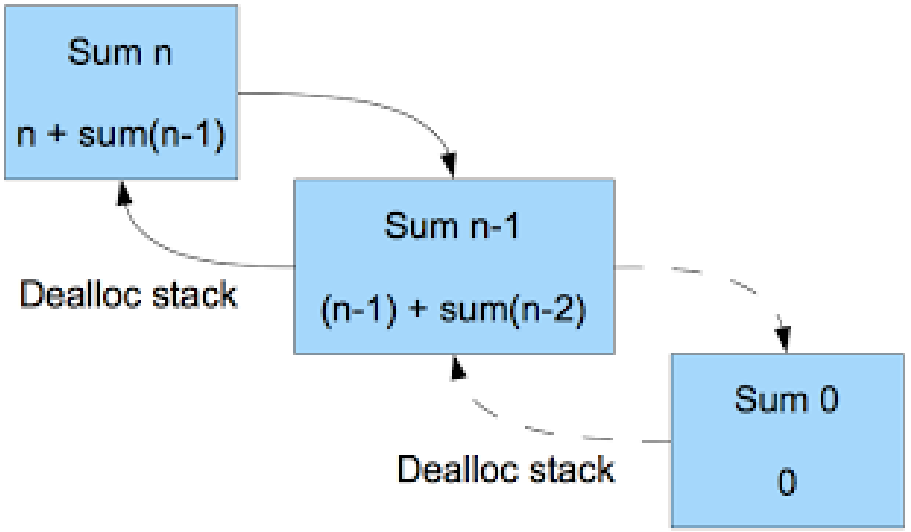


Creating a Sum function (Iterative)

```
//Our initial total is zero
int total = 0;


//We want the sum from 1 + 2 + ... + 9 + 10
int n = 10;


/* The following for loop will calculate the summation
from 1 - n */
for ( int i = 1; i <= n; i++ ) {
      total = total + i;
}
```

Creating a Sum function (Recursive)

```
int sum(int n) {

      //Return 0 when n is 0
      if ( n <= 0 )
            return 0;
      else //recursive call
            return n + sum(n-1);
}
```



The Recursive Factorial Function

- The **factorial** of a **non-negative** integer n is the ***product of all positive integers*** less or equal to n

- Factorial of n is denoted by $n!$

- The factorial of **0** is= **1**

$$0 ! = 1$$

$$n ! = n \times (n-1) \times \ldots \times 2 \times 1 \quad \text{if} \quad n > 0$$

The Recursive Factorial Function

- **Factorial** of n **can be expressed** in terms of the **factorial of** n**-1**

$$0 ! = 1$$

$$n ! = n \times (n-1) !$$

- The **base case** is $n = 0$

- **Recursive function:**

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

Character count - Recursive

```
#include <iostream>
using namespace std;

// Function prototype
int numChars(char, char [], int);

int main()
{
   char array[] = "abcddddef";

   /* Display the number of times the letter
   'd' appears in the string. */

   cout << "The letter d appears "
   << numChars('d', array, 0) << " times.\n";

   return 0;
}
```

```
int numChars(char search, char str[], int subscript)
{
 if (str[subscript] == '\0')
 {
     // Base case: The end of the string is reached.
        return 0;
 }
 else if (str[subscript] == search)
 {
    /* Recursive case: A matching character was found.
  Return 1 plus the number of times the search character
  appears in the rest of the string.*/
       return 1 + numChars(search, str,subscript+1);
 }
 else
 {
    /* Recursive case: A character that does not match the
  search character was found. Return the number of times
  the search character appears in the rest of the string.
  */
       return 0+ numChars(search, str, subscript+1);
 }
}
```

Printing a Sequence of Numbers in Reverse

```
void print(int n) {

    if ( n <= 0 )
      return; //Base condition

     cout << n << " "; //Prints number n
     print(n-1); //Calls itself with (n-1)
     return; //Returns from the function
}
```

print(3) produces → 3 2 1

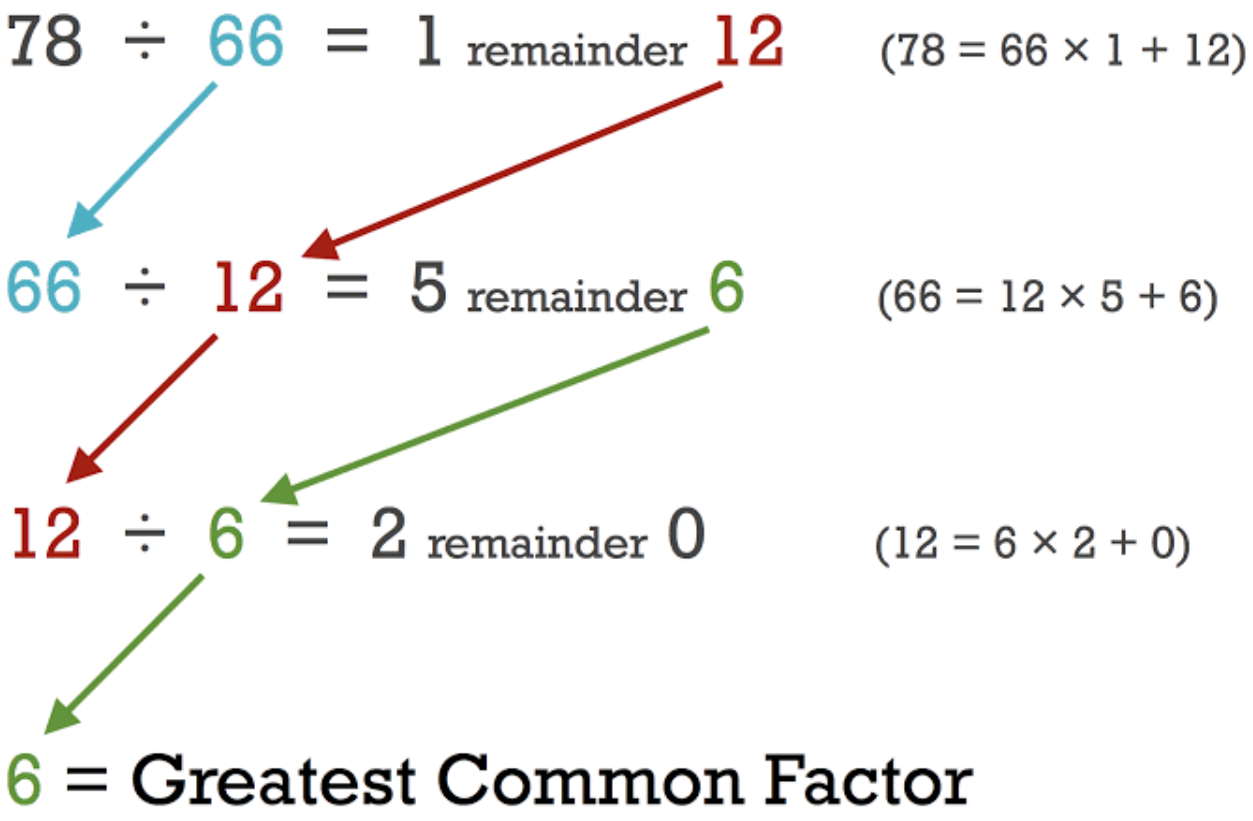
Printing a Sequence of Numbers in Ascending Order

Example:

Input Number: 5

Output: 1 2 3 4 5

Finding gcd


78 ÷ 66 = 1 remainder 12
(78 = 66 × 1 + 12)
66 ÷ 12 = 5 remainder 6
(66 = 12 × 5 + 6)
12 ÷ 6 = 2 remainder 0
(12 = 6 × 2 + 0)
6 = Greatest Common Factor

The Recursive gcd Function

```
int gcd(int x, int y)
{
    if (x % y == 0) //base case
       return y;
    else
       return gcd(y, x % y);
}
```

Solving Recursively Defined Problems

- The **natural definition** of some **problems** **leads to a recursive solution**

- Example: **Fibonacci numbers**:
  `1, 1, 2, 3, 5, 8, 13, 21, ...`

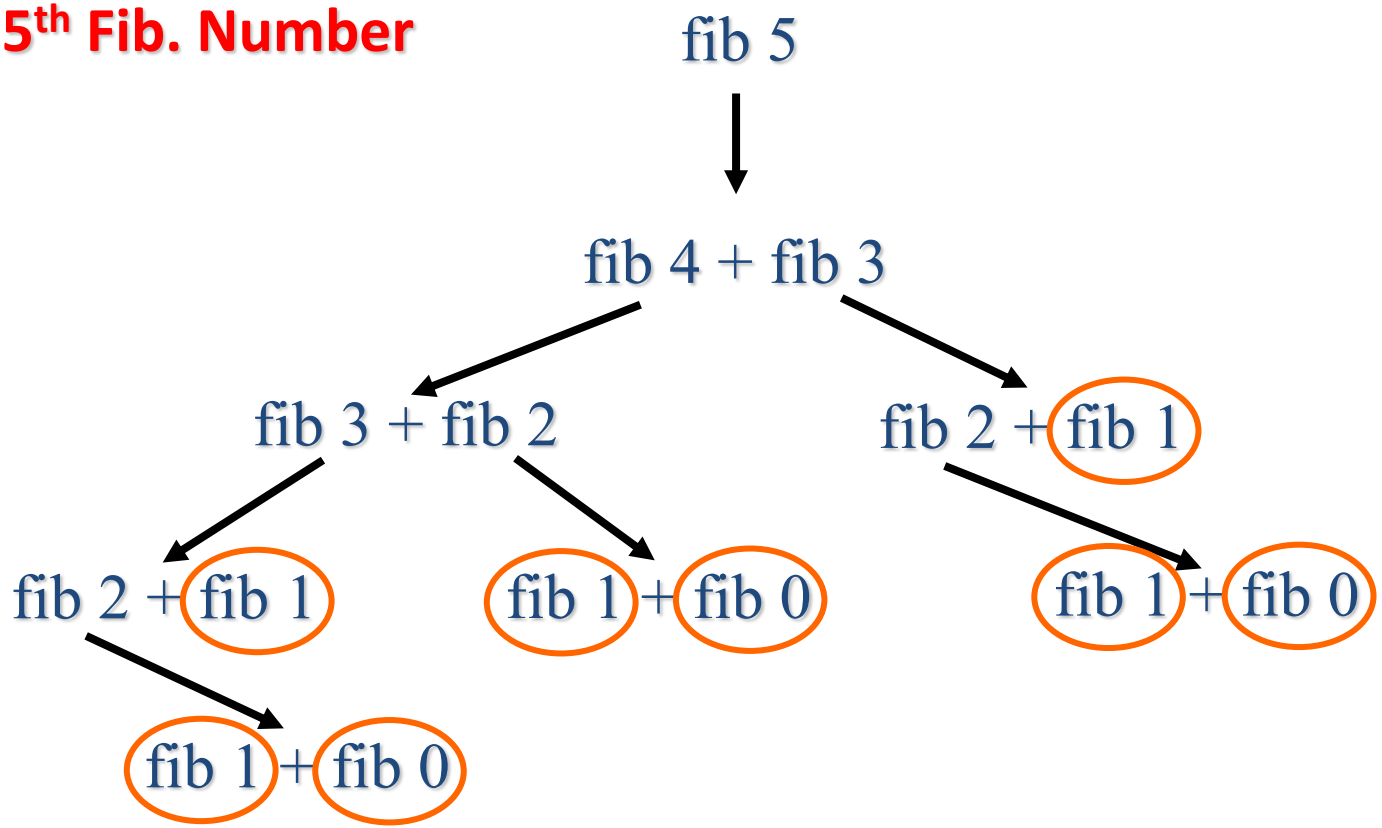
- After the **starting two numbers**, **each term** is the **sum** of the **two preceding terms**

- **Recursive solution:**
  ```
  fib(n) = fib(n - 1) + fib(n - 2);
  ```

- **Base cases**: `n == 0, n == 1`

Recursion

5th Fib. Number



= 5

Recursive Fibonacci Function

```
#include <iostream>
using namespace std;

int fib(int n) {
     if (n <= 0)          // base case
       return  0;
     else if (n==1)     // base case
       return 1;
     else
       return fib(n-1) + fib(n-2);
}

int main() {
    int n;
    cin>>n;
    cout<<n<<"th Fibonacci number is: "<<fib(n);
    return 0;
}
```

Printing Patterns using Recursion

```
Input : n = 5
Output :
* * * * *
* * * *
* * *
* *
*


Input : n = 7
Output :
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

Printing Patterns using Recursion

Code credits: https://www.geeksforgeeks.org/

```
void printPatternRowRecur(int n)
{
    // base condition
    if (n < 1)
        return;

    // print the remnaining stars of the n-th row recursively
    cout << "* ";
    printPatternRowRecur(n-1);
}
```

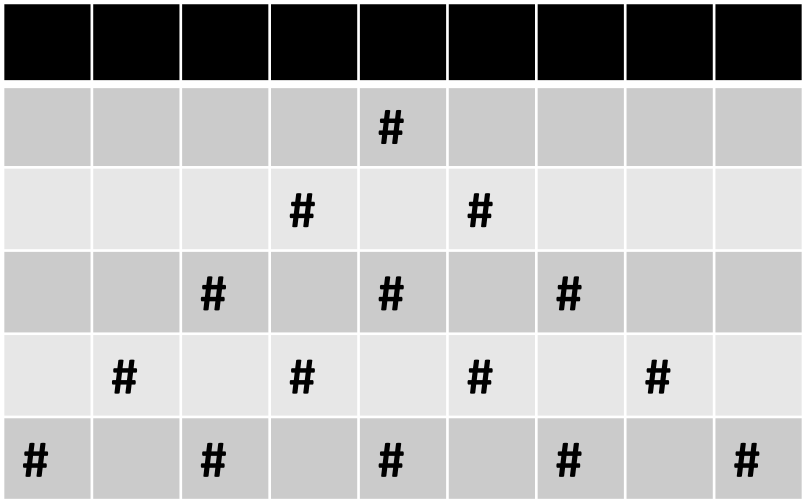
Printing Patterns using Recursion

Input:

Draw a Pyramid of size: 5

Output:

```
    #
   # #
  # # #
 # # # #
# # # # #
```



Printing Patterns using Recursion

Code credits: https://www.tutorialspoint.com/

```
int main(){
   int n = 5;
   Pyramid(n, n);
   return 0;
}
```

```
// function to print spaces
void print_space(int space){
   if (space == 0)
      return;
   cout << " ";
   print_space(space - 1);
}
```

```
// function to print the pattern
void Pyramid(int n, int num){
   // base case
   if (n == 0)
      return;
   print_space(n - 1);
   print_hash(num - n + 1);
   cout << endl;
   // recursively calling pattern()
   Pyramid(n - 1, num);
}
```

```
// function to print hash
void print_hash(int pat){
   if (pat == 0)
      return;
   cout << "# ";
   // recursively calling hash()
   print_hash(pat - 1);
}
```

Recursion VS. Iteration

- **Benefits (+), disadvantages(-) of Recursive Implementation**
  - + **Natural formulation** of **solution** to **certain problems**
  - + **Results** in **shorter**, **simpler functions**
  - – **May not execute very efficiently**

- **Benefits (+), disadvantages(-) for iterative Implementation**
  - + **Executes** more **efficiently** **than recursion**
  - – **May not be as natural as recursion for some problems**