# Module 14 – part 1 – Recursion


BITS Pilani
Pilani Campus

Dr. Jagat Sesh Challa
Department of Computer Science & Information Systems

# Module Overview

innovate achieve lead

- Recursion
- Recursive Data
- Recursive Functions
- Examples
- Merge Sort

innovate achieve lead

BITS Pilani

Pilani Campus

# Recursion

# Recursion

- Recursion is the process of defining something in terms of itself, and is sometimes called *circular definition*.
- A recursive process is one in which objects are defined in terms of other objects of the same type.
- The entire class of objects can then be built up from a few initial values and a small number of rules.
- Ex: Fibonacci  Number series

innovate achieve lead

BITS Pilani

Pilani Campus

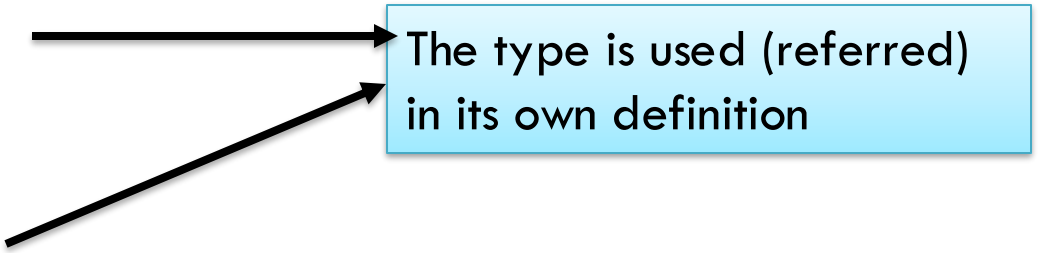
# Recursive Data

# Recursive Data



Consider the linked list representation of sequential access list:

```
typedef node NODE;
typedef NODE *LINK;
struct node {
   int elem;
   LINK next;
};
```



# Recursive Data

- Consider a high level description of a sequential access list:
  - *An empty list is a sequential access list;*
  - *Adding an element to a sequential access list produces another sequential access list*

- Such definitions are called inductive definitions or recursive definitions.

innovate achieve lead

BITS Pilani

Pilani Campus

# Recursive Functions

# Recursive functions

- *A recursive function is one which calls itself (repeatedly)*
- Recursive functions are useful in evaluating certain types of mathematical function. (we'll see examples)

Simple Example:

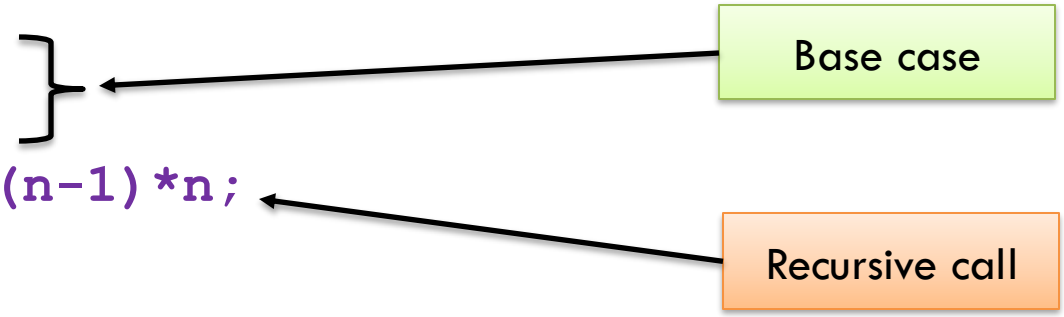
```
main()
{
  printf("This is an example of recursive function");
  main();
}
```

When this program is executed. The line is printed repeatedly and indefinitely. We might have to abruptly terminate the execution.

# Factorial Calculation

```
long int fact(int n)     /* non-recursive */
{
  int t, ans;
  ans = 1;
  for(t=1; t<=n; t++)
    ans = ans*t;
  return(ans);
}

long int factr(int n)  /* recursive */
{
  int ans;
  if(n==1)
      return(1);
  answer = factr(n-1)*n;
  return(ans);
}
```



# Power function (recursive)

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

Base case

Recursive call

# Summing up the elements of an array



Algorithm Design:

1. If we have only one element, then the sum is simple.
2. Otherwise, we use the sum of the first element and the sum of the rest.

Base case

Recursive call

```
int sum(int first,int last,int array[])
{   if (first == last)
        return (array[first]);
    return(array[first]+sum(first+1,last,array));
}
```

# Sum of array elements – execution sequence



For example:

Sum(1 8 3 2) = 1 + Sum(8 3 2) =

8 + Sum(3 2) =

3 + Sum (2)=

2

3 + 2 = 5

8 + 5 = 13

1 + 13 = 14

Answer = 14

# Recursive Version of Fibonacci Series



```
int fib(int num)     /*Fibonacci value of a number */
{
  switch(num)
  { case 0: return(0);
            break;
    case 1: return(1);
            break;
    default:          /*Including recursive calls */
            return(fib(num - 1) + fib(num - 2));
            break;
  }
}
```

Base cases

Recursive call

# Analysis

innovate achieve lead

| Input Value | Number of time fib is called |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 3 |
| 3 | 5 |
| 4 | 9 |
| 5 | 15 |
| 6 | 25 |
| 7 | 41 |
| 8 | 67 |
| 9 | 109 |

# How recursive functions works???



- When a function calls itself, a new set of local variables and parameters are allocated storage on the stack, and the function code is executed from the top with these new variables.

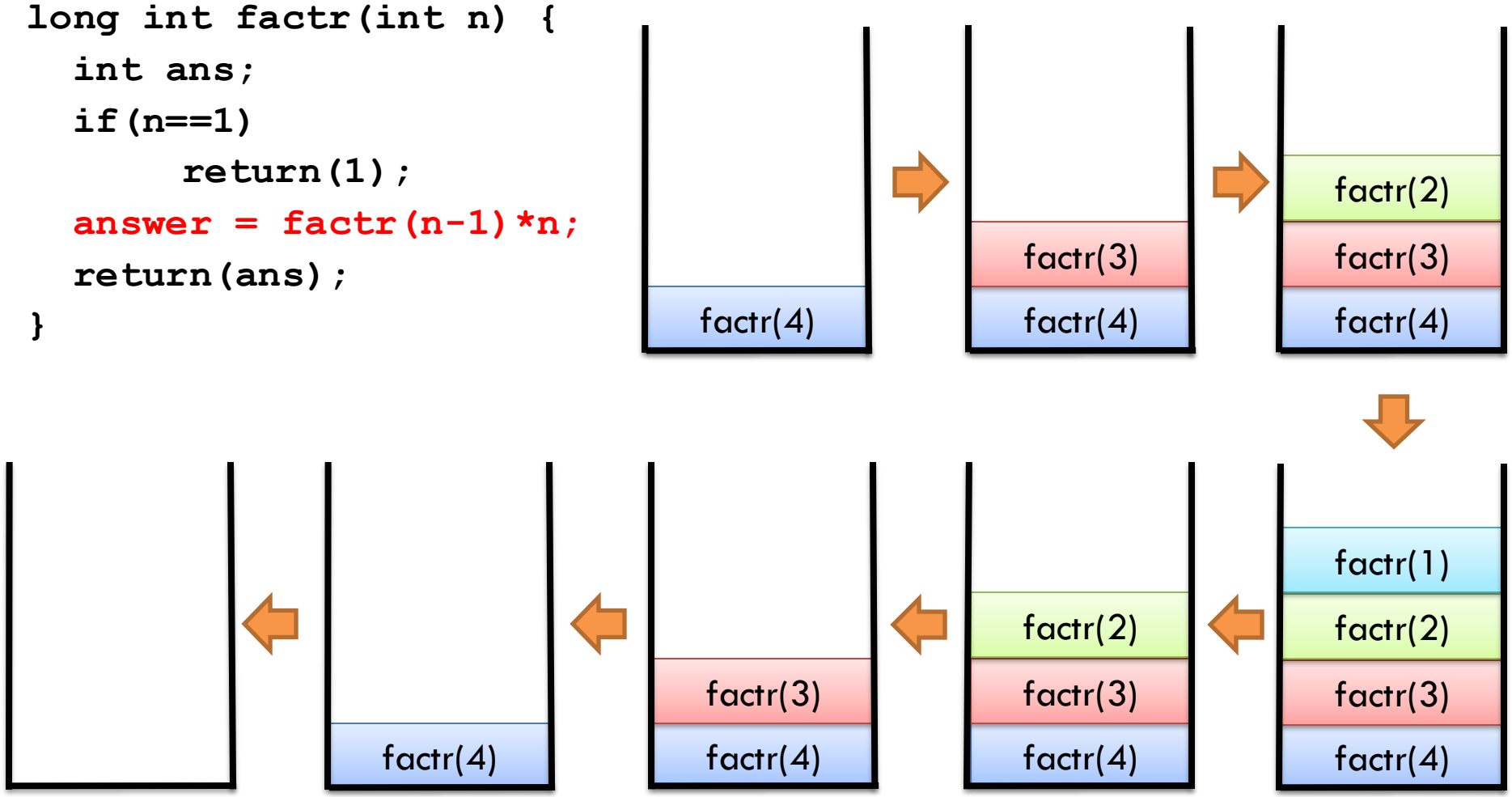
- A recursive call does not make a new copy of the function. Only the arguments are new.

- As each recursive call returns, the old local variables and parameters are removed from the stack and execution resumes at the point of the function call inside the function.

# Recursive Call Stack (Example)



```
long int factr(int n) {
   int ans;
   if(n==1)
        return(1);
   answer = factr(n-1)*n;
   return(ans);
}
```



# Another Example

```
int main()
{
    static int i=5;
    if (--i){
        printf("%d ",i);
        main();
    }
}
```

# Cons of recursion

- No significant reduction in code size as well as no improvement in memory utilization.
- Also, the recursive versions of most routines may execute a bit slower than their iterative equivalents because of the overhead of the repeated function calls.
- In fact, many recursive calls to a function could cause a stack overrun.

# Pros of recursion



- The main advantage to recursive functions is that you can use them to create clearer and simpler versions of several algorithms.

- For example, the *MergeSort* and the Quicksort are quite difficult to implement in an iterative way. Also, some problems, especially ones related to artificial intelligence, lend themselves to recursive solutions.

- Finally, some people seem to think recursively more easily than iteratively.

# Precaution

- When writing recursive functions, you must have an **if** statement somewhere to force the function to return without the recursive call being executed.

- If you don't, the function will never return once you call it.

# Exercise

Write a function:

```
count(int number,int array,int Size)
```

that counts the number of times number appears in array. The array has Size elements. The function should be recursive. Write a test program to test it.

innovate achieve lead

BITS Pilani

Pilani Campus

# *Thank you*

## Q & A