Data Structures and its Applications

Dinesh Singh

Department of Computer Science & Engineering

DATA STRUCTURES AND ITS APPLICATIONS

Application of stacks– Functions, nested functions and Recursion

Dinesh Singh
Department of Computer Science & Engineering

Application of stacks – Functions, nested functions

Activation record :

- When functions are called, The system (or the program) must remember the place where the call was made, so that it can return there after the function is complete.
- It must also remember all the local variables, processor registers, and the like, so that information will not be lost while the function is working.
- This information is considered as large data structure . This structure is sometimes called the invocation record or the activation record for the function call.

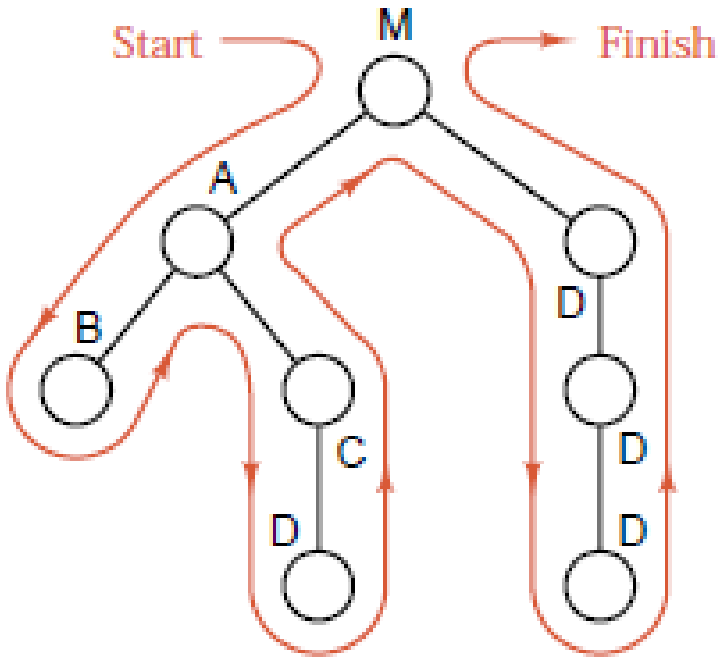
Application of stacks – Functions, nested functions and Recursion

- Suppose that there are three functions called A,B and C. M is the main function.
- Suppose that A invokes B and B invokes C. Then B will not have finished its work until C has finished and returned. Similarly, A is the first to start work, but it is the last to be finished, not until sometime after B has finished and returned.
- Thus the sequence by which function activity proceeds is summed up as the property last in, first out.
- The machine's task of assigning temporary storage area (activation records) used by functions would be in same order (LIFO).
- Since LIFO property is used, the machine allocates these records in the stack
- Hence a stack plays a key role in invoking functions in a computer system.

Application of stacks – Functions, nested functions

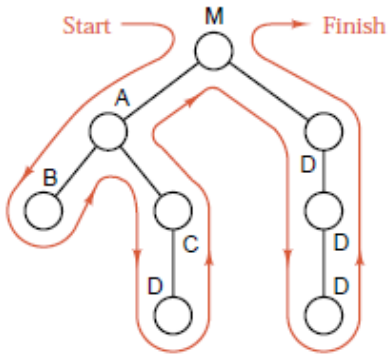
Example :

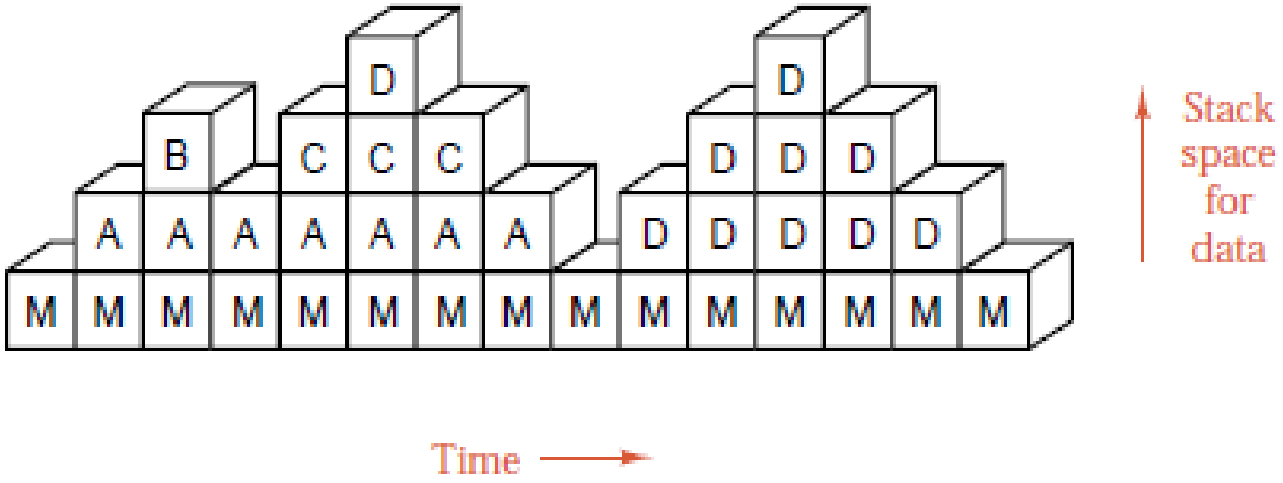
Consider the following showing the order in which the functions are invoked



Application of stacks – Functions, nested functions

The Sequence of stack frames of the activation records of the function calls is given below. Each vertical column shows the contents of the stack at a given time, and changes to the stack are portrayed by reading through the frames from left to right.



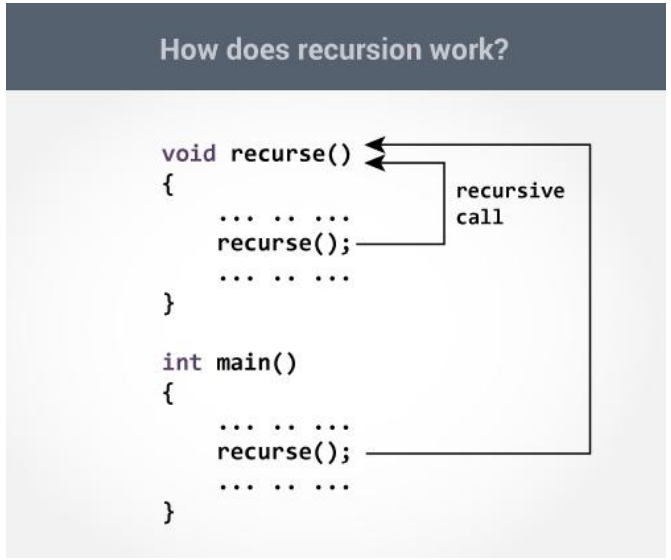


Application of stacks – Recursion

Recursion :

- Recursion is a computer programming technique involving the use of a procedure, subroutine or a function.

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function.



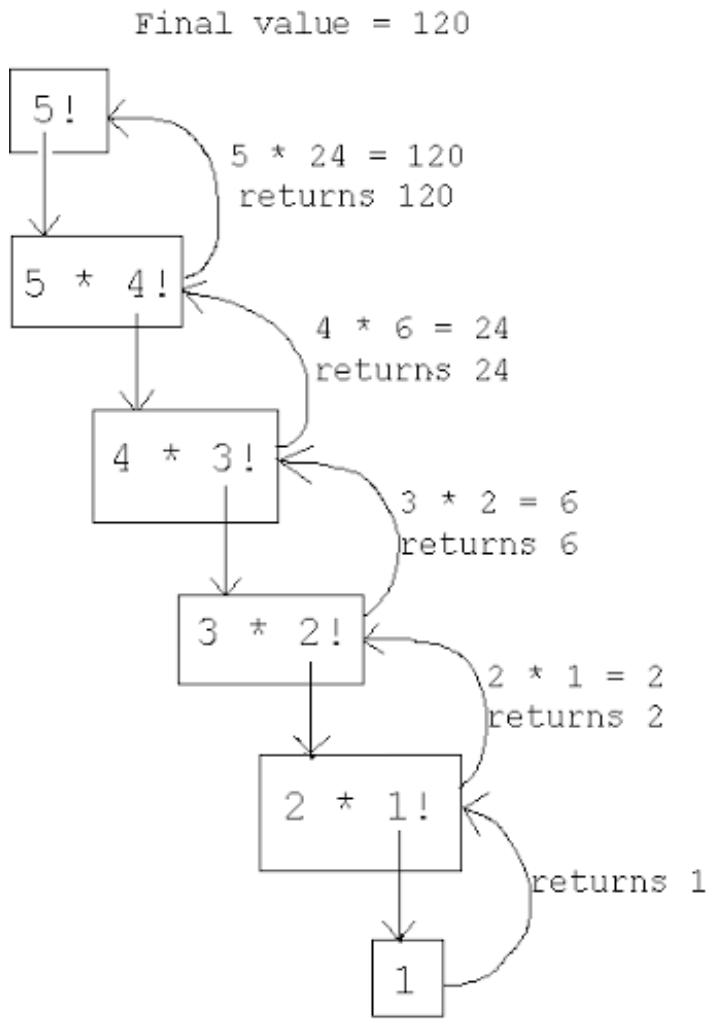
Application of stacks – Recursion

How a particular problem is solved using recursion?

- The idea is to represent a problem in terms of one or more smaller problems.
- A way is found to solve these smaller problems and then build up a solution to the entire problem.
- The sub problems are in turn broken down into smaller sub problems until the solution to the smallest sub problem is known.
- The solution to the smallest sub problem is called the base case.
- In the recursive program, the solution to thebase case is provided

Application of stacks – Recursion

Example 1: Factorial of a Number n



Recursive definition :

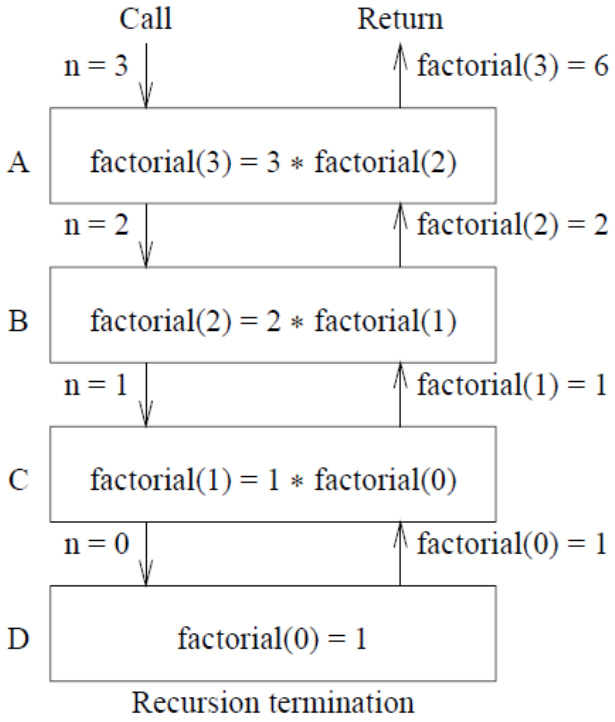
n! = 1 if n=1

n!= n * (n-1)! If n>1

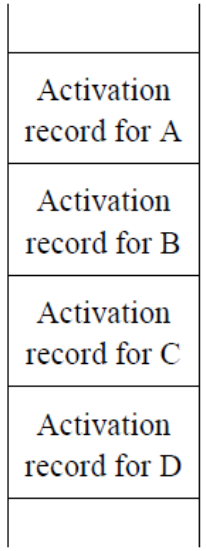
Application of stacks – Recursion

Recursive function to compute factorial of n

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



(a)



(b)

Application of stacks – Recursion

Recursive function to find the product of a*b

a*b= a if b=1;

a*b = a*(b-1)+a if b >1;

To evaluate 6 * 3

6*3 = 6*2 + 6 = 6*1 + 6 + 6 = 6 + 6 + 6 = 18

```
multiply(int a, int b)
{
   int p;
    if (b==1)
     return a
   p= multiply(a,b-1) + a;
  return p;
}
```

Application of stacks – Recursion

Fibonacci Sequence

The fibonacci sequence is the sequence of integers
1, 1, 2, 3, 5, 8, 13, 21, 34...

Each element is the sum of two preceding elements

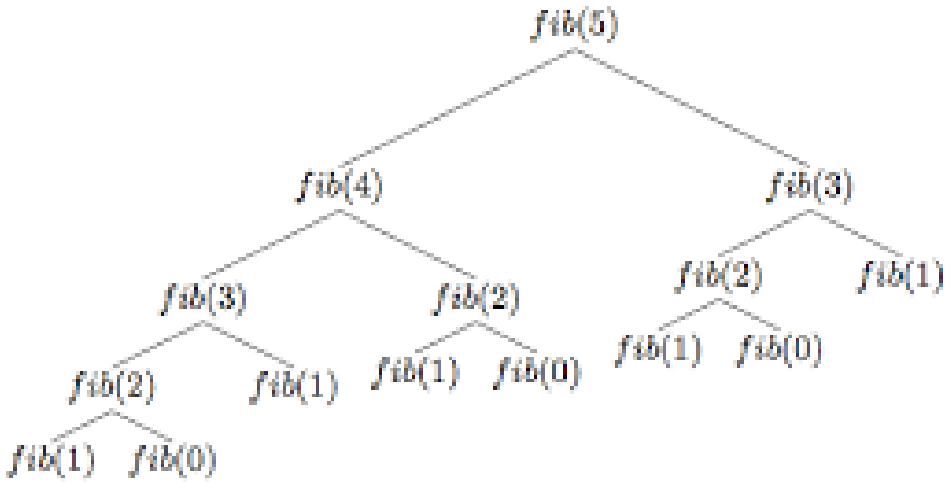
If we consider fib(0) = 1 and fib(1)=1
then recursive definition to compute the nth element in the sequence is

fib(n) = n if n=0 or n=1
fib(n) = fib(n-1) + fib(n-2) for n >=2

Application of stacks – Recursion

Recursive function to compute the nth element in the Fibonacci Sequence

```
 fib(int n)
{
   int x,y;
   if ( (n==0) || (n==1
     return n;
  x= fib(n-1) ;
  y=fib(n-2);
 return x+y;
}
```



Application of stacks – Recursion

Recursive function to find the sum of elements of an array

```
int sum(int *a, int n)
//a is pointer to the array, n is the index of the last element of the array
{
  int s;
  if(n==0) // base condition
   return a[0];
  s= sum(a,n-1) + a[n]; // compute sum of n-1 elements and add the nth element
  return s;
}
```

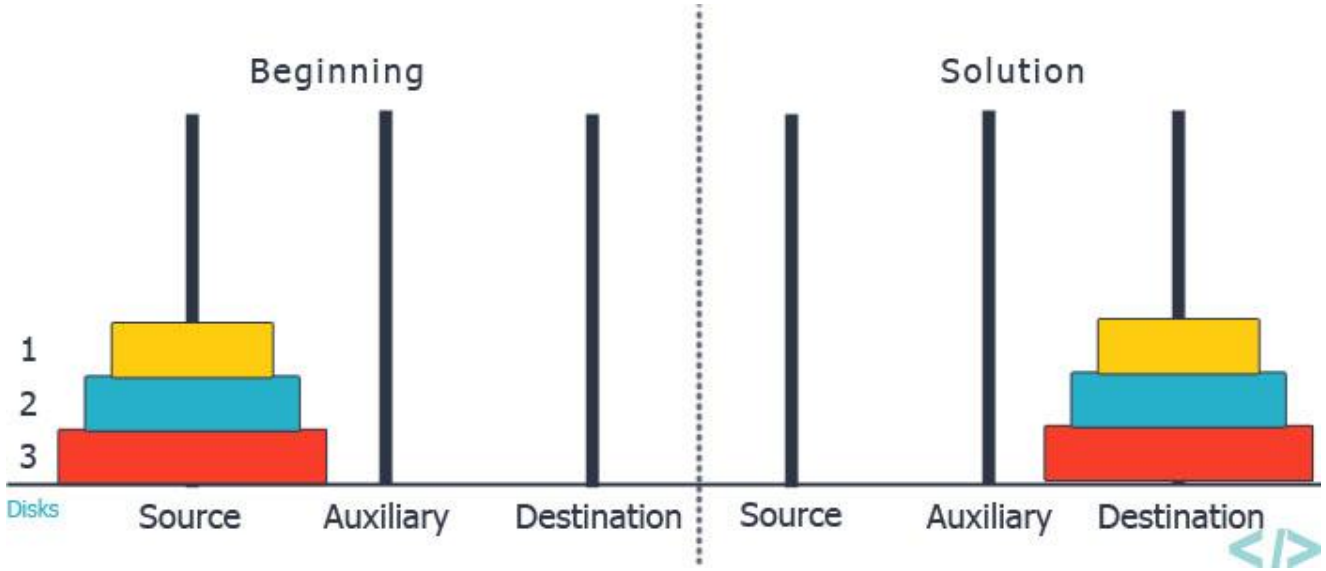
Application of stacks – Recursion

Recursive function to display the elements of the linked list in the reverse order

```
int display(struct node *p)
{
  if(p->next!=NULL)
     display(p->next)
  printf("%d ",p->data);
}
```

Tower of Hanoi

Three Pegs A, B and C exists. N disks of differing diameters are placed on peg A. The Larger disk is always below a smaller disk. The objective is to move the N disks from Peg A to Peg C using Peg B as the auxillary peg



Application of stacks – Recursion

Tower of Hanoi – recursive solution

If a solution to n-1 disks is found, then the problem would be solved. Because in the trivial case of one disk, the solution would be to move the single disk from Peg A to Peg C.

To move n disks from A to C , the recursive solution would be as follows

1. If n=1 move the single disk from A to C and stop
2. Move the top n-1 disks from A to B using C as auxillary
3. Move the remaining disk from A to C
4. Move n-1 disks from B to C using A as the auxillary

Application of stacks – Recursion

Recursive function for Tower of Hanoi

```
void tower(int n,char src,char tmp,char dst)
 {
  if(n==1)
  {
   printf("\nMove disk %d from %c to %c",n,src,dst);
   return;
  }
  tower(n-1,src,dst,tmp);
  printf("\nMove disk %d form %c to %c",n,src,dst);
  tower(n-1,tmp,src,dst);
  return;
 }
```

Application of stacks – Recursion

Recursive function for Tower of Hanoi

Solution for 4 disks

Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 3 from peg A to peg B
Move disk 1 from peg C to peg A
Move disk 2 from peg C to peg B
Move disk 1 from peg A to peg B
Move disk 4 from peg A to peg C
Move disk 1 from peg B to peg C
Move disk 2 from peg B to peg A
Move disk 1 from peg C to peg A
Move disk 3 from peg B to peg C
Move disk 1 from peg A to peg B
Move disk 2 from peg A to peg C
Move disk 1 from peg B to peg C

Solution for moving 3 disks from A to B
Move 4^{th} disk from A to C
Solution for moving 3 disks from B to C

Multiple-Choice-Questions(MCQ's)

1. Why is a stack used to manage function calls in most programming languages?

a) To allow random access to function calls.

b) To store the order of calls and return addresses (LIFO).

c) Because stacks consume less memory than arrays.

d) To store all function calls in the heap.

Multiple-Choice-Questions(MCQ's)

1. Why is a stack used to manage function calls in most programming languages?

a) To allow random access to function calls.

b) To store the order of calls and return addresses (LIFO).

c) Because stacks consume less memory than arrays.

d) To store all function calls in the heap.

2. When a recursive function is called, what gets stored in the stack for each function call?

a) Only the function name.

b) Return address, local variables, and parameters.

c) Only local variables.

d) Only parameters.

2. When a recursive function is called, what gets stored in the stack for each function call?

a) Only the function name.

b) Return address, local variables, and parameters.

c) Only local variables.

d) Only parameters.

3. Which of the following is true about nested function calls?

a) The most recent function call is executed first.

b) The first function call executes last due to the LIFO property of the stack.

c) Function calls are stored in a queue.

d) Nested calls do not require a stack.

3. Which of the following is true about nested function calls?

a) The most recent function call is executed first.

b) The first function call executes last due to the LIFO property of the stack.

c) Function calls are stored in a queue.

d) Nested calls do not require a stack.

4. What happens if recursion is not terminated properly?

a) The stack overflows.

b) The program compiles successfully but returns 0.

c) It creates an infinite loop without using stack.

d) Nothing happens, the function continues normally.

4. What happens if recursion is not terminated properly?

a) The stack overflows.

b) The program compiles successfully but returns 0.

c) It creates an infinite loop without using stack.

d) Nothing happens, the function continues normally.

5. How does the stack help in recursion?

a) By storing the base case condition.

b) By storing function frames to return to the previous state.

c) By allowing random jumps in memory.

d) By precomputing recursive calls.

5. How does the stack help in recursion?

a) By storing the base case condition.

b) By storing function frames to return to the previous state.

c) By allowing random jumps in memory.

d) By precomputing recursive calls.

THANK YOU

Dinesh Singh

Department of Computer Science & Engineering