JADAVPUR UNIVERSITY

Department of Electronics and TeleCommunication Engineering

Data Structures and Algorithms Lab

Assignment - 3 : Postfix Evaluation and Tower of Hanoi

Programming Language : C (ANSI)

==========

Sandip Dutta
2nd Year
Roll - 001910701017

==========

1. **Evaluate postfix expressions using a Stack. You can use , as a delimiter between two tokens in an expression. You can use $ as the end of input. Always show the intermediate steps including the content of the stack. Take some example postfix expressions. Here is one example:**
   6, 2, 3, +, -, 3, 8, 2, /, +, *, 2, ^, 3, +, $
   (Use ^ for exponentiation)

==========

```
/* Stack implementation with fixed memory array */
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <ctype.h>
# include <math.h>

/* Macros for easy interpretation */
# define NULL_STACK -1
# define MAX_STACK_CAPACITY 100
# define END_OF_EXPRESSION '$'
# define MAX_EXPRESSION_LENGTH 100

/* Stack has a fixed memory capacity and a top pointer */
typedef struct stk{
    int capacity;
    int top;
    int* arr;
}Stack;

/* Init a stack */
Stack* initStack(const int capacity){
    Stack *stk = malloc(sizeof(Stack));

    stk -> top = NULL_STACK;
    stk -> capacity = capacity;
    stk -> arr = malloc(capacity * sizeof(int));

    if(stk == NULL || stk->arr == NULL){
        printf("\nNot enough memory! Aborting\n");
        return NULL;
    }

    return stk;
}

/* Checker method for empty or not */
int isEmpty(const Stack* stk){
    return (stk->top == NULL_STACK);
}

/* Checker to see if stack is full or not */
int isFull(const Stack* stk){
    return (stk->top == stk->capacity - 1);
}

/* Size of the stack */
int size(const Stack *stk){
    return stk->top + 1;
}

/* push data into stack */
void push(Stack *stk, int data){
    /* If full, OVERFLOW */
    if(isFull(stk)){
        printf("\nOVERFLOW\n");
        return;
    }
    ++(stk -> top);
    stk -> arr[stk -> top] = data;
}

/* Pop data from stack */
int pop(Stack *stk){
    if(isEmpty(stk)){
```

```
        return END_OF_EXPRESSION;
    }
    return stk->arr[stk->top--];
}

/* print the stack */
void printStack(const Stack *stk){
    int lim = NULL_STACK;
    int i = stk -> top;
    printf("Stack Contents : ");

    if(isEmpty((stk))) return;
    while(i > lim){
        printf("%d ", stk->arr[i]);
        i--;
    }
    printf("\n");
}


int evaluatePostfixExpresssion(char* expression){
      Stack* stack = initStack(strlen(expression));
      int i, value1, value2;

      if (!stack){
        printf("No memory could be allocated. Operation failed.\n");
        exit(EXIT_FAILURE);
    }

      for (i = 0; expression[i] != END_OF_EXPRESSION; ++i){

           if (isdigit(expression[i])){
                push(stack, expression[i] - '0');
            printf("%c Pushed into the stack\n", expression[i]);
            printStack(stack);
       }

           else {
                int val1 = pop(stack);
                int val2 = pop(stack);
            if(val1 == '$' || val2 == '$') return END_OF_EXPRESSION;

                switch(expression[i]){
                    case '+':
                   printf("%d + %d = %d ... Pushing into stack\n", val2, val1, val2 + val1);
                   push(stack, val2 + val1);
                   printStack(stack);
                   break;
                    case '-':
                   printf("%d - %d = %d ... Pushing into stack\n", val2, val1, val2 + val1);
                   push(stack, val2 - val1);
                   printStack(stack);
                   break;
                    case '*':
                   printf("%d * %d = %d ... Pushing into stack\n", val2, val1, val2 * val1);
                   push(stack, val2 * val1);
                   printStack(stack);
                   break;
                    case '/':
                   printf("%d / %d = %d ... Pushing into stack\n", val2, val1, val2 / val1);
                   push(stack, val2 / val1);
                   printStack(stack);
                   break;
               case '^':
                   printf("%d ^ %d = %d ... Pushing into stack\n", val2, val1, (int)pow(val2,
val1));
                   push(stack, (int)pow(val2, val1));
                   printStack(stack);
                   break;
                }
           }
      }
      return pop(stack);
```

```
}

int main(){
    char expression[MAX_EXPRESSION_LENGTH];
    int finalExpressionValue;
    char endChar = END_OF_EXPRESSION;

    printf("Enter an expression [MAX LEN 100] : ");
    scanf("%s", expression);
    strncat(expression, &endChar, 1);


    finalExpressionValue = evaluatePostfixExpresssion(expression);
    if(finalExpressionValue != '$')
        printf("Value of Expression is : %d\n", finalExpressionValue);
    else
        printf("YOU MADE A MISTAKE SOMEWHERE!!\n GARBAGE OUTPUT!!");
    return 0;
}
```

==========

OUTPUT:

1. Cases where Postfix expression is wrong or some error is there in the expression

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23+++
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 + 3 = 5 ... Pushing into stack
Stack Contents : 5
YOU MADE A MISTAKE SOMEWHERE!!
 GARBAGE OUTPUT!!%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : ^
YOU MADE A MISTAKE SOMEWHERE!!
 GARBAGE OUTPUT!!%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : +++
YOU MADE A MISTAKE SOMEWHERE!!
 GARBAGE OUTPUT!!%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23^+
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 ^ 3 = 8 ... Pushing into stack
Stack Contents : 8
YOU MADE A MISTAKE SOMEWHERE!!
 GARBAGE OUTPUT!!%
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
```

2. Normal Simple Expressions for all operators (PTO)

```
(base) ✗ sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23+
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 + 3 = 5 ... Pushing into stack
Stack Contents : 5
Value of Expression is : 5
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23-
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 - 3 = 5 ... Pushing into stack
Stack Contents : -1
Value of Expression is : -1
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23*
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 * 3 = 6 ... Pushing into stack
Stack Contents : 6
Value of Expression is : 6
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23/
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 / 3 = 0 ... Pushing into stack
Stack Contents : 0
Value of Expression is : 0
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./postfix
Enter an expression [MAX LEN 100] : 23^
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 ^ 3 = 8 ... Pushing into stack
Stack Contents : 8
Value of Expression is : 8
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ±
```

3. Larger Expressions

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main gcc -ansi -o postfix ./Postix_stack.c -lm
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./postfix
Enter an expression [MAX LEN 100] : 23+4-
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
2 + 3 = 5 ... Pushing into stack
Stack Contents : 5
4 Pushed into the stack
Stack Contents : 4 5
5 - 4 = 9 ... Pushing into stack
Stack Contents : 1
Value of Expression is : 1
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./postfix
Enter an expression [MAX LEN 100] :  234*+5+
2 Pushed into the stack
Stack Contents : 2
3 Pushed into the stack
Stack Contents : 3 2
4 Pushed into the stack
Stack Contents : 4 3 2
3 * 4 = 12 ... Pushing into stack
Stack Contents : 12 2
2 + 12 = 14 ... Pushing into stack
Stack Contents : 14
5 Pushed into the stack
Stack Contents : 5 14
14 + 5 = 19 ... Pushing into stack
Stack Contents : 19
Value of Expression is : 19
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main
```

2. Write a program for the Tower of Hanoi problem. Experiment with a varying number of discs. Show the intermediate moves in form of messages like the following: Move disk ... from rod ... to rod ...

```
================================================================================================
IMPLEMENTATION 1 : USING STACK TRACING
================================================================================================
/*
NOTE : THIS CODE USES CALL STACK, ANOTHER CODE IS PROVIDED LATER
*/

# include <stdio.h>
# include <stdlib.h>
# include <execinfo.h>
# define BT_BUFFER_SIZE 2048

/*

Call Stack - whenever a function is called, to know the current
control, C maintains a call stack that is which function the program
is currently in.

In this program we print the current stack call using
execinfo library.

THIS IS A LINUX ONLY(IE GCC ONLY) IMPLEMENTATION.

*/

/*
printCallStack - Prints the call stack
 */
void printCallStack(){
      int numPointers;
      void *buffer[BT_BUFFER_SIZE];
      char **functionCalls;
      int j;

      printf("\n");
      numPointers = backtrace(buffer, BT_BUFFER_SIZE);
      printf("\n CALL STACK HAS %d FUNCTIONS\n", numPointers);

      functionCalls = backtrace_symbols(buffer, numPointers);
      if (functionCalls == NULL) {
            printf("EMPTY STACK\n\n");
            exit(EXIT_FAILURE);
      }

      for (j = 0; j < numPointers; j++)
            printf("\t%s\n", functionCalls[j]);

      printf("\n");
      free(functionCalls);
}

/* Tower of Hanoi using Recursion and Stack Calling */
void towerOfHanoi(int diskNumber, char from, char to, char aux, int printStackTrace){
      if(printStackTrace) printCallStack();
      printf("\n CURRENT ARGS [ SOURCE : %c , AUX : %c, DESTINATION : %c]\n", from, aux, to);
      if (diskNumber == 1)   {
            printf("Move disk 1 from  %c to %c\n", from, to);
            return;
      }
      towerOfHanoi(diskNumber - 1, from, aux, to, printStackTrace);
      printf("Move disk %d from  %c to %c\n", diskNumber, from, to);
      towerOfHanoi(diskNumber - 1, aux, to, from, printStackTrace);
}

int main(){
      int n;
      int printStackTrace;
      printf("Do we need to print the call stack? Enter your choice 1[YES] or 0[NO] : ");
      scanf("%d", &printStackTrace);
```

```
    printf("Enter the number of disks [Input preferably less than 10] : ");
    scanf("%d", &n);
      towerOfHanoi(n, 'A', 'C', 'B', printStackTrace);
      return 0;
}
```

==========

OUTPUT

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± gcc -ansi -o hanoi ./Tower_of_Hanoi.c
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./hanoi
Do we need to print the call stack? Enter your choice 1[YES] or 0[NO] : 1
Enter the number of disks [Input preferably less than 10] : 2


 CALL STACK HAS 5 FUNCTIONS
       ./hanoi(+0x8f2) [0x55c7b446a8f2]
       ./hanoi(+0x9fd) [0x55c7b446a9fd]
       ./hanoi(+0xb25) [0x55c7b446ab25]
       /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f9bd89abbf7]
       ./hanoi(+0x7da) [0x55c7b446a7da]


 CURRENT ARGS [ SOURCE : A , AUX : B, DESTINATION : C]


 CALL STACK HAS 6 FUNCTIONS
       ./hanoi(+0x8f2) [0x55c7b446a8f2]
       ./hanoi(+0x9fd) [0x55c7b446a9fd]
       ./hanoi(+0xa5e) [0x55c7b446aa5e]
       ./hanoi(+0xb25) [0x55c7b446ab25]
       /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f9bd89abbf7]
       ./hanoi(+0x7da) [0x55c7b446a7da]


 CURRENT ARGS [ SOURCE : A , AUX : C, DESTINATION : B]
Move disk 1 from  A to B
Move disk 2 from  A to C


 CALL STACK HAS 6 FUNCTIONS
       ./hanoi(+0x8f2) [0x55c7b446a8f2]
       ./hanoi(+0x9fd) [0x55c7b446a9fd]
       ./hanoi(+0xa9b) [0x55c7b446aa9b]
       ./hanoi(+0xb25) [0x55c7b446ab25]
       /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7) [0x7f9bd89abbf7]
       ./hanoi(+0x7da) [0x55c7b446a7da]


 CURRENT ARGS [ SOURCE : B , AUX : A, DESTINATION : C]
Move disk 1 from  B to C
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ±
```

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ± ./hanoi
Do we need to print the call stack? Enter your choice 1[YES] or 0[NO] : 0
Enter the number of disks [Input preferably less than 10] : 3

 CURRENT ARGS [ SOURCE : A , AUX : B, DESTINATION : C]

 CURRENT ARGS [ SOURCE : A , AUX : C, DESTINATION : B]

 CURRENT ARGS [ SOURCE : A , AUX : B, DESTINATION : C]
Move disk 1 from  A to C
Move disk 2 from  A to B

 CURRENT ARGS [ SOURCE : C , AUX : A, DESTINATION : B]
Move disk 1 from  C to B
Move disk 3 from  A to C

 CURRENT ARGS [ SOURCE : B , AUX : A, DESTINATION : C]

 CURRENT ARGS [ SOURCE : B , AUX : C, DESTINATION : A]
Move disk 1 from  B to A
Move disk 2 from  B to C

 CURRENT ARGS [ SOURCE : A , AUX : B, DESTINATION : C]
Move disk 1 from  A to C
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ±
```

```
================================================================================================
IMPLEMENTATION 2 : USING STATIC STACK
================================================================================================
/* ToWER of hanoi using user defined stack */
# include <stdio.h>
# include <stdlib.h>
# include <limits.h>
# include <math.h>
/* Macros for easy interpretation */
# define NULL_STACK -1
# define MAX_STACK_CAPACITY 1000

/* Stack has a fixed memory capacity and a top pointer */
typedef struct stk{
    int capacity;
    int top;
    int* arr;
}Stack;

/* Init a stack */
Stack* initStack(const int capacity){
    Stack *stk = malloc(sizeof(Stack));

    stk -> top = NULL_STACK;
    stk -> capacity = capacity;
    stk -> arr = malloc(capacity * sizeof(int));

    if(stk == NULL || stk->arr == NULL){
        printf("\nNot enough memory! Aborting\n");
        return NULL;
    }

    return stk;
}

/* Checker method for empty or not */
int isEmpty(const Stack* stk){
    return (stk->top == NULL_STACK);
}

/* Checker to see if stack is full or not */
int isFull(const Stack* stk){
    return (stk->top == stk->capacity - 1);
}

/* Size of the stack */
int size(const Stack *stk){
    return stk->top + 1;
}

/* push data into stack */
void push(Stack *stk, int data){
    /* If full, OVERFLOW */
    if(isFull(stk)){
        printf("\nOVERFLOW\n");
        return;
    }
    ++(stk -> top);
    stk -> arr[stk -> top] = data;
}

/* Pop data from stack */
int pop(Stack *stk){
    if(isEmpty(stk)){
        return INT_MIN;
    }
    return stk->arr[stk->top--];
}

/* print the stack */
void printStack(const Stack *stk){
    int lim = NULL_STACK;
```

```
    int i = stk -> top;
    printf("Stack Contents -> (top) ");

    if(isEmpty((stk))) {
        printf("\n");
        return;
    }

    while(i > lim){
        printf("%d -> ", stk->arr[i]);
        i--;
    }
    printf("\n");
}

/* Tower of Hanoi specific utilites */
void printMove(char from, char to, char diskno){
     printf("\n\tSTEP : Move the disk %d from %c to %c\n", diskno, from, to);
}

void moveDisks(Stack *src, Stack *dest, char s, char d){
    int pole1Top = pop(src);
    int pole2Top = pop(dest);

    if (pole1Top == INT_MIN){
            /* Empty */
        push(src, pole2Top);
        printMove(d, s, pole2Top);
    }

    else if (pole2Top == INT_MIN){
            /* Empty */
        push(dest, pole1Top);
        printMove(s, d, pole1Top);
    }

    else if (pole1Top > pole2Top){
            /* Put greater pole disk size */
        push(src, pole1Top);
        push(src, pole2Top);
        printMove(d, s, pole2Top);
    }

    else{
        push(dest, pole2Top);
        push(dest, pole1Top);
        printMove(s, d, pole1Top);
    }
}


/* Hanoi main function */
void towerOfHanoi(int num_of_disks, Stack *src, Stack *aux, Stack *dest){
    int i, total_num_of_moves;
    char s = 'A', d = 'B', a = 'C';

   /*  If number of disks is even, then interchange
    dest and aux*/
    if (num_of_disks % 2 == 0){
        char temp = d;
        d = a;
        a = temp;
    }
    total_num_of_moves = pow(2, num_of_disks) - 1;

    for (i = num_of_disks; i >= 1; i--)
        push(src, i);

    for (i = 1; i <= total_num_of_moves; i++){
        printf("SOURCE : ");
        printStack(src);
        printf("AUX : ");
        printStack(aux);
```

```
        printf("DEST : ");
        printStack(dest);
        if (i % 3 == 1)
        moveDisks(src, dest, s, d);

        else if (i % 3 == 2)
        moveDisks(src, aux, s, a);

        else if (i % 3 == 0)
        moveDisks(aux, dest, a, d);
    }
}

int main(){
    int num_of_disks;

    Stack *src, *dest, *aux;

      printf("Enter number of disks [Less than 10 as huge runtime] : ");
      scanf("%d", &num_of_disks);

    if(num_of_disks > 10 || num_of_disks < 1){
        perror("BAD INPUT. ABORT\n");
        exit(EXIT_FAILURE);
    }

    /* Create three stacks of size 'num_of_disks'
    to hold the disks */
    src = initStack(num_of_disks);
    aux = initStack(num_of_disks);
    dest = initStack(num_of_disks);

    towerOfHanoi(num_of_disks, src, aux, dest);
    return 0;
}
```

==========

OUTPUT :

1. Small Inputs and Invalid Characters

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 92347017230
BAD INPUT. ABORT
: Success
(base) ✘ sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : -1
BAD INPUT. ABORT
: Success
(base) ✘ sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 0
BAD INPUT. ABORT
: Success
(base) ✘ sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 1
SOURCE : Stack Contents -> (top) 1 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to B
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 2
SOURCE : Stack Contents -> (top) 1 -> 2 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from C to B
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main
```

2. Medium Inputs

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main gcc -ansi -o hanoi2 ./Tower_of_Hanoi_STACK.c -lm
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 3
SOURCE : Stack Contents -> (top) 1 -> 2 -> 3 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to B
SOURCE : Stack Contents -> (top) 2 -> 3 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to C
SOURCE : Stack Contents -> (top) 3 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from B to C
SOURCE : Stack Contents -> (top) 3 ->
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 3 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 1 from C to A
SOURCE : Stack Contents -> (top) 1 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 2 from C to B
SOURCE : Stack Contents -> (top) 1 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 1 from A to B
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main
```

3. Large Input (Spans multiple pages)

```
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main gcc -ansi -o hanoi2 ./Tower_of_Hanoi_STACK.c -lm
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3 main ./hanoi2
Enter number of disks [Less than 10 as huge runtime] : 6
SOURCE : Stack Contents -> (top) 1 -> 2 -> 3 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 -> 3 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top) 3 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 3 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 3 from A to C
SOURCE : Stack Contents -> (top) 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 2 from B to C
SOURCE : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 4 -> 5 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 4 from A to B
SOURCE : Stack Contents -> (top) 5 -> 6 ->
AUX : Stack Contents -> (top) 4 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 4 ->
```

```
        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 4 ->
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 2 from C to A
SOURCE : Stack Contents -> (top) 2 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 4 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 2 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 4 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 3 from C to B
SOURCE : Stack Contents -> (top) 1 -> 2 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 3 -> 4 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 -> 5 -> 6 ->
AUX : Stack Contents -> (top) 3 -> 4 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top) 5 -> 6 ->
AUX : Stack Contents -> (top) 2 -> 3 -> 4 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 5 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 2 -> 3 -> 4 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 5 from A to C
SOURCE : Stack Contents -> (top) 6 ->
AUX : Stack Contents -> (top) 1 -> 2 -> 3 -> 4 ->
DEST : Stack Contents -> (top) 5 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 6 ->
AUX : Stack Contents -> (top) 2 -> 3 -> 4 ->
DEST : Stack Contents -> (top) 5 ->

        STEP : Move the disk 2 from B to C
SOURCE : Stack Contents -> (top) 1 -> 6 ->
AUX : Stack Contents -> (top) 3 -> 4 ->
DEST : Stack Contents -> (top) 2 -> 5 ->
```

```
        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 6 ->
AUX : Stack Contents -> (top) 3 -> 4 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 5 ->

        STEP : Move the disk 3 from B to A
SOURCE : Stack Contents -> (top) 3 -> 6 ->
AUX : Stack Contents -> (top) 4 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 5 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 3 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 4 ->
DEST : Stack Contents -> (top) 2 -> 5 ->

        STEP : Move the disk 2 from C to A
SOURCE : Stack Contents -> (top) 2 -> 3 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 4 ->
DEST : Stack Contents -> (top) 5 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 2 -> 3 -> 6 ->
AUX : Stack Contents -> (top) 4 ->
DEST : Stack Contents -> (top) 5 ->

        STEP : Move the disk 4 from B to C
SOURCE : Stack Contents -> (top) 1 -> 2 -> 3 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 4 -> 5 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 -> 3 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 -> 4 -> 5 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top) 3 -> 6 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 1 -> 4 -> 5 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 3 -> 6 ->
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top) 4 -> 5 ->

        STEP : Move the disk 3 from A to C
SOURCE : Stack Contents -> (top) 6 ->
AUX : Stack Contents -> (top) 1 -> 2 ->
DEST : Stack Contents -> (top) 3 -> 4 -> 5 ->
```

```
        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 6 ->
AUX : Stack Contents -> (top) 2 ->
DEST : Stack Contents -> (top) 3 -> 4 -> 5 ->

        STEP : Move the disk 2 from B to C
SOURCE : Stack Contents -> (top) 1 -> 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 2 -> 3 -> 4 -> 5 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 6 ->
AUX : Stack Contents -> (top)
DEST : Stack Contents -> (top) 1 -> 2 -> 3 -> 4 -> 5 ->

        STEP : Move the disk 6 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 6 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 -> 4 -> 5 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 1 -> 6 ->
DEST : Stack Contents -> (top) 2 -> 3 -> 4 -> 5 ->

        STEP : Move the disk 2 from C to A
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top) 1 -> 6 ->
DEST : Stack Contents -> (top) 3 -> 4 -> 5 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 2 ->
AUX : Stack Contents -> (top) 6 ->
DEST : Stack Contents -> (top) 3 -> 4 -> 5 ->

        STEP : Move the disk 3 from C to B
SOURCE : Stack Contents -> (top) 1 -> 2 ->
AUX : Stack Contents -> (top) 3 -> 6 ->
DEST : Stack Contents -> (top) 4 -> 5 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top) 3 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 4 -> 5 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 2 -> 3 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 4 -> 5 ->
```

```
        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 -> 3 -> 4 ->
AUX : Stack Contents -> (top) 5 -> 6 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top) 3 -> 4 ->
AUX : Stack Contents -> (top) 2 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top) 3 -> 4 ->
AUX : Stack Contents -> (top) 1 -> 2 -> 5 -> 6 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 3 from A to C
SOURCE : Stack Contents -> (top) 4 ->
AUX : Stack Contents -> (top) 1 -> 2 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 4 ->
AUX : Stack Contents -> (top) 2 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 2 from B to C
SOURCE : Stack Contents -> (top) 1 -> 4 ->
AUX : Stack Contents -> (top) 5 -> 6 ->
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 4 ->
AUX : Stack Contents -> (top) 5 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 4 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 2 from C to A
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 3 ->
```

```
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 2 from B to C
SOURCE : Stack Contents -> (top) 1 -> 4 ->
AUX : Stack Contents -> (top) 5 -> 6 ->
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 4 ->
AUX : Stack Contents -> (top) 5 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 4 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 1 -> 2 -> 3 ->

        STEP : Move the disk 1 from C to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 2 -> 3 ->

        STEP : Move the disk 2 from C to A
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top) 1 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 1 from B to A
SOURCE : Stack Contents -> (top) 1 -> 2 ->
AUX : Stack Contents -> (top) 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 3 ->

        STEP : Move the disk 3 from C to B
SOURCE : Stack Contents -> (top) 1 -> 2 ->
AUX : Stack Contents -> (top) 3 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top)

        STEP : Move the disk 1 from A to C
SOURCE : Stack Contents -> (top) 2 ->
AUX : Stack Contents -> (top) 3 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 2 from A to B
SOURCE : Stack Contents -> (top)
AUX : Stack Contents -> (top) 2 -> 3 -> 4 -> 5 -> 6 ->
DEST : Stack Contents -> (top) 1 ->

        STEP : Move the disk 1 from C to B
(base) sandip@Machine /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_3  main
```