Dr. Ohidujjaman Tuhin
Assistant Professor
Dept. of CSE, UIU

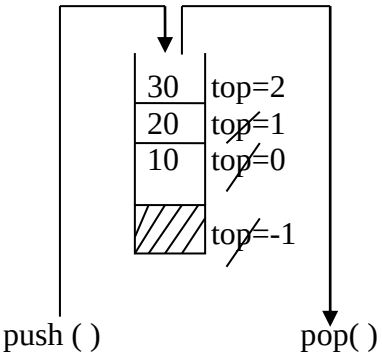
Basic of Stack

Definition: A Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be Last In First Out (LIFO) or First In Last Out(FILO). LIFO implies that the element that is inserted last, comes out first and FILO means that the component that is inserted first, comes out last.

It behaves like a stack of plates, where the last plate added is the first one to be removed. Think of it this way:

- Pushing an element onto the stack is like adding a new plate on top.
- Popping an element removes the top plate from the stack.

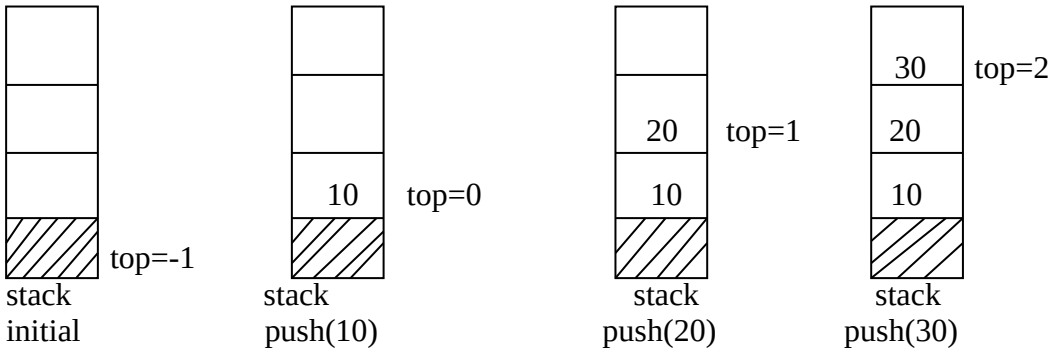
Stack is LIFO (Last in First Out) data structure. The following figure shows the stack structure.

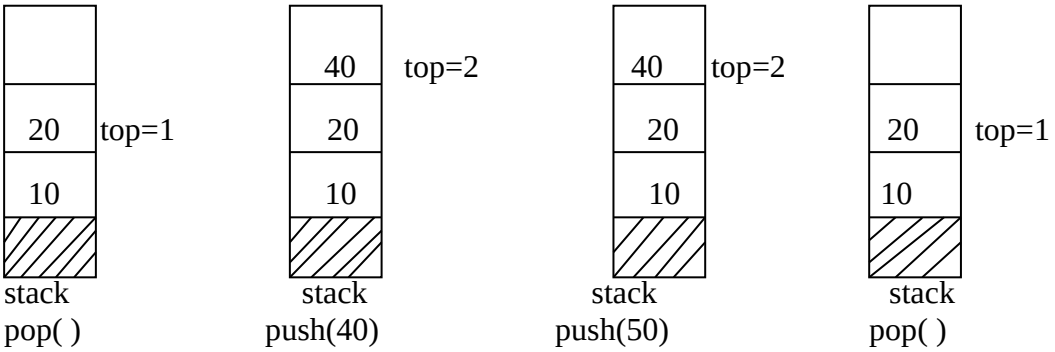


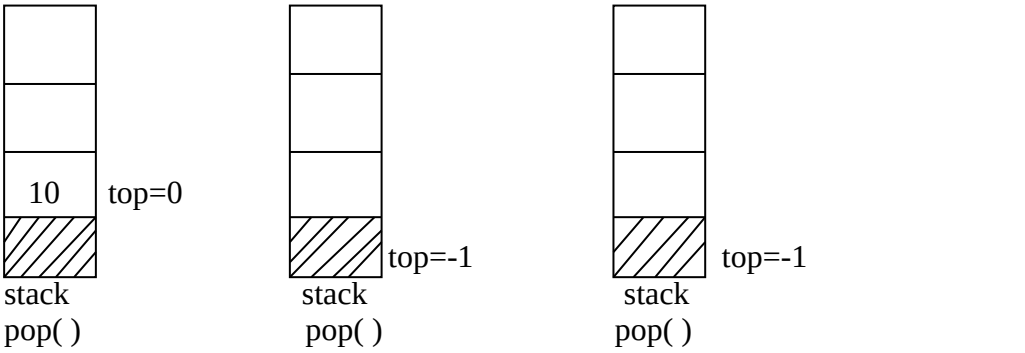
Stack Operation

There are two basic operation of Stack. They are 'push' and 'pop' operations. Push operation refers to insert data into 'stack' and, Pop operation represents to remove data from 'stack'. Push and Pop operation and the stack status are in the following figures:

Let Stack size m=3 i.e array size [3]; and inserted data item are such as 10, 20, 30, and 40





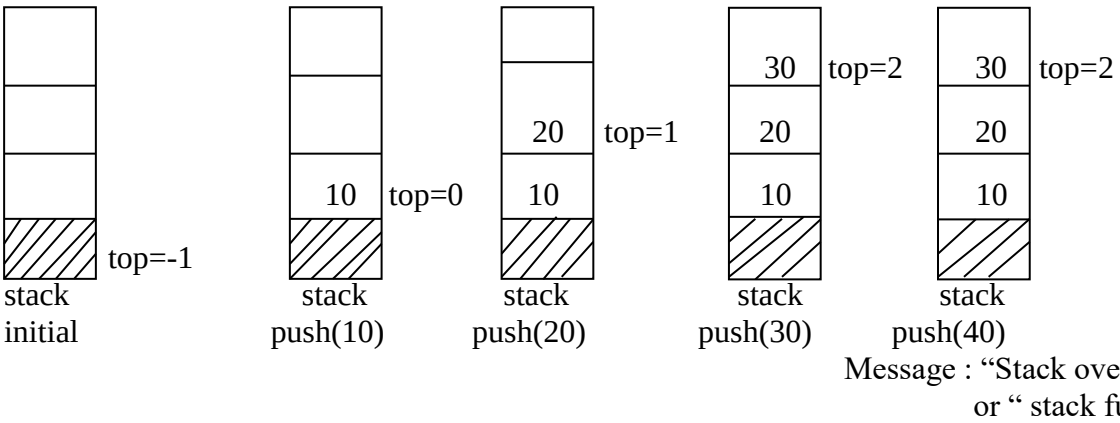


Here we initialize the value of ‘top’= -1; due to stack is empty.

Implementation of Stack

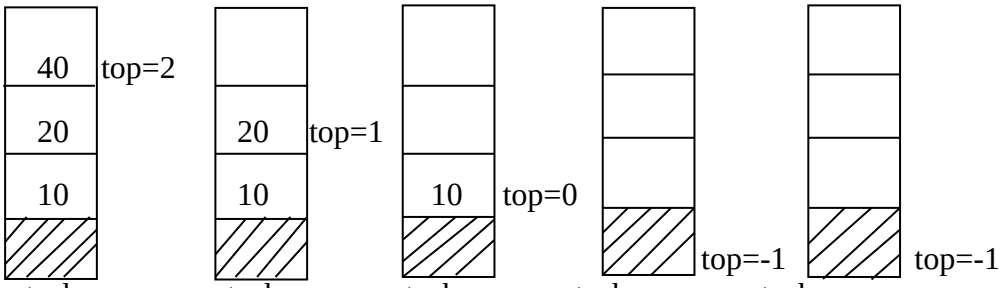
Stack can be implemented using two ways such as ‘array’ and ‘linked lists’. Push and Pop operations of stack are performed using array and linked lists.

Implementation of Stack using Array



Push operation logic:

```
if(top+1>=m)                //m= stack size ; stack[m]
{
printf("\nStack overflow");
}
else
{
stack[top+1]=x;             // x=inserted data item
top=top+1;
}
```



stack push(40) | stack pop( ) | stack pop( ) | stack pop( ) | stack pop( )

Message: "Stack underflow" or stack empty

Pop operation logic:

```
if(top= = -1)
{
printf("\n Stack underflow");
}
else
{
 x=stack[top];
 printf("\nThe pop value: %d",x);
 top=top-1;
}
```

Write a C program for push and pop operation of stack using array:

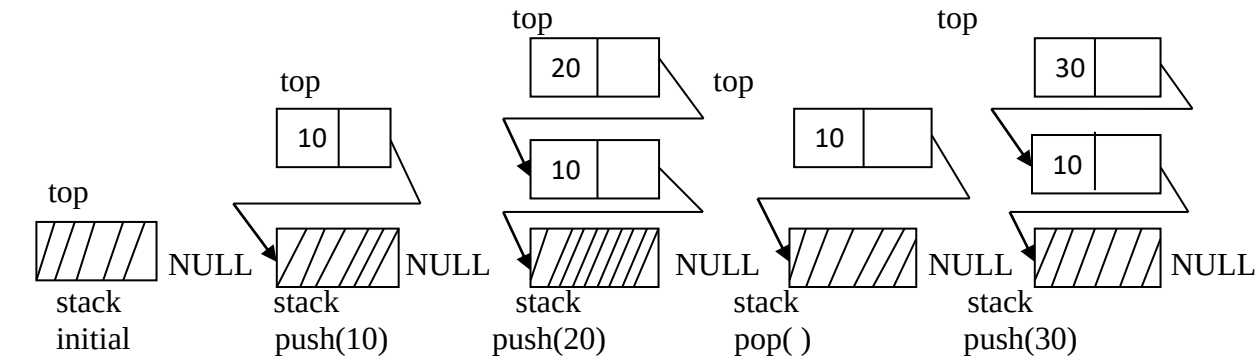
```
#include<stdio.h>
#define SIZE 3
int top=-1;
int stack[SIZE]={0};
int push(int x);
int pop( );
int main( )
{
        int i,x;
        for(i=1;i<5;i++)
        {
        printf("Enter the push value:");
        scanf("%d",&x);
        push(x);
        }
       pop( );
        pop( );
        pop( );
        pop( );
        printf("\n");
        return 0;
}
int push(int x)
{
        if(top+1>=SIZE)
        {
                printf("\nOverflow!!");
        }
        else
        {
                stack[top+1]=x;
                top=top+1;
        }
        return x;
}
int pop( ){
        int x;
        if(top==-1)
        {
                printf("\nUnderflow");
                return -1;
        }
        else
        {
                x=stack[top];
                printf("\nThe pop value:%d",x);
                top=top-1;
                return x;
        }
}
```

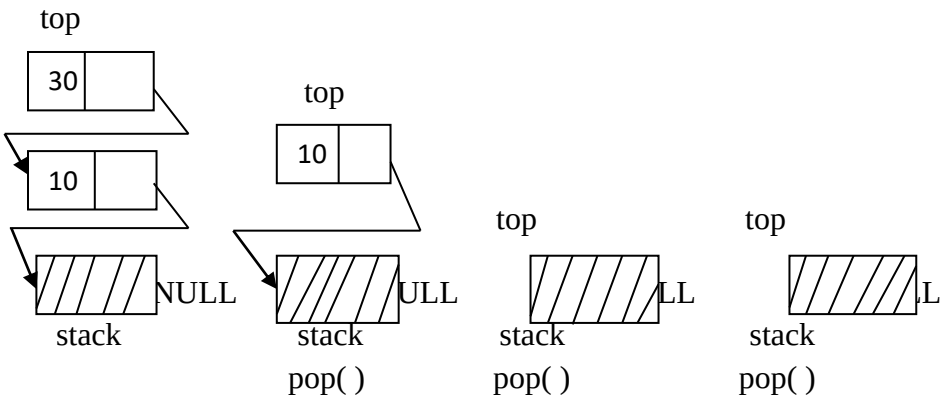
Output on monitor:

```
Enter the push value:10
Enter the push value:20
Enter the push value:30
Enter the push value:40
Stack is Overflow!!
The pop value:30
The pop value:20
The pop value:10
Stack is Underflow
```

Implementation of Stack using Linked lists

The Push and Pop operations of stack using linked list are in the following:

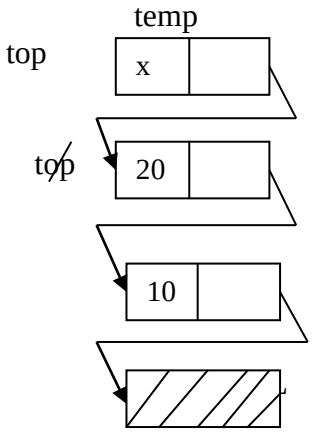




Message: “Stack underflow”

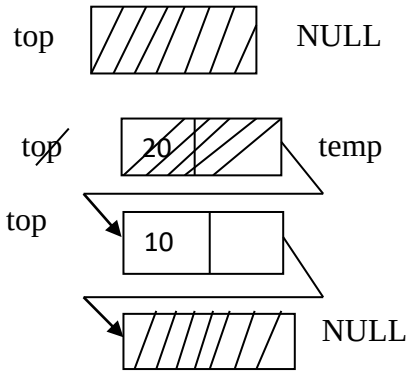
Push operation:

```
temp=(node*)malloc(sizeof(node));
                if(temp!=NULL)
                {
                   printf("\n Enter a data:=");
                   scanf("%d",&x);
                   temp->data=x;
                   temp->next=top;
                   top=temp;
                }
                else
                {
                   printf("\nMemory Out");
                }
```



Pop operation:

```
if (top==NULL)
 {
    printf("\nStack Underflow\n");
     exit(0);
  }
  else
  {
        temp=top;
         printf(" %d ",temp->data);
         top=top->next;
         free(temp);
  }
```



Write a C program for push and pop operation of stack using Linked list:

```
 #include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
struct list
{
        int data;
        struct list *next;
};
typedef struct list node;
int push( int x);
node *pop(node *top, int x);
void display(node *top);
int main()
{
```

```
        int x;
        node *top;
        node *temp;
        char ans;
        top=NULL;
        do
        {
                fflush(stdout);
                printf("\n Do you want to create a node(Y/N):=");
                fflush(stdin);
                ans=toupper(getchar());
                if(ans=='Y')
                {
                        temp=(node*)malloc(sizeof(node));
                        if(temp!=NULL)
                        {
                                fflush(stdout);
                                printf("\n Enter a data:=");
                                fflush(stdin);
                                scanf("%d",&x);
                                temp->data=x;
                                temp->next=top;
                                top=temp;
                        }
                        else
                        {
                                printf("\nMemory Out");
                        }
                }
        }
        while(ans=='Y');
        display(top);
        top=pop(top, x);
        return 0;
}
void display(node *top)
{
   node *temp1;
   printf("\n Created list:=");
   temp1=top;
   while(temp1!=NULL)
     {
       printf("   %d",temp1->data);
       temp1=temp1->next;
      }
         return;
```

```
}
node *pop(node *top, int x)
{
  node *temp;
  int i;
  for( i=1;i<5;i++)
 {
  if (top==NULL)
  {
    printf("\nStack Underflow\n");
          exit(0);
  }
  else
  {
         temp=top;
         printf("\nPop value:%d\n",temp->data);
         top=top->next;
         free(temp);
  }
 }
return top;
}
```

Output on monitor:

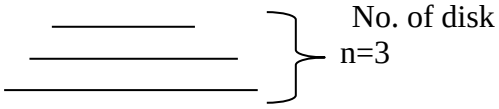
```
Do you want to create a node(Y/N):=y
Enter a data:=20
Do you want to create a node(Y/N):=y
Enter a data:=30
Do you want to create a node(Y/N):=n
Created list:=   30   20   10
Pop value:30
Pop value:20
Pop value:10
Stack Underflow
```

Stack Applications

There are several stack applications such as Tower of Hanoi, Infix to Postfix conversion, Postfix evaluation, Reverse a string and Depth First Search. These applications are details in the following:

Tower of Hanoi

Tower of Hanoi is one of the perceptible applications of stack. In this stack application components should be move from source to destination with some conditions. For example, let three disks are organized in a source location (i) and need to move them into the destination (k).



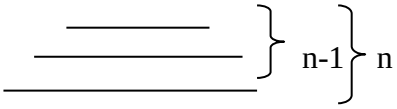
Source (i) Destination (k)

If we want to move from source (i) to destination (k), but the conditions are below:

1. It should move one disk at a time.

2. It must not put larger on smaller.

So, according to the conditions, it is not possible. Therefore, it needs to take Helper location (j).



Source (i) Helper(j) Destination (k)

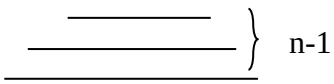
Step1: First move (n-1) disks from source (i) to helper (j)

Source (i) Helper(j) Destination (k)

Step2: Move the residual disk from source (i) to destination (k)

Source (i) Helper(j) Destination (k)

Step3: Move (n-1) disks from helper (j) to destination (k)



Source (i) Helper(j) Destination (k)

If No. of disk, n=1; source(i) destination(k)

It moves from source (i) to direct the destination (k).

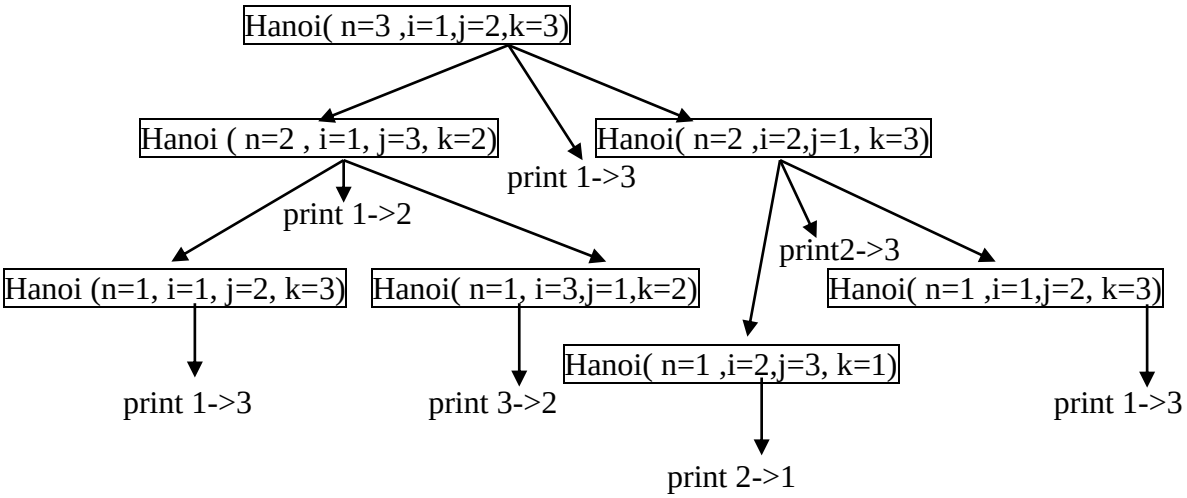
source (i) destination(k)

Recursive algorithm of Tower of Hanoi

```
void Hanoi(int n, int i, int j, int k)
{
if (n= =1)
printf("\n% d ->%d" i, k);
else
{
Hanoi(n-1, i, k, j);
printf("\n% d ->%d" i, k);
Hanoi(n-1, j, i, k);
}
return;
}
```

Recursion Tree:

Let, No. of disk, n=3, source (i) =1, helper (j) =2, destination (k) =3



Write a C program of recursive algorithm for Tower of Hanoi:

```
#include<stdio.h>
void Hanoi(int n, int i, int j, int k);
void main()
{
        int n;
        int i=1;
        int j=2;
        int k=3;
        printf("\n Enter the value of n:=");
       scanf("%d",&n);
        Hanoi(n,1,2,3);
        printf("\n");
}
void Hanoi(int n, int i, int j, int k)
{
if (n==1)
{printf("\n% d ->%d", i,k);
}
else
{
Hanoi(n-1, i, k, j);
printf("\n% d ->%d", i, k);
Hanoi(n-1, j, i, k);
}
return;
}
```

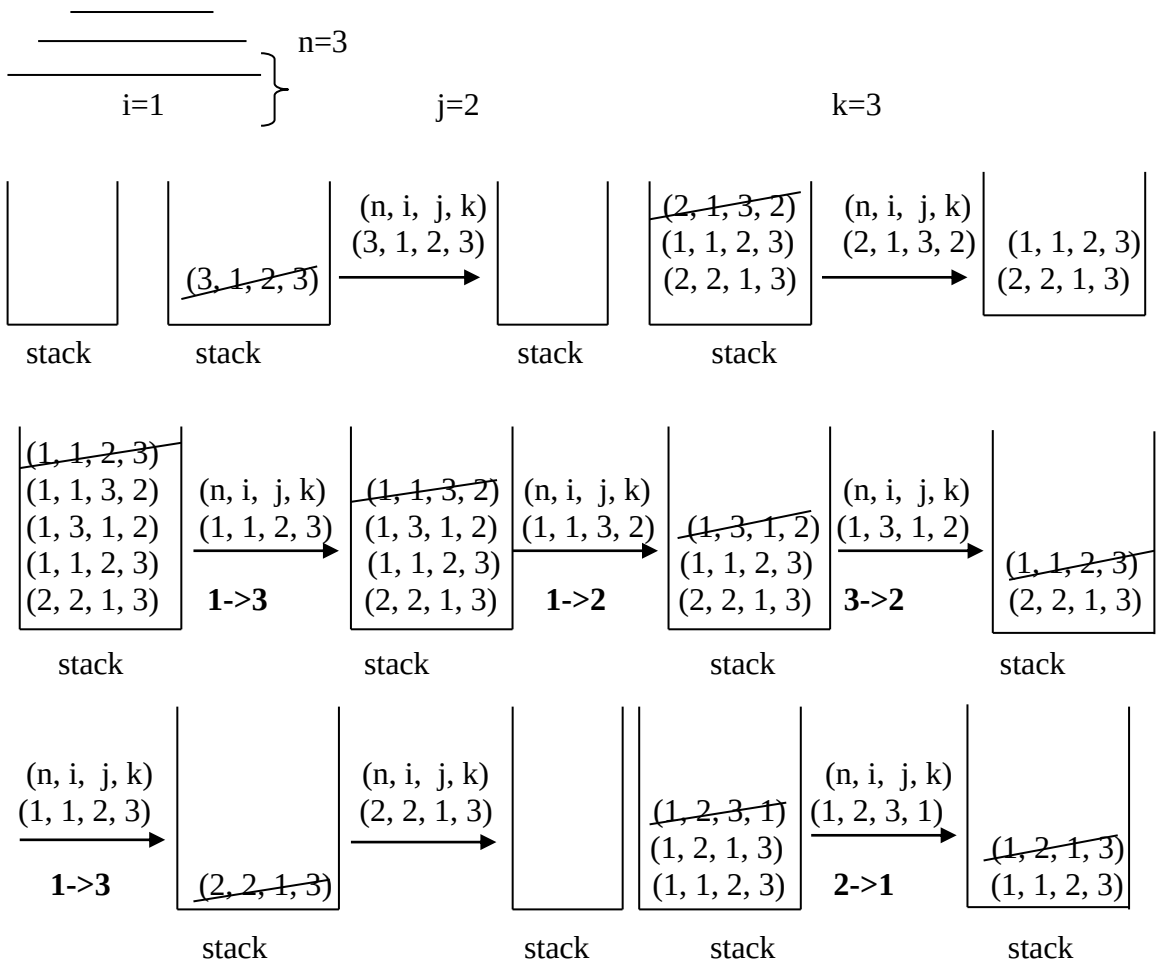
```
Output on monitor: Enter the value of n:=3
 1 ->3
 1 ->2
 3 ->2
 1 ->3
 2 ->1
 2 ->3
 1 ->3
```

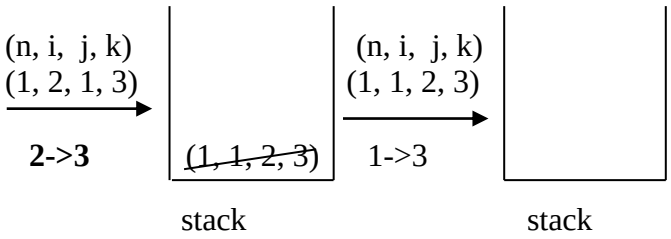
Iterative Algorithm for Tower of Hanio (Stack Algorithm)

```
stack<=(n,1,2,3)
while(stack!=empty) {
   (n, i, j, k)<=stack
    if((n=1)      {
       printf("\n% d ->%d" i, k);
     }
    else
     {
      stack<=(n-1, j, i, k)
      stack<=(1, i, j, k)
       stack<=(n-1, i, k, j)
     }
}
```

Mechanism:

Let three disks are organized in a source location (i) and need to move them into the destination (k).





Output sequence

1 ->3
1 ->2
3 ->2
1 ->3
2 ->1
2 ->3
1 ->3