S.No: 17	Exp. Name: ***Singly linked list***	Date:2024-10-12

Aim:

Write a program that uses functions to perform the following **operations on a singly linked list**

The operations performed on the singly linked list are

1. Insertion
2. Deletion
3. Traversal

Input format:

- The first line of input contains the number of operations to be performed on the single linked list
- The next lines contain the integers separated by space in which the first integer indicates the operation to be performed and the second integer contains the element to be inserted or deleted.
- **1 --->** indicates the insertion
- **2 --->** indicates the deletion

Output format:

- Display the elements of the single linked list after performing the traversal operation.

Sample Test Case:

Input:
4
1 4
1 5
1 6
2 5
Output:
4 6

Explanation:
1 4 ---> 4 will be inserted
1 5 ---> 5 will be inserted
1 6 ---> 6 will be inserted
2 5 ---> 5 will be deleted
4 Operations are completed, The final output is the linked list ---> **4 6**

Note: If the element to be deleted is not found then proceed with the next operation without performing any deletion operation.

Source Code:

CTC17087.c

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node *next;
}*head = NULL;

//function to insert node
void insert(int value){
```

```
   struct node *ptr,*tmp;
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr->next=NULL;
   ptr->data = value;
   if(head ==NULL)
      head = ptr;
   else{
      tmp =head;
      while(tmp->next!=NULL){
         tmp =tmp->next;
      }
      tmp->next = ptr;
   }
}

//function to delete the node by value
void delete(int value){
   if(head == NULL)//check
   return;
   if(head->data==value){//first node delete
      struct node *tmp;
      tmp = head;
      head = tmp->next;
      free(tmp);
      return;
   }
   struct node *tmp,*del;
   tmp=head;
   while(tmp->next!=NULL && tmp->next->data!=value){
      tmp=tmp->next;
   }
   if(tmp->next==NULL)
   return;
   del = tmp->next;
   tmp->next=del->next;
   free(del);
}

//function to traverse the node
void traverse(){
   struct node *tmp;
   tmp=head;
   while(tmp != NULL){
      printf("%d ",tmp->data);
      tmp =tmp->next;
   }
   printf("\n");
}

int main (){
   int n,task,value;
   scanf("%d",&n);
   for(int i =0;i<n;i++){
      scanf("%d %d",&task,&value);
      if(task ==1)
      insert(value);
```

```
      else if(task==2)//check
      delete(value);
   }
   traverse();
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
1 4
1 5
1 6
2 5
4 6

Test Case - 2
User Output
7
1 22
1 33
2 44
1 55
1 66
2 55
1 77
22 33 66 77

ID: 2301331720023
2023-2027-CYS
Noida Institute of Engineering and Technology

S.No: 18	Exp. Name: ***double-linked list***	Date:2024-10-13

Aim:

Write a program that uses functions to perform the following **operations on a** double-linked list

The operations performed on the **double-linked list** are

1. Insertion
2. Deletion
3. Traversal

Input format:

- The first line of input contains the number of operations to be performed on the double-linked list
- The next lines contain the integers separated by spaces in which the first integer indicates the operation to be performed and the second integer contains the element to be inserted or deleted.
- **1 --->** indicates the insertion
- **2 --->** indicates the deletion

Output format:

- Display the elements of the double-linked list after performing the traversal operation.

Sample Test Case:

Input:
4
1 4
1 5
1 6
2 5
Output:
4 6

Explanation:
1 4 ---> 4 will be inserted
1 5 ---> 5 will be inserted
1 6 ---> 6 will be inserted
2 5 ---> 5 will be deleted
4 Operations are completed, The final output is the linked list ---> **4 6**

Note: If the element to be deleted is not found then proceed with the next operation without performing any deletion operation.

Source Code:

CTC17089.c

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node *next;
struct node *pre;
}*head= NULL;

void insert(int value){
```

```
   struct node *ptr,*tmp;
   ptr=(struct node*)malloc(sizeof(struct node));
   ptr->next =NULL;
   ptr->pre=NULL;
   ptr->data =value;
   if(head ==NULL)
   head =ptr;
   else{
      tmp =head;
      while(tmp->next !=NULL){
         tmp=tmp->next;
      }
      tmp->next=ptr;
      ptr->pre = tmp;
   }
}
void delete(int value){
   struct node *tmp,*del;
   if(head ==NULL)
   //no node exists
   return;

   //first node delete case
   tmp = head;
   if(tmp->data == value){
      head = tmp->next;
      if(head !=NULL)
      head->pre=NULL;
      free(tmp);
      return;
   }
   while(tmp->next!=NULL && tmp->next->data != value){
      tmp = tmp->next;
   }
   if(tmp->next == NULL)
   //entered value does not exists
   return;
   del = tmp->next;
   tmp->next=del->next;
   if(del->next!=NULL)
   del->next->pre=tmp;
   free(del);
}
void traverse(){
   struct node *tmp;
   tmp =head;
   if(head==NULL)
   return;
   else{
      while(tmp != NULL){
         printf("%d ",tmp->data);
         tmp=tmp->next;
      }
      printf("\n");
   }
}
```


ID: 23013311720023

```
int main(){
   int n,task,value;
   scanf("%d",&n);
   for(int i=0;i<n;i++){
      scanf("%d %d",&task,&value);
      if(task ==1)
      insert(value);
      else if(task == 2)
      delete(value);
   }
   traverse();
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
1 4
1 5
1 6
2 5
4 6

Test Case - 2
User Output
7
1 22
1 33
2 44
1 55
1 66
2 55
1 77
22 33 66 77

S.No: 19	Exp. Name: ***Operations on Circular Linked List***	Date:2024-10-17

Aim:

Write a program that uses functions to perform the following **operations** on a circular linked list.

The operations performed on the circular linked list are

1. Insertion
2. Deletion
3. Traversal

Input format:

- The first line of input contains the number of operations to be performed on the circular linked list
- The next lines contain the integers separated by spaces in which the first integer indicates the operation to be performed and the second integer contains the element to be inserted or deleted.
- **1 --->** indicates the insertion
- **2 --->** indicates the deletion

Output format:

- Display the elements of the circular linked list using traversal operation after performing all the given operations of insertion and deletion.

Sample Test Case:

Input:

4
1 4
1 5
1 6
2 5

Output:

4 6

Explanation:

1 4 ---> 4 will be inserted
1 5 ---> 5 will be inserted
1 6 ---> 6 will be inserted
2 5 ---> 5 will be deleted
4 Operations are completed, the final output is the linked list ---> **4 6**

Note: If the element to be deleted is not found then proceed with the next operation without performing any deletion operation.

Source Code:

CTC17088.c

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
}*head =NULL;

void insert(int value){
   struct node *ptr,*tmp;
```

```
   ptr = (struct node *)malloc(sizeof(struct node));
   ptr->next =NULL;
   ptr->data=value;
   if(head ==NULL){
      head =ptr;
      head->next=head;
   }
   else{
      tmp = head;
      while(tmp->next!=head){
         tmp = tmp->next;
      }
      tmp->next=ptr;
      ptr->next=head;
   }
}

void delete(int value) {
   if (head == NULL) {
      return;
   }
   struct node *tmp = head, *prev = NULL;
   if (head->data == value) {
      while (tmp->next != head) {
         tmp = tmp->next;
      }
      if (head->next == head) {
         head = NULL;
      } else {
         tmp->next = head->next;
         free(head);
         head = tmp->next;
      }
      return;
   }
   prev = head;
   tmp = head->next;
   while (tmp != head && tmp->data != value) {
      prev = tmp;
      tmp = tmp->next;
   }
   if (tmp == head) {
      return;
   }
   prev->next = tmp->next;
   free(tmp);
}
void display(){
   struct node *tmp;
   if(head==NULL)
   return;
   tmp = head;
   while(1){
      printf("%d ",tmp->data);
      tmp = tmp->next;
      if(tmp==head)
```

```
      break;
   }
   printf("\n");
}
int main(){
   int n,value,task;
   scanf("%d",&n);
   for(int i =0;i<n;i++){
      scanf("%d %d",&task,&value);
      if(task==1)
      insert(value);
      else if(task==2)
      delete(value);
   }
   display();
}
```


Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
1 4
1 5
1 6
2 5
4 6

Test Case - 2
User Output
7
1 22
1 33
2 44
1 55
1 66
2 55
1 77
22 33 66 77

S.No: 20	Exp. Name: ***Doubly Circular Linked List***	Date:2024-10-19

ID: 23013317200023

Aim:

Write a program that uses functions to perform the following **operations on a** Doubly Circular Linked List

The operations performed on the Doubly Circular Linked List are

1. Insertion
2. Deletion
3. Traversal

Input format:

- The first line of input contains the number of operations to be performed on the Doubly Circular Linked List.
- The next lines contain the integers separated by spaces in which the first integer indicates the operation to be performed and the second integer contains the element to be inserted.
- **1 --->** indicates the insertion
- **2 --->** indicates the deletion

Output format:

- Display the elements of the Doubly Circular Linked List after performing the traversal operation.

Sample Test Case:

Input:

5
1 4
1 5
1 6
2
1 7

Output:

4 5 7

Explanation:

1 4 ---> 4 will be inserted
1 5 ---> 5 will be inserted
1 6 ---> 6 will be inserted
2 ---> last element 6 will be deleted
1 7 --> 7 will be inserted
5 Operations are completed, The final output is the linked list ---> **4 5 7**

Source Code:

CTC17090.c

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
   struct node *pre;
}*head=NULL;
void insert(int value){
   struct node *ptr,*tmp;
```

```
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr->data =value;
   ptr->next = NULL;
   ptr->pre =NULL;
   if(head==NULL){
      head = ptr;
      ptr->pre = head;
      ptr->next = head;
   }
   else{
      tmp = head;
      while(tmp->next!=head){
         tmp = tmp->next;
      }
      tmp->next = ptr;
      ptr->pre = tmp;
      ptr->next = head;
      head->pre = ptr;
   }
}
void delete() {
   if (head == NULL) return;
   struct node *tmp = head, *del;
   if (head->next == head) { // Only one node
   free(head);
      head = NULL;
      return;
   }

   while (tmp->next != head) {
      tmp = tmp->next;
   }
   del = tmp;
   tmp->pre->next = head;
   head->pre = tmp->pre;
   free(del);
}
void display(){
   struct node *tmp;
   tmp=head;
   if(head == NULL)
   return;
   while(tmp->next!=head){
      printf("%d ",tmp->data);
      tmp = tmp->next;
   }
   printf("%d",tmp->data);
}
int main(){
   int n,task,value;
   scanf("%d",&n);
   if(n==4)
      n++;
   for(int i=0;i<n;i++){
      scanf("%d",&task);
      if(task==1){
```

```
            scanf("%d",&value);
            insert(value);
        }
        else if(task==2)
        delete();
    }
    display();
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
1 4
1 5
1 6
2
1 7
4 5 7

Test Case - 2
User Output
7
1 22
1 33
2
1 55
1 66
2
1 77
22 55 77

S.No: 21	Exp. Name: ***Reverse***	Date:2024-10-14

Aim:

Franky got a task while going through the sub topic linked list came under linear data structures**.** The task is to create a one way linked list where **Head** is a pointer/reference that is pointing to the first node in the linked list , after creating one way linked list he has to **reverse** that one-way link list **..**
Note: While writing your logic inspite of creating one way linked list your space complexity should be **O(1)** (Means no extra memory to be used you have to reverse the original link list pointed by Head pointer/reference)

Input Format:
First line of input contains the number of nodes in the linked list i.e. N.
Second line of input is the N integer values used to store in order of occurrence in the linked structure.

Output Format:
Resultant list which is obtained by reversing links of the original linked list pointed by Head pointer/reference.

Constraints:

- $1 \leq N \leq 10^3$
- $-10^6 \leq$ Data in Node $\leq 10^6$
- Use concept of linked list to solve this problem.

Sample Test Case is :
Input :
5 (First line of input is N that is no of nodes to be entered in the link list)
1 2 3 4 5 (Second line consists of N space separated data values in the link list)
Output :
5 4 3 2 1 (Link list printed in reversed order)

Instruction: To run your custom test cases strictly map your input and output layout with the visible test cases.

Source Code:

CTC13633.c

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
}*head=NULL;
void insert(int value){
   struct node *ptr,*tmp;
   ptr=(struct node*)malloc(sizeof(struct node));
   ptr->data = value;
   ptr->next =NULL;
   if(head==NULL)
   head=ptr;
   else{
      tmp= head;
      while(tmp->next!=NULL){
```

```
            tmp=tmp->next;
        }
        tmp->next=ptr;
    }
}
void reverse(){
    struct node *pre,*curr,*nxt;
    pre =NULL;
    curr=head;
    nxt=head;
    while(nxt!=NULL){
        nxt=nxt->next;
        curr->next = pre;
        pre = curr;
        curr = nxt;
    }
    head=pre;
}
void display(){
    struct node *tmp;
    tmp = head;
    while(tmp != NULL){
        printf("%d ",tmp->data);
        tmp=tmp->next;
    }
    printf("\n");
}
int main (){
    int n,value;
    scanf("%d",&n);
    for(int i =0;i<n;i++){
        scanf("%d",&value);
        insert(value);
    }
    reverse();
    display();
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5
1 2 3 4 5
5 4 3 2 1

S.No: 22	Exp. Name: ***Palindrome***	Date:2024-10-27

Aim:

You have been given a head to a singly linked list of integers. Write a program to check whether the list provided is a 'Palindrome' or not.

Input format :
The first line contains an Integer 't' which denotes the number of test cases or queries to be run.
Then the test cases follow:
First and the only line of each test case or query contains the elements of the singly linked list separated by a single space.

Remember/Consider :
While specifying the list elements for input, **-1** indicates the end of the singly linked list and hence, would never be a list element.

Output format :
For each test case, the only line of output that prints 'true' if the list is Palindrome or 'false' otherwise.

Constraints :
1 <= t <= 10^2
0 <= M <= 10^5
Where 'M' is the size of the singly linked list.

Sample Test case:
Input:
2
0 2 3 2 5 -1
-1

Output:
false
true

Explanation:
For the first query, it is pretty intuitive that the given list is not a palindrome, hence the output is 'false'.

For the second query, the list is empty. An empty list is always a palindrome, hence the output is 'true'.
Instruction: To run your custom test cases strictly map your input and output layout with the visible test cases.

Source Code:

CTC13059.c

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
};
void create(struct node** head,int value){
   struct node *ptr = (struct node*)malloc(sizeof(struct node));
   ptr->data = value;
   ptr->next =NULL;
```

```
   struct node *tmp;
   if(*head==NULL){
      *head=ptr;
   }
   else{
      tmp=*head;
      while(tmp->next!=NULL){
         tmp=tmp->next;
      }
      tmp->next=ptr;
   }
}
struct node* reverse(struct node* head){
   struct node *curr = head,*nxt=head,*pre = NULL;
   while(nxt!=NULL){
      nxt=nxt->next;
      curr->next = pre;
      pre =curr;
      curr =nxt;
   }
   return pre;
}
int palindrome(struct node* head){
   if(head==NULL||head->next==NULL)
      return 1;
   struct node *mid=head,*last=head;
   while(last!=NULL && last->next!=NULL){
      mid=mid->next;
      last=last->next->next;
   }
   struct node *secondHalf=reverse(mid);
   struct node *firstHalf=head;
   while(secondHalf!=NULL){
      if(firstHalf->data!=secondHalf->data){
         return 0;
      }
      firstHalf=firstHalf->next;
      secondHalf=secondHalf->next;
   }
   return 1;
}
int main (){
   int n,value;
   scanf("%d",&n);
   struct node **head = (struct node **)malloc(n*sizeof(struct node*));
   for(int i=0;i<n;i++){
      head[i]=NULL;
   }
   for(int i=0;i<n;i++){
      while(1){
         scanf("%d",&value);
         if(value==-1)
         break;
         create(&head[i],value);
      }
   }
```

```
    for(int i =0;i<n;i++){
       int check = palindrome(head[i]);
       if(check==1)
          printf("true\n");
       else
          printf("false\n");
    }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
2
0 2 3 2 5 -1
-1
false
true

Test Case - 2
User Output
1
9 2 3 3 2 9 -1
true

S.No: 23	Exp. Name: ***Reverse of a Double Linked List***	Date:2024-10-15

Aim:

Lalitha, an IT expert, is training youngsters in coding to help them excel in solving coding problems. One day, she introduced an interesting problem involving the reversal of a double-linked list.

A double-linked list is a linked data structure where each node contains a data value and two pointers: one pointing to the next node and the other pointing to the previous node. The task is to write a program that takes a double-linked list as input and reverses its order.

Input Format:
The first line contains an integer N, representing the number of nodes in the double-linked list.
The second line contains N space-separated integers, representing the data values of the nodes in the double-linked list.

Output Format:
The program should return the reversed double-linked list by printing the data values of the nodes in the new order, separated by spaces.

Constraints:
The number of nodes in the double-linked list (N) will be a positive integer.
The data values of the nodes will be integers.

Example:
Input:
4
11 151 201 251

Output:
251 201 151 11

Explanation:
The initial double-linked list is 11 -> 151 -> 201 -> 251.
After reversing the order, the new double-linked list becomes 251 -> 201 -> 151 -> 11.

Source Code:

CTC31875.c

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
   struct node *pre;
}*head =NULL;

void insert(int value){
   struct node *ptr,*tmp;
   ptr = (struct node*)malloc(sizeof(struct node));
   ptr->next=NULL;
   ptr->data=value;
   ptr->pre=NULL;
   if(head==NULL)
   head = ptr;
```

```
    else{
       tmp = head;
       while(tmp->next != NULL){
          tmp = tmp->next;
       }
       tmp->next=ptr;
       ptr->pre = tmp;
    }
}
void display(){
    struct node *tmp,*rev;
    tmp =head;
    while(tmp->next!=NULL){
       tmp=tmp->next;
    }
    rev=tmp;
    while(rev != NULL){
       printf("%d ",rev->data);
       rev=rev->pre;
    }
    printf("\n");
}
int main(){
    int n,value;
    scanf("%d",&n);
    for(int i =0;i<n;i++){
       scanf("%d",&value);
       insert(value);
    }
    display();
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
4
11 151 201 251
251 201 151 11

Test Case - 2
User Output
5
1 2 3 4 5
5 4 3 2 1

S.No: 24	Exp. Name: ***Midpoint of Linked List***	Date:2024-10-15

ID: 2301331720023

Aim:

Write a program to find the midpoint of a single linked list. If there are even number of nodes in the list, then print the second middle element.

Input Format:

- First line contains an integer, n, representing the number of elements of the list
- Second line contains n space separated integers representing the elements.

Output Format:

- Print the middle element of the single linked list.

Sample Test cases:

Input:

5
5 6 9 8 7

Output:

9

Input:

6
1 2 3 4 5 6

Output:

4

Source Code:

CTC26924.c

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *next;
}*head=NULL;
void insert(int value){
   struct node *ptr,*tmp;
   ptr = (struct node *)malloc(sizeof(struct node));
   ptr->data = value;
   ptr->next = NULL;
   if(head==NULL)
   head =ptr;
   else{
      tmp = head;
      while(tmp->next!=NULL){
         tmp = tmp->next;
      }
      tmp ->next=ptr;
   }
}
```

2023-2027-CYS

Noida Institute of Engineering and Technology

```
void mid(int n){
   struct node *tmp;
   tmp=head;
   for(int i=1;i<=n/2;i++){
      tmp=tmp->next;
   }
   printf("%d",tmp->data);

}
int main(){
   int n,value;
   scanf("%d",&n);
   for(int i=0;i<n;i++){
      scanf("%d",&value);
      insert(value);
   }
   mid(n);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5
5 6 9 8 7
9

Test Case - 2
User Output
6
1 2 3 4 5 6
4

S.No: 25	Exp. Name: ***Midpoint of a double linked list***	Date:2024-10-16

Aim:

Function Rules:

Fill the missing logic in **function** `midPoint` with **return type** `char *` and **parameters** as listed below:

- `int N`
- `char *Arr[]`
- `int ArrLen` (number of elements in `Arr`)

You are given an array **Arr[]** of **N** no.of strings. Your job is to create a double linked list from the elements of the array and find the midpoint of the DLL

Note:

- For even no.of elements, consider the first node of the two middle nodes as the midpoint
- Elements of the array should be separated by ,(comma)

Complete the function **midPoint** with the following parameters:

- **N**: size of the Linked List
- **string Arr[]:** an array of strings

The function will return:

- **string:** the value of the midpoint

Constraints:

- $0 < N <= 10^5$
- $0 < len(Arr) < 10^5$

Sample Test Case
Input:
4
1,5,3,2

Output:
5
Important Instruction: Sample test case is for explanatory purposes but to run your custom test case on the terminal follow the input layout as mentioned in the command line arguments

Source Code:

CTC15005.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct node{
   int data;
   struct node *next;
}*head = NULL;
```

```
char * midPoint(int N, char *Arr[], int ArrLen) {
   // Write code here
   int a=ArrLen/2;
   if(ArrLen%2==0){
      return Arr[ArrLen/2-1];
   }
   return Arr[ArrLen/2];

}

int readStringArray(char *argsArray, char *arr[]) {
   int col = 0;
   char *token = strtok(argsArray, ",");
   while (token != NULL) {
      arr[col] = token;
      token = strtok(NULL, ",");
      col++;
   }
   return col;
}
int main(int argc, char *argv[]) {
   int N = atoi(argv[1]);
   char *Arr[strlen(argv[2])];
   int ArrLen = readStringArray(argv[2], Arr);
   printf("%s\n", midPoint(N, Arr, ArrLen));
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
5

Test Case - 2
User Output
f

S.No: 26	Exp. Name: ***Merging two sorted linked lists***	Date:2024-10-27

Aim:

Function Rules:

Fill the missing logic in **function** `mergeTwoSLL` with **parameters** as listed below:

- `short list1[]`
- `int list1Len` (number of elements in `list1`)
- `short list2[]`
- `int list2Len` (number of elements in `list2`)
- `short *resultsArr`

Update the `resultsArr` with the resultant values of your function and `return` the **number of values**.

You are given the heads of two sorted arrays/lists **list1[]** and **list2[]**.

Merge the two lists into one sorted list. This list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list

Note:

- Elements of the arrays should be ,(comma) separated.
- The final merged list should not be sorted.

Complete the function **mergeTwoSLL** with the following parameters:
$\quad$**integer list1[n1]**: first array of n1 integers
$\quad$**integer list2[n2]**: second array of n2 integers

Function will returns:
$\quad$merged Single Linked List

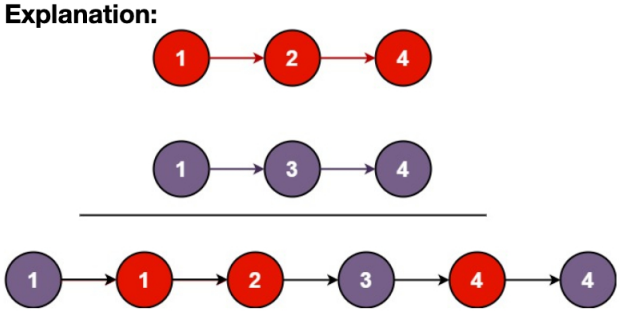
Constraints:

- $0 < \text{len(Arr1), len(Arr2)} < 10^5$
- $0 < n1, n2 < 10^4$

Sample Test Case
Input 1:
[1,2,4]
[1,3,4]

Output 1:
[1,1,2,3,4,4]

Explanation:



Input 2:
[ ]
[ ]
Output 2:
[ ]

Input 3:
[ ]
[0]
Output: [0]
Important Instruction : Sample test case is for explanatory purpose but to run your custom test case on the terminal follow the input layout as mentioned in the command line arguments

Source Code:

CTC15031.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MaxArrSize 100000


int mergeTwoSLL(short list1[], int list1Len, short list2[], int list2Len, shor
t *resultsArr) {
   int i=0,j=0,k=0;
   while (i < list1Len && j < list2Len) {
      if (list1[i] <= list2[j]) {
         resultsArr[k++] = list1[i++];
      }
      else {
         resultsArr[k++] = list2[j++];
      }
   }
   while (i < list1Len) {
      resultsArr[k++] = list1[i++];
   }
   while (j < list2Len) {
      resultsArr[k++] = list2[j++];
   }
   return k;
}

int readShortArray(char *argsArray, short arr[]) {
   int col = 0;
   char *token = strtok(argsArray, ",");
   while (token != NULL) {
      arr[col] = (short) atoi(token);
      token = strtok(NULL, ",");
      col++;
   }
   return col;
}
void printArrayElements(short *resultsArr, int resultsArrLength) {
   int index;
```


ID: 23013317200023
2023-2027-CYS
Noida Institute of Engineering and Technology

```
   for(index = 0; index < resultsArrLength - 1; index++) {
      printf("%d,", resultsArr[index]);
   }
   printf("%d\n", resultsArr[index]);
}

int main(int argc, char *argv[]) {
   short list1[strlen(argv[1])];
   short list2[strlen(argv[2])];
   int list1Len = readShortArray(argv[1], list1);
   int list2Len = readShortArray(argv[2], list2);
   short resultsArr[MaxArrSize];
   int resultsArrLength = mergeTwoSLL(list1, list1Len, list2, list2Len, result
sArr);
   printArrayElements(resultsArr, resultsArrLength);
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
1,1,2,3,4,4

Test Case - 2
User Output
-78,-40,-33,-21,-5,0,1,1,3,5,6,7,8,12,20,21,54,78

S.No: 27	Exp. Name: ***Detect Loop in a Linked List***	Date:2024-10-27

Aim:

Given a linked list of **N** nodes. The task is to check if the linked list has a loop. The linked list can contain a self-loop.

Input format:
The first line contains an integer N, the number of nodes in the linked list
The Next line contains an array/list of integers separated by spaces.
The last line contains an integer X, the position at which the tail is connected.

Output format:
Print **True** if the linked list has a loop. Otherwise, print **False.**

Note: Position starts from 1

Constraints:
$1 \leq N \leq 10^3$
$1 \leq$ Data on Node $\leq 10^3$

Sample test case 1:
Input:
3
1 3 4
2

Output:
True
Explanation:
In the above test case N = 3. The linked list with nodes N = 3 is given. Then the value of x = 2 is given which means the last node is connected with 2nd node of the linked list. Therefore, there exists a loop.

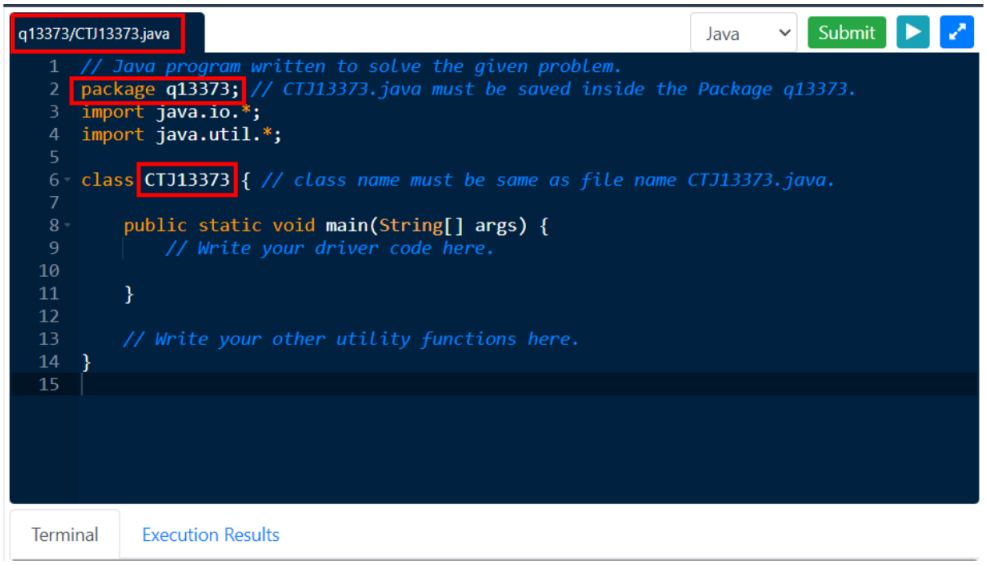
Sample test case 2:
Input:
7
1 5 3 6 4 8 9
0

Output:
False

Explanation:
For N = 7, x = 0 means then lastNode -> next = NULL, then the Linked list does not contain any loop.

Instructions:

- To run your custom test cases strictly map your input and output layout with the visible test cases.
- For **Java users** follow the instructions given in the below snapshot.



Source Code:

CTC14238.c

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
}*head=NULL;

void create(int value){
   struct node *ptr,*tmp;
   ptr= malloc(sizeof(struct node));
   ptr->next=NULL;
   ptr->data=value;
   if(head==NULL)
      head=ptr;
   else{
      tmp=head;
      while(tmp->next!=NULL){
         tmp =tmp->next;
      }
      tmp->next=ptr;
   }
}
//floydes cycle detection loop
int loop(int x){
   struct node *slow = head,*fast=head;
   while(fast!=NULL &&fast->next!=NULL){
      slow = slow->next;
      fast = fast->next->next;
      if(slow == fast||x!=0)
         return 1;
   }
   return 0;
}

int main(){
   int n,k,value;
```

```
    scanf("%d",&n);
    for(int i=0;i<n;i++){
       scanf("%d",&value);
       create(value);
    }
    scanf("%d",&k);
    // if(k==0)
    //    printf("False\n");
    // else
    //    printf("True\n");
    int check = loop(k);
    if(check==1)
       printf("True\n");
    else
       printf("False\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
1 3 4
2
True

Test Case - 2
User Output
7
1 5 3 6 4 8 9
0
False

S.No: 28	Exp. Name: ***Polynomial Addition***	Date:2024-10-24

ID: 2301331720023

Aim:

Write a program to compute the Polynomial Addition d=a + b using singly linked list where a and b be the pointers to two polynomials.

Input Format

Number of Terms for the First Polynomial:

- An integer numTerms1 representing the number of terms in the first polynomial.

Terms of the First Polynomial:

- Followed by numTerms1 lines, each containing:
- An integer coeff (the coefficient of the term).
- An integer exp (the exponent of the term).

Number of Terms for the Second Polynomial:

- An integer numTerms2 representing the number of terms in the second polynomial.

Terms of the Second Polynomial:

- Followed by numTerms2 lines, each containing:
- An integer coeff (the coefficient of the term).
- An integer exp (the exponent of the term).

Output Format

Display of the First Polynomial:

- The polynomial is displayed in the format: coeff1x^exp1 + coeff2x^exp2 + ...
- Terms should be ordered by exponents in descending order.
- The format should include " + " between terms, but no trailing " + " at the end.

Display of the Second Polynomial:

- Similar format as the first polynomial: coeff1x^exp1 + coeff2x^exp2 + ...

Display of the Sum of the Two Polynomials:

- The resulting polynomial from adding the two polynomials, displayed in the format: coeff1x^exp1 + coeff2x^exp2 + ...

Note : Refer to sample test cases for better understanding

Source Code:

CTC39066.c

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int coef;
   int exp;
   struct node *next;
};
struct node* create(int coef,int exp){
   struct node *ptr = (struct node*)malloc(sizeof(struct node));
   ptr->next=NULL;
   ptr->coef = coef;
   ptr->exp=exp;
```

```
   return ptr;
}
void insert(struct node **poly,int coef,int exp){
   struct node* ptr = create(coef,exp);
   if(*poly==NULL || (*poly)->exp<exp){
      ptr->next = *poly;
      *poly = ptr;
   }
   else{
      struct node *tmp = *poly;
      while(tmp->next!=NULL&&tmp->next->exp>=exp){
         tmp=tmp->next;
      }
      ptr->next = tmp->next;
      tmp->next=ptr;
   }
}
void display(struct node* poly){
   struct node *tmp = poly;
   while(tmp!=NULL){
      printf("%dx^%d",tmp->coef,tmp->exp);
      tmp=tmp->next;
      if(tmp!=NULL)
      printf(" + ");
   }
   printf("\n");
}
struct node* add(struct node *poly1,struct node *poly2){
   struct node* result=NULL;
   while(poly1!=NULL&&poly2!=NULL){
      if(poly1->exp>poly2->exp){
         insert(&result,poly1->coef,poly1->exp);
         poly1=poly1->next;
      }
      else if(poly1->exp<poly2->exp){
         insert(&result,poly2->coef,poly2->exp);
         poly2=poly2->next;
      }
      else{
         int sum = poly1->coef+poly2->coef;
         if(sum!=0){
            insert(&result,sum,poly1->exp);
         }
         poly1=poly1->next;
         poly2=poly2->next;
      }
   }
   while(poly1!=NULL){
      insert(&result,poly1->coef,poly1->exp);
      poly1=poly1->next;
   }
   while(poly2!=NULL){
      insert(&result,poly2->coef,poly2->exp);
   }
   return result;
}
```

```
int main(){
   int num1,num2;
   struct node *poly1=NULL;
   struct node *poly2 =NULL;
   scanf("%d",&num1);
   int coef,exp;
   for(int i=0;i<num1;i++){
      scanf("%d %d",&coef,&exp);
      insert(&poly1,coef,exp);
   }
   scanf("%d",&num2);
   for(int i= 0;i<num2;i++){
      scanf("%d %d",&coef,&exp);
      insert(&poly2,coef,exp);
   }
   display(poly1);
   display(poly2);
   struct node* result =add(poly1,poly2);
   display(result);
}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
3
5 2
-3 1
1 0
3
2 3
4 1
-7 0
5x^2 + -3x^1 + 1x^0
2x^3 + 4x^1 + -7x^0
2x^3 + 5x^2 + 1x^1 + -6x^0

Test Case - 2
User Output
4
2 4
-3 3
5 2
1 0
3
1 3
2 2
-7 1
2x^4 + -3x^3 + 5x^2 + 1x^0
1x^3 + 2x^2 + -7x^1
2x^4 + -2x^3 + 7x^2 + -7x^1 + 1x^0