Ex no: 1.1(a)	PROGRAM TO FIND THE NUMBER OF PAIRS OF INTEGERS IN THE ARRAY
Date:	

Aim

To write a C program to find the number of pairs of integers in the array whose sum is equal to sum.

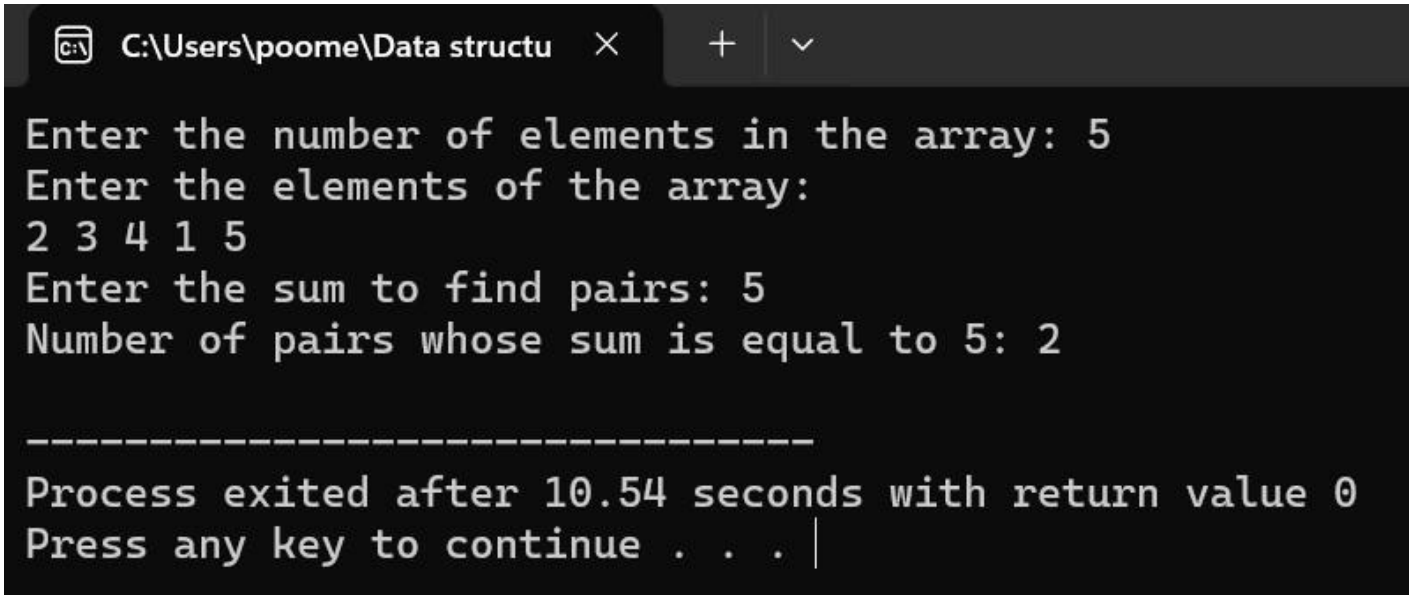
Pseudo Code

```
BEGIN
    Declare a[N], Sum,i,j
    Set Pair count =0
    Get the values of N, a [N], Sum
    If a [i]+ a[j] = = Sum
        Pair Count ++
    ENDIF
    Print Pair count
END
```

Source Code

```
#include <stdio.h>
int main()
{
    int N, sum, pairCount = 0;
    printf("Enter the number of elements in the array: ");
scanf("%d", &N);    int array[N];    int i,j;
    printf("Enter the elements of the array:\n");
for (i = 0; i < N; i++)
    {
        scanf("%d", &array[i]);
    }
    printf("Enter the sum to find pairs: ");
scanf("%d", &sum);
    for (i = 0; i < N - 1; i++)
    {
        for (j = i + 1; j < N; j++)
        {
            if (array[i] + array[j] == sum)
            {
                pairCount++;
            }
        }
    }
    printf("Number of pairs whose sum is equal to %d: %d\n", sum, pairCount);
return 0;
}
```

Output



Result

Thus the program to find the number of pairs of integers in the array whose sum is equal to sum was successfully executed and the output was verified.

Ex No:1.1(b)	PROGRAM TO FIND THE MAJORITY ELEMENT IN THE ARRAY
Date:	

Aim

To write a C program to find the majority element in the array.

Pseudo Code

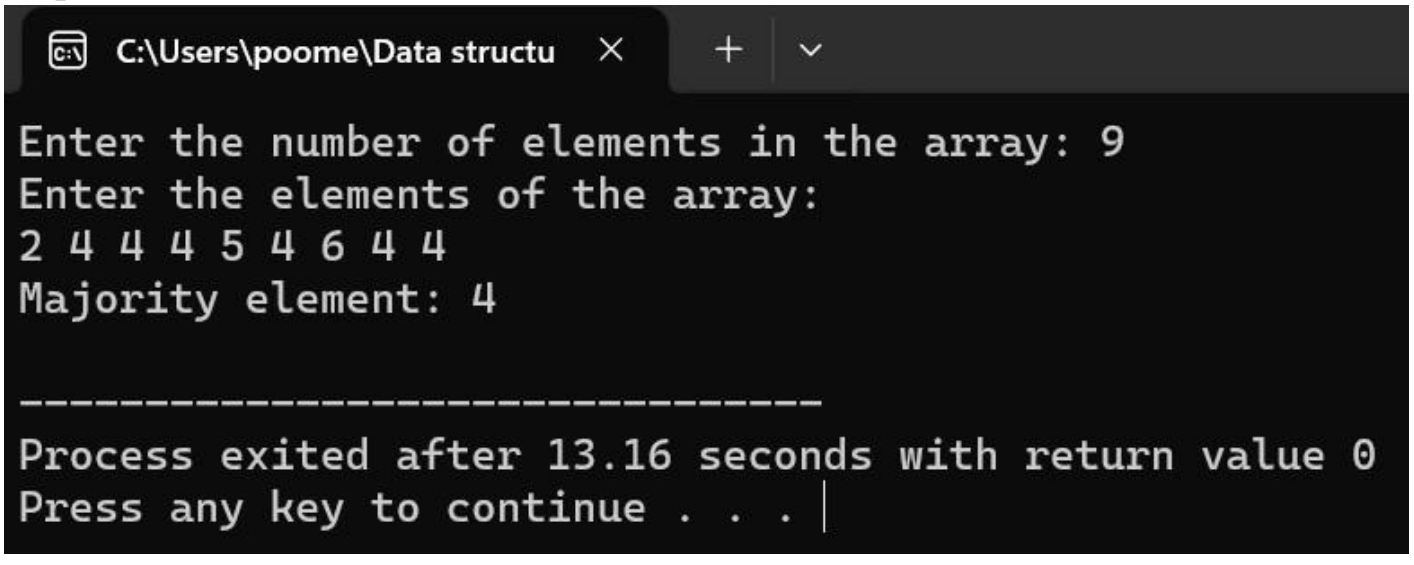
```
BEGIN
    Declare a [N], majority.
    Set count=0
    Get the value of N. a[N]
    If a[i]= majority
        Count ++
    If Count> N/2
        Print majority
    Else
        Print No majority
END
```

Source Code

```
#include <stdio.h> int
main()
{
    int N, majorityElement, count = 0;
printf("Enter the number of elements in the array: ");
    scanf("%d",
&N);        int
array[N];    int i;
    printf("Enter the elements of the array:\n");
for (i = 0; i < N; i++)
    {
        scanf("%d", &array[i]);
    }
    majorityElement = array[0];
count = 1;
    for (i = 1; i < N; i++)
    {
        if (array[i] == majorityElement)
        {
```

```
                        count++;
                }
                else
                {
                        count--;
        if (count == 0)
                        {
                                majorityElement = array[i];
                count = 1;
                        }
                }
}       count =
0;
        for (i = 0; i < N; i++)
        {
                if (array[i] == majorityElement)
                {
                        count++;
                }
        }
        if (count > N / 2)
        {
                printf("Majority element: %d\n", majorityElement);
        }
        else
        {
                printf("No majority element\n");
        }
        return 0;
}
```

Output



Result Thus the program to find the majority element was successfully executed and the output was verified.

Ex No:1.1(c)	**PRORAM TO FIND THE MISSING NUMBER FROM THE FIRST N INTEGERS**
Date:	

Aim

To write a C program to find the missing number from the first N integers.

Pseudo Code

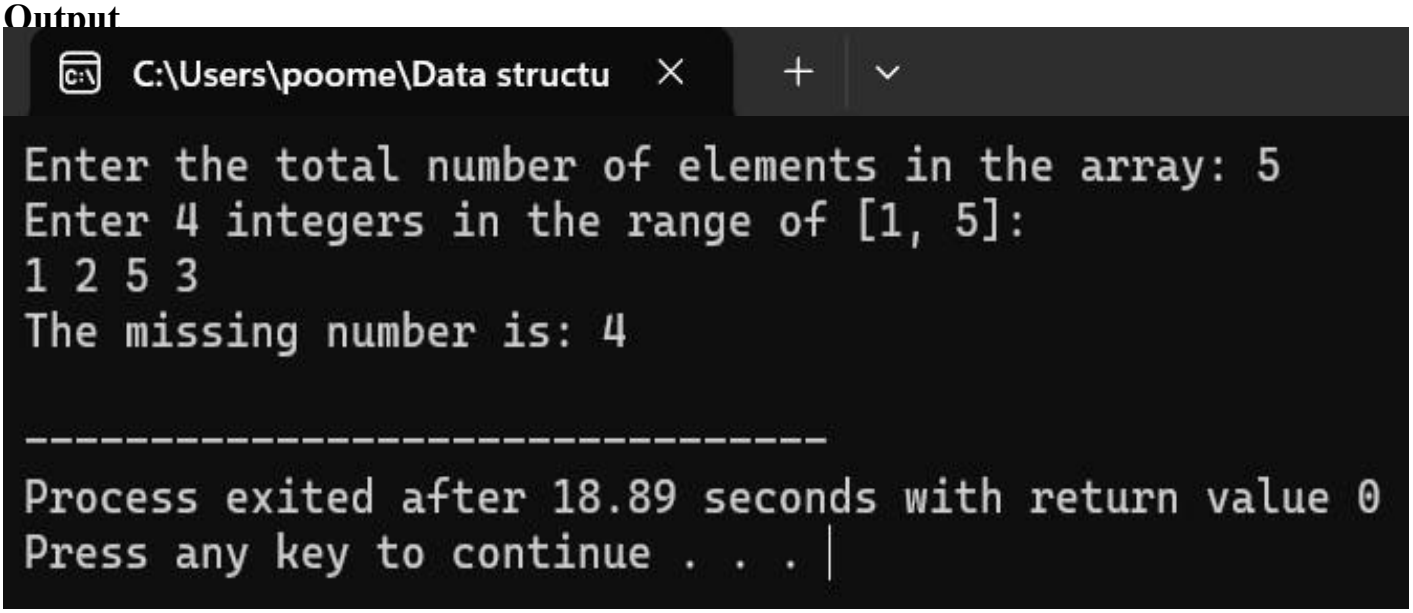
BEGIN
Declare N,expected sum, arr[N], missing
Set actual sum=0
Get the values of N,expected scam
Calculate expected sum = (N*(N+1))/2
Calculate actual Sum
Calculate Missing number = expected sum -actual sum
Print missing number
END

Source Code

```
#include <stdio.h> int
main()
{
        int N, expectedSum, actualSum = 0, missingNumber;
        printf("Enter the total number of elements in the array:
"); scanf("%d", &N);       int array[N - 1];          int i;
```

```
        printf("Enter %d integers in the range of [1, %d]:\n", N - 1, N);
for (i = 0; i < N - 1; i++)
        {
                scanf("%d", &array[i]);
actualSum += array[i];
        }
        expectedSum = (N * (N + 1)) / 2;      missingNumber =
expectedSum - actualSum;              printf("The missing
number is: %d\n", missingNumber);             return 0;
}
```

Output



Result

Thus the program to find the missing number from the first N integers was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:1.2(a)	**PROGRAM TO REVERSE THE VOWELS FOR THE GIVEN STRING**
Date:	

Aim

To write a C orogram to reverse the vowels for the given string.

Pseudo Code

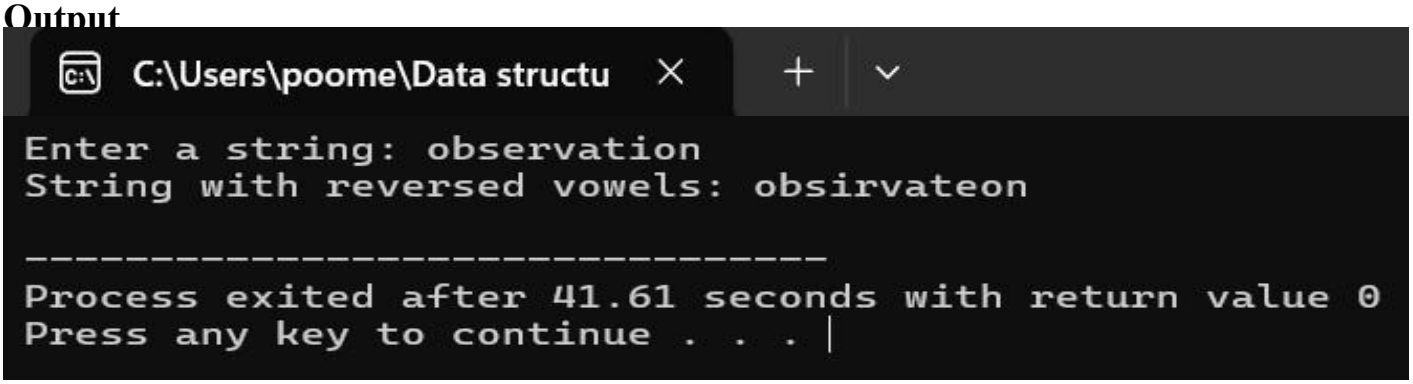
```
BEGIN
    Declare str[100]
    Get input from user
    If ch== a ||ch == e ||ch ==i ||ch==o ||ch==u|| ch== A ||ch == E ||ch ==I ||ch==O ||ch==U
        Push stack
    If ch== vowel
        Pop the top element
    Print the modified string
END
```

Source Code

```
#include <stdio.h>
#include <string.h> int
isVowel(char c)
{
	switch(c)
	{
		case 'a':
case 'e':		case 'i':
	case 'o':
case 'u':		case 'A':
	case 'E':
case 'I':		case 'O':
	case 'U':
return 1;		default:
		return 0;
	}
}
void reverseString(char str[])
{
	int length = strlen(str);
	int i;
	for (i = 0; i < length / 2; i++)
	{
		char temp = str[i];
		str[i] = str[length - i - 1];
		str[length - i - 1] = temp;
	}
}
void reverseVowels(char str[])
{
	int length =
strlen(str);	int start = 0;
int end = length - 1;
	while (start < end)
	{
		while (start < end && !isVowel(str[start]))
		{
			start++;
		}
		while (start < end && !isVowel(str[end]))
		{
			end--;
		}
		char temp = str[start];
str[start] = str[end];
```

```
                str[end] = temp;
start++;
                end--;
        }
}
int main()
{
        char str[100];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
        str[strcspn(str, "\n")] = '\0';
reverseVowels(str);
        printf("String with reversed vowels: %s\n", str);
return 0;
}
```

Output



Result

Thus the program to find the missing number from the first N integers was successfully executed and the output was verified.

Ex No:1.2(b)	**PROGRAM TO FIND THE VALUE OF Nth FIBONACCI SERIES**
Date:	

Aim

To write a C program to find the value of Nth fibonacci series.

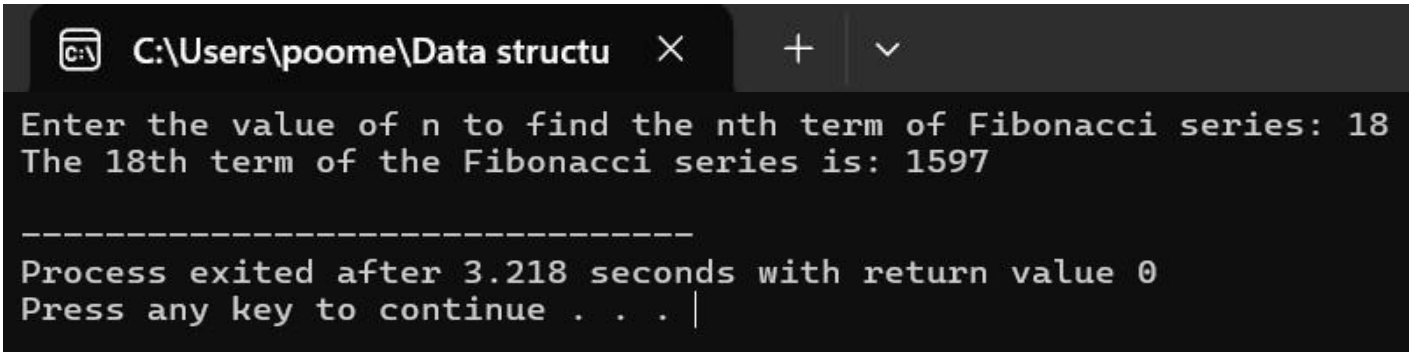
Pseudo Code

```
BEGIN
        Declare function fibonacci(n)
        If n==0
return 0S               If
n==1                    return 1
Else
                        return fibonacci (n-1) + fibonacci (n-2)
        Get n
        Call the fibonacci function
        Print result
END
```

Source Code

```
#include <stdio.h> int
fibonacci(int n)
{
        if (n == 0)
                return 0;
        else if (n == 1)
                return 1;
else
                return fibonacci(n-1) + fibonacci(n-2);
} int
main()
{
        int n, result;
        printf("Enter the value of n to find the nth term of Fibonacci series: ");
scanf("%d", &n);
        if (n < 0)
        {
                printf("Invalid input! n should be a non-negative integer.\n");
        }
        else
        {
                result = fibonacci(n-1);
                printf("The %dth term of the Fibonacci series is: %d\n", n, result);
        }
        return 0;
}
```

Output



Result

Thus the program to find the value of Nth fibonacci series was successfully executed and the output was verified.

Ex No:1.2(c)	PROGRAM TO REPLACE THE OCCURENCE OF THE CHARACTERS
Date:	

Aim

To write a C program to replace the occurence of the characters.

Pseudo Code

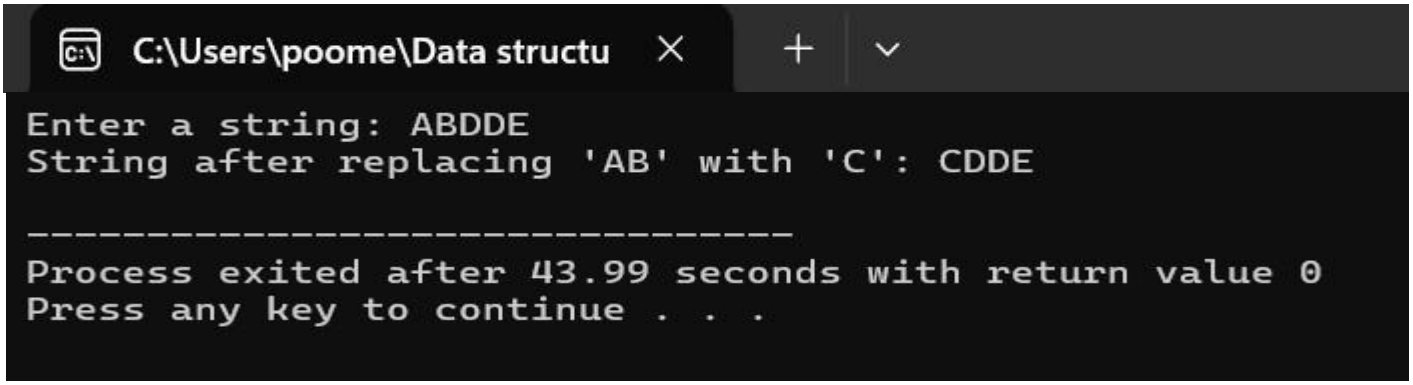
```
BEGIN
    Declare str [100]
    While i<length
        If str[i] == A and str[i]==B
            Append c to new-str
        Else
            Append str[i] to new-str
            i++
        Endif
    End while
    Display new-str
END
```

Source Code

```
#include <stdio.h> #include
<string.h> void
replaceABwithC(char str[])
{
    int length = strlen(str);
    int i;
    int j = 0;
    for (i = 0; i < length; i++)
    {
        if (str[i] == 'A' && str[i + 1] == 'B')
        {
            str[j] = 'C';
            i++;
    }
        else
        {
            str[j] = str[i];
        }
        j++;
    }
    str[j] = '\0';
} int
main()
{
```

```
        char str[100];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
str[strcspn(str, "\n")] = '\0';
        replaceABwithC(str);
        printf("String after replacing 'AB' with 'C': %s\n", str);
        return 0;
}
```

Output



Result

Thus the program to replace the occurence of the characters was successfully executed and the output was verified.

ALGORITHM	15	

PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:1.3(a)	**PROGRAM TO IMPLEMENT THE STACK**
Date:	

Aim

To write a C program to implement the stack.

Pseudo Code

BEGIN

Define a Stack Structure for each node in the
Detine a structure for the special Stack
Implement Push() operation
Implement Pop() operation
Implement isEmpty() operation
Implement isFull() operation
Implement getMin() operation

END

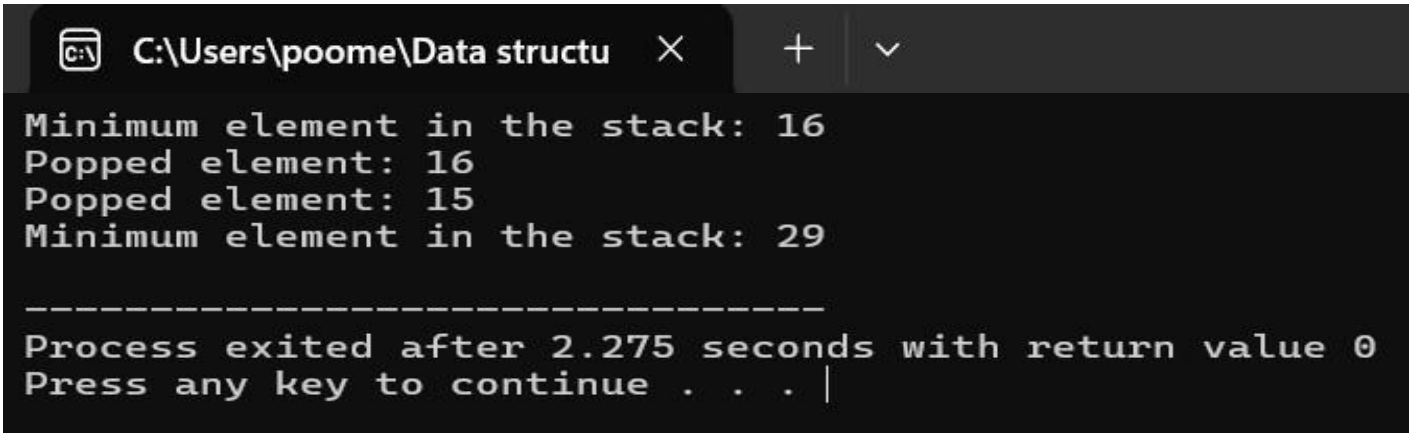
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h> struct
StackNode
{
    int data;
    struct StackNode* next;
};
struct StackNode* newNode(int data)
{
    struct StackNode* stackNode = (struct StackNode*)malloc(sizeof(struct
StackNode));   stackNode->data = data;        stackNode->next = NULL;
    return stackNode;
}
struct Stack
{
    struct StackNode* top;
```

```
}; struct Stack*
createStack()
{  struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
        stack->top = NULL;
        return stack;
}
int isEmpty(struct Stack* stack)
{
        return stack->top == NULL;
} void push(struct Stack** stack, int
data)
{
        struct StackNode* stackNode = newNode(data);      stackNode-
>next = (*stack)->top;
        (*stack)->top = stackNode;
}
int pop(struct Stack** stack)
{
        if (isEmpty(*stack))
        {
                printf("Stack is empty\n");
return INT_MIN;
        }
        struct StackNode* temp = (*stack)->top;
int popped = temp->data;
        (*stack)->top = (*stack)->top->next;
        free(temp);
return popped;
}
int peek(struct Stack* stack)
{
        if (isEmpty(stack))
        {
                printf("Stack is empty\n");
return INT_MIN;
        }
        return stack->top->data;
}
int getMin(struct Stack* stack)
{
        if (isEmpty(stack))
        {
                printf("Stack is empty\n");
return INT_MIN;
        }
        return peek(stack);
```

```
} int
main()
{
        struct Stack* stack =
createStack();          push(&stack, 18);
push(&stack, 19);       push(&stack, 29);
push(&stack, 15);       push(&stack, 16);
        printf("Minimum element in the stack: %d\n",
getMin(stack));         printf("Popped element: %d\n", pop(&stack));
printf("Popped element: %d\n", pop(&stack));
printf("Minimum element in the stack: %d\n", getMin(stack));
        return 0;
}
```

Output



Result Thus the program to implement the stack was successfully executed and the output was verified.

Ex No:1.3(b)	**PROGRAM TO CONVERT INFIX TO POSTFIX EXPRESSION**
Date:	

Aim

To write a C program to convert the given expression from infix to postfix expression.

Pseudo Code

BEGIN

Define a structure for Stack node Define a structure for Stack
Implement Push () operation
Implement Pop () operation
Implement is isEmpty() operation
Implement precedence operation
Implement intix to postfix ()Operation

END

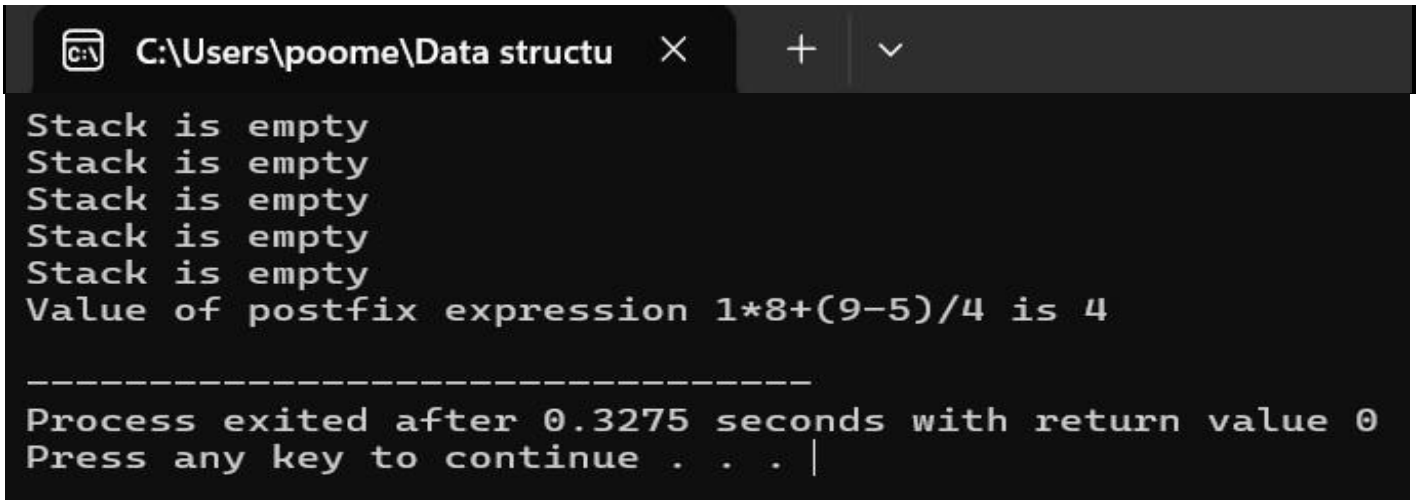
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h> struct
StackNode
{
```

```
        int data;
        struct StackNode* next;
};
struct StackNode* newNode(int data)
{
        struct StackNode* stackNode = (struct StackNode*)malloc(sizeof(struct
StackNode));   stackNode->data = data;        stackNode->next = NULL;
        return stackNode;
}
struct Stack
{
        struct StackNode* top;
}; struct Stack*
createStack()
{
        struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
        stack->top = NULL;
        return stack;
}
int isEmpty(struct Stack* stack)
{
        return stack->top == NULL;
}
void push(struct Stack** stack, int data)
{
        struct StackNode* stackNode = newNode(data);       stackNode-
>next = (*stack)->top;
        (*stack)->top = stackNode;
} int pop(struct Stack**
stack)
{
        if (isEmpty(*stack))
        {
                printf("Stack is empty\n");
                return -1;
        }
        struct StackNode* temp = (*stack)->top;
int popped = temp->data;
        (*stack)->top = (*stack)->top->next;
        free(temp);
        return popped;
}
int evaluatePostfix(char* exp)
{
        struct Stack* stack = createStack();
        int i, val;         for (i
= 0; exp[i]; ++i)
```

```
        {
                if (isdigit(exp[i]))
                {
                        push(&stack, exp[i] - '0');
                }
                else
                {
                        int operand2 = pop(&stack);
        int operand1 = pop(&stack);
                        switch (exp[i])
                        {
                                case '+':
                                        push(&stack, operand1 + operand2);
                                        break;
                case '-':
                                        push(&stack, operand1 - operand2);
                                        break;
                case '*':
                                        push(&stack, operand1 * operand2);
                                        break;
                case '/':
                                        push(&stack, operand1 / operand2);
                        break;
                        }
                }
        }
        return pop(&stack);
} int
main()
{
        char exp[] = "1*8+(9-5)/4";
        printf("Value of postfix expression %s is %d\n", exp, evaluatePostfix(exp));
return 0;
}
```

Output



Result

Thus the program to convert the given expression from infix to postfix expression was successfully executed and the output was verified.

Ex No:1.3(c)	PROGRAM TO FIND LENGTH OF THE LONGEST VALID

Date:

PARENTHESIS SUBSTRING

Aim

To write a C program to find length of the longest valid parenthesis substring.

Pseudo Code

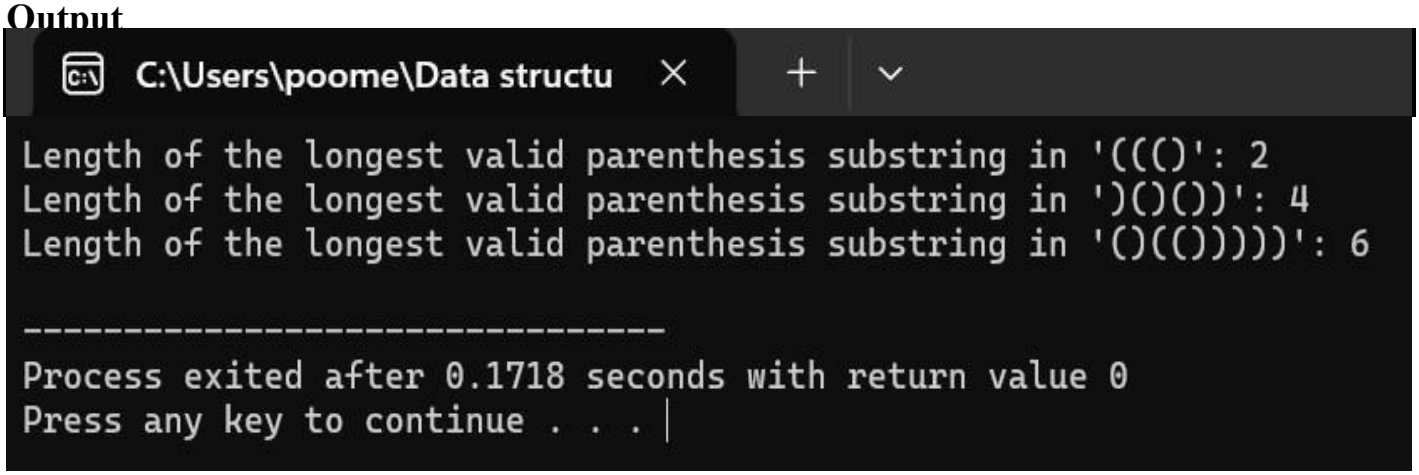
```
BEGIN
    Get the String
    Define a stack
    If Ch== (
        Push the index
    Endif
    If ch==)
        If stack is empty
            update last unpaired to current index
        Else
        If it is non empty
     pop the top of stack
            update maximum length
        Endif
        If Stack is empty
            Set last unpaired to the index of last Popped from the stack
        Endif
    Endif
    Display the length of longest valid Parenthesis
END
```

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> int
max(int a, int b)
{
    return (a > b) ? a : b;
}
int longestValidParentheses(char* s)
{
    int i;
    int maxLen = 0;
    int length = strlen(s);
    int* dp = (int*)malloc(length *
sizeof(int));    memset(dp, 0, length * sizeof(int));
for (i = 1; i < length; i++)
    {
        if (s[i] == ')')
        {
            if (s[i - 1] == '(')
            {
```

```
                        dp[i] = (i >= 2 ? dp[i - 2] : 0) + 2;
                    }
                    else if (i - dp[i - 1] > 0 && s[i - dp[i - 1] - 1] == '(')
                    {
                        dp[i] = dp[i - 1] + ((i - dp[i - 1]) >= 2 ? dp[i - dp[i - 1] - 2] : 0) + 2;
                    }
                    maxLen = max(maxLen, dp[i]);
                }
            }
            free(dp);
            return maxLen;
}
int main()
{
            char str1[] = "(((()";
char str2[] = ")()())";   char
str3[] = "()((()))))";
            printf("Length of the longest valid parenthesis substring in '%s': %d\n", str1,
longestValidParentheses(str1));
            printf("Length of the longest valid parenthesis substring in '%s': %d\n", str2,
longestValidParentheses(str2));
            printf("Length of the longest valid parenthesis substring in '%s': %d\n", str3,
longestValidParentheses(str3));          return 0;
}
```

Output



Result

Thus the program to find length of the longest valid parenthesis substring was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

s

Ex No:2.1(a)	PROGRAM TO GIVE AN ALGORITHM FOR REVERSING A QUEUE Q.
Date:	

Aim

To write C program Give an algorithm for reversing a queue Q. **Pseudo**

Code

BEGIN:

Start the progeam define a structure
for queue Declare the initialize
queue() method void initialize
Quene (Quene q) { q-> front=q-
>rear=-1
check whether the quere is empty or not
Dedare enqueue() operation. Implement
dequeue() operation void print Queme
(Quene *q) { for (i=q->front; i<=rear
>real; i++) printf ("%d", q->arr[i]);
}

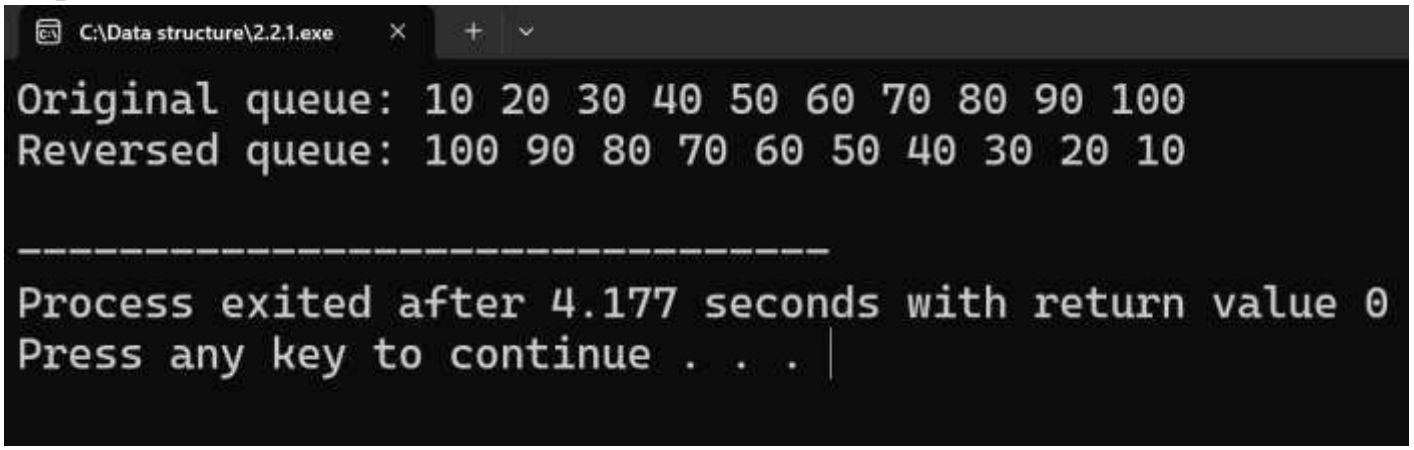
END

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100 typedef
struct {
   int arr[MAX_SIZE];
   int front, rear;
} Queue;
void initializeQueue(Queue *q) {     q-
>front = q->rear = -1;
}
int isEmpty(Queue *q) {
return (q->front == -1);
} void enqueue(Queue *q, int x) {
if (q->rear == MAX_SIZE - 1) {
printf("Queue is full\n");
       return;
}

   if (q->front == -1)         q-
>front = 0;     q->rear++;
q->arr[q->rear] = x;
}
int dequeue(Queue *q) {     if
(isEmpty(q)) {
printf("Queue is empty\n");
       return -1;
   }
```

```
   int x = q->arr[q->front];
if (q->front == q->rear)
q->front = q->rear = -1;
   else        q-
>front++;    return
x;
}
void reverseQueue(Queue *q) {
   if (isEmpty(q))
      return;
   int stack[MAX_SIZE];    int
top = -1;    while (!isEmpty(q))
stack[++top] = dequeue(q);
while (top >= 0)
enqueue(q, stack[top--]);
}
void printQueue(Queue *q) {
int i;
   for (i = q->front; i <= q->rear; i++)
printf("%d ", q->arr[i]);    printf("\n");
} int main()
{    Queue
q;
   initializeQueue(&q);
enqueue(&q, 10);    enqueue(&q,
20);    enqueue(&q, 30);
enqueue(&q, 40);    enqueue(&q,
50);    enqueue(&q, 60);
enqueue(&q, 70);    enqueue(&q,
80);    enqueue(&q, 90);
enqueue(&q, 100);
printf("Original queue: ");
   printQueue(&q);
reverseQueue(&q);
printf("Reversed queue: ");
printQueue(&q);    return 0;
}
```

Output



Result

Thus the program to give an algorithm to reversing a queue Q was successfully executed and the output was verified.

Ex No:2.1(b)	PROGRAM WE ARE GIVEN A STACK DATA STRUCTURE WITH PUSH AND POP OPERATION
Date:	

Aim Pseudo Code

To write C program We are given a stack data structure with push and pop operations

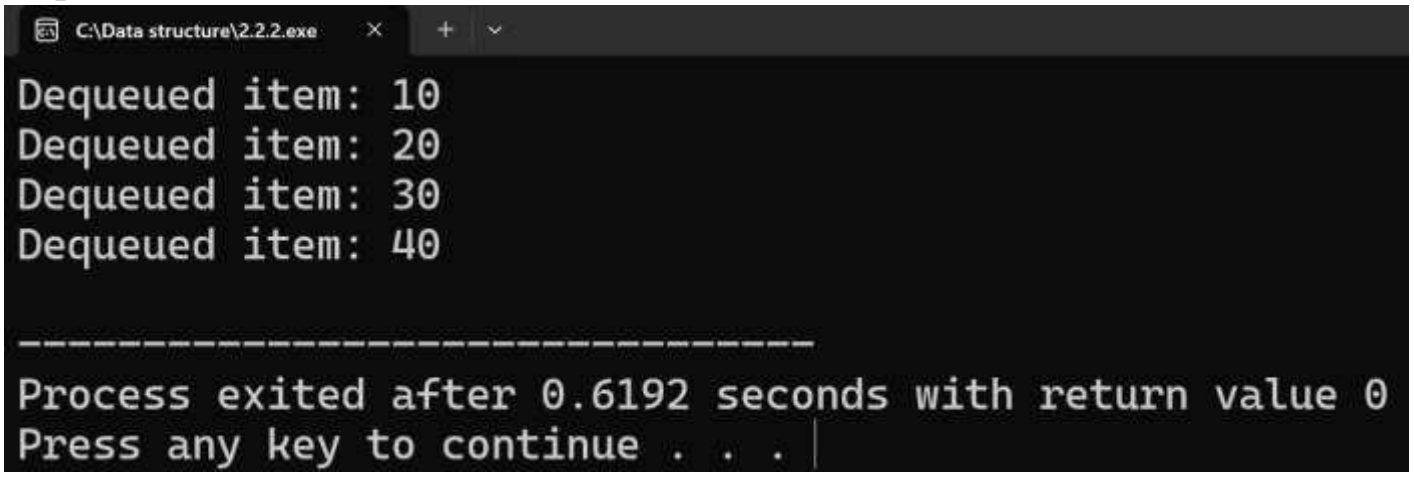
```
BEGIN
        Define a structice for stack  void
        initalize stack (stack*s) { s->top
        = -1;
        } check whether the queue is
        empty
        Implement push() operation
        Implement pop() operation
        Void initialize Queue (queue *q){
                Initialize stack(&(q->s1));
                Initialize stack(&(q->s2));
        }
                Implement enqueue() operation
                Implement dequeue () operation Display
        the result.
END
```

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100 typedef
struct {
   int arr[MAX_SIZE];
   int top; }
Stack;
void initializeStack(Stack *s) {     s-
>top = -1;
```

```
} int isEmpty(Stack *s)
{    return (s->top == -
1);
}
void push(Stack *s, int x) {    if
(s->top == MAX_SIZE - 1) {
printf("Stack overflow\n");
      return;
   }
   s->arr[++(s->top)] = x;
} int pop(Stack *s) {    if
(isEmpty(s)) {        printf("Stack
underflow\n");
      return -1;
   }
   return s->arr[(s->top)--];
}
typedef struct {
Stack s1;
   Stack s2; }
Queue;
void initializeQueue(Queue *q) {
initializeStack(&(q->s1));    initializeStack(&(q->s2));
} void enqueue(Queue *q, int x) {
while (!isEmpty(&(q->s1)))
push(&(q->s2), pop(&(q->s1)));
push(&(q->s1), x);    while
(!isEmpty(&(q->s2)))
push(&(q->s1), pop(&(q->s2)));
}
int dequeue(Queue *q) {    if
(isEmpty(&(q->s1))) {
printf("Queue is empty\n");
      return -1;
   }
   return pop(&(q->s1));
} int main()
{    Queue
q;
   initializeQueue(&q);    enqueue(&q, 10);
enqueue(&q, 20);    enqueue(&q, 30);
printf("Dequeued item: %d\n", dequeue(&q));
printf("Dequeued item: %d\n", dequeue(&q));
enqueue(&q, 40);    enqueue(&q, 50);
printf("Dequeued item: %d\n", dequeue(&q));
printf("Dequeued item: %d\n", dequeue(&q));
return 0; }
```

Output



Result

Thus the program We are given a stack data structure with push and pop operations was successfully executed and the output was verified.

Ex No:2.1(C)	PROGRAM IMPLEMENT A FIRST IN FIRST OUT (FIFO) QUEUE USING ONLY TWO STACKS
Date:	

Aim

To write C program Implement a first in first out (FIFO) queue using only two stacks. **Pseudo Code**

BEGIN

Define a structure my queue with two integer arrays

My queue create(){

Create a new myqueue instance , allocate memory for input stack and output stack and initialize input top and output top.

}

Push(myqueue *obj ,int x){

Push an element onto the input stack

}

Pop(myqueue *obj){

Pop an element from the my queue

}

Peek(myqueue *obj){

If (obj ->output top==-1){

Obj->output stack [obj -> output top];

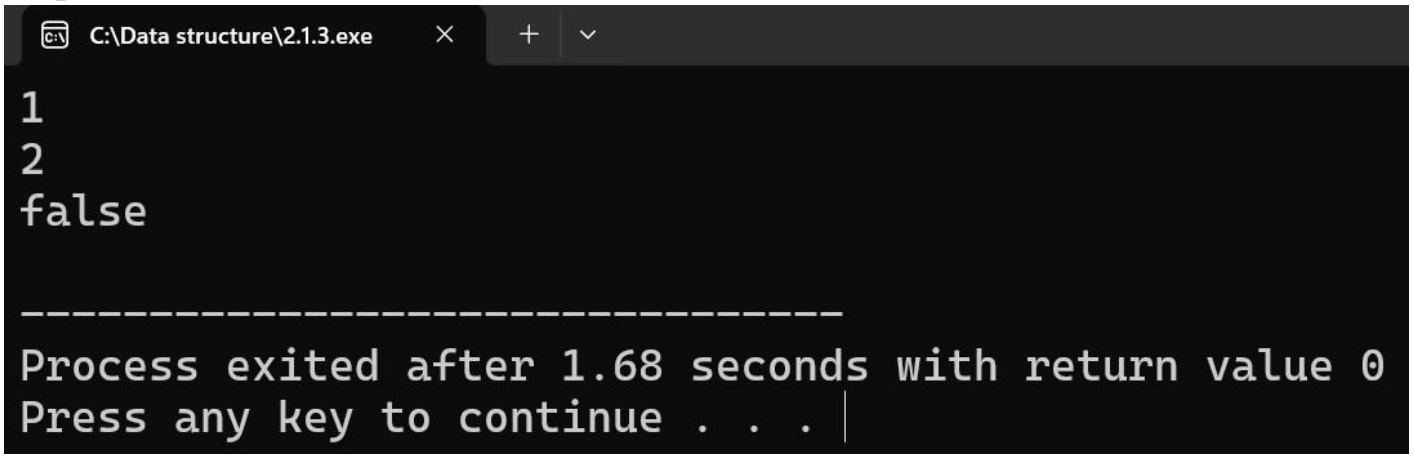
```
                    }
                }
        Empty(myqueue*obj){
                Check if the my queue is empty
                }
        Myqueue free(myqueue*obj){
                Free(obj);
        }
END
```

Source Code

```
#include <stdbool.h>
#include <stdio.h> #include
<stdlib.h>

typedef struct {
int *inputStack;
int inputTop;    int
*outputStack;
  int outputTop; }
MyQueue;

MyQueue* myQueueCreate() {
  MyQueue* queue = (MyQueue*)malloc(sizeof(MyQueue));    queue-
>inputStack = (int*)malloc(sizeof(int) * 1000);    queue->inputTop = -1;
  queue->outputStack = (int*)malloc(sizeof(int) * 1000);
queue->outputTop = -1;    return queue;
}

void push(MyQueue* obj, int x) {
  obj->inputStack[++(obj->inputTop)] = x;
}

int pop(MyQueue* obj) {
  if (obj->outputTop == -1) {        while (obj->inputTop > -1) {            obj-
>outputStack[++(obj->outputTop)] = obj->inputStack[obj->inputTop--];
    }
  }
  return obj->outputStack[obj->outputTop--];
}

int peek(MyQueue* obj) {    if (obj->outputTop == -1) {        while (obj-
>inputTop > -1) {            obj->outputStack[++(obj->outputTop)] = obj-
>inputStack[obj->inputTop--];
    }
  }
  return obj->outputStack[obj->outputTop];
}
```

```
bool empty(MyQueue* obj) {     return (obj->inputTop == -
1) && (obj->outputTop == -1);
}

void myQueueFree(MyQueue* obj) {
   free(obj->inputStack);
free(obj->outputStack);     free(obj);
} int main()
{
   MyQueue* obj = myQueueCreate();     push(obj, 1);
push(obj, 2);     printf("%d\n", pop(obj)); // Output: 1
printf("%d\n", peek(obj)); // Output: 2     printf("%s\n",
empty(obj) ? "true" : "false"); // Output: false
myQueueFree(obj);     return 0;
}
```

Output



Result

Thus the program Implement a first in first out (FIFO) queue using only two stacks was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:2.2(a)	PROGRAM TO HEAD OF A SORTED LINKED LIST, DELETE ALL DUPLICATES
Date:	

Aim

To write C program Given the head of a sorted linked list, delete all duplicates such that each element appears only once

Pseudo Code

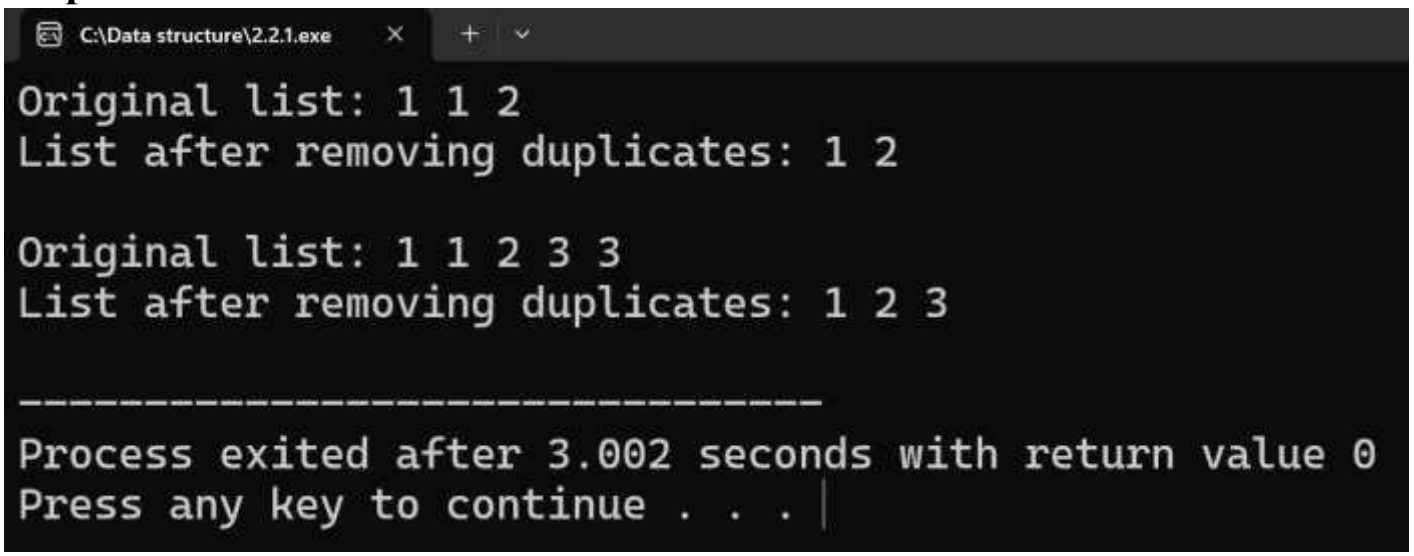
```
BEGIN
        Defime the structure for linked list node  check whether
        the head is Now  struct listNode New node (int val) {
        struct listNode "Node = (struct list Node *) malloc  (sized
        (struct list Node));
                Node-> val = val;
                Node->nout = NULL;
                return node;
                }
        Implement pritlist () method
        Display the result
ENG
```

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
   int val;
   struct ListNode *next;
};
struct ListNode* deleteDuplicates(struct ListNode* head) {
if (head == NULL)         return NULL;     struct ListNode
*current = head;     while (current->next != NULL) {
if (current->val == current->next->val) {             struct
ListNode *temp = current->next;             current->next =
current->next->next;             free(temp);         } else {
          current = current->next;
       }
   }
   return head;
}
struct ListNode* newNode(int val) {
   struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
node->val = val;     node->next = NULL;     return node;
} void printList(struct ListNode* head)
{     struct ListNode* current = head;
```

```
while (current != NULL) {
printf("%d ", current->val);
        current = current->next;
    }
    printf("\n");
} int main()
{     struct
ListNode*
head1 =
newNode(1
);     head1-
>next =
newNode(1
);
    head1->next->next = newNode(2);

    printf("Original list: ");
printList(head1);
    head1 = deleteDuplicates(head1);
printf("List after removing duplicates: ");
printList(head1);     struct ListNode* head2
= newNode(1);     head2->next =
newNode(1);     head2->next->next =
newNode(2);     head2->next->next->next =
newNode(3);
    head2->next->next->next->next = newNode(3);
printf("\nOriginal list: ");     printList(head2);
    head2 = deleteDuplicates(head2);
printf("List after removing duplicates: ");
printList(head2);     return 0;
}
```

Output



Result

Thus the program Given the head of a sorted linked list, delete all duplicates such that each element appears only once was successfully executed and the output was verified.

Ex No:2.2(b)	PROGRAM TO HEAD OF A SINGLY LINKED LIST AND TWO INTEGERS LEFT AND RIGHT
Date:	

Aim

To write C program Given the head of a singly linked list and two integers left and right.

Pseudo Code BEGIN

Define a structure list node with an integer variable reverse between (struct ListNode*head,int left,int right){

Return head;

}

New Node(int val){

node->val=val; node->next=NULL;

return node;

}

Print List(struct ListNode*head){

Print the values of the ListNodes in the linked list; }

END

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
   int val;
   struct ListNode *next;
}; struct ListNode* reverseBetween(struct ListNode* head, int left, int
right) {    if (head == NULL || left == right)        return head;
   struct ListNode* dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
dummy->next = head;    struct ListNode* prev = dummy;    for (int i = 1; i <
left; i++)        prev = prev->next;    struct ListNode* current = prev->next;
```

```
   for (int i = left; i < right; i++) {        struct
ListNode* nextNode = current->next;
current->next = nextNode->next;
nextNode->next = prev->next;
       prev->next = nextNode;
   }
   return dummy->next;
}
struct ListNode* newNode(int val) {
   struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
node->val = val;     node->next = NULL;     return node;
}
void printList(struct ListNode* head) {
struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
       current = current->next;
   }
   printf("\n");
} int main()
{
   struct ListNode* head1 = newNode(1);
head1->next = newNode(2);     head1->next->next
= newNode(3);     head1->next->next->next =
newNode(4);
   head1->next->next->next->next = newNode(5);
printf("Original list: ");     printList(head1);
   head1 = reverseBetween(head1, 2, 4);     printf("List after
reversing from position 2 to position 4: ");
printList(head1);     struct ListNode* head2 = newNode(5);
   printf("\nOriginal list: ");
printList(head2);
   head2 = reverseBetween(head2, 1, 1);     printf("List after
reversing from position 1 to position 1: ");
printList(head2);     return 0; }
```

Output

```
Original list: 1 2 3 4 5
List after reversing from position 2 to position 4: 1 4 3 2 5

Original list: 5
List after reversing from position 1 to position 1: 5

--------------------------------
Process exited after 1.302 seconds with return value 0
Press any key to continue . . .
```

Result

Thus the program Given the head of a singly linked list and two integers left and right was successfully executed and the output was verified.

Ex No:2.2(c)	PROGRAM ALTERNATING SPLIT OF A GIVEN SINGLY LINKED LIST
Date:	

Aim Pseudo Code

To write C program Alternating split of a given Singly Linked List.

```
BEGIN
        Define the structure for Linked List  Void
        Alternating split() method  implement print list
        () method Declare freelist () method void
        freelist (struct listtNode * head){ struct
        listNode*current = head;  while (curent !=
        NULL) { struct listNode *temp = Current
        Current Curent -> rent;  free (temp);
                }
        }
        Display the result;
END
```

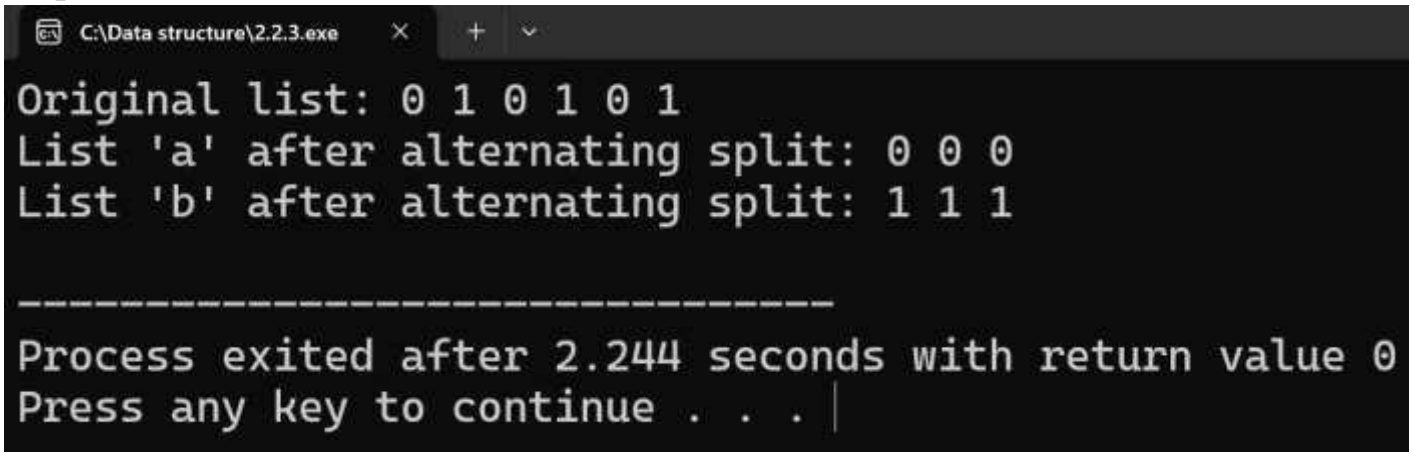
Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
   int val;
   struct ListNode *next;
};
void AlternatingSplit(struct ListNode* head, struct ListNode** a, struct ListNode** b) {
if (head == NULL)
```

```
        return;
    struct ListNode* current = head;
struct ListNode* aTail = NULL;     struct
ListNode* bTail = NULL;     struct
ListNode* aHead = NULL;
    struct ListNode* bHead = NULL;
int count = 0;     while (current !=
NULL) {
        struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
newNode->val = current->val;         newNode->next = NULL;         if (count % 2 ==
0) {             if (aTail == NULL) {                 aHead = newNode;                 aTail =
newNode;
            } else {
                aTail->next = newNode;
                aTail = newNode;
            }
        } else {
            if (bTail == NULL) {
bHead = newNode;                 bTail
= newNode;
            } else {
                bTail->next = newNode;
                bTail = newNode;
            }
        }
        current = current->next;
count++;
    }
    *a = aHead;
*b = bHead;
}
struct ListNode* newNode(int val) {
    struct ListNode* node = (struct ListNode*)malloc(sizeof(struct ListNode));
node->val = val;     node->next = NULL;     return node;
} void printList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
        current = current->next;
    }
    printf("\n");
} void freeList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {         struct
ListNode* temp = current;
current = current->next;
        free(temp);
```

```
    }
} int main()
{
    struct ListNode* head = newNode(0);     head-
>next = newNode(1);     head->next->next =
newNode(0);     head->next->next->next =
newNode(1);     head->next->next->next->next =
newNode(0);
    head->next->next->next->next->next = newNode(1);
printf("Original list: ");     printList(head);     struct
ListNode* a;     struct ListNode* b;
    AlternatingSplit(head, &a, &b);
printf("List 'a' after alternating split: ");
printList(a);
    printf("List 'b' after alternating split: ");
printList(b);     freeList(a);     freeList(b);
return 0;
}
```

Output



Result

Thus the program Alternating split of a given Singly Linked List was successfully executed and the output was verified

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:2.3(a)	**PROGRAM INSERT VALUE IN SORTEDWAY IN DOUBLY LINKED LIST**
Date:	

Aim

To write C program Insert value in sorted way in a sorted doubly linked list**.**

Pseudo Code

```
BEGIN
        Define a structure for doubly linked list
        Define a Structuree for list Node
        Declare the insert Sorted () method  voId
        yuint list (struct listmode* head) { Struct
        listNode * current = head; while
        (current!=NULL){
                        Printf('%d', current ->val); Current = current-> rent  }
                }
        Display the result
END
```

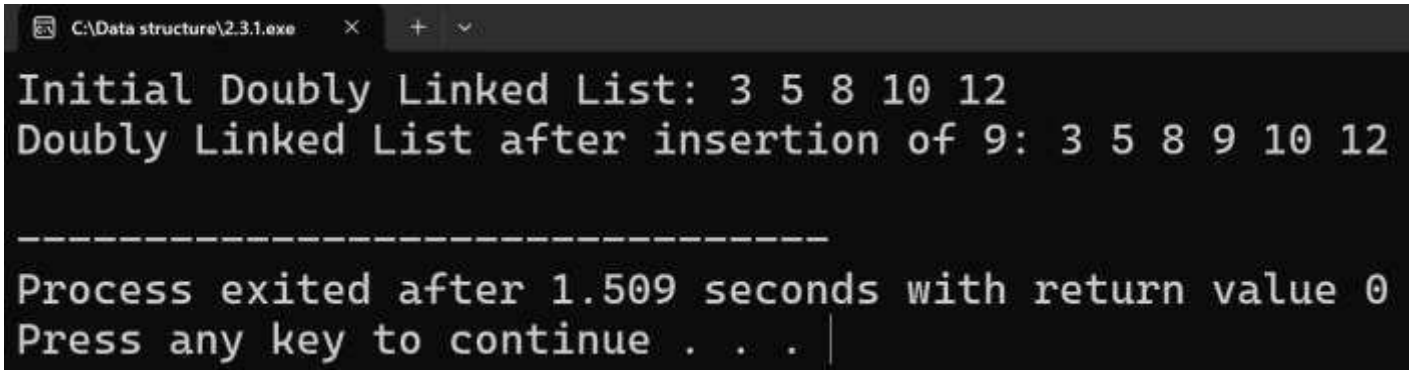
Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
```

```
   int val;
   struct ListNode *prev;
struct ListNode *next;
};
struct ListNode* createNode(int val) {
   struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->val = val;
newNode->prev = NULL;
newNode->next = NULL;     return
newNode;
}
void insertSorted(struct ListNode** head, int val) {
struct ListNode* newNode = createNode(val);     if
(*head == NULL || val <= (*head)->val) {
newNode->next = *head;        if (*head != NULL)
          (*head)->prev = newNode;
*head = newNode;
       return;
   }
   struct ListNode* current = *head;     while (current->next !=
NULL && current->next->val < val)        current = current-
>next;     newNode->next = current->next;     if (current->next
!= NULL)        current->next->prev = newNode;     current-
>next = newNode;     newNode->prev = current;
} void printList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
       current = current->next;
   }
   printf("\n");
} void freeList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {        struct
ListNode* temp = current;
current = current->next;
       free(temp);
   }
} int main()
{
   struct ListNode* head = NULL;
   insertSorted(&head, 3);     insertSorted(&head,
5);     insertSorted(&head, 8);
insertSorted(&head, 10);     insertSorted(&head,
12);     printf("Initial Doubly Linked List: ");
printList(head);     insertSorted(&head, 9);
```

```
printf("Doubly Linked List after insertion of 9: ");
printList(head);     freeList(head);     return 0; }
```

Output



Result

Thus the program Insert value in sorted way in a sorted doubly linked list was successfully executed and the output was verified.

Ex No:2.3(b)	PROGRAM TO DELETE ALL OCCURANCES OF A GIVEN KEY IN DOUBLY LINKED LIST
Date:	

Aim

To write C Delete all occurrences of a given key in a doubly linked list. Given a doubly linked list and a key x.

Pseudo Code

```
BEGIN
        Define a structure for linked hist
        Define a structure for hit node
        Implement printlist() method
        Void Freelist (struct ListNode * head) {
                Struct listNode * current = head;
                while (curent!=NULL){
                        Struct list Node * temp = current;
                        Current = Current -> next;  free
                        (temp);
                }
        }
END
```

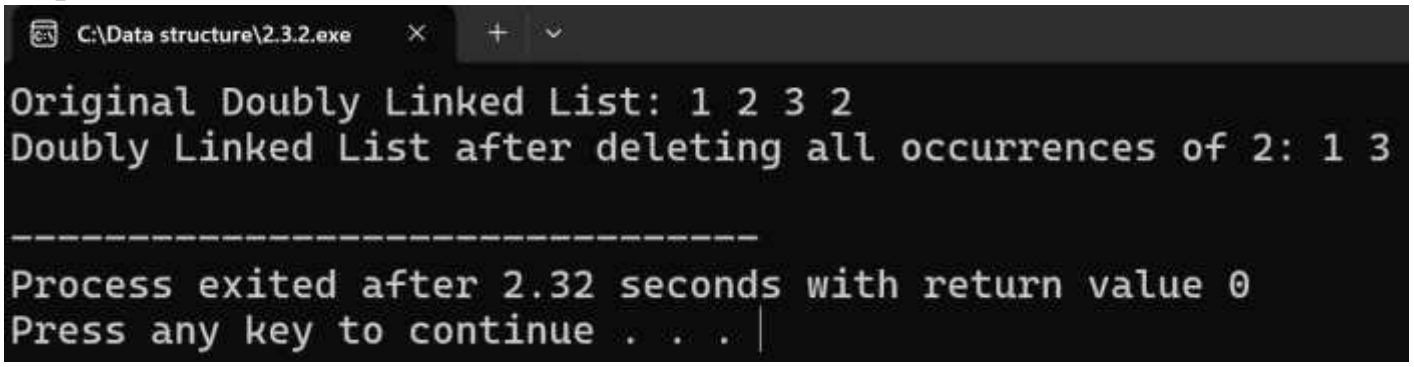
Source Code

```
#include <stdio.h> #include
<stdlib.h>
```

```
struct ListNode {
   int val;
   struct ListNode *prev;
struct ListNode *next;
};
struct ListNode* createNode(int val) {
   struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->val = val;
newNode->prev = NULL;
newNode->next = NULL;     return
newNode;
}
struct ListNode* deleteKey(struct ListNode* head, int key) {
if (head == NULL)        return NULL;     while (head !=
NULL && head->val == key) {
       struct ListNode* temp = head;
head = head->next;        if (head
!= NULL)            head->prev =
NULL;
       free(temp);
   }
   struct ListNode* current = head;     while
(current != NULL) {        if (current->val ==
key) {            struct ListNode* temp = current;
current->prev->next = current->next;            if
(current->next != NULL)                current-
>next->prev = current->prev;            current =
current->next;
          free(temp);
} else {
          current = current->next;
       }
   }
   return head;
} void printList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
       current = current->next;
   }
   printf("\n");
} void freeList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {        struct
ListNode* temp = current;
current = current->next;
```

```
        free(temp);
    }
} int main()
{
    struct ListNode* head = createNode(1);     head-
>next = createNode(2);     head->next->prev = head;
head->next->next = createNode(3);     head->next-
>next->prev = head->next;     head->next->next->next
= createNode(2);     head->next->next->next->prev =
head->next->next;
    printf("Original Doubly Linked List: ");
printList(head);     int key = 2;
    head = deleteKey(head, key);     printf("Doubly Linked List after
deleting all occurrences of %d: ", key);     printList(head);
freeList(head);     return 0; }
```

Output



Result

Thus the program Delete all occurrences of a given key in a doubly linked list was successfully executed and the output was verified.

Ex No:2.3(c)	PROGRAM TO SEARCH AN ELEMENT IN A CIRCULAR SINGLY LINKED LISTS
Date:	

Aim

To write C program Search an element in a CIRCULAR SINGLY Linked List**.**

Pseudo Code

```
BEGIN
        Define a structure for linked
        Define list a structure for list Node  Void
        insert End () method
        bool search Iterative (struct Node * head,int x)
        { if (head == NULL)
                return false;
                struct Node current = head;
                do { if (current ->data ==n)
                return true;
                        current (current! = head);
```

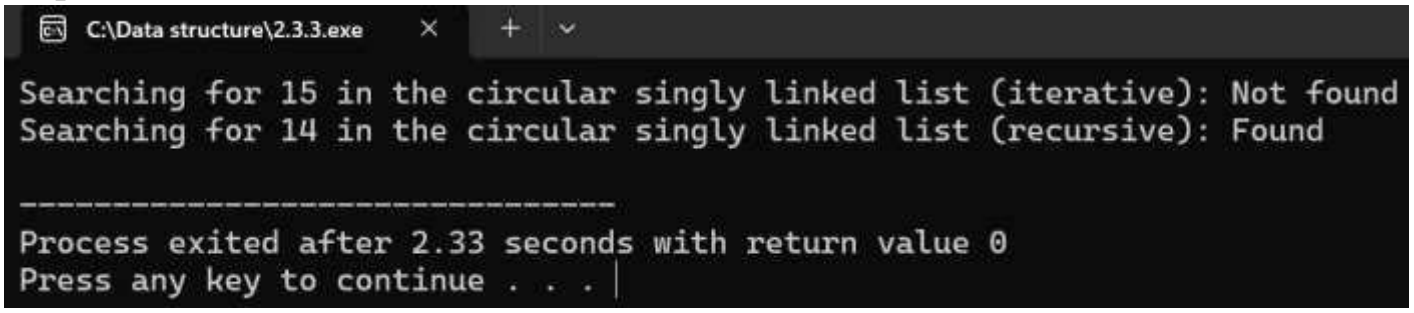
```
                                return false;
                }
        Display the result;
END
```

Source Code

```
#include <stdio.h>
#include <stdlib.h> #include
<stdbool.h>S
struct Node {
   int data;    struct
Node *next;
};
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {        printf("Memory allocation failed\n");
exit(1);
   }
   newNode->data  =  data;
newNode->next  =  NULL;
return newNode;
} void insertEnd(struct Node** head, int data)
{    struct Node* newNode =
createNode(data);    if (*head == NULL) {
*head = newNode;
      newNode->next = *head;
   } else {
      struct Node* current = *head;
while (current->next != *head)
current = current->next;        current-
>next = newNode;
      newNode->next = *head;
   }
}
bool searchIterative(struct Node* head, int x) {
if (head == NULL)
      return false;    struct
Node* current = head;    do {
      if (current->data == x)
return true;        current =
current->next;    } while
(current != head);    return
false;
}
bool searchRecursive(struct Node* head, int x) {
if (head == NULL)
      return false;    if
(head->data == x)
```

```
return true;     if (head-
>next == head)
       return false;     return
searchRecursive(head->next, x);
} int main()
{
   struct Node* head = NULL;
insertEnd(&head, 14);
insertEnd(&head, 21);
insertEnd(&head, 11);
insertEnd(&head, 30);
insertEnd(&head, 10);     int
key = 15;
   printf("Searching for %d in the circular singly linked list (iterative): %s\n", key, searchIterative(head,
key) ? "Found" : "Not found");     key = 14;
   printf("Searching for %d in the circular singly linked list (recursive): %s\n", key, searchRecursive(head,
key) ? "Found" : "Not found");     struct Node* current = head;     if (head != NULL) {
       do {
          struct Node* temp = current;
current = current->next;             free(temp);
       } while (current != head);
   }
return 0;
}
```

Output



Result

Thus the program Search an element in a CIRCULAR SINGLY Linked List was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:2.4(a)	**PROGRAM TO REVERSE THE LIST, AND RETURN THE**

Date:	REVERSED LIST

Aim Pseudo Code

To write C program Given the head of a singly linked list, reverse the list, and return the reversed list.

BEGIN

Define a Structure listsnode with an integer variable
Create Node (int val): Create a new List Node with The given Value insert End (struct list Node** head, int Val): Invest a new ListNode with given value of end of the list. Print list (struct ListNode * head): Print the Values of the List Nodes in the list. reverse List (istruct List Node* head): Reverse the order of ListNodes in the list, Print the reversed List

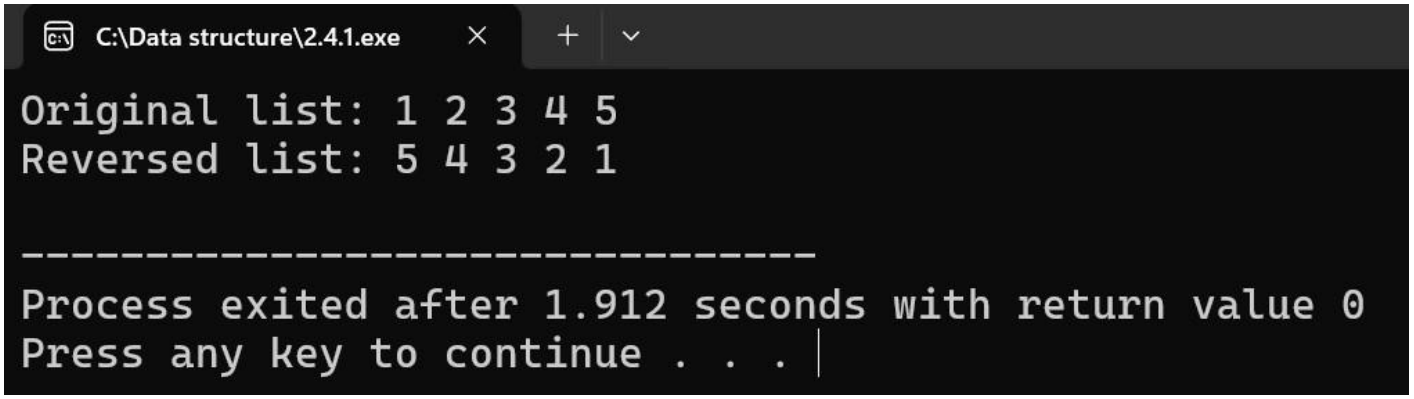
END

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
   int val;
   struct ListNode *next;
};
struct ListNode* createNode(int val) {
   struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->val = val;
newNode->next = NULL;     return
newNode;
} void insertEnd(struct ListNode** head, int val)
{     struct ListNode* newNode =
createNode(val);     if (*head == NULL) {
       *head = newNode;
   } else {
       struct ListNode* current = *head;
while (current->next != NULL)
current = current->next;         current->next
= newNode;
   }
} void printList(struct ListNode* head)
{     struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
       current = current->next;
   }
   printf("\n");
}
struct ListNode* reverseList(struct ListNode* head) {
struct ListNode* prev = NULL;     struct ListNode*
current = head;     struct ListNode* next = NULL;
while (current != NULL) {         next = current->next;
```

```
        current->next = prev;
prev = current;         current
= next;
    }
    return prev;
} int main()
{
    struct ListNode* head = NULL;
    insertEnd(&head, 1);
insertEnd(&head, 2);
insertEnd(&head, 3);
insertEnd(&head, 4);
insertEnd(&head, 5);
printf("Original list: ");
printList(head);     head =
reverseList(head);
printf("Reversed list: ");
printList(head);     struct
ListNode* current = head;     while
(head != NULL) {         struct
ListNode* temp = head;         head
= head->next;
        free(temp);
    }
    return 0; }
```

Output



Result

Thus the program Given the head of a singly linked list, reverse the list, and return the reversed list was successfully executed and the output was verified.

Ex No:2.4(b)	PROGRAM TO MERGE TWO SORTED LINKED LIST IN PLACE
Date:	

Aim Pseudo Code

To write C program to merge two sorted linked lists in-place.

```
BEGIN
        define a structure listNode with an integer Variable  Create
        Node (int val){
        create a new list Node with the given Value & NULL next pointer
        }
        insertEnd (struct ListNode ** head, int val) if
                (*head = =NULL) {
                        *head = new Node;
        }
        Else { current = current -> next;
                 }
        Print List (struct List Node * head) {
                Print the Valves of the listnode in the list.
        }
        Merge lists (struct List Node * hand 1, struct List Node * head2){
                Merge two sorted linted lists into one sorted linked list
END.
```

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct ListNode {
   int val;
   struct ListNode *next;
};
struct ListNode* createNode(int val) {
   struct ListNode* newNode = (struct ListNode*)malloc(sizeof(struct ListNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->val = val;
newNode->next = NULL;     return
newNode;
} void insertEnd(struct ListNode** head, int val)
{     struct ListNode* newNode =
createNode(val);     if (*head == NULL) {
       *head = newNode;
   } else {
       struct ListNode* current = *head;
while (current->next != NULL)
current = current->next;         current->next
= newNode;
   }
```

```
} void printList(struct ListNode* head)
{    struct ListNode* current = head;
while (current != NULL) {
printf("%d ", current->val);
current = current->next;
   }
   printf("\n");
}
struct ListNode* mergeLists(struct ListNode* head1, struct ListNode* head2) {
if (head1 == NULL)        return head2;    if (head2 == NULL)
      return head1;
   struct ListNode* dummy = createNode(0);
struct ListNode* current = dummy;    while
(head1 != NULL && head2 != NULL) {
      if (head1->val < head2->val) {
current->next = head1;            head1
= head1->next;
      } else {
         current->next = head2;
         head2 = head2->next;
      }
      current = current->next;
   }
   if (head1 != NULL)
current->next = head1;    if
(head2 != NULL)
      current->next = head2;    struct ListNode*
mergedHead = dummy->next;
   free(dummy);
return mergedHead;
} int main()
{
   struct ListNode* head1 = NULL;    struct ListNode*
head2 = NULL;    int N = 4, M = 3;    int N_values[] = {5,
10, 15, 40};    int M_values[] = {2, 3, 20};    for (int i = 0; i
< N; i++)        insertEnd(&head1, N_values[i]);    for (int i =
0; i < M; i++)        insertEnd(&head2, M_values[i]);
printf("First linked list: ");    printList(head1);
printf("Second linked list: ");    printList(head2);    struct
ListNode* mergedHead = mergeLists(head1, head2);
   printf("Merged linked list: ");
printList(mergedHead);
struct ListNode* current;
while (head1 != NULL) {
current = head1;        head1 =
head1->next;
free(current);
   }
```

```
   while (head2 != NULL) {
current = head2;        head2
= head2->next;
free(current);
   }
return 0; }
```

Output

```
First linked list: 5 10 15 40
Second linked list: 2 3 20
Merged linked list: 2 3 5 10 15 20 40
free(): double free detected in tcache 2
Aborted
```

Result

Thus the program to merge two sorted linked lists in-place was successfully executed and the output was verified.

Ex No:2.4(c)	PROGRAM TO IMPLEMENT THE STACK AND QUEUE DATA STRUCTURE USING LINKED LIST
Date:	

Aim Pseudo Code

To Write C program to implement the Stack and Queue Data Structure using Linked List.

BEGIN

Define a structure Node with an integer variable Variabele 'data' and a pointer to the next node next.
Create Node (int data): create a new node with given data. init Stack (struct stack *stack): Initialize the stock by setting top pointer to NULL.
is Empty stack (struct stack *stack):check if stack is empty
push (struct stack * stack, int data); push the data pop
(struct stack * stack):pop the data peek (struct stack *
stack):
it Queue (struct Queue queue) { queue ->
front =queue-> rear = NULL
}
is empty Queue (struct Queue* queue) {
return queue -> front= = NULL;
}
enqueue (struct Queue* queue, int data) { Enqueue
data into the queue;
}
dequeue(struct Queue* queue) {
dequeue data from the queue
}
Front (struct Queue* queue) { get the data at the front of the
queue without removing it
print Stack (struct stack * stack) { struct
Node * current = stack->top; print
the elements of the stack
} print Queue (struct Queue* queue)
{ struct node*current = queue ->
front; }

END

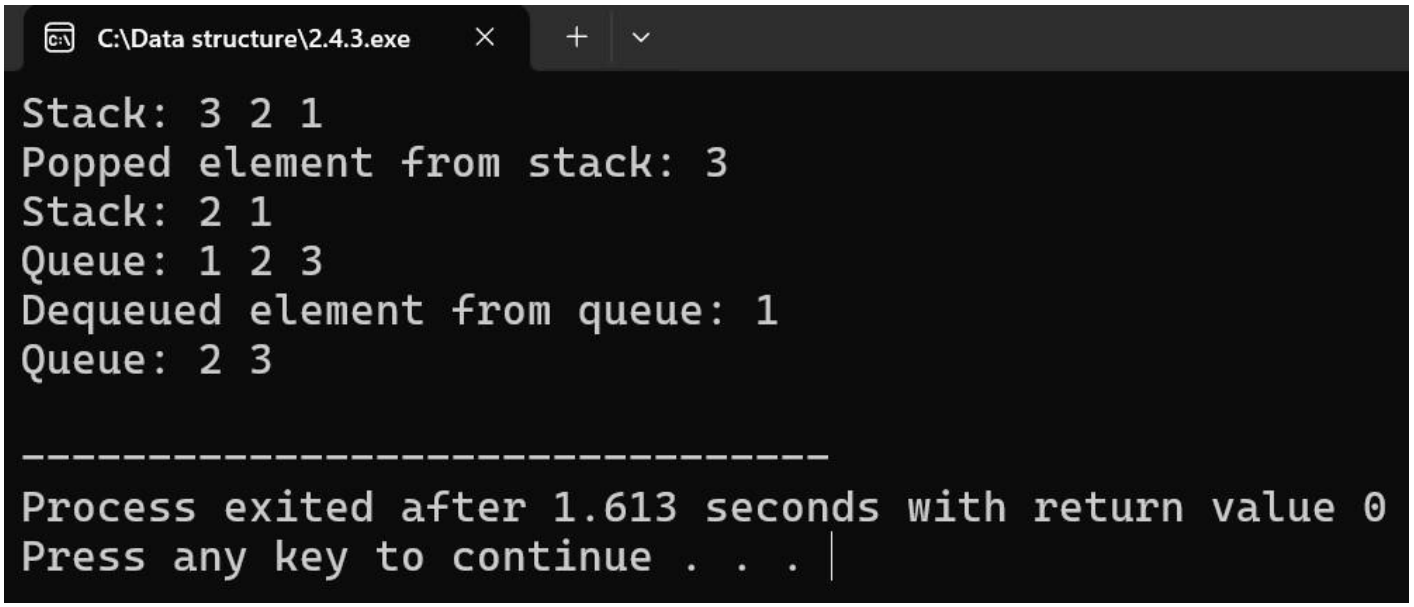
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> struct
Node {
   int data;
   struct Node *next;
```

```
};
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {        printf("Memory allocation failed\n");
exit(1);
   }
   newNode->data  =  data;
newNode->next  =  NULL;
return newNode;
} struct Stack {
struct Node *top;
}; void initStack(struct Stack
*stack) {    stack->top = NULL;
} bool isEmptyStack(struct Stack *stack)
{    return stack->top == NULL;
} void push(struct Stack *stack, int data) {
struct Node* newNode = createNode(data);
newNode->next = stack->top;    stack->top =
newNode;
} int pop(struct Stack *stack) {    if
(isEmptyStack(stack)) {
printf("Error: Stack is empty\n");
      exit(1);
   }
   struct Node* temp = stack->top;
int data = temp->data;    stack->top =
temp->next;    free(temp);    return
data; } int peek(struct Stack *stack) {
if (isEmptyStack(stack)) {
printf("Error: Stack is empty\n");
      exit(1);
   }
   return stack->top->data;
} struct Queue {    struct
Node *front, *rear;
}; void initQueue(struct Queue *queue)
{         queue->front  =  queue->rear  =
NULL;
}
bool isEmptyQueue(struct Queue *queue) {
return queue->front == NULL;
}
void enqueue(struct Queue *queue, int data) {
struct Node* newNode = createNode(data);    if
(isEmptyQueue(queue)) {
      queue->front = queue->rear = newNode;
   } else {        queue->rear->next =
newNode;
```

```
        queue->rear = newNode;
    }
}
int dequeue(struct Queue *queue) {
if (isEmptyQueue(queue)) {
        printf("Error: Queue is empty\n");
exit(1);
    }
    struct Node* temp = queue->front;
int data = temp->data;     queue-
>front = temp->next;     free(temp);
return data;
}
int front(struct Queue *queue) {
if (isEmptyQueue(queue)) {
        printf("Error: Queue is empty\n");
exit(1);
    }
    return queue->front->data;
} void printStack(struct Stack *stack)
{     struct Node* current = stack-
>top;
    printf("Stack: ");     while
(current != NULL) {
printf("%d ", current->data);
        current = current->next;
    }
printf("\n");
}
void printQueue(struct Queue *queue) {
struct Node* current = queue->front;
    printf("Queue: ");     while
(current != NULL) {
printf("%d ", current->data);
        current = current->next;
    }
printf("\n");
} int main() {     struct Stack stack;     struct Queue queue;
initStack(&stack);     initQueue(&queue);     push(&stack,
1);     push(&stack, 2);     push(&stack, 3);
printStack(&stack);     printf("Popped element from stack:
%d\n", pop(&stack));
    printStack(&stack);     enqueue(&queue, 1);     enqueue(&queue,
2);     enqueue(&queue, 3);     printQueue(&queue);
printf("Dequeued element from queue: %d\n", dequeue(&queue));
printQueue(&queue);     return 0;
}
```

Output



Result

Thus the program to implement the Stack and Queue Data Structure using Linked List was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	

INITIAL OF FACULTY	

Ex No:2.5(a)	PROGRAM TO COUNT NUMBER OF NODES THAT LIE IN THE GIVEN RANGE
Date:	

Aim Pseudo Code

To Write C program to count number of nodes that lie in the given range.

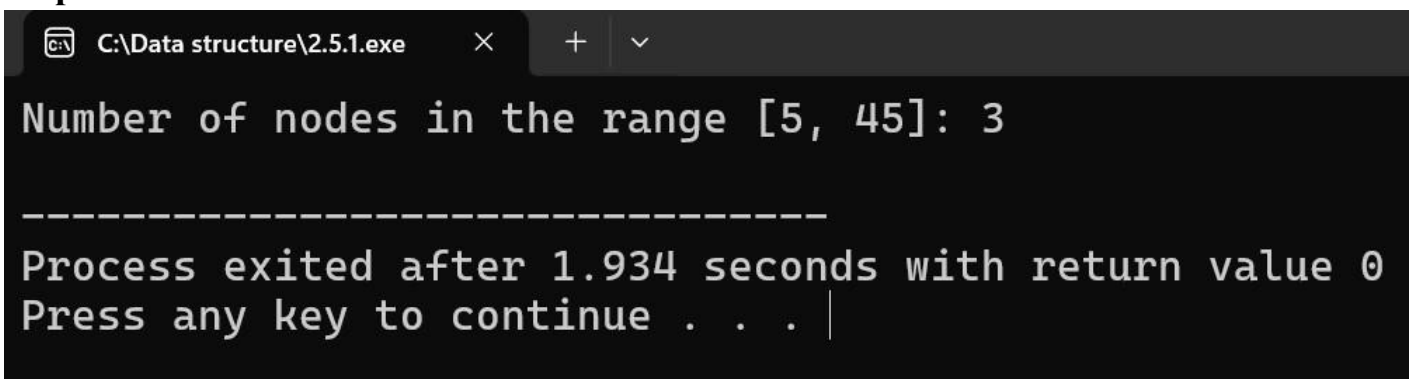
```
BEGIN
        define a Structure Trees Node with an integer variable create Node (int val) {  create
        a new Tree Node with the given Value and Nuse left and right pointers.
        }
        insert (struct Tree Node *root, int Val) {
        if (root ==NULL)  return create
                Node (val); if (val <root
                ->Val)
                root ->left = insert (root ->left, val);
        else if (val > root ->val) return root:
        }
        Count nodes in range (struct Tree node*root,int low,int high)
        {
                Count the number of Nodes In the binarysearch tree that fall with Range [low, high]
        }
END
```

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct TreeNode {
   int val;
   struct TreeNode *left;
struct TreeNode *right;
};
struct TreeNode* createNode(int val) {
   struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
```

```
   newNode->val = val;     newNode->left =
newNode->right = NULL;     return newNode;
}
struct TreeNode* insert(struct TreeNode* root, int val) {
if (root == NULL)         return createNode(val);     if
(val < root->val)         root->left = insert(root->left, val);
else if (val > root->val)         root->right = insert(root-
>right, val);     return root;
}
int countNodesInRange(struct TreeNode* root, int low, int high) {
if (root == NULL)         return 0;     if (root->val < low)
return countNodesInRange(root->right, low, high);     if (root-
>val > high)
       return countNodesInRange(root->left, low, high);
   return 1 + countNodesInRange(root->left, low, high) + countNodesInRange(root->right, low, high);
} int main()
{
   struct TreeNode* root = createNode(10);
root->left = createNode(5);     root->right
= createNode(50);     root->left->left =
createNode(1);     root->right->left =
createNode(40);     root->right->right =
createNode(100);
   int low = 5, high = 45;
   int count = countNodesInRange(root, low, high);
   printf("Number of nodes in the range [%d, %d]: %d\n", low, high, count);
free(root->left->left);     free(root->right->left);     free(root->right->right);
free(root->left);     free(root->right);     free(root);     return 0; }
```

Output



Result

Thus the C program to count number of nodes that lie in the given range was successfully executed and the output was verified.

Ex No:2.5(b)	PROGRAM TO FIND THE SUM OF ALL THE LEAF NODES
Date:	

Aim Pseudo Code

To Write C program to find the sum of all the leaf nodes.

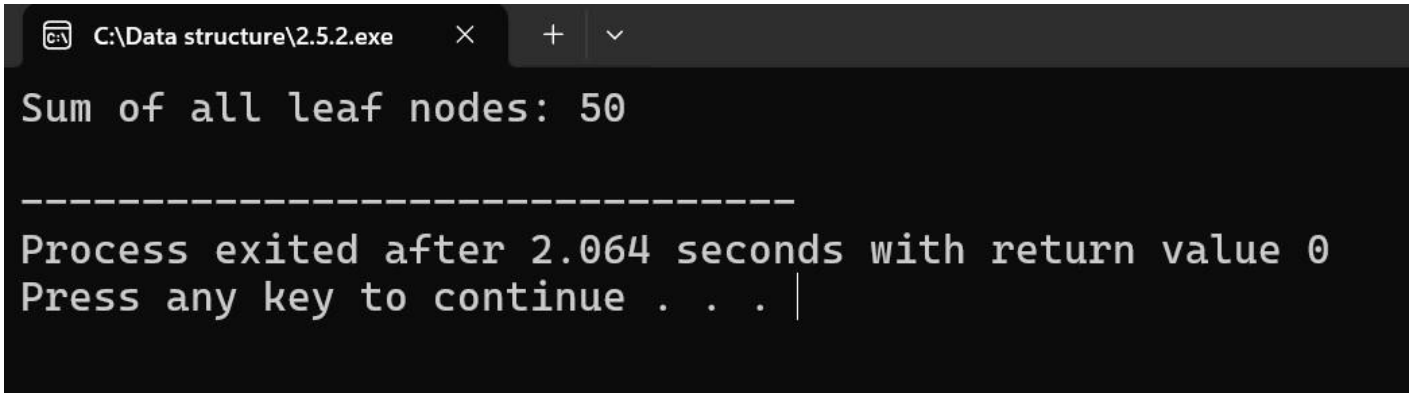
```
BEGIN
    Define a Structure Tree Node with an integer Variable create node (int val)
    {
    create a new Tree Node with the given and NULL left and right pointers
    }
    Insert (struct Tree Node*root, int Val) { if (val
        <root->val) root ->left = Insert (root ->left,
        val);
        else if (val> root ->val)
        return root;
    }
    Sum Of Loaf Nodes (struct Tree Node* toot) {
        Calculate the sum of leaf nodes in the binary search tree;  }
END
```

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct TreeNode {
   int val;
   struct TreeNode *left;
struct TreeNode *right;
};
struct TreeNode* createNode(int val) {
   struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->val = val;     newNode->left =
newNode->right = NULL;     return newNode;
}
```

```
struct TreeNode* insert(struct TreeNode* root, int val) {
if (root == NULL)         return createNode(val);     if
(val < root->val)         root->left = insert(root->left, val);
else if (val > root->val)         root->right = insert(root-
>right, val);     return root;
}
int sumOfLeafNodes(struct TreeNode* root) {
   if (root == NULL)
return 0;
   if (root->left == NULL && root->right == NULL)         return root-
>val;     return sumOfLeafNodes(root->left) + sumOfLeafNodes(root-
>right);
} int main()
{
   struct TreeNode* root = createNode(10);
   root->left = createNode(5);
   root->right = createNode(20);     root-
>left->left = createNode(4);     root->left-
>right = createNode(10);     root->right->left
= createNode(14);     root->right->right =
createNode(22);     int sum =
sumOfLeafNodes(root);     printf("Sum of all
leaf nodes: %d\n", sum);
   free(root->left->left);
free(root->left->right);
free(root->right->left);
free(root->right->right);
free(root->left);
free(root->right);
free(root);     return 0; }
```

Output



Result

Thus the program to find the sum of all the leaf nodes was successfully executed and the output was verified.

Ex No:2.5(c)	PROGRAM TO IMPLEMENTING THE FOLLOWING OPERATION IN BINARY TREE
Date:	

Aim

To Write C program for implementing the following operations in a Binary Tree. a) Insertion()

b) In-order Traversal()

c) Pre-order Traversal()

d) Post-order Traversal()

Pseudo Code

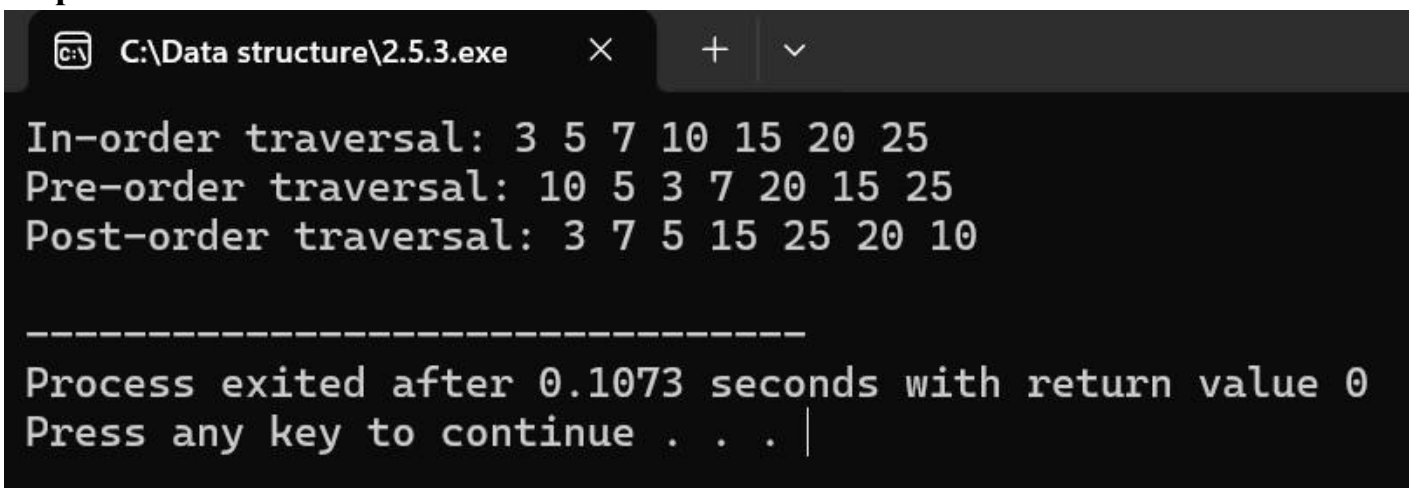
```
BEGIN
        Define a structure mede tree Node with integer Variable
        Create Node (int val) { create a new TreeNode with the given Value and NULL left
                and  right pointers
        }
        insert (struct Tree Node *root, int val) {
                if (root == NULL) return create
                Node (val);
                else if(val>root-> val) root -> right = insert (root
                        -> right, val) { return root
        }
        in Order (struct Tres Node *root) { perform in order
                traversal of the brinary search tree
        }
        preOrder (struct Tree Node *root) {
                Perform a preorder traversal of the binary search tree }
        Post order (struct tree node *root){
                Perform a post order traversal of the binary search tree
        }
END
```

Source Code

```
#include <stdio.h> #include
<stdlib.h>
struct TreeNode {
   int val;
   struct TreeNode *left;
struct TreeNode *right;
}; struct TreeNode* createNode(int val) {     struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));     if (newNode == NULL) {
printf("Memory allocation failed\n");         exit(1);
   }
   newNode->val = val;     newNode->left =
newNode->right = NULL;     return newNode;
}
struct TreeNode* insert(struct TreeNode* root, int val) {
if (root == NULL)         return createNode(val);     if
(val < root->val)         root->left = insert(root->left, val);
else if (val > root->val)
       root->right = insert(root->right, val);
   return root; } void inOrder(struct
TreeNode* root) {
   if (root == NULL)
       return;     inOrder(root-
>left);     printf("%d ", root-
>val);     inOrder(root->right);
} void preOrder(struct TreeNode* root)
{
   if (root == NULL)
       return;     printf("%d
", root->val);
preOrder(root->left);
preOrder(root->right);
}
void postOrder(struct TreeNode* root) {
if (root == NULL)
       return;     postOrder(root-
>left);     postOrder(root->right);
printf("%d ", root->val);
} int main() {     struct TreeNode*
root = NULL;     root =
insert(root, 10);     root =
insert(root, 5);     root =
insert(root, 20);     root =
insert(root, 3);     root =
insert(root, 7);     root =
insert(root, 15);     root =
insert(root, 25);     printf("In-order
traversal: ");     inOrder(root);
```

```
printf("\n");     printf("Pre-order
traversal: ");     preOrder(root);
printf("\n");     printf("Post-order
traversal: ");     postOrder(root);
printf("\n");     return 0;
}
```

Output



Result

Thus the program to find the sum of all the leaf nodes was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:2.6(a)	**PROGRAM FOR FINDING A PATH**
Date:	

Aim Pseudo Code

` To Write C program for finding a path.

BEGIN

Define a structure Tree Node with a boolean Variable

Create Node (bool in store){

Create of the new Tree node with the given value of ' is store' and NULL left and right pointers.

is Store (struct TreeNode *root) { return root -> is store;

}

Can Reach Store (struct Tree Node * root) {

if (root == NULL) return false;

if (isStore (root)) return

true;

return Can Reach Store (root →left) // con Reach Store ->right); }

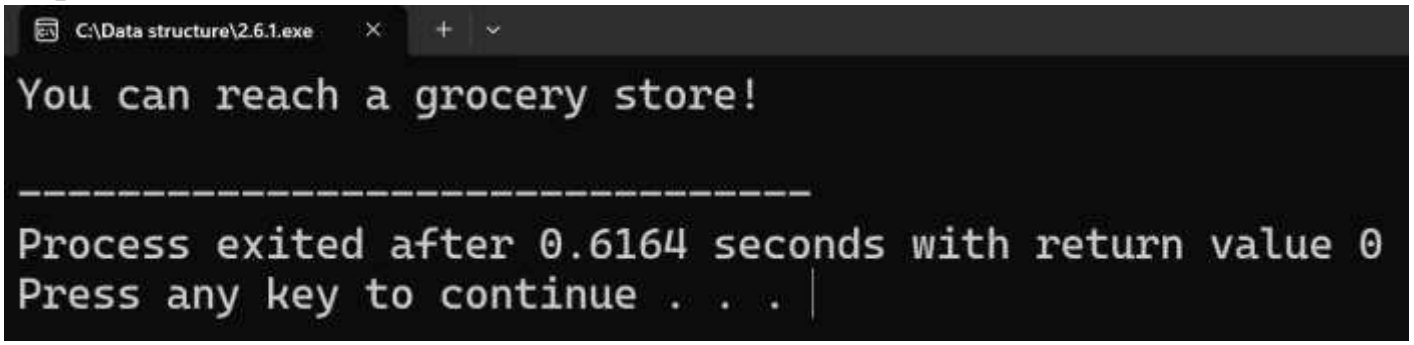
END

Source Code

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h> struct
TreeNode {     bool
isStore;     struct
TreeNode *left;     struct
TreeNode *right;
```

```
};
struct TreeNode* createNode(bool isStore) {
   struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
if (newNode == NULL) {        printf("Memory allocation failed\n");        exit(1);
   }
   newNode->isStore = isStore;     newNode-
>left = newNode->right = NULL;     return
newNode;
}
bool isStore(struct TreeNode* root) {
return root->isStore;
}
bool canReachStore(struct TreeNode* root) {
   if (root == NULL)
return false;     if
(isStore(root))
return true;
   return canReachStore(root->left) || canReachStore(root->right);
} int main()
{
   struct TreeNode* root = createNode(false);
root->left = createNode(true);     root->right
= createNode(false);     root->right->left =
createNode(true);     root->right->right =
createNode(true);     if (canReachStore(root))
      printf("You can reach a grocery store!\n");
else
      printf("You cannot reach a grocery store.\n");
return 0;
}
```

Output



Result

Thus the program for finding a path was successfully executed and the output was verified.

Ex No:2.6(b)	PROGRAM TO IMPLEMENTATION BFS AND DFS TRAVEL METHODS
Date:	

Aim Pseudo Code

To Write C program to implement BFS and DFS Traversal methods.

BEGIN

Define a Structure Node with a integer Variabile data and a pointer to the next Node 'next'
Create Node (int data) { create a new node with the given data and
NULL next pointer
}
create Graph (int num Vertices) { create a new graph with the given number of Vertices, initialize
adjacency lits and Visited array;

```
        }
        add edge(struct graph * graph ,int src,int test){ add
        an edge between source and destination; }
        DFS (struct Graph * graph, intvertex) {
                Graph ->  Visited [vertex] = true;
        Perform depth-first search traversal starting from the given vertex }
        BFS (struct Graph * graph, int Start Ventex) {
        Perform Breadth -first search traversal for starting vertex; }
        For graph (struct graph *graph){
                Free (graph)
        }
END
```

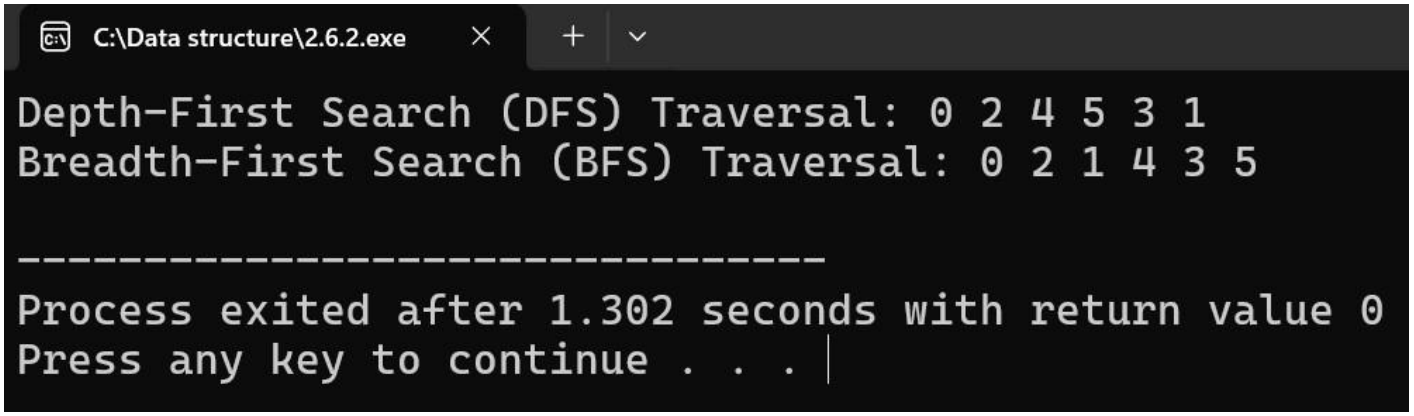
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h> struct
Node {
   int data;
   struct Node* next;
}; struct Graph {     int
numVertices;     struct
Node** adjLists;
   bool* visited;
};
struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = data;     newNode->next = NULL;     return
newNode;
}
struct Graph* createGraph(int numVertices) {
   struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));     graph-
>numVertices = numVertices;     graph->adjLists = (struct
Node**)malloc(numVertices * sizeof(struct Node*));     graph->visited =
(bool*)malloc(numVertices * sizeof(bool));  int i;
   for (i = 0; i < numVertices; i++) {         graph-
>adjLists[i] = NULL;
       graph->visited[i] = false;
}
   return graph; }
void addEdge(struct Graph* graph, int src, int dest) {
struct Node* newNode = createNode(dest);
newNode->next = graph->adjLists[src];     graph-
>adjLists[src] = newNode;     newNode =
createNode(src);     newNode->next = graph-
>adjLists[dest];     graph->adjLists[dest] = newNode;
} void DFS(struct Graph* graph, int vertex) {
graph->visited[vertex] = true;     printf("%d ",
vertex);     struct Node* adjList = graph-
>adjLists[vertex];     while (adjList != NULL) {
```

```
int connectedVertex = adjList->data;        if
(!graph->visited[connectedVertex]) {
          DFS(graph, connectedVertex);
       }
       adjList = adjList->next;
   }
}
void BFS(struct Graph* graph, int startVertex) {     int
queue[graph->numVertices];     int front = 0, rear = -1;
queue[++rear] = startVertex;     graph-
>visited[startVertex] = true;     while (front <= rear) {
int currentVertex = queue[front++];        printf("%d ",
currentVertex);        struct Node* temp = graph-
>adjLists[currentVertex];        while (temp != NULL) {
int adjVertex = temp->data;            if (!graph-
>visited[adjVertex]) {               queue[++rear] =
adjVertex;
             graph->visited[adjVertex] = true;
          }
          temp = temp->next;
       }
   }
}
void freeGraph(struct Graph* graph) {
int i;
   for (i = 0; i < graph->numVertices; i++) {
struct Node* adjList = graph->adjLists[i];
while (adjList != NULL) {            struct
Node* temp = adjList;            adjList =
adjList->next;            free(temp);
       }
   }
   free(graph->adjLists);
free(graph->visited);
free(graph); } int main()
{
   struct Graph* graph = createGraph(6);
   addEdge(graph, 0, 1);     addEdge(graph, 0, 2);
addEdge(graph, 1, 3);     addEdge(graph, 1, 4);
addEdge(graph, 2, 4);     addEdge(graph, 3, 4);
addEdge(graph, 3, 5);     addEdge(graph, 4, 5);
printf("Depth-First Search (DFS) Traversal: ");
   DFS(graph, 0);
printf("\n"); int i;
   for (i = 0; i < graph->numVertices; i++) {
      graph->visited[i] = false;
   }
   printf("Breadth-First Search (BFS) Traversal: ");
```

```
   BFS(graph, 0);
printf("\n");
freeGraph(graph);     return
0;
}
```

Output



Result

Thus the program to implement BFS and DFS Traversal methods was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:3.1(a)	PROGRAM USING BUBBLE SORT
Date:	

Aim Pseudo Code

To write a C program to perform bubble sort.

```
BEGIN
VOID BubbleSort(int arr[],int size)
        FOR(i=0;i<size-1;i++)
                FOR(j=0;j<size-i-1;j++)
                        IF(arr[j]>arr[j+1]) temp=arr[j];
                        arr[j]=arr[j+1] arr[j+1]=temp;
END
```

Source Code

```
#include <stdio.h>  int main() { int
arr[50], num, x, y, temp;
        printf("Please Enter the Number of Elements you want in the array: "); scanf("%d",
        &num);
          printf("Please Enter the Value of Elements: "); for(x
        = 0; x < num; x++)
          scanf("%d", &arr[x]);
          for(x = 0; x < num - 1; x++) { for(y
          = 0; y < num - x - 1; y++)
            { if(arr[y] > arr[y + 1]) {
            temp = arr[y]; arr[y]
              = arr[y + 1];
              arr[y + 1] = temp; } } } printf("Array
        after implementing bubble sort: "); for(x = 0; x
        < num; x++) { printf("%d ", arr[x]); }
        return 0;}
```

Output

```
Number of Elements: 6
Please Enter the Value of Elements: 2 4 5 1 3 6
Array after implementing bubble sort: 1 2 3 4 5 6
--------------------------------
Process exited after 11.2 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program to implement bubble sort was successfully executed and the output was verified.

Ex No:3.1(b)	PROGRAM TO IMPLEMENT INSERTION SORT
Date:	

Aim Pseudo Code

BEGIN

To write a C program to implement insertion sort.

VOID InsertionSort(int arr[],int size) for(j=i-1;(j>=0) &&temp<arr[j];j--) arr[j+1]=arr[j]; arr[j+1]=temp

END

Source Code

```
#include <stdio.h> void printArray(int
array[], int size) { for(int i = 0; i <
size; i++) { printf("%d ", array[i]);
} printf("\n"); }
void insertionSort(int array[], int size) {
for(int step = 1; step < size; step++) {
int key = array[step]; int j = step - 1;
while (key < array[j] && j >= 0) { array[j
+ 1] = array[j];
--j; } array[j + 1] = key;
}}
int main() { int data[] = {5,10,12,8,14}; int size =
sizeof(data) / sizeof(data[0]);
insertionSort(data, size);
printf("Sorted array in ascending order:\n");
printArray(data, size); return 0;
}
```

Output

```
Sorted array in ascending order:
5 8 10 12 14

--------------------------------
Process exited after 0.04933 seconds with return value 0
Press any key to continue . . .
```

Result Thus, the program to implement bubble sort was successfully executed and the output was verified.

Ex No:3.1(c)	PROGRAM TO IMPLEMENT MERGE SORT
Date:	

Aim Pseudo Code

BEGIN

To write a C program to implement merge sort.

function mergeSort(array) if length of array <= 1. return array. middle = length of array / 2. leftArray = mergeSort(first half of array) rightArray = mergeSort(second half of array)
return merge(leftArray, rightArray)

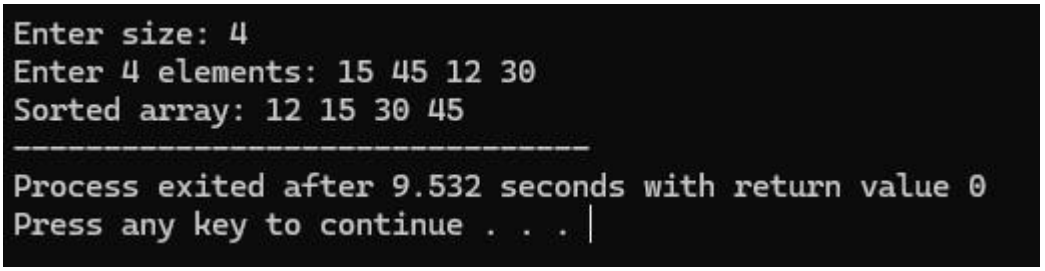
END **Source**

Code

```
#include <stdio.h> int mergeSort(int a[], int low, int mid,
        int high) {
          int k, l = low; int i = low; int m = mid
          + 1; int temp[50]; while ((l <= mid)
          && (m <= high)) { if
             (a[l] <= a[m]) temp[i++]
                = a[l++];
             else
                temp[i++] = a[m++];    }
          while (l <= mid) temp[i++]
             = a[l++];
          while (m <= high)
             temp[i++] = a[m++];
          for (k = low; k <= high; k++)
             a[k] = temp[k];}
        int partition(int arr[], int low, int high) { if
          (low < high) { int mid = (low + high) /
             2; partition(arr, low, mid);
             partition(arr,
             mid + 1, high); mergeSort(arr, low, mid,
             high);    }}

int main() { int arr[50], n, i;
          printf("Enter size: ");
          scanf("%d", &n);

          printf("Enter %d elements: ", n);
       for (i = 0; i < n; i++) {
             scanf("%d", &arr[i]);
             } partition(arr, 0, n -
             1); printf("Sorted
             array: "); for (i = 0;
             i
             < n; i++) { printf("%d
                ", arr[i]);
```

```
    }
        return 0;
}
```

Output



Result

Thus, the program to implement merge sort was successfully executed and the output was verified.

Ex No:3.1(d)	PROGRAM TO IMPLEMENT QUICK SORT
Date:	

Aim Pseudo Code

BEGIN

To write a C program to implement quick sort.

function partitionFunc(left, right, pivot)
leftPointer = left rightPointer = right - 1 while True do while
A[++leftPointer] < pivot do end while while rightPointer > 0
&& A[--rightPointer] > pivot do end
while
if leftPointer >= rightPointer
break
else swap
leftPointer,rightPointer
end if
end while swap leftPointer,right
return
leftPointer
end function
END

Source Code

```
#include<stdio.h> void quicksort(int number[25], int
first, int last) { int i, j, pivot, temp; if(first < last)
{ pivot = first; i = first; j = last; while(i
< j) {
while(number[i] <= number[pivot] && i < last)
i++;
while(number[j] > number[pivot]) j--
;
if(i < j) {
temp = number[i];
number[i] = number[j];
number[j] = temp; }
} temp = number[pivot];
number[pivot] = number[j];
number[j] = temp; quicksort(number, first,
j - 1);
```

Output

```
No of elements: 5
Enter 5 elements: 14 53 10 21 24
Sorted elements: 10 14 21 24 53
--------------------------------
Process exited after 22.77 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program to implement quick sort was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:3.2(a)	PROGRAM TO INSERT KEYS INTO HASH TABLE USING LINEAR PROBING
Date:	

Aim Pseudo Code

To write a C program to insert given keys into hash table using linear probing.

```
BEGIN
    void countingSort(int array[], int size)
      for (int i = 0; i < size; i++)
        count[array[i]]++;
      for (int i = 1; i <= MAX_VALUE; i++)
        count[i] += count[i - 1];
      for (int i = size - 1; i >= 0; i--)
        output[count[array[i]] - 1] =
        array[i]; count[array[i]]--;
      for (int i = 0; i < size; i++) array[i]
        = output[i];
END
```

Source Code

```
#include <stdio.h> int LinearProbing(int arr[], int n, int
    table_size) {
      int i, j, HashKey, element; for (i = 0; i
      < n; i++) { printf("Enter [%d] element:
      ", i + 1); scanf("%d", &element);
        j = 0;
        while (1) {
          HashKey = ((element % table_size) + j) % table_size; if
          (arr[HashKey] == -1) {
            arr[HashKey] = element; break;
          } else
            j++;
        }
      }
} int main() { int table_size, n, arr[50], i;
printf("Enter table_size: "); scanf("%d",
&table_size); printf("Enter number of elements: ");
scanf("%d", &n); for (i = 0; i < table_size; i++)
arr[i] = -1;
      LinearProbing(arr, n, table_size);
      printf("Hash Table:\n");
```

```
    for (i = 0; i < table_size; i++)
        { printf(" %d %d\n", i,
        arr[i]);
    }
}
```

Output

```
Enter table_size: 5
Enter number of elements: 4
Enter [1] element: 2
Enter [2] element: 4
Enter [3] element: 6
Enter [4] element: 8
Hash Table:
 0 -1
 1 6
 2 2
 3 8
 4 4

--------------------------------
Process exited after 21.38 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program to insert given keys into hash table using linear probing was successfully executed and the output was verified.

Ex No:3.2(b)	PROGRAM TO INSERT KEYS INTO HASH MAP USING QUADRATIC PROBING
Date:	

Aim Pseudo Code

BEGIN

To write a C program to insert given keys into hash map using quadratic probing.

```
void QuadraticProbing(int arr[], int n, int table_size)
    while (1)
       HashKey = ((element % table_size) + j * j) % table_size;
       if (arr[HashKey] == -1) { arr[HashKey] = element;
          break;
       else
          j++;
```

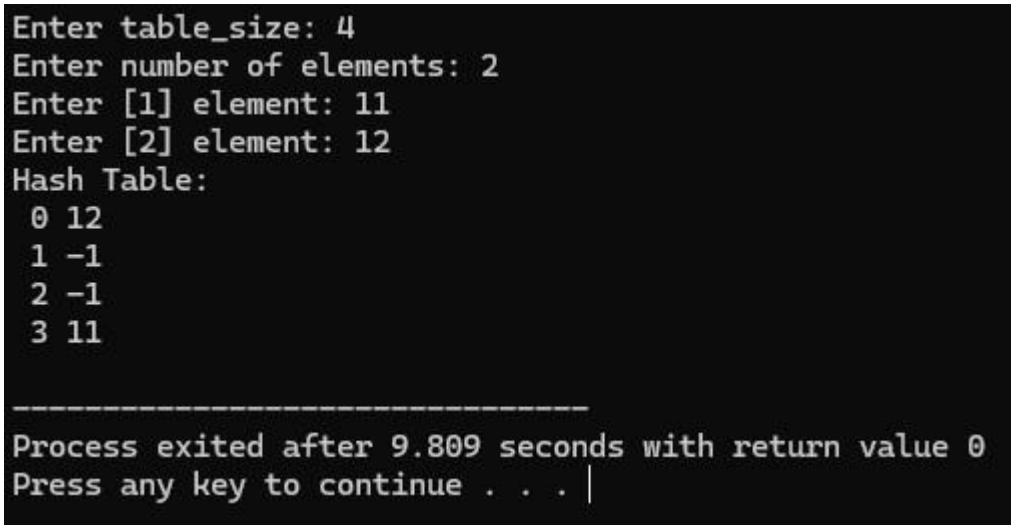
END

Source Code

```
#include <stdio.h> void QuadraticProbing(int arr[], int n, int
table_size) { int i, j, HashKey, element; for(i = 0; i < n;
i++) { printf("Enter [%d] element: ", i + 1);
      scanf("%d", &element);
      j = 0; while
      (1) {
         HashKey = ((element % table_size) + j * j) % table_size; if
         (arr[HashKey] == -1) { arr[HashKey]
            = element; break;
           } else j++;
      }
   }
} int main() { int table_size, n, arr[50], i;
printf("Enter table_size: "); scanf("%d",
   &table_size); printf("Enter number of
   elements: "); scanf("%d", &n); for (i
   = 0; i < table_size; i++) arr[i] = -1;
```

```
QuadraticProbing(arr, n, table_size); printf("Hash Table:\n");
for (i = 0; i < table_size; i++) {
printf(" %d %d\n", i, arr[i]);
   }
}
```

Output



Result

Thus, the program to insert given keys into hash map using quadratic probing was successfully executed and the output was verified.

Ex No:3.2(c)	PROGRAM TO INSERT KEYS INTO THE HASH TABLE USING DOUBLE HASHING
Date:	

Aim Pseudo Code

BEGIN

To write a C program to insert given keys into hash table using double hashing.

```
void DoubleHashing(int arr[], int n, int table_size) {
    while (1)
      HashKey = ((element % table_size) + j * r - (element % r)) % table_size if
       (arr[HashKey] == -1) arr[HashKey]
          = element;
          break;
       else
          j++;
```

END

Source Code

```
#include <stdio.h> void DoubleHashing(int arr[], int n, int
    table_size) { int i, j, HashKey, element;
      int flag = 0; int r
      = 7;
      for (i = 0; i < table_size; i++)
        { if (flag == 1) break;
      }
      for (i = 0; i < n; i++) { printf("Enter
        [%d] element: ", i + 1);
        scanf("%d", &element);
        j = 0; while
        (1) {
          HashKey = ((element % table_size) + j * r - (element % r)) % table_size; if
```

```
                (arr[HashKey] == -1) { arr[HashKey]
                   = element; break;
                } else
                   j++;       }    }
} int main() { int table_size, n, arr[50], i;
printf("Enter table_size: "); scanf("%d",
          &table_size); printf("Enter number of
          elements: "); scanf("%d", &n); for (i
          = 0; i < table_size; i++)
              arr[i] = -1;
              DoubleHashing(arr,          n,
             table_size);          printf("Hash
             Table:\n");
             for (i = 0; i < table_size; i++)
                {  printf("  %d  %d\n",  i,
                arr[i]);
             }
          }
```

Output

```
Enter table_size: 4
Enter number of elements: 4
Enter [1] element: 13
Enter [2] element: 14
Enter [3] element: 15
Enter [4] element: 16
Hash Table:
 0 15
 1 14
 2 13
 3 16

--------------------------------
Process exited after 15.5 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program to using linked list was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		

Ex No:3.3(a)	**PROGRAM TO IMPLEMENT LINEAR SEARCH**
Date:	

Aim Pseudo Code

BEGIN

To write a C program to implement linear search.

int linearSearch(int a[], int n, int val)
for (int i = 0; i < n; i++) if (a[i] == val) return i + 1
return -1

END

Source Code

```
#include <stdio.h> int linearSearch(int a[], int
n, int val) { for (int i = 0; i < n; i++) {
if (a[i] == val)
return i + 1;
}
return -1;
} int main()
{
int a[] = {48,65,23,57,98,51,45,62};
int val;
int n = sizeof(a) / sizeof(a[0]); printf("The
elements of the array: ");
```

```
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\nElement to be searched: ");
    scanf("%d",&val);    int    res    =
    linearSearch(a, n, val); if
    (res == -1)
    printf("\nElement is not present in the array");
    Else printf("\nElement is present at %d position of array", res);
    return 0;
}
```

Output

```
The elements of the array: 48 65 23 57 98 51 45 62
Element to be searched: 98

Element is present at 5 position of array
--------------------------------
Process exited after 10.5 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program was to implement linear search was successfully executed and the output was verified.

Ex No:3.3(b)	PROGRAM TO IMPLEMENT BINARY SEARCH
Date:	

Aim Pseudo Code

BEGIN

To write a C program to implement binary seach.

```
int binarySearch(int a[], int beg, int end, int val)
    if (end >= beg) mid = (beg + end) / 2;
    if
      (a[mid] == val) return
        mid + 1 else
      if (a[mid] < val) return binarySearch(a, mid
        + 1, end, val)
       Else return binarySearch(a, beg, mid - 1, val)
     return -1;
```

END

Source Code

```
#include <stdio.h> int binarySearch(int a[], int beg, int end,
    int val) { int mid;
      if (end >= beg) { mid =
        (beg + end) / 2; if
```

```
            (a[mid] == val) { return
               mid + 1;
            }  else  if  (a[mid]  <  val)  {  return
               binarySearch(a, mid + 1, end, val);
            } else { return binarySearch(a, beg, mid -
          1, val); } }
          return -1;
} int main()
{
          int a[] = {52,11,21,78,92,14,56,32,57};
          int val; int n = sizeof(a) / sizeof(a[0]);
          printf("The elements of the array are: ");
          for (int i = 0; i < n; i++) printf("%d ",
          a[i]);
          printf("\nElement  to  be  searched  ");
          scanf("%d",&val);
          int res = binarySearch(a, 0, n - 1, val); if
          (res == -1)
            printf("\nElement is not present in the array: ");
              else
              printf("\nElement is present at %d position of array.", res);
           return 0;
          }
```

Output

```
The elements of the array are: 52 11 21 78 92 14 56 32 57
Element to be searched 21

Element is present at 3 position of array.
--------------------------------
Process exited after 2.484 seconds with return value 0
Press any key to continue . . .
```

Result

Thus, the program to implement binary search was successfully executed and the output was verified.

ALGORITHM	**15**	
PROGRAM	**30**	
EXECUTION	**30**	
OUTPUT & RESULT	**15**	
VIVA	**10**	
TOTAL	**100**	
INITIAL OF FACULTY		