



NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY
Department of Computer Science Engineering

LAB PRACTICAL FILE

Data Structures

Professor
Prof. R.S. Rao

Student
Kushagra Lakhwani

Academic Session 2021-2022

Contents

1	Mean and Median	4
1.1	Objective	4
1.2	Algorithm	4
1.3	Code	5
1.4	Output	6
2	Array Insertion/Deletion	7
2.1	Objective	7
2.2	Algorithm	7
2.2.1	Deletion	7
2.2.2	Insertion	7
2.3	Code	7
2.4	Output	8
3	Search inside Array	9
3.1	Objective	9
3.2	Algorithm	9
3.3	Code	9
3.4	Output	10
4	Sort Array	11
4.1	Objective	11
4.2	Algorithm	11
4.3	Code	11
4.4	Output	12
5	Linked List	13
5.1	Objective	13
5.2	Algorithm	13
5.3	Code	13
5.4	Output	15
6	Linked List Deletion	16
6.1	Objective	16
6.2	Algorithm	16
6.3	Code	17
6.4	Output	19
7	Stack	20
7.1	Objective	20
7.2	Algorithm	20
7.3	Code	20
7.4	Output	22

8	Linked List Operations	23
8.1	Objective	23
8.2	Algorithm	23
8.2.1	Traversal	23
8.2.2	Insertion	23
8.2.3	Deletion	24
8.3	Code	25
8.4	Output	28
9	Inorder Traversal	29
9.1	Objective	29
9.2	Algorithm	29
9.3	Code	29
9.4	Output	30
10	Depth First Search	31
10.1	Objective	31
10.2	Algorithm	31
10.3	Code	31
10.4	Output	33
11	Breath First Search	35
11.1	Objective	35
11.2	Algorithm	35
11.3	Code	35
11.4	Output	35

1 Mean and Median

1.1 Objective

Write a program to find the mean and the median of the numbers stored in an array.

1.2 Algorithm

```
1 Start
2 Step 1 -> declare function to calculate mean
3     double mean(int arr[], int size)
4     declare int sum = 0
5     Loop For int i = 0 and i < size and i++
6     Set sum += arr[i]
7 End
8     return (double)sum/(double)size
9
10 Step 2 -> declare function to calculate median
11     double median(int arr[], int size)
12     call sort(arr, arr+size)
13     IF (size % 2 != 0)
14     return (double)arr[size/2]
15 End
16     return (double)(arr[(size-1)/2] + arr[size/2])/2.0
17
18 Step 3 -> In main()
19     Declare int arr[] = {3,5,2,1,7,8}
20     Declare int size = sizeof(arr)/sizeof(arr[0])
21     Call mean(arr, size)
22     Call median(arr, size)
23 Stop
```

1.3 Code

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int arr[50], n;
    printf("Enter the size of the arr:\n");
    scanf("%d", &n);
    printf("enter the elements: \n");
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    printf("Your array is:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    printf("your sorted array here: \n");
    int i, j, a;
    for (i = 0; i < n; ++i)
        for (j = i + 1; j < n; ++j)
            if (arr[i] > arr[j])
            {
                a = arr[i];
                arr[i] = arr[j];
                arr[j] = a;
            }
    printf("Your array is:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        int a = arr[i];
        sum += a;
    }
    int mean = sum / n;
    int median = arr[n / 2];

    printf("\n");
    printf("your mean is given as: %d\n", mean);
    printf("your median is given as: %d\n", median);
}
```

1.4 Output

```
Enter the size of the arr:
5

enter the elements:
21
23
44
55
11

Your array is:
21 23 44 55 11
your sorted array here:
Your array is:
11 21 23 44 55

your mean is given as: 30
your median is given as: 23
```

2 Array Insertion/Deletion

2.1 Objective

Write a Program to insert and delete an element from an array.

2.2 Algorithm

2.2.1 Deletion

```
1 Start
2 Set J = K
3 Repeat steps 4 and 5 while J < N
4     Set LA[J] = LA[J + 1]
5     Set J = J+1
6 Set N = N-1
7 Stop
```

2.2.2 Insertion

```
1 Begin
2 IF N = MAX, return
3 ELSE
4 N = N + 1
5 SEEK Location index
6 For All Elements from A[index] to A[N]
7     Move to next adjacent location
8     A[index] = New_Element
9 End
```

2.3 Code

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int arr[100];
    int i, item, pos, size = 7;
    printf("Enter 7 elements: ");
    // reading array
    for (i = 0; i < size; i++)
        scanf("%d", &arr[i]);
    // print the original array
    printf("Array before insertion: ");
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
```

```

    // read element to be inserted
    printf("Enter the element to be inserted: ");
    scanf("%d", &item);
    // read position at which element is to be inserted
    printf("Enter the position at which the element is to be inserted: ");
    scanf("%d", &pos);
    // increase the size
    size++;
    // shift elements forward
    for (i = size - 1; i >= pos; i--)
        arr[i] = arr[i - 1];
    // insert item at position
    arr[pos - 1] = item;
    // print the updated array
    printf("Array after insertion: ");
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}

```

2.4 Output

```

Enter 7 elements: 2 4 6 5 1 5 7
Array before insertion: 2 4 6 5 1 5 7
Enter the element to be inserted: 3
Enter the position at which the element is to be inserted: 2
Array after insertion: 2 3 4 6 5 1 5 7

```


3 Search inside Array

3.1 Objective

Write a program to search for a number in array.

3.2 Algorithm

```
1 Start
2   [Initialize counter variable. ] Set i = 0
3   Repeat Step 04 and 05 for i = 0 to i < n
4   if a[i] = x, then jump to step 07
5   [Increase counter. ] Set i = i + 1
6   [End of step 03 loop. ]
7   Print x found at i + 1 position and go to step 09
8   Print x not found (if a[i] != x, after all the iteration of the
9   above for loop. )
10 Stop
```

3.3 Code

```
#include <stdio.h>
int main(int argc, char const *argv[])
{
    int arr[10], Size, i, Search, Flag;
    printf("\n Please Enter the size of an array : ");
    scanf("%d", &Size);
    printf("\n Please Enter %d elements of an array: \n", Size);
    for (i = 0; i < Size; i++)
        scanf("%d", &arr[i]);
    printf("\n Please Enter the Search Element : ");
    scanf("%d", &Search);
    Flag = 0;
    for (i = 0; i < Size; i++)
        if (arr[i] == Search)
        {
            Flag = 1;
            break;
        }
    if (Flag == 1)
        printf("\n We found the Search Element %d at Position %d ", Search, i + 1);
    else
        printf("\n Sorry!! We haven't found the the Search Element %d ", Search);
}
```

```
    return 0;  
}
```

3.4 Output

```
Please Enter the size of an array : 5  
  
Please Enter 5 elements of an array:  
1 2 3 4 5  
  
Please Enter the Search Element : 2  
  
We found the Search Element 2 at Position 2
```

4 Sort Array

4.1 Objective

Write a program to sort an array

4.2 Algorithm

```
1 START
2   INITIALIZE arr[] = {5, 2, 8, 7, 1 }..
3   SET temp =0
4   length= sizeof(arr)/sizeof(arr[0])
5   PRINT "Elements of Original Array"
6   SET i=0 REPEAT STEP 7 and STEP 8 UNTIL i<length
7   PRINT arr[i]
8   i=i+1
9   SET i=0 REPEAT STEP 10 to STEP UNTIL i<n
10  SET j=i+1 REPEAT STEP 11 UNTIL j<length
11  if(arr[i]>arr[j]) then
12      temp = arr[i]
13      arr[i]=arr[j]
14      arr[j]=temp
15      j=j+1
16      i=i+1
17  PRINT new line
18  PRINT "Elements of array sorted in ascending order"
19  SET i=0 REPEAT STEP 17 and STEP 18 UNTIL i<length
20  PRINT arr[i]
21  i=i+1
22  RETURN 0
23 END.
```

4.3 Code

```
#include <stdio.h>
int main()
{
    // Initialize array
    int arr[] = {5, 2, 8, 7, 1};
    int temp = 0;
    // Calculate length of array arr
    int length = sizeof(arr) / sizeof(arr[0]);
    // Displaying elements of original array
    printf("Elements of original array: \n");
    for (int i = 0; i < length; i++)
        printf("%d ", arr[i]);
    // Sort the array in ascending order
```

```

for (int i = 0; i < length; i++)
    for (int j = i + 1; j < length; j++)
        if (arr[i] > arr[j])
        {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
printf("\n");
// Displaying elements of array after sorting
printf("Elements of array sorted in ascending order: \n");
for (int i = 0; i < length; i++)
    printf("%d ", arr[i]);
return 0;
}

```

4.4 Output

```

Elements of original array:
5 2 8 7 1
Elements of array sorted in ascending order:
1 2 5 7 8

```

5 Linked List

5.1 Objective

Write a program to implement a linked list.

5.2 Algorithm

```
1 Create a class Node which has two attributes: data and next. Next is a
  pointer to the next node.
2 Create another class which has two attributes: head and tail.
3
4 addNode() will add a new node to the list:
5   a. Create a new node.
6   b. It first checks, whether the head is equal to null which means the
  list is
7   empty.
8   c. If the list is empty, both head and tail will point to the newly
  added node.
9   d. If the list is not empty, the new node will be added to end of the
  list such that tail's next will point to the newly added node. This new
  node will become the new tail of the list.
10
11 display() will display the nodes present in the list:
12
13 Define a node current which initially points to the head of the list.
14   a. Traverse through the list till current points to null.
15   b. Display each node by making current to point to node next to it in
  each iteration.
```

5.3 Code

```
#include <stdio.h>
#include <stdlib.h>
// Represent a node of singly linked list
struct node
{
    int data;
    struct node *next;
};
// Represent the head and tail of the singly linked list
struct node *head, *tail = NULL;
// addNode() will add a new node to the list
void addNode(int data)
{
    // Create a new node
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
```

```

newNode->data = data;
newNode->next = NULL;
// Checks if the list is empty
if (head == NULL)
{
    // If list is empty, both head and tail will point to new node
    head = newNode;
    tail = newNode;
}
else
{
    // newNode will be added after tail such that tail's next will point to newNod
    tail->next = newNode;
    // newNode will become new tail of the list
    tail = newNode;
}
}
// display() will display all the nodes present in the list
void display()
{
    // Node current will point to head
    struct node *current = head;
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    printf("Nodes of singly linked list: \n");
    while (current != NULL)
    {
        // Prints each node by incrementing pointer
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}
int main()
{
    // Add nodes to the list
    addNode(1);
    addNode(2);
    addNode(3);
    addNode(4);
    // Displays the nodes present in the list

```

```
    display();  
    return 0;  
}
```

5.4 Output

```
Nodes of singly linked list:  
1 2 3 4
```

6 Linked List Deletion

6.1 Objective

Write a program to insert and delete a node from linked list.

6.2 Algorithm

```
1 Start
2 Step 1 -> Implement a function to insert at the beginning of linked list
3     void insertAtBeginning(struct Node** head, int newdata)
4         Allocate memory for new node
5         Set data of new node = newdata
6         Set next pointer of new node = head
7         Set head pointer = new node
8 End
9 Step 2 -> Implement a function to insert at the end of the linked list
10    void insertAtEnd(struct Node** head, int newdata)
11        Allocate memory for newNode
12        Set data of newNode = newdata
13        Set next pointer of newNode = NULL
14        Let lastNode = Head
15        If(lastNode = NULL)
16            Set head = newNode
17            return
18        Else
19            Traverse till lastNode -> next!=NULL
20            Set lastNode -> next=newNode
21 return
22 End
23 Step 3 -> Implement a function to insert after a given node
24 void insertAfter(struct Node* prev_Node, int newdata)
25     if(prev_Node = NULL)
26         print("previous node cannot be NULL")
27         return
28     Else
29         Allocate memory for newNode
30         Set data of newNode = newdata
31         Set next pointer of newNode = prev_Node -> next
32         Set next pointer of prev_Node = newNode
33 End
34 Step 4 -> Implement a function to delete a given node
35 void deleteNode(struct Node** head, int key)
36     Let Node*temp = head
37     Traverse while(temp!=NULL)
38     If(temp -> data=key)
39         Free(temp)
40         Return
41     Else
42         temp=temp -> next
43 Stop
```


6.3 Code

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
void InsertAtBeginning(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    *head_ref = new_node;
}
void insertAfter(struct Node *prev_node, int new_data)
{
    if (prev_node == NULL)
    {
        printf("the given previous node cannot be NULL");
        return;
    }
    struct Node *new_node =
        (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}
void InsertAtEnd(struct Node **head_ref, int new_data)
{
    struct Node *new_node =
        (struct Node *)malloc(sizeof(struct Node));
    struct Node *last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
}
```

```

        return;
    }
void deleteNode(struct Node **head_ref, int key)
{
    struct Node *temp = *head_ref, *prev;
    if (temp != NULL && temp->data == key)
    {
        *head_ref = temp->next;
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL)
        return;
    prev->next = temp->next;
    free(temp);
}
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}
int main()
{
    struct Node *head = NULL;
    append(&head, 6);
    push(&head, 7);
    push(&head, 1);
    append(&head, 4);
    insertAfter(head->next, 8);
    printf("Created Linked list is: ");
    printList(head);
    return 0;
}

```

6.4 Output

```
Created Linked list is: 1 7 8 6 4  
Linked list after deletion: 1 7 6 4
```

7 Stack

7.1 Objective

Write a program to implement a stack using an array.

7.2 Algorithm

```
1 Start
2 Step 1 -> Implement a function to push elements into stack
3     void push()
4         if(top > size of array -1)
5             print("stack overflow")
6         else
7             input the value to be pushed
8             top=top+1
9             stack[top]=new_element
10 End
11 Step 2 -> Implement a function to pop from stack
12     void pop()
13         if(top < -1)
14             print("stack underflow")
15         else
16             delete stack[top]
17             top=top-1
18     End
19 Stop
```

7.3 Code

```
#include <stdio.h>
int stack[100], choice, n, top, x, i;
void push(void);
void pop(void);
void display(void);
int main(int argc, char const *argv[])
{
    top = -1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d", &n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-----");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d", &choice);
```

```

        switch (choice)
        {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("\n\t EXIT POINT ");
                break;
            default:
                printf("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
    } while (choice != 4);
    return 0;
}

void push()
{
    if (top >= n - 1)
        printf("\n\tSTACK is over flow");
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d", &x);
        top++;
        stack[top] = x;
    }
}

void pop()
{
    if (top <= -1)
        printf("\n\t Stack is under flow");
    else
    {
        printf("\n\t The popped elements is %d", stack[top]);
        top--;
    }
}

void display()
{

```

```

if (top >= 0)
{
    printf("\n The elements in STACK \n");
    for (i = top; i >= 0; i--)
        printf("\n%d", stack[i]);
    printf("\n Press Next Choice");
}
else
    printf("\n The STACK is empty");
}

```

7.4 Output

```

Enter the size of STACK[MAX=100]: 4

      STACK OPERATIONS USING ARRAY
-----
      1.PUSH
      2.POP
      3.DISPLAY
      4.EXIT
Enter the Choice:1
Enter a value to be pushed:2

Enter the Choice:2

      The popped elements is 2
Enter the Choice:3

The STACK is empty
Enter the Choice:4

      EXIT POINT

```

8 Linked List Operations

8.1 Objective

Write a program for insertion, deletion and traversal in linked list.

8.2 Algorithm

8.2.1 Traversal

```
1 Step 1: [INITIALIZE] SET PTR = HEAD
2 Step 2: Repeat Steps 3 and 4 while PTR != NULL
3 Step 3: Apply process to PTR -> DATA
4 Step 4: SET PTR = PTR->NEXT
5 [END OF LOOP]
6 Step 5: EXIT
```

8.2.2 Insertion

```
1 (AT BEGINNING)
2   Step 1: IF AVAIL = NULL
3   Write OVERFLOW
4   Go to Step 7
5   [END OF IF]
6   Step 2: SET NEW_NODE = AVAIL
7   Step 3: SET AVAIL = AVAIL -> NEXT
8   Step 4: SET NEW_NODE -> DATA = VAL
9   Step 5: SET NEW_NODE -> NEXT = HEAD
10  Step 6: SET HEAD = NEW_NODE
11  Step 7: EXIT
12
13 (AT END)
14   Step 1: IF AVAIL = NULL
15   Write OVERFLOW
16   Go to Step 10
17   [END OF IF]
18   Step 2: SET NEW_NODE = AVAIL
19   Step 3: SET AVAIL = AVAIL -> NEXT
20   Step 4: SET NEW_NODE -> DATA = VAL
21   Step 5: SET NEW_NODE -> NEXT = NULL
22   Step 6: SET PTR = HEAD
23   Step 7: Repeat Step 8 while PTR -> NEXT != NULL
24   Step 8: SET PTR = PTR -> NEXT
25   [END OF LOOP]
26   Step 9: SET PTR -> NEXT = NEW_NODE
27   Step 10: EXIT
28
29 (AFTER A GIVEN NODE)
30   Step 1: IF AVAIL = NULL
31   Write OVERFLOW
```

```

32   Go to Step 12
33   [END OF IF]
34   Step 2: SET NEW_NODE = AVAIL
35   Step 3: SET AVAIL = AVAIL -> NEXT
36   Step 4: SET NEW_NODE -> DATA = VAL
37   Step 5: SET PTR = HEAD
38   Step 6: SET PREPTR = PTR
39   Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA != NUM
40   Step 8: SET PREPTR = PTR
41   Step 9: SET PTR = PTR -> NEXT
42   [END OF LOOP]
43   Step 1 : PREPTR -> NEXT = NEW_NODE
44   Step 11: SET NEW_NODE -> NEXT = PTR
45   Step 12: EXIT

```

8.2.3 Deletion

```

1  (AT BEGINNING)
2   Step 1: IF HEAD = NULL
3   Write UNDERFLOW
4   Go to Step 5
5   [END OF IF]
6   Step 2: SET PTR = HEAD
7   Step 3: SET HEAD = HEAD -> NEXT
8   Step 4: FREE PTR
9   Step 5: EXIT
10  (AT END)
11  Step 1: IF HEAD = NULL
12  Write UNDERFLOW
13  Go to Step 8
14  [END OF IF]
15  Step 2: SET PTR = HEAD
16  Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
17  Step 4: SET PREPTR = PTR
18  Step 5: SET PTR = PTR -> NEXT
19  [END OF LOOP]
20  Step 6: SET PREPTR -> NEXT = NULL
21  Step 7: FREE PTR
22  Step 8: EXIT
23
24  (AFTER A NODE)
25  Step 1: IF HEAD = NULL
26  Write UNDERFLOW
27  Go to Step 10
28  [END OF IF]
29  Step 2: SET PTR = HEAD
30  Step 3: SET PREPTR = PTR
31  Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
32  Step 5: SET PREPTR = PTR
33  Step 6: SET PTR = PTR -> NEXT
34  [END OF LOOP]
35  Step 7: SET TEMP = PTR

```



```

36 Step 8: SET PREPTR -> NEXT = PTR -> NEXT
37 Step 9: FREE TEMP
38 Step 10 : EXIT

```

8.3 Code

```

#include <stdio.h>
#include <stdlib.h>
// Create a node
struct Node
{
    int data;
    struct Node *next;
};
// Insert at the beginning
void insertAtBeginning(struct Node **head_ref, int new_data)
{
    // Allocate memory to a node
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    // insert the data
    new_node->data = new_data;
    new_node->next = (*head_ref);
    // Move head to new node
    (*head_ref) = new_node;
}
// Insert a node after a node
void insertAfter(struct Node *prev_node, int new_data)
{
    if (prev_node == NULL)
    {
        printf("the given previous node cannot be NULL");
        return;
    }
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = prev_node->next;
    prev_node->next = new_node;
}
// Insert the the end
void insertAtEnd(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    struct Node *last = *head_ref; /* used in step 5*/
    new_node->data = new_data;

```

```

new_node->next = NULL;
if (*head_ref == NULL)
{
    *head_ref = new_node;
    return;
}
while (last->next != NULL)
    last = last->next;
last->next = new_node;
return;
}
// Delete a node
void deleteNode(struct Node **head_ref, int key)
{
    struct Node *temp = *head_ref, *prev;
    if (temp != NULL && temp->data == key)
    {
        *head_ref = temp->next;
        free(temp);
        return;
    }
    // Find the key to be deleted
    while (temp != NULL && temp->data != key)
    {
        prev = temp;
        temp = temp->next;
    }
    // If the key is not present
    if (temp == NULL)
        return;
    // Remove the node
    prev->next = temp->next;
    free(temp);
}
// Search a node
int searchNode(struct Node **head_ref, int key)
{
    struct Node *current = *head_ref;
    while (current != NULL)
    {
        if (current->data == key)
            return 1;
        current = current->next;
    }
}

```

```

    }
    return 0;
}
// Sort the linked list
void sortLinkedList(struct Node **head_ref)
{
    struct Node *current = *head_ref, *index = NULL;
    int temp;
    if (head_ref == NULL)
        return;
    else
        while (current != NULL)
        {
            // index points to the node next to current
            index = current->next;
            while (index != NULL)
                if (current->data > index->data)
                {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
            index = index->next;
            current = current->next;
        }
}
// Print the linked list
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}

int main(int argc, char const *argv[])
{
    struct Node linkedList;
    insertAtBeginning(&linkedList, 30);
    insertAtEnd(&linkedList, 20);
    insertAtEnd(&linkedList, 10);
    printList(&linkedList);
}

```

```
    deleteNode(&linkedList, 20);  
    printList(&linkedList);  
  
    sortLinkedList(&linkedList);  
    return 0;  
}
```

8.4 Output

9 Inorder Traversal

9.1 Objective

Write a program for Inorder Traversal of Binary Search Tree.

9.2 Algorithm

```
1 Traverse the left sub-tree in in-order
2 Visit the root
3 Traverse the right sub-tree in in-order
4 Repeat Steps 2 to 4 while TREE != NULL
5 INORDER(TREE -> LEFT)
6 Write TREE -> DATA
7 INORDER(TREE->RIGHT)
8 [END OF LOOP]
9 END
```

9.3 Code

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

typedef struct BST
{
    int data;
    struct BST *leftChild, *rightChild;
} node;

void inorder(node *root)
{
    if (root != NULL)
    {
        inorder(root->leftChild);
        printf("%d ", root->data);
        inorder(root->rightChild);
    }
}

node *newNode(int data)
{
    node *temp;
    temp = (node *)malloc(sizeof(node));
    if (temp == NULL)
    {
```

```

        fprintf(stderr, "Memory failure \n ");
        exit(1);
    }
    temp->data = data;
    temp->leftChild = NULL;
    temp->rightChild = NULL;
    return temp;
}

node *insert(node *root, int data)
{
    if (root == NULL)
        root = newNode(data);
    else
    {
        if (data < root->data)
            root->leftChild = insert(root->leftChild, data);
        else
            root->rightChild = insert(root->rightChild, data);
    }

    return root;
}

int main()
{
    struct node *bst = NULL;
    bst = insert(bst, 70);
    insert(bst, 10);
    insert(bst, 90);
    insert(bst, 40);
    insert(bst, 30);
    insert(bst, 60);
    inorder(bst);

    return 0;
}

```

9.4 Output

```
10 30 40 60 70 90
```

10 Depth First Search

10.1 Objective

Write a program for Depth first search.

10.2 Algorithm

```
1 For the DFS implementation we'll put each vertex of the graph into one of
  two categories:
2   1. Not Visited
3   2. Visited
4 Now we'll mark each vertex as visited while avoiding cycles.
5 The algorithm will go as follows:
6   Start by putting any one of the graph's vertices on top of a stack.
7   Take the top item of the stack and add it to the visited list.
8   Create a list of that vertex's adjacent nodes. Add the ones which aren
  't in the visited list to the top of the stack.
9   Keep repeating steps 2 and 3 until the stack is empty.
```

10.3 Code

```
// DFS algorithm in C
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int vertex;
    struct node *next;
};
struct node *createNode(int v);
struct Graph
{
    int numVertices;
    int *visited;
    // We need int** to store a two dimensional array.
    // Similarly, we need struct node** to store an array of Linked lists
    struct node **adjLists;
};
// DFS algo
void DFS(struct Graph *graph, int vertex)
{
    struct node *adjList = graph->adjLists[vertex];
    struct node *temp = adjList;
    graph->visited[vertex] = 1;
```

```

printf("Visited %d \n", vertex);
while (temp != NULL)
{
    int connectedVertex = temp->vertex;
    if (graph->visited[connectedVertex] == 0)
        DFS(graph, connectedVertex);
    temp = temp->next;
}
}

// Create a node
struct node *createNode(int v)
{
    struct node *newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Create graph
struct Graph *createGraph(int vertices)
{
    struct Graph *graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct node *));
    graph->visited = malloc(vertices * sizeof(int));
    int i;
    for (i = 0; i < vertices; i++)
    {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Add edge
void addEdge(struct Graph *graph, int src, int dest)
{
    // Add edge from src to dest
    struct node *newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

```



```

// Print the graph
void printGraph(struct Graph *graph)
{
    int v;
    for (v = 0; v < graph->numVertices; v++)
    {
        struct node *temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp)
        {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main()
{
    struct Graph *graph = createGraph(4);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
    printGraph(graph);
    DFS(graph, 2);
    return 0;
}

```

10.4 Output

```

Adjacency list of vertex 0
2 -> 1 ->

Adjacency list of vertex 1
2 -> 0 ->

Adjacency list of vertex 2
3 -> 1 -> 0 ->

Adjacency list of vertex 3
2 ->

Visited 2

```

```
Visited 3  
Visited 1  
Visited 0
```

11 Breath First Search

11.1 Objective

Write a program for Breadth first search.

11.2 Algorithm

11.3 Code

11.4 Output