# BEHARA COLLEGE OF ENGINEERING & TECHNOLOGY

**88th ward, G.V.M.C, NARAVA, VISAKHAPATNAM**

# DATA STRCTURES LABORATORY MANUAL

**LAB CODE : R23PC04** **SCHEME:R23**

**(Common to All branches of Engineering)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**88th ward, G.V.M.C, NARAVA, VISAKHAPATNAM**

## VISION

To Excel in the emerging areas of Computer Science and Engineering by imparting quality education , relevant practices and inculcating human values to transform the students as potential resources to contribute innovatively to meet industrial needs and social expectations.

## MISSION

M1: To provide strong fundamentals and value - based technical education for Computer Science applications through effective teaching learning methodologies.

M2: To transform lives of the students by nurturing ethical values, creativity and novelty to become Entrepreneurs and establish start-ups.

M3: To Impart high quality experiential learning to get expertise in modern software tools and to cater to the real time requirements of the industry

M4: To provide a conducive environment for faculty to engage in and train students in progressive and convergent research themes through collaborative linkages with industry and academia by establishing Centres of Excellence.

M5: To inculcate problem solving and team building skills and promote lifelong learning with a sense of societal and ethical responsibilities.

## PROGRAM OUTCOMES (POs)

**Engineering Graduates will be able to:**

**PO1.ENGINEERING KNOWLEDGE:** Apply the knowledge of mathematics, science, engineeringfundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2.PROBLEM ANALYSIS:** Identify, formulate, review research literature, and analyze complexengineering problems reaching substantiated conclusions using first principles of mathematics,natural sciences, and engineering sciences.

**PO3.DESIGN/DEVELOPMENT OF SOLUTIONS:** Design solutions for complex engineering problems anddesign system components or processes that meet the specified needs with appropriateconsideration for the public health and safety, and the cultural, societal, and environmentalconsiderations.

**PO4.CONDUCT INVESTIGATIONS OF COMPLEX PROBLEMS:** Use research-based knowledge and researchmethods including design of experiments, analysis and interpretation of data, and synthesis of theinformation to provide valid conclusions.

**PO5.MODERN TOOL USAGE:** Create, select, and apply appropriate techniques, resources, and modernengineering and IT tools including prediction and modelling to complex engineering activitieswith an understanding of the limitations.

**PO6. THE ENGINEER AND SOCIETY:** Apply reasoning informed by the contextual knowledge to assesssocietal, health, safety, legal and cultural issues and the consequent responsibilities relevant tothe professional engineering practice.

**PO7.ENVIRONMENT AND SUSTAINABILITY:** Understand the impact of the professional engineeringsolutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.

**PO8.ETHICS:** Apply ethical principles and commit to professional ethics and responsibilities andnorms of the engineering practice.

**PO9.INDIVIDUAL AND TEAM WORK:** Function effectively as an individual, and as a member or leaderin diverse teams, and in multidisciplinary settings.

**PO10.COMMUNICATION:** Communicate effectively on complex engineering activities with theengineering community and with society at large, such as, being able to comprehend and writeeffective reports and design documentation, make effective presentations, and give and receiveclear instructions.

**PO11.PROJECT MANAGEMENT AND FINANCE:** Demonstrate knowledge and understanding of theengineering and management principles and apply these to one's own work, as a member andleader in a team, to manage projects and in multidisciplinary environments.

**PO12.LIFE-LONG LEARNING:** Recognize the need for, and have the preparation and ability to engage inindependent and life-long learning in the broadest context of technological change.

## Program Specific Outcomes (PSOs)

The Computer Science & Engineering graduate will

| | | |
|---|---|---|
| **PSO-1** | **Professional Skills:** | The Computer Engineering Graduates are able to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics and networking for efficient design of computer based systems of varying complexity. |
| **PSO-2** | **Successful Career and Entrepreneurship:** | The Computer Engineering Graduates are able to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur and enthusiastic for higher studies/employability in the field of Computer Science & Engineering. |

## Program Educational Objectives (PEOs)

The Programme Educational Objectives of the B.Tech in Computer Science & Engineering programme are given below and are numbered from PEO1 to PEO4.

| | |
|---|---|
| **PEO-1** | To provide the graduates with solid foundation in computer science and engineering along with fundamentals of Mathematics and Sciences in formulating, analyzing, designing, modelling, programming and implementation with global competence and helps the graduates for life-long learning. |
| **PEO-2** | To Promote collaborative learning and team work spirit through multi -disciplinary projects and diverse professional activities and prepare graduates with recent technological developments related to core subjects like programming, databases, design of compilers and Network Security aspects and future technologies so as to contribute effectively for Research & Development by participating in professional activities like publishing and seeking copy rights. |
| **PEO-3** | To train graduates to choose a decent career option either in high degree of employability /Entrepreneur or, in higher education by empowering students with sustainable progress, ability to handle critical situations and training to excel in competitive examinations. |

| | |
|---|---|
| **PEO-4** | To exhibit professionalism, ethical attitude, communication, managerial skills, teamwork, and social responsibility in their profession and adapt to current trends by engaging in continuous learning. |

# GENERAL INSTRUCTIONS

## Do's

1. Please leave footwear outside the laboratory at the designated place.
2. Please keep your belongings such as bags in the designated place.
3. Students must present a valid ID card before entering the computer lab.
4. Maintain Discipline in the laboratory.
5. Work on the designated computers only.
6. Do keep your computer and workspace clean and organized.
7. Do save your work frequently to avoid data loss.
8. Do follow ethical guidelines and respect copyright when using digital resources.
9. Students must inform the in-charge Lecturer of any observed hardware or software failures.
10. Turn off the respective systems and arrange the chairs before you leaving the laboratory.

## Don'ts

1. Don't touch the computer with a pen or pencil.
2. Don't eat or drink near your computer.
3. Don't touch wires or cables when the computer is on.
4. Do not install, uninstall or alter any software on the computer.
5. Students are not allowed to use personal Pen Drives, CDs, DVDs etc., in a Computer Lab. Only prescribed official Pen Drives, CDs, DVDs etc. will be used in the Computer Lab to avoid VIRUS in Computers.
6. The use of cell phones is prohibited in the computer lab.
7. Don't forget to turn off your computer before leaving.

## IT WORKSHOP OBJECTIVES AND OUTCOMES
## (Common to all branches of Engineering)

**Course Objectives:**

- To introduce the internal parts of a computer, peripherals, I/O ports, connecting cables
- To demonstrate configuring the system as Dual boot both Windows and other Operating Systems Viz. Linux, BOSS
- To teach basic command line interface commands on Linux.
- To teach the usage of Internet for productivity and self-paced life-long learning
- To introduce Compression, Multimedia and Antivirus tools and Office Tools such as Word processors, Spread sheets and Presentation tools.

**Course Outcomes:**

CO1: Perform Hardware troubleshooting.
CO2: Understand Hardware components and inter dependencies.
CO3: Safeguard computer systems from viruses/worms.
CO4: Document/ Presentation preparation.
CO5: Perform calculations using spreadsheets.

# INDEX

## Exercise 1
## Array Manipulation

**i) Write a program to reverse an array.**

**AIM: C program to reverse an array**

**SOURCE CODE:**

```
#include <stdio.h>
 int main()
{
    int size;
    printf("Enter size of the array: ");
    scanf("%d",&size);
    printf("Enter Array Elements: ");
    int arr[size];
    for(int i=0;i<size;i++)
    scanf("%d",&arr[i]);
    printf("Entered Array is: ");
    for(int i=0;i<size;i++)
    printf("%d ",arr[i]);
    int start=0,end=size-1;
    while(start<end)
    {
        int temp=arr[start];
        arr[start]=arr[end];
        arr[end]=temp;
        start++;
        end--;
    }
    printf("\nReversed array is: ");
    for(int i=0;i<size;i++)
    printf("%d ",arr[i]);
    return 0;
}
```

**OUTPUT:**

```
Enter size of the array: 5
Enter Array Elements: 34  78  12  55  90
Entered Array is: 34 78 12 55 90
Reversed array is: 90 55 12 78 34
```

**ii) C Programs to implement the Searching Techniques – Linear & Binary Search**

**a) AIM: C Programs to implement the Linear Searching Techniques**

**SOURCE CODE:**

```
#include<stdio.h>
int linearSearch(int arr[], int size, int key)
{
 int index;
 for (index = 0; index < size; index++)
 if (arr[index] == key) // comparing each element with the key element
 return index;
 return -1;
}
int main()
{
 int arr[20], size, key, i, index;
 printf("Number of elements in the list: ");
 scanf("%d", &size);
 printf("Enter elements of the list: ");
 for (i = 0; i < size; i++)
 scanf("%d", &arr[i]);
 printf("Enter the element to search ie. key element: ");
 scanf("%d", &key);
 index = linearSearch(arr, size, key); // calling the function
 if (index == -1) // condition to check whether key found or not
 printf("Key element not found");
 else
 printf("Key element found at index %d", index); // printing the index if key found
 return 0;
}
```

**OUTPUT:**

```
Number of elements in the list: 5
Enter elements of the list: 45 67 12 3 56
Enter the element to search ie. key element: 56
Key element found at index 4
```

**b) AIM: C Programs to implement the Binary  Searching Techniques**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[20], i, n, key, low, high, mid;
 printf("Number of elements in the list: ");
 scanf("%d", &n);
printf("Enter the array elements in ascending order");
for(i = 0; i < n; i++)
{
 scanf("%d", &a[i]);
}
printf("Enter the key element\n");
scanf("%d", &key);
low = 0;   high = n - 1;
while(high >= low)
{
 mid = (low + high) / 2;
 if(key == a[mid])
 break;
 else
 {
 if(key > a[mid])
 low = mid + 1;
 else
 high = mid - 1;
 }
}
if(key == a[mid])
 printf("The key element is found at location %d", mid + 1);
else
 printf("the key element is not found");
getch();
}
```

**OUTPUT:**

```
Number of elements in the list: 5
Enter the array elements in ascending order: 23   24   35   67   89
Enter the key element   67     The key element is found at location 4
```

**iii) C Programs to implement Sorting Techniques – Bubble, Selection and Insertion Sort**

**a) AIM: C Programs to implement Sorting Techniques – Bubble Sort**

**SOURCE CODE:**

```
#include<stdio.h>
#include<stdlib.h>
void bubblesort (int a[], int size);
void main ()
{
 int a[50], n, i;
 printf ("\n Enter the size of the array");
 scanf ("%d", &n);
 printf ("\n Enter the array elements: \n");
 for (i = 0; i < n; i++)
 scanf ("%d", &a[i]);
 bubblesort (a, n);
 printf ("\n The sorted array is\n");
 for (i = 0; i < n; i++)
 printf ("%d\t", a[i]);
}
void bubblesort (int a[], int size)
{
 int temp, i, j,k;
 for (i = 0; i < size; i++)
{
 for (j = 0; j < size - 1; j++)
{
 if (a[j] > a[j + 1])
  {
 temp = a[j];
 a[j] = a[j + 1];
 a[j + 1] = temp;

  }
}
printf("\nAfter pass %d : ", i);
 for (k = 0; k < size; k++)
  {
 printf ("%d\t", a[k]);
  }
  }
}
```

**OUTPUT:**

```
Enter the size of the array5
Enter the array elements:
344
12
56
33
78
After pass 0 : 12 56 33 78 344
After pass 1 : 12 33 56 78 344
After pass 2 : 12 33 56 78 344
After pass 3 : 12 33 56 78 344
After pass 4 : 12 33 56 78 344
The sorted array is
12 33 56 78 344
```

**b) AIM: C Programs to implement Sorting Techniques – Selection Sort**

**SOURCE CODE:**

```
#include <stdio.h>
void selection_sort();
int a[30], n;
void main()
{
 int i;
 printf("\nEnter size of an array: ");
 scanf("%d", &n);
 printf("\nEnter elements of an array:\n");
 for(i=0; i<n; i++)
 scanf("%d", &a[i]);
 selection_sort();
 printf("\n\nAfter sorting:\n");
 for(i=0; i<n; i++)
 printf("\n%d", a[i]);
}
void selection_sort()
{
 int i, j, min, temp;
 for (i=0; i<n; i++)
  {
```

```
 min = i;
 for (j=i+1; j<n; j++)
 {
 if (a[j] < a[min])
 min = j;
 }
 temp = a[i];
 a[i] = a[min];
 a[min] = temp;

 printf("\nAfter pass %d : ", i);
 for (int k = 0; k < n; k++)
 {
 printf ("%d\t", a[k]);
 }
 }
}
```

**OUTPUT:**

```
Enter size of an array: 5
Enter elements of an array:
45
2
3
47
8
After pass 0 : 2 45 3 47 8
After pass 1 : 2 3 45 47 8
After pass 2 : 2 3 8 47 45
After pass 3 : 2 3 8 45 47
After pass 4 : 2 3 8 45 47
After sorting:
2
3
8
45
47
```

**c) AIM: C Programs to implement Sorting Techniques – Insertion Sort**

**SOURCE CODE:**

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

```
void InsertionSort (int a[], int n)
{
 int j, p, tmp;
for (p = 1; p < n; p++)
 {
 tmp = a[p];
 for (j = p; j > 0 && a[j - 1] > tmp; j--)
a[j] = a[j - 1];
 a[j] = tmp;
 printf("\nAfter pass %d : ", p);
 for (int k = 0; k < n; k++)
 {
 printf ("%d\t", a[k]);
 }
 }
}
int main ()
{
 int i, n, a[10];
 printf ("Enter the number of elements :: ");
 scanf ("%d", &n);
 printf ("Enter the elements :: ");
 for (i = 0; i < n; i++)
 {
 scanf ("%d", &a[i]);
 }
 InsertionSort (a, n);
 printf ("The sorted elements are :: ");
 for (i = 0; i < n; i++)
 printf ("%d ", a[i]);
 printf ("\n");
 return 0;
}
```

**OUTPUT:**

```
Enter the number of elements :: 5
Enter the elements :: 45  1  67   89  33
After pass 1 : 1 45 67 89 33
After pass 2 : 1 45 67 89 33
After pass 3 : 1 45 67 89 33
After pass 4 : 1 33 45 67 89
The sorted elements are :: 1 33 45 67 89
```

## Exercise 2
## Linked List Implementation

**i) Implement a singly linked list and perform insertion and deletion operations.**

**AIM: C program to Implement a singly linked list**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void insertAtBeginning(int);
void insertAtEnd(int);
void insertBetween(int,int,int);
void display();
void removeBeginning();
void removeEnd();
void removeSpecific(int);
struct Node
{
 int data;
 struct Node *next;
}*head = NULL;
void main()
{
 int choice,value,choice1,loc1,loc2;
 while(1)
{
 printf("\n\n****** MENU ******\n1. Insert\n2. Delete\n3. Display 4.Exit \nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
while(1)
{
 printf("Where you want to insert: \n1. At Beginning\n2. At End\n3.  Between\nEnter your choice: 
");
 scanf("%d",&choice1);
switch(choice1)
{
 case 1: insertAtBeginning(value);
break;
```

```
 case 2: insertAtEnd(value);
break;
 case 3: printf("Enter the two values where you wanto insert: ");
scanf("%d%d",&loc1,&loc2);
insertBetween(value,loc1,loc2);
break;
 default: printf("\nWrong Input!! Try again!!!\n\n");
goto mainMenu;
}
goto subMenuEnd;
}
subMenuEnd:
break;

 case 2: printf("How do you want to Delete: \n1. From Beginning\n2. From End\n3.
 Spesific\nEnter your choice: ");
scanf("%d",&choice1);
switch(choice1)
{
 case 1: removeBeginning();
break;
 case 2: removeEnd();
break;
 case 3: printf("Enter the value which you wanto delete: ");
scanf("%d",&loc2);
removeSpecific(loc2);
break;
 default: printf("\nWrong Input!! Try again!!!\n\n");
goto mainMenu;
}
break;
case 3: display();
break;
case 4: exit(0);
 default: printf("\nWrong input!!! Try again!!\n\n");
 }
 }
}
void insertAtBeginning(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 if(head == NULL)
 {
 newNode->next = NULL;
 head = newNode;
 }
 else
 {
 newNode->next = head;
 head = newNode;
 }
 printf("\nOne node inserted!!!\n");
}
void insertAtEnd(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 newNode->next = NULL;
 if(head == NULL)
head = newNode;
 else
 {
 struct Node *temp = head;
 while(temp->next != NULL)
temp = temp->next;
 temp->next = newNode;
 }
 printf("\nOne node inserted!!!\n");
}
void insertBetween(int value, int loc1, int loc2)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 if(head == NULL)
 {
 newNode->next = NULL;
 head = newNode;
 }
 else
 {
 struct Node *temp = head;
 while(temp->data != loc1 && temp->data != loc2)
temp = temp->next;
 newNode->next = temp->next;
 temp->next = newNode;
 }
 printf("\nOne node inserted!!!\n");
}
void removeBeginning()
{
 if(head == NULL)
printf("\n\nList is Empty!!!");
 else
 {
 struct Node *temp = head;
 if(head->next == NULL)
 {
head = NULL;
free(temp);
 }
 else
 {
head = temp->next;
free(temp);
printf("\nOne node deleted!!!\n\n");
 }
 }
}
void removeEnd()
{
 if(head == NULL)
 {
 printf("\nList is Empty!!!\n");
 }
 else
 {
 struct Node *temp1 = head,*temp2;
 if(head->next == NULL)
head = NULL;
 else
 {
while(temp1->next != NULL)
{
 temp2 = temp1;
 temp1 = temp1->next;
}
temp2->next = NULL;
 }
 free(temp1);
 printf("\nOne node deleted!!!\n\n");
 }
}
void removeSpecific(int delValue)
{
 struct Node *temp1 = head, *temp2;
 while(temp1->data != delValue)
 {
 if(temp1 -> next == NULL){
printf("\nGiven node not found in the list!!!");
goto functionEnd;
 }
 temp2 = temp1;
 temp1 = temp1 -> next;
 }
 temp2 -> next = temp1 -> next;
 free(temp1);
 printf("\nOne node deleted!!!\n\n");
 functionEnd:
}
void display()
{
 if(head == NULL)
 {
 printf("\nList is Empty\n");
 }
 else
 {
 struct Node *temp = head;
 printf("\n\nList elements are - \n");
 while(temp->next != NULL)
 {
printf("%d --->",temp->data);
temp = temp->next;
 }
 printf("%d --->NULL",temp->data);
 }
```

```
}
```

**OUTPUT:**

```
****** MENU ******
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 1
Enter the value to be insert: 12
Where you want to insert:
1. At Beginning
2. At End
3. Between
Enter your choice: 1
One node inserted!!!
```

**ii) Develop a program to reverse a linked list iteratively and recursively.**

**a) AIM:C program to reverse a linked list recursively**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data; // data field
   struct node* next;
};
void display(struct node* head)
{
   struct node* current = head; // current node set to head

   printf("traversing the list...\n");
   while (current != NULL) { //traverse until current node isn't NULL
      printf("%d ", current->data);
      current = current->next; // go to next node
   }
}
void reverse_display(struct node* head)
{
   if (head) {
      //recursive call to display in reverse order
      reverse_display(head->next);
      printf("%d ", head->data);
   }
}
struct node* creatnode(int d)
{
   struct node* temp = malloc(sizeof(struct node));
   temp->data = d;
   temp->next = NULL;
   return temp;
}
int main()
{
   printf("creating the linked list by inserting new nodes at the end\n");
   printf("enter 0 to stop building the list, else enter any integer\n");
   int k, count = 1, x;
```

```
    struct node *curr, *temp;
    scanf("%d", &k);
    struct node* head = creatnode(k); //buliding list, first node
    scanf("%d", &k);
    temp = head;

    //////////////////inserting at the end//////////////////////
    while (k) {
        curr = creatnode(k);
        temp->next = curr; //appending each node
        temp = temp->next;
        scanf("%d", &k);
    }
    display(head); // displaying the list
    printf("\ndisplaying in reverse order...\n");
    reverse_display(head); //display in reverse order
    return 0;
}
```

**OUTPUT:**

```
creating the linked list by inserting new nodes at the end
enter 0 to stop building the list, else enter any integer
45 67 12 89 43 0
traversing the list...
45 67 12 89 43
displaying in reverse order...
43 89 12 67 45
```

**b) AIM: C program to reverse a linked list iteratively**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data; //Data part
    struct node *next; //Address part
}*head;
void createList(int n);
void reverseList();
void displayList();
```

```
int main()
{
   int n, choice;
   printf("Enter the total number of nodes: ");
   scanf("%d", &n);
   createList(n);

   printf("\nData in the list \n");
   displayList();
   printf("\nPress 1 to reverse the order of singly linked list\n");
   scanf("%d", &choice);
   if(choice == 1)
   {
      reverseList();
   }
   printf("\nData in the list\n");
   displayList();
   return 0;
}
void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   if(n <= 0)
   {
      printf("List size must be greater than zero.\n");
      return;
   }
   head = (struct node *)malloc(sizeof(struct node));
   if(head == NULL)
   {
      printf("Unable to allocate memory.");
   }
   else
   {
       printf("Enter the data of node 1: ");
      scanf("%d", &data);

      head->data = data; // Link the data field with data
      head->next = NULL; // Link the address field to NULL
      temp = head;
```

```
        for(i=2; i<=n; i++)
        {
            newNode = (struct node *)malloc(sizeof(struct node));
            if(newNode == NULL)
            {
                printf("Unable to allocate memory.");
                break;
            }
            else
            {
                printf("Enter the data of node %d: ", i);
                scanf("%d", &data);

                newNode->data = data; // Link the data field of newNode with data
                newNode->next = NULL; // Link the address field of newNode with NULL

                temp->next = newNode; // Link previous node i.e. temp to the newNode
                temp = temp->next;
            }
        }
        printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");
    }
}
void reverseList()
{
    struct node *prevNode, *curNode;
    if(head != NULL)
    {
        prevNode = head;
        curNode = head->next;
        head = head->next;
        prevNode->next = NULL; // Make first node as last node
        while(head != NULL)
        {
            head = head->next;
            curNode->next = prevNode;

            prevNode = curNode;
            curNode = head;
        }
        head = prevNode; // Make last node as head
        printf("SUCCESSFULLY REVERSED LIST\n");
    }
```

```
}
void displayList()
{
    struct node *temp;

     if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        while(temp != NULL)
        {
            printf("Data = %d\n", temp->data); // Print the data of current node
            temp = temp->next;                 // Move to next node
        }
    }
}
```

**OUTPUT:**

```
Enter the total number of nodes: 5
Enter the data of node 1: 45
Enter the data of node 2: 67
Enter the data of node 3: 12
Enter the data of node 4: 45
Enter the data of node 5: 33
SINGLY LINKED LIST CREATED SUCCESSFULLY
Data in the list
Data = 45
Data = 67
Data = 12
Data = 45
Data = 33
Press 1 to reverse the order of singly linked list    1
SUCCESSFULLY REVERSED LIST
Data in the list
Data = 33
Data = 45
Data = 12
Data = 67
Data = 45
```

**iii) Solve problems involving linked list traversal and manipulation.**

**AIM: C program to perform linked list traversal and manipulation**

**SOURCE CODE:**

```
#include<stdio.h>
#include<stdlib.h>
void create(int);
void traverse();
struct node
{
        int data;
        struct node *next;
};
struct node *head;
void main ()
{
        int choice,item;
        do
        {
                printf("\n1.Append List\n2.Traverse\n3.Exit\n4.Enter your choice?");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:  printf("\nEnter the item\n");
                                 scanf("%d",&item);
                                 create(item);
                                break;
                        case 2:   traverse();      break;
                        case 3:  exit(0);        break;
                        default:
                        printf("\nPlease enter valid choice\n");
                }

        }while(choice != 3);
}
void create(int item)
        {
                struct node *ptr = (struct node *)malloc(sizeof(struct node *));
                if(ptr == NULL)
                {
                        printf("\nOVERFLOW\n");
                }
                else
                {
                        ptr->data = item;
                        ptr->next = head;
                        head = ptr;
                        printf("\nNode inserted\n");
                }

        }
void traverse()
        {
                struct node *ptr;
                ptr = head;
                if(ptr == NULL)
                {
                        printf("Empty list..");
                }
                else
                {
                        printf("printing values . . . . .\n");
                        while (ptr!=NULL)
                        {
                                printf("\n%d",ptr->data);
                                ptr = ptr -> next;
                        }
                }
         }
```

**OUTPUT:**

```
1.Append List
2.Traverse
3.Exit
4.Enter your choice?1
Enter the item 34
Node inserted
1.Append List
2.Traverse
3.Exit
4.Enter your choice?2
printing values . . . . .
34
```

## Exercise 3
## Linked List Applications

**i) Create a program to detect and remove duplicates from a linked list.**

**AIM:C program to detect and remove duplicates from a linked list**

**SOURCE CODE:**

```
struct node
{
int data;
struct node* next;
};
void insert_elements(struct node** head, int new_data)
{
struct node* new_node = (struct node*) malloc(sizeof(struct node));
new_node -> data = new_data;
new_node -> next = (*head);
(*head) = new_node;
}
void display_list(struct node *node)
{
while (node!=NULL)
{
printf("%d ", node->data);
node = node -> next;
}
}
void remove_duplicate_elements(struct node* head)
{
struct node* current = head;

struct node* next_next;

if (current == NULL)
return;

while (current -> next != NULL)
{
if (current -> data == current -> next -> data)
{
next_next = current -> next -> next;
free(current -> next);
current -> next = next_next;
}
else
{
current = current -> next;
}
}
}
int main()
{
struct node* head = NULL;
int n;
printf("\nEnter the total number of elements : ");
scanf("%d", &n);
printf("\nEnter the sorted linked list : ");
int i;
for(i = 0; i < n; i++)
{
int data;
scanf("%d", &data);
insert_elements(&head, data);
}
printf("\nLinked list before removing duplicates : ");
display_list(head);
printf("\n");

remove_duplicate_elements(head);

printf("\nLinked list after removing duplicates : ");
display_list(head);
printf("\n");


return 0;
}
```

**OUTPUT:**

```
Enter the total number of elements : 10
Enter the sorted linked list : 1 2 3 3 4 4 5 5 5 6
Linked list before removing duplicates : 6 5 5 5 4 4 3 3 2
Linked list after removing duplicates : 6 5 4 3 2 1
```

**ii) Implement a linked list to represent polynomials and perform addition.**

**AIM: C program to Implement a linked list to represent polynomials and perform addition**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct Node
 {
   int coeff;
   int exp;
   struct Node * next;
}* poly = NULL;
void create()
{
   struct Node * t, * last = NULL;
   int num, i;
   printf("Enter number of terms: ");
   scanf("%d", & num);
   printf("Enter each term with coeff and exp:\n");
   for (i = 0; i < num; i++)
{
      t = (struct Node * ) malloc(sizeof(struct Node));
      scanf("%d%d", & t -> coeff, & t -> exp);
      t -> next = NULL;
      if (poly == NULL)
 {
         poly = last = t;
      } else {
         last -> next = t;
         last = t;
      }
   }
}
void Display(struct Node * p)
{
   printf("%dx%d ", p -> coeff, p -> exp);
      p = p -> next;

   while (p)
{
        printf("+ %dx%d ", p -> coeff, p -> exp);
        p = p -> next;
    }
    printf("\n");
}
long Eval(struct Node * p, int x)
{
    long val = 0;
    while (p) {
        val += p -> coeff * pow(x, p -> exp);
        p = p -> next;
    }

    return val;
}
int main()
{
    int x;
    create();
    Display(poly);

    printf("Enter value of x: ");
    scanf("%d", &x);

    printf("%ld\n", Eval(poly, x));
    return 0;
}
```

**OUTPUT:**

```
Enter number of terms: 3
Enter each term with coeff and exp:
4 2
3 2
4 1
4x2 + 3x2 + 4x1
Enter value of x: 2
36
```

**iii) Implement a double-ended queue (deque) with essential operations.**

**AIM:C program to Implement a double-ended queue (deque) with essential operations**

**SOURCE CODE:**

```
# include<stdio.h>
# define Size 5
 int deque_arr[Size];
int front = -1;
int rear = -1;
void insert_rear()
{
    int added_item;
    if((front == 0 && rear == Size-1) || (front == rear+1))
    {   printf("Queue Overflow\n");
        return;}
    if (front == -1)  /* if queue is initially empty */
    {   front = 0;
        rear = 0;}
    else
    if(rear == Size-1)  /*rear is at last position of queue */
        rear = 0;
    else
        rear = rear+1;
            printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[rear] = added_item ;
}
void insert_front()
{   int added_item;
    if((front == 0 && rear == Size-1) || (front == rear+1))
    {   printf("Queue Overflow \n");
        return;  }
    if (front == -1)/*If queue is initially empty*/
    {   front = 0;
        rear = 0;    }
    else
    if(front== 0)
        front=Size-1;
    else
        front=front-1;
    printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[front] = added_item ;  }
void delete_front()
{   if (front == -1)
    {   printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[front]);
    if(front == rear) /*Queue has only one element */
    {   front = -1;
        rear=-1;
    }
    else
        if(front == Size-1)
            front = 0;
        else
            front = front+1;
}
void delete_rear()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",deque_arr[rear]);
    if(front == rear) /*queue has only one element*/
    {
        front = -1;
        rear=-1;
    }
    else
        if(rear == 0)
            rear=Size-1;
        else
            rear=rear-1;    }
void display_queue()
{
    int front_pos = front,rear_pos = rear;

    if(front == -1)
    {   printf("Queue is empty\n");
        return;
    }
    printf("Queue elements :\n");
    if( front_pos <= rear_pos )
    {
        while(front_pos <= rear_pos)
        {
            printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
    }
    else
    {
        while(front_pos <= Size-1)
        {   printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
        front_pos = 0;
        while(front_pos <= rear_pos)
        {
            printf("%d ",deque_arr[front_pos]);
            front_pos++;
        }
    }
    printf("\n");
}
void input_que()
{   int choice;
    do
    {   printf("1.Insert at rear\n");
        printf("2.Delete from front\n");
        printf("3.Delete from rear\n");
        printf("4.Display\n");
        printf("5.Quit\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
         switch(choice)
        {   case 1:
            insert_rear();
            break;
         case 2:
            delete_front();
            break;
         case 3:
            delete_rear();
            break;
          case 4:
            display_queue();
            break;
          case 5:
            break;
          default:
            printf("Wrong choice\n");
          }
     }while(choice!=5);
}
void output_que()
{   int choice;
     do
     {   printf("1.Insert at rear\n");
         printf("2.Insert at front\n");
         printf("3.Delete from front\n");
         printf("4.Display\n");
         printf("5.Quit\n");
         printf("Enter your choice : ");
         scanf("%d",&choice);
         switch(choice)
         {
          case 1:
            insert_rear();
            break;
          case 2:
            insert_front();
            break;
          case 3:
            delete_front();
            break;
          case 4:
            display_queue();
            break;
          case 5:
            break;
          default:
            printf("Wrong choice\n");
          }
     }while(choice!=5);
}
```

```
void main()
{   int choice;
    printf("1.Input restricted dequeue\n");
    printf("2.Output restricted dequeue\n");
    printf("Enter your choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
     case 1 :
        input_que();
        break;
     case 2:
        output_que();
        break;
     default:
        printf("Wrong choice\n");
    }
}
```

**OUTPUT:**

```
1.Input restricted dequeue
2.Output restricted dequeue
Enter your choice : 1
1.Insert at rear
2.Delete from front
3.Delete from rear
4.Display
5.Quit
Enter your choice : 1
Input the element for adding in queue : 23
1.Insert at rear
2.Delete from front
3.Delete from rear
4.Display
5.Quit
Enter your choice : 1
Input the element for adding in queue : 56
1.Insert at rear
2.Delete from front
3.Delete from rear
4.Display
5.Quit
Enter your choice : 5
```

# Exercise 4
# Double Linked List Implementation

**i) Implement a doubly linked list and perform various operations to understand its properties and applications.**

**AIM: C program to Implement a doubly linked list**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void insertAtBeginning(int);
void insertAtEnd(int);
void insertAtAfter(int,int);
void deleteBeginning();
void deleteEnd();
void deleteSpecific(int);
void display();
struct Node
{
 int data;
 struct Node *previous, *next;
}*head = NULL;
void main()
{
 int choice1, choice2, value, location;
 while(1)
 {
 printf("\n*********** MENU *************\n");
 printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
 scanf("%d",&choice1);
 switch(choice1)
 {
 case 1: printf("Enter the value to be inserted: ");
 scanf("%d",&value);
 while(1)
 {
 printf("\nSelect from the following Inserting options\n");
 printf("1. At Beginning\n2. At End\n3. After a Node\n4. Cancel\nEnter your
 choice: ");
 scanf("%d",&choice2);
 switch(choice2)
 {
 case 1: insertAtBeginning(value);
 break;
 case 2: insertAtEnd(value);
 break;
 case 3: printf("Enter the location after which you want to insert: ");
 scanf("%d",&location);
 insertAtAfter(value,location);
 break;
 case 4: goto EndSwitch;
 default: printf("\nPlease select correct Inserting option!!!\n");
 }
 }
 case 2: while(1)
 {
 printf("\nSelect from the following Deleting options\n");
 printf("1. At Beginning\n2. At End\n3. Specific Node\n4. Cancel\nEnter your
 choice: ");
 scanf("%d",&choice2);
 switch(choice2)
 {
 case 1: deleteBeginning();
 break;
 case 2: deleteEnd();
 break;
 case 3: printf("Enter the Node value to be deleted: ");
 scanf("%d",&location);
 deleteSpecific(location);
 break;
 case 4: goto EndSwitch;
 default: printf("\nPlease select correct Deleting option!!!\n");
 }
 }
 EndSwitch: break;
 case 3: display(); break;
case 4: exit(0);
 default: printf("\nPlease select correct option!!!");
 }
 }
}
void insertAtBeginning(int value)
```

```
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 newNode -> previous = NULL;
 if(head == NULL)
 {
 newNode -> next = NULL;
 head = newNode;
 }
 else
 {
 newNode -> next = head;
 head = newNode;
 }
 printf("\nInsertion success!!!");
}
void insertAtEnd(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 newNode -> next = NULL;
 if(head == NULL)
 {
 newNode -> previous = NULL;
 head = newNode;
 }
 else
 {
 struct Node *temp = head;
 while(temp -> next != NULL)
 temp = temp -> next;
 temp -> next = newNode;
 newNode -> previous = temp;
 }
 printf("\nInsertion success!!!");
}
void insertAtAfter(int value, int location)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 if(head == NULL)
 {
 newNode -> previous = newNode -> next = NULL;
 head = newNode;
 }
 else
 {
 struct Node *temp1 = head, *temp2;
 while(temp1 -> data != location)
 {
 if(temp1 -> next == NULL)
 {
 printf("Given node is not found in the list!!!");
 goto EndFunction;
 }
 else
 {
 temp1 = temp1 -> next;
 }
 }
 temp2 = temp1 -> next;
 temp1 -> next = newNode;
 newNode -> previous = temp1;
 newNode -> next = temp2;
 temp2 -> previous = newNode;
 printf("\nInsertion success!!!");
 }
 EndFunction:
}
void deleteBeginning()
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp = head;
 if(temp -> previous == temp -> next)
 {
 head = NULL;
 free(temp);
 }
 else
 {
```

```
 head = temp -> next;
 head -> previous = NULL;
 free(temp);
 }
 printf("\nDeletion success!!!");
 }
}
void deleteEnd()
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp = head;
 if(temp -> previous == temp -> next)
 {
 head = NULL;
 free(temp);
 }
 else
 {
 while(temp -> next != NULL)
 temp = temp -> next;
 temp -> previous -> next = NULL;
 free(temp);
 }
 printf("\nDeletion success!!!");
 }
}
void deleteSpecific(int delValue)
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp = head;
 while(temp -> data != delValue)
 {
 if(temp -> next == NULL)
 {
 printf("\nGiven node is not found in the list!!!");
 goto FuctionEnd;
 }
 else
 {
 temp = temp -> next;
 }
 }
 if(temp == head)
 {
 head = NULL;
 free(temp);
 }
 else
 {
 temp -> previous -> next = temp -> next;
 free(temp);
 }
 printf("\nDeletion success!!!");
 }
 FuctionEnd:
}
void display()
{
 struct Node *current = head;
 if(head == NULL)
 {
 printf("List is empty\n");
 return;
 }
 while(current != NULL)
 {
 printf("%d ",current->data);
 current = current->next;
 }
 printf("\n");
}
```

**OUTPUT:**

```
*********** MENU *************
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 12
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 1
Insertion success!!!
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 4
```

**ii) Implement a circular linked list and perform insertion, deletion, and traversal.**

**AIM: C program to Implement a circular linked list**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void insertAtBeginning(int);
void insertAtEnd(int);
void insertAtAfter(int,int);
void deleteBeginning();
void deleteEnd();
void deleteSpecific(int);
void display();
 struct Node
{
 int data;
 struct Node *next;
}*head = NULL;
void main()
{
 int choice1, choice2, value, location;
 while(1)
 {
 printf("\n*********** MENU *************\n");
 printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");
 scanf("%d",&choice1);
 switch(choice1)
 {
 case 1: printf("Enter the value to be inserted: ");
 scanf("%d",&value);
 while(1)
 {
 printf("\nSelect from the following Inserting options\n");
 printf("1. At Beginning\n2. At End\n3. After a Node\n4. Cancel\nEnter your  choice: ");
 scanf("%d",&choice2);
 switch(choice2)
 {
 case 1: insertAtBeginning(value);   break;
 case 2: insertAtEnd(value);    break;
 case 3: printf("Enter the location after which you want to insert: ");
 scanf("%d",&location);
 insertAtAfter(value,location);
 break;
 case 4: goto EndSwitch;
 default: printf("\nPlease select correct Inserting option!!!\n");
 }
 }
 case 2: while(1)
 {
 printf("\nSelect from the following Deleting options\n");
 printf("1. At Beginning\n2. At End\n3. Specific Node\n4. Cancel\nEnter your
 choice: ");
 scanf("%d",&choice2);
 switch(choice2)
 {
 case 1: deleteBeginning();   break;
 case 2: deleteEnd();     break;
 case 3: printf("Enter the Node value to be deleted: ");
 scanf("%d",&location);
 deleteSpecific(location);
 break;
 case 4: goto EndSwitch;
 default: printf("\nPlease select correct Deleting option!!!\n");
 }
 }
 EndSwitch: break;
 case 3: display();  break;
case 4 : exit(0);
 default: printf("\nPlease select correct option!!!");
 }
 }
}
void insertAtBeginning(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 if(head == NULL)
 {
 head = newNode;
 newNode -> next = head;
 }
```

```
 else
 {
 struct Node *temp = head;
 while(temp -> next != head)
 temp = temp -> next;
 newNode -> next = head;
 head = newNode;
 temp -> next = head;
 }
 printf("\nInsertion success!!!");
}
void insertAtEnd(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 if(head == NULL)
 {
 head = newNode;
 newNode -> next = head;
 }
 else
 {
 struct Node *temp = head;
 while(temp -> next != head)
 temp = temp -> next;
 temp -> next = newNode;
 newNode -> next = head;
 }
 printf("\nInsertion success!!!");
}
void insertAtAfter(int value, int location)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode -> data = value;
 if(head == NULL)
 {
 head = newNode;
 newNode -> next = head;
 }
 else
 {
```

```
 struct Node *temp = head;
 while(temp -> data != location)
 {
 if(temp -> next == head)
 {
 printf("Given node is not found in the list!!!");
 goto EndFunction;
 }
 else
 {
 temp = temp -> next;
 }
 }
 newNode -> next = temp -> next;
 temp -> next = newNode;
 printf("\nInsertion success!!!");
 }
 EndFunction:
}
void deleteBeginning()
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp = head;
 if(temp -> next == head)
 {
 head = NULL;
 free(temp);
 }
 else{
 head = head -> next;
 free(temp);
 }
 printf("\nDeletion success!!!");
 }
}
void deleteEnd()
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp1 = head, *temp2;
 if(temp1 -> next == head)
 {
 head = NULL;
 free(temp1);
 }
 else{
 while(temp1 -> next != head){
 temp2 = temp1;
 temp1 = temp1 -> next;
 }
 temp2 -> next = head;
 free(temp1);
 }
 printf("\nDeletion success!!!");
 }
}
void deleteSpecific(int delValue)
{
 if(head == NULL)
 printf("List is Empty!!! Deletion not possible!!!");
 else
 {
 struct Node *temp1 = head, *temp2;
 while(temp1 -> data != delValue)
 {
 if(temp1 -> next == head)
 {
 printf("\nGiven node is not found in the list!!!");
 goto FunctionEnd;
 }
 else
 {
 temp2 = temp1;
 temp1 = temp1 -> next;
 }
 }
 if(temp1 -> next == head)
 {
 head = NULL;
 free(temp1);
 }
 else
 {
 if(temp1 == head)
 {
 temp2 = head;
 while(temp2 -> next != head)
 temp2 = temp2 -> next;
 head = head -> next;
 temp2 -> next = head;
 free(temp1);
 }
 else
 {
 if(temp1 -> next == head)
 {
 temp2 -> next = head;
 }
 else
 {
 temp2 -> next = temp1 -> next;
 }
 free(temp1);
 }
 }
 printf("\nDeletion success!!!");
 }
 FunctionEnd:
}
void display()
{
 struct Node *ptr;
 ptr=head;
 if(head == NULL)
 {
 printf("\nnothing to print");
 }
 else
 {
 printf("\n printing values ... \n");
 while(ptr -> next != head)
 {
 printf("%d\n", ptr -> data);
 ptr = ptr -> next;
 }
 printf("%d\n", ptr -> data);
 }
}
```

**OUTPUT:**

```
*********** MENU *************
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be inserted: 122
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 1
Insertion success!!!
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 4
*********** MENU *************
1. Insert
2. Delete
3. Display
4.Exit
Enter your choice: 1
Enter the value to be inserted: 23
Select from the following Inserting options
1. At Beginning
2. At End
3. After a Node
4. Cancel
Enter your choice: 2
Insertion success!!!
```

## Exercise 5
## Stack Operations

**i) Implement a stack using arrays and linked lists.**

**a) AIM: C program to Implement a stack using arrays**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 10
void push(int);
void pop();
void display();
int stack[SIZE], top = -1;
void main()
{
 int value, choice;
 while(1)
{
 printf("\n\n***** MENU *****\n");
 printf("1. Push\n2. Pop\n3. Display\n4. Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
push(value);
break;
case 2: pop();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
 }
 }
}
void push(int value)
{
 if(top == SIZE-1)
 printf("\nStack is Full!!! Insertion is not possible!!!");
 else
 {
 top++;
 stack[top] = value;
 printf("\nInsertion success!!!");
 }
}
void pop()
{
 if(top == -1)
 printf("\nStack is Empty!!! Deletion is not possible!!!");
 else
 {
 printf("\nDeleted : %d", stack[top]);
 top--;
 }
}
void display()
{
 if(top == -1)
 printf("\nStack is Empty!!!");
 else
 {
 int i;
 printf("\nStack elements are:\n");
 for(i=top; i>=0; i--)
printf("%d\n",stack[i]);
 }
}
```

**OUTPUT:**

```
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 12
Insertion success!!!
```

```
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 23
Insertion success!!!
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Deleted : 23
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements are:12
***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4
```

**b) AIM: C program to Implement a stack using Linked list**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct Node
{
 int data;
 struct Node *next;
}*top = NULL;
void push(int);
void pop();
void display();
void main()
{
 int choice, value;
 printf("\n:: Stack using Linked List ::\n");
 while(1)
 {
 printf("\n****** MENU ******\n");
 printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
case 1: printf("Enter the value to be insert: ");
scanf("%d", &value);
push(value);
break;
case 2: pop(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
 }
 }
}
void push(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 if(top == NULL)
 newNode->next = NULL;
 else
 newNode->next = top;
 top = newNode;
 printf("\nInsertion is Success!!!\n");
}
void pop()
{
 if(top == NULL)
 printf("\nStack is Empty!!!\n");
 else
 {
 struct Node *temp = top;
 printf("\nDeleted element: %d", temp->data);
 top = temp->next;
 free(temp);
 }
}
void display()
{
 if(top == NULL)
 printf("\nStack is Empty!!!\n");
 else
 {
 struct Node *temp = top;
 while(temp->next != NULL)
 {
printf("%d--->",temp->data);
temp = temp -> next;
 }
 printf("%d--->NULL",temp->data);
 }
}
```

**OUTPUT:**

```
:: Stack using Linked List ::
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 23
Insertion is Success!!!
****** MENU ******
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 45
Insertion is Success!!!
```

**ii) Write a program to evaluate a postfix expression using a stack.**

**AIM:C program to evaluate a postfix expression using a stack**

**SOURCE CODE:**

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define SIZE 40
int pop();
void push(int);
char postfix[SIZE];
int stack[SIZE], top = -1;
int main()
{
int i, a, b, result, pEval;
char ch;
for(i=0; i<SIZE; i++)
{
stack[i] = -1;
}
printf("\nEnter a postfix expression: ");
scanf("%s",postfix);
for(i=0; postfix[i] != '\0'; i++)
{
ch = postfix[i];
if(isdigit(ch))
{
push(ch-'0');
}
else if(ch == '+' || ch == '-' || ch == '*' || ch == '/')
{
b = pop();
a = pop();
switch(ch)
{
case '+': result = a+b; break;
case '-': result = a-b; break;
case '*': result = a*b; break;
case '/': result = a/b; break;
case '%':result = a%b; break;
}
push(result);
}
}
pEval = pop();
printf("\nThe postfix evaluation is: %d\n",pEval);
return 0;
}
void push(int n)
{
if (top < SIZE -1)
{
stack[++top] = n;
}
else
{
printf("Stack is full!\n");
exit(-1);
}
}
int pop()
{
int n;
if (top > -1)
{
n = stack[top];
stack[top--] = -1;
return n;
}
else
{
printf("Stack is empty!\n");
exit(-1);
}
}
```

**OUTPUT:**

```
Enter a postfix expression: 5 6 + - 5 2 +1 / The postfix evaluation is: 5
```

**iii) Implement a program to check for balanced parentheses using a stack.**

**AIM: Implementing a C program to check for balanced parentheses using a stack**

**SOURCE CODE:**

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

#define MAX_SIZE 100

char stack[MAX_SIZE];
int top = -1;

void push(char data)
 {
   if (top == MAX_SIZE - 1)
{
      printf("Overflow stack!\n");
      return;
   }
   top++;
   stack[top] = data;
}

char pop()
{
   if (top == -1)
{
      printf("Empty stack!\n");
      return ' ';
   }
   char data = stack[top];
   top--;
   return data;
}

int is_matching_pair(char char1, char char2)
{
   if (char1 == '(' && char2 == ')')
{
      return 1;
```

```
    } else if (char1 == '[' && char2 == ']')
{
        return 1;
    } else if (char1 == '{' && char2 == '}')
{
        return 1;
    } else {
        return 0;
    }
}

int isBalanced(char* text)
{
    int i;
    for (i = 0; i < strlen(text); i++)
{
        if (text[i] == '(' || text[i] == '[' || text[i] == '{')
{
            push(text[i]);
        } else if (text[i] == ')' || text[i] == ']' || text[i] == '}')
{
            if (top == -1)
{
                return 0; // If no opening bracket is present
            } else if (!is_matching_pair(pop(), text[i]))
{
                return 0; // If closing bracket doesn't match the last opening bracket
            }
        }
    }
    if (top == -1)
{
        return 1; // If the stack is empty, the expression is balanced
    }
else
{
        return 0; // If the stack is not empty, the expression is not balanced
    }
}

int main()
{
   char text[MAX_SIZE];
```

```
  printf("Input an expression in parentheses: ");
  scanf("%s", text);
  if (isBalanced(text))
{
      printf("The expression is balanced.\n");
  }
else
{
      printf("The expression is not balanced.\n");
  }
  return 0;
}
```

**OUTPUT:**

```
Input an expression in parentheses: ([]
The expression is not balanced.

Input an expression in parentheses: ([])
The expression is balanced.
```

## Exercise 6
## Queue Operations

**i) Implement a queue using arrays and linked lists.**

**a) AIM:C program to Implement a queue using arrays**

**SOURCE CODE:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 10
void enQueue(int);
void deQueue();
void display();
int queue[SIZE], front = -1, rear = -1;
void main()
{
 int value, choice;
 while(1)
 {
 printf("\n\n***** MENU *****\n");
 printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice)
 {
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
enQueue(value);
break;
case 2: deQueue();
break;
case 3: display();
break;
case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
 }
 }
}
void enQueue(int value)
{
 if(rear == SIZE-1)
 printf("\nQueue is Full!!! Insertion is not possible!!!");
 else
 {
 if(front == -1)
front = 0;
 rear++;
 queue[rear] = value;
 printf("\nInsertion success!!!");
 }
}
void deQueue()
{
 if(front == rear)
 printf("\nQueue is Empty!!! Deletion is not possible!!!");
 else
{
 printf("\nDeleted : %d", queue[front]);
 front++;
 if(front == rear)
front = rear = -1;
 }
}
void display()
{
 if(rear == -1)
 printf("\nQueue is Empty!!!");
 else
 {
 int i;
 printf("\nQueue elements are:\n");
 for(i=front; i<=rear; i++)
 printf("%d\t",queue[i]);
 }
}
```

**OUTPUT:**

```
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 23
Insertion success!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 56
Insertion success!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 2
Deleted : 23
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 3
Queue is Empty!!!
***** MENU *****
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your choice: 4
```

**b) AIM: C program to Implement a queue using linked lists**

**SOURCE CODE:**

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

```
}*front = NULL,*rear = NULL;
void insert(int);
void delete();
void display();
void main()
{
 int choice, value;
 printf("\n:: Queue Implementation using Linked List ::\n");
 while(1){
 printf("\n****** MENU ******\n");
 printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
 printf("Enter your choice: ");
 scanf("%d",&choice);
 switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d", &value);
insert(value);
break;
case 2: delete(); break;
case 3: display(); break;
case 4: exit(0);
default: printf("\nWrong selection!!! Please try again!!!\n");
 }
 }
}
void insert(int value)
{
 struct Node *newNode;
 newNode = (struct Node*)malloc(sizeof(struct Node));
 newNode->data = value;
 newNode -> next = NULL;
 if(front == NULL)
 front = rear = newNode;
 else{
 rear -> next = newNode;
 rear = newNode;
 }
 printf("\nInsertion is Success!!!\n");
}
void delete()
{
 if(front == NULL)
 printf("\nQueue is Empty!!!\n");
 else{
 struct Node *temp = front;
 front = front -> next;
 printf("\nDeleted element: %d\n", temp->data);
 free(temp);
 }
}
void display()
{
 if(front == NULL)
 printf("\nQueue is Empty!!!\n");
 else{
 struct Node *temp = front;
 while(temp->next != NULL){
printf("%d--->",temp->data);
temp = temp -> next;
 }
 printf("%d--->NULL\n",temp->data);
 }
}
```

**OUTPUT:**

```
:: Queue Implementation using Linked List ::
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 23
Insertion is Success!!!
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 45
Insertion is Success!!!
****** MENU ******
1. Insert
2. Delete
```

```
3. Display
4. Exit
Enter your choice: 2
Deleted element: 23
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
45--->NULL
****** MENU ******
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
```

**ii) Develop a program to simulate a simple printer queue system.**

**AIM: Developing a C program to simulate a simple printer queue system**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 10
typedef struct Node
{
   int data;
   struct Node* next;
} Node;
typedef struct
 {
   Node* front;
   Node* rear;
   int size;
} Queue;
void initQueue(Queue* queue)
{
   queue->front = NULL;
   queue->rear = NULL;
   queue->size = 0;
}
int isEmpty(Queue* queue)
{
   return queue->size == 0;
}
void enqueue(Queue* queue, int data)
{
   if (queue->size >= MAX_QUEUE_SIZE)
{
      printf("Queue is full. Cannot enqueue more jobs.\n");
      return;
   }
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
   newNode->next = NULL;
   if (isEmpty(queue))
{
      queue->front = newNode;
```

```
        queue->rear = newNode;
    }
else
{
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
    queue->size++;
}
int dequeue(Queue* queue)
{
    if (isEmpty(queue))
{
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    Node* temp = queue->front;
    int data = temp->data;
    queue->front = queue->front->next;      free(temp);
    queue->size--;
    return data;
}
int main()
{
    Queue printerQueue;
    initQueue(&printerQueue);
    enqueue(&printerQueue, 1);
    enqueue(&printerQueue, 2);
    enqueue(&printerQueue, 3);
    printf("Printing Jobs:\n");
    while (!isEmpty(&printerQueue))
{
        int job = dequeue(&printerQueue);
        printf("Job %d\n", job);
    }
    return 0;
}
```

**OUTPUT:**

```
Printing Jobs:
Job 1
Job 2
Job 3
```

**iii) Solve problems involving circular queues.**

**AIM: C program to implement circular queues**

**SOURCE CODE:**

```
#include<stdio.h>
# define MAX 5
int cqueue_arr[MAX];
int front = -1;    int rear = -1;
void insert(int item)
{
if((front == 0 && rear == MAX-1) || (front == rear+1))
{
printf("Queue Overflow \n");
return;
}
if(front == -1)
{
front = 0;
rear = 0;
}
else
{
if(rear == MAX-1)
rear = 0;
else
rear = rear+1;
}
cqueue_arr[rear] = item ;
}
void deletion()
{
if(front == -1)
{
printf("Queue Underflow \n");
return ;
}
printf("Element deleted from queue is : %d \n",cqueue_arr[front]);
if(front == rear)
{
front = -1;    rear=-1;
}
```

```
else
{
if(front == MAX-1)
front = 0;
else
front = front+1;
}
}
void display()
{
int front_pos = front,rear_pos = rear;
if(front == -1)
{
printf("Queue is empty \n");
return;
}
printf("Queue elements : \n");
if( front_pos <= rear_pos )
while(front_pos <= rear_pos)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
else
{
while(front_pos <= MAX-1)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
front_pos = 0;
while(front_pos <= rear_pos)
{
printf("%d ",cqueue_arr[front_pos]);
front_pos++;
}
}
printf("\n");
}
int main()
{
int choice,item;
do
```

```
{
printf("1.Insert \n");
printf("2.Delete \n");
printf("3.Display \n");
printf("4.Quit \n") ;
printf("Enter your choice : \n");
scanf("%d",&choice);
switch(choice)
{
case 1 :
printf("Input the element for insertion in queue : ");
scanf("%d", &item);
insert(item);   break;
case 2 : deletion();     break;
case 3:  display();      break;
case 4:   break;
default:
printf("Wrong choice \n");
}
}while(choice!=4);
return 0;
}
```

**OUTPUT:**

```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 23
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice : 1
Input the element for insertion in queue : 56
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice :  3
Queue elements : 23 56
```

## Exercise 7
## Stack and Queue Applications

**i) Use a stack to evaluate an infix expression and convert it to postfix.**

**AIM: Use a stack to evaluate an infix expression and convert it to postfix**

**SOURCE CODE:**

```
#include<stdio.h>
#include<ctype.h>
char stack[100];
int top = -1;
void push(char x)
{
 stack[++top] = x;
}
char pop()
{
 if(top == -1)
 return -1;
 else
 return stack[top--];
}
int priority(char x)
{
 if(x == '(')
 return 0;
 if(x == '+' || x == '-')
 return 1;
 if(x == '*' || x == '/')
 return 2;
 return 0;
}
int main()
{
 char exp[100];
 char *e, x;
 printf("Enter the expression : ");
 scanf("%s",exp);
 printf("\n");
 e = exp;
 while(*e != '\0')
```

```
{
if(isalnum(*e))
printf("%c ",*e);
else if(*e == '(')
push(*e);
else if(*e == ')')
{
while((x = pop()) != '(')
printf("%c ", x);
}
else
{
while(priority(stack[top]) >= priority(*e))
printf("%c ",pop());
push(*e);
}
e++;
}
while(top != -1)
{
printf("%c ",pop());
}return 0;
}
```

**OUTPUT:**

```
Enter the expression : a+b+c*d/f
a b + c d * f / +
```

**ii) Create a program to determine whether a given string is a palindrome or not.**

**AIM: Create a C program to determine whether a given string is a palindrome or not**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 50

int top = -1, front = 0;
int stack[MAX];
void push(char);
void pop();

void main()
{
   int i, choice;
   char s[MAX], b;
   while (1)
   {
      printf("1-enter string\n2-exit\n");
      printf("enter your choice\n");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("Enter the String\n");
         scanf("%s", s);
         for (i = 0;s[i] != '\0';i++)
         {
            b = s[i];
            push(b);
         }
         for (i = 0;i < (strlen(s) / 2);i++)
         {
            if (stack[top] == stack[front])
            {
               pop();
               front++;
            }
            else
```

```
                {
                    printf("%s is not a palindrome\n", s);
                    break;
                }
            }
            if ((strlen(s) / 2)  ==  front)
                printf("%s is palindrome\n",  s);
            front  =  0;
            top  =  -1;
            break;
        case 2:
            exit(0);
        default:
            printf("enter correct choice\n");
        }
    }
}

/* to push a character into stack */
void push(char a)
{
    top++;
    stack[top]  =  a;
}

/* to delete an element in stack */
void pop()
{
    top--;
}
```

**OUTPUT:**

```
1-enter string
2-exit
enter your choice
1
Enter the String
mam
mam is palindrome
1-enter string
2-exit
enter your choice
2
```

**iii) Implement a stack or queue to perform comparison and check for symmetry**

**AIM: Implement a stack or queue to perform comparison and check for symmetry**

**SOURCE CODE:**

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

// Function to check if a string is symmetrical
bool isSymmetrical(char str[]) {
    int len = strlen(str);

    for (int i = 0; i < len / 2; i++) {
        if (str[i] != str[len - i - 1]) {
            return false;
        }
    }
    return true;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    if (isSymmetrical(str)) {
        printf("The string is symmetrical.\n");
    } else {
        printf("The string is not symmetrical.\n");
    }

    return 0;
}
```

**OUTPUT:**

```
Enter a string: mam
The string is symmetrical.
```

## Exercise 8
## Binary Search Tree

**i) Implementing a BST using Linked List.**

**AIM: Write a C program to Implementing a BST using Linked List**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 int data;
 struct node *left;
 struct node *right;
};
struct node *root = NULL;
struct node *create_node(int);
void insert(int);
struct node *delete (struct node *, int);
void inorder(struct node *);
int get_data();
int main()
{
 int userChoice;
 char userActive = 'Y';
 int data;
 struct node* result = NULL;
 while (userActive == 'Y' || userActive == 'y')
 {
 printf("\n\n------- Binary Search Tree ------\n");
 printf("\n1. Insert");
 printf("\n2. Delete");
printf("\n\n3. Inorder display ");
 printf("\n4. Exit");
 printf("\n\nEnter Your Choice: ");
 scanf("%d", &userChoice);
 printf("\n");

switch(userChoice)
 {
 case 1:
```

```
 data = get_data();
 insert(data);
 break;
 case 2:
 data = get_data();
 root = delete(root, data);
 break;

 case 3:
 inorder(root);
 break;
 case 4:
 printf("\n\nProgram was terminated\n");
 break;
 default:
 printf("\n\tInvalid Choice\n");
 break;
 }
 printf("\n___________\nDo you want to continue? ");
 scanf(" %c", &userActive);
 }
 return 0;
}
struct node *create_node(int data)
{
 struct node *new_node = (struct node *)malloc(sizeof(struct node));
 if (new_node == NULL)
 {
 printf("\nMemory for new node can't be allocated");
 return NULL;
 }
 new_node->data = data;
 new_node->left = NULL;
 new_node->right = NULL;
 return new_node;
}
void insert(int data)
{
 struct node *new_node = create_node(data);
 if (new_node != NULL)
 {
 if (root == NULL)
 {
```

```
 root = new_node;
 printf("\n* node having data %d was inserted\n", data);
 return;
 }
 struct node *temp = root;
 struct node *prev = NULL;
 while (temp != NULL)
 {
 prev = temp;
 if (data > temp->data)
 {
 temp = temp->right;
 }
 else
 {
 temp = temp->left;
 }
 }
 if (data > prev->data)
 {
 prev->right = new_node;
 }
 else
 {
 prev->left = new_node;
 }
 printf("\n* node having data %d was inserted\n", data);
 }
}
struct node *delete (struct node *root, int key)
{
 if (root == NULL)
 {
 return root;
 }
 if (key < root->data)
 {
 root->left = delete (root->left, key);
 }
 else if (key > root->data)
 {
 root->right = delete (root->right, key);
 }
 else
 {
 if (root->left == NULL)
 {
 struct node *temp = root->right;
 free(root);
 return temp;
 }
 else if (root->right == NULL)
 {
 struct node *temp = root->left;
 free(root);
 return temp;
 }
 struct node *temp = smallest_node(root->right);
 root->data = temp->data;
 root->right = delete (root->right, temp->data);
 }
 return root;
}
struct node *smallest_node(struct node *root)
{
 struct node *curr = root;
 while (curr != NULL && curr->left != NULL)
 {
 curr = curr->left;
 }
 return curr;
}
struct node *largest_node(struct node *root)
{
 struct node *curr = root;
 while (curr != NULL && curr->right != NULL)
 {
 curr = curr->right;
 }
 return curr;
}
void inorder(struct node *root)
{
 if (root == NULL)
 {
```

```
 return;
 }
 inorder(root->left);
 printf("%d ", root->data);
 inorder(root->right);
}
int get_data()
{
 int data;
 printf("\nEnter Data: ");
 scanf("%d", &data);
 return data;
}
```

**OUTPUT:**

```
------- Binary Search Tree ------
1. Insert
2. Delete
3. Inorder
4. Exit
Enter Your Choice: 1
Enter Data: 12
* node having data 12 was inserted
___________
Do you want to continue? y
------ Binary Search Tree ------
1. Insert
2. Delete
3. Inorder
4. Exit
Enter Your Choice: 1
Enter Data: 56
* node having data 56 was inserted
___________
Do you want to continue? y
------- Binary Search Tree ------
1. Insert
2. Delete
3. Inorder
4. Exit
Enter Your Choice: 4
12 56
```

**ii) Traversing of BST.**

**AIM: Write a C program to Implement Traversing of BST**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *left;
   struct Node *right;
};
struct Node* createNode(int data)
 {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
struct Node* insert(struct Node* node, int data)
{
   if (node == NULL)
 {
      return createNode(data);
   }
   if (data < node->data)
{
      node->left = insert(node->left, data);
   } else if (data > node->data)
 {
      node->right = insert(node->right, data);
   }
   return node;
}
void inorder(struct Node* root)
{
   if (root != NULL)
{
      inorder(root->left);
      printf("%d ", root->data);
```

```
        inorder(root->right);
    }
}

void preorder(struct Node* root)
{
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void postorder(struct Node* root)
 {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
int main()
{
    struct Node* root = NULL;
    int n, data;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter the node values:\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &data);
        root = insert(root, data);
    }
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    printf("Preorder traversal: ");
    preorder(root);
    printf("\n");
    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");
    return 0;
}
```

**OUTPUT:**

```
Enter the number of nodes: 6
Enter the node values:3  6  2  7  9  4
Inorder traversal: 2 3 4 6 7 9
Preorder traversal: 3 2 6 4 7 9
Postorder traversal: 2 4 9 7 6 3
```

## Exercise 9
## Hashing

**i) Implement a hash table with collision resolution techniques.**

**AIM: Write a C program to Implement a hash table with collision resolution techniques**

**SOURCE CODE:**

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

#define SIZE 10
struct Node
{
   int key;
   char *value;
   struct Node *next;
};
struct HashTable
{
   struct Node *table[SIZE];
};
struct Node* createNode(int key, char *value)
 {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->key = key;
   newNode->value = strdup(value);
   newNode->next = NULL;
   return newNode;
}
struct HashTable* createHashTable()
{
   struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct HashTable));
   for (int i = 0; i < SIZE; i++) {
      hashTable->table[i] = NULL;
   }
   return hashTable;
}
int hash(int key)
{
   return key % SIZE;
}
```

```
void insert(struct HashTable* hashTable, int key, char *value)
{
   int index = hash(key);
   struct Node *newNode = createNode(key, value);
   if (hashTable->table[index] == NULL)
{
      hashTable->table[index] = newNode;
   }
else
{
      struct Node *temp = hashTable->table[index];
      while (temp->next != NULL)
{
         temp = temp->next;
      }
      temp->next = newNode;
   }
}
char* search(struct HashTable* hashTable, int key)
{
   int index = hash(key);
   struct Node *temp = hashTable->table[index];
   while (temp != NULL)
{
      if (temp->key == key)
{
         return temp->value;
      }
      temp = temp->next;
   }
   return NULL;
}
int main()
{
   struct HashTable* hashTable = createHashTable();
   int key;
   char value[100];
   printf("Enter key-value pairs (key value), enter -1 to stop:\n");
   while (1) {
      scanf("%d", &key);
      if (key == -1)
{
         break;
```

```
        }
        scanf("%s", value);
        insert(hashTable, key, value);
    }

    printf("Enter the key to search: ");
    scanf("%d", &key);
    char* result = search(hashTable, key);
    if (result != NULL)
{
        printf("Value for key %d: %s\n", key, result);
    }
else
{
        printf("Key %d not found.\n", key);
    }
    return 0;
}
```

**OUTPUT:**

```
Enter key-value pairs (key value), enter -1 to stop:
01 ABC
02 XYZ
-1
Enter the key to search: 02
Value for key 2: XYZ
```

**ii) Write a program to implement a simple cache using hashing.**

**AIM: Write a C program to implement a simple cache using hashing**

**SOURCE CODE:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define CACHE_SIZE 10
struct CacheNode
{
   int key;
   char *value;
   struct CacheNode *next;
};
struct Cache
{
   struct CacheNode *table[CACHE_SIZE];
};
struct CacheNode* createCacheNode(int key, char *value)
{
   struct CacheNode* newNode = (struct CacheNode*)malloc(sizeof(struct CacheNode));
   newNode->key = key;
   newNode->value = strdup(value);
   newNode->next = NULL;
   return newNode;
}
struct Cache* createCache()
 {
   struct Cache* cache = (struct Cache*)malloc(sizeof(struct Cache));
   for (int i = 0; i < CACHE_SIZE; i++)
{
      cache->table[i] = NULL;
   }
   return cache;
}
int hash(int key)
 {
   return key % CACHE_SIZE;
}
void insert(struct Cache* cache, int key, char *value)
{
    int index = hash(key);
    struct CacheNode *newNode = createCacheNode(key, value);
    if (cache->table[index] == NULL)
{
        cache->table[index] = newNode;
    }
else
{
        struct CacheNode *temp = cache->table[index];
        while (temp->next != NULL)
 {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
char* search(struct Cache* cache, int key)
 {
    int index = hash(key);
    struct CacheNode *temp = cache->table[index];
    while (temp != NULL)
{
        if (temp->key == key)
{
            return temp->value;
        }
        temp = temp->next;
    }
    return NULL;
}

int main()
{
    struct Cache* cache = createCache();
    int key;
    char value[100];
    printf("Enter key-value pairs (key value), enter -1 to stop:\n");
    while (1) {
        scanf("%d", &key);
        if (key == -1)
{
            break;
        }
        scanf("%s", value);
        insert(cache, key, value);
    }
    printf("Enter the key to search: ");
    scanf("%d", &key);
    char* result = search(cache, key);
    if (result != NULL)
{
        printf("Value for key %d: %s\n", key, result);
    } else {
        printf("Key %d not found.\n", key);
    }

    return 0;
}
```

**OUTPUT:**

```
Enter key-value pairs (key value), enter -1 to stop:
01 QWE
02ASD
03 RTY
-1
Enter the key to search: 09
Key 9 not found.
```

**Textbooks:**

1. Data Structures and algorithm analysis in C, Mark Allen Weiss, Pearson, 2nd Edition.
2. Fundamentals of data structures in C, Ellis Horowitz, Sartaj Sahni, Susan Anderson-Freed, Silicon Press, 2008

**Reference Books:**

1. Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter Sanders
2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft
3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David Ranum
4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
5. Algorithms in C, Parts 1-5 (Bundle): Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms by Robert Sedgewick.