**G.L. BAJAJ INSTITUTE OF TECHNOLOGY & MANAGEMENT, GREATER NOIDA**



**DATA STRUCTURES**

**LAB MANUAL**

**(BCS-351)**

**LAB MANUAL**

**COURSE:** B. TECH (CSE)

**SEM**: IIIrd

## Dept. of Computer Science & Engineering

**G. L. BAJAJ INSTITUTE OF TECHNOLOGY AND MANAGEMENT**

**Plot no. 2, Knowledge Park III, Gr. Noida**

**Table of Content**

1. Vision & Mission of Institute
2. Vision & Mission of Department
3. Program Educational Objectives
4. Program Outcomes
5. Course Outcomes with Blooms Taxonomy
6. Program Specific Outcomes
7. Course Scheme as per university
8. Course Syllabus as per university
9. Mapping of COs - POs & PSOs
10. Rubrics used for Continuous Evaluation for lab session
11. Experiment List (1 ……… n) (all programs as per university list mapped with COs & PSOs)

Each experiment contains followings

1. Objective of the Experiment
2. Code
3. Output (Snapshot)
4. Course Beyond syllabus (at least 2 experiments mapped with COs & PSOs)

Each experiment contains followings

1. Objective of the Experiment
2. Code
3. Output (Snapshot)

**INSTITUTE VISION & MISSION**

**VISION:**

* To be an institute of repute, providing professionally competent and socially sensitive engineers.

**MISSION:**

* To equip with the latest technologies to be globally competitive professionals.
* To inculcate qualities of leadership, professionalism, corporate understanding and executive competence.
* To imbibe and enhance human values, ethics and morals in our students.

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**VISION OF THE DEPARTMENT**

### To build strong teaching environment that responds to the need of industry and challenges of society.

**MISSION OF THE DEPARTMENT**

### **M1:** Developing strong mathematical & computing skill set among the students.

**M2:** Extending the role of computer science and engineering in diverse areas like Internet of things (IoT), Artificial intelligence & Machine Learning and Data Analytics.

**M3:** Imbibing the students with a deep understanding of professional ethics and high integrity to serve the Nation.

**M4:** Providing an environment to the students for their growth both as individuals and as globally competent Computer Science professional and encouragement for innovation & start-up culture.

**PROGRAM EDUCATIONAL OUTCOMES (PEOs)**

**PEO1:** Graduates will work in the area of application software development, artificial intelligence & Machine learning, data analytics, and Internet of things.

**PEO2:** Graduates will become successful software professional with leadership and managerial quality in the modern software industry based on their strong skill on theoretical and practical foundation.

**PEO3:** Graduates will exhibit professional ethics and moral value with capability of working as an individual and as a team to contribute towards the needs of the industry and society.

**PROGRAMME OUTCOMES (POs)**

**Engineering Graduates will be able to:**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, healthy ,safety,legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Course Outcomes with Bloom’s Taxonomy**

|  |  |  |
| --- | --- | --- |
| **Data Structure Lab (BCS351)** | | |
| Course Outcome (CO) | | Bloom’s Knowledge Level (KL) |
| At the end of course, the student will be able to | | |
| **BCS351.1** | Student will be able to Describe how arrays, linked lists, stacks, queues, trees, and graphs are represented in memory, used by the algorithms and their common applications. | K2, K3 |
| **BCS351.2** | Student will be able to Discuss the computational efficiency of the sorting and searching algorithms. | K2, K3 |
| **BCS351.3** | Student will be able to Implementation of Trees and Graphs and perform various operations on these data structure. | K3, K4 |
| **BCS351.4** | Student will be able to Understanding the concept of recursion, application of recursion and its implementation and removal of recursion. | K3, K4 |
| **BCS351.5** | Identify the alternative implementations of data structures with respect to its performance to solve a real-world problem. | K3, K4 |

# **PROGRAM SPECIFIC OUTCOMES *(*PSOs*)***

**PSO.1:** Student will be able to use problem solving skills to develop efficient algorithmic solutions.

**PSO.2:** Student will be able to develop solution for a given problem in the area of artificial intelligence, data analytics, computer vision and IOT through conducive environment and infrastructure.

**Course Syllabus as per university**

**BCS351- Data Structure Lab**

**List of Experiments (Indicative & not limited to)**

**1. Implementing Sorting Techniques:** Bubble Sort, Insertion Sort, Selection Sort, Shell, Sort, Radix Sort, Quick sort

**2. Implementing Searching and Hashing Techniques:** Linear search, Binary search, Methods for Hashing: Modulo Division, Digit Extraction, Fold shift, Fold Boundary, Linear Probe for Collision Resolution. Direct and Subtraction hashing

**3. Implementing Stacks:** Array implementation, Linked List implementation, Evaluation of postfix expression and balancing of parenthesis, Conversion of infix notation to postfix notation

**4. Implementing Queue:** Linked List implementation of ordinary queue, Array implementation of circular queue, Linked List implementation of priority queue, Double ended queue

**5. Implementing Linked List:** Singly Linked Lists, Circular Linked List, Doubly Linked Lists: Insert, Display, Delete, Search, Count, Reverse (SLL), Polynomial, Addition, Comparative study of arrays and linked list

**6. Implementing Trees:** Binary search tree: Create, Recursive traversal: preorder, post order, in order, Search Largest, Node, Smallest Node, Count number of nodes, Heap: Min Heap, Max Heap: reheap Up, reheap Down, Delete, Expression Tree, Heapsort

**7. Implementing Graphs:** Represent a graph using the Adjacency Matrix, BFS, Find the minimum spanning tree (using any method Kruskal’s Algorithm or Prim’s Algorithm) Self Learning Topics: Shortest Path Algorithm

**List of Experiments**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **List of Experiments** | **CO** |
| **1** | **A.** To implement bubble sort.  **B.** To implement insertion sort.  **C.** To implement selection sort.  **D.** To implement quick sort. | CO1 |
| **2** | **A.** To implement linear search.  **B.** To implement binary search. | CO1 |
| **3.** | **A.** To implement stack using array.  **B.** To implement stack using linked list.  **C.** To implement the conversion of infix notation to postfix notation. | CO2 |
| **4** | **A.** To implement queue using linked list.  **B.** To implement circular queue using linked list.  **C.** To implement priority queue using array. | CO2 |
| **5** | **A.** To implement singly linked list.  **B.** To implement circular linked list.  **C.** To implement doubly linked list.  **D.** To implement polynomial addition. | CO3 |
| **6** | **A.** To implement binary search tree using linked list.  **B.** To implement recursive traversal: Preorder, Postorder, and Inorder.  **C.** To implement heap sort. | CO4 |
| **7** | **A.** To implement a graph using the adjacent matrix.  **B.** To implement BFS using linked list.  **C.** To implement any minimum spanning tree algorithm. | CO5 |

**DATA STRUCTURES LAB MANUAL (BCS-351)**

|  |  |
| --- | --- |
| **COs** | **COURSE OUTCOMES** |
| **BCS351.1** | Student will be able to implement and evaluate different searching and sorting algorithms. |
| **BCS351.2** | Students will be able to apply the concepts of stack and queue to solve the problems. |
| **BCS351.3** | Student will be able to understand how to implement linked list and applications. |
| **BCS351.4** | Student will be able to understand implementation concept of tree and apply fundamental algorithmic problems including Tree traversals, BST. |
| **BCS351.5** | Student will be able to understand implementation concept of graph and apply fundamental algorithmic problems including graph traversals and MST. |

**Mapping of Program Outcomes with Course Outcomes (COs)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CO-PO Matrix | | | | | | | | | | | | |
| Course Outcomes | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** |
| **BCS351.1** |  |  |  |  |  |  |  |  |  |  |  |  |
| **BCS351.2** |  |  |  |  |  |  |  |  |  |  |  |  |
| **BCS351.3** |  |  |  |  |  |  |  |  |  |  |  |  |
| **BCS351.4** |  |  |  |  |  |  |  |  |  |  |  |  |
| **BCS351.5** |  |  |  |  |  |  |  |  |  |  |  |  |
| **CO-PSO Matrix** | | | | | | | | | | | | |
| **COs** | **PSO1** | | | | | | **PSO2** | | | | | |
| **BCS351.1** |  | | | | | |  | | | | | |
| **BCS351.2** |  | | | | | |  | | | | | |
| **BCS351.3** |  | | | | | |  | | | | | |
| **BCS351.4** |  | | | | | |  | | | | | |
| **BCS351.5** |  | | | | | |  | | | | | |

**Grade Policy for Laboratory**

We give the grades to the students i.e. **A/α, B/β, C/γ** for their performance in lab as per the given table:

|  |  |  |
| --- | --- | --- |
| **Performance** | **Record** | **Viva-Voce** |
| Attention | Time line | Experimental Knowledge |
| Effectiveness | Presentation | Expression |
| Discipline | Quality of Content | Communication |
| Dedication | Competence | Confidence |

The grades **α, β, γ**/ **A, B, C** are awarded in lab record, lab performance and viva voce as per the above define table.

|  |  |
| --- | --- |
| **Grade** | **Range of marks** |
| **A/α** | 4-5 Marks |
| **B/β** | 3-4 Marks |
| **C/γ** | 2-3 marks |

###### **Experiment No: 1.A**

***Aim*:** To implement Bubble Sorting.

***Program:***

#include <stdio.h>

void bubbleSort(int arr[], int size) {

int i, j, temp;

for (i = 0; i< size - 1; i++) {

for (j = 0; j < size - i - 1; j++) {

if (arr[j] >arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main () {

int arr[20];

int i, n;

printf("Enter the number of items in the array: ");

scanf("%d", &n);

printf("Enter the data in the array:\n");

for (i = 0; i< n; i++) {

scanf("%d", &arr[i]);

}

bubbleSort(arr, n);

printf("\nSorted Array:\n");

for (i = 0; i< n; i++) {

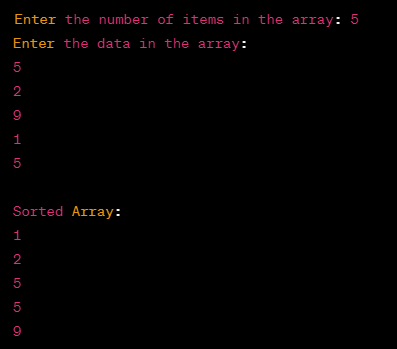
printf("%d\n", arr[i]);

}

return 0;

}

***Output:***

******

###### **Experiment No: 1.B**

***Aim*:** To implement Insertion Sorting.

***Program:***

#include<stdio.h>

#include<conio.h>

void insert(int [],int);

void main()

{

int a[20],i,n;

clrscr();

printf("Enter the number of items in the array");

scanf("%d",&n);

printf("Enter the data in the array");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

insert(a,n);

getch();

}

void insert(int a[],int n)

{

int i,j,temp;

for(i=1;i<n;i++)

{

temp=a[i];

for(j=i-1;j>=0;j--)

{

if(a[j]>temp)

{

a[j+1]=a[j];

}

else

break;

}

a[j+1]=temp;

}

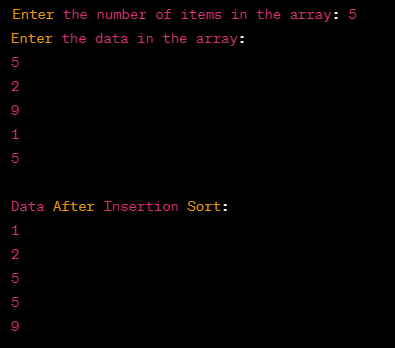
printf("Data After Insertion Sort");

for(i=0;i<n;i++)

printf("\n%d",a[i]);

}

***Output:***

****

###### **Experiment No: 1.C**

***Aim*:** To implement Selection Sorting.

***Program:***

#include<stdio.h>

#include<conio.h>

void select(int [],int);

int min(int [],int,int);

void main()

{

int a[20],i,n;

clrscr();

printf("Enter the number of items in the array");

scanf("%d",&n);

printf("Enter the data in the array");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

select(a,n);

getch();

}

void select(int a[],int n)

{

int i,loc,temp;

loc=0;

temp=0;

for(i=0;i<n;i++)

{

loc=min(a,i,n);

temp=a[loc];

a[loc]=a[i]; a[i]=temp;

}

printf("\nData After Selection Sort");

for(i=0;i<n;i++)

printf("\n%d",a[i]);

}

int min(int a[],int lb,intub)

{

int m=lb;

while(lb<ub)

{

if(a[lb]<a[m])

{

m=lb;

}

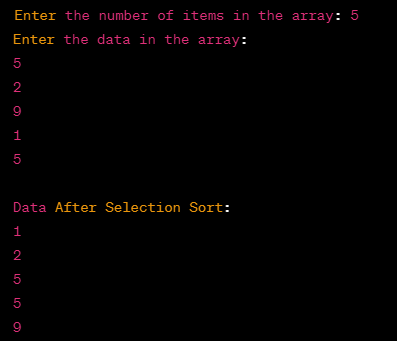
lb++;

}

return m;

}

***Output:***



###### **Experiment No: 1.D**

***Aim*:** To implement Quick Sorting.

***Program:***

#include <stdio.h>

void swap(int\* a, int\* b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements: ", n);

for (int i = 0; i< n; i++) {

scanf("%d", &arr[i]);

}

printf("Unsorted array: ");

for (int i = 0; i< n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

quickSort(arr, 0, n - 1);

printf("Sorted array: ");

for (int i = 0; i< n; i++) {

printf("%d ", arr[i]);

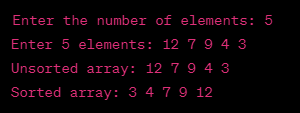
}

printf("\n");

return 0;

}

***Output:***

****

###### **Experiment No: 2.A**

***Aim*:** To implement Linear Search.

***Program:***

#include <stdio.h>

#include <conio.h>

void main()

{

int a[10], i, item, flag = 0;

clrscr();

printf("Enter the data in the array");

for (i = 0; i< 10; i++)

{

scanf("%d", &a[i]);

}

printf("Enter the element to be searched");

scanf("%d", &item);

for (i = 0; i< 10; i++)

{

if (item == a[i])

{

flag = 1;

break;

}

}

if (flag == 0)

printf("Element Not Found");

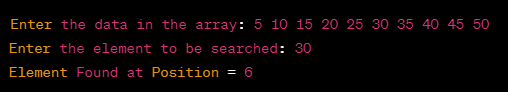
else

printf("Element Found at Position =%d", i);

getch();

}

***Output:***

******

###### **Experiment No: 2.B**

***Aim*:** To implement Binary Search.

***Program:***

#include<stdio.h>

#include<conio.h>

void main()

{

int a[20],n,mid,beg,i,end,item,loc=-1;

clrscr();

printf("Enter the number of elements to be entered\n");

scanf("%d",&n);

printf("Enter the elements in ascending order");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

printf("Enter the element to be searched");

scanf("%d",&item);

beg=0;

end=n-1;

while(beg<=end)

{

mid=(beg+end)/2; if(item==a[mid])

{

loc=mid;

break;

}

else if(a[mid]<item)

beg=mid+1;

else

end=mid-1;

}

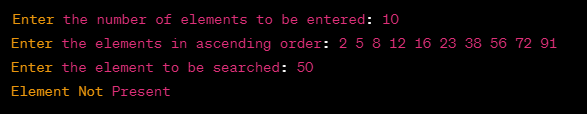
if(loc==-1)

printf("Element Not Present");

else

printf("Element found at =%d",loc);

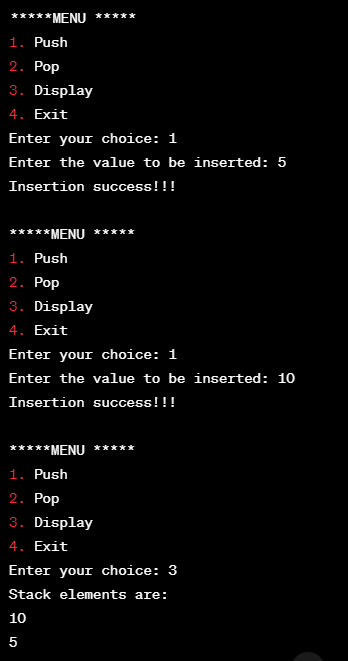
***Output:***

******

###### **Experiment No: 3.A**

***Aim*:** To implement Stack using Array.

1. ***Program:***
2. #include <stdio.h>
3. #include <conio.h>
4. #define SIZE 10
5. void push(int);
6. void pop();
7. void display();
8. int stack[SIZE], top = -1;
9. void main()
10. {
11. int value, choice;
12. clrscr();
13. while (1)
14. {
15. printf("\n\n\*\*\*\*\*MENU \*\*\*\*\*\n");
16. printf("1. Push\n2. Pop\n3. Display\n4. Exit");
17. printf("\nEnter your choice: ");
18. scanf("%d", &choice);
19. switch (choice)
20. {
21. case 1:
22. printf("Enter the value to be insert: ");
23. scanf("%d", &value);
24. push(value);
25. break;
26. case 2:
27. pop();
28. break;
29. case 3:
30. display();
31. break;
32. case 4:
33. exit(0);
34. default:
35. printf("\nWrong selection!!! Try again!!!");
36. }
37. }
38. }
39. void push(int value)
40. {
41. if (top == SIZE - 1)
42. printf("\nStack is Full!!! Insertion is not possible!!!");
43. else
44. {
45. top++;
46. stack[top] = value;
47. printf("\nInsertion success!!!");
48. }
49. }
50. void pop()
51. {
52. if (top == -1)
53. printf("\nStack is Empty!!! Deletion is not possible!!!");
54. else
55. {
56. printf("\nDeleted : %d", stack[top]);
57. top--;
58. }
59. }
60. void display()
61. {
62. if (top == -1)
63. printf("\nStack is Empty!!!");
64. else
65. {
66. int i;
67. printf("\nStack elements are:\n");
68. for (i = top; i>= 0; i--)
69. printf("%d\n", stack[i]);
70. }
71. }
72. ***Output:***

******

###### **Experiment No: 3.B**

***Aim*:** To implement Stack using Linked List.

***Program:***

#include <stdio.h>

#include <conio.h>

struct Node

{

int data;

struct Node \* next;

}\*top = NULL;

void push(int);

void pop();

void display();

void main()

{

int choice, value;

clrscr();

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;

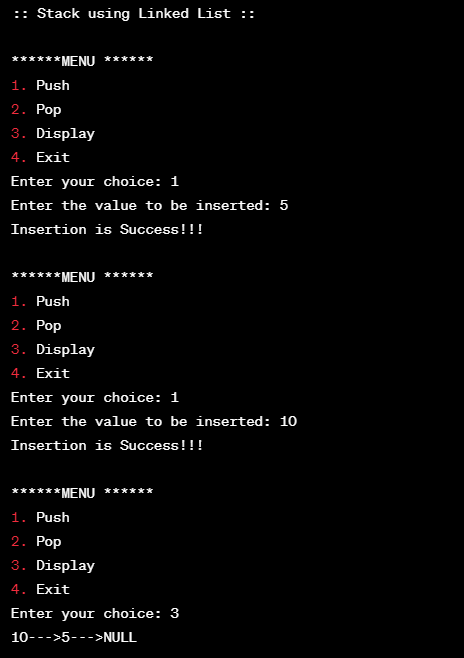
}

printf("%d--->NULL", temp->data);

}

}

***Output:***

******

###### **Experiment No: 3.C**

***Aim*:** To implement the conversion of Infix to Postfix expression.

***Program:***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Define a structure for a stack

struct Stack {

int top;

unsigned capacity;

char\* array;

};

// Function to create a stack of a given capacity

struct Stack\* createStack(unsigned capacity) {

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->capacity = capacity;

stack->top = -1;

stack->array = (char\*)malloc(stack->capacity \* sizeof(char));

return stack;

}

// Function to check if a character is an operator

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

// Function to return the precedence of an operator

int precedence(char op) {

if (op == '+' || op == '-')

return 1;

if (op == '\*' || op == '/')

return 2;

return 0;

}

// Function to push a character onto the stack

void push(struct Stack\* stack, char item) {

if (stack->top == stack->capacity - 1) {

printf("Stack overflow\n");

exit(1);

}

stack->array[++stack->top] = item;

}

// Function to pop a character from the stack

char pop(struct Stack\* stack) {

if (stack->top == -1) {

printf("Stack underflow\n");

exit(1);

}

return stack->array[stack->top--];

}

// Function to convert infix expression to postfix notation

void infixToPostfix(char\* infix) {

struct Stack\* stack = createStack(strlen(infix));

int i, j;

i = j = 0;

while (infix[i] != '\0') {

char c = infix[i];

if (isalnum(c)) {

printf("%c", c);

} else if (c == '(') {

push(stack, c);

} else if (c == ')') {

while (stack->top != -1 && stack->array[stack->top] != '(') {

printf("%c", pop(stack));

}

if (stack->top != -1 && stack->array[stack->top] != '(') {

printf("Invalid expression\n");

exit(1);

} else {

pop(stack);

}

} else {

while (stack->top != -1 && precedence(c) <= precedence(stack->array[stack->top])) {

printf("%c", pop(stack));

}

push(stack, c);

}

i++;

}

while (stack->top != -1) {

printf("%c", pop(stack));

}

}

int main() {

char infix[100];

printf("Enter an infix expression: ");

scanf("%s", infix);

printf("Postfix expression: ");

infixToPostfix(infix);

printf("\n");

return 0;

}

***Output:***

******

###### **Experiment No: 4.A**

***Aim*:** To implement the Queue using Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Node structure for the linked list

struct Node {

int data;

struct Node\* next;

};

// Queue structure

struct Queue {

struct Node\* front; // Pointer to the front of the queue

struct Node\* rear; // Pointer to the rear of the queue

};

// Function to create a new node

struct Node\* newNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

// Function to create an empty queue

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->front = queue->rear = NULL;

return queue;

}

// Function to check if the queue is empty

int isEmpty(struct Queue\* queue) {

return (queue->front == NULL);

}

// Function to enqueue (add) an element to the queue

void enqueue(struct Queue\* queue, int data) {

struct Node\* newNode = newNode(data);

// If the queue is empty, set the new node as both front and rear

if (isEmpty(queue)) {

queue->front = queue->rear = newNode;

return;

}

// Add the new node at the end and update the rear pointer

queue->rear->next = newNode;

queue->rear = newNode;

}

// Function to dequeue (remove) an element from the queue

int dequeue(struct Queue\* queue) {

// If the queue is empty, return -1 (or any sentinel value)

if (isEmpty(queue)) {

printf("Queue is empty. Cannot dequeue.\n");

return -1; // Placeholder value for an empty queue

}

// Store the front node and move the front pointer to the next node

struct Node\* temp = queue->front;

queue->front = queue->front->next;

// If dequeued node was the only node, update rear to NULL

if (queue->front == NULL) {

queue->rear = NULL;

}

int dequeuedData = temp->data;

free(temp);

return dequeuedData;

}

// Function to display the queue elements

void display(struct Queue\* queue) {

struct Node\* current = queue->front;

if (isEmpty(queue)) {

printf("Queue is empty.\n");

return;

}

printf("Queue elements: ");

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

// Example usage with user input

int main() {

struct Queue\* queue = createQueue();

int choice, data;

do {

printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to enqueue: ");

scanf("%d", &data);

enqueue(queue, data);

break;

case 2:

printf("Dequeued element: %d\n", dequeue(queue));

break;

case 3:

display(queue);

break;

case 4:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please enter a valid option.\n");

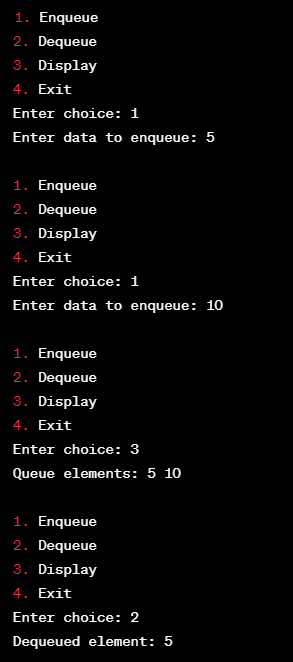
}

} while (choice != 4);

return 0;

}

***Output:***

******

###### **Experiment No: 4.B**

***Aim*:** To implement the circular Queue using Array.

***Program:***

#include <stdio.h>

#define SIZE 5 /\* Size of Circular Queue \*/

int CQ[SIZE], f = -1, r = -1; /\* Global declarations \*/

int CQinsert(int elem) { /\* Function for Insert operation \*/

if ((f == 0 && r == SIZE - 1) || (r == (f - 1) % (SIZE - 1))) {

printf("\n\nOverflow!!!! Queue is full.\n\n");

return -1; /\* Indicate insertion failure \*/

} else if (f == -1) {

f = r = 0;

CQ[r] = elem;

} else if (r == SIZE - 1 &&f != 0) {

r = 0;

CQ[r] = elem;

} else {

r++;

CQ[r] = elem;

}

return 0; /\* Indicate successful insertion \*/

}

int CQdelete() { /\* Function for Delete operation \*/

int deletedElem;

if (f == -1) {

printf("\n\nUnderflow!!!! Queue is empty.\n\n");

return -1; /\* Indicate deletion failure \*/

}

deletedElem = CQ[f];

if (f == r) {

f = r = -1;

} else if (f == SIZE - 1) {

f = 0;

} else {

f++;

}

return deletedElem;

}

int main() { /\* Main Program \*/

int opn, elem;

do {

printf("\n ### Circular Queue Operations ### \n\n");

printf(" Press 1-Insert, 2-Delete, 3-Exit\n");

printf(" Your option ? ");

scanf("%d", &opn);

switch (opn) {

case 1:

printf("\n Read the element to be Inserted ? ");

scanf("%d", &elem);

if (CQinsert(elem) == -1) {

// Failed insertion

printf("\n Insertion failed. Queue is full.\n");

} else {

printf("\n Element %d inserted successfully.\n", elem);

}

break;

case 2:

elem = CQdelete();

if (elem != -1) {

printf("\n Deleted Element is %d\n", elem);

} else {

printf("\n Deletion failed. Queue is empty.\n");

}

break;

case 3:

printf("\n Terminating \n");

break;

default:

printf("\n Invalid Option !!! Try Again !! \n");

break;

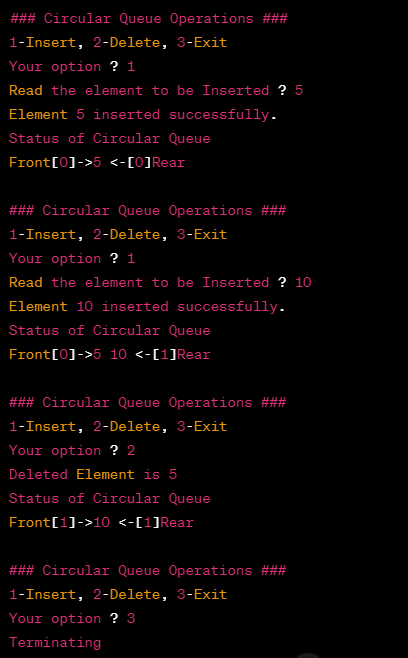
}

} while (opn != 3);

return 0;

}

***Output:***



###### **Experiment No: 4.C**

***Aim*:** To implement the Priority Queue using Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a node in the priority queue

struct Node {

int data;

int priority;

struct Node\* next;

};

// Function to create a new node

struct Node\* createNode(int data, int priority) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->priority = priority;

newNode->next = NULL;

return newNode;

}

// Function to insert an element into the priority queue

void enqueue(struct Node\*\* head, int data, int priority) {

struct Node\* newNode = createNode(data, priority);

if (\*head == NULL || priority < (\*head)->priority) {

newNode->next = \*head;

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL && current->next->priority <= priority) {

current = current->next;

}

newNode->next = current->next;

current->next = newNode;

}

}

// Function to remove and return the element with the highest priority

int dequeue(struct Node\*\* head) {

if (\*head == NULL) {

printf("Priority queue is empty.\n");

exit(1);

}

struct Node\* temp = \*head;

int data = temp->data;

\*head = (\*head)->next;

free(temp);

return data;

}

// Function to check if the priority queue is empty

int isEmpty(struct Node\* head) {

return head == NULL;

}

// Function to display the elements in the priority queue

void display(struct Node\* head) {

if (head == NULL) {

printf("Priority queue is empty.\n");

return;

}

printf("Priority queue elements:\n");

while (head != NULL) {

printf("Data: %d, Priority: %d\n", head->data, head->priority);

head = head->next;

}

printf("\n");

}

int main() {

struct Node\* pq = NULL;

int choice, data, priority;

while (1) {

printf("1. Enqueue\n");

printf("2. Dequeue\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data and priority to enqueue: ");

scanf("%d %d", &data, &priority);

enqueue(&pq, data, priority);

display(pq);

break;

case 2:

if (!isEmpty(pq)) {

printf("Dequeued element: %d\n", dequeue(&pq));

display(pq);

} else {

printf("Priority queue is empty.\n");

}

break;

case 3:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

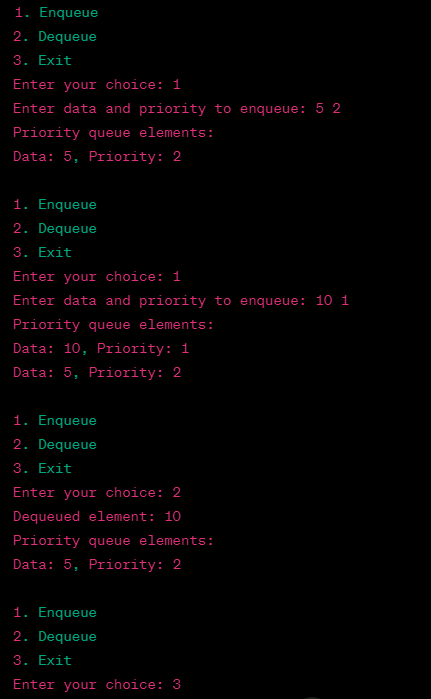
}

}

return 0;

}

***Output:***

******

###### **Experiment No: 5.A**

***Aim*:** To implement the Singly Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a single linked list node

struct Node {

int data;

struct Node\* next;

};

// Function to insert a new node at the beginning of the list

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

// Function to insert a new node at the end of the list

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

// Function to insert a new node at a specific position in the list

void insertAtPosition(struct Node\*\* head, int data, int position) {

if (position < 0) {

printf("Invalid position. Position must be non-negative.\n");

return;

}

if (position == 0) {

insertAtBeginning(head, data);

return;

}

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

struct Node\* current = \*head;

int currentPosition = 0;

while (current != NULL &&currentPosition< position - 1) {

current = current->next;

currentPosition++;

}

if (current == NULL) {

printf("Invalid position. Position exceeds the length of the list.\n");

return;

}

newNode->next = current->next;

current->next = newNode;

}

// Function to delete a node at the beginning of the list

void deleteAtBeginning(struct Node\*\* head) {

if (\*head == NULL) {

printf("The list is already empty.\n");

return;

}

struct Node\* temp = \*head;

\*head = (\*head)->next;

free(temp);

}

// Function to delete a node at the end of the list

void deleteAtEnd(struct Node\*\* head) {

if (\*head == NULL) {

printf("The list is already empty.\n");

return;

}

if ((\*head)->next == NULL) {

free(\*head);

\*head = NULL;

return;

}

struct Node\* current = \*head;

while (current->next->next != NULL) {

current = current->next;

}

free(current->next);

current->next = NULL;

}

// Function to delete a node at a specific position in the list

void deleteAtPosition(struct Node\*\* head, int position) {

if (\*head == NULL) {

printf("The list is already empty.\n");

return;

}

if (position < 0) {

printf("Invalid position. Position must be non-negative.\n");

return;

}

if (position == 0) {

deleteAtBeginning(head);

return;

}

struct Node\* current = \*head;

struct Node\* prev = NULL;

int currentPosition = 0;

while (current != NULL &&currentPosition< position) {

prev = current;

current = current->next;

currentPosition++;

}

if (current == NULL) {

printf("Invalid position. Position exceeds the length of the list.\n");

return;

}

prev->next = current->next;

free(current);

}

// Function to display the elements of the list

void display(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, data, position;

while (1) {

printf("1. Insert at the beginning\n");

printf("2. Insert at the end\n");

printf("3. Insert at a specific position\n");

printf("4. Delete at the beginning\n");

printf("5. Delete at the end\n");

printf("6. Delete at a specific position\n");

printf("7. Display\n");

printf("8. Quit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at the beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to insert at the end: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

printf("Enter data to insert: ");

scanf("%d", &data);

printf("Enter position to insert at: ");

scanf("%d", &position);

insertAtPosition(&head, data, position);

break;

case 4:

deleteAtBeginning(&head);

break;

case 5:

deleteAtEnd(&head);

break;

case 6:

printf("Enter position to delete at: ");

scanf("%d", &position);

deleteAtPosition(&head, position);

break;

case 7:

display(head);

break;

case 8:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

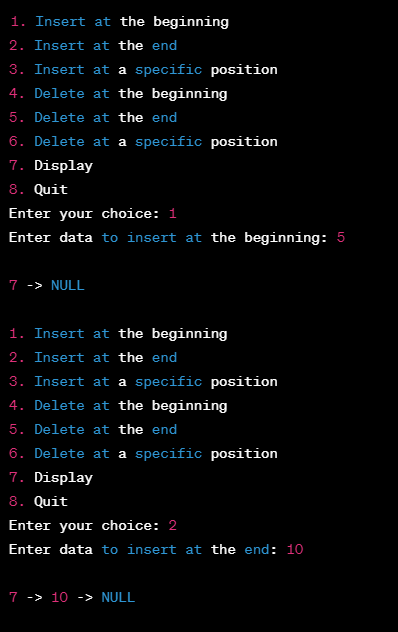
}

}

return 0;

}

***Output:***

******

###### **Experiment No: 5.B**

***Aim*:** To implement the Circular Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a circular singly linked list node

struct Node {

int data;

struct Node\* next;

};

// Function to insert a new node at the beginning of the circular list

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

if (\*head == NULL) {

\*head = newNode;

newNode->next = \*head;

} else {

struct Node\* current = \*head;

while (current->next != \*head) {

current = current->next;

}

current->next = newNode;

newNode->next = \*head;

\*head = newNode;

}

}

// Function to delete a node with a given value from the circular list

void deleteNode(struct Node\*\* head, int key) {

if (\*head == NULL) {

printf("The circular list is empty.\n");

return;

}

struct Node\* current = \*head;

struct Node\* prev = NULL;

do {

if (current->data == key) {

if (prev == NULL) {

struct Node\* last = \*head;

while (last->next != \*head) {

last = last->next;

}

last->next = current->next;

\*head = current->next;

} else {

prev->next = current->next;

}

free(current);

printf("Node with data %d deleted from the circular list.\n", key);

return;

}

prev = current;

current = current->next;

} while (current != \*head);

printf("Node with data %d not found in the circular list.\n", key);

}

// Function to display the elements of the circular list

void display(struct Node\* head) {

if (head == NULL) {

printf("The circular list is empty.\n");

return;

}

struct Node\* current = head;

do {

printf("%d -> ", current->data);

current = current->next;

} while (current != head);

printf("...\n"); // Indicates the circular structure

}

int main() {

struct Node\* head = NULL;

int choice, data, key;

while (1) {

printf("1. Insert at the beginning\n");

printf("2. Delete a node\n");

printf("3. Display\n");

printf("4. Quit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at the beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to delete: ");

scanf("%d", &key);

deleteNode(&head, key);

break;

case 3:

printf("Circular Linked List:\n");

display(head);

break;

case 4:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

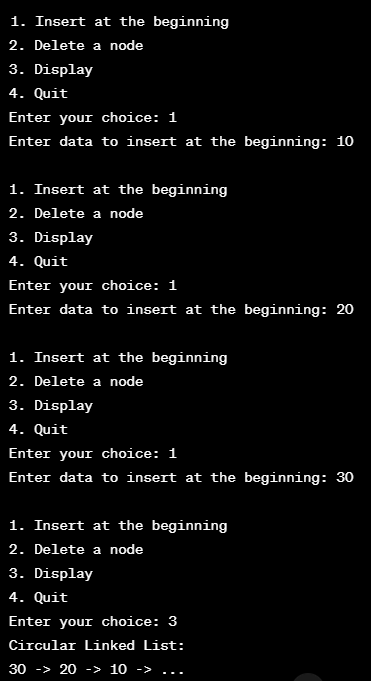
}

}

return 0;

}

***Output:***



###### **Experiment No: 5.C**

***Aim*:** To implement the Doubly Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a doubly linked list node

struct Node {

int data;

struct Node\* prev;

struct Node\* next;

};

// Function to insert a new node at the beginning of the doubly linked list

void insertAtBeginning(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->prev = NULL;

if (\*head == NULL) {

newNode->next = NULL;

\*head = newNode;

} else {

newNode->next = \*head;

(\*head)->prev = newNode;

\*head = newNode;

}

}

// Function to insert a new node at the end of the doubly linked list

void insertAtEnd(struct Node\*\* head, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->prev = NULL;

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

newNode->prev = current;

}

}

// Function to delete a node with a given value from the doubly linked list

void deleteNode(struct Node\*\* head, int key) {

if (\*head == NULL) {

printf("The doubly linked list is empty.\n");

return;

}

struct Node\* current = \*head;

while (current != NULL) {

if (current->data == key) {

if (current->prev != NULL) {

current->prev->next = current->next;

} else {

\*head = current->next;

}

if (current->next != NULL) {

current->next->prev = current->prev;

}

free(current);

printf("Node with data %d deleted from the doubly linked list.\n", key);

return;

}

current = current->next;

}

printf("Node with data %d not found in the doubly linked list.\n", key);

}

// Function to display the elements of the doubly linked list

void display(struct Node\* head) {

if (head == NULL) {

printf("The doubly linked list is empty.\n");

return;

}

struct Node\* current = head;

while (current != NULL) {

printf("%d <-> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

int choice, data, key;

while (1) {

printf("1. Insert at the beginning\n");

printf("2. Insert at the end\n");

printf("3. Delete a node\n");

printf("4. Display\n");

printf("5. Quit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert at the beginning: ");

scanf("%d", &data);

insertAtBeginning(&head, data);

break;

case 2:

printf("Enter data to insert at the end: ");

scanf("%d", &data);

insertAtEnd(&head, data);

break;

case 3:

printf("Enter data to delete: ");

scanf("%d", &key);

deleteNode(&head, key);

break;

case 4:

printf("Doubly Linked List:\n");

display(head);

break;

case 5:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

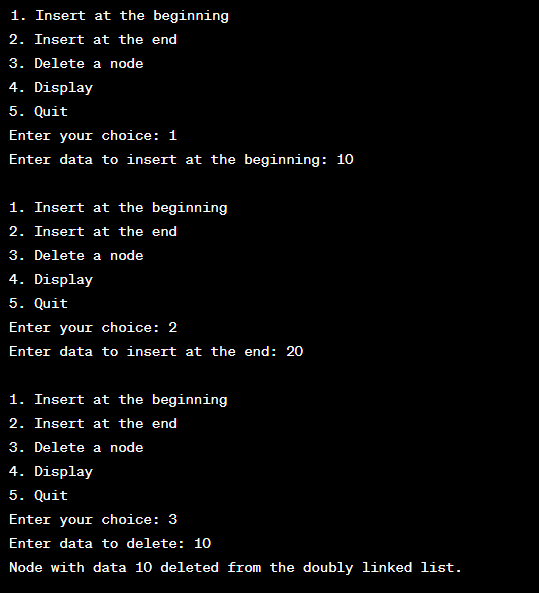
}

}

return 0;

}

***Output:***

******

###### **Experiment No: 5.D**

***Aim*:** To implement the Doubly Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a polynomial term

struct Term {

int coefficient;

int exponent;

struct Term\* next;

};

// Function to create a new term

struct Term\* createTerm(int coef, int exp) {

struct Term\* newTerm = (struct Term\*)malloc(sizeof(struct Term));

newTerm->coefficient = coef;

newTerm->exponent = exp;

newTerm->next = NULL;

return newTerm;

}

// Function to insert a term into a polynomial

void insertTerm(struct Term\*\* poly, int coef, int exp) {

struct Term\* newTerm = createTerm(coef, exp);

if (\*poly == NULL) {

\*poly = newTerm;

} else {

struct Term\* current = \*poly;

struct Term\* prev = NULL;

while (current != NULL && current->exponent > exp) {

prev = current;

current = current->next;

}

if (current != NULL && current->exponent == exp) {

current->coefficient += coef;

free(newTerm);

} else {

newTerm->next = current;

if (prev != NULL) {

prev->next = newTerm;

} else {

\*poly = newTerm;

}

}

}

}

// Function to display a polynomial

void displayPolynomial(struct Term\* poly) {

if (poly == NULL) {

printf("0\n");

return;

}

while (poly != NULL) {

printf("%dx^%d", poly->coefficient, poly->exponent);

if (poly->next != NULL) {

printf(" + ");

}

poly = poly->next;

}

printf("\n");

}

// Function to add two polynomials

struct Term\* addPolynomials(struct Term\* poly1, struct Term\* poly2) {

struct Term\* result = NULL;

while (poly1 != NULL || poly2 != NULL) {

int coef1 = 0, coef2 = 0;

int exp1 = -1, exp2 = -1;

if (poly1 != NULL) {

coef1 = poly1->coefficient;

exp1 = poly1->exponent;

poly1 = poly1->next;

}

if (poly2 != NULL) {

coef2 = poly2->coefficient;

exp2 = poly2->exponent;

poly2 = poly2->next;

}

int sum = coef1 + coef2;

insertTerm(&result, sum, exp1); // Add the terms to the result polynomial

}

return result;

}

int main() {

struct Term\* poly1 = NULL;

struct Term\* poly2 = NULL;

struct Term\* result = NULL;

int coef, exp, terms;

printf("Enter the number of terms in the first polynomial: ");

scanf("%d", &terms);

printf("Enter the terms (coefficient exponent) for the first polynomial:\n");

for (int i = 0; i< terms; i++) {

scanf("%d %d", &coef, &exp);

insertTerm(&poly1, coef, exp);

}

printf("Enter the number of terms in the second polynomial: ");

scanf("%d", &terms);

printf("Enter the terms (coefficient exponent) for the second polynomial:\n");

for (int i = 0; i< terms; i++) {

scanf("%d %d", &coef, &exp);

insertTerm(&poly2, coef, exp);

}

printf("First polynomial: ");

displayPolynomial(poly1);

printf("Second polynomial: ");

displayPolynomial(poly2);

result = addPolynomials(poly1, poly2);

printf("Result of addition: ");

displayPolynomial(result);

// Free memory

struct Term\* temp;

while (poly1 != NULL) {

temp = poly1;

poly1 = poly1->next;

free(temp);

}

while (poly2 != NULL) {

temp = poly2;

poly2 = poly2->next;

free(temp);

}

while (result != NULL) {

temp = result;

result = result->next;

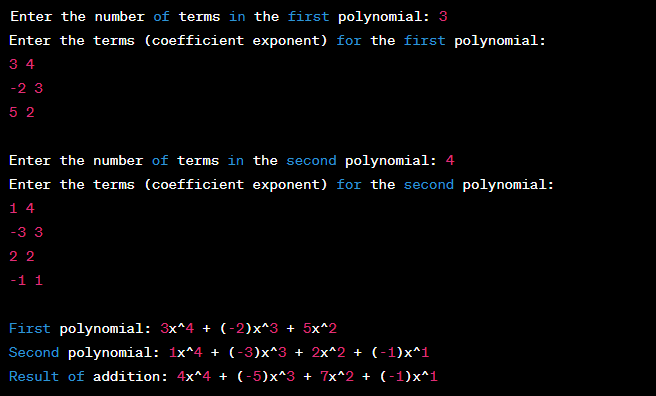
free(temp);

}

return 0;

}

***Output:***



###### **Experiment No: 6.A**

***Aim*:** To implement Binary search tree using Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Structure for a Node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new Node

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

// Function to insert a Node into the BST

struct Node\* insert(struct Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else if (data > root->data) {

root->right = insert(root->right, data);

}

return root;

}

// Function to print the BST in inorder traversal

void inorderTraversal(struct Node\* root) {

if (root != NULL) {

inorderTraversal(root->left);

printf("%d ", root->data);

inorderTraversal(root->right);

}

}

int main() {

struct Node\* root = NULL;

int choice, data;

while (1) {

printf("Binary Search Tree Menu:\n");

printf("1. Insert a node\n");

printf("2. Print in-order traversal\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert: ");

scanf("%d", &data);

root = insert(root, data);

break;

case 2:

printf("In-order traversal: ");

inorderTraversal(root);

printf("\n");

break;

case 3:

exit(0);

default:

printf("Invalid choice!\n");

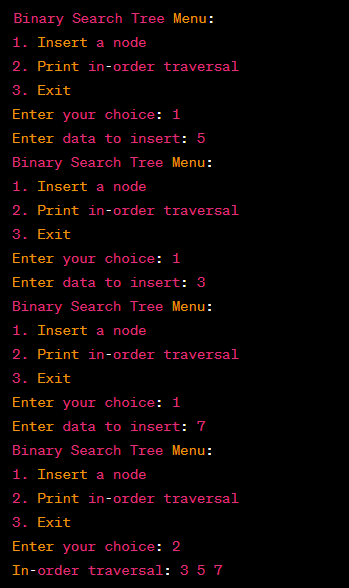
}

}

return 0;

}

***Output:***

******

###### **Experiment No: 6.B**

***Aim*:** To implement recursive traversal: Preorder, Postorder, Inorder.

***Program:***

#include <stdio.h>

#include <stdlib.h>

// Define a structure for a binary tree node

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new binary tree node

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;

}

// Function to build a binary tree using user input

struct Node\* buildTree() {

int data;

printf("Enter data (or -1 for empty node): ");

scanf("%d", &data);

if (data == -1) {

return NULL;

}

struct Node\* root = createNode(data);

printf("Enter left child of %d:\n", data);

root->left = buildTree();

printf("Enter right child of %d:\n", data);

root->right = buildTree();

return root;

}

// Function for pre-order traversal

void preOrder(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->data);

preOrder(root->left);

preOrder(root->right);

}

}

// Function for in-order traversal

void inOrder(struct Node\* root) {

if (root != NULL) {

inOrder(root->left);

printf("%d ", root->data);

inOrder(root->right);

}

}

// Function for post-order traversal

void postOrder(struct Node\* root) {

if (root != NULL) {

postOrder(root->left);

postOrder(root->right);

printf("%d ", root->data);

}

}

int main() {

struct Node\* root = NULL;

printf("Enter the elements of the binary tree:\n");

root = buildTree();

printf("Binary Tree Traversals:\n");

printf("Pre-order traversal: ");

preOrder(root);

printf("\n");

printf("In-order traversal: ");

inOrder(root);

printf("\n");

printf("Post-order traversal: ");

postOrder(root);

printf("\n");

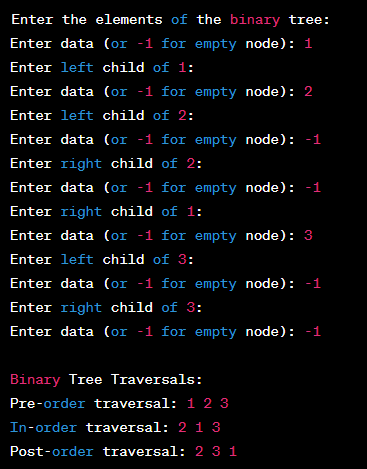
// Free memory (optional)

// You can add code here to free the dynamically allocated memory for the tree

return 0;

}

***Output:***

******

**xperiment No: 18**

###### **Experiment No: 6.C**

***Aim*:** To implement Heap Sort.

***Program:***

#include<stdio.h>

void heapsort(int[],int);

void heapify(int[],int);

void adjust(int[],int);

main() {

int n,i,a[50];

system("clear");

printf("\nEnter the limit:");

scanf("%d",&n);

printf("\nEnter the elements:");

for (i=0;i<n;i++)

scanf("%d",&a[i]);

heapsort(a,n);

printf("\nThe Sorted Elements Are:\n");

for (i=0;i<n;i++)

printf("\t%d",a[i]);

printf("\n");

}

void heapsort(int a[],int n) {

int i,t;

heapify(a,n);

for (i=n-1;i>0;i--) {

t = a[0];

a[0] = a[i];

a[i] = t;

adjust(a,i);

}

}

void heapify(int a[],int n) {

int k,i,j,item;

for (k=1;k<n;k++) {

item = a[k];

i = k;

j = (i-1)/2;

while((i>0)&&(item>a[j])) {

a[i] = a[j];

i = j;

j = (i-1)/2;

}

a[i] = item;

}

}

void adjust(int a[],int n) {

int i,j,item;

j = 0;

item = a[j];

i = 2\*j+1;

while(i<=n-1) {

if(i+1 <= n-1)

if(a[i] <a[i+1])

i++;

if(item<a[i]) {

a[j] = a[i];

j = i;

i = 2\*j+1;

} else

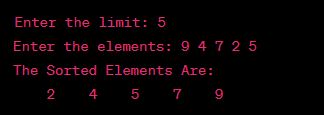
break;

}

a[j] = item;

}

***Output:***



###### 

###### **Experiment No: 7.A**

***Aim*:** To implement a graph using adjacent matrix.

***Program:***

#include <stdio.h>

#define MAX\_NODES 100

// Define the graph structure

struct Graph {

int vertices;

int adjacencyMatrix[MAX\_NODES][MAX\_NODES];

};

// Function to initialize the graph

void initGraph(struct Graph \*graph, int vertices) {

graph->vertices = vertices;

// Initialize the adjacency matrix with zeros

for (int i = 0; i< vertices; i++) {

for (int j = 0; j < vertices; j++) {

graph->adjacencyMatrix[i][j] = 0;

}

}

}

// Function to add an edge between two vertices

void addEdge(struct Graph \*graph, int source, int destination) {

// Since it's an undirected graph, we set both entries to 1

graph->adjacencyMatrix[source][destination] = 1;

graph->adjacencyMatrix[destination][source] = 1;

}

// Function to print the adjacency matrix

void printGraph(struct Graph \*graph) {

printf("Adjacency Matrix:\n");

for (int i = 0; i< graph->vertices; i++) {

for (int j = 0; j < graph->vertices; j++) {

printf("%d ", graph->adjacencyMatrix[i][j]);

}

printf("\n");

}

}

int main() {

struct Graph graph;

int numVertices, numEdges, source, destination;

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

// Initialize the graph

initGraph(&graph, numVertices);

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

// Add edges to the graph

for (int i = 0; i<numEdges; i++) {

printf("Enter edge (source destination): ");

scanf("%d %d", &source, &destination);

addEdge(&graph, source, destination);

}

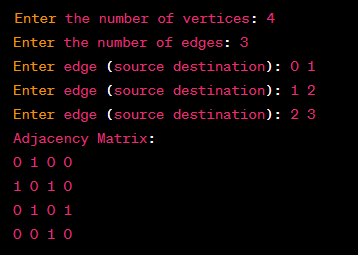
// Print the adjacency matrix

printGraph(&graph);

return 0;

}

***Output:***

******

###### **Experiment No: 7.B**

***Aim*:** To implement BFS using Linked List.

***Program:***

#include <stdio.h>

#include <stdlib.h>

struct Node {

int vertex;

struct Node\* next;

};

struct Graph {

int vertices;

struct Node\*\* adjList;

};

struct Node\* createNode(int v) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->vertices = vertices;

graph->adjList = (struct Node\*\*)malloc(vertices \* sizeof(struct Node\*));

for (int i = 0; i< vertices; ++i) {

graph->adjList[i] = NULL;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct Node\* newNode = createNode(dest);

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

// For undirected graph, add edge from dest to src as well

newNode = createNode(src);

newNode->next = graph->adjList[dest];

graph->adjList[dest] = newNode;

}

void bfs(struct Graph\* graph, int startVertex) {

int\* visited = (int\*)malloc(graph->vertices \* sizeof(int));

for (int i = 0; i< graph->vertices; ++i) {

visited[i] = 0;

}

struct Node\* queue = NULL;

visited[startVertex] = 1;

printf("BFS Traversal from vertex %d: ", startVertex);

struct Node\* temp = createNode(startVertex);

enqueue(&queue, temp);

while (!isEmpty(queue)) {

startVertex = dequeue(&queue);

printf("%d ", startVertex);

struct Node\* trav = graph->adjList[startVertex];

while (trav) {

int adjVertex = trav->vertex;

if (visited[adjVertex] == 0) {

visited[adjVertex] = 1;

struct Node\* newNode = createNode(adjVertex);

enqueue(&queue, newNode);

}

trav = trav->next;

}

}

printf("\n");

}

void enqueue(struct Node\*\* head, struct Node\* newNode) {

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

int dequeue(struct Node\*\* head) {

int val = (\*head)->vertex;

\*head = (\*head)->next;

return val;

}

int isEmpty(struct Node\* head) {

return (head == NULL);

}

int main() {

int vertices, edges, src, dest;

printf("Enter the number of vertices: ");

scanf("%d", &vertices);

struct Graph\* graph = createGraph(vertices);

printf("Enter the number of edges: ");

scanf("%d", &edges);

for (int i = 0; i< edges; ++i) {

printf("Enter edge (source destination): ");

scanf("%d %d", &src, &dest);

addEdge(graph, src, dest);

}

int startVertex;

printf("Enter the starting vertex for BFS: ");

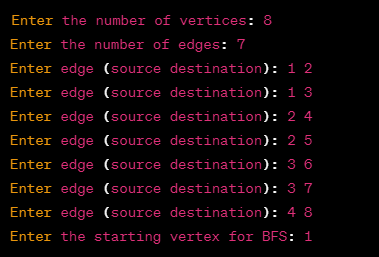
scanf("%d", &startVertex);

bfs(graph, startVertex);

return 0;

}

***Output:***

******

******

###### **Experiment No: 7.C**

***Aim*:** To implement any minimum spanning tree algorithm.

***Program:***

#include <stdio.h>

#include <stdbool.h>

#define V 5 // Number of vertices in the graph

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (!mstSet[v] && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[V][V]) {

printf("Edge Weight\n");

for (int i = 1; i< V; i++) {

printf("%d - %d %d\n", parent[i], i, graph[i][parent[i]]);

}

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool mstSet[V];

for (int i = 0; i< V; i++) {

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++) {

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, graph);

}

int main() {

int graph[V][V] = {{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}};

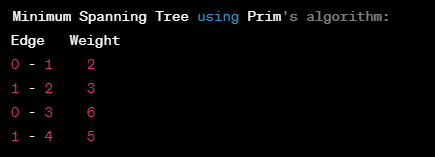
printf("Minimum Spanning Tree using Prim's algorithm:\n");

primMST(graph);

return 0;

}

***Output:***

******

**Content Beyond Syllabus**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **List of Experiments** | **CO** |
| **1** | To implement Count Sort. | CO1 |

###### **Experiment No: 1**

***Aim*:** To implement count sort.

***Program:***

#include <stdio.h>

#include <stdlib.h>

void countingSort(int arr[], int n) {

int max = arr[0];

int min = arr[0];

// Find the maximum and minimum values in the array

for (int i = 1; i< n; i++) {

if (arr[i] > max) {

max = arr[i];

}

if (arr[i] < min) {

min = arr[i];

}

}

int range = max - min + 1;

// Create a count array to store the count of each element

int\* count = (int\*)malloc(range \* sizeof(int));

int\* output = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i< range; i++) {

count[i] = 0;

}

// Count the occurrences of each element

for (int i = 0; i< n; i++) {

count[arr[i] - min]++;

}

// Calculate the cumulative count

for (int i = 1; i< range; i++) {

count[i] += count[i - 1];

}

// Build the sorted output array

for (int i = n - 1; i>= 0; i--) {

output[count[arr[i] - min] - 1] = arr[i];

count[arr[i] - min]--;

}

// Copy the sorted elements back to the original array

for (int i = 0; i< n; i++) {

arr[i] = output[i];

}

free(count);

free(output);

}

int main() {

int arr[] = {4, 2, 2, 8, 3, 3, 1};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original Array: ");

for (int i = 0; i< n; i++) {

printf("%d ", arr[i]);

}

countingSort(arr, n);

printf("\nSorted Array: ");

for (int i = 0; i< n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

***Output:***