Program 1:

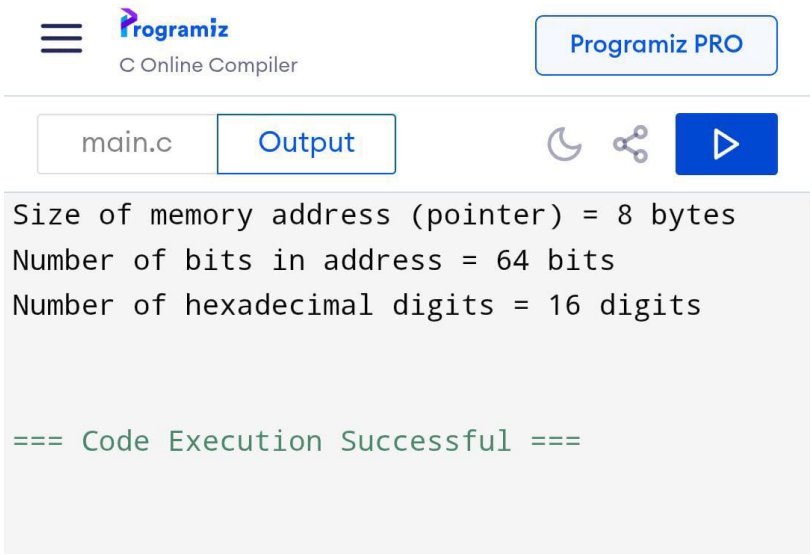
INITIALIZATION OF MALLOC() FUNCTION INTO 0

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   int *arr, i, n = 5;
   arr = (int *)malloc(n * sizeof(int));
   memset(arr, 0, n * sizeof(int));
   printf("Array elements are:\n");
   for(i = 0; i < n; i++)
   {
      printf("%d ", arr[i]);
   }
   free(arr);

   return 0;
}
```

Program 2:

PRINTING THE ADDRESS OF THE POINTER

```
#include <stdio.h>
int main()
{
   int *p;
   int bytes, bits, digits;
   bytes = sizeof(p);
   bits = bytes * 8;
   digits = bits / 4;
  printf("Size of memory address (pointer) = %d bytes\n", bytes);
   printf("Number of bits in address = %d bits\n", bits);
   printf("Number of hexadecimal digits = %d digits\n", digits);

   return 0;
}
```



Program 3:

SELECTION SORT WITH DYNAMIC MEMORY ALLOCATION

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *arr, n, i, j, min, temp;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   arr = (int *)malloc(n * sizeof(int));
   if (arr == NULL)
   {
      printf("Memory allocation failed");
      return 1;
   }
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
    for (i = 0; i < n - 1; i++)
   {
      min = i;
      for (j = i + 1; j < n; j++)
      {
         if (arr[j] < arr[min])
            min = j;
      }
     if (min != i)
      {
         temp = arr[i];
         arr[i] = arr[min];
```

```
            arr[min] = temp;
        }
    }
    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    free(arr);
    return 0;
}
```

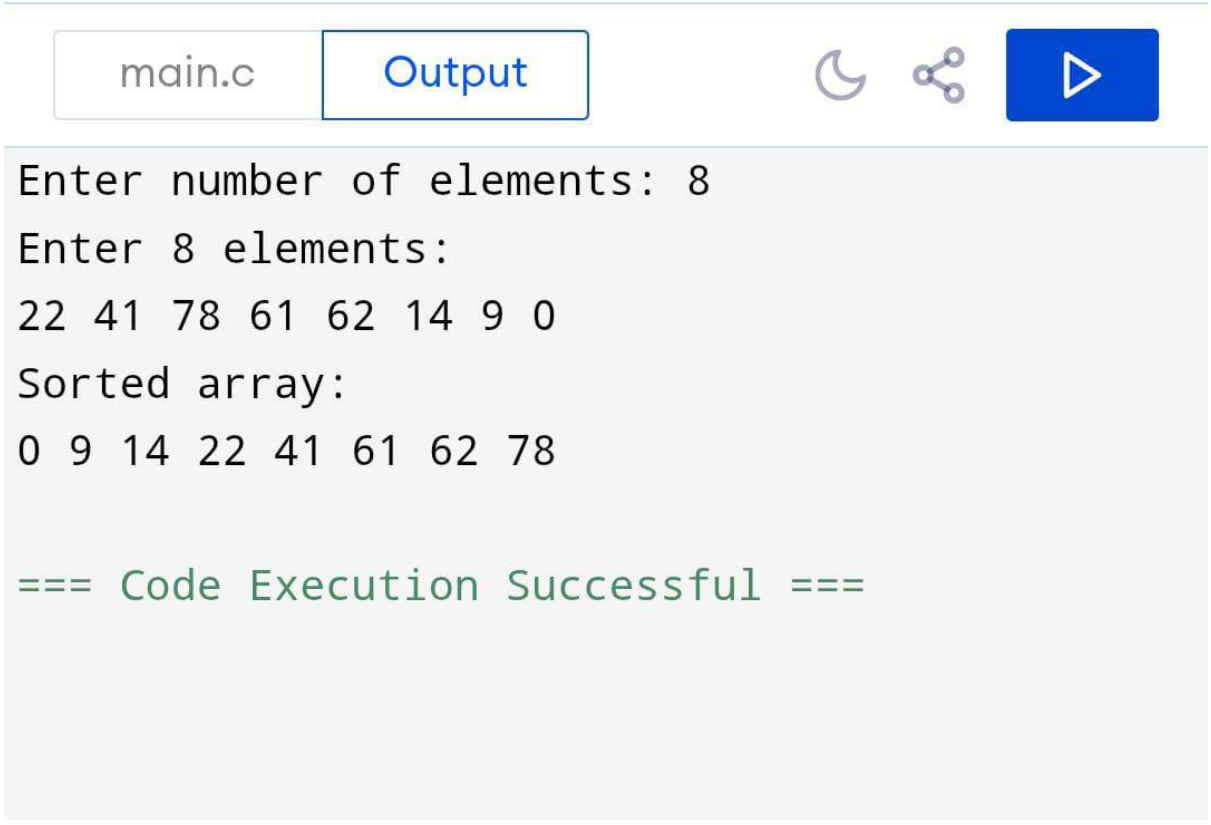

```
Enter number of elements: 5
Enter 5 elements:
22 9 09 08 2006
Sorted array:
8 9 9 22 2006

=== Code Execution Successful ===
```

Program 4:

SELECTION SORT WITHOUT USING DYNAMIC MEMORY ALLOCATION

```
#include <stdio.h>
int main()
{
   int arr[50], n, i, j, min, temp;
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   for (i = 0; i < n - 1; i++)
   {
      min = i;
      for (j = i + 1; j < n; j++)
      {
         if (arr[j] < arr[min])
            min = j;
      }

      if (min != i)
      {
         temp = arr[i];
         arr[i] = arr[min];
         arr[min] = temp;
      }
   }

   printf("Sorted array:\n");
   for (i = 0; i < n; i++)
```

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

    return 0;
}
```



Program 5:

DECLARATION OF 2-DIMENSIONAL ARRAY USING DYNAMIC MEMORY ALLOCATION

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i, j, rows = 3, cols = 4;
   int **arr;
   arr = (int **)malloc(rows * sizeof(int *));
   for (i = 0; i < rows; i++)
      arr[i] = (int *)malloc(cols * sizeof(int));
       for (i = 0; i < rows; i++)
      for (j = 0; j < cols; j++)
         arr[i][j] = i + j;
   for (i = 0; i < rows; i++)
   {
      for (j = 0; j < cols; j++)
         printf("%d ", arr[i][j]);
      printf("\n");
   }
   for (i = 0; i < rows; i++)
      free(arr[i]);
       free(arr);
       return 0;
}
```


main.c
Output
0 1 2 3
1 2 3 4
2 3 4 5
=== Code Execution Successful ===

Program 6:

C PROGRAM FOR PATTERN MATCHING IN GENERAL WAY

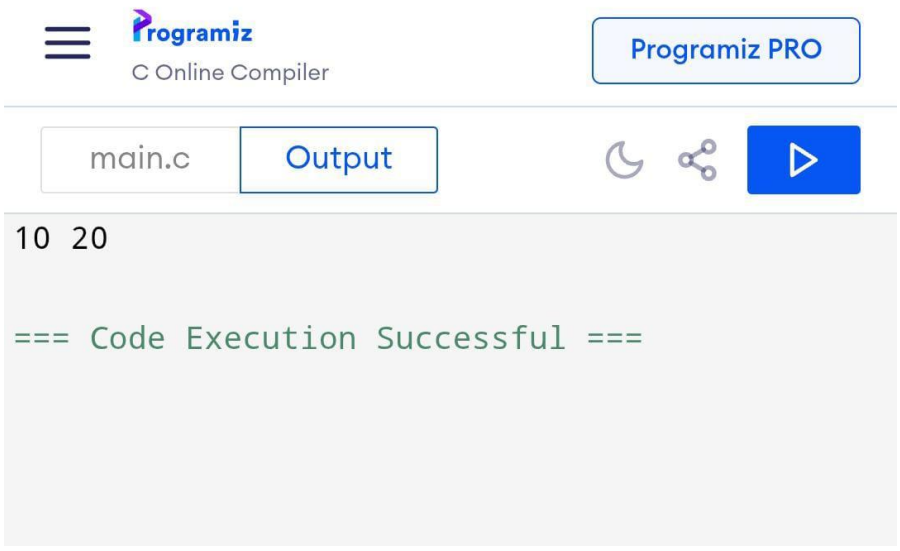
```
#include <stdio.h>
#include <string.h>
int main()
{
   char STR[100], PAT[100];
   int i, j, found = 0;
   printf("Enter the main string: ");
   Scanf("%d", &STR) ;
  printf("Enter the pattern string: ");
   Scanf("%d", &PAT) ;
 for (i = 0; i <= strlen(STR) - strlen(PAT); i++)
  {
     for (j = 0; j < strlen(PAT); j++)
     {
        if (STR[i + j] != PAT[j])
           break;
     }
    if (j == strlen(PAT))
     {
        found = 1;
        printf("Pattern found at position %d\n", i + 1);
        break;
     }
  }
  if (!found)
     printf("Pattern not found\n");
   return 0;
}
```

```
Enter the main string: parupriya
Enter the pattern string: Pattern found at
    position 1


=== Code Execution Successful ===
```

Program 7:

SELF REFERENTIAL STRUCTURE

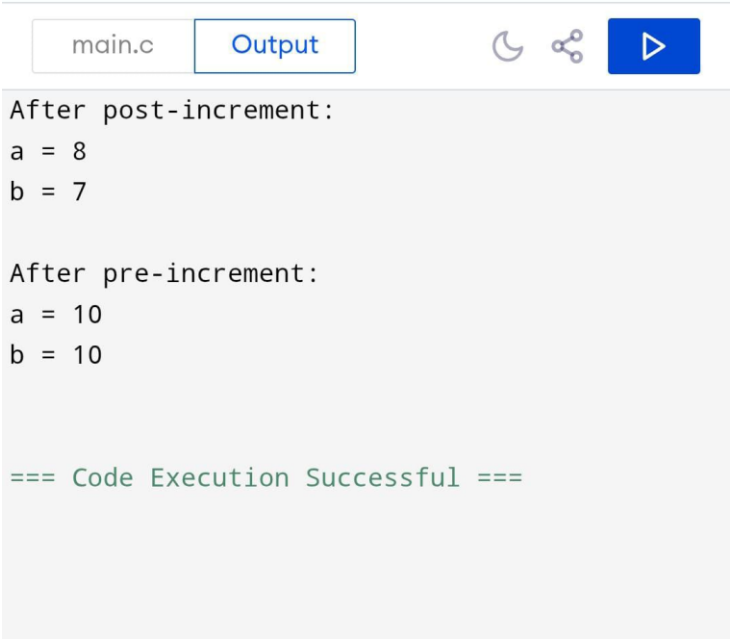
```
#include <stdio.h>
struct node
{
 int data;
struct node *next;
};
int main()
{
   struct node n1, n2;
    n1.data = 10;
   n1.next = &n2;
    n2.data = 20;
   n2.next = NULL
   printf("%d %d", n1.data, n2.data);
  return 0;
}
```



Program 8:

POST INCREEMENT AND PRE INCREEMENT

```
#include <stdio.h>
int main() {
   int a = 5, b;
   b = a++;
   printf("After post-increment:\n");
   printf("a = %d\n", a);
   printf("b = %d\n\n", b);
   a = 5;
   b = ++a;
   printf("After pre-increment:\n");
   printf("a = %d\n", a);
   printf("b = %d\n", b);
   return 0;
}
```



Program 9:

POST DECREEMENT AND PRE DECREEMENT

```
#include <stdio.h>
 int main() {
 int x = 10;
 printf("Pre-decrement (--x): %d\n", --x);
 printf("Value of x after pre-decrement: %d\n", x);
 x = 10;
 printf("Post-decrement (x--): %d\n", x--);
 printf("Value of x after post-decrement: %d\n", x);
 return 0;
}

}
```


```
Pre-decrement (--x): 5
Value of x after pre-decrement: 5
Post-decrement (x--): 4
Value of x after post-decrement: 3


=== Code Execution Successful ===
```

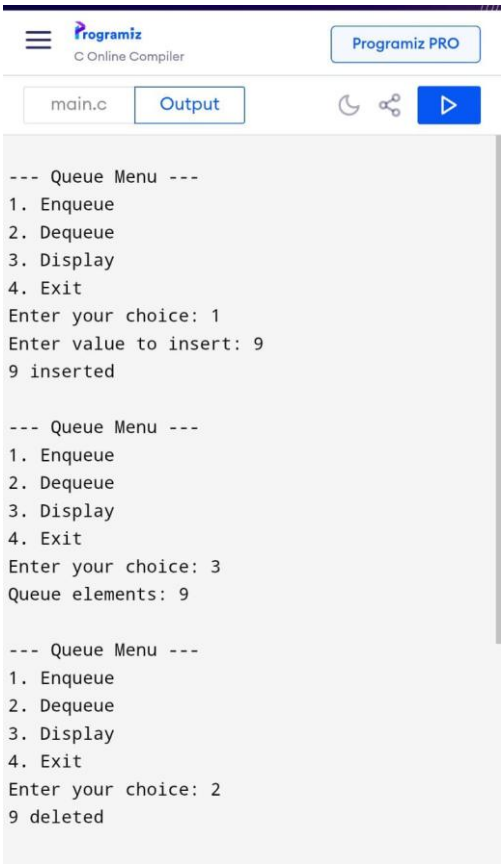
Program 10:

C PROGRAM FOR REGULAR QUEUE

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int value) {
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
   } else {
      if (front == -1)
         front = 0;
      rear++;
      queue[rear] = value;
      printf("%d inserted\n", value);
   }
}
void dequeue() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow\n");
   } else {
      printf("%d deleted\n", queue[front]);
      front++;
   }
}
void display() {
   if (front == -1 || front > rear) {
      printf("Queue is empty\n");
   } else {
      printf("Queue elements: ");
```

```
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    int choice, value;
    do {
        printf("\n--- Queue Menu ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
     switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            enqueue(value);
            break;
     case 2:
            dequeue();
            break;
    case 3:
            display();
            break;
  case 4:
            printf("Exiting program\n");
            break;
```

```
  default:  printf("Invalid choice\n") }
   }
 while (choice != 4);
 return 0;
}
```




```
--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
9 deleted

--- Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program


=== Code Execution Successful ===
```

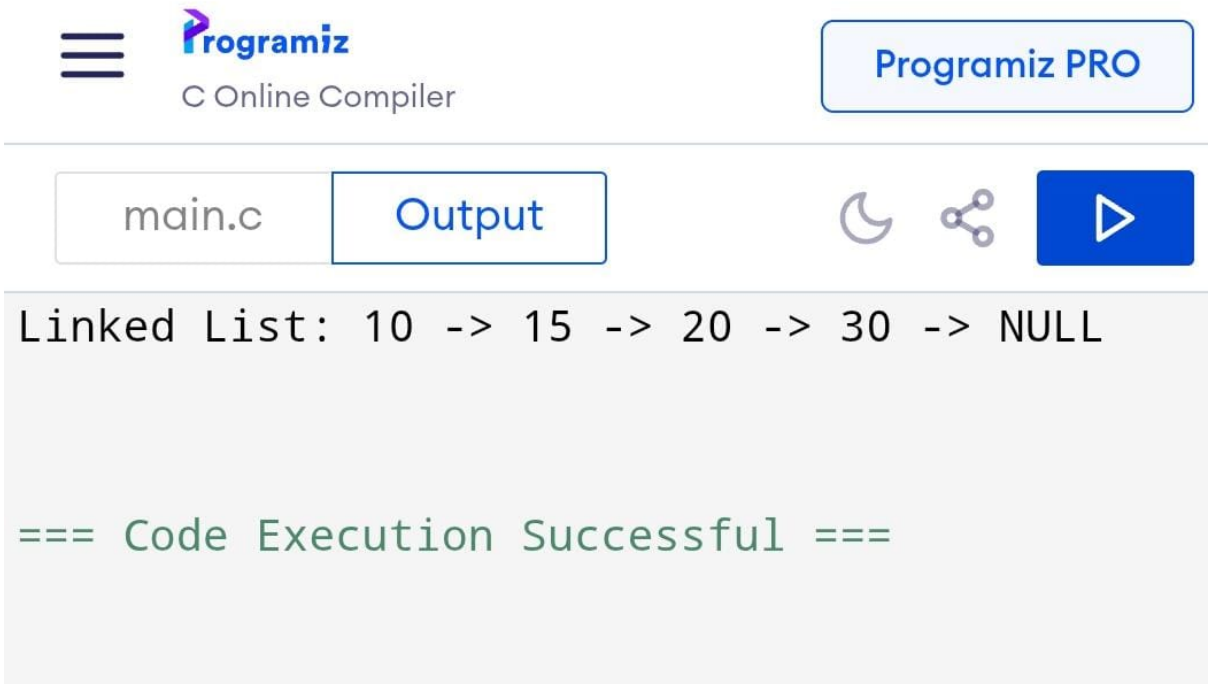
Program 11A:

C PROGRAM TO PERFORM INSERTION USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *next;
};
struct Node *head = NULL;
void insertBeginning(int value) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;
}
void insertEnd(int value) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *temp = head;
   newNode->data = value;
   newNode->next = NULL;
if (head == NULL) {
      head = newNode;
      return;
   }
  while (temp->next != NULL)
      temp = temp->next;
 temp->next = newNode;
}
void insertAtPosition(int value, int position) {
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
```

```
    struct Node *temp = head;
    int I;
 newNode->data = value;
  if (position == 1) {
      newNode->next = head;
      head = newNode;
      return;
    }
  for (i = 1; i < position - 1 && temp != NULL; i++)
      temp = temp->next;
  if (temp == NULL) {
      printf("Position out of range\n");
      return;
    }
     newNode->next = temp->next;
    temp->next = newNode;
}
void display() {
    struct Node *temp = head;
    while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insertBeginning(10);
    insertEnd(20);
    insertEnd(30);
    insertAtPosition(15, 2);
   printf("Linked List: ");
```

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



Program 11B:

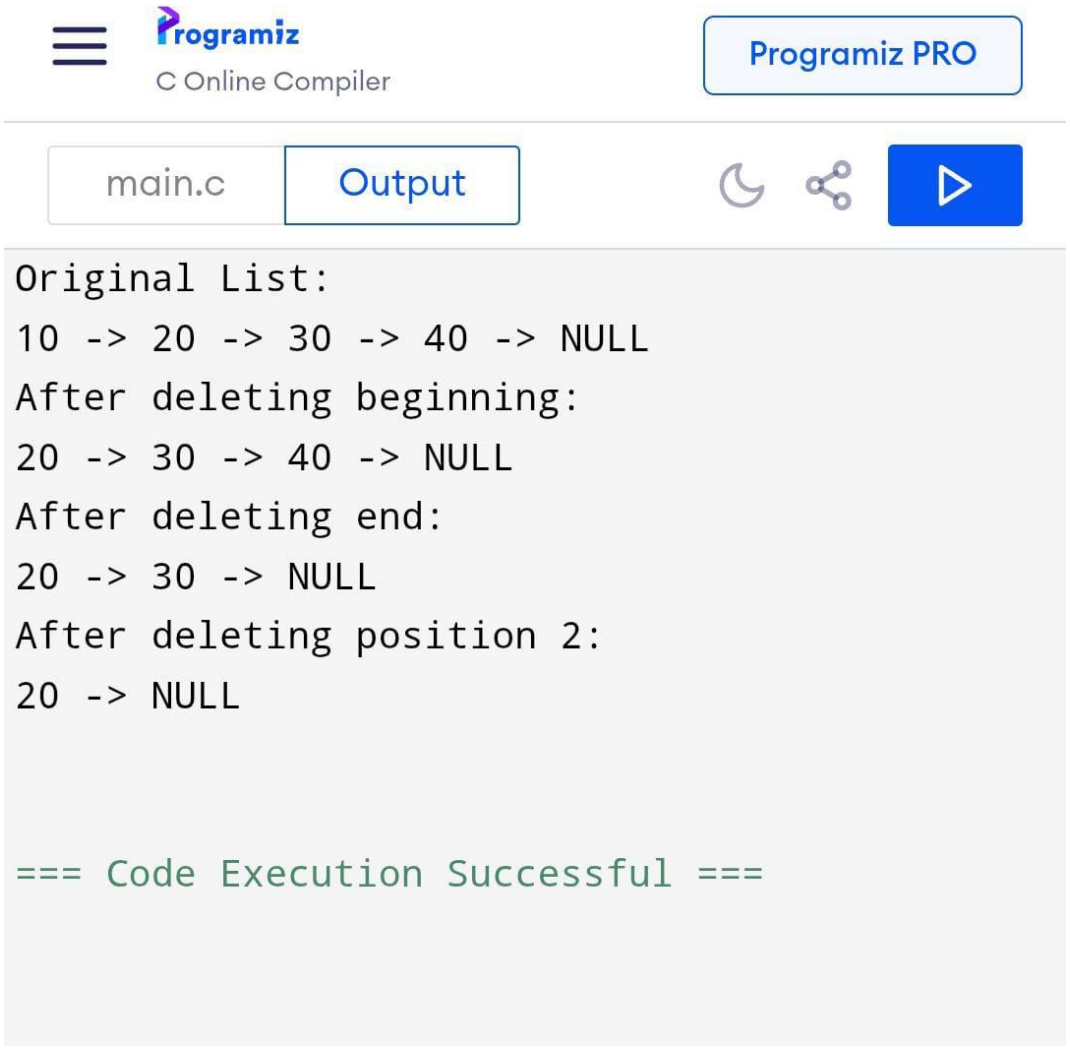
C PROGRAM TO PERFORM DELETION USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *next;
};
struct Node *head = NULL;
void insert(int value) {
   struct Node *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;
   }
}
void deleteBegin() {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   struct Node *temp = head;
   head = head->next;
   free(temp);
```

```
}
void deleteEnd() {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
 if (head->next == NULL) {
      free(head);
      head = NULL;
      return;
   }
   struct Node *temp = head;
   while (temp->next->next != NULL)
      temp = temp->next;
   free(temp->next);
   temp->next = NULL;
}
void deletePosition(int pos) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
 if (pos == 1) {
      deleteBegin();
      return;
   }
  struct Node *temp = head;
   for (int i = 1; i < pos - 1 && temp->next != NULL; i++)
      temp = temp->next;
 if (temp->next == NULL) {
      printf("Invalid position\n");
```

```
        return;
    }
  struct Node *del = temp->next;
    temp->next = del->next;
    free(del);
}
void display() {
    struct Node *temp = head;
    if (temp == NULL) {
        printf("List is empty\n");
        return;
    }
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
int main() {
    insert(10);
    insert(20);
    insert(30);
    insert(40);
 printf("Original List:\n");
    display();
 deleteBegin();
    printf("After deleting beginning:\n");
    display();
 deleteEnd();
    printf("After deleting end:\n");
    display();
```

```
 deletePosition(2);
 printf("After deleting position 2:\n");
   display();
   return 0;
}
```



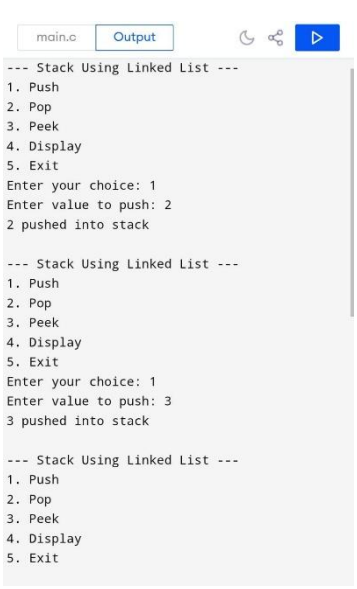
Program 12:

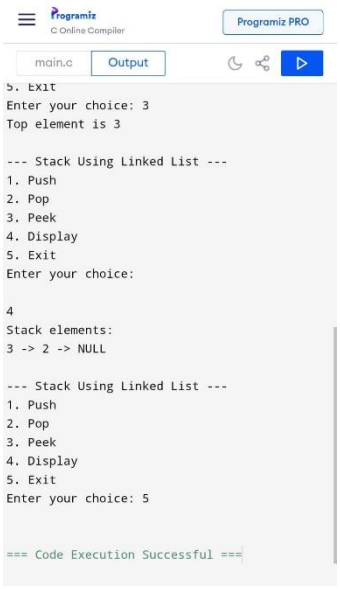
LINKED LIST USING STACKS

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *next;
};
struct Node *top = NULL;
void push(int value) {
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
 if (newNode == NULL) {
      printf("Stack Overflow\n");
      return;
   }
  newNode->data = value;
   newNode->next = top;
   top = newNode;
 printf("%d pushed into stack\n", value);
}
void pop() {
   if (top == NULL) {
      printf("Stack Underflow\n");
      return;
   }
  struct Node *temp = top;
   printf("%d popped from stack\n", top->data);
   top = top->next;
   free(temp);
```

```
}
void peek() {
   if (top == NULL) {
      printf("Stack is empty\n");
   } else {
      printf("Top element is %d\n", top->data);
   }
}
void display() {
   struct Node *temp = top;
 if (top == NULL) {
      printf("Stack is empty\n");
      return;
   }
  printf("Stack elements:\n");
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}
int main() {
   int choice, value;
while (1) {
      printf("\n--- Stack Using Linked List ---\n");
      printf("1. Push\n2. Pop\n3. Peek\n4. Display\n5. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);
  switch (choice) {
         case 1:
            printf("Enter value to push: ");
```

```
            scanf("%d", &value);
            push(value);
            break;
 case 2:
            pop();
            break;
 case 3:
            peek();
            break;
  case 4:
            display();
            break;
case 5:
            exit(0);
 default: printf("Invalid choice\n");
      }
   }
 return 0;
}
```





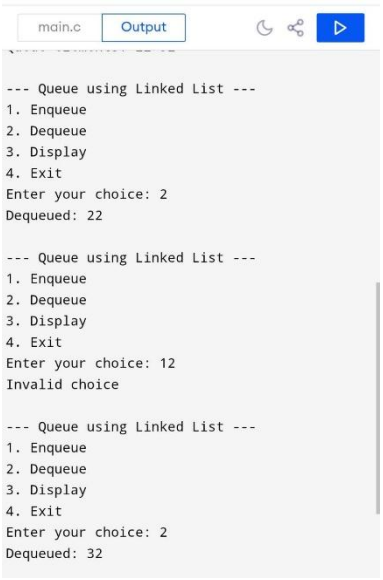
Program 13:

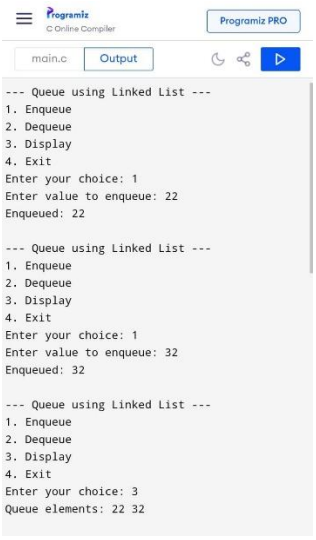
LINKED LIST USING QUEUES

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *next;
};
struct Node *front = NULL;
struct Node *rear = NULL;
void enqueue(int value) {
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
 if (rear == NULL) {
      front = rear = newNode;
   } else {
      rear->next = newNode;
      rear = newNode;
   }
   printf("Enqueued: %d\n", value);
}
void dequeue() {
   struct Node *temp;
 if (front == NULL) {
      printf("Queue is Empty\n");
      return;
   }
 temp = front;
```

```
    printf("Dequeued: %d\n", temp->data);
    front = front->next:
 if (front == NULL) {
        rear = NULL;
    }
    free(temp);
}
void display() {
    struct Node *temp;
 if (front == NULL) {
        printf("Queue is Empty\n");
        return;
    }
 temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    int choice, value;
 while (1) {
        printf("\n--- Queue using Linked List ---\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
        case 1:
            printf("Enter value to enqueue: ");
            scanf("%d", &value);
            enqueue(value);
            break;
 case 2:
            dequeue();
            break;
case 3:
            display();
            break;
case 4:
            exit(0);
default:
            printf("Invalid choice\n");
      }
   }
   return 0;
)
```





Program 14:

TIME TAKEN FOR ARRAY AND LINKED LIST DURING INSERTION AND DELETION

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 10000
struct Node {
   int data;
   struct Node* next;
};
struct Node* insertLL(struct Node* head, int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   return newNode;
}
struct Node* deleteLL(struct Node* head) {
   if (head == NULL)
      return NULL;
   struct Node* temp = head;
   head = head->next;
   free(temp);
   return head;
}
int main() {
   int arr[SIZE];
   int n = SIZE;
   clock_t start, end;
   double time_array, time_ll;
```

```
    for (int i = 0; i < n; i++)
        arr[i] = i;
    start = clock();
    for (int i = n; i > 0; i--)
        arr[i] = arr[i - 1];
    arr[0] = 999;
    end = clock();
   time_array = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Array Insertion Time: %f seconds\n", time_array);
    start = clock();
    for (int i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];
    end = clock();
   time_array = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Array Deletion Time: %f seconds\n", time_array);
    struct Node* head = NULL;
    for (int i = 0; i < n; i++)
        head = insertLL(head, i);
    start = clock();
    head = insertLL(head, 999);
    end = clock();
   time_ll = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Linked List Insertion Time: %f seconds\n", time_ll);
    start = clock();
    head = deleteLL(head);
    end = clock();
   time_ll = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("Linked List Deletion Time: %f seconds\n", time_ll);
  }
}
}
```

main.c Output

```
Array Insertion Time: 0.000012 seconds
Array Deletion Time: 0.000021 seconds
Linked List Insertion Time: 0.000007 seconds
Linked List Deletion Time: 0.000003 seconds


=== Code Execution Successful ===
```

Program 15:

C PROGRAM FOR REPRESTATION OF SPARSE MATRIX

```
#include <stdio.h>
int main() {
   int matrix[10][10], sparse[20][3];
   int rows, cols;
   int i, j, k = 1;
   printf("Enter number of rows and columns: ");
   scanf("%d %d", &rows, &cols);
   printf("Enter elements of matrix:\n");
   for (i = 0; i < rows; i++) {
      for (j = 0; j < cols; j++) {
         scanf("%d", &matrix[i][j]);
         if (matrix[i][j] != 0) {
            sparse[k][0] = i;
            sparse[k][1] = j;
            sparse[k][2] = matrix[i][j];
            k++;
         }
      }
   }
   sparse[0][0] = rows;
   sparse[0][1] = cols;
   sparse[0][2] = k - 1;
   printf("\nSparse Matrix (3-Tuple Form):\n");
   printf("Row Column Value\n");
   for (i = 0; i < k; i++) {
      printf("%d\t%d\t%d\n", sparse[i][0], sparse[i][1], sparse[i][2]);
   }
```

```
    return 0;
}
```

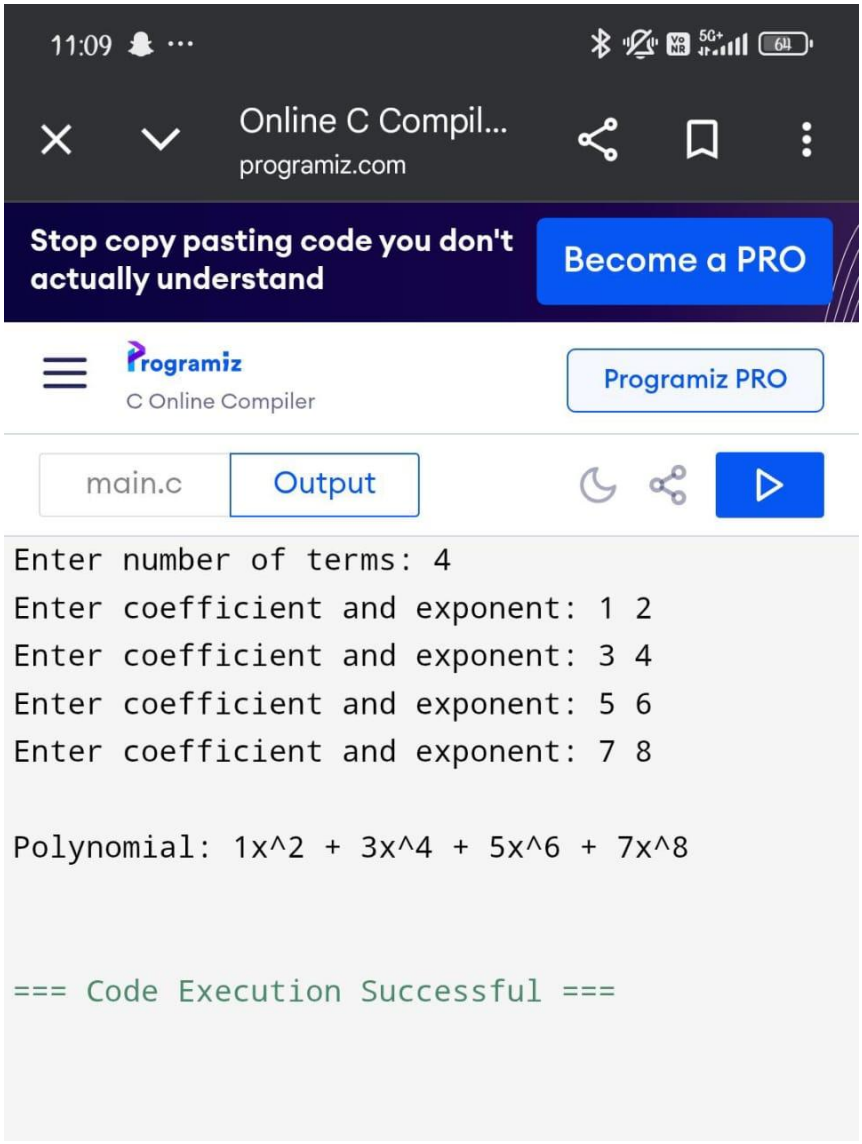

```
Enter number of rows and columns: 3 3
Enter elements of matrix:
8 1 0 0 0 1 0 0 3

Sparse Matrix (3-Tuple Form):
Row Column Value
3   3   4
0   0   8
0   1   1
1   2   1
2   2   3


=== Code Execution Successful ===
```

Program 16:

C PROGRAM FOR POLYNOMIAL REPRESENTATION

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int coeff;
   int exp;
   struct node *next;
};
struct node* createNode(int c, int e) {
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->coeff = c;
   newNode->exp = e;
   newNode->next = NULL;
   return newNode;
}
struct node* insertTerm(struct node *head, int c, int e) {
   struct node *newNode = createNode(c, e);
 if (head == NULL) {
      return newNode;
   }
 struct node *temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
   return head;
}
void displayPolynomial(struct node *head) {
   struct node *temp = head;
```

```
    while (temp != NULL) {
        printf("%dx^%d", temp->coeff, temp->exp);
        if (temp->next != NULL)
            printf(" + ");
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    struct node *poly = NULL;
    int n, coeff, exp;
  printf("Enter number of terms: ");
    scanf("%d", &n);
 for (int i = 0; i < n; i++) {
        printf("Enter coefficient and exponent: ");
        scanf("%d %d", &coeff, &exp);
        poly = insertTerm(poly, coeff, exp);
    }
    printf("\nPolynomial: ");
    displayPolynomial(poly);
    return 0;
}
```


11:09
Online C Compil...
programiz.com
Stop copy pasting code you don't actually understand
Become a PRO
Programiz
C Online Compiler
Programiz PRO
main.c
Output
Enter number of terms: 4
Enter coefficient and exponent: 1 2
Enter coefficient and exponent: 3 4
Enter coefficient and exponent: 5 6
Enter coefficient and exponent: 7 8
Polynomial: 1x^2 + 3x^4 + 5x^6 + 7x^8
=== Code Execution Successful ===

Program 17:

C PROGRAM TO CREATE A TREE

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *left;
   struct Node *right;
};
struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
void inorder(struct Node* root) {
   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->data);
      inorder(root->right);
   }
}
int main() {
   struct Node* root = createNode(1);
   root->left = createNode(2);
   root->right = createNode(3);
   root->left->left = createNode(4);
   root->left->right = createNode(5);
  printf("Inorder Traversal of Binary Tree:\n");
```

```
    inorder(root);
 return 0;
}
```


```
Inorder Traversal of Binary Tree:
2 2 7 1 6

=== Code Execution Successful ===
```

Program 18:

C PROGRAM TO CONSTRUCT A TREE USING A ARRAY

```
#include <stdio.h>
#define MAX 50
int tree[MAX];
void insert(int value, int index)
{
   if (index >= MAX)
   {
      printf("Tree is full, cannot insert %d\n", value);
      return;
   }
   tree[index] = value;
}
void display()
{
   printf("\nTree elements (Array Representation):\n");
   for (int i = 1; i < MAX; i++)
   {
      if (tree[i] != -1)
         printf("Index %d -> %d\n", i, tree[i]);
   }
}
int main()
{
   int i;
   for (i = 0; i < MAX; i++)
      tree[i] = -1;
   insert(10, 1);
   insert(20, 2);
```

```
    insert(30, 3);
    insert(40, 4);
    insert(50, 5);
    insert(60, 6);
    insert(70, 7);
  display();
    return 0;
}
```

main.c | Output

```
Tree elements (Array Representation):
Index 1 -> 100
Index 2 -> 200
Index 3 -> 300
Index 4 -> 400
Index 5 -> 500
Index 6 -> 600
Index 7 -> 700


=== Code Execution Successful ===
```

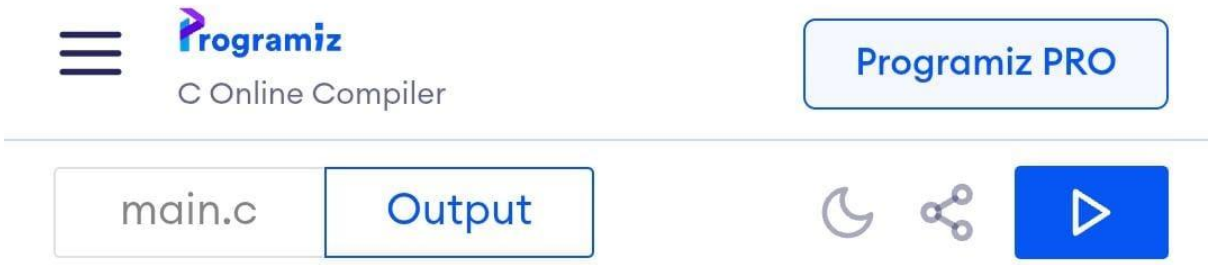
Program 19:

C PROGRAM TO CONSTRCT A BINARY TREE USING QUEUES

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *left, *right;
};
struct Queue {
   int front, rear;
   struct Node *arr[100];
};
struct Node* createNode(int data) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}
void initQueue(struct Queue *q) {
   q->front = q->rear = -1;
}
void enqueue(struct Queue *q, struct Node *node) {
   q->arr[++q->rear] = node;
}
struct Node* dequeue(struct Queue *q) {
   return q->arr[++q->front];
}
struct Node* createTree() {
   int data;
   struct Queue q;
```

```
    initQueue(&q);

    printf("Enter root value (-1 for no node): ");
    scanf("%d", &data);
  if (data == -1)
        return NULL;
  struct Node* root = createNode(data);
    enqueue(&q, root);
   while (q.front != q.rear) {
        struct Node* current = dequeue(&q);
   printf("Enter left child of %d (-1 for no node): ", current->data);
        scanf("%d", &data);
        if (data != -1) {
            current->left = createNode(data);
            enqueue(&q, current->left);
        }
   printf("Enter right child of %d (-1 for no node): ", current->data);
        scanf("%d", &data);
        if (data != -1) {
            current->right = createNode(data);
            enqueue(&q, current->right);
        }
    }
    return root;
}
void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
```

```
}
int main() {
    struct Node* root = createTree();
 printf("\nInorder Traversal of Binary Tree:\n");
    inorder(root);
 return 0;
}
```




```
Enter root value (-1 for no node): 10
Enter left child of 10 (-1 for no node): 20
Enter right child of 10 (-1 for no node): 30
Enter left child of 20 (-1 for no node): -1
Enter right child of 20 (-1 for no node): -1
Enter left child of 30 (-1 for no node): 40
Enter right child of 30 (-1 for no node): -1
Enter left child of 40 (-1 for no node): -1
Enter right child of 40 (-1 for no node): -1

Inorder Traversal of Binary Tree:
20 10 40 30

=== Code Execution Successful ===
```

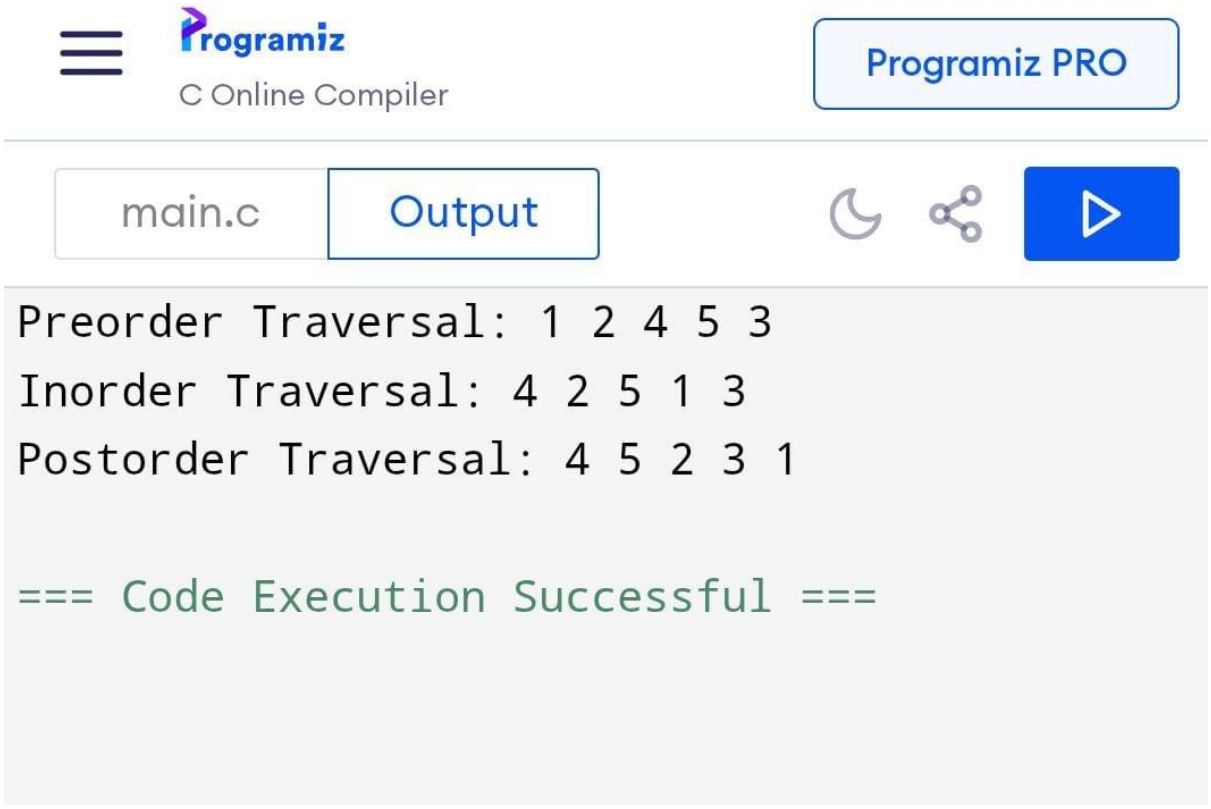
Program 20:

C PROGRAM FOR BINARY TREE TRAVERSAL

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

```
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(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;
  root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    printf("Preorder Traversal: ");
    preorder(root);
    printf("\nInorder Traversal: ");
    inorder(root);
    printf("\nPostorder Traversal: ");
    postorder(root);
    return 0;
```

```
}
```


Programiz
C Online Compiler
Programiz PRO
main.c
Output
Preorder Traversal: 1 2 4 5 3
Inorder Traversal: 4 2 5 1 3
Postorder Traversal: 4 5 2 3 1
=== Code Execution Successful ===

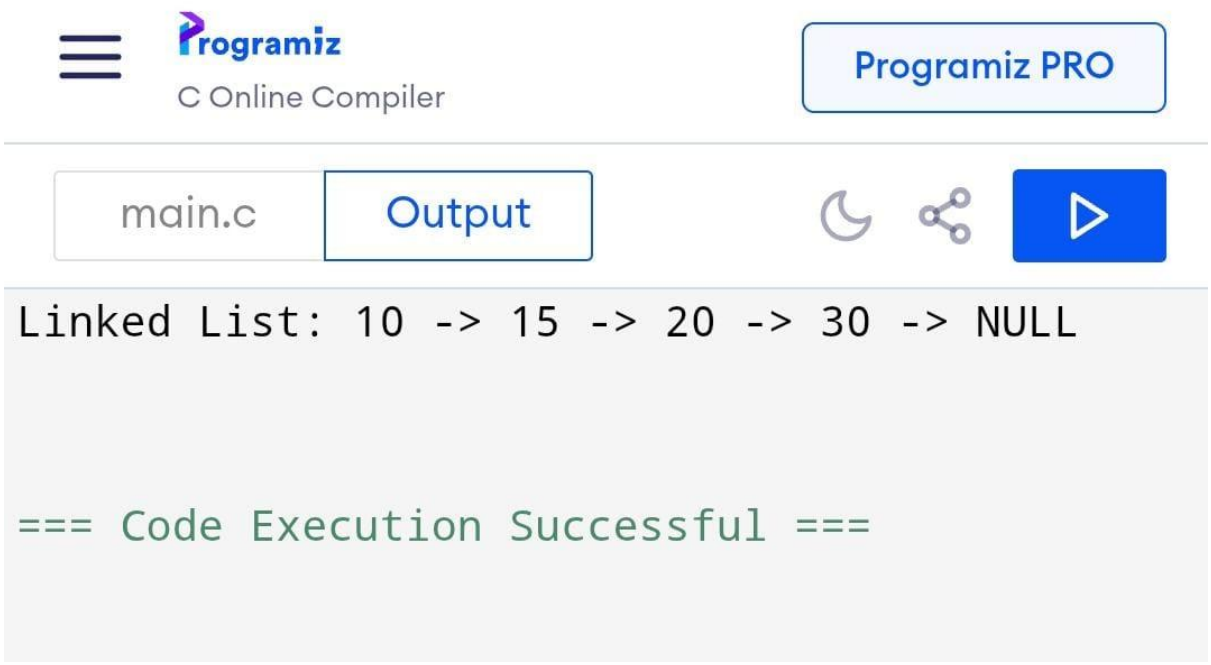
Program 21:

C PROGRAM TO INSERT NODE IN SINGLY LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *next;
};
struct Node *head = NULL;
void insert_begin(int value) {
   struct Node *newNode;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = head;
   head = newNode;
}
void insert_end(int value) {
   struct Node *newNode, *temp;
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->next = NULL;
 if (head == NULL) {
      head = newNode;
      return;
   }
   temp = head;
   while (temp->next != NULL) {
      temp = temp->next;
   }
   temp->next = newNode;
```

```
}
void insert_position(int value, int position) {
   struct Node *newNode, *temp;
   int i;
 if (position == 1) {
      insert_begin(value);
      return;
   }
   newNode = (struct Node *)malloc(sizeof(struct Node));
   newNode->data = value;
   temp = head;
   for (i = 1; i < position - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position not valid\n");
      return;
   }
   newNode->next = temp->next;
   temp->next = newNode;
}
void display() {
   struct Node *temp = head;
   if (temp == NULL) {
      printf("Linked list is empty\n");
      return;
   }
  printf("Linked List: ");
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
```

```
    }
    printf("NULL\n");
}
int main() {
    insert_begin(10);
    insert_end(20);
    insert_end(30);
    insert_position(15, 2);
  display();
    return 0;
}
```



Program 22:

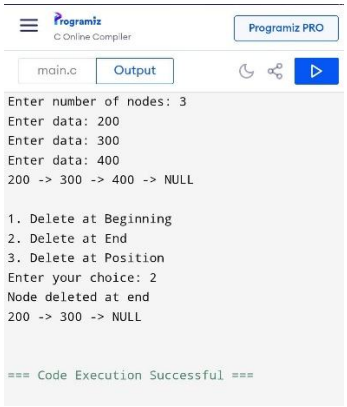
C PROGRAM TO DELETE NODE IN SINGLY LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
void create() {
   int n, i;
   struct node *temp, *newnode;
    printf("Enter number of nodes: ");
   scanf("%d", &n);
  for (i = 0; i < n; i++) {
      newnode = (struct node *)malloc(sizeof(struct node));
      printf("Enter data: ");
      scanf("%d", &newnode->data);
      newnode->next = NULL;
 if (head == NULL) {
         head = newnode;
         temp = head;
      } else {
         temp->next = newnode;
         temp = newnode;
      }
   }
}
void delete_begin() {
   struct node *temp;
```

```
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        temp = head;
        head = head->next;
        free(temp);
        printf("Node deleted at beginning\n");
    }
}
void delete_end() {
    struct node *temp, *prev;
    if (head == NULL) {
        printf("List is empty\n");
    } else if (head->next == NULL) {
        free(head);
        head = NULL;
        printf("Last node deleted\n");
    } else {
        temp = head;
        while (temp->next != NULL) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = NULL;
        free(temp);
        printf("Node deleted at end\n");
    }
}
void delete_pos() {
    int pos, i;
    struct node *temp, *prev;
```

```
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
   printf("Enter position to delete: ");
    scanf("%d", &pos);
  if (pos == 1) {
        delete_begin();
        return;
    }
  temp = head;
    for (i = 1; i < pos; i++) {
        prev = temp;
        temp = temp->next;
        if (temp == NULL) {
            printf("Invalid position\n");
            return;
        }
    }
   prev->next = temp->next;
    free(temp);
    printf("Node deleted at position %d\n", pos);
}
void display() {
    struct node *temp = head;
    if (head == NULL) {
        printf("List is empty\n");
    } else {
        while (temp != NULL) {
            printf("%d -> ", temp->data);
```

```
        temp = temp->next;
      }
      printf("NULL\n");
   }
}
int main() {
   int choice;
  create();
   display();
 printf("\n1. Delete at Beginning\n2. Delete at End\n3. Delete at Position\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);
 switch (choice) {
      case 1: delete_begin(); break;
      case 2: delete_end(); break;
      case 3: delete_pos(); break;
      default: printf("Invalid choice\n");
   }
display();
return 0;
}
```

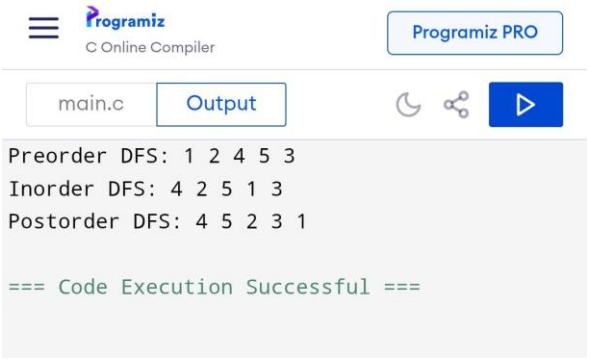


Program 23:

DEPTH FIRST SEARCH

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *left;
   struct node *right;
};
struct node* createNode(int value) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
void preorder(struct node* root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorder(root->left);
      preorder(root->right);
   }
}
* root)
{
   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->data);
      inorder(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 = createNode(1);
   root->left = createNode(2);
   root->right = createNode(3);
   root->left->left = createNode(4);
   root->left->right = createNode(5);
   printf("Preorder DFS: ");
   preorder(root);
  printf("\nInorder DFS: ");
   inorder(root);
  printf("\nPostorder DFS: ");
   postorder(root);
return 0;
}
```



Program 24:

BREADTH FIRST SEARCH

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *left;
   struct Node *right;
};
struct Queue {
   int front, rear;
   int size;
   struct Node **array;
};
struct Node* createNode(int data) {
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;
   return node;
}
struct Queue* createQueue(int size) {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = 0;
   queue->rear = -1;
   queue->size = size;
   queue->array = (struct Node**)malloc(size * sizeof(struct Node*));
   return queue;
}
int isEmpty(struct Queue* queue) {
```

```
    return queue->rear < queue->front;
}
void enqueue(struct Queue* queue, struct Node* node) {
    queue->array[++queue->rear] = node;
}
struct Node* dequeue(struct Queue* queue) {
    return queue->array[queue->front++];
}
void breadthFirstTraversal(struct Node* root) {
    if (root == NULL)
        return;
 struct Queue* queue = createQueue(100);
    enqueue(queue, root);
  printf("Breadth First Traversal: ");
while (!isEmpty(queue)) {
        struct Node* temp = dequeue(queue);
        printf("%d ", temp->data);
if (temp->left != NULL)
            enqueue(queue, temp->left);
 if (temp->right != NULL)
            enqueue(queue, temp->right);
    }
}
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
 breadthFirstTraversal(root);
return 0;
```

```
}
```


Programiz
C Online Compiler
Programiz PRO
main.c
Output
Breadth First Traversal: 1 2 3 4 5
=== Code Execution Successful ===

Program 25:

LEVEL ORDER

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node *left;
   struct Node *right;
};
struct Queue {
   int front, rear;
   int size;
   struct Node **array;
};
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 Queue* createQueue(int size) {
   struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
   queue->front = queue->rear = -1;
   queue->size = size;
   queue->array = (struct Node**)malloc(size * sizeof(struct Node*));
   return queue;
}
int isEmpty(struct Queue* queue) {
   return queue->front == -1;
```

```
}
void enqueue(struct Queue* queue, struct Node* node) {
   if (queue->rear == queue->size - 1)
      return;
  if (queue->front == -1)
      queue->front = 0;
  queue->array[++queue->rear] = node;
}
struct Node* dequeue(struct Queue* queue) {
   if (isEmpty(queue))
      return NULL;
 struct Node* temp = queue->array[queue->front];
 if (queue->front == queue->rear)
      queue->front = queue->rear = -1;
   else
      queue->front++;
 return temp;
}
void levelOrder(struct Node* root) {
   if (root == NULL)
      return;
 struct Queue* queue = createQueue(100);
   enqueue(queue, root);
while (!isEmpty(queue)) {
      struct Node* current = dequeue(queue);
      printf("%d ", current->data);
  if (current->left != NULL)
         enqueue(queue, current->left);
 if (current->right != NULL)
         enqueue(queue, current->right);
   }
```

```
}
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
printf("Level Order Traversal: ");
    levelOrder(root);
return 0;
}
```

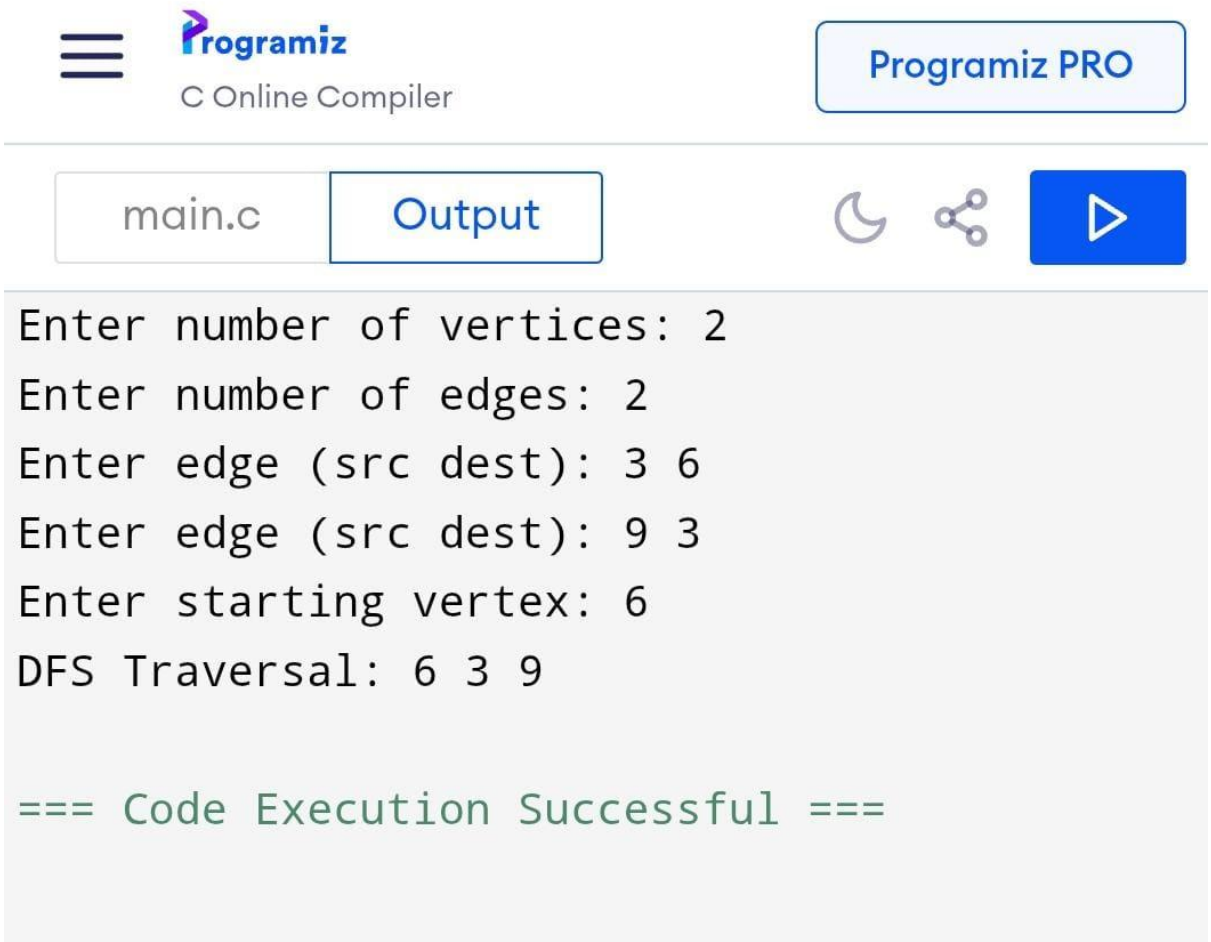
Program 26:

DFS AND BFS USING ADJACENCY LIST (STACK AND QUEUE)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
struct Node {
   int vertex;
   struct Node* next;
};
int stack[MAX];
int top = -1;
int visited[MAX];
struct Node* adjList[MAX];
void push(int v) {
   stack[++top] = v;
}
int pop() {
   return stack[top--];
}
struct Node* createNode(int v) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}
void addEdge(int src, int dest) {
   struct Node* newNode = createNode(dest);
   newNode->next = adjList[src];
   adjList[src] = newNode;
```

```
 newNode = createNode(src);
   newNode->next = adjList[dest];
   adjList[dest] = newNode;
}
void DFS(int start) {
   push(start);
 while (top != -1) {
      int v = pop();
 if (!visited[v]) {
         printf("%d ", v);
         visited[v] = 1;
      }
  struct Node* temp = adjList[v];
      while (temp != NULL) {
         if (!visited[temp->vertex]) {
            push(temp->vertex);
         }
         temp = temp->next;
      }
   }
}
int main() {
   int vertices, edges, src, dest, start;
  printf("Enter number of vertices: ");
   scanf("%d", &vertices);
 for (int i = 0; i < vertices; i++) {
      adjList[i] = NULL;
      visited[i] = 0;
   }
   printf("Enter number of edges: ");
   scanf("%d", &edges);
```

```
    for (int i = 0; i < edges; i++) {
      printf("Enter edge (src dest): ");
      scanf("%d %d", &src, &dest);
      addEdge(src, dest);
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    printf("DFS Traversal: ");
    DFS(start);
  return 0;
}
```

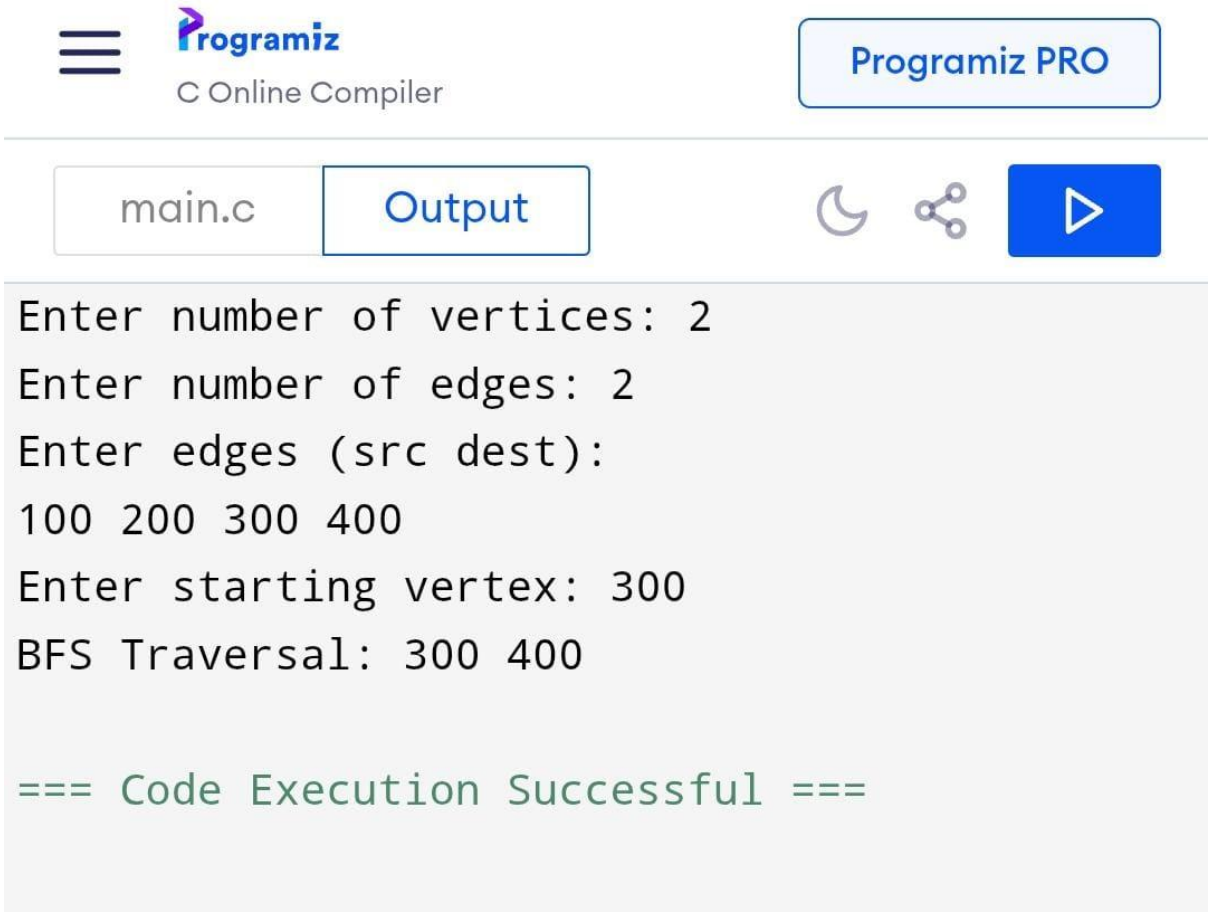


BFS USING ADJACENCY (QUEUE)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
struct Node {
   int vertex;
   struct Node* next;
};
int queue[MAX];
int front = 0, rear = -1;
int visited[MAX];
struct Node* adjList[MAX];
void enqueue(int v) {
   queue[++rear] = v;
}
int dequeue() {
   return queue[front++];
}
struct Node* createNode(int v) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->vertex = v;
   newNode->next = NULL;
   return newNode;
}
void addEdge(int src, int dest) {
   struct Node* newNode = createNode(dest);
   newNode->next = adjList[src];
   adjList[src] = newNode;
}
void BFS(int start) {
```

```
    enqueue(start);
    visited[start] = 1;
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);
        struct Node* temp = adjList[current];
        while (temp != NULL) {
            if (!visited[temp->vertex]) {
                visited[temp->vertex] = 1;
                enqueue(temp->vertex);
            }
            temp = temp->next;
        }
    }
}
int main() {
    int vertices, edges, src, dest, start;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    for (int i = 0; i < vertices; i++) {
        adjList[i] = NULL;
        visited[i] = 0;
    }
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    printf("Enter edges (src dest):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(src, dest);
    }
    printf("Enter starting vertex: ");
```

```
    scanf("%d", &start)
    printf("BFS Traversal: ");
    BFS(start);
    return 0;
}
```



Program 27:

C PROGRAM FOR CALLOC AND MALLOC FUNCTIONS CHECKING IF JUNK/ZERO IS INITIALIZED TO THEM

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int i, n = 5;
   int *m_ptr, *c_ptr;
   m_ptr = (int *)malloc(n * sizeof(int));
   c_ptr = (int *)calloc(n, sizeof(int));
   if (m_ptr == NULL || c_ptr == NULL) {
      printf("Memory allocation failed\n");
      return 1;
   }
    printf("Values after malloc():\n");
   for (i = 0; i < n; i++) {
      printf("%d ", m_ptr[i]);   // Junk values
   }
   printf("\n\nValues after calloc():\n");
   for (i = 0; i < n; i++) {
      printf("%d ", c_ptr[i]);   // Zero initialized
   }
   free(m_ptr);
   free(c_ptr);
   return 0;
}
```


Programiz
C Online Compiler
Programiz PRO
main.c
Output
Values after malloc():
0 0 0 0 0
Values after calloc():
0 0 0 0 0
=== Code Execution Successful ===

Program 28:

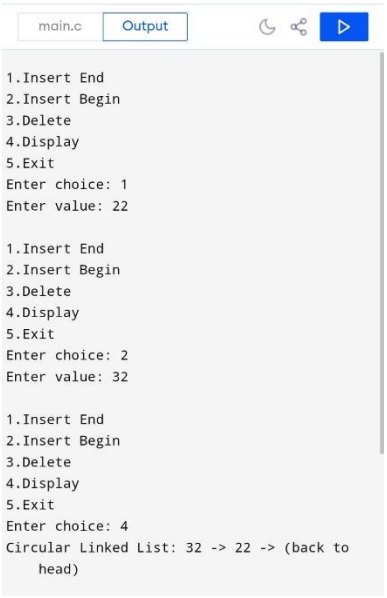
CIRCULAR LINKED LIST BASIC OPERATION

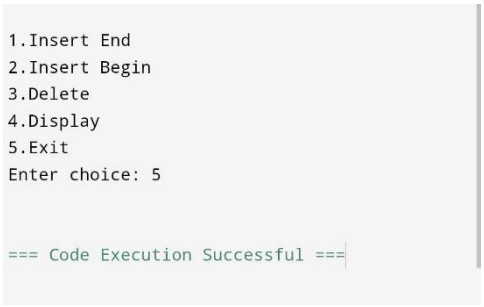
```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *last = NULL;
void insertEnd(int value) {
   struct node *newNode = (struct node *)malloc(sizeof(struct node));
   newNode->data = value;
if (last == NULL) {
      last = newNode;
      last->next = last;
   } else {
      newNode->next = last->next;
      last->next = newNode;
      last = newNode;
   }
}
void insertBegin(int value) {
   struct node *newNode = (struct node *)malloc(sizeof(struct node));
   newNode->data = value;
 if (last == NULL) {
      last = newNode;
      last->next = last;
   } else {
      newNode->next = last->next;
      last->next = newNode;
```

```
    }
}
void deleteNode(int key) {
    struct node *temp, *prev;
    if (last == NULL) {
        printf("List is empty\n");
        return;
    }
temp = last->next;
    prev = last;
 do {
        if (temp->data == key) {
            if (temp == last && temp->next == last) {
                last = NULL;
            } else {
                prev->next = temp->next;
                if (temp == last)
                    last = prev;
            }
            free(temp);
            printf("Node deleted\n");
            return;
        }
        prev = temp;
        temp = temp->next;
    } while (temp != last->next);

    printf("Node not found\n");
}
void display() {
    struct node *temp;
```

```
    if (last == NULL) {
        printf("List is empty\n");
        return;
    }
  temp = last->next;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != last->next);
    printf("(back to head)\n");
}
int main() {
    int choice, value;
   while (1) {
        printf("\n1.Insert End\n2.Insert Begin\n3.Delete\n4.Display\n5.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
switch (choice) {
        case 1:
            printf("Enter value: ");
            scanf("%d", &value);
            insertEnd(value);
            break;
    case 2:
            printf("Enter value: ");
            scanf("%d", &value);
            insertBegin(value);
            break;
  case 3:
            printf("Enter value to delete: ");
```

```
            scanf("%d", &value);
            deleteNode(value);
            break;
        case 4:
            display();
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}
```

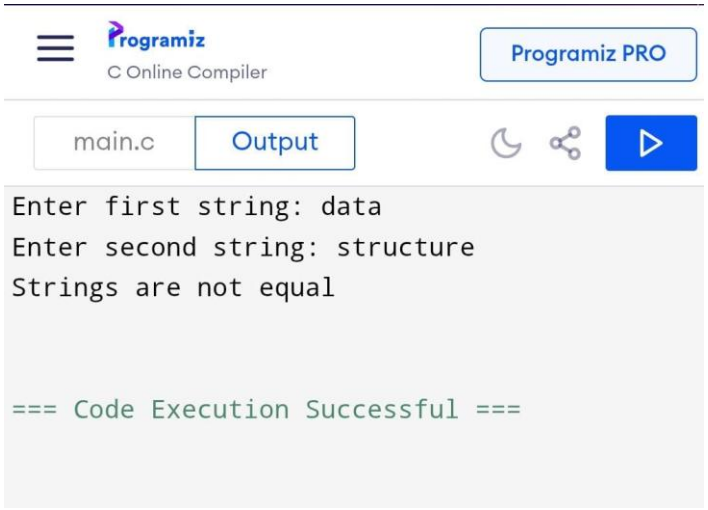




Program 29:

HOW TO IMPLEMENT COMPARISON OF 2 STRINGS USING BUILT IN FUNCTION

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[50], str2[50];
 printf("Enter first string: ");
   gets(str1);
 printf("Enter second string: ");
   gets(str2);
  if (strcmp(str1, str2) == 0)
      printf("Both strings are equal\n");
   else
      printf("Strings are not equal\n");
return 0;
}
```



Program 30:

IN LINKED LIST INSERTION IN THE MIDDLE AND DELETION IN MIDDLE

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
void create(int value) {
   struct node *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;
   }
}
void display() {
   struct node *temp = head;
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   while (temp != NULL) {
      printf("%d -> ", temp->data);
```

```
        temp = temp->next;
    }
    printf("NULL\n");
}
int count() {
    int c = 0;
    struct node *temp = head;
    while (temp != NULL) {
        c++;
        temp = temp->next;
    }
    return c;
}
void insert_middle(int value) {
    int pos, i = 1;
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = value;
  pos = count() / 2 + 1;
struct node *temp = head;
    while (i < pos - 1) {
        temp = temp->next;
        i++;
    }
  newnode->next = temp->next;
    temp->next = newnode;
}
void delete_middle() {
    int pos, i = 1;
    struct node *temp = head, *prev = NULL;
  pos = count() / 2 + 1;
while (i < pos) {
```

```
        prev = temp;
        temp = temp->next;
        i++;
    }
prev->next = temp->next;
    printf("Deleted element: %d\n", temp->data);
    free(temp);
}
int main() {
    create(10);
    create(20);
    create(30);
    create(40);
    create(50);
   printf("Original List:\n");
    display();
   insert_middle(25);
    printf("\nAfter Insertion in Middle:\n");
    display();
  delete_middle();
    printf("\nAfter Deletion from Middle:\n");
display();
return 0;
}
```

main.c | Output

```
Original List:
10 -> 20 -> 30 -> 40 -> 50 -> NULL

After Insertion in Middle:
10 -> 20 -> 25 -> 30 -> 40 -> 50 -> NULL
Deleted element: 30

After Deletion from Middle:
10 -> 20 -> 25 -> 40 -> 50 -> NULL


=== Code Execution Successful ===
```

Program 31:

CIRCULAR QUEUE USING ARRAY (MODULO DIVISION,QUEUE FULL,QUEUE EMPTY)

```
#include <stdio.h>
#define SIZE 5

int cq[SIZE];
int front = -1, rear = -1;
void enqueue(int item)
{
   if ((rear + 1) % SIZE == front)
   {
      printf("Queue is FULL\n");
      return;
   }
 if (front == -1)  // First insertion
      front = 0;
 rear = (rear + 1) % SIZE;
   cq[rear] = item;
   printf("Inserted: %d\n", item);
}
void dequeue()
{
   if (front == -1)
   {
      printf("Queue is EMPTY\n");
      return;
   }
  printf("Deleted: %d\n", cq[front]);
 if (front == rear)  // Only one element
   {
```

```
        front = rear = -1;
    }
    else
    {
        front = (front + 1) % SIZE;
    }
}
void display()
{
    int i;
   if (front == -1)
    {
        printf("Queue is EMPTY\n");
        return;
    }

    printf("Queue elements: ");
    i = front;
    while (i != rear)
    {
        printf("%d ", cq[i]);
        i = (i + 1) % SIZE;
    }
    printf("%d\n", cq[rear]);
}
int main()
{
    int choice, item;
  while (1)
    {
        printf("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
```

```
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
          printf("Enter element: ");
          scanf("%d", &item);
          enqueue(item);
          break;

        case 2:
          dequeue();
          break;

        case 3:
          display();
          break;

        case 4:
          return 0;

        default:
          printf("Invalid choice\n");
        }
      }
    }
```


```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 3
Queue elements: 8

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter choice: 4


=== Code Execution Successful ===
```

Program 32:

SPARSE MATRIX REPRESENTATION USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int row;
   int col;
   int value;
   struct Node *next;
};
struct Node* createNode(int r, int c, int val) {
   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->row = r;
   newNode->col = c;
   newNode->value = val;
   newNode->next = NULL;
   return newNode;
}
void insertNode(struct Node **head, int r, int c, int val) {
   struct Node *newNode = createNode(r, c, val);
  if (*head == NULL) {
      *head = newNode;
      return;
   }
  struct Node *temp = *head;
   while (temp->next != NULL)
      temp = temp->next;
  temp->next = newNode;
}
void displaySparse(struct Node *head) {
```

```
    if (head == NULL) {
        printf("Sparse Matrix is empty\n");
        return;
    }
    printf("\nRow\tColumn\tValue\n");
    printf("----------------------\n");
    while (head != NULL) {
        printf("%d\t%d\t%d\n", head->row, head->col, head->value);
        head = head->next;
    }
}
void displayNormalMatrix(struct Node *head, int rows, int cols) {
    int matrix[rows][cols];
    int i, j;
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            matrix[i][j] = 0;
    while (head != NULL) {
        matrix[head->row][head->col] = head->value;
        head = head->next;
    }
    printf("\nNormal Matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main() {
    struct Node *head = NULL;
```

```
  int rows, cols, i, j, val;
  printf("Enter number of rows and columns: ");
  scanf("%d %d", &rows, &cols);
 printf("Enter matrix elements:\n");
  for (i = 0; i < rows; i++) {
     for (j = 0; j < cols; j++) {
        scanf("%d", &val);
        if (val != 0) {
           insertNode(&head, i, j, val);
        }
     }
  }
 displaySparse(head);
 displayNormalMatrix(head, rows, cols);
  return 0;
}
```