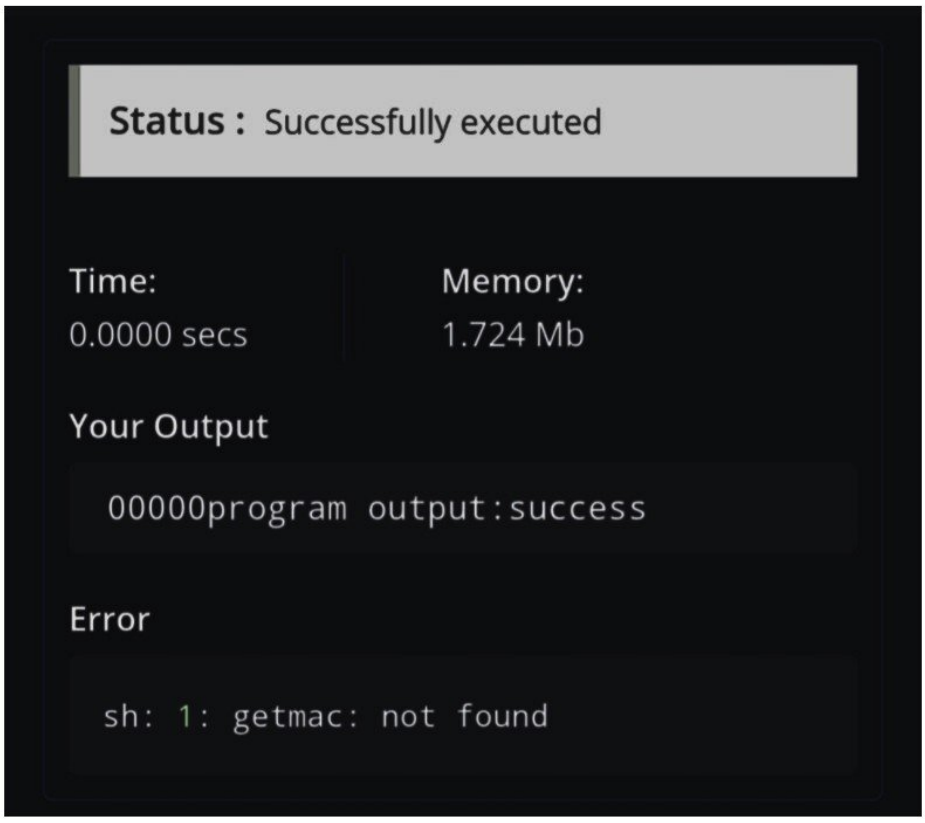
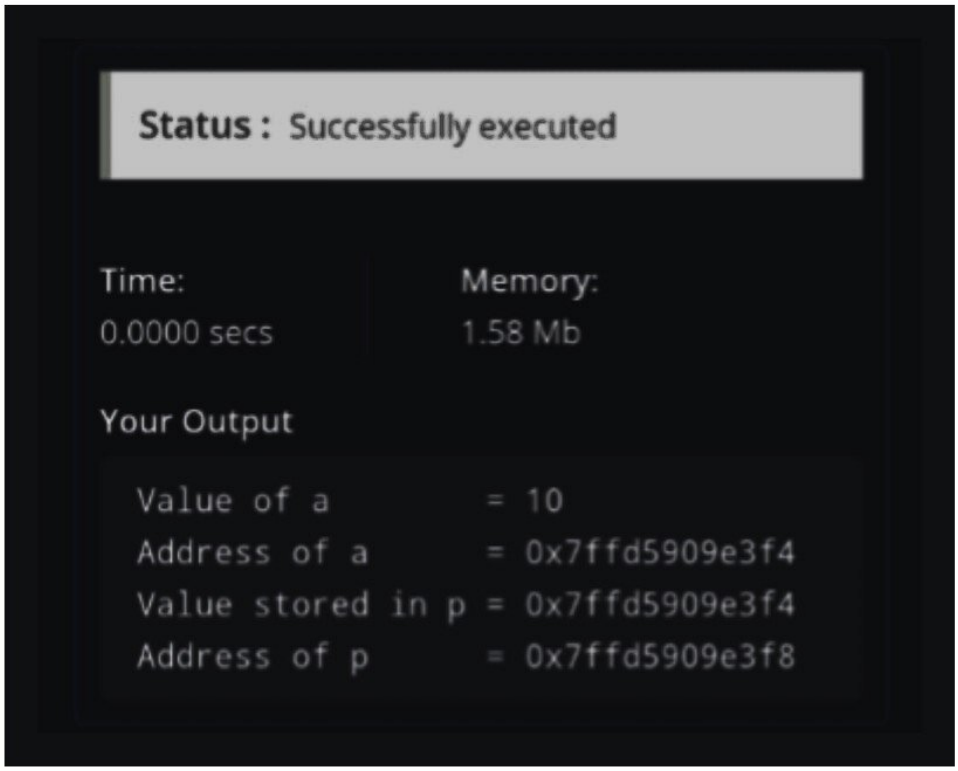
1.Initilization of malloc() functIon Into zero

```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
int n=5, *a;
a=(int*) malloc(n*sizeof(int));
int i;
for(i=0;i<n;i++)
{
a[i]=0;
printf("%d",a[i]);
}
printf("program output:success\n");
system("getmac");
return 0;
}
```



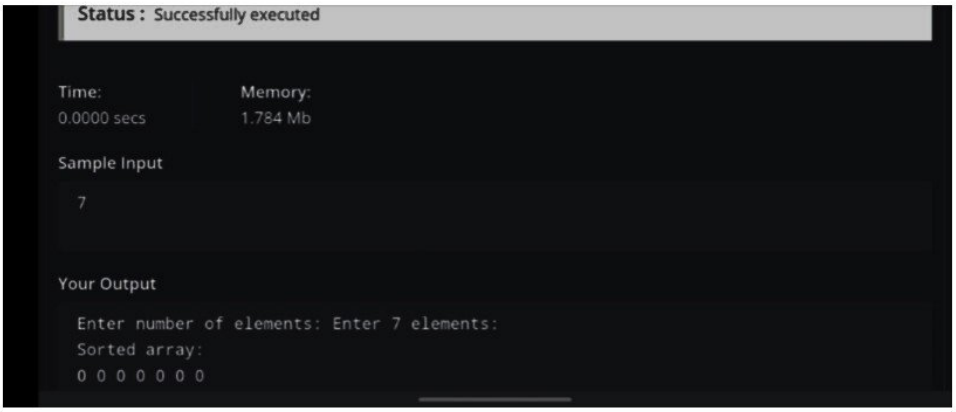
2.Printing the address of the pointer

```
#include <stdio.h>

int main() {
    int a = 10;
    int *p = &a;

    printf("Value of a  = %d\n", a);
    printf("Address of a = %p\n", &a);
    printf("Value stored in p = %p\n", p);
    printf("Address of p      = %p\n", &p);

    return 0;
}
```



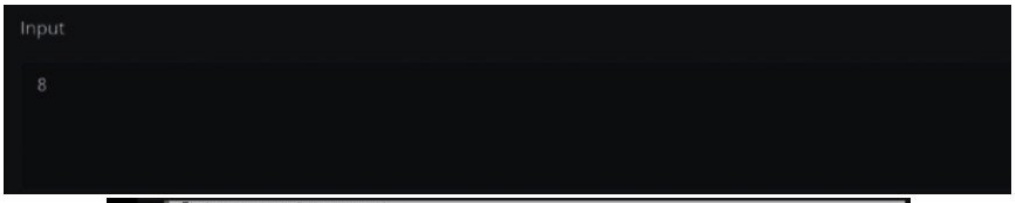
3.Selection sort with dynamic memory allocation

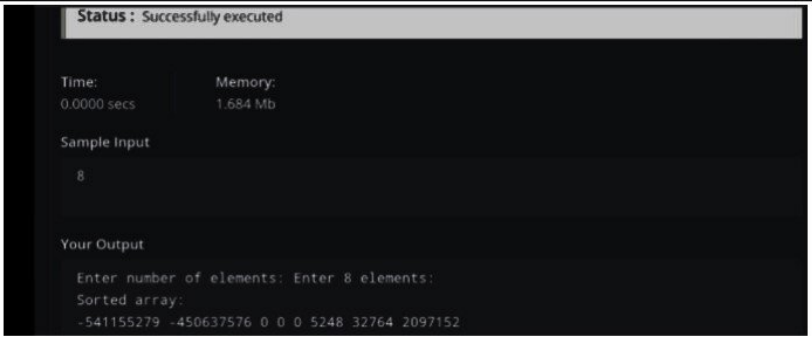
```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int n, i, j, min, temp;
   int *array
   printf("Enter number of elements: ");
   scanf("%d", &n);
   arr = (int *)malloc(n * sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation failed!\n");
      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;
}
```



4.Selection sort without dynamic memory allocation

```
#include <stdio.h>

int main() {
   int n, i, j, min, temp;
   int arr[50];

   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;
}
```

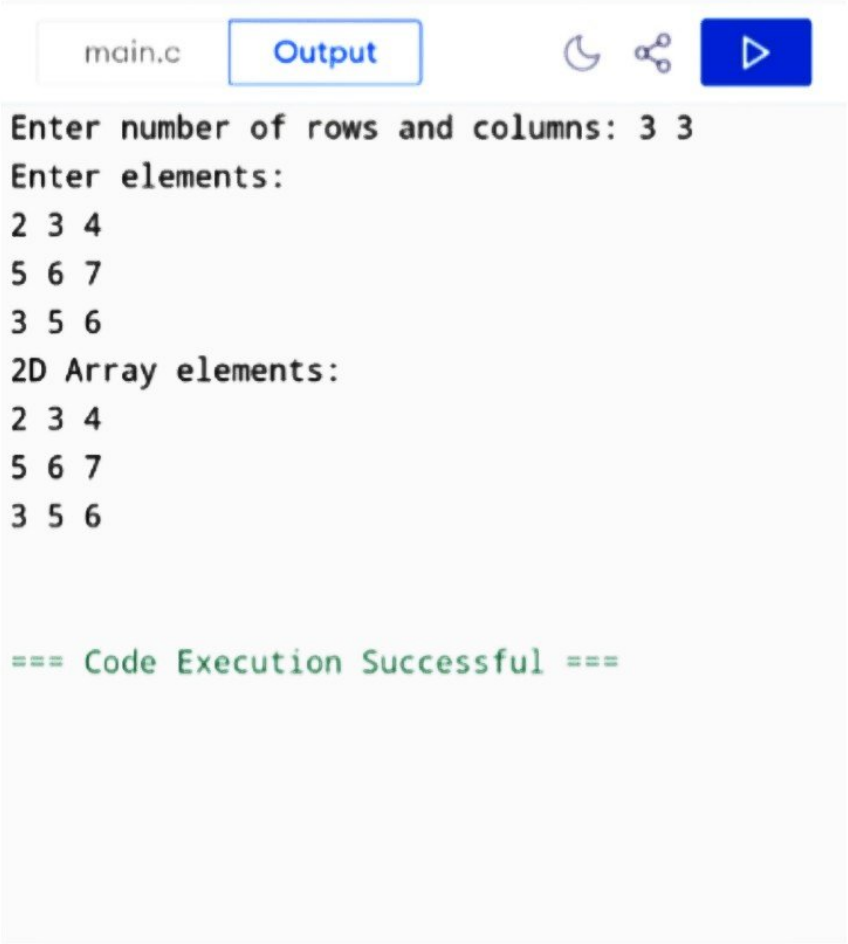




5.Declaration of two-dimensional array using dynamic memory allocation

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

int main() {
   int i, j, rows, cols;
   int **arr;

   printf("Enter number of rows and columns: ");
   scanf("%d %d", &rows, &cols);

   arr = (int **)malloc(rows * sizeof(int *));
   for (i = 0; i < rows; i++) {
      arr[i] = (int *)malloc(cols * sizeof(int));
   }
   printf("Enter elements:\n");
   for (i = 0; i < rows; i++) {
      for (j = 0; j < cols; j++) {
         scanf("%d", &arr[i][j]);
      }
   }
   printf("2D Array elements:\n");
   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;
}
```

5.Declaration of two-dimensional array using dynamic memory allocation



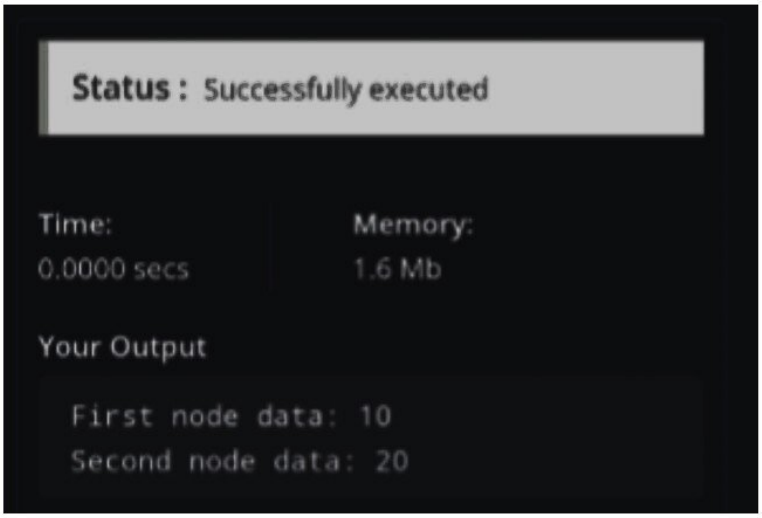
6.Pattern matching

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

int main() {
   char text[100], pattern[50];
   int i, j, n, m, found = 0;

   printf("Enter the text: ");
   scanf("%s", text);

   printf("Enter the pattern: ");
   scanf("%s", pattern);
   n = strlen(text);
   m = strlen(pattern);
   for (i = 0; i <= n - m; i++) {
      for (j = 0; j < m; j++) {
         if (text[i + j] != pattern[j])
            break;
      }
      if (j == m) {
         printf("Pattern found at position %d\n", i + 1);
         found = 1;
      }
   }
   if (!found)
      printf("Pattern not found\n");
   return 0;
}
```


```
Enter the text: abcabcabc
Enter the pattern: abc
Pattern found at position 1
Pattern found at position 4
Pattern found at position 7


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

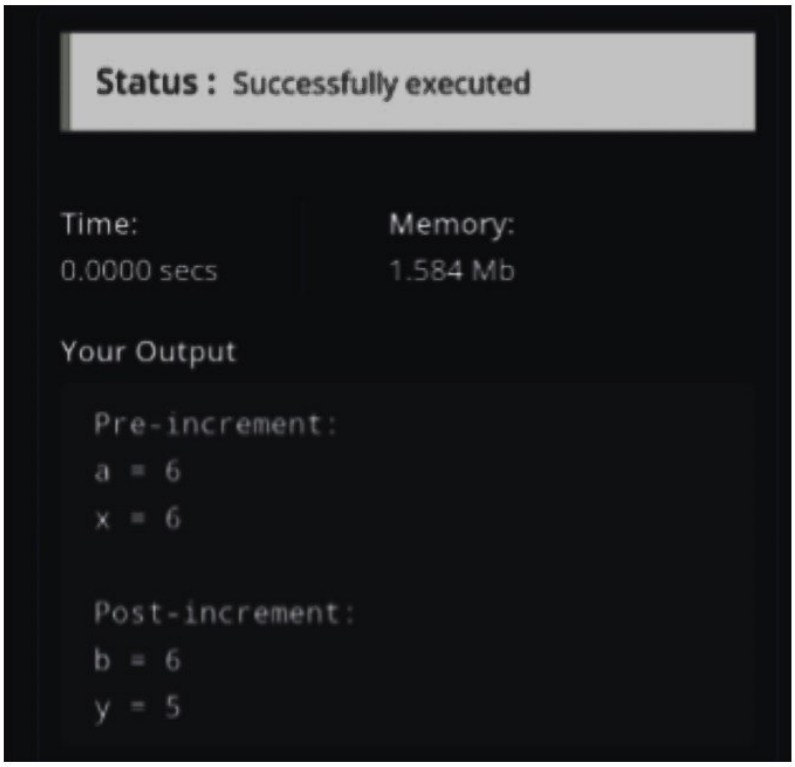
7.Self-Referential structure

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

struct node
{
   int data;
   struct node *next;
};

int main()
{
   struct node *first, *second;
   first = (struct node *)malloc(sizeof(struct node));
   second = (struct node *)malloc(sizeof(struct node));

   first->data = 10;
   first->next = second;

   second->data = 20;
   second->next = NULL;

   printf("First node data: %d\n", first->data);
   printf("Second node data: %d\n", first->next->data);
   free(first);
   free(second);

   return 0;
}
```



8.Pre increment and Post increment

```
#include <stdio.h>

int main() {
    int a = 5, b = 5;
    int x, y;

    x = ++a;
    printf("Pre-increment:\n");
    printf("a = %d\n", a);
    printf("x = %d\n\n", x);

    y = b++;
    printf("Post-increment:\n");
    printf("b = %d\n", b);
    printf("y = %d\n", y);

    return 0;
}
```

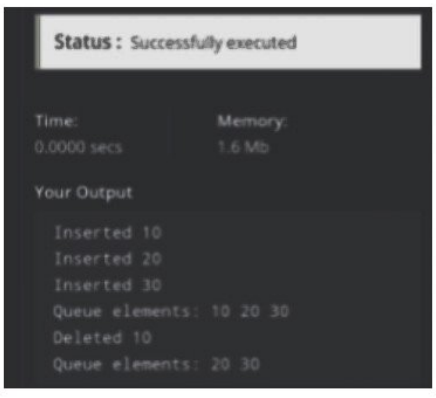


9.Post decrement and Pre decrement

```
#include <stdio.h>

int main() {
    int a = 5, b = 5;
    int x, y;

    x = --a;
    printf("Pre-decrement:\n");
    printf("a = %d\n", a);
    printf("x = %d\n\n", x);

    y = b--;
    printf("Post-decrement:\n");
    printf("b = %d\n", b);
    printf("y = %d\n", y);

    return 0;
}
```

10.Regular Queue

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

int queue[MAX];
int front = -1, rear = -1;

void enqueue(int item) {
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
      return;
   }
   if (front == -1)
      front = 0;
   queue[++rear] = item;
   printf("Inserted %d\n", item);
}

void dequeue() {
   if (front == -1 || front > rear) {
      printf("Queue Underflow\n");
      return;
   }
   printf("Deleted %d\n", queue[front++]);
}

void display() {
   if (front == -1 || front > rear) {
      printf("Queue is Empty\n");
      return;
   }
   printf("Queue elements: ");
   for (int i = front; i <= rear; i++)
      printf("%d ", queue[i]);
   printf("\n");
}

int main() {
   enqueue(10);
   enqueue(20);
   enqueue(30);
   display();

   dequeue();
   display();

   return 0;
}
```



11.Operations of linked lists

```
int main() {
    int choice, value;
    while (1) {
        printf("\n1. Insert\n2. Traverse\n3. Delete\n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

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

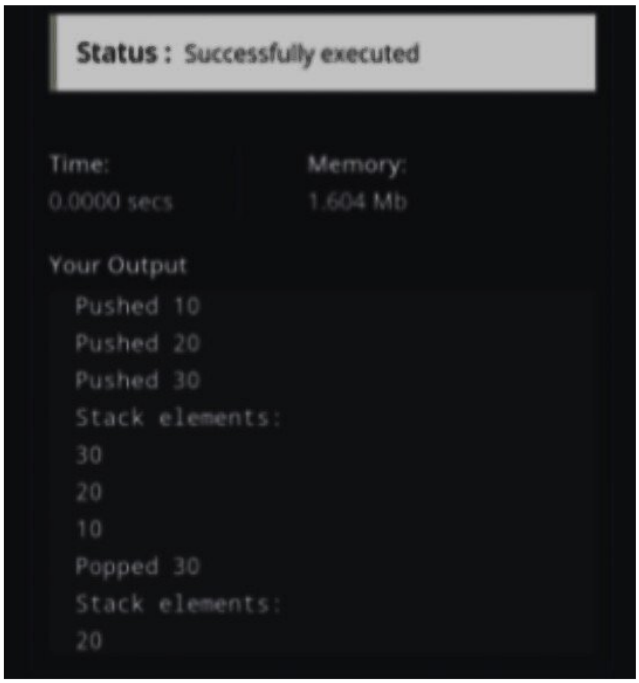

```
1. Insert
2. Traverse
3. Delete
4. Exit
Enter choice: 1
Enter value: 10

1. Insert
2. Traverse
3. Delete
4. Exit
Enter choice: 1
Enter value: 20

1. Insert
2. Traverse
3. Delete
4. Exit
Enter choice: 2
Linked List: 10 -> 20 -> NULL
```

```
1. Insert
2. Traverse
3. Delete
4. Exit
Enter choice: 3

1. Insert
2. Traverse
3. Delete
4. Exit
Enter choice: 2
Linked List: 20 -> NULL
```

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("Pushed %d\n", value);
}

void pop() {
   struct node *temp;

   if (top == NULL) {
      printf("Stack Underflow\n");
      return;
   }

   temp = top;
   printf("Popped %d\n", temp->data);
   top = top->next;
   free(temp);
}

void display() {
   struct node *temp;

   if (top == NULL) {
      printf("Stack is Empty\n");
      return;
   }

   printf("Stack elements:\n");
   temp = top;
   while (temp != NULL) {
      printf("%d\n", temp->data);
      temp = temp->next;
   }
}

int main() {
   push(10);
   push(20);
   push(30);

   display();

   pop();
   display();

   return 0;
}
```

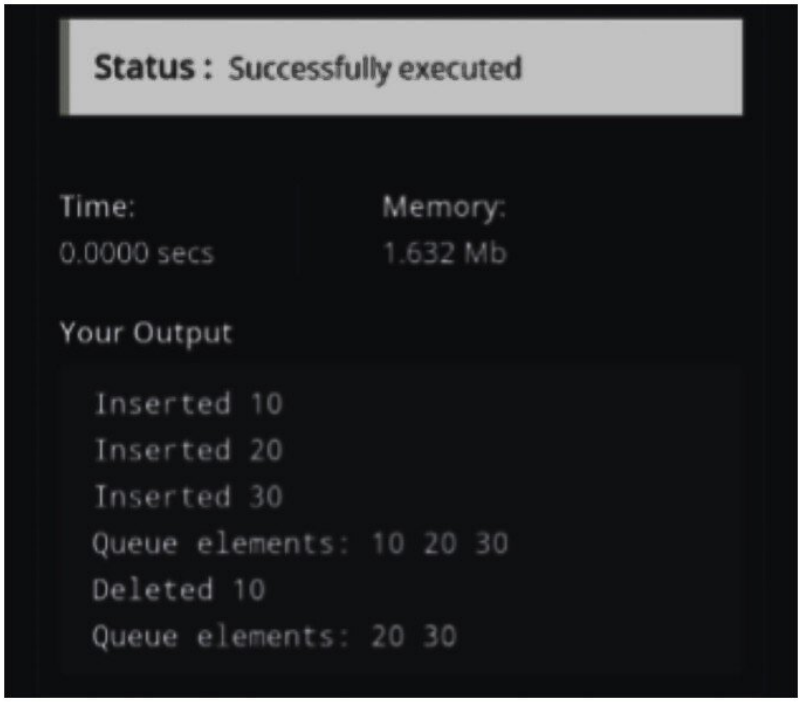
12.Linked list using Stacks



13.Linked list using Queues

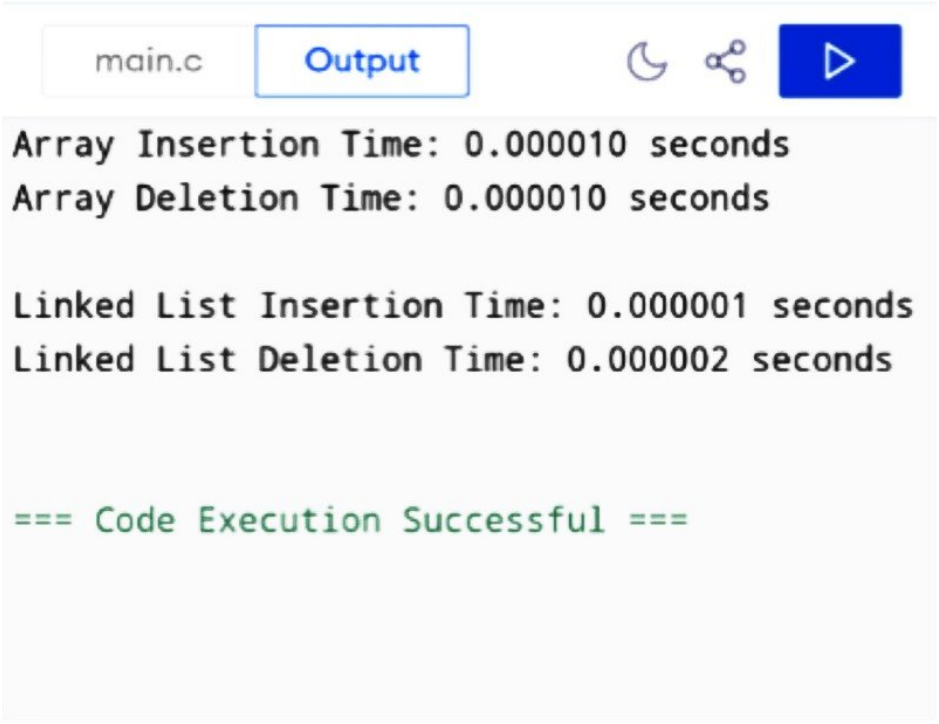
```
#include <stdio.h>
#include <stdlib.h>

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));
   if (newNode == NULL) {
      printf("Queue Overflow\n");
      return;
   }
   newNode->data = value;
   newNode->next = NULL;
   if (front == NULL) {
      front = rear = newNode;
   } else {
      rear->next = newNode;
      rear = newNode;
   }
   printf("Inserted %d\n", value);
}

void dequeue() {
   struct node *temp;

   if (front == NULL) {
      printf("Queue Underflow\n");
      return;
   }
   temp = front;
   printf("Deleted %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;
   }
   printf("Queue elements: ");
   temp = front;
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
   }
   printf("\n");
}
int main() {
   enqueue(10);
   enqueue(20);
   enqueue(30);

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

13.Linked list using Queues



14.Time taken for arrays and linked lists during insertion and deletion

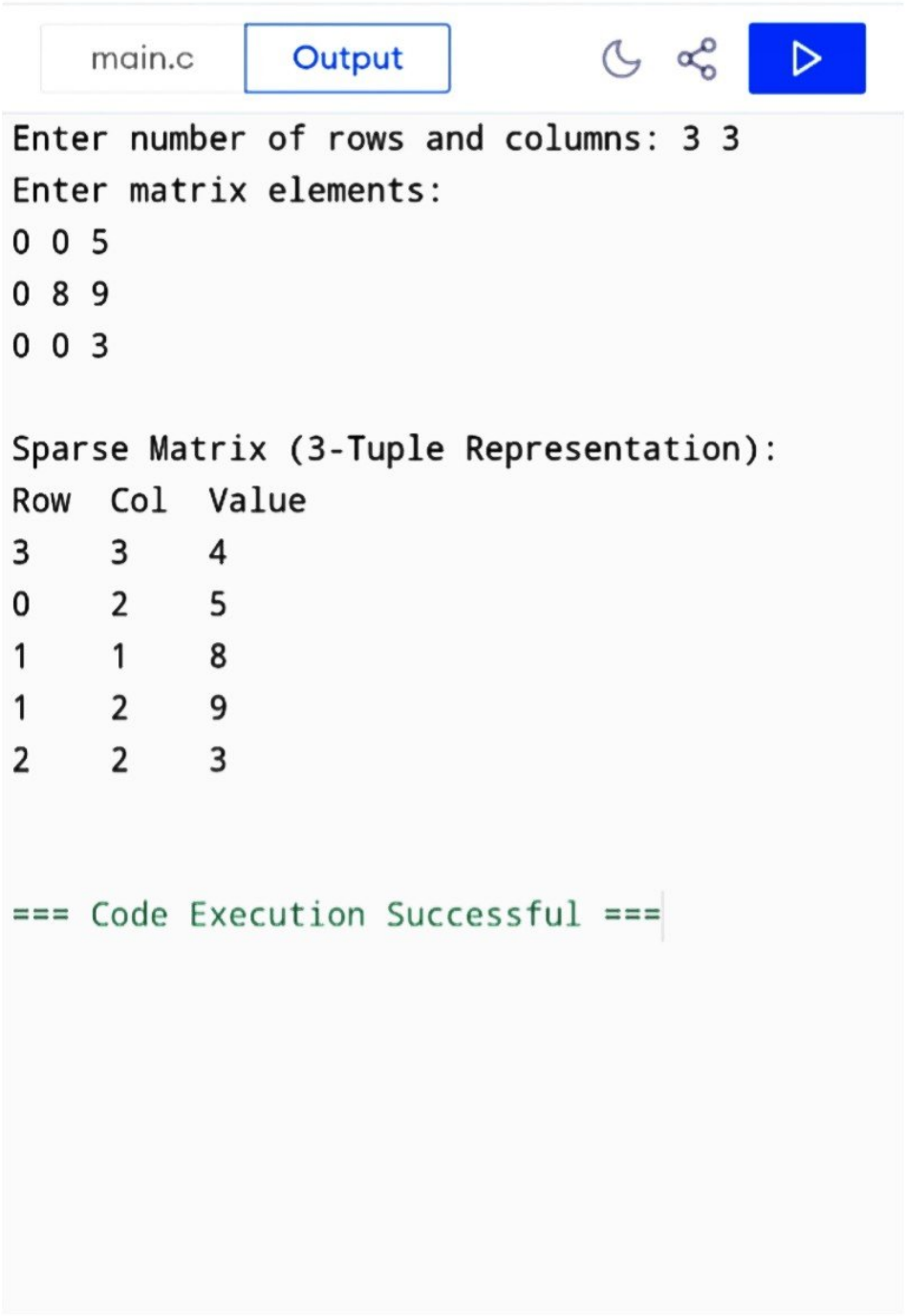
```
    for (i = 0; i < SIZE; i++)
        head = insertLL(head, i);
    start = clock();
    head = insertLL(head, 999);
    end = clock();
    time_ll = ((double)(end - start))
    printf("Linked List Insertion Time: %f seconds\n", time_ll);
    start = clock();
    head = deleteLL(head);
    end = clock();

    time_ll = ((double)(end - start))
    printf("Linked List Deletion Time: %f seconds\n", time_ll);
    return 0;
}
```



15.Sparse Matrix

```
#include <stdio.h>

int main() {
   int a[10][10], sparse[20][3];
   int i, j, m, n, k = 1;

   printf("Enter number of rows and columns: ");
   scanf("%d %d", &m, &n);

   printf("Enter matrix elements:\n");
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &a[i][j]);
      }
   }
   sparse[0][0] = m;
   sparse[0][1] = n;
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++) {
         if (a[i][j] != 0) {
            sparse[k][0] = i;
            sparse[k][1] = j;
            sparse[k][2] = a[i][j];
            k++;
         }
      }
   }
   sparse[0][2] = k - 1;

   // Di
   printf("\nSparse Matrix:\n");
   printf("Row  Col  Value\n");
   for (i = 0; i < k; i++) {
      printf("%d    %d    %d\n",
            sparse[i][0], sparse[i][1], sparse[i][2]);
   }
   return 0;
}
```

15.Sparse Matrix

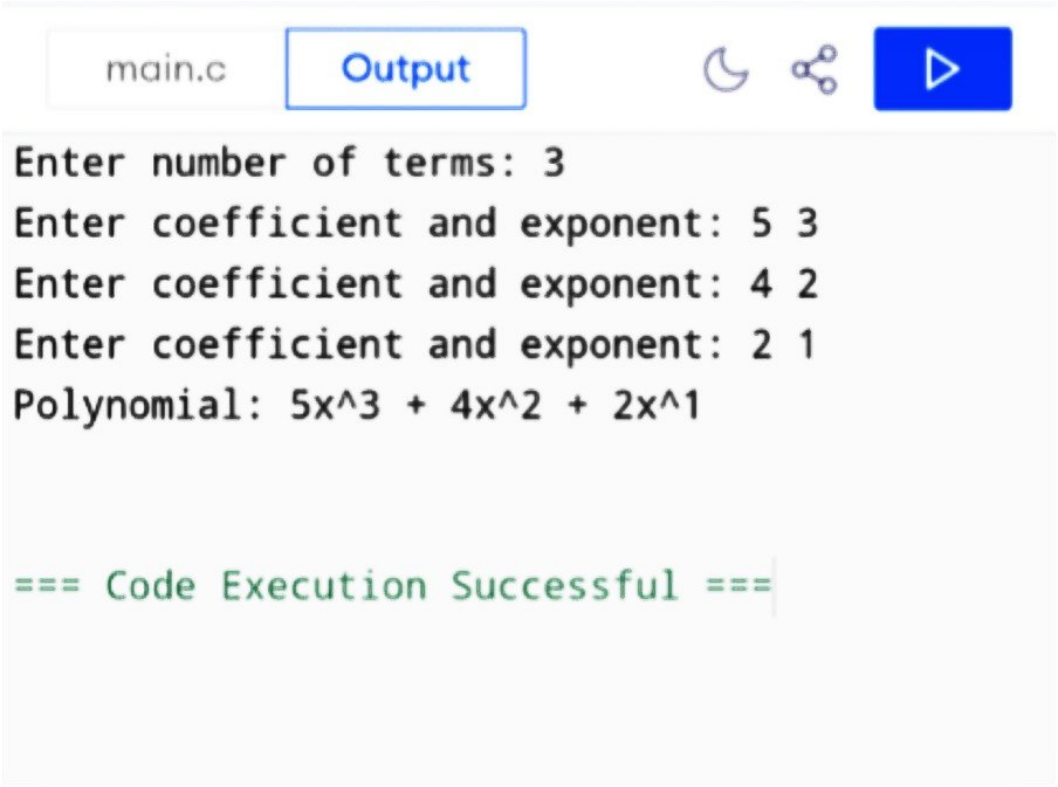


16.Polynomial Representation

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

struct node {
   int coeff;
   int exp;
   struct node *next;
};

struct node* createNode(int c, int e) {
   struct node *temp = (struct node*)malloc(sizeof(struct node));
   temp->coeff = c;
   temp->exp = e;
   temp->next = NULL;
   return temp;
}

struct node* insert(struct node *head, int c, int e) {
   struct node *temp = createNode(c, e);
   struct node *ptr;

   if (head == NULL) {
      head = temp;
   } else {
      ptr = head;
      while (ptr->next != NULL)
         ptr = ptr->next;
      ptr->next = temp;
   }
   return head;
}

void display(struct node *head) {
   struct node *ptr = head;
   while (ptr != NULL) {
      printf("%dx^%d", ptr->coeff, ptr->exp);
      if (ptr->next != NULL)
         printf(" + ");
      ptr = ptr->next;
   }
   printf("\n");
}
```

16.Polynomial Representation

```
int main() {
    struct node *poly = NULL;
    int n, c, e;

    printf("Enter number of terms: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter coefficient and exponent: ");
        scanf("%d %d", &c, &e);
        poly = insert(poly, c, e);
    }

    printf("Polynomial: ");
    display(poly);

    return 0;
}
```



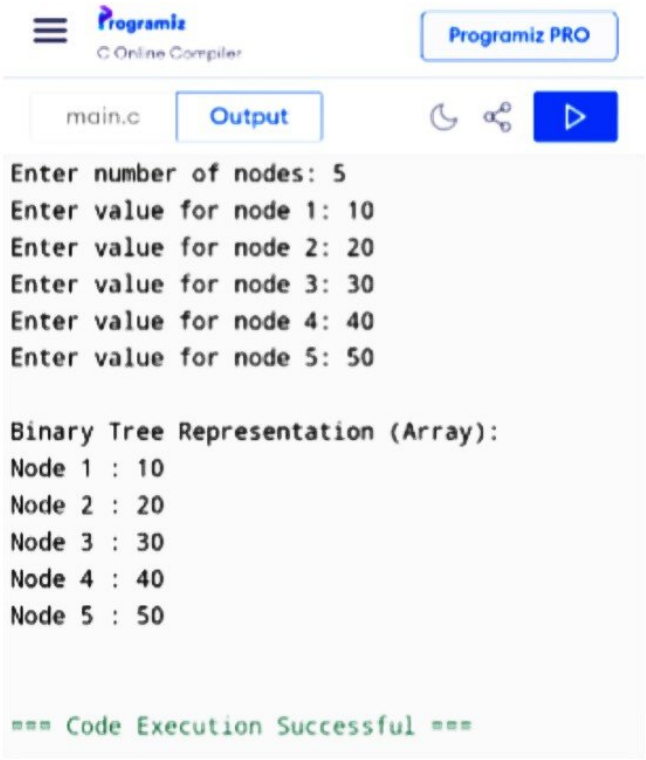
17.How to create a tree

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

struct node {
   int data;
   struct node *left, *right;
};
struct node* create() {
   int x;
   struct node *newnode;
   newnode = (struct node*)malloc(sizeof(struct node));

   printf("Enter data (-1 for no node): ");
   scanf("%d", &x);
   if (x == -1)
      return NULL;

   newnode->data = x;
   printf("Enter left child of %d\n", x);
   newnode->left = create();
   printf("Enter right child of %d\n", x);
   newnode->right = create();
   return newnode;
}
int main() {
   struct node *root;
   printf("Create Binary Tree\n");
   root = create();
   return 0;
}
```

17.How to create a tree

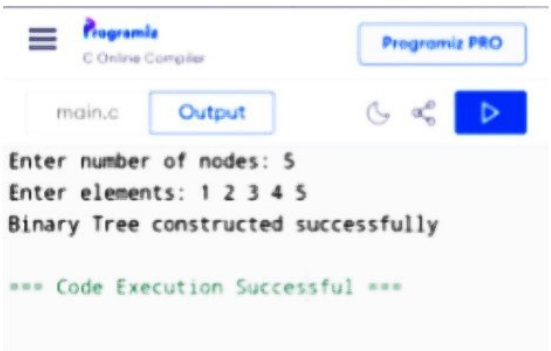
18.Construct a tree using arrays

```c
#include <stdio.h>

int main()
{
   int tree[50], n, i;

   printf("Enter number of nodes: ");
   scanf("%d", &n);
   for(i = 1; i <= n; i++)
   {
      printf("Enter value for node %d: ", i);
      scanf("%d", &tree[i]);
   }

   printf("\nBinary Tree Representation (Array):\n");
   for(i = 1; i <= n; i++)
   {
      printf("Node %d : %d\n", i, tree[i]);
   }

   return 0;
}
```



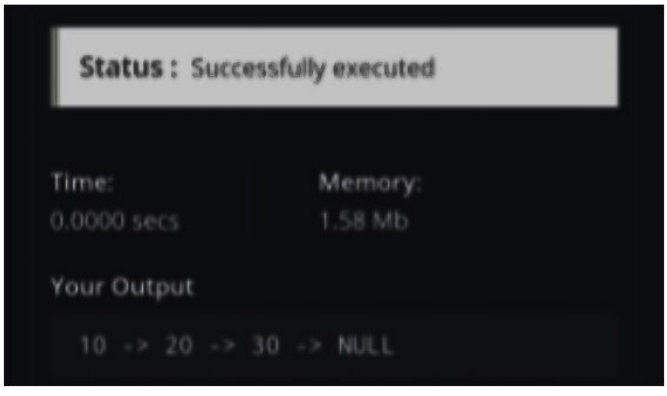
19.Construct a binary tree using queues

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

struct node {
   int data;
   struct node *left, *right;
};

struct node* create(int data) {
   struct node* n = (struct node*)malloc(sizeof(struct node));
   n->data = data;
   n->left = n->right = NULL;
   return n;
}

int main() {
   struct node *root, *temp;
   struct node *q[20];
   int front = 0, rear = 0, n, i;

   printf("Enter number of nodes: ");
   scanf("%d", &n);

   int a[n];
   printf("Enter elements: ");
   for (i = 0; i < n; i++)
      scanf("%d", &a[i]);

   root = create(a[0]);
   q[rear++] = root;

   i = 1;
   while (i < n) {
      temp = q[front++];

      temp->left = create(a[i++]);
      q[rear++] = temp->left;

      if (i < n) {
         temp->right = create(a[i++]);
         q[rear++] = temp->right;
      }
   }

   printf("Binary Tree constructed successfully");
   return 0;
}
```



20.Insertion of nodes

```
#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 display() {
   struct node *temp = head;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

int main() {
   insert(10);
   insert(20);
   insert(30);

   display();
   return 0;
}
```

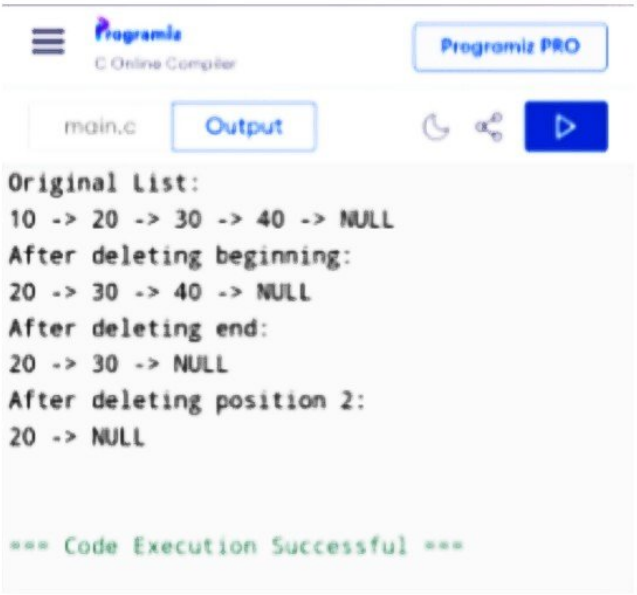


21.Deletion of nodes

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

struct node {
   int data;
   struct node *next;
};
struct node *head = NULL;
void insert(int val) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = val;
   newnode->next = NULL;

   if (head == NULL) {
      head = newnode;
   } else {
      struct node *temp = head;
      while (temp->next != NULL)
         temp = temp->next;
      temp->next = newnode;
   }
}
void delete_begin() {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   struct node *temp = head;
   head = head->next;
   free(temp);
}

void delete_end() {
   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
void delete_position(int pos) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
```

21.Deletion of nodes

```
void delete_position(int pos) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    if (pos == 1) {
        delete_begin();
        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;
    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();

    delete_begin();
    printf("After deleting beginning:\n");
    display();

    delete_end();
    printf("After deleting end:\n");
    display();

    delete_position(2);
    printf("After deleting position 2:\n");
    display();

    return 0;
}
```



22.Depth search tree

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

#define MAX 100
struct Node {
   int vertex;
   struct Node* next;
};

struct Graph {
   int numVertices;
   struct Node* adjLists[MAX]
   int visited[MAX];

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

void createGraph(struct Graph* graph, int vertices) {
   graph->numVertices = vertices;

   for (int i = 0; i < vertices; i++) {
      graph->adjLists[i] = NULL;
      graph->visited[i] = 0;
   }
}
void addEdge(struct Graph* graph, int src, int dest) {
   struct Node* newNode = createNode(dest);
   newNode->next = graph->adjLists[src];
   graph->adjLists[src] = newNode;
   newNode = createNode(src);
   newNode->next = graph->adjLists[dest];
   graph->adjLists[dest] = newNode;
}
```

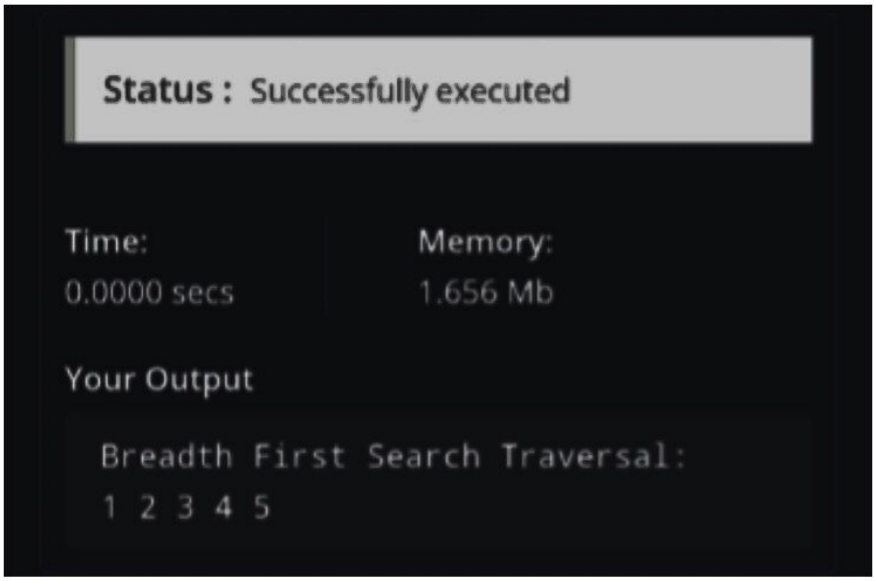
22.Depth search tree

```
void DFS(struct Graph* graph, int startVertex) {
    struct Node* adjList = graph->adjLists[startVertex];
    struct Node* temp = adjList;

    graph->visited[startVertex] = 1;
    printf("%d ", startVertex);

    while (temp != NULL) {
        int connectedVertex = temp->vertex;

        if (graph->visited[connectedVertex] == 0) {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}
int main() {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

    int vertices, edges;
    printf("Enter number of vertices: ");
    scanf("%d", &vertices);
    createGraph(graph, vertices);
    printf("Enter number of edges: ");
    scanf("%d", &edges);
    printf("Enter edges (u v):\n");
    for (int i = 0; i < edges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(graph, u, v);
    }
    printf("DFS Traversal starting from vertex 0:\n");
    DFS(graph, 0);

    return 0;
}
```

23.Breadth search tree

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

struct node {
   int data;
   struct node *left, *right;
};

struct node* createNode(int data) {
   struct node* newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->left = newNode->right = NULL;
   return newNode;
}
struct node* queue[50];
int front = -1, rear = -1;

void enqueue(struct node* root) {
   if (rear == 49) return;
   if (front == -1) front = 0;
   queue[++rear] = root;
}

struct node* dequeue() {
   if (front > rear) return NULL;
   return queue[front++];
}
void BFS(struct node* root) {
   if (root == NULL) return;
   enqueue(root);
   while (front <= rear) {
      struct node* temp = dequeue();
      printf("%d ", temp->data);
      if (temp->left)
         enqueue(temp->left);
      if (temp->right)
         enqueue(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);
   printf("Breadth First Search Traversal:\n");
   BFS(root);
   return 0;
}
```

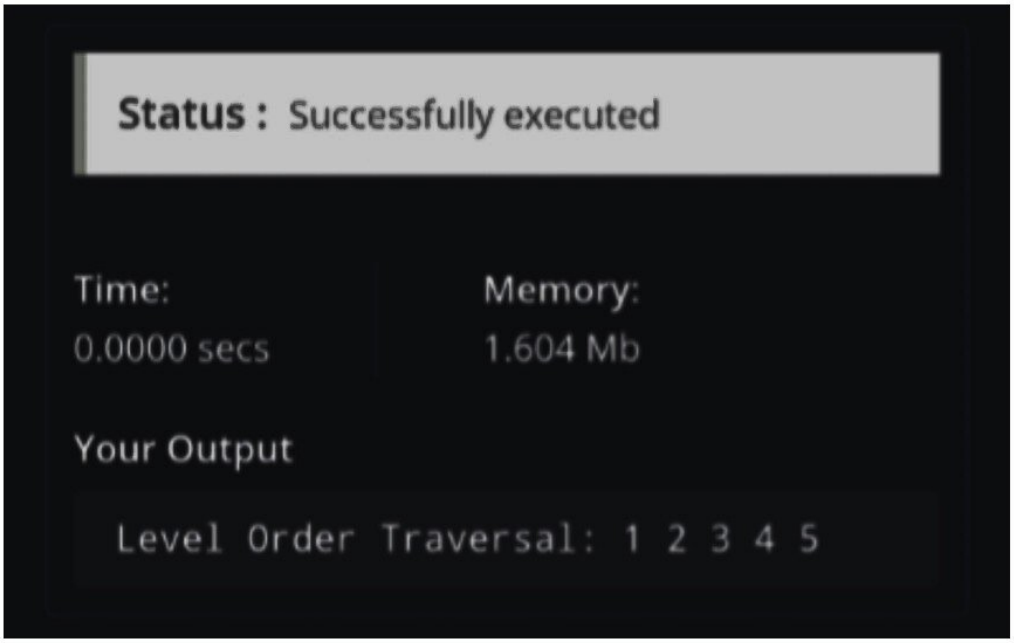
23.Breadth search tree



24.Level order

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

struct node {
   int data;
   struct node *left, *right;
};
struct node* newNode(int data) {
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}
struct node* queue[50];
int front = -1, rear = -1;
void enqueue(struct node* root) {
   if (rear == 49) return;
   if (front == -1) front = 0;
   queue[++rear] = root;
}
struct node* dequeue() {
   if (front > rear) return NULL;
   return queue[front++];
}
void levelOrder(struct node* root) {
   if (root == NULL) return;
   enqueue(root);
   while (front <= rear) {
      struct node* temp = dequeue();
      printf("%d ", temp->data);
      if (temp->left != NULL)
         enqueue(temp->left);
      if (temp->right != NULL)
         enqueue(temp->right);
   }
}
int main() {
   struct node* root = newNode(1);
   root->left = newNode(2);
   root->right = newNode(3);
   root->left->left = newNode(4);
   root->left->right = newNode(5);
   printf("Level Order Traversal: ");
   levelOrder(root);

   return 0;
}
```

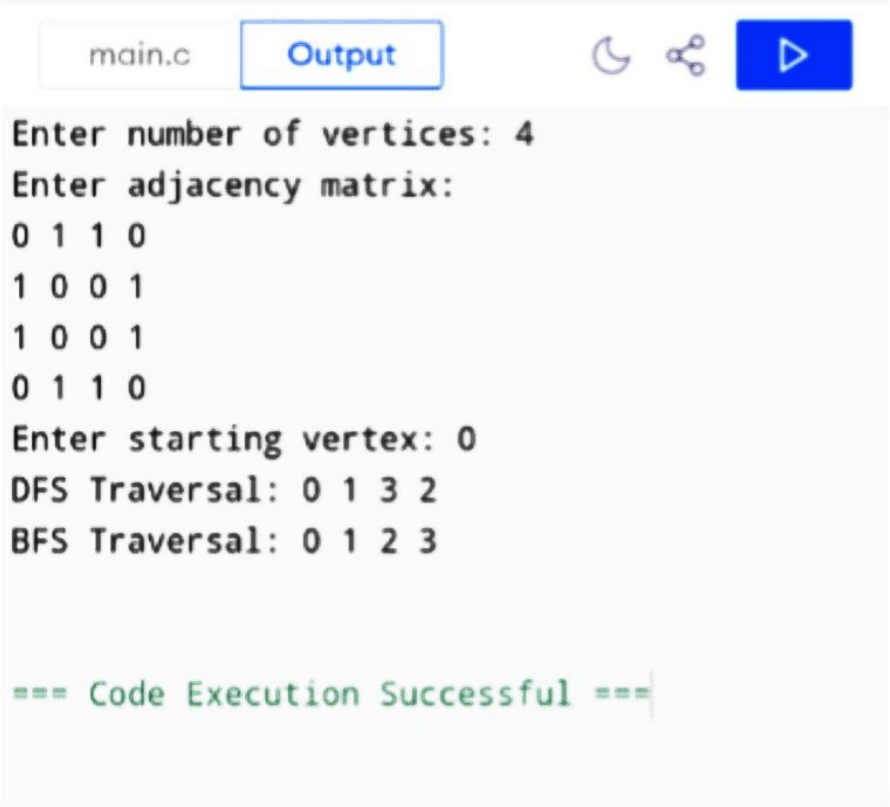
24.Level order



25.DFS and BFS using adjacency list

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

#define MAX 10

int adj[MAX][MAX], visited[MAX], n;
int stack[MAX], top = -1;
void push(int v) { stack[++top] = v; }
int pop() { return stack[top--]; }
int queue[MAX], front = 0, rear = -1;
void enqueue(int v) { queue[++rear] = v; }
int dequeue() { return queue[front++]; }
void DFS(int start) {
   int i, v;
   for (i = 0; i < n; i++)
      visited[i] = 0;

   push(start);

   printf("DFS Traversal: ");
   while (top != -1) {
      v = pop();
      if (!visited[v]) {
         printf("%d ", v);
         visited[v] = 1;
      }
      for (i = n - 1; i >= 0; i--) {
         if (adj[v][i] && !visited[i])
            push(i);
      }
   }
   printf("\n");
}
void BFS(int start) {
   int i, v;
   for (i = 0; i < n; i++)
      visited[i] = 0;
   enqueue(start);
   visited[start] = 1;
   printf("BFS Traversal: ");
   while (front <= rear) {
      v = dequeue();
      printf("%d ", v);
      for (i = 0; i < n; i++) {
         if (adj[v][i] && !visited[i]) {
            enqueue(i);
            visited[i] = 1;
         }
      }
   }
   printf("\n");
}
int main() {
   int i, j, start;
   printf("Enter number of vertices: ");
   scanf("%d", &n);
   printf("Enter adjacency matrix:\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         scanf("%d", &adj[i][j]);
   printf("Enter starting vertex: ");
   scanf("%d", &start);

   DFS(start);
   BFS(start);
   return 0;
}
```

25.DFS and BFS using adjacency list

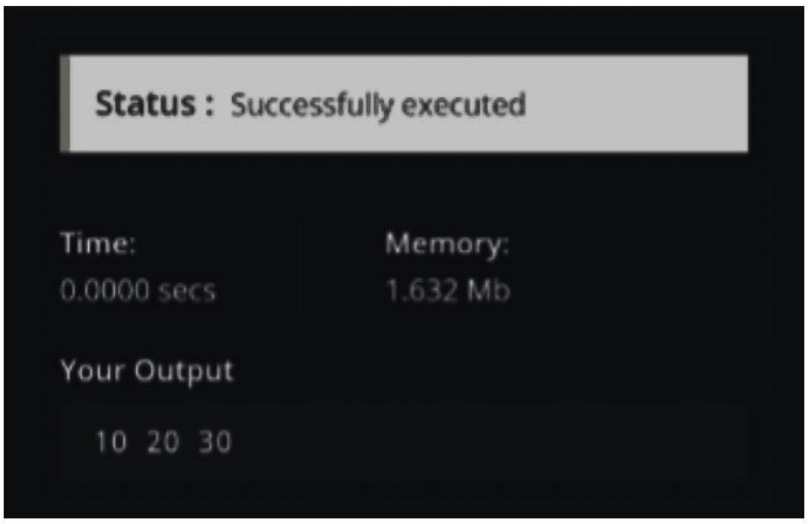


26.Calloc and Malloc functions checking if junk/zero is initialized to them

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

int main() {
   int i, n = 5;
   int *m, *c;
   m = (int *)malloc(n * sizeof(int));
   c = (int *)calloc(n, sizeof(int));
   printf("Values after malloc():\n");
   for (i = 0; i < n; i++)
      printf("%d ", m[i]);
   printf("\n\nValues after calloc():\n");
   for (i = 0; i < n; i++)
      printf("%d ", c[i]);
   free(m);
   free(c);
   return 0;
}
```

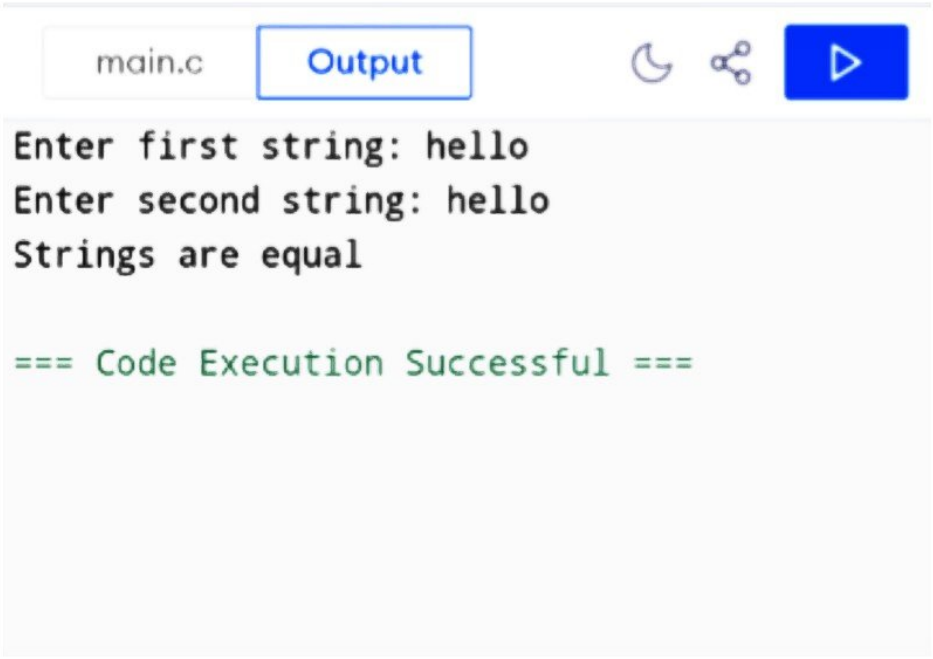
27.Circular linked lists basic operations

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

struct Node {
   int data;
   struct Node* next;
};
struct Node* insertEnd(struct Node* head, int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   if (!head) {
      newNode->next = newNode;
      return newNode;
   }
   struct Node* temp = head;
   while (temp->next != head) temp = temp->next;
   temp->next = newNode;
   newNode->next = head;
   return head;
}
void display(struct Node* head) {
   if (!head) { printf("Empty\n"); return; }
   struct Node* temp = head;
   do { printf("%d ", temp->data); temp = temp->next; } while (temp != head);
   printf("\n");
}

int main() {
   struct Node* head = NULL;
   head = insertEnd(head, 10);
   head = insertEnd(head, 20);
   head = insertEnd(head, 30);
   display(head);
   return 0;
}
```



28.How to implement comparison of 2 strings using built in function

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

int main()
{
   char s1[50], s2[50];
   printf("Enter first string: ");
   scanf("%s", s1);
   printf("Enter second string: ");
   scanf("%s", s2);
   if (strcmp(s1, s2) == 0)
      printf("Strings are equal");
   else
      printf("Strings are not equal");
   return 0;
}
```

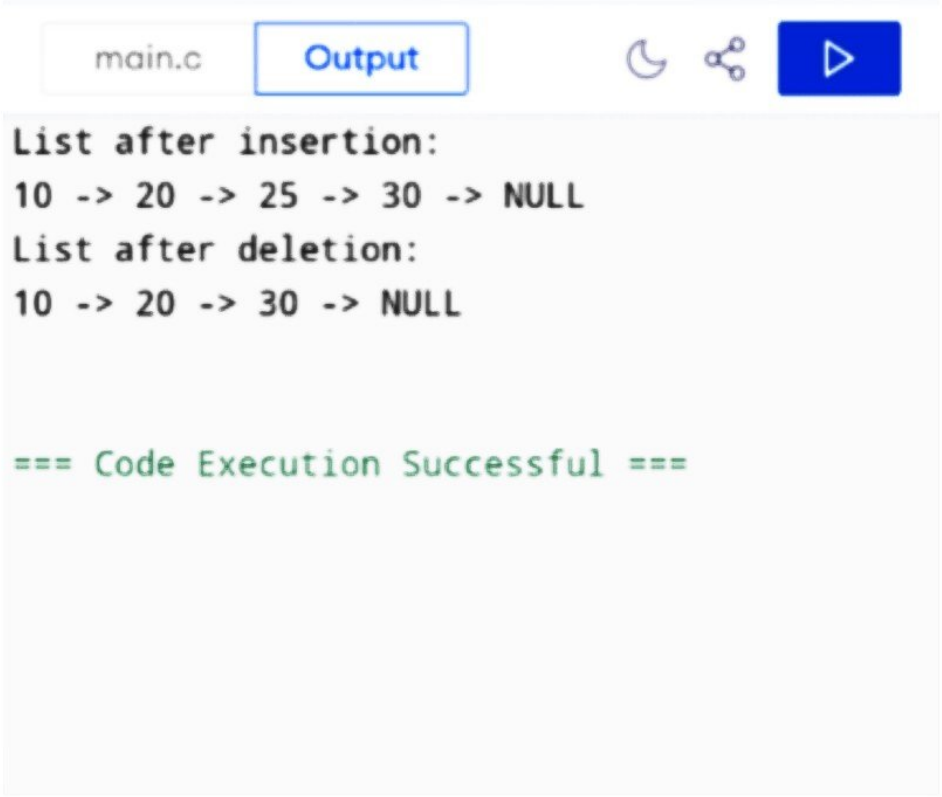


29.In linked list insertion in the middle and deletion in the middle

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

struct node {
   int data;
   struct node *next;
};

struct node *head = NULL;
void display() {
   struct node *temp = head;
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

void insert_middle(int data, int pos) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   newnode->data = data;
   if (pos == 1) {
      newnode->next = head;
      head = newnode;
      return;
   }

   struct node *temp = head;
   for (int i = 1; i < pos - 1 && temp != NULL; i++) {
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position not valid\n");
      return;
   }
   newnode->next = temp->next;
   temp->next = newnode;
}
void delete_middle(int pos) {
   if (head == NULL) {
      printf("List is empty\n");
      return;
   }
   struct node *temp = head;
   if (pos == 1) {
      head = temp->next;
      free(temp);
      return;
   }
   struct node *prev = NULL;
   for (int i = 1; i < pos && temp != NULL; i++) {
      prev = temp;
      temp = temp->next;
   }
   if (temp == NULL) {
      printf("Position not valid\n");
      return;
   }
```

29.In linked list insertion in the middle and deletion in the middle

```
  prev->next = temp->next;
    free(temp);
}
int main() {
    insert_middle(10, 1);
    insert_middle(20, 2);
    insert_middle(30, 3);
    insert_middle(25, 3);
    printf("List after insertion:\n");
    display();
    delete_middle(3);
    printf("List after deletion:\n");
    display();

    return 0;
}
```



30.Binary tree traversal

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

struct node {
   int data;
   struct node *left, *right;
};
struct node* create(int data) {
   struct node* newnode = (struct node*)malloc(sizeof(struct node));
   newnode->data = data;
   newnode->left = newnode->right = NULL;
   return newnode;
}
void inorder(struct node* root) {
   if (root != NULL) {
      inorder(root->left);
      printf("%d ", root->data);
      inorder(root->right);
   }
}
void preorder(struct node* root) {
   if (root != NULL) {
      printf("%d ", root->data);
      preorder(root->left);
      preorder(root->right);
   }
}
void postorder(struct node* root) {
   if (root != NULL) {
      postorder(root->left);
      postorder(root->right);
      printf("%d ", root->data);
   }
}
int main() {
   struct node* root = create(1);
   root->left = create(2);
   root->right = create(3);
   root->left->left = create(4);
   root->left->right = create(5);
   printf("Inorder Traversal: ");
   inorder(root);
   printf("\nPreorder Traversal: ");
   preorder(root);
   printf("\nPostorder Traversal: ");
   postorder(root);
   return 0;
}
```

30.Binary tree traversal