1. Write a program in C to print the Minimum number of nodes in an AVL Tree with given height.

```
// Function to find
// minimum number of nodes
int AVLnodes(int height)
{
   // Base Conditions
   if (height == 0)
      return 1;
   else if (height == 1)
      return 2;

   // Recursive function call
   // for the recurrence relation
   return (1 + AVLnodes(height - 1) + AVLnodes(height - 2));
}


int main()
{
   int H = 3;
   printf(“%d”,AVLnodes(H));
}
```

2.Write a Function to Add and Remove Edge in Adjacency Matrix representation of a Graph.

```
void addEdge(int x, int y)
   {
      // Checks if the vertices
      // exist in the graph
      if ((x < 0) || (x >= n)) {
         printf("Vertex  %d  does not exist!",x);
      }
```

```
    if ((y < 0) || (y >= n)) {
        printf("Vertex  %d  does not exist!",y);
    }

    // Checks if it is a self edge
    if (x == y) {
        printf( "Same Vertex!");
    }

    else {
        // Insert edge
        g[y][x] = 1;
        g[x][y] = 1;
    }
}

// Function to update adjacency
// matrix for edge removal
void removeEdge(int x, int y)
{
    // Checks if the vertices
    // exist in the graph
    if ((x < 0) || (x >= n)) {
        printf("Vertex  %d  does not exist!",x);
    }
    if ((y < 0) || (y >= n)) {
        printf("Vertex  %d  does not exist!",y);       }

    // Checks if it is a self edge
    if (x == y) {
        printf("Same Vertex!");
    }

    else {
        // Remove edge
        g[y][x] = 0;
        g[x][y] = 0;
    }
}
```

```
};
```

3. Program in c to check whether a cycle exists or not in a graph

```
#include <stdlib.h>

typedef struct {
   unsigned int first;
   unsigned int second;
} edge;

static unsigned int cyclic_recursive(const edge *edges, unsigned int n,
unsigned int *visited,
      unsigned int order, unsigned int vertex, unsigned int predecessor)
{
   unsigned int i;
   unsigned int cycle_found = 0;
   visited[vertex] = 1;
   for (i = 0; i < n && !cycle_found; i++) {
      if (edges[i].first == vertex || edges[i].second == vertex) {
         /* Adjacent */
         const unsigned int neighbour = edges[i].first == vertex ?
               edges[i].second : edges[i].first;
         if (visited[neighbour] == 0) {
            /* Not yet visited */
            cycle_found = cyclic_recursive(edges, n, visited, order, neighbour,
vertex);
         }
         else if (neighbour != predecessor) {
            /* Found a cycle */
            cycle_found = 1;
         }
      }
   }
   return cycle_found;
}

unsigned int cyclic(const edge *edges, unsigned int n, unsigned int order)
{
```

```
    unsigned int *visited = calloc(order, sizeof(unsigned int));
    unsigned int cycle_found;
    if (visited == NULL) {
        return 0;
    }
    cycle_found  = cyclic_recursive(edges, n, visited, order, 0, 0);
    free(visited);
    return cycle_found;
}
```

4. Write a procedure to return the degree of a given node

```
int findDegree(struct graph *G, int ver)
{
    // Traverse through row of ver and count
    // all connected cells (with value 1)
    int degree = 0;
    for (int i=0; i<G->v; i++)

        // if src to des is 1 the degree count
        if (G-> dir[ver][i] == 1)
            degree++;

    return degree;
}
```

5. Program to count the number of nodes in a graph

```
struct node

{

    int info;

    struct node *left, *right;

};

struct node *createnode(int key)

{
```

```
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->info = key;
    newnode->left = NULL;
    newnode->right = NULL;
    return(newnode);
}
static int count = 0;
int countnodes(struct node *root)
{
    if(root != NULL)
    {
        countnodes(root->left);
        count++;
        countnodes(root->right);
    }
    return count;
}
```