WEEK1 LAB1 :

1.Write a program to construct a binary tree to support the following operations.
Assume no duplicate elements while constructing the tree.
i. Given a key, perform a search in the binary search tree. If the key is found then display “key found” else insert the key in the binary search tree.
ii. Display the tree using inorder, preorder and post order traversal methods

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

typedef struct node
{
struct node *leftnode;
struct node *rightnode;
int element;
} * NODE;

NODE insert(NODE first, int element)
{

if (first == NULL)
{
NODE n = (NODE)malloc(sizeof(struct node));
n->leftnode = NULL;
n->rightnode = NULL;
n->element = element;
return n;
}
else if (element < first->element)
first->leftnode = insert(first->leftnode, element);
else if (element == first->element)
printf("Duplicate node is : \n");
else
first->rightnode = insert(first->rightnode, element);
return first;
```

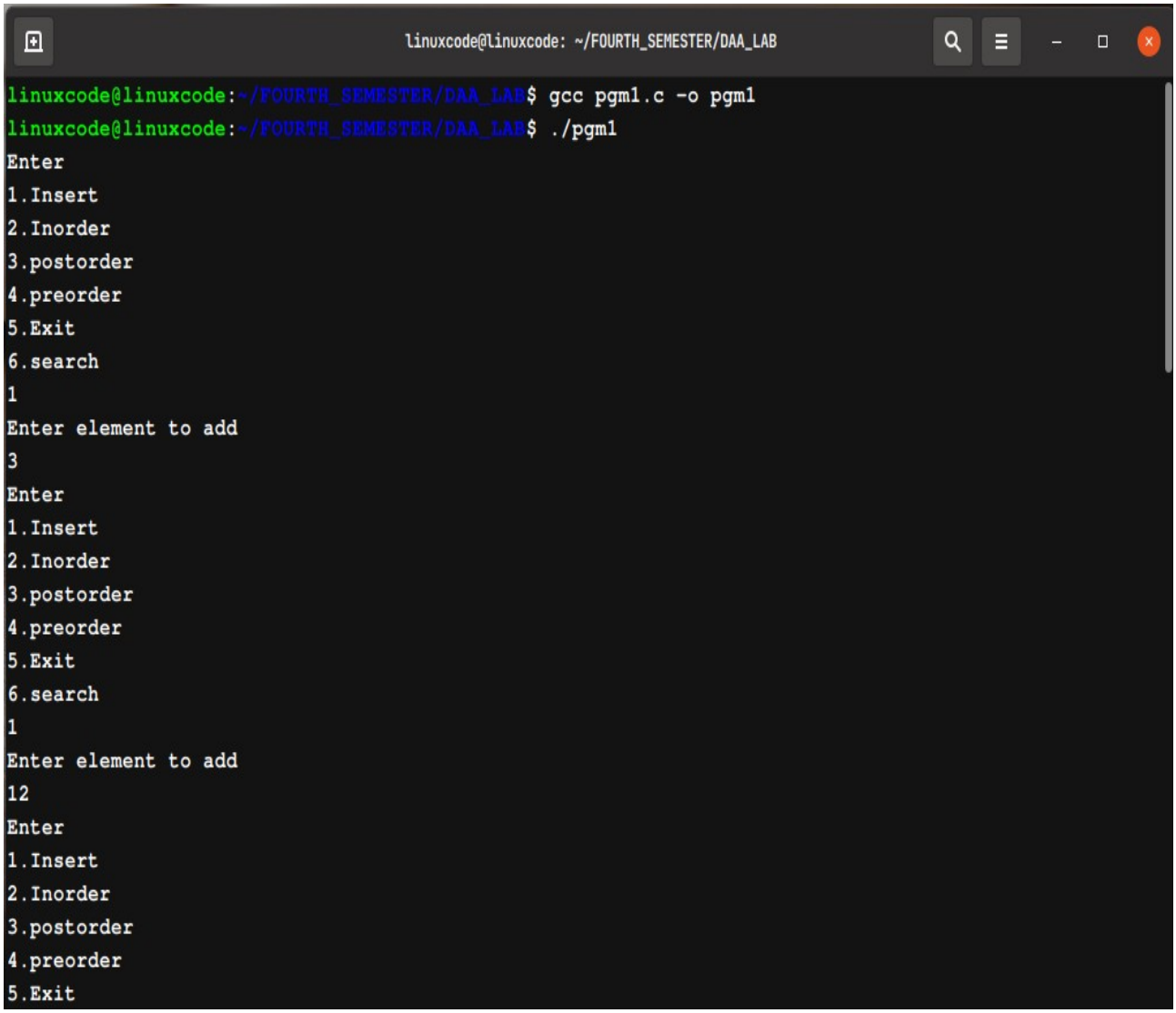
```
}

void inorder(NODE first)
{
if (first != NULL)
{
inorder(first->leftnode);
printf("%d\t", first->element);
inorder(first->rightnode);
}
}
void postorder(NODE first)
{
if (first != NULL)
{
postorder(first->leftnode);
postorder(first->rightnode);
printf("%d\t", first->element);
}
}
void preorder(NODE first)
{
if (first != NULL)
{
printf("%d\t", first->element);
preorder(first->leftnode);
preorder(first->rightnode);
}
}
NODE createNode(int element)
{
NODE n = (NODE)malloc(sizeof(struct node));
n->element = element;
n->leftnode = NULL;
n->rightnode = NULL;
return n;
}
```

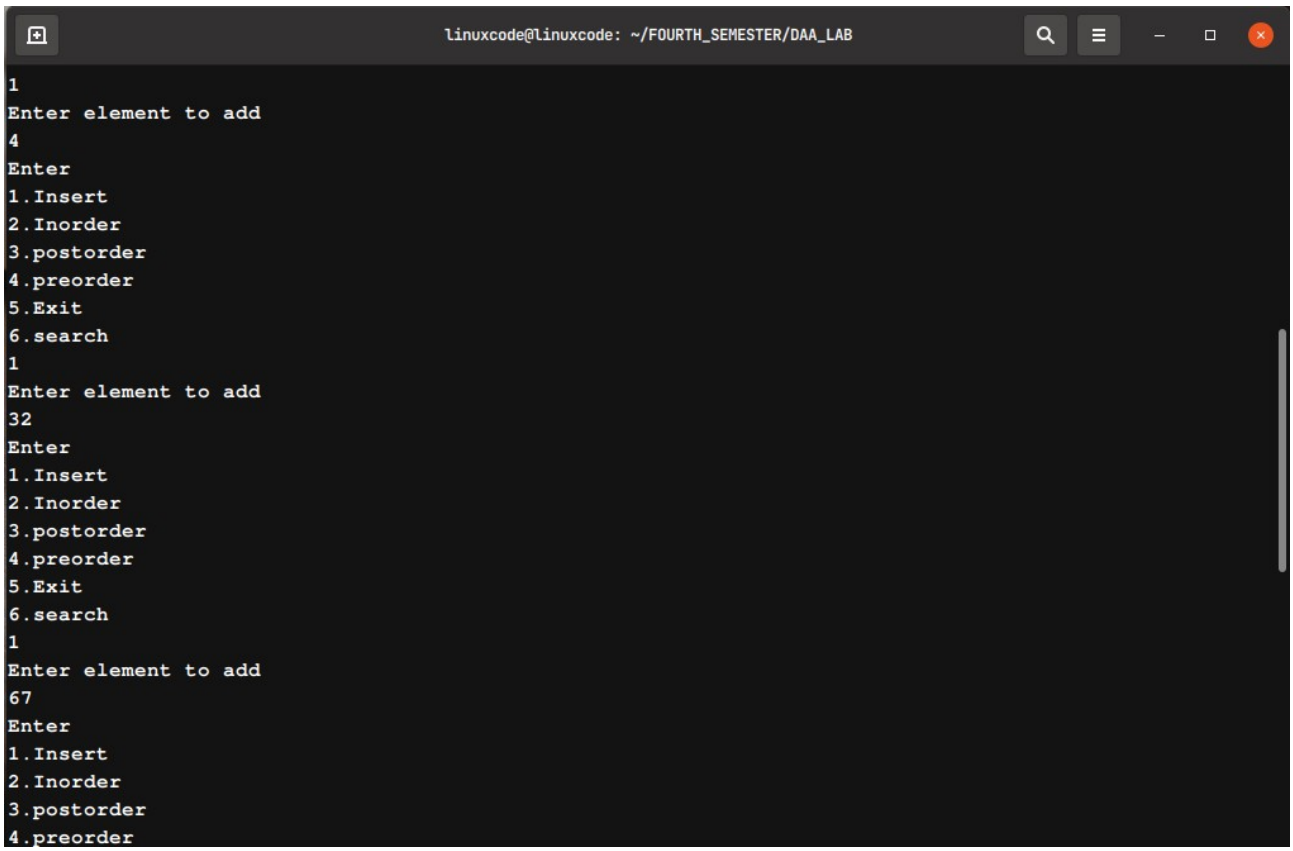
```
void search_func(NODE first1, int search, int *ptr)
{
NODE first = first1;
NODE temp;
while (first != NULL)

{
temp = first;
if (first->element == search)
{
*ptr = 1;
return;
}
else if (search > first->element)
first = first->rightnode;
else
first = first->leftnode;
}
if (*ptr == 0)
{
if (search < temp->element)
temp->leftnode = createNode(search);
else
temp->rightnode = createNode(search);
}
}
int main()
{
NODE first = NULL;
int element;
int choice;
int search;
int ptr = 0;
while (1)
{
printf("Enter\n1.Insert\n2.Inorder\n3.postorder\n4.preorder\n5.Exit\n6.search\n");
scanf("%d", &choice);
```

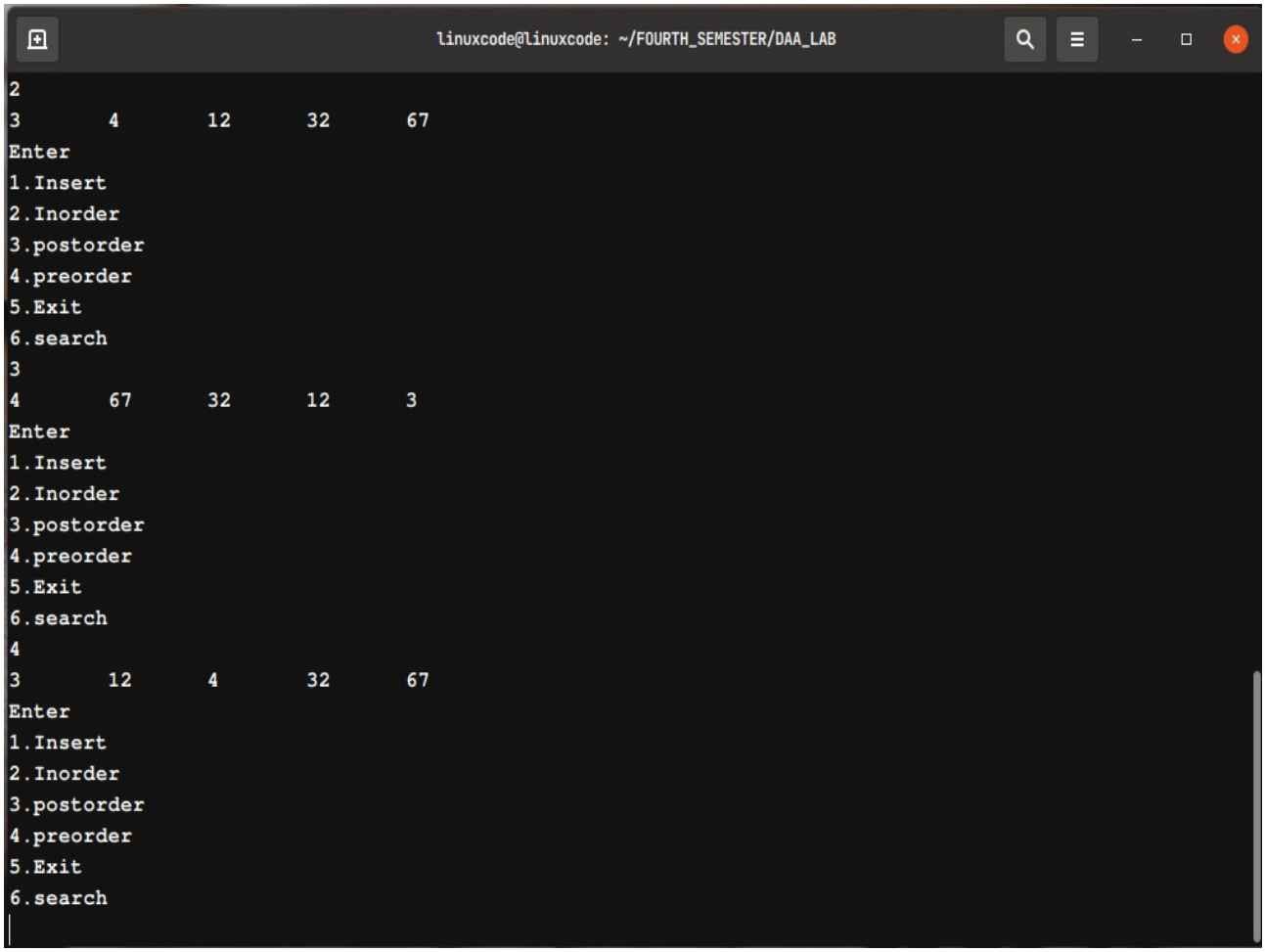
```
switch (choice)
{
case 1:
printf("Enter element to add \n");
scanf("%d", &element);
first = insert(first, element);
break;
case 2:
inorder(first);
printf("\n");
break;
case 3:
postorder(first);
printf("\n");
break;
case 4:
preorder(first);
printf("\n");
break;
case 5:
exit(0);
break;
case 6:
printf("Enter element to search\n");
scanf("%d", &search);
ptr = 0;
search_func(first, search, &ptr);
if (ptr == 1)
{
printf("Element found\n");
}
else
{
printf("Element not found\n");
}
break;
```

```
}
}
return 0;
}
```

OUTPUT:




linuxcode@linuxcode: ~/FOURTH_SEMESTER/DAA_LAB
1
Enter element to add
4
Enter
1.Insert
2.Inorder
3.postorder
4.preorder
5.Exit
6.search
1
Enter element to add
32
Enter
1.Insert
2.Inorder
3.postorder
4.preorder
5.Exit
6.search
1
Enter element to add
67
Enter
1.Insert
2.Inorder
3.postorder
4.preorder


linuxcode@linuxcode: ~/FOURTH_SEMESTER/DAA_LAB
2
3 4 12 32 67
Enter
1.Insert
2.Inorder
3.postorder
4.preorder
5.Exit
6.search
3
4 67 32 12 3
Enter
1.Insert
2.Inorder
3.postorder
4.preorder
5.Exit
6.search
4
3 12 4 32 67
Enter
1.Insert
2.Inorder
3.postorder
4.preorder
5.Exit
6.search

2.Write a program to implement the following graph representations and display them.
i. Adjacency list
ii. Adjacency matrix

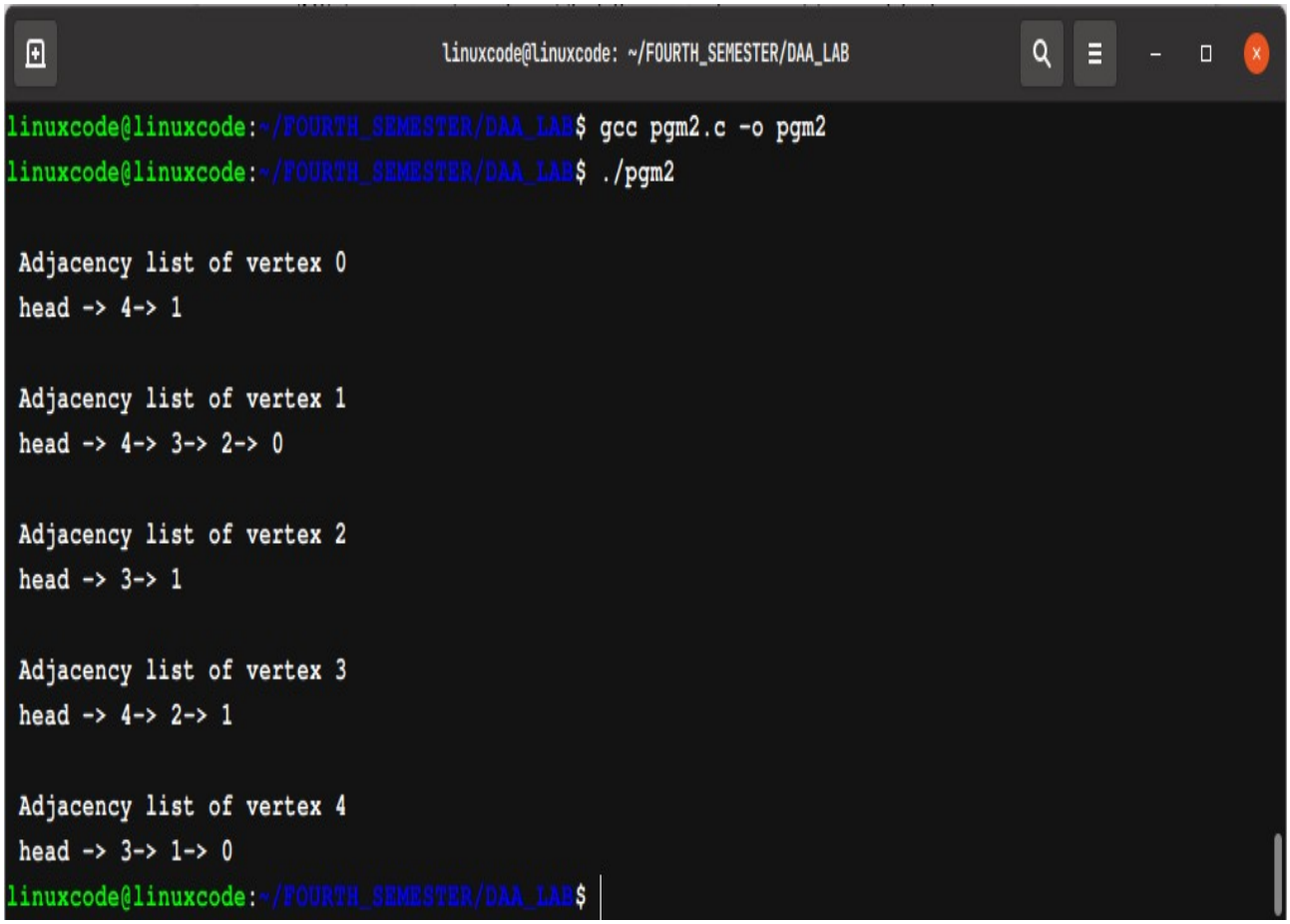
part1->Adjecency List

```
#include <stdio.h>
#include <stdlib.h>
struct adlistnode
{
int dest;
struct adlistnode *next;
};
struct adlist
{
struct adlistnode *head;
};
struct Graph
{
int V;
struct adlist *array;
};
struct adlistnode *newAdjListNode(int dest)
{
struct adlistnode *newNode = (struct adlistnode *)malloc(sizeof(struct adlistnode));
newNode->dest = dest;
newNode->next = NULL;
return newNode;
}
struct Graph *createGraph(int V)
{
struct Graph *graph = (struct Graph *)malloc(sizeof(struct Graph));
graph->V = V;
graph->array = (struct adlist *)malloc(V * sizeof(struct adlist));
for (int i = 0; i < V; i++)
```

```
graph->array[i].head = NULL;
return graph;
}
void addEdge(struct Graph *graph, int src, int dest)
{
struct adlistnode *newNode = newAdjListNode(dest);
newNode->next = graph->array[src].head;
graph->array[src].head = newNode;
newNode = newAdjListNode(src);
newNode->next = graph->array[dest].head;
graph->array[dest].head = newNode;
}
void printGraph(struct Graph *graph)
{
int v;
for (v = 0; v < graph->V; ++v)
{
struct adlistnode *pCrawl = graph->array[v].head;
printf("\n Adjacency list of vertex %d\n head ", v);
while (pCrawl)
{
printf("-> %d", pCrawl->dest);
pCrawl = pCrawl->next;
}
printf("\n");
}
}
int main()
{
int V = 5;
struct Graph *graph = createGraph(V);
addEdge(graph, 0, 1);
addEdge(graph, 0, 4);
addEdge(graph, 1, 2);
addEdge(graph, 1, 3);
addEdge(graph, 1, 4);
addEdge(graph, 2, 3);
```

```
addEdge(graph, 3, 4);
printGraph(graph);
return 0;
}
```

OUTPUT :

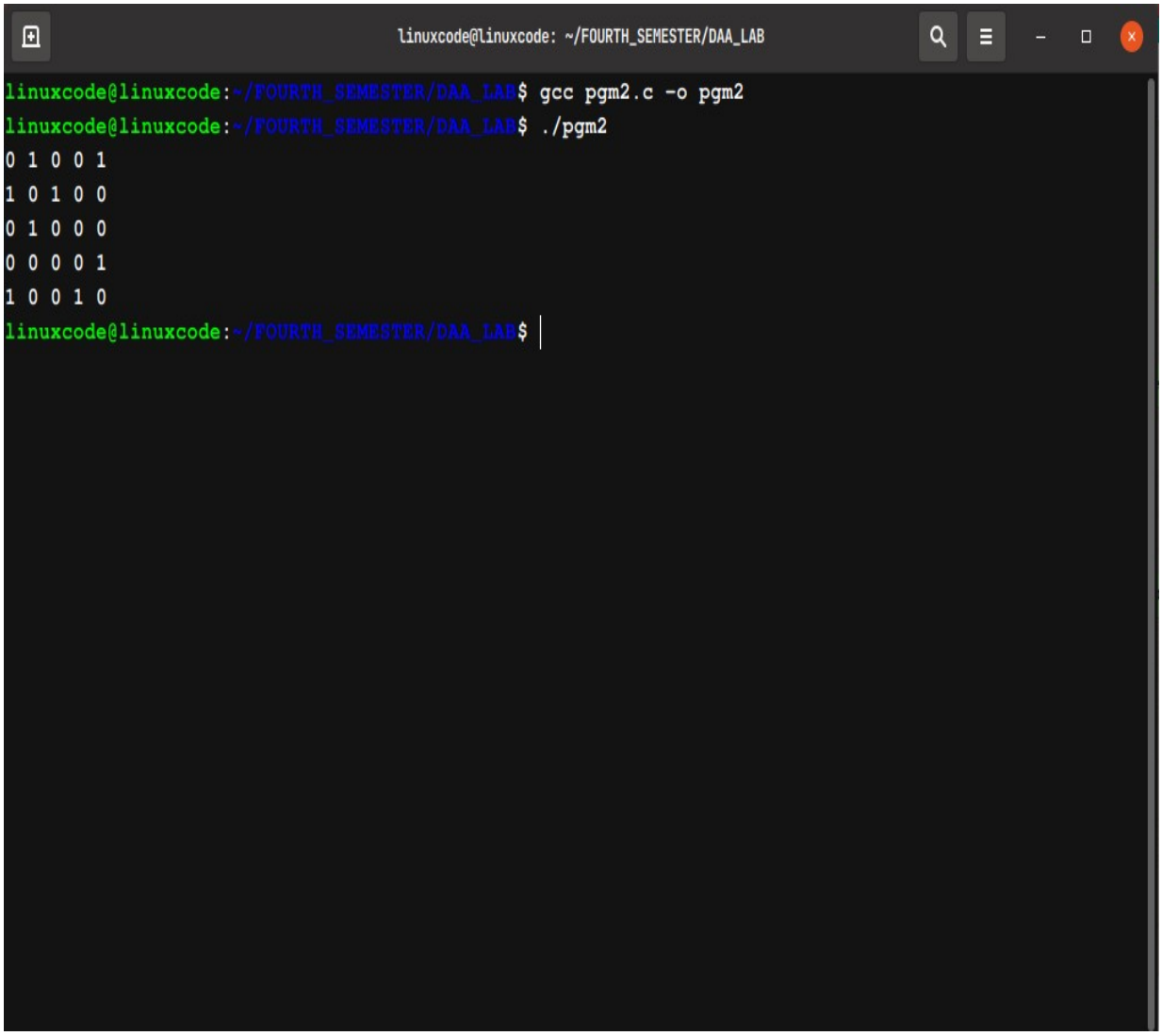


part2->adjecency matrix

```
#include <stdio.h>
int n, m;
void createAdjecencyMatrix(int Adj[][n + 1], int arr[][2])
{
for (int i = 0; i < n + 1; i++)
{
for (int j = 0; j < n + 1; j++)
{
Adj[i][j] = 0;
```

```
}
}
for (int i = 0; i < m; i++)
{
int x = arr[i][0];
int y = arr[i][1];
Adj[x][y] = 1;
Adj[y][x] = 1;
}
}
void printAdjecencyMatrix(int Adj[][n + 1])
{
for (int i = 1; i < n + 1; i++)
{
for (int j = 1; j < n + 1; j++)
{
printf("%d ", Adj[i][j]);
}
printf("\n");
}
}
int main()
{
n = 5;
int arr[][2] = {{1, 2}, {2, 3}, {4, 5}, {1, 5}};
m = sizeof(arr) / sizeof(arr[0]);
int Adj[n + 1][n + 1];
createAdjecencyMatrix(Adj, arr);
printAdjecencyMatrix(Adj);
return 0;
}
```

OUTPUT :


linuxcode@linuxcode: ~/FOURTH_SEMESTER/DAA_LAB
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB$ gcc pgm2.c -o pgm2
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB$ ./pgm2
0 1 0 0 1
1 0 1 0 0
0 1 0 0 0
0 0 0 0 1
1 0 0 1 0
linuxcode@linuxcode:~/FOURTH_SEMESTER/DAA_LAB$