DSL Assignment 1

```
/*
Assignment Title: 1. Write a program to implement stack using arrays.
Name:
Roll Number:
Batch: S1
Academic Year: 2023-2024
*/

#include<stdio.h>
#include<conio.h>
#define SIZE 10

void push(int);
void pop();
void display();
int stack[SIZE], top = -1;
void main()
{
 int value, choice;
 //clrscr();
 while(1){
 printf("\n\n***** MENU *****\n");
 printf("1. Push\n2. Pop\n3. Display\n4. Exit");
 printf("\nEnter your choice: ");
 scanf("%d",&choice);
 switch(choice){
case 1: printf("Enter the value to be insert: ");
scanf("%d",&value);
push(value);
break;
case 2: pop();
break;
case 3: display();
break;
 case 4: exit(0);
default: printf("\nWrong selection!!! Try again!!!");
 }
 }
}
void push(int value){
 if(top == SIZE-1)
 printf("\nStack is Full!!! Insertion is not possible!!!");
else{
 top++;
 stack[top] = value;
 printf("\nInsertion success!!!");
 }
}
void pop(){
 if(top == -1)
 printf("\nStack is Empty!!! Deletion is not possible!!!");
else{
 printf("\nDeleted : %d", stack[top]);
 top--;
 }
}
void display(){
 if(top == -1)

 printf("\nStack is Empty!!!");
 else{
 int i;
 printf("\nStack elements are:\n");
for(i=top; i>=0; i--)
printf("%d\n",stack[i]);
 }
}


/*
OUTPUT

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to be insert: 2

Insertion success!!!

***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are: 2


***** MENU *****
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 8.435 s
Press any key to continue.
*/
```

DSL Practical 2

```
/*
Practical Title: 2. Write a program to evaluate a given postfix
expression using stacks.
Name:
Roll Number:
Batch: S1
Academic Year: 2023-2024
*/

#include<stdio.h>
int stack[20]; int
top = -1; void
push(int x)
```

```
{
 stack[++top] = x;
}
int pop()
{
 return stack[top--];
}
int main()
{
 char exp[20];
 char *e;
 int n1,n2,n3,num;
 printf("Enter the expression :: ");
scanf("%s",exp);
 e = exp;
 while(*e != '\0')
 {
 if(isdigit(*e))
 {
 num = *e - 48;
 push(num);
 }
 else
 {
 n1 = pop();
 n2 = pop();
 switch(*e)
 {
 case '+':
 {
 n3 = n1 + n2;
 break;
 }
 case '-':
 {
 n3 = n2 - n1;
 break;
 }
 case '*':
 {
 n3 = n1 * n2;
 break;
 }
 case '/':
 {

 n3 = n2 / n1;
 break;
 }
 }
 push(n3);
 }
 e++;
 }
 printf("\nThe result of expression %s = %d\n\n",exp,pop());
return 0;
}

/*
OUTPUT
```

t n

```
Enter the expression :: 23+

The result of expression 23+ = 5


Process returned 0 (0x0)   execution time : 4.765 s
Press any key to continue.
*/
```

m y

t n

DSL Practical 3

```
/*
Practical Title: 3. Write a program to implement circular queue using
arrays.
Name:
Roll Number:
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
#define size 5
void insertq(int[], int);
void deleteq(int[]);
void display(int[]);
int front = - 1;
int rear = - 1;
int main()
{
 int n, ch;
 int queue[size];
 do
 {
 printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0.
Exit");
 printf("\nEnter Choice 0-3? : ");
 scanf("%d", &ch);
 switch (ch)
 {
 case 1:
 printf("\nEnter number: ");
 scanf("%d", &n);
 insertq(queue, n);
 break;
 case 2:
 deleteq(queue);
 break;
 case 3:
 display(queue);
 break;
 }
 }while (ch != 0);
}
void insertq(int queue[], int item)
{
 if ((front == 0 && rear == size - 1) || (front == rear + 1))
 {
 printf("queue is full");
 return;
 }
 else if (rear == - 1)
 {
 rear++;
 front++;
 }
 else if (rear == size - 1 && front > 0)
 {
 rear = 0;
```

m y

```
 }
 else
 {
 rear++;
 }
 queue[rear] = item;
}
void display(int queue[])
{
 int i;
 printf("\n");
 if (front > rear)
 {
 for (i = front; i < size; i++)
 {
 printf("%d ", queue[i]);
 }
 for (i = 0; i <= rear; i++)
 printf("%d ", queue[i]);
 }
 else
 {
 for (i = front; i <= rear; i++)
 printf("%d ", queue[i]);
 }
}
void deleteq(int queue[])
{
 if (front == - 1)
 {
 printf("Queue is empty ");
 }
 else if (front == rear)
 {
 printf("\n %d deleted", queue[front]);
front = - 1;
 rear = - 1;
 }
 else
 {
 printf("\n %d deleted", queue[front]);
front++;
 }
}


/*
OUTPUT

 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1

Enter number: 6
 Circular Queue:
1. Insert
```

```
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 3

6

 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? :
*/
```

DSL Assignment 4

```
/*
Assignment Title: 4.Write a Program To Implement Double Ended
Queue(Dequeue Using Arrays).
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include<stdio.h>

#define MAX 30
typedef struct dequeue
{
int data[MAX];
int rear,front;
}dequeue;
void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void   enqueueR(dequeue   *p,int   x);
void   enqueueF(dequeue   *p,int   x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);
void main()
{

int i,x,op,n;
dequeue q;
initialize(&q);
do
{ printf("\n1.Create\n2.Insert(rear\n3.Insert(front\n4.Delete(rear");
printf("\n5.Delete(front)\n6.Print\n7.Exit\nEnter your choice:");
scanf("%d",&op);
switch(op)
{
case 1: printf("\nEnter number of elements:");
scanf("%d",&n);
initialize(&q);
printf("\nEnter the data:");
for(i=0;i<n;i++)
{
scanf("%d",&x);
if(full(&q))
{
printf("\nQueue is full!!");
exit(0);
}
enqueueR(&q,x);
}
break;
case 2: printf("\nEnter element to be inserted:");
scanf("%d",&x);
if(full(&q))
{
printf("\nQueue is full!!");
exit(0);
}
enqueueR(&q,x);
break;
case 3: printf("\nEnter the element to be inserted:");
scanf("%d",&x);
if(full(&q))
{
printf("\nQueue is full!!");
exit(0);
}
enqueueR(&q,x);
break;
case 4: if(empty(&q))
{
printf("\nQueue is empty!!");
exit(0);
}
x=dequeueR(&q);
printf("\nElement deleted is %d\n",x);
break;
case 5: if(empty(&q))
{
printf("\nQueue is empty!!");
exit(0);
}
x=dequeueF(&q);
printf("\nElement deleted is %d\n",x);
break;
case 6: print(&q);
break;
default: break;
}
}while(op!=7);
getch();
}
void initialize(dequeue *P)
{
P->rear=-1;
P->front=-1;
}
int empty(dequeue *P)
{
if(P->rear==-1)
return(1);
return(0);
}
int full(dequeue *P)
{
if((P->rear+1)%MAX==P->front)
return(1);
return(0);
}
void enqueueR(dequeue *P,int x)
{
if(empty(P))
{
P->rear=0;
P->front=0;
P->data[0]=x;
```

```
}
else
{
P->rear=(P->rear+1)%MAX;
P->data[P->rear]=x;
}
void enqueueF(dequeue *P,int x)
{
if(empty(P))
{
P->rear=0;
P->front=0;
P->data[0]=x;
}
else
{
P->front=(P->front-1+MAX)%MAX;
P->data[P->front]=x;
}
}
}
int dequeueF(dequeue *P)
{
int x;
   x =P->data[P->front];
   if(P->rear==P->front) //delete the last element
initialize(P);
else
P->front=(P->front+1)%MAX;
return(x);
}

int dequeueR(dequeue*P)
{
int x;
x=P->data[P->rear];
if(P->rear==P->front)
initialize(P);
else
P->rear=(P->rear-1+MAX)%MAX;
return(x);
}
void print(dequeue *P)
{
if(empty(P))
{
printf("\nQueue is empty!!");
exit(0);
}
int i;
i=P->front;
while(i!=P->rear)
{
printf("\n%d",P->data[i]);
i=(i+1)%MAX;
}
    printf("\n%d\n",P->data[P->rear]); }
/*
OUTPUT
```

```
1.Create
2.Insert(rear
3.Insert(front
4.Delete(rear
5.Delete(front)
6.Print
7.Exit
Enter your choice:1

Enter number of elements:1

Enter the data:55

1.Create
2.Insert(rear
3.Insert(front
4.Delete(rear
5.Delete(front)
6.Print
7.Exit
Enter your choice:2

Enter element to be inserted:1

1.Create
2.Insert(rear
3.Insert(front
4.Delete(rear
5.Delete(front)
6.Print
7.Exit
Enter your choice:3

Enter the element to be inserted:2

1.Create
2.Insert(rear
3.Insert(front
4.Delete(rear
5.Delete(front)
6.Print
7.Exit
Enter your choice:6

55 1 2

1.Create
2.Insert(rear
3.Insert(front
4.Delete(rear
5.Delete(front)
6.Print
7.Exit
Enter your choice:7
Process returned 13 (0xD)   execution time : 59.926 s
Press any key to continue.
*/
```

DSL Assignment 5A

```
/*
Assignment Title: 5A. Write programs to implement the following data
structures:(a) Single linked list.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct stud
{
char usn[11],name[15],branch[4],phno[11];
int sem;
struct stud *next;
}*f=NULL,*r=NULL,*t=NULL;
void ins(int ch)
{
t=(struct stud*)malloc(sizeof(struct stud));
printf("\nEnter USN:");
scanf("%s",t->usn);
printf("Enter Name:");
scanf("%s",t->name);
printf("Enter Branch:");
scanf("%s",t->branch);
printf("Enter Sem:");
scanf("%d",&t->sem);
printf("Enter Phno:");
scanf("%s",t->phno);
t->next=NULL;
if(!r)
f=r=t;
else
{
if(ch)
{
r->next=t;
r=t;
}
else
{
t->next=f;
f=t;
}
}
}
void del(int ch)
{
if(!f)
printf("\nList Empty");
else
{
struct stud *t1;
if(f==r)
```

```
{
t1=f;
f=r=NULL;
}
else if(ch)
{
t1=r;
for(t=f;t->next!=r;t=t->next)
r=t;
r->next=NULL;
}
else
{
t1=f;
f=f->next;
}
printf("\nElement deleted is:\n");
printf("USN:%s\nName:%s\nBranch:%s\nSem:%d\nPhno:%s\n",t1->usn,t1-
>name,t1->branch,t1->sem,t1->phno);
free(t1);
}
}
void disp()
{
if(!f)
printf("\nList Empty!!!");
else
printf("\nList elements are:\n");
for(t=f;t;t=t->next)
printf("\nUSN:%s\nName:%s\nBranch:%s\nSem:%d\nPhno:%s\n",t->usn,t-
>name,t->branch,t->sem,t->phno);
}
void main()
{
int ch,n,i;
printf("\n........Menu..........,\n");
printf("1.Create\n");
printf("2.Display\n");
printf("3.Insert at end\n");
printf("4.Delete at end\n");
printf("5.Insert at beg\n");
printf("6.Delete at beg\n");
printf("7.Exit\n");
while(1)
{
printf("\nEnter choice:");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter no. of nodes:");
scanf("%d",&n);
for(i=0;i<n;i++)
ins(0);
break;
case 2:disp();break;
case 3:ins(1);break;
case 4:del(1);break;
case 5:ins(0);break;
case 6:del(0);break;
case 7:exit(0); default:printf("\nInvalid
choice!!!!");
} } }


/*
OUTPUT

........Menu..........,
1.Create
2.Display
3.Insert   at   end
4.Delete   at   end
5.Insert   at   beg
6.Delete at beg
7.Exit

Enter choice:1 Enter

no. of nodes:1

Enter USN:3434
Enter      Name:Vivek
Enter      Branch:CSD
Enter Sem:2
Enter Phno:9876543210
List elements are:
USN:3434
Name:Vivek
Branch:CSD
Sem:2
Phno:9876543210
*/
```

DSL Assignment 5B

```
/*
Assignment Title: 5B. Program In C For The Following Operations On Doubly linked List.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
struct node
{
 char ssn[25],name[25],dept[10],designation[25];
 int sal;
 long int phone;
 struct node *llink;
 struct node *rlink;
};
typedef struct node* NODE;
NODE first = NULL;
int count=0;
NODE create()
{
 NODE enode;
 enode = (NODE)malloc(sizeof(struct node));
 if( enode== NULL)
 {
 printf("\nRunning out of memory");
 exit(0);
 }
 printf("\nEnter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee: \n");
 scanf("%s %s %s %s %d %ld", enode->ssn, enode->name, enode->dept, enode->designation, &enode->sal, &enode->phone);
 enode->llink=NULL;
 enode->rlink=NULL;
 count++;
 return enode;
}
NODE insertfront()
{
 NODE temp;
 temp = create();
 if(first == NULL)
 {
 return temp;
 }
 temp->rlink = first;
 first->llink = temp;
 return temp;
}
void display()
{
 NODE cur;
 int nodeno=1;
 cur = first;
 if(cur == NULL)
 printf("\nNo Contents to display in DLL");
while(cur!=NULL)
 {
printf("\nENode:%d||SSN:%s|Name:%s|Department:%s|Designation:%s|Salary:%d
|Phone no:%ld");
 (nodeno, cur->ssn, cur->name,cur->dept, cur->designation, cur->sal, cur-
>phone);
 cur = cur->rlink;
 nodeno++;
 }
 printf("\nNo of employee nodes is %d",count);
}
NODE deletefront()
{
 NODE temp;
 if(first == NULL)
 {
 printf("\nDoubly Linked List is empty");
 return NULL;
 }
 if(first->rlink== NULL)
 {
 printf("\nThe employee node with the ssn:%s is deleted", first->ssn);
free(first);
 count--;
 return NULL;
 }
 temp = first;
 first = first->rlink;
 temp->rlink = NULL;
 first->llink = NULL;
 printf("\nThe employee node with the ssn:%s is deleted",temp->ssn);
 free(temp);
 count--;
 return first;
}
NODE insertend()
{
 NODE cur, temp;
 temp = create();
 if(first == NULL)
 {
 return temp;
 }
 cur= first;
 while(cur->rlink!=NULL)
 {
 cur = cur->rlink;
 }
 cur->rlink = temp;
 temp->llink = cur;
 return first;
}
NODE deleteend()
{
 NODE prev,cur;
```

```
if(first == NULL)  {
 printf("\nDoubly Linked List is empty");
 return NULL;
 }
 if(first->rlink == NULL)
 {
 printf("\nThe employee node with the ssn:%s is deleted",first->ssn);
 free(first);
 count--;
 return NULL;
 }
 prev=NULL;
 cur=first;
 while(cur->rlink!=NULL)
 {
 prev=cur;
 cur = cur->rlink;
 }
 cur->llink = NULL;
 printf("\nThe employee node with the ssn:%s is deleted",cur->ssn);
 free(cur);
 prev->rlink = NULL;
 count--;
 return first;
}
void deqdemo()
{
 int ch;
 while(1)
 {
 printf("\nDemo Double Ended Queue Operation");
 printf("\n1:InsertQueueFront\n 2: DeleteQueueFront\n3:InsertQueueRear\n
4:DeleteQueueRear\n 5:DisplayStatus\n 6: Exit \n");
 scanf("%d", &ch);
 switch(ch)
 {
 case 1: first=insertfront();
 break;
 case 2: first=deletefront();
 break;
 case 3: first=insertend();
 break;
 case 4: first=deleteend();
 break;
 case 5: display();
 break;
 default : return;
 }
 }
}
void main()
{
 int ch,i,n;
 while(1)
 {
 printf("\n\n~~~Menu~~~");
 printf("\n1:Create DLL of Employee Nodes");
 printf("\n2:DisplayStatus");
printf("\n3:InsertAtEnd");
```

```
printf("\n4:DeleteAtEnd");
printf("\n5:InsertAtFront");
 printf("\n6:DeleteAtFront");
 printf("\n7:Double Ended Queue Demo using DLL");
 printf("\n8:Exit \n");
 printf("\nPlease enter your choice: ");
 scanf("%d",&ch);
 switch(ch)
 {
 case 1 : printf("\nEnter the no of Employees: ");
scanf("%d",&n);
 for(i=1;i<=n;i++)
 first = insertend();
 break;
 case 2: display();
 break;
 case 3: first = insertend();
 break;
 case 4: first = deleteend();
 break;
 case 5: first = insertfront();
 break;
 case 6: first = deletefront();
 break;
 case 7: deqdemo();
 break;
 case 8 : exit(0);
 default: printf("\nPlease Enter the valid choice");  }
 }
}


/*
OUTPUT
~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd
4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 1

Enter the no of Employees: 1

Enter the ssn,Name,Department,Designation,Salary,PhoneNo of the employee:
5244
Vivek
CSD
Head
50000
9876543210
~~~Menu~~~
1:Create DLL of Employee Nodes
2:DisplayStatus
3:InsertAtEnd
```

```
4:DeleteAtEnd
5:InsertAtFront
6:DeleteAtFront
7:Double Ended Queue Demo using DLL
8:Exit

Please enter your choice: 2

Process returned -1073741819 (0xC0000005)   execution time : 38.489 s
Press any key to continue.
*/
```

DSL Assignment 6

```
/*
Assignment Title: 6. Write a program to Binary Search Tree(BST).
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include<stdio.h>
#include<stdlib.h>
struct node {
 int key;
 struct node *left, *right;
};
// A utility function to create a new BST node
struct node *newNode(int item) {
 struct node *temp = (struct node *) malloc(sizeof(struct node));
temp->key = item;
 temp->left = temp->right = NULL;
 return temp;
}
// A utility function to do inorder traversal of BST
void inorder(struct node *root) {
 if (root != NULL) {
 inorder(root->left);
 printf("%d ", root->key);
 inorder(root->right);
 }
}
/* A utility function to insert a new node with given key in BST */
struct node* insert(struct node* node, int key) {
 /* If the tree is empty, return a new node */
 if (node == NULL)
 return newNode(key);
 /* Otherwise, recur down the tree */
 if (key < node->key)
 node->left = insert(node->left, key);
 else if (key > node->key)
 node->right = insert(node->right, key);
 /* return the (unchanged) node pointer */
 return node;
}
// Driver Program to test above functions
int main() {
 /* Let us create following BST
 50
 / \
 30 70
/ \ / \
20 40 60 80 */
 struct node *root = NULL;
 root = insert(root, 50);
 insert(root, 30);
 insert(root, 20);
 insert(root, 40);
 insert(root, 70);
 insert(root, 60);
 insert(root, 80);
 // print inoder traversal of the BST
inorder(root);
 printf("\nis the created BST:");
return 0;
}


/*
OUTPUT
20 30 40 50 60 70 80
is the created BST:
Process returned 0 (0x0)   execution time : 0.340 s
Press any key to continue.
*/
```

DSL Assignment 7A

```
/*
Assignment Title: 7A.Implement The Following Sorting Algorithm:
A)Insertion Sort.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
// Function to print an array
void printArray(int array[], int size) {
 for (int i = 0; i < size; i++) {
 printf("%d ", array[i]);
 }
 printf("\n");
}
void insertionSort(int array[], int size) {
 for (int step = 1; step < size; step++) {
 int key = array[step];
 int j = step - 1;
 // Compare key with each element on the left of it until an element
smaller
//than
 // it is found.
 // For descending order, change key<array[j] to key>array[j].
 while (key < array[j] && j >= 0) {
 array[j + 1] = array[j];
 --j;
 }
 array[j + 1] = key;
 }
}
// Driver code
int main() {
 int data[] = {9, 5, 1, 4, 3};
 int size = sizeof(data) / sizeof(data[0]);
 insertionSort(data, size);
 printf("Sorted array in ascending order:\n");
  printArray(data, size);
}


/*
OUTPUT

Sorted array in ascending order: 1 3
4 5 9

Process returned 0 (0x0)   execution time : 0.298 s
Press any key to continue.
*/
/*
Assignment Title: 7B. Implement The Following Sorting Algorithm:B)Merge
Sort.
Name:
Roll Number: 06
```

```
DSL Assignment 7B




Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
// Merge two subarrays L and M into arr
void merge(int arr[], int p, int q, int r) {
 // Create L ← A[p..q] and M ← A[q+1..r]
 int n1 = q - p + 1;
 int n2 = r - q;
 int L[n1], M[n2];
 for (int i = 0; i < n1; i++)
 L[i] = arr[p + i];
 for (int j = 0; j < n2; j++)
 M[j] = arr[q + 1 + j];
 // Maintain current index of sub-arrays and main array
 int i, j, k;
 i = 0;
 j = 0;
 k = p;
 // Until we reach either end of either L or M, pick larger among
 // elements L and M and place them in the correct position at A[p..r]
while (i < n1 && j < n2) {
 if (L[i] <= M[j]) {
 arr[k] = L[i];
 i++;
 } else {
 arr[k] = M[j];
 j++;
 }
 k++;
 }
 // When we run out of elements in either L or M,
 // pick up the remaining elements and put in A[p..r]
 while (i < n1) {
 arr[k] = L[i];
 i++;
 k++;
 }
 while (j < n2) {
 arr[k] = M[j];
 j++;
 k++;
 }
}
// Divide the array into two subarrays, sort them and merge them
void mergeSort(int arr[], int l, int r) {
 if (l < r) {
 // m is the point where the array is divided into two subarrays
 int m = l + (r - l) / 2;
 mergeSort(arr, l, m);
 mergeSort(arr, m + 1, r);
 // Merge the sorted subarrays
 merge(arr, l, m, r);
 }
}
// Print the array
void printArray(int arr[], int size) {
 for (int i = 0; i < size; i++)
 printf("%d ", arr[i]);
 printf("\n");
}
// Driver program
int main() {
 int arr[] = {6, 5, 12, 10, 9, 1};
 int size = sizeof(arr) / sizeof(arr[0]);
mergeSort(arr, 0, size - 1);
 printf("Sorted array: \n");
 printArray(arr, size);
}


/*
OUTPUT

Sorted array: 1 5
6 9 10 12

Process returned 0 (0x0)   execution time : 0.278 s
Press any key to continue.
*/
```

```
/*
Assignment Title: 7C.Implement The Following Sorting Algorithm: C)Quick
Sort.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
// function to swap elements
void swap(int *a, int *b) {
 int t = *a;
 *a = *b;
 *b = t;
}
// function to find the partition position
int partition(int array[], int low, int high) {

 // select the rightmost element as pivot
int pivot = array[high];

 // pointer for greater element
 int i = (low - 1);
 // traverse each element of the array
// compare them with the pivot
 for (int j = low; j < high; j++) {
 if (array[j] <= pivot) {

 // if element smaller than pivot is found
 // swap it with the greater element pointed by i
i++;

 // swap element at i with element at j
 swap(&array[i], &array[j]);
 }
 }
 // swap the pivot element with the greater element at i
swap(&array[i + 1], &array[high]);

 // return the partition point
 return (i + 1);
}
void quickSort(int array[], int low, int high) {
if (low < high) {

 // find the pivot element such that
 // elements smaller than pivot are on left of pivot
// elements greater than pivot are on right of pivot
int pi = partition(array, low, high);

 // recursive call on the left of pivot
quickSort(array, low, pi - 1);

 // recursive call on the right of pivot
quickSort(array, pi + 1, high);
 }
}
```

```
// function to print array elements void
printArray(int array[], int size) {   for
(int i = 0; i < size; ++i) {
 printf("%d ", array[i]);
 }
 printf("\n");
}
// main function
int main() {
 int data[] = {8, 7, 2, 1, 0, 9, 6};

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

 printf("Unsorted Array\n");
printArray(data, n);

 // perform quicksort on data
quickSort(data, 0, n - 1);

 printf("Sorted array in ascending order: \n");
printArray(data, n);
}


/*
OUTPUT

Unsorted Array
8 7 2 1 0 9 6
Sorted array in ascending order: 0 1
2 6 7 8 9

Process returned 0 (0x0)   execution time : 0.009 s
Press any key to continue.
*/
```

DSL Assignment 7D

```
/*
Assignment Title: 7D.Implement The Following Sorting Algorithm: D)Heap
Sort.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>

 // Function to swap the the position of two elements
void swap(int *a, int *b) {
 int temp = *a;
 *a = *b;
 *b = temp;
 }

 void heapify(int arr[], int n, int i) {
 // Find largest among root, left child and right child
int largest = i;
 int left = 2 * i + 1;
 int right = 2 * i + 2;

 if (left < n && arr[left] > arr[largest])
largest = left;

 if (right < n && arr[right] > arr[largest])
largest = right;

 // Swap and continue heapifying if root is not largest
if (largest != i) {
 swap(&arr[i], &arr[largest]);
 heapify(arr, n, largest);
 }
 }

 // Main function to do heap sort
 void heapSort(int arr[], int n) {
 // Build max heap
 for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);

 // Heap sort
 for (int i = n - 1; i >= 0; i--) {
swap(&arr[0], &arr[i]);

 // Heapify root element to get highest element at root again
heapify(arr, i, 0);
 }
 }

 // Print an array
 void printArray(int arr[], int n) {
for (int i = 0; i < n; ++i)
 printf("%d ", arr[i]);
 printf("\n");
 }
 // Driver code
 int main() {
 int arr[] = {1, 12, 9, 5, 6, 10};
 int n = sizeof(arr) / sizeof(arr[0]);
heapSort(arr, n);

 printf("Sorted array is \n");
printArray(arr, n);
   printArray(arr, n);
 }


/*
OUTPUT

Sorted array is 1 5
6 9 10 12 1 5 6 9
10 12

Process returned 0 (0x0)   execution time : 0.274 s
Press any key to continue.
*/
```

t n

DSL Assignment 8A

```
/*
Assignment Title: 8A.Write A Program For Implementation Of Graph
Traversals By Applying:A)BFS.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
#include <stdlib.h>
#define SIZE 40
struct queue {
 int items[SIZE];
 int front;
 int rear;
};
struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);
struct node {
 int vertex;
 struct node* next;
};
struct node* createNode(int);
struct Graph {
 int numVertices;
 struct node** adjLists;
 int* visited;
};
// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
 struct queue* q = createQueue();
 graph->visited[startVertex] = 1;
 enqueue(q, startVertex);
 while (!isEmpty(q)) {
 printQueue(q);
 int currentVertex = dequeue(q);
 printf("Visited %d\n", currentVertex);
 struct node* temp = graph->adjLists[currentVertex];
while (temp) {
 int adjVertex = temp->vertex;
 if (graph->visited[adjVertex] == 0) {
 graph->visited[adjVertex] = 1;
 enqueue(q, adjVertex);
 }
 temp = temp->next;
 }
 }
}
// Creating a node
struct node* createNode(int v) {
 struct node* newNode = malloc(sizeof(struct node));
newNode->vertex = v;
 newNode->next = NULL;
```

m y

```
 return newNode;
}
// Creating a graph
struct Graph* createGraph(int vertices) {
 struct Graph* graph = malloc(sizeof(struct Graph));
 graph->numVertices = vertices;
 graph->adjLists = malloc(vertices * sizeof(struct node*));
graph->visited = malloc(vertices * sizeof(int));
 int i;
 for (i = 0; i < vertices; i++) {
 graph->adjLists[i] = NULL;
 graph->visited[i] = 0;
 }
 return graph;
}
// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
 // Add edge from src to dest
 struct node* newNode = createNode(dest);
 newNode->next = graph->adjLists[src];
 graph->adjLists[src] = newNode;
 // Add edge from dest to src
 newNode = createNode(src);
 newNode->next = graph->adjLists[dest];
 graph->adjLists[dest] = newNode;
}
// Create a queue
struct queue* createQueue() {
 struct queue* q = malloc(sizeof(struct queue));
 q->front = -1;
 q->rear = -1;
 return q;
}
// Check if the queue is empty
int isEmpty(struct queue* q) {
 if (q->rear == -1)
 return 1;
 else
 return 0;
}
// Adding elements into queue
void enqueue(struct queue* q, int value) {
 if (q->rear == SIZE - 1)
 printf("\nQueue is Full!!");
 else {
 if (q->front == -1)
 q->front = 0;
 q->rear++;
 q->items[q->rear] = value;
 }
}
// Removing elements from queue
int dequeue(struct queue* q) {
 int item;
 if (isEmpty(q)) {
 printf("Queue is empty");
 item = -1;
 } else {
 item = q->items[q->front];
 q->front++;
 if (q->front > q->rear) {
 printf("Resetting queue ");
 q->front = q->rear = -1;
 }
 }
 return item;
}
// Print the queue
void printQueue(struct queue* q) {
 int i = q->front;
 if (isEmpty(q)) {
 printf("Queue is empty");
 } else {
 printf("\nQueue contains \n");
 for (i = q->front; i < q->rear + 1; i++) {
printf("%d ", q->items[i]);
 }
 }
}
int main() {
 struct Graph* graph = createGraph(6);
 addEdge(graph, 0, 1);
 addEdge(graph, 0, 2);
 addEdge(graph, 1, 2);
 addEdge(graph, 1, 4);
 addEdge(graph, 1, 3);
 addEdge(graph, 2, 4);
 addEdge(graph, 3, 4);
 bfs(graph, 0);
 return 0;
}


/*
OUTPUT

Queue contains
0 Resetting queue Visited 0

Queue contains 2 1
Visited 2

Queue contains 1 4
Visited 1

Queue contains 4 3
Visited 4

Queue contains
3 Resetting queue Visited 3

Process returned 0 (0x0)   execution time : 0.282 s
Press any key to continue.
*/
```

DSL Assignment 8B

```
/*
```

```
Assignment Title: 8B.Write A Program For Implementation Of Graph
Traversals By Applying:B)DFS.
Name:
Roll Number: 06
Batch: S1
Academic Year: 2023-2024
*/

#include <stdio.h>
#include <stdlib.h>
struct node {
 int vertex;
 struct node* next;
};
struct node* createNode(int v);
struct Graph {
 int numVertices;
 int* visited;
 // We need int** to store a two dimensional array.
 // Similary, we need struct node** to store an array of Linked lists
struct node** adjLists;
};
// DFS algo
void DFS(struct Graph* graph, int vertex) {
 struct node* adjList = graph->adjLists[vertex];
 struct node* temp = adjList;
 graph->visited[vertex] = 1;
 printf("Visited %d \n", vertex);
 while (temp != NULL) {
 int connectedVertex = temp->vertex;
 if (graph->visited[connectedVertex] == 0) {
 DFS(graph, connectedVertex);
 }
 temp = temp->next;
 }
}
// Create a node
struct node* createNode(int v) {
 struct node* newNode = malloc(sizeof(struct node));
 newNode->vertex = v;
 newNode->next = NULL;
 return newNode;
}
// Create graph
struct Graph* createGraph(int vertices) {
 struct Graph* graph = malloc(sizeof(struct Graph));
 graph->numVertices = vertices;
 graph->adjLists = malloc(vertices * sizeof(struct node*));
 graph->visited = malloc(vertices * sizeof(int));
 int i;
 for (i = 0; i < vertices; i++) {
 graph->adjLists[i] = NULL;
 graph->visited[i] = 0;
 }
 return graph;
}
// Add edge

void addEdge(struct Graph* graph, int src, int dest) {
// Add edge from src to dest
```

```
 struct node* newNode = createNode(dest);
 newNode->next = graph->adjLists[src];
 graph->adjLists[src] = newNode;
 // Add edge from dest to src
 newNode = createNode(src);
 newNode->next = graph->adjLists[dest];
 graph->adjLists[dest] = newNode;
}
// Print the graph
void printGraph(struct Graph* graph) {
 int v;
 for (v = 0; v < graph->numVertices; v++) {
 struct node* temp = graph->adjLists[v];
 printf("\n Adjacency list of vertex %d\n ", v);
while (temp) {
 printf("%d -> ", temp->vertex);
 temp = temp->next;
 }
 printf("\n");
 }
}
int main() {
 struct Graph* graph = createGraph(4);
 addEdge(graph, 0, 1);
 addEdge(graph, 0, 2);
 addEdge(graph, 1, 2);
 addEdge(graph, 2, 3);
 printGraph(graph);
 DFS(graph, 2);
 return 0;
}


/*
OUTPUT

Adjacency list of vertex 0  2
-> 1 ->

 Adjacency list of vertex 1  2
-> 0 ->

 Adjacency list of vertex 2  3
-> 1 -> 0 ->

 Adjacency list of vertex 3  2
->
Visited 2
Visited 3
Visited 1
Visited 0

Process returned 0 (0x0)   execution time : 0.006 s
Press any key to continue.
*/
```