Department of Computer Application

HINDU INSTITUTE OF MANAGEMENT

(Affiliated to DCRUST, Murthal | Approved by AICTE, New Delhi)

Data Strucures

BCA-104C

LAB FILE

(2020-2021)

for

1st Semester

Submitter To:

Mrs. Aruna

Submitter By:

Shubham Dahiya
BCA (1st Sem.)
20012041042

-Dynamic Memory Allocation using malloc

```
//Malloc Dynamic Memory Allocation
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int* ptr;
	int n, i;

	printf("Enter number of elements:");
	scanf("%d",&n);
	printf("Entered number of elements: %d\n", n);

		ptr = (int*)malloc(n * sizeof(int));

	if (ptr == NULL) {
		printf("Memory not allocated.\n");
		exit(0);
	}
	else {

		printf("Memory successfully allocated using malloc.\n");

		for (i = 0; i < n; ++i) {
			ptr[i] = i + 1;
```

```
                }

                printf("The elements of the array are: ");
                for (i = 0; i < n; ++i) {
                        printf("%d, ", ptr[i]);
                }
        }

        return 0;
}
```

OUTPUT:

Enter number of elements: 5

Memory successfully allocated using malloc.

The elements of the array are: 1, 2, 3, 4, 5,

-Dynamic Memory Allocation using calloc

```
//Calloc Dynamic Memory  Allocation
#include <stdio.h>
#include <stdlib.h>

int main()
{
        int* ptr;
        int n, i;

        n = 5;
        printf("Enter number of elements: %d\n", n);
```

```
        ptr = (int*)calloc(n, sizeof(int));



        if (ptr == NULL) {
                printf("Memory not allocated.\n");
                exit(0);
        }
        else {



                printf("Memory successfully allocated using calloc.\n");



                for (i = 0; i < n; ++i) {
                        ptr[i] = i + 1;
                }



                printf("The elements of the array are: ");
                for (i = 0; i < n; ++i) {
                        printf("%d, ", ptr[i]);
                }
        }

        return 0;
}
```

OUTPUT:

Enter number of elements: 5

Memory successfully allocated using calloc.

The elements of the array are: 1, 2, 3, 4, 5,

-Dynamic Memory Allocation using realloc

```
//Realloc Method
#include <stdio.h>
#include <stdlib.h>

int main()
{
	int* ptr;
	int n, i;

	n = 5;
	printf("Enter number of elements: %d\n", n);

		ptr = (int*)calloc(n, sizeof(int));

	if (ptr == NULL) {
		printf("Memory not allocated.\n");
		exit(0);
	}
	else {
```

```
printf("Memory successfully allocated using calloc.\n");

        for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
}

printf("The elements of the array are: ");
for (i = 0; i < n; ++i) {
        printf("%d, ", ptr[i]);
}

n = 10;
printf("\n\nEnter the new size of the array: %d\n", n);

ptr = realloc(ptr, n * sizeof(int));

printf("Memory successfully re-allocated using realloc.\n");

for (i = 5; i < n; ++i) {
        ptr[i] = i + 1;
}
```

```
        printf("The elements of the array are: ");
        for (i = 0; i < n; ++i) {
            printf("%d, ", ptr[i]);
        }

        free(ptr);
    }

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Memory successfully allocated using calloc.
The elements of the array are: 1, 2, 3, 4, 5,

Enter the new size of the array: 10
Memory successfully re-allocated using realloc.
The elements of the array are: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

free method - Dynamic Memory Allocation

```
//free Method
#include <stdio.h>
#include <stdlib.h>

int main()
```

```
{
        int *ptr, *ptr1;
        int n, i;


        n = 5;
        printf("Enter number of elements: %d\n", n);


        ptr = (int*)malloc(n * sizeof(int));

        ptr1 = (int*)calloc(n, sizeof(int));

        if (ptr == NULL || ptr1 == NULL) {
                printf("Memory not allocated.\n");
                exit(0);
        }
        else {


                printf("Memory successfully allocated using malloc.\n");


                free(ptr);
                printf("Malloc Memory successfully freed.\n");

                printf("\nMemory successfully allocated using calloc.\n");
```

```
        free(ptr1);
        printf("Calloc Memory successfully freed.\n");
    }

    return 0;
}
```

OUTPUT:

```
Enter number of elements: 5
Memory successfully allocated using malloc.
Malloc Memory successfully freed.

Memory successfully allocated using calloc.
Calloc Memory successfully freed.
```

-ARRAY Traversal

```
//Array Traversal

#include <stdio.h>

void printArray(int* arr, int n)
{
    int i;

    printf("Array: ");
```

```
	for (i = 0; i < n; i++) {
		printf("%d ", arr[i]);
	}
	printf("\n");
}

int main()
{
	int arr[] = { 2, -1, 5, 6, 0, -3 };
	int n = sizeof(arr) / sizeof(arr[0]);

	printArray(arr, n);

	return 0;
}
```

OUTPUT:

Array: 2 -1 5 6 0 -3

Array Insertion

```
//ARRAY Insertion:

#include <stdio.h>

int main()
{
```

```
    int arr[100] = { 0 };
    int i, x, pos, n = 10;


    for (i = 0; i < 10; i++)
        arr[i] = i + 1;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    x = 50;
    pos = 5;

        n++;


    for (i = n-1; i >= pos; i--)
        arr[i] = arr[i - 1];

    arr[pos - 1] = x;

    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

OUTPUT:

1 2 3 4 5 6 7 8 9 10

1 2 3 4 50 5 6 7 8 9 10

Array Deletion

```
//Array Deletion

#include <stdio.h>

int main()
{
  int array[100], position, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d elements\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  printf("Enter the location where you wish to delete element\n");
  scanf("%d", &position);

  if (position >= n+1)
    printf("Deletion not possible.\n");
  else
  {
    for (c = position - 1; c < n - 1; c++)
      array[c] = array[c+1];

    printf("Resultant array:\n");

    for (c = 0; c < n - 1; c++)
      printf("%d\n", array[c]);
  }

  return 0;
}
```

OUTPUT:

Enter number of elements in array:

5

Enter %d elements

15

16

17

18

18

19

Enter the location where you wish to delete element

3

Resultant array:

15

16

18

18

19

Array Linear Search

```
//Linear Search
#include<stdio.h>


int main()
{
   int a[20],i,x,n;
   printf("How many elements?");
   scanf("%d",&n);
```

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

    printf("\nEnter element to search:");
    scanf("%d",&x);

    for(i=0;i<n;++i)
        if(a[i]==x)
            break;

    if(i<n)
        printf("Element found at index %d",i);
    else
        printf("Element not found");

    return 0;
}
```

OUTPUT:

```
How many elements?6
Enter array elements:
38
65
34
63
```

24

64

Enter element to search

34

Element found at index 3

Binary Search

```
//Binary Search
#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)
{
	if (r >= l) {
		int mid = l + (r - l) / 2;


		if (arr[mid] == x)
			return mid;


		if (arr[mid] > x)
			return binarySearch(arr, l, mid - 1, x);


		return binarySearch(arr, mid + 1, r, x);
	}
```

```
        return -1;
}

int main(void)
{
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = sizeof(arr) / sizeof(arr[0]);
        int x = 10;
        int result = binarySearch(arr, 0, n - 1, x);
        (result == -1) ? printf("Element is not present in array")
                                : printf("Element is present at index %d",
                                                        result);
        return 0;
}
```

OUTPUT:

Element is present at index 3

Bubble Sort

```
//Bubble Sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
        int temp = *xp;
        *xp = *yp;
        *yp = temp;
```

```
}

void bubbleSort(int arr[], int n)
{
int i, j;
for (i = 0; i < n-1; i++)


        for (j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                        swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
        int i;
        for (i=0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

int main()
{
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        int n = sizeof(arr)/sizeof(arr[0]);
        bubbleSort(arr, n);
```

```
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

Sorted array:

11 12 22 25 34 64 90

Selection Sort

```
//Selection Sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;


    for (i = 0; i < n-1; i++)
    {
```

```
                min_idx = i;
                for (j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                        min_idx = j;


                swap(&arr[min_idx], &arr[i]);
        }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
        int i;
        for (i=0; i < size; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

int main()
{
        int arr[] = {64, 25, 12, 22, 11};
        int n = sizeof(arr)/sizeof(arr[0]);
        selectionSort(arr, n);
        printf("Sorted array: \n");
        printArray(arr, n);
```

```
	return 0;
}
```

OUTPUT:

Sorted array:

11 12 22 25 64

Insertion Sort

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>

void insertionSort(int arr[], int n)
{
	int i, key, j;
	for (i = 1; i < n; i++) {
		key = arr[i];
		j = i - 1;

		/* Move elements of arr[0..i-1], that are
		greater than key, to one position ahead
		of their current position */
		while (j >= 0 && arr[j] > key) {
			arr[j + 1] = arr[j];
			j = j - 1;
		}
		arr[j + 1] = key;
```

```
        }
}

void printArray(int arr[], int n)
{
        int i;
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

int main()
{
        int arr[] = { 12, 11, 13, 5, 6 };
        int n = sizeof(arr) / sizeof(arr[0]);

        insertionSort(arr, n);
        printArray(arr, n);

        return 0;
}
```

OUTPUT:

5 6 11 12 13

Merging of two Arrays

//MERGING of two arrays

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int a[10],b[10],c[20],m,n,o,i,j,k,temp;
  printf("Enter size of Array1\n");
  scanf("%d",&n);
  printf("Enter size of Array2\n");
  scanf("%d",&m);
  o=m+n; //size of third array
  printf("Enter Elements of Array1\n");
  for(i=0;i<n;i++){
    scanf("%d",&a[i]);
  }
  printf("Enter Elements of Array2\n");
  for(i=0;i<m;i++){
    scanf("%d",&b[i]);
  }
  //sorting first array
  for(i=0;i<n;i++){
    for(j=0;j<n-1-i;j++){
      if(a[j]>a[j+1]){
        temp=a[j];
        a[j]=a[j+1];
        a[j+1]=temp;
      }
    }
  }
```

```
//sorting second array
for(i=0;i<m;i++){
  for(j=0;j<m-1-i;j++){
    if(b[j]>b[j+1]){
      temp=b[j];
      b[j]=b[j+1];
      b[j+1]=temp;
    }
  }
}
printf("Elements of Array1\n");
for(i=0;i<n;i++){
  printf("a[%d]=%d\n",i,a[i]);
}
printf("Elements of Array2\n");
for(i=0;i<m;i++){
  printf("b[%d]=%d\n",i,b[i]);
}
j=0;
k=0;
for(i=0;i<o;i++){ // merging two arrays
  if(a[j]<=b[k]){
    c[i]=a[j];
    j++;
  }
  else{
    c[i]=b[k];
```

```
        k++;
      }
    }
    printf("Merged array is :\n");
    for(i=0;i<o;i++){
      printf("c[%d]=%d\n",i,c[i]);
    }
}
```

OUTPUT:

```
Enter Elements of Array1
1
2
3
4
Enter Elements of Array2
6
8
3
Elements of Array1
a[0]=1
a[1]=2
a[2]=3
a[3]=4
Elements of Array2
b[0]=3
b[1]=6
```

b[2]=8

Merged array is:

c[0]=1

c[1]=2

c[2]=3

c[3]=3

c[4]=4

c[5]=6

c[6]=8

Matrix Addition

```
//Matrix Addition
#include<stdio.h>

int main()
{
int a[5][5],b[5][5],c[5][5],i,j,m,n;
printf("How many rows and columns?");
scanf("%d%d",&m,&n);
printf("\nEnter first matrix:\n");
for(i=0;i<m;++i)
for(j=0;j<n;++j)
scanf("%d",&a[i][j]);
printf("\nEnter second matrix:\n");
for(i=0;i<m;++i)
for(j=0;j<n;++j)
scanf("%d",&b[i][j]);
```

```
printf("\nMatrix after addition:\n");

for(i=0;i<m;++i)

{

for(j=0;j<n;++j)

{

c[i][j]=a[i][j]+b[i][j];

printf("%d ",c[i][j]);

}

printf("\n");

}


return 0;

}
```

OUTPUT:

```
How many rows and columns?3
3

Enter first matrix:
2 6 9
3 2 0
2 4 1

Enter second matrix:
3 4 1
6 7 9
11 3 5

Matrix after addition:
5 10 10
9 9 9
13 7 6
```

Matrix Multiplication

```
//Mulitplication of Matrix
#include<stdio.h>
#include<stdlib.h>
int main(){
int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;
system("cls");
printf("enter the number of row=");
scanf("%d",&r);
printf("enter the number of column=");
scanf("%d",&c);
printf("enter the first matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter the second matrix element=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
scanf("%d",&b[i][j]);
}
```

```
}

printf("multiply of the matrix=\n");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
mul[i][j]=0;
for(k=0;k<c;k++)
{
mul[i][j]+=a[i][k]*b[k][j];
}
}
}
//for printing result
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
printf("%d\t",mul[i][j]);
}
printf("\n");
}
return 0;
}
```

OUTPUT:

enter the number of row=3

enter the number of column=3

enter the first matrix element=

1 1 1

2 2 2

3 3 3

enter the second matrix element=

1 1 1

2 2 2

3 3 3

multiply of the matrix=

6 6 6

12 12 12

18 18 18

Linked List

```
//Linked List
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;         // Data
   struct node *next; // Address
}*head;



void createList(int n);
```

```
void traverseList();



int main()
{
   int n;

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

   createList(n);

   printf("\nData in the list \n");
   traverseList();

   return 0;
}

void createList(int n)
{
   struct node *newNode, *temp;
   int data, i;

   head = (struct node *)malloc(sizeof(struct node));



   if(head == NULL)
```

```
{
   printf("Unable to allocate memory.");
   exit(0);
}


printf("Enter the data of node 1: ");
scanf("%d", &data);

head->data = data; // Link data field with data
head->next = NULL; // Link address field to NULL


temp = head;
for(i=2; i<=n; i++)
{
   newNode = (struct node *)malloc(sizeof(struct node));

         if(newNode == NULL)
   {
      printf("Unable to allocate memory.");
      break;
   }

   printf("Enter the data of node %d: ", i);
   scanf("%d", &data);
```

```
        newNode->data = data; // Link data field of newNode
        newNode->next = NULL; // Make sure new node points to NULL

        temp->next = newNode; // Link previous node with newNode
        temp = temp->next;    // Make current node as previous node
    }
}

void traverseList()
{
    struct node *temp;

    if(head == NULL)
    {
        printf("List is empty.");
        return;
    }

    temp = head;
    while(temp != NULL)
    {
        printf("Data = %d\n", temp->data); // Print data of current node
        temp = temp->next;              // Move to next node
    }
}
```

OUTPUT:

```
Enter the total number of nodes: 5
Enter the data of node 1: 10
Enter the data of node 2: 20
Enter the data of node 3: 30
Enter the data of node 4: 40
Enter the data of node 5: 50

Data in the list
Data = 10
Data = 20
Data = 30
Data = 40
Data = 50
```

Circular Linked List

```
//Circular Linked List
#include <stdio.h>
#include <stdlib.h>


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

```
void createList(int n);
void displayList();



int main()
{
   int n, data, choice=1;

   head = NULL;



   while(choice != 0)
   {
           printf("CIRCULAR LINKED LIST PROGRAM\n");

      printf("1. Create List\n");
      printf("2. Display list\n");
      printf("0. Exit\n");
      printf("Enter your choice : ");

      scanf("%d", &choice);

      switch(choice)
      {
         case 1:
            printf("Enter the total number of nodes in list: ");
            scanf("%d", &n);
```

```
            createList(n);
            break;
        case 2:
            displayList();
            break;
        case 0:
            break;
        default:
            printf("Error! Invalid choice. Please choose between 0-2");
      }

      printf("\n\n\n\n\n");
   }

   return 0;
}



void createList(int n)
{
   int i, data;
   struct node *prevNode, *newNode;

   if(n >= 1)
   {
      head = (struct node *)malloc(sizeof(struct node));
```

```
printf("Enter data of 1 node: ");
scanf("%d", &data);

head->data = data;
head->next = NULL;

prevNode = head;



for(i=2; i<=n; i++)
{
   newNode = (struct node *)malloc(sizeof(struct node));

   printf("Enter data of %d node: ", i);
   scanf("%d", &data);

   newNode->data = data;
   newNode->next = NULL;

   prevNode->next = newNode;

   prevNode = newNode;
}

prevNode->next = head;

printf("\nCIRCULAR LINKED LIST CREATED SUCCESSFULLY\n");
```

```
    }
}

void displayList()
{
    struct node *current;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        current = head;
        printf("DATA IN THE LIST:\n");

        do {
            printf("Data %d = %d\n", n, current->data);

            current = current->next;
            n++;
        }while(current != head);
    }
}
```

OUTPUT:

CIRCULAR LINKED LIST PROGRAM

1. Create List

2. Display list

0. Exit

Enter your choice : 1

Enter the total number of nodes in list: 5

Enter data of 1 node: 10

Enter data of 2 node: 20

Enter data of 3 node: 30

Enter data of 4 node: 40

Enter data of 5 node: 50

Doubly Linked List

//Doubly Linked List

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

struct node {
   int data;
   struct node * prev;
   struct node * next;
}*head, *last;
```

```
void createList(int n);
void displayListFromFirst();
void displayListFromEnd();



int main()
{
   int n, choice;

   head = NULL;
   last = NULL;

   printf("Enter the number of nodes you want to create: ");
   scanf("%d", &n);

   createList(n); // Create list of n nodes

   printf("\nPress 1 to display list from First");
   printf("\nPress 2 to display list from End : ");
   scanf("%d", &choice);

   if(choice==1)
   {
      displayListFromFirst();
   }
   else if(choice == 2)
   {
```

```
        displayListFromEnd();
    }

    return 0;
}



void createList(int n)
{
    int i, data;
    struct node *newNode;

    if(n >= 1)
    {
        head = (struct node *)malloc(sizeof(struct node));

        if(head != NULL)
        {
            printf("Enter data of 1 node: ");
            scanf("%d", &data);

            head->data = data;
            head->prev = NULL;
            head->next = NULL;

            last = head;
```

```
    for(i=2; i<=n; i++)
    {
        newNode = (struct node *)malloc(sizeof(struct node));

        if(newNode != NULL)
        {
            printf("Enter data of %d node: ", i);
            scanf("%d", &data);

            newNode->data = data;
            newNode->prev = last; // Link new node with the previous node
            newNode->next = NULL;

            last->next = newNode; // Link previous node with the new node
            last = newNode;         // Make new node as last/previous node
        }
        else
        {
            printf("Unable to allocate memory.");
            break;
        }
    }

    printf("\nDOUBLY LINKED LIST CREATED SUCCESSFULLY\n");
}
else
```

```
        {
            printf("Unable to allocate memory");
        }
    }
}


void displayListFromFirst()
{
    struct node * temp;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        printf("\n\nDATA IN THE LIST:\n");

        while(temp != NULL)
        {
            printf("DATA of %d node = %d\n", n, temp->data);

            n++;
```

```
            /* Move the current pointer to next node */
            temp = temp->next;
        }
    }
}


/**
 * Display the content of the list from last to first
 */
void displayListFromEnd()
{
    struct node * temp;
    int n = 0;

    if(last == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = last;
        printf("\n\nDATA IN THE LIST:\n");

        while(temp != NULL)
        {
            printf("DATA of last-%d node = %d\n", n, temp->data);
```

```
            n++;

            temp = temp->prev;
        }
    }
}
```

OUTPUT:

Enter the number of nodes you want to create: 5

Enter data of 1 node: 10

Enter data of 2 node: 20

Enter data of 3 node: 30

Enter data of 4 node: 40

Enter data of 5 node: 50

DOUBLY LINKED LIST CREATED SUCCESSFULLY

Press 1 to display list from First

Press 2 to display list from End : 1

DATA IN THE LIST:

DATA of 1 node = 10

DATA of 2 node = 20

Stack Implementation

```
//Stack Program
#include<stdio.h>

#include<stdlib.h>

#define Size 4

int Top=-1, inp_array[Size];
void Push();
void Pop();
void show();

int main()
{
   int choice;

   while(1)
   {
      printf("\nOperations performed by Stack");
      printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");
      printf("\n\nEnter the choice:");
      scanf("%d",&choice);

      switch(choice)
      {
```

```
            case 1: Push();
                    break;
            case 2: Pop();
                    break;
            case 3: show();
                    break;
            case 4: exit(0);

            default: printf("\nInvalid choice!!");
        }
    }
}

void Push()
{
    int x;

    if(Top==Size-1)
    {
        printf("\nOverflow!!");
    }
    else
    {
        printf("\nEnter element to be inserted to the stack:");
        scanf("%d",&x);
        Top=Top+1;
        inp_array[Top]=x;
```

```
    }
}

void Pop()
{
    if(Top==-1)
    {
        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nPopped element:  %d",inp_array[Top]);
        Top=Top-1;
    }
}

void show()
{


    if(Top==-1)
    {
        printf("\nUnderflow!!");
    }
    else
    {
        printf("\nElements present in the stack: \n");
```

```
        for(int i=Top;i>=0;--i)
            printf("%d\n",inp_array[i]);
    }
}
```

OUTPUT:

```
Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4.End

Enter the choice:1

Enter element to be inserted to the stack:10

Operations performed by Stack
1.Push the element
2.Pop the element
3.Show
4.End

Enter the choice:3

Elements present in the stack:
10
```

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:2

Popped element: 10

Operations performed by Stack

1.Push the element

2.Pop the element

3.Show

4.End

Enter the choice:3

Underflow!!

Queue Implementation

```
//Queue
#include <stdio.h>
# define SIZE 100
void enqueue();
void dequeue();
void show();
int inp_arr[SIZE];
```

```
int Rear = - 1;
int Front = - 1;
main()
{
   int ch;
   while (1)
   {
      printf("1.Enqueue Operation\n");
      printf("2.Dequeue Operation\n");
      printf("3.Display the Queue\n");
      printf("4.Exit\n");
      printf("Enter your choice of operations : ");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:
         enqueue();
         break;
         case 2:
         dequeue();
         break;
         case 3:
         show();
         break;
         case 4:
         exit(0);
         default:
```

```
            printf("Incorrect choice \n");
        }
    }
}

void enqueue()
{
    int insert_item;
    if (Rear == SIZE - 1)
       printf("Overflow \n");
    else
    {
        if (Front == - 1)

        Front = 0;
        printf("Element to be inserted in the Queue\n : ");
        scanf("%d", &insert_item);
        Rear = Rear + 1;
        inp_arr[Rear] = insert_item;
    }
}

void dequeue()
{
    if (Front == - 1 || Front > Rear)
    {
        printf("Underflow \n");
```

```
        return ;
    }
    else
    {
        printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
        Front = Front + 1;
    }
}

void show()
{

    if (Front == - 1)
        printf("Empty Queue \n");
    else
    {
        printf("Queue: \n");
        for (int i = Front; i <= Rear; i++)
            printf("%d ", inp_arr[i]);
        printf("\n");
    }
}
```

OUTPUT:

1.Enqueue Operation

2.Dequeue Operation

3.Display the Queue

4.Exit

Enter your choice of operations : 1

Element to be inserted in the Queue: 10

1.Enqueue Operation

2.Dequeue Operation

3.Display the Queue

4.Exit

Enter your choice of operations : 1

Element to be inserted in the Queue: 20

1.Enqueue Operation

2.Dequeue Operation

3.Display the Queue

4.Exit

Enter your choice of operations : 3

Queue:

10 20

1.Enqueue Operation

2.Dequeue Operation

3.Display the Queue

4.Exit

Enter your choice of operations : 2

Element deleted from the Queue: 10

1.Enqueue Operation

2.Dequeue Operation

3.Display the Queue

4.Exit

Enter your choice of operations: 3

Queue:

20

Tree Implementation –Inorder, Preorder, Postorder

```
//Tree Traversal Inorder Preorder Postorder
#include <stdio.h>
#include <stdlib.h>

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

struct node* newNode(int data)
{
   struct node* node
      = (struct node*)malloc(sizeof(struct node));
   node->data = data;
   node->left = NULL;
   node->right = NULL;

   return (node);
```

```
}

void printPostorder(struct node* node)
{
   if (node == NULL)
      return;

   printPostorder(node->left);

   printPostorder(node->right);

   printf("%d ", node->data);
}

void printInorder(struct node* node)
{
   if (node == NULL)
      return;

   printInorder(node->left);

   printf("%d ", node->data);

   printInorder(node->right);
}

void printPreorder(struct node* node)
```

```
{
    if (node == NULL)
        return;

    printf("%d ", node->data);

    printPreorder(node->left);


    printPreorder(node->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("\nPreorder traversal of binary tree is \n");
    printPreorder(root);

    printf("\nInorder traversal of binary tree is \n");
    printInorder(root);

    printf("\nPostorder traversal of binary tree is \n");
```

```
    printPostorder(root);


    getchar();
    return 0;
}
```

OUPUT:

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```