1. Write a c program for linear search which find an element in an unsorted list.
2. Write a c program for binary search which find the location of a given element in a list.
3. Write a c program for sorting using bubble sort method.
4. Write a c program for sorting using quick sort. (Partition exchange sort) method.
5. Write a c program for sorting using straight selection sort.
6. Write a c program for sorting using insertion sort.
7. Write a c program for sorting using shell-sort method.
8. Write a c program for sorting using merge sort method.
9. Write a c program for sorting using radix sort method.
10. Write a c program for implementing of stack and its operation.
11. Write a c program which convert infix string to postfix string.
12. Write a c program which evaluates a postfix string.
13. Write a c program for implementing a simple queue and its operation.
14. Write a c program for implementing a double ended queue and its operation.
15. Write a c program for implementing a circular queue and its operation.
16. Write a c program for implementing a Singly linked list and its operation.
17. Write a c program for implementing a Doubly linked list and its operation.
18. Write a c program to insert an element into Sorted linked list.
19. Write a c program for create a binary tree and its operation.
20. Write a c program for DFS and BFS technique.

1. Write a c program for linear search which find an element in an unsorted list.

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

int searchItem(int *, int);

int searchItem(int a[], int search)
{
   int j = 0;

   for (j = 0; j < 5; j++)
   {
      if (a[j] == search)
      {
         return j;
      }
   }
   return -1;
}

void main()
{
   int arr[5] = {18, 30, 15, 70, 12};

   int item = 30, i;

    clrscr();
    int result = searchItem(arr, item);

   for (i = 0; i < 5; i++)
   {
      printf("%d ", arr[i]);
   }

   if (result == -1)
   {
      printf("\nItem Not Found..!\n");
   }

   else
   {
      printf("\nElement %d is found at %d position", item, result);
   }

   getch();
```

```
}
```

Output: -

18 30 15 70 12

Element 30 is found at 1 postion.

2. Write a c program for binary search which find the location of a given element in a list.

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

int binarySearch(int array[], int x, int beg, int end)
{
   while (beg <= end)
   {
      int mid = (beg + end) / 2;

      if (array[mid] == x)
      {
         return mid;
      }

      else if (array[mid] < x)
      {
         beg = mid + 1;
      }

      else
         end = mid - 1;
   }
   return -1;
}

void main()
{
   int array[10] = {4, 10, 16, 24, 32, 46, 76, 112, 144, 186};

   int search = 10;

   int result = binarySearch(array, search, 0, 9);
   clrscr();
   if (result == -1){
      printf("Not found");
```

```
}
else {
      printf("Element is found at index %d", result);
}
   getch();
}
```

Output:-

Element is found at index 1

3. Write a c program for sorting using bubble sort method.

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

#define size 5

void main()
{
   int i, j, temp;
   int arr[size] = {18, 30, 15, 70, 12};
   clrscr();
   printf("\nUnsorted Array : \n");
   for (i = 0; i < size; i++)
   {
      printf("%d ", arr[i]);
   }

   for (i = 0; i < size - 1; i++)
   {
      for (j = 0; j < size - 1 - i; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }

   printf(" \nSorted array is: \n");
   for (i = 0; i < size; i++)
   {
```

```
        printf("%d ", arr[i]);
    }
    getch();
}
```

Output:-

```
Unsorted Array :
18 30 15 70 12

Sorted array is:
12 15 18 30 70
```

4. Write a c program for sorting using quick sort. (Partition exchange sort) method.

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

void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int a[], int s, int e)
{
    int pivot = a[e];
    int pIndex = s;
    int i;
    for (i = s; i <= e - 1; i++)
    {
        if (a[i] <= pivot)
        {
            swap(&a[i], &a[pIndex]);
            pIndex++;
        }
    }
    swap(&a[e], &a[pIndex]);
    return pIndex;
}

void quickSort(int a[], int s, int e)
{
```

```
    int p;
    if (s < e)
    {
        p = partition(a, s, e);
        quickSort(a, s, p - 1);
        quickSort(a, p + 1, e);
    }
}

void main()
{
    int i = 0;
    int arr[10] = {70, 40, 30, 80, 90, 50};
    clrscr();

    while (i <= 5)
    {
        printf("%d ", arr[i]);
        i++;
    }
    printf("\n");

    quickSort(arr, 0, 5);
    i = 0;

    printf("\n");

    while (i <= 5)
    {
        printf("%d ", arr[i]);
        i++;
    }
  getch();
}
```

Output:-

70 40 30 80 90 50

30 40 50 70 80 90

5. Write a c program for sorting using straight selection sort.

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

#define size 6

int main()
{
   int i, min, findMin, temp;

   int arr[size] = {6, 2, 1, 4, 3, 5};
   clrscr();

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

   for (i = 0; i < size - 1; i++)
   {
      min = i;
      for (findMin = i + 1; findMin < size; findMin++)
      {
         if (arr[findMin] < arr[min])
         {
            min = findMin;
         }
      }
      temp = arr[i];
      arr[i] = arr[min];
      arr[min] = temp;
   }

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

Output:-

Before Sort..

6 2 1 4 3 5

After Sort..

1 2 3 4 5 6

6. Write a c program for sorting using insertion sort.

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

#define SIZE 5

void main()
{
   int arr[SIZE] = {12, 11, 13, 5, 6};

   int i, key, j;
   clrscr();

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

   printf("\n");

   for (i = 1; i < SIZE; i++)
   {
      key = arr[i];
      j = i - 1;

      while (j >= 0 && arr[j] > key)
      {
         arr[j + 1] = arr[j];
         j--;
```

```
        }
        arr[j + 1] = key;
    }

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

Output:-

```
Before Sort..

12 11 13 5 6

After Sort..

5 6 11 12 13
```

7. Write a c program for sorting using shell-sort method.

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

    void shellSort(int a[], int n)
    {
        int gap = n / 2;
        while (gap > 0)
        {
            int i;
            for (i = gap; i < n; i++)
            {
                int key = a[i];
                int j = i;

                while (j >= gap && a[j - gap] > key)
                {
                    a[j] = a[j - gap];
                    j = j - gap;
                }
```

```
            a[j] = key;
        }
        gap = gap / 2;
    }
  }

void main()
{
  int i;
  int a[9] = {33, 31, 40, 8, 12, 17, 25, 42, 21};
  clrscr();

  printf("Before sorting array elements are - \n");
  for (i = 0; i < 9; i++)
    printf("%d ", a[i]);

  shellSort(a, 9);

  printf("\nAfter applying shell sort, the array elements are - \n");
  for (i = 0; i < 9; i++)
    printf("%d ", a[i]);

  getch();
}
```

Output:-

Before sorting array elements are

33 31 40 8 12 17 25 42 21

After applying shell sort, the array elements are

8 12 17 21 25 31 33 40 42

8. Write a c program for sorting using merge sort method.

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


void merge(int a[], int start, int mid, int end)
{
  int i = start;   // Starting index of first subarray
```

```
    int j = mid + 1; // Starting index of second subarray
    int k = start;   // Starting index of merged subarray
    int temp[10];    // Temporary array to store the merged subarray

    while (i <= mid && j <= end) // Traverse both the subarrays and in each iteration
add smaller of both elements in temp
    {
        if (a[i] <= a[j]) //
        {
            temp[k] = a[i];
            i++;
            k++;
        }
        else
        {
            temp[k] = a[j];
            j++;
            k++;
        }
    }

    while (i <= mid) // Add remaining elements of first subarray in temp.
    {
        temp[k] = a[i];
        i++;
        k++;
    }
    while (j <= end) // Add remaining elements of second subarray in temp.
    {
        temp[k] = a[j];
        j++;
        k++;
    }
    for (i = start; i <= end; i++) // Copy the merged subarray into original array.
    {
        a[i] = temp[i];
    }
}
void divide(int a[], int start, int end)
{
    if (start < end) // if there is more than one element.
    {
        int mid = (start + end) / 2; // find the mid index.

        divide(a, start, mid);   // divide the array into two halves.
        divide(a, mid + 1, end); // divide the array into two halves.
```

```
        merge(a, start, mid, end); // merge the two halves.
    }
}

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

int main()
{

    int a[8] = {50, 10, 70, 60, 80, 40, 30, 20};

    int n = sizeof(a) / sizeof(a[0]);
    clrscr();

    printf("Before sorting array elements are - \n");
    printArray(a, n);
    divide(a, 0, n - 1);

    printf("After sorting array elements are - \n");
    printArray(a, n);
    getch();
}
```

Output:-

Before sorting array elements are

50 10 70 60 80 40 30 20

After sorting array elements are

10 20 30 40 50 60 70 80

9. Write a c program for sorting using radix sort method.

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

int getMax(int a[], int n)
{
   int max = a[0];
   for (int i = 1; i < n; i++)
   {
      if (a[i] > max)
         max = a[i];
   }
   return max; // maximum element from the array
}

void countingSort(int a[], int n, int place) // function to implement counting sort
{
   int output[n + 1];
   int count[10] = {0};

   // Calculate count of elements
   for (int i = 0; i < n; i++)
      count[(a[i] / place) % 10]++;

   // Calculate cumulative frequency
   for (int i = 1; i < 10; i++)
      count[i] += count[i - 1];

   // Place the elements in sorted order
   for (int i = n - 1; i >= 0; i--)
   {
      output[count[(a[i] / place) % 10] - 1] = a[i];
      count[(a[i] / place) % 10]--;
   }

   for (int i = 0; i < n; i++)
      a[i] = output[i];
}

// function to implement radix sort
void radixsort(int a[], int n)
{
   // get maximum element from array
   int max = getMax(a, n);

   // Apply counting sort to sort elements based on place value
```

```
    for (int place = 1; max / place > 0; place *= 10)
        countingSort(a, n, place);
}

// function to print array elements
void printArray(int a[], int n)
{
    for (int i = 0; i < n; ++i)
    {
        printf("%d  ", a[i]);
    }
    printf("\n");
}

int main()
{
    int a[] = {181, 289, 390, 121, 145, 736, 514, 888, 122};
    int n = sizeof(a) / sizeof(a[0]);

    printf("Before sorting array elements are - \n");
    printArray(a, n);

    radixsort(a, n);

    printf("After applying Radix sort, the array elements are - \n");
    printArray(a, n);
}
```

Output:-

```
Before sorting array elements are

181 289 390 121 145 736 514 888 122


After applying Radix sort, the array elements are

121 122 145 181 289 390 514 736 888
```

10. Write a c program for implementing of stack and its operation.

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

```
#define SIZE 5

int stack[SIZE];
int top = -1;

void push(int value)
{
        if (top == SIZE - 1)
        {
          printf("Stack is Full..\n");
        }
        else
        {
          top++;
          stack[top] = value;
          printf("Element Pushed : %d \n", value);
        }
}

void pop()
{
        if (top == -1)
        {
          printf("Stack is Empty..\n");
        }
        else
        {
          printf("Element Popped : %d \n", stack[top]);
          top--;
        }
}

int length()
{
        int i = 0;
        if (top == -1)
        {
          printf("Stack is Empty..\n");
        }
        else
        {
          while (i <= top)
          {
                  i++;
          }
        }
```

```
        return i;
}

void display()
{
        int i = 0;
        if (top == -1)
        {
          printf("Stack is Empty..\n");
        }
        else
        {
          while (i <= top)
          {
                  printf("%d \n", stack[i]);
                  i++;
          }
        }
}

void peek()
{
        if (top == -1)
          printf("Stack is Empty..\n");

        else
          printf("Top Item is : %d \n", stack[top]);
}

void main()
{
        int choice;
        int item, len;
        clrscr();
        while (1)
        {
          printf("1. Push. \n");
          printf("2. Pop. \n");
          printf("3. Length \n");
          printf("4. Display. \n");
          printf("5. Peek. \n");
          printf("0. Exit. \n");

          printf("Enter Your Choice : ");
          scanf("%d", &choice);
```

```
        switch (choice)
        {
        case 1:
                printf("Enter Element to Push : ");
                scanf("%d", &item);
                push(item);
                break;
        case 2:
                pop();
                break;
        case 3:
                len = length();
                printf("Length : %d \n", len);
                break;
        case 4:
                display();
                break;
        case 5:
                peek();
                break;
        case 0:
                printf("Bye Bye \n \n");
                exit(1);

        default:
                printf("Invalid Choice. \n\n");
        }
      }
}
```

Output:-

1. Push
2. Pop
3. Length
4. Display
5. Peek

0. Exit

Enter your choice : 1

Enter Element to push : 10

Element pushed :10

11. Write a c program which convert infix string to postfix string.

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

char stack[100];
int top = -1;

void push(char x)
{
   top++;
   stack[top] = x;
}

char pop()
{
   char item;

   if (top == -1)
   {
      return -1;
   }
   else
   {
      item = stack[top];
      top--;
      return item;
   }
}

int priority(char sym)
{
   if (sym == '^')
      return 3;
   else if (sym == '/' || sym == '*')
      return 2;
   else if (sym == '+' || sym == '-')
      return 1;
   else
      return -1;
}

void main()
```

```
{
   char exp[10];
   char *e, x;

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

   printf("\n");

   e = exp;

   while (*e != '\0')
   {
      if (isalnum(*e))
         printf("%c ", *e);

      else if (*e == '(')
         push(*e);

      else if (*e == ')')
      {
         while (x = pop() != '(')
            printf("%c ", x);
      }

      else
      {
         while (priority(stack[top]) >= priority(*e))
         {
            printf("%c ", pop());
         }
         push(*e);
      }

      e++;
   }

   while (top != -1)
   {
      printf("%c ", pop());
   }
}
```

Output:-

```
Enter the expression: a + b * c / d

a b c * d / +
```

12. Write a c program which evaluates a postfix string.

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

int stack[20];
int top = -1;

void push(int x)
{
   stack[++top] = x;
}

int pop()
{
   return stack[top--];
}

void main()
{
   char exp[20];
   char *e;
   int n1, n2, n3, num;
   clrscr();
   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());
      getch();
}
```

Output:-

Enter the expression : 45 +

The result of expression 45 + = 9

13. Write a c program for implementing a simple queue and its operation.

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

int myqueue[SIZE];
int front = -1, rear = -1;
```

```
void enQueue(int value)
{
   if (rear == SIZE - 1)
      printf("\nQueue is Full.. \n\n");
   else
   {
      rear++;
      myqueue[rear] = value;
      printf("\nInserted : %d \n\n", value);
      if (front == -1)
      {
         front++;
      }
   }
}

void deQueue()
{
   if (front == -1)
      printf("\nQueue is Empty!!\n");
   else
   {
      printf("\nItem Deleted : %d \n", myqueue[front]);
      front++;

      if (front > rear)
      {
         front = -1;
         rear = -1;
      }
   }
}

void display()
{
   if (front == -1)
      printf("\nQueue is Empty!!!");
   else
   {
      int i = front;
      printf("\nQueue elements are:\n");

      while (i <= rear)
      {
         printf("%d  ", myqueue[i]);
```

```
            i++;
        }
    }
    printf("\n");
}

void length()
{
    int count = 0;
    int i = front;
    if (front == -1)
    {
        printf("Queue is Empty..\n");
    }
    else
    {
        while (i <= rear)
        {
            count++;
            i++;
        }
        printf("Length is : %d \n", count);
    }
}

void main()
{
    int choice;
    int item, len;
    clrscr();
    while (1)
    {
        printf("1. enQueue. \n");
        printf("2. deQueue. \n");
        printf("3. Display. \n");
        printf("4. Length. \n");
        printf("0. Exit. \n");

        printf("Enter Your Choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter Element to enQueue : ");
            scanf("%d", &item);
```

```
            enQueue(item);
            break;
        case 2:
            deQueue();
            break;
        case 3:
            display();
            break;
        case 4:
            length();
            break;
        case 0:
            exit(0);
        default:
            printf("Invalid Choice. \n\n");
        }
    }
}
```

Output:-

1. enQueue
2. deQueue
3. Display
4. Length.
   0.Exit

```
Enter your choice: 1
Enter element to enQueue: 10

Inserted: 10
```

14. Write a c program for implementing a double ended queue and its operation.

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

#define size 5

void insertFront(int);
void insertRear(int);
void deleteFront();
void deleteRear();
void display();
```

```
void getFront();
void getRear();

int DEQ[size];
int front = -1;
int rear = -1;

//  insertFront function will insert the value from the front.
void insertFront(int value)
{
   if (front == 0 && rear == size - 1  | |  front == rear + 1)
   {
      printf("Queue is Full..\n");
   }

   else if (front == -1 && rear == -1)
   {
      front = 0;
      rear = 0;
      DEQ[front] = value;
      printf("Element Inserted from front : %d \n", value);
   }

   else if (front == 0)
   {
      front = size - 1;
      DEQ[front] = value;
      printf("Element Inserted from front : %d \n", value);
   }
   else
   {
      front--;
      DEQ[front] = value;
      printf("Element Inserted from front : %d \n", value);
   }
}

// insertRear function will insert the value from the rear
void insertRear(int value)
{
   if ((front == 0 && rear == size - 1)  | |  (front == rear + 1))
   {
      printf("Queue is Full..\n");
   }

   else if ((front == -1) && (rear == -1))
```

```
    {
        rear = 0;
        front = 0;
        DEQ[rear] = value;
        printf("Element Inserted from rear : %d \n", value);
    }

    else if (rear == size - 1)
    {
        rear = 0;
        DEQ[rear] = value;
        printf("Element Inserted from rear : %d \n", value);
    }

    else
    {
        rear++;
        DEQ[rear] = value;
        printf("Element Inserted from rear : %d \n", value);
    }
}

void deleteFront()
{
    if ((front == -1) && (rear == -1))
    {
        printf("Queue is empty..\n");
    }
    else if (front == rear)
    {
        printf("\nDeleted from front element is %d \n", DEQ[front]);
        front = -1;
        rear = -1;
    }
    else if (front == size - 1)
    {
        printf("\nDeleted from front element is %d \n", DEQ[front]);
        front = 0;
    }

    else
    {
        printf("\nDeleted from front element is %d \n", DEQ[front]);
        front++;
    }
}
```

```
void deleteRear()
{
   if ((front == -1) && (rear == -1))
   {
      printf("Queue is empty");
   }
   else if (front == rear)
   {
      printf("\nDeleted from rear element is %d \n", DEQ[rear]);
      front = -1;
      rear = -1;
   }
   else if (rear == 0)
   {
      printf("\nDeleted from rear element is %d \n", DEQ[rear]);
      rear = size - 1;
   }
   else
   {
      printf("\nDeleted from rear element is %d \n", DEQ[rear]);
      rear--;
   }
}

void display()
{
   int i = front;
   printf("\n\nElements in a Queue are: ");

   while (i != rear)
   {
      printf("%d ", DEQ[i]);
      i = (i + 1) % size;
   }
   printf("%d", DEQ[rear]);
}

void getFront()
{
   if ((front == -1) && (rear == -1))
   {
      printf("Queue is empty");
   }

   else
```

```
    {
        printf("\nThe Value at front is : %d", DEQ[front]);
    }
}

void getRear()
{
    if ((front == -1) && (rear == -1))
    {
        printf("Queue is empty");
    }
    else
    {
        printf("\nThe value at rear is : %d", DEQ[rear]);
    }
}

void main()
{
    int choice;
    clrscr();

    while (choice != 0)
    {
        printf("\n\n1. Insert at front");
        printf("\n2. Insert at rear");
        printf("\n3. Delete from front");
        printf("\n4. Delete from rear");
        printf("\n5. Display");
        printf("\n6. Get front");
        printf("\n7. Get rear");
        printf("\n0. Exit");

        printf("\nEnter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            int value;
            printf("\nEnter the value to be inserted at front : ");
            scanf("%d", &value);

            insertFront(value);
            break;
```

```
}
case 2:
{
   int value;

   printf("\nEnter the value to be inserted at rear : ");
   scanf("%d", &value);

   insertRear(value);
   break;
}

case 3:
{
   deleteFront();
   break;
}
case 4:
{
   deleteRear();
   break;
}
case 5:
{
   display();
   break;
}
case 6:
{
   getFront();
   break;
}
case 7:
{
   getRear();
   break;
}
case 0:
{
   printf("\nExiting..\n");
   exit(1);
   break;
}

default:
{
```

```
            printf("\nInvalid choice");
            break;
        }
        }
    }
  getch();
}
```

Output:-

1. Insert at front
2. Insert at Rear
3. Delete from front
4. Delete from Rear
5. Display
6. Get front
7. Get Rear

0.Exit

Enter your Choice : 1

Enter the value to be inserted at front: 20

Element Inserted from front: 20

15. Write a c program for implementing a circular Queue and its operation.

```
#include <stdio.h>

#define SIZE 5

int CQueue[SIZE];

int front = -1;
int rear = -1;

void enQueue(int value)
{
   if (front == rear + 1  ||  front == 0 && rear == SIZE - 1)
   {
```

```
        printf("Queue is Full..\n");
    }
    else
    {
        rear = (rear + 1) % SIZE;
        CQueue[rear] = value;

        printf("Inserted : %d  \n", value);

        if (front == -1)
        {
            front = 0;
        }
    }
}

int deQueue()
{
    if ((front == -1) && (rear == -1)) // check CQueue is empty
    {
        printf("\nQueue is Empty..");
    }
    else
    {
        printf("\nThe dequeued element is %d", CQueue[front]);
        front = (front + 1) % SIZE;

        if (front == rear)
        {
            printf("\nThe dequeued element is %d", CQueue[front]);
            front = -1;
            rear = -1;
        }
    }
}

void display()
{
    int i = front;
    if (front == -1 && rear == -1)
    {
        printf("\n Queue is empty..");
    }
    else
    {
        printf("\nElements in a Queue are :");
```

```
        while (i != rear)
        {
            printf("%d ", CQueue[i]);
            i = (i + 1) % SIZE;
        }
        printf("%d ", CQueue[i]);
    } // END OF ELSE
} // END OF DISPLAY FUN.

void main()
{
    int choice = 1, x;
    clrscr();
    while (1)
    {
        printf("\n1: Insert.");
        printf("\n2: Delete.");
        printf("\n3: Display.");
        printf("\n0: Exit.");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:

            printf("Enter the element : ");
            scanf("%d", &x);
            enQueue(x);
            break;
        case 2:
            deQueue();
            break;
        case 3:
            display();
            break;
        case 0:
            exit(0);
            break;
        default:
            printf("Enter a Valid Choice.\n");
        }
    }
}
```

Output:-

1. Insert
2. Delete
3. Display
0. Exit

Enter your Choice : 1

Enter the Elements: 10

Inserted : 10

16. Write a c program for implementing a Singly linked list and its operation.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
void append();
void addAtBegin();
void addAtAfter();
int length();
void display();
void deleteNode();

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

struct node *head = NULL;

int len;
void append()
{
   struct node *temp;

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

   printf("Enter Data : ");
```

```
    scanf("%d", &temp->data);

    printf("Data Inserted : %d \n\n", temp->data);

    temp->link = NULL;
    if (head == NULL)
        head = temp;
    else
    {
        struct node *p;
        p = head;
        while (p->link != NULL)
        {
            p = p->link;
        }
        p->link = temp;
    }
}

void addAtBegin()
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    printf("Enter Data to Insert at Begin : ");
    scanf("%d", &temp->data);
    temp->link = head;
    head = temp;

    printf("Data Inserted at Begin : %d \n", temp->data);
}

void addAtAfter()
{
    struct node *travel = head, *temp;
    int loc, len, i = 1;
    printf("Enter Location : ");
    scanf("%d", &loc); // 7
    len = length();
    if (loc > len)
    {
        printf("Invalid Location.\n");
        printf("Total Nodes is : %d \n\n", len);
    }
    else
    {
        while (i < loc)
```

```
        {
            travel = travel->link;
            i++;
        }
        temp = (struct node *)malloc(sizeof(struct node));
        printf("Enter Data To Insert in Node :");
        scanf("%d", &temp->data);
        temp->link = NULL;

        temp->link = travel->link;
        travel->link = temp;
    }
}

int length()
{
    int count = 0;
    struct node *temp;
    temp = head;
    while (temp != NULL)
    {
        count++;
        temp = temp->link;
    }
    return count;
}

void deleteNode()
{
    struct node *temp = head;
    int loc;
    display();
    printf("Enter Location To Delete : ");
    scanf("%d", &loc);
    if (loc > length())
    {
        printf("Invalid Location.\n\n");
    }

    else if (loc == 1)
    {
        head = temp->link;
        temp->link = NULL;
        free(temp);
    }
    else
```

```
    {
        struct node *travel = head, *del;
        int i = 1;
        while (i < loc - 1)
        {
            travel = travel->link;
            i++;
        }
        del = travel->link;
        travel->link = del->link;
        del->link = NULL;
        free(del);
    }
}

void display()
{
    int cnt = 1;
    if (head == NULL)
    {
        printf("No Elements in Linked List..\n\n");
    }
    else
    {
        struct node *temp;
        temp = head;

        while (temp != NULL)
        {
            printf("%d. %d\n", cnt, temp->data);
            temp = temp->link;
            cnt++;
        }
    }
}

void main()
{
    int ch;
    clrscr();
    while (1)
    {
        printf("\n1: Append.");
        printf("\n2: Add At Begin.");
        printf("\n3: Add At After.");
        printf("\n4: Length.");
```

```
        printf("\n5: Display.");
        printf("\n6: Delete.");
        printf("\n0: Quit.");
        printf("\n\n Enter Your Choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            append();
            break;
        case 2:
            addAtBegin();
            break;
        case 3:
            addAtAfter();
            break;
        case 4:
            len = length();
            printf("Length : %d \n", len);
            break;
        case 5:
            display();
            break;
        case 6:
            deleteNode();
            break;
        case 0:
            printf("Bye Bye..!. \n\n");
            exit(0);
        default:
            printf("Invalid Choice. \n\n");
        }
    }
}
```

Output:-

1.Append
2.Add At Begin
3.Add At After
4.Length
5.Display
6.Delete

0.Quit

Enter your choice : 1
Enter Data : 10

Data Inserted: 10

17. Write a c program for implementing a Doubly linked list and its operation.

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

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

struct node *head = NULL;

void prepend();
void append();
int lenght();
void display();
void addAtAfter();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void search();

void main()
{
   int choice = 0;
   int len;
   while (choice != 9)
   {
      printf("1.Insert at begining \n");
      printf("2.Insert at last \n");
      printf("3.Insert at any random location \n");
      printf("4.Delete from Beginning \n");
      printf("5.Delete from last \n");
```

```
    printf("6.Delete the node after the given data \n");
    printf("7.Search \n");
    printf("8.Display\n");
    printf("9.Length\n");
    printf("0.Exit\n");

    printf("\nEnter your choice : ");
    scanf("\n%d", &choice);

    switch (choice)
    {
    case 1:
       prepend();
       break;
    case 2:
       append();
       break;
    case 3:
       addAtAfter();
       break;
    case 4:
       deletion_beginning();
       break;
    case 5:
       deletion_last();
       break;
    case 6:
       deletion_specified();
       break;
    case 7:
       search();
       break;
    case 8:
       display();
       break;
    case 9:
       len = lenght();
       printf("Length is : %d", len);
       break;
    case 0:
       exit(0);
       break;
    default:
       printf("Please enter valid choice..");
    }
}
```

```
}

void append()
{
   struct node *ptr, *temp;
   temp = (struct node *)malloc(sizeof(struct node));

   scanf("%d", &temp->data);

   temp->next = NULL;
   temp->prev = NULL;

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

void prepend()
{
   struct node *temp;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter Item value : ");
   scanf("%d", &temp->data);

   temp->next = NULL;
   temp->prev = NULL;

   if (head == NULL)
   {

      head = temp;
   }
   else
   {
      temp->next = head;
      head->prev = temp;
      head = temp;
```

```
    }
}

int lenght()
{
    struct node *prt = head;
    int count = 0;

    while (prt != NULL)
    {
        count++;
        prt = prt->next;
    }
    return count;
}

void addAtAfter()
{
    struct node *ptr, *temp;
    int len, loc, i = 1;

    scanf("%d", &loc);

    len = lenght();
    if (loc > len)
    {
        printf("Invalid Location..\n");
    }
    else
    {
        temp = (struct node *)malloc(sizeof(struct node));
        scanf("%d", &temp->data);
        temp->prev = NULL;
        temp->next = NULL;
        ptr = head;
        while (i < loc)
        {
            ptr = ptr->next;
            i++;
        }
        temp->next = ptr->next;
        ptr->next->prev = temp;
        temp->prev = ptr;
        ptr->next = temp;
    }
}
```

```
void deletion_beginning()
{
   struct node *del;
   if (head == NULL)
   {
      printf("\n There is No Node in List..\n");
   }
   else if (head->next == NULL)
   {
      head = NULL;
      free(head);
      printf("\n Node Deleted\n");
   }
   else
   {
      del = head;
      head = head->next;
      head->prev = NULL;
      free(del);
      printf("\nNode Deleted\n");
   }
}

void deletion_last()
{
   struct node *ptr;
   if (head == NULL)
   {
      printf("\n There is No Node in List..\n");
   }
   else if (head->next == NULL)
   {
      head = NULL;
      free(head);
      printf("\nNode Deleted\n");
   }
   else
   {
      ptr = head;
      if (ptr->next != NULL)
      {
         ptr = ptr->next;
      }
      ptr->prev->next = NULL;
      free(ptr);
```

```
        printf("\nNode deleted\n");
    }
}

void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while (ptr->data != val)
        ptr = ptr->next;

    if (ptr->next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if (ptr->next->next == NULL)
    {
        ptr->next = NULL;
    }
    else
    {
        temp = ptr->next;
        ptr->next = temp->next;
        temp->next->prev = ptr;
        free(temp);
        printf("\nnode deleted\n");
    }
}

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

void search()
{
    struct node *ptr;
```

```
    int item, i = 0, found = 0;
    ptr = head;
    if (ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d", &item);
        while (ptr != NULL)
        {
            if (ptr->data == item)
            {
                printf("\nitem found at location %d \n", i + 1);
                found = 1;
                break;
            }
            else
            {
                found = 0;
            }
            i++;
            ptr = ptr->next;
        }
        if (found == 0)
        {
            printf("\nItem not found\n");
        }
    }
}
```

Output:-

1.Insert at beginning
2.Insert at last
3.Insert at any random location
4.Delete from beginning
5.Delete from last
6.Delete the node after given data
7.Search

8.Display
9.Length
0.Exit

Enter your choice: 1

Enter Item Value : 10

18. Write a c program to insert an element into Sorted linked list.

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

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

struct node *head = NULL;

void append(int item)
{
   struct node *temp;
   temp = (struct node *)malloc(sizeof(struct node));
   temp->data = item;
   temp->link = NULL;
   if (head == NULL)
      head = temp;
   else
   {
      struct node *p;
      p = head;
      while (p->link != NULL)
      {
         p = p->link;
      }
      p->link = temp;
   }
}

void insert(int data)
{
```

```
    int key = data;

    struct node *travel;

    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->data = data;
    temp->link = NULL;


    if (head == NULL || key < head->data)
    {
        temp->link = head;
        head = temp;
    }

    else
    {
        travel = head;
        while (travel->link != NULL && travel->link->data < key)
        {
            travel = travel->link;
        }
        temp->link = travel->link;
        travel->link = temp;
    }
}

void display()
{
    struct node *p = head;
    while (p != NULL)
    {
        printf("%d ", p->data);
        p = p->link;
    }
}

int main()
{
    append(10);
    append(20);
    append(30);
    append(50);

    clrscr();
```

```
    printf("Before Insertion : ");
    display();

    insert(40);

    printf("\nAfter Insertion : ");
    display();
    return 0;
}
```

Output:-

Before Insertion:

10 20 30 50

After Insertion:

10 20 30 40 50

19. Write a c program for create a binary tree and its operation.

```
// Data Structure All Practicals Completed.

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

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

struct node *create()
{
    struct node *p;
    int x;
    printf("Enter -1 for no node : ");
    scanf("%d", &x);
    if (x == -1)
    {
```

```
        return NULL;
    }
    p = (struct node *)malloc(sizeof(struct node));
    p->data = x;
    printf("Enter left child of %d: ", x);
    p->left = create();
    printf("Enter right child of %d: ", x);
    p->right = create();
    return p;
}

void inorder(struct node *t)
{
    if (t != NULL)
    {
        inorder(t->left);
        printf("%d ", t->data);
        inorder(t->right);
    }
}

int main()
{
    struct node *root;
    root = create();

    printf("Inorder traversal: ");
    inorder(root);

    getch();
}
```

Output:-

20. Write a c program for DFS and BFS technique.

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

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

void printCurrentLevel(struct node *root, int level);
int height(struct node *node);

void printLevelOrder(struct node *root)
{
   int h = height(root);
   int i;
   for (i = 1; i <= h; i++)
      printCurrentLevel(root, i);
}

/* Print nod    es at a current level */
void printCurrentLevel(struct node *root, int level)
{
   if (root == NULL)
      return;
   if (level == 1)
      printf("%d ", root->data);
   else if (level > 1)
   {
      printCurrentLevel(root->left, level - 1);
      printCurrentLevel(root->right, level - 1);
   }
}

/* Compute the "height" of a tree -- the number of
   nodes along the longest path from the root node
   down to the farthest leaf node.*/
int height(struct node *node)
{
   if (node == NULL)
      return 0;
   else
   {
```

```
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}

struct node *create()
{
    struct node *p;
    int x;
    printf("Enter -1 for no node : ");
    scanf("%d", &x);
    if (x == -1)
    {
        return NULL;
    }
    p = (struct node *)malloc(sizeof(struct node));
    p->data = x;
    printf("Enter left child of %d: ", x);
    p->left = create();
    printf("Enter right child of %d: ", x);
    p->right = create();
    return p;
}

// DFS
void preorder(struct node *t)
{
    if (t != NULL)
    {
        printf("%d ", t->data);
        preorder(t->left);
        preorder(t->right);
    }
}

void inorder(struct node *t)
{
    if (t != NULL)
    {
```

```
        inorder(t->left);
        printf("%d ", t->data);
        inorder(t->right);
    }
}

void postorder(struct node *t)
{
    if (t != NULL)
    {
        postorder(t->left);
        postorder(t->right);
        printf("%d ", t->data);
    }
}

int main()
{
    struct node *root;
    root = create();
    printf("Preorder traversal: ");
    preorder(root);
    printf("\n");
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    printf("Postorder traversal: ");
    postorder(root);

    printf("\n\n(BFS) OR Level Order traversal of binary tree is \n");
    printLevelOrder(root);

    return 0;
}
```

Output:-