

## 1. Simple C Programs

### 1. C Examples on Integers

```

1. /*
2.  * C program to check whether a given integer is odd or even
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int ival, remainder;
9.
10.    printf("Enter an integer : ");
11.    scanf("%d", &ival);
12.    remainder = ival % 2;
13.    if (remainder == 0)
14.        printf("%d is an even integer\n", ival);
15.    else
16.        printf("%d is an odd integer\n", ival);
17. }
```

```

1. /*
2.  * C program to find the sum of odd and even numbers from 1 to N
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int i, num, odd_sum = 0, even_sum = 0;
9.
10.    printf("Enter the value of num\n");
11.    scanf("%d", &num);
12.    for (i = 1; i <= num; i++)
13.    {
14.        if (i % 2 == 0)
15.            even_sum = even_sum + i;
16.        else
17.            odd_sum = odd_sum + i;
18.    }
19.    printf("Sum of all odd numbers = %d\n", odd_sum);
```

```

20.     printf("Sum of all even numbers = %d\n", even_sum);
21. }
```

```

1. /*
2. * C program to check whether a given integer is positive
3. * or negative
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int number;
10.
11.    printf("Enter a number \n");
12.    scanf("%d", &number);
13.    if (number >= 0)
14.        printf("%d is a positive number \n", number);
15.    else
16.        printf("%d is a negative number \n", number);
17. }
```

```

1. /*
2. * C program to find the number of integers divisible by
3. * 5 between the given range num1 and num2, where num1 < num2.
4. *
5. * Also find the sum of all these integer numbers which are divisible
6. * by 5 and display the total.
7. */
8. #include <stdio.h>
9.
10. void main()
11. {
12.     int i, num1, num2, count = 0, sum = 0;
13.
14.     printf("Enter the value of num1 and num2 \n");
15.     scanf("%d %d", &num1, &num2);
16.     /* Count the number and compute their sum*/
17.     printf("Integers divisible by 5 are \n");
```

```

18.     for (i = num1; i < num2; i++)
19.     {
20.         if (i % 5 == 0)
21.         {
22.             printf("%3d,", i);
23.             count++;
24.             sum = sum + i;
25.         }
26.     }
27.     printf("\n Number of integers divisible by 5 between %d and %d =
28. %d\n", num1, num2, count);
29.     printf("Sum of all integers that are divisible by 5 = %d\n", sum);
30. }
```

```

1. /*
2.  * C program to read two integers M and N and to swap their values.
3.  * Use a user-defined function for swapping. Output the values of M
4.  * and N before and after swapping.
5. */
6. #include <stdio.h>
7. void swap(float *ptr1, float *ptr2);
8.
9. void main()
10. {
11.     float m, n;
12.
13.     printf("Enter the values of M and N \n");
14.     scanf("%f %f", &m, &n);
15.     printf("Before Swapping:M = %5.2ftN = %5.2f\n", m, n);
16.     swap(&m, &n);
17.     printf("After Swapping:M = %5.2ftN = %5.2f\n", m, n);
18. }
19./* Function swap - to interchanges the contents of two items */
20.void swap(float *ptr1, float *ptr2)
21.{
22.    float temp;
23.
24.    temp = *ptr1;
25.    *ptr1 = *ptr2;
26.    *ptr2 = temp;
27. }
```

```

1. /*
2.  * C program to accept two integers and check if they are equal
3. */
4. #include <stdio.h>
5. void main()
6. {
7.     int m, n;
8.
9.     printf("Enter the values for M and N\n");
10.    scanf("%d %d", &m, &n);
11.    if (m == n)
12.        printf("M and N are equal\n");
13.    else
14.        printf("M and N are not equal\n");
15. }

```

```

1. /*
2.  * C program to accept an integer & find the sum of its digits
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     long num, temp, digit, sum = 0;
9.
10.    printf("Enter the number \n");
11.    scanf("%ld", &num);
12.    temp = num;
13.    while (num > 0)
14.    {
15.        digit = num % 10;
16.        sum = sum + digit;
17.        num /= 10;
18.    }
19.    printf("Given number = %ld\n", temp);
20.    printf("Sum of the digits %ld = %ld\n", temp, sum);
21. }

```

## 2. C Examples on Number Conversion

```

1. /*
2.  * C program to convert the given binary number into decimal

```

```

3.  /*
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int num, binary_val, decimal_val = 0, base = 1, rem;
9.
10.    printf("Enter a binary number(1s and 0s) \n");
11.    scanf("%d", &num); /* maximum five digits */
12.    binary_val = num;
13.    while (num > 0)
14.    {
15.        rem = num % 10;
16.        decimal_val = decimal_val + rem * base;
17.        num = num / 10 ;
18.        base = base * 2;
19.    }
20.    printf("The Binary number is = %d \n", binary_val);
21.    printf("Its decimal equivalent is = %d \n", decimal_val);
22. }
```

```

1. /*
2. * C program to accept a decimal number and convert it to binary
3. * and count the number of 1's in the binary number
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     long num, decimal_num, remainder, base = 1, binary = 0, no_of_1s = 0;
10.
11.    printf("Enter a decimal integer \n");
12.    scanf("%ld", &num);
13.    decimal_num = num;
14.    while (num > 0)
15.    {
16.        remainder = num % 2;
17.        /* To count no.of 1s */
18.        if (remainder == 1)
19.        {
20.            no_of_1s++;
21.        }
22.        binary = binary + remainder * base;
23.        num = num / 2;
24.        base = base * 10;
25.    }
```

```

26.     printf("Input number is = %d\n", decimal_num);
27.     printf("Its binary equivalent is = %ld\n", binary);
28.     printf("No.of 1's in the binary number is = %d\n", no_of_1s);
29. }
30. /*
31. * C program to convert given number of days to a measure of time given
32. * in years, weeks and days. For example 375 days is equal to 1 year
33. * 1 week and 3 days (ignore Leap year)
34. */
35. #include <stdio.h>
36. #define DAYSINWEEK 7
37.
38. void main()
39. {
40.     int ndays, year, week, days;
41.
42.     printf("Enter the number of daysn");
43.     scanf("%d", &ndays);
44.     year = ndays / 365;
45.     week =(ndays % 365) / DAYSINWEEK;
46.     days =(ndays % 365) % DAYSINWEEK;
47.     printf ("%d is equivalent to %d years, %d weeks and %d daysn",
48.             ndays, year, week, days);
49. }
```

```

1. /*
2. * C Program to Convert Binary to Octal
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     long int binarynum, octalnum = 0, j = 1, remainder;
9.
10.    printf("Enter the value for binary number: ");
11.    scanf("%ld", &binarynum);
12.    while (binarynum != 0)
13.    {
14.        remainder = binarynum % 10;
15.        octalnum = octalnum + remainder * j;
16.        j = j * 2;
17.        binarynum = binarynum / 10;
18.    }
19.    printf("Equivalent octal value: %lo", octalnum);
20.    return 0;
21. }
```

```

1. /*
2. * C Program to Convert Binary to Hexadecimal
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     long int binaryval, hexadecimalval = 0, i = 1, remainder;
9.
10.    printf("Enter the binary number: ");
11.    scanf("%ld", &binaryval);
12.    while (binaryval != 0)
13.    {
14.        remainder = binaryval % 10;
15.        hexadecimalval = hexadecimalval + remainder * i;
16.        i = i * 2;
17.        binaryval = binaryval / 10;
18.    }
19.    printf("Equivalent hexadecimal value: %lx", hexadecimalval);
20.    return 0;
21. }
```

```

1. /*
2. * C program to Convert Decimal to Octal
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     long decimalnum, remainder, quotient;
9.     int octalNumber[100], i = 1, j;
10.
11.    printf("Enter the decimal number: ");
12.    scanf("%ld", &decimalnum);
13.    quotient = decimalnum;
14.    while (quotient != 0)
15.    {
16.        octalNumber[i++] = quotient % 8;
17.        quotient = quotient / 8;
18.    }
19.    printf("Equivalent octal value of decimal no %d: ", decimalnum);
20.    for (j = i - 1; j > 0; j--)
21.        printf("%d", octalNumber[j]);
22.    return 0;
23. }
```

23. }

```

1. /*
2.  * C program to Convert Decimal to Hexadecimal
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     long decimalnum, remainder, quotient;
9.     int i = 1, j, temp;
10.    char hexadecimalnum[100];
11.
12.    printf("Enter decimal number: ");
13.    scanf("%ld", &decimalnum);
14.
15.    quotient = decimalnum;
16.
17.    while (quotient != 0)
18.    {
19.        temp = quotient % 16;
20.        // To convert integer into character
21.        if (temp == 0; j--)
22.            printf("%c", hexadecimalnum[j]);
23.    return 0;
24. }
```

```

1. /*
2.  * C Program to Convert Roman Number to Decimal Number
3.  */
4. #include <stdio.h>
5. #include <string.h>
6.
7. int digit(char);
8.
9. int main()
10. {
11.    char romanval[1000];
12.    int i = 0;
13.    long int number = 0;
14.
15.    printf("Enter roman num (Valid digits are I, V, X, L, C, D, M):\n");
16.    scanf("%s", romanval);
17.    while (romanval[i])
18.    {
```

```
19.         if (digit(romanval[i]) == 2)
20.     {
21.         if (digit(romanval[i]) == digit(romanval[i+1]))
22.             number = number + digit(romanval[i]);
23.         else
24.         {
25.             number = number + (digit(romanval[i + 1]) -
26.                 digit(romanval[i]));
27.             i++;
28.         }
29.         i++;
30.     }
31.     printf("Its decimal value is : %ld", number);
32.     return 0;
33. }
34.
35. int digit(char c)
36. {
37.     int value = 0;
38.     switch (c)
39.     {
40.     case 'I':
41.         value = 1;
42.         break;
43.     case 'V':
44.         value = 5;
45.         break;
46.     case 'X':
47.         value = 10;
48.         break;
49.     case 'L':
50.         value = 50;
51.         break;
52.     case 'C':
53.         value = 100;
54.         break;
55.     case 'D':
56.         value = 500;
57.         break;
58.     case 'M':
59.         value = 1000;
60.         break;
61.     case '0':
62.         value = 0;
63.         break;
64.     default:
65.         value = -1;
```

```
66.     }
67.     return value;
68. }
```

```
1. /*
2.  * C Program to Convert Octal to Binary
3.  */
4. #include <stdio.h>
5. #define MAX 1000
6.
7. int main()
8. {
9.     char octalnum[MAX];
10.    long i = 0;
11.
12.    printf("Enter any octal number: ");
13.    scanf("%s", octalnum);
14.    printf("Equivalent binary value: ");
15.    while (octalnum[i])
16.    {
17.        switch (octalnum[i])
18.        {
19.            case '0':
20.                printf("000"); break;
21.            case '1':
22.                printf("001"); break;
23.            case '2':
24.                printf("010"); break;
25.            case '3':
26.                printf("011"); break;
27.            case '4':
28.                printf("100"); break;
29.            case '5':
30.                printf("101"); break;
31.            case '6':
32.                printf("110"); break;
33.            case '7':
34.                printf("111"); break;
35.            default:
36.                printf("\n Invalid octal digit %c ", octalnum[i]);
37.                return 0;
38.        }
39.        i++;
40.    }
41.    return 0;
42. }
```

```
1. /*
2.  * C Program to Convert Hexadecimal to Binary
3.  */
4. #include <stdio.h>
5. #define MAX 1000
6.
7. int main()
8. {
9.     char binarynum[MAX], hexa[MAX];
10.    long int i = 0;
11.
12.    printf("Enter the value for hexadecimal ");
13.    scanf("%s", hexa);
14.    printf("\n Equivalent binary value: ");
15.    while (hexa[i])
16.    {
17.        switch (hexa[i])
18.        {
19.            case '0':
20.                printf("0000"); break;
21.            case '1':
22.                printf("0001"); break;
23.            case '2':
24.                printf("0010"); break;
25.            case '3':
26.                printf("0011"); break;
27.            case '4':
28.                printf("0100"); break;
29.            case '5':
30.                printf("0101"); break;
31.            case '6':
32.                printf("0110"); break;
33.            case '7':
34.                printf("0111"); break;
35.            case '8':
36.                printf("1000"); break;
37.            case '9':
38.                printf("1001"); break;
39.            case 'A':
40.                printf("1010"); break;
41.            case 'B':
42.                printf("1011"); break;
43.            case 'C':
44.                printf("1100"); break;
45.            case 'D':
```

```

46.         printf("1101"); break;
47.     case 'E':
48.         printf("1110"); break;
49.     case 'F':
50.         printf("1111"); break;
51.     case 'a':
52.         printf("1010"); break;
53.     case 'b':
54.         printf("1011"); break;
55.     case 'c':
56.         printf("1100"); break;
57.     case 'd':
58.         printf("1101"); break;
59.     case 'e':
60.         printf("1110"); break;
61.     case 'f':
62.         printf("1111"); break;
63.     default:
64.         printf("\n Invalid hexa digit %c ", hexa[i]);
65.         return 0;
66.     }
67.     i++;
68. }
69. return 0;
70. }
```

```

1. /*
2.  * C Program to Convert Numbers to Roman Numerals
3.  */
4. #include <stdio.h>
5.
6. void predigit(char num1, char num2);
7. void postdigit(char c, int n);
8.
9. char romanval[1000];
10. int i = 0;
11. int main()
12. {
13.     int j;
14.     long number;
15.
16.     printf("Enter the number: ");
17.     scanf("%d", &number);
18.     if (number <= 0)
19.     {
20.         printf("Invalid number");
```

```
21.         return 0;
22.     }
23.     while (number != 0)
24.     {
25.         if (number >= 1000)
26.         {
27.             postdigit('M', number / 1000);
28.             number = number - (number / 1000) * 1000;
29.         }
30.         else if (number >= 500)
31.         {
32.             if (number < (500 + 4 * 100))
33.             {
34.                 postdigit('D', number / 500);
35.                 number = number - (number / 500) * 500;
36.             }
37.             else
38.             {
39.                 predigit('C','M');
40.                 number = number - (1000-100);
41.             }
42.         }
43.         else if (number >= 100)
44.         {
45.             if (number < (100 + 3 * 100))
46.             {
47.                 postdigit('C', number / 100);
48.                 number = number - (number / 100) * 100;
49.             }
50.             else
51.             {
52.                 predigit('L', 'D');
53.                 number = number - (500 - 100);
54.             }
55.         }
56.         else if (number >= 50 )
57.         {
58.             if (number < (50 + 4 * 10))
59.             {
60.                 postdigit('L', number / 50);
61.                 number = number - (number / 50) * 50;
62.             }
63.             else
64.             {
65.                 predigit('X','C');
66.                 number = number - (100-10);
67.             }
}
```

```
68.     }
69.     else if (number >= 10)
70.     {
71.         if (number < (10 + 3 * 10))
72.         {
73.             postdigit('X', number / 10);
74.             number = number - (number / 10) * 10;
75.         }
76.         else
77.         {
78.             predigit('X', 'L');
79.             number = number - (50 - 10);
80.         }
81.     }
82.     else if (number >= 5)
83.     {
84.         if (number < (5 + 4 * 1))
85.         {
86.             postdigit('V', number / 5);
87.             number = number - (number / 5) * 5;
88.         }
89.         else
90.         {
91.             predigit('I', 'X');
92.             number = number - (10 - 1);
93.         }
94.     }
95.     else if (number >= 1)
96.     {
97.         if (number < 4)
98.         {
99.             postdigit('I', number / 1);
100.            number = number - (number / 1) * 1;
101.        }
102.        else
103.        {
104.            predigit('I', 'V');
105.            number = number - (5 - 1);
106.        }
107.    }
108. }
109. printf("Roman number is: ");
110. for(j = 0; j < i; j++)
111.     printf("%c", romanval[j]);
112. return 0;
113. }
114.
```

```

115.     void predigit(char num1, char num2)
116.     {
117.         romanval[i++] = num1;
118.         romanval[i++] = num2;
119.     }
120.
121.     void postdigit(char c, int n)
122.     {
123.         int j;
124.         for (j = 0; j < n; j++)
125.             romanval[i++] = c;
126.     }

```

```

1. /*
2. * C Program to Convert Octal to Decimal
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main()
8. {
9.
10.    long int octal, decimal = 0;
11.    int i = 0;
12.
13.    printf("Enter any octal number: ");
14.    scanf("%ld", &octal);
15.    while (octal != 0)
16.    {
17.        decimal = decimal +(octal % 10)* pow(8, i++);
18.        octal = octal / 10;
19.    }
20.    printf("Equivalent decimal value: %ld",decimal);
21.    return 0;
22. }

```

```

1. /*
2. * C Program to Convert a Number Decimal System to Binary System using Recursion
3. */
4. #include <stdio.h>
5.
6. int convert(int);
7.
8. int main()
9. {

```

```

10.     int dec, bin;
11.
12.     printf("Enter a decimal number: ");
13.     scanf("%d", &dec);
14.     bin = convert(dec);
15.     printf("The binary equivalent of %d is %d.\n", dec, bin);
16.
17.     return 0;
18. }
19.
20. int convert(int dec)
21. {
22.     if (dec == 0)
23.     {
24.         return 0;
25.     }
26.     else
27.     {
28.         return (dec % 2 + 10 * convert(dec / 2));
29.     }
30. }
```

```

1. /*
2.  * C Program to Convert Binary Code of a Number into its Equivalent
3.  * Gray's Code without using Recursion
4.  */
5. #include <stdio.h>
6. #include <math.h>
7.
8. int bintogray(int);
9.
10. int main ()
11. {
12.     int bin, gray;
13.
14.     printf("Enter a binary number: ");
15.     scanf("%d", &bin);
16.     gray = bintogray(bin);
17.     printf("The gray code of %d is %d\n", bin, gray);
18.     return 0;
19. }
20.
21. int bintogray(int bin)
22. {
23.     int a, b, result = 0, i = 0;
```

```

25.     while (bin != 0)
26.     {
27.         a = bin % 10;
28.         bin = bin / 10;
29.         b = bin % 10;
30.         if ((a && !b) || (!a && b))
31.         {
32.             result = result + pow(10, i);
33.         }
34.         i++;
35.     }
36.     return result;
37. }
```

```

1. /*
2.  * C Program to Convert Binary Code of a Number into its Equivalent
3.  * Gray's Code using Recursion
4. */
5. #include <stdio.h>
6.
7. int bintogray(int);
8.
9. int main ()
10. {
11.     int bin, gray;
12.
13.     printf("Enter a binary number: ");
14.     scanf("%d", &bin);
15.     gray = bintogray(bin);
16.     printf("The gray code of %d is %d\n", bin, gray);
17.     return 0;
18. }
19.
20. int bintogray(int bin)
21. {
22.     int a, b, result = 0, i = 0;
23.
24.     if (!bin)
25.     {
26.         return 0;
27.     }
28.     else
29.     {
30.         a = bin % 10;
31.         bin = bin / 10;
32.         b = bin % 10;
```

```

33.         if ((a && !b) || (!a && b))
34.         {
35.             return (1 + 10 * bintogray(bin));
36.         }
37.     else
38.     {
39.         return (10 * bintogray(bin));
40.     }
41. }
42. }
```

### 3. C Examples on Recursion

```

1. /*
2.  * C Program to find Sum of Digits of a Number using Recursion
3. */
4. #include <stdio.h>
5.
6. int sum (int a);
7.
8. int main()
9. {
10.     int num, result;
11.
12.     printf("Enter the number: ");
13.     scanf("%d", &num);
14.     result = sum(num);
15.     printf("Sum of digits in %d is %d\n", num, result);
16.     return 0;
17. }
18.
19. int sum (int num)
20. {
21.     if (num != 0)
22.     {
23.         return (num % 10 + sum (num / 10));
24.     }
25.     else
26.     {
27.         return 0;
28.     }
29. }
```

```

1. /*
2.  * C program to find the reverse of a number using recursion
3. */
```

```

4. #include <stdio.h>
5. #include <math.h>
6.
7. int rev(int, int);
8.
9. int main()
10. {
11.     int num, result;
12.     int length = 0, temp;
13.
14.     printf("Enter an integer number to reverse: ");
15.     scanf("%d", &num);
16.     temp = num;
17.     while (temp != 0)
18.     {
19.         length++;
20.         temp = temp / 10;
21.     }
22.     result = rev(num, length);
23.     printf("The reverse of %d is %d.\n", num, result);
24.     return 0;
25. }
26.
27. int rev(int num, int len)
28. {
29.     if (len == 1)
30.     {
31.         return num;
32.     }
33.     else
34.     {
35.         return (((num % 10) * pow(10, len - 1)) + rev(num / 10, --len));
36.     }
37. }
```

```

1. /*
2.  * C Program to find Sum of N Numbers using Recursion
3.  */
4. #include <stdio.h>
5.
6. void display(int);
7.
8. int main()
9. {
10.     int num, result;
```

```

12.     printf("Enter the Nth number: ");
13.     scanf("%d", &num);
14.     display(num);
15.     return 0;
16. }
17.
18. void display(int num)
19. {
20.     static int i = 1;
21.
22.     if (num == i)
23.     {
24.         printf("%d\n", num);
25.         return;
26.     }
27.     else
28.     {
29.         printf("%d ", i);
30.         i++;
31.         display(num);
32.     }
33. }
```

```

1. /*
2. * C Program to find whether a Number is Prime or Not using Recursion
3. */
4. #include <stdio.h>
5.
6. int primeno(int, int);
7.
8. int main()
9. {
10.     int num, check;
11.     printf("Enter a number: ");
12.     scanf("%d", &num);
13.     check = primeno(num, num / 2);
14.     if (check == 1)
15.     {
16.         printf("%d is a prime number\n", num);
17.     }
18.     else
19.     {
20.         printf("%d is not a prime number\n", num);
21.     }
22.     return 0;
23. }
```

```

24.
25. int primeno(int num, int i)
26. {
27.     if (i == 1)
28.     {
29.         return 1;
30.     }
31.     else
32.     {
33.         if (num % i == 0)
34.         {
35.             return 0;
36.         }
37.         else
38.         {
39.             return primeno(num, i - 1);
40.         }
41.     }
42. }
```

```

1. /*
2. * C Program to Print Binary Equivalent of an Integer using Recursion
3. */
4. #include <stdio.h>
5.
6. int binary_conversion(int);
7.
8. int main()
9. {
10.     int num, bin;
11.
12.     printf("Enter a decimal number: ");
13.     scanf("%d", &num);
14.     bin = binary_conversion(num);
15.     printf("The binary equivalent of %d is %d\n", num, bin);
16. }
17.
18. int binary_conversion(int num)
19. {
20.     if (num == 0)
21.     {
22.         return 0;
23.     }
24.     else
25.     {
26.         return (num % 2) + 10 * binary_conversion(num / 2);
```

```
27.     }
28. }
```

```
1. /*
2.  * C Program to find Product of 2 Numbers using Recursion
3. */
4. #include <stdio.h>
5.
6. int product(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter two numbers to find their product: ");
13.     scanf("%d%d", &a, &b);
14.     result = product(a, b);
15.     printf("Product of %d and %d is %d\n", a, b, result);
16.     return 0;
17. }
18.
19. int product(int a, int b)
20. {
21.     if (a < b)
22.     {
23.         return product(b, a);
24.     }
25.     else if (b != 0)
26.     {
27.         return (a + product(a, b - 1));
28.     }
29.     else
30.     {
31.         return 0;
32.     }
33. }
```

#### 4. C Examples on Special Numbers

```
1. /*
2.  * C program to find the biggest of three numbers
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
```

```

8.     int num1, num2, num3;
9.
10.    printf("Enter the values of num1, num2 and num3\n");
11.    scanf("%d %d %d", &num1, &num2, &num3);
12.    printf("num1 = %d\nnum2 = %d\nnum3 = %d\n", num1, num2, num3);
13.    if (num1 > num2)
14.    {
15.        if (num1 > num3)
16.        {
17.            printf("num1 is the greatest among three \n");
18.        }
19.        else
20.        {
21.            printf("num3 is the greatest among three \n");
22.        }
23.    }
24.    else if (num2 > num3)
25.        printf("num2 is the greatest among three \n");
26.    else
27.        printf("num3 is the greatest among three \n");
28. }
```

```

1. /*
2. * C program to accept an integer and reverse it
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     long num, reverse = 0, temp, remainder;
9.
10.    printf("Enter the number\n");
11.    scanf("%ld", &num);
12.    temp = num;
13.    while (num > 0)
14.    {
15.        remainder = num % 10;
16.        reverse = reverse * 10 + remainder;
17.        num /= 10;
18.    }
19.    printf("Given number = %ld\n", temp);
20.    printf("Its reverse is = %ld\n", reverse);
21. }
```

```
1. /*
```

```

2.  * C program to reverse a given integer number and check
3.  * whether it is a palindrome. Display the given number
4.  * with appropriate message
5.  */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     int num, temp, remainder, reverse = 0;
11.
12.     printf("Enter an integer \n");
13.     scanf("%d", &num);
14.     /* original number is stored at temp */
15.     temp = num;
16.     while (num > 0)
17.     {
18.         remainder = num % 10;
19.         reverse = reverse * 10 + remainder;
20.         num /= 10;
21.     }
22.     printf("Given number is = %d\n", temp);
23.     printf("Its reverse is = %d\n", reverse);
24.     if (temp == reverse)
25.         printf("Number is a palindrome \n");
26.     else
27.         printf("Number is not a palindrome \n");
28. }
```

```

1. /*
2.  * C Program to Find the Sum of two Binary Numbers
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.
9.     long binary1, binary2;
10.    int i = 0, remainder = 0, sum[20];
11.
12.    printf("Enter the first binary number: ");
13.    scanf("%ld", &binary1);
14.    printf("Enter the second binary number: ");
15.    scanf("%ld", &binary2);
16.    while (binary1 != 0 || binary2 != 0)
17.    {
18.        sum[i++] =(binary1 % 10 + binary2 % 10 + remainder) % 2;
```

```

19.         remainder = (binary1 % 10 + binary2 % 10 + remainder) / 2;
20.         binary1 = binary1 / 10;
21.         binary2 = binary2 / 10;
22.     }
23.     if (remainder != 0)
24.         sum[i++] = remainder;
25.     --i;
26.     printf("Sum of two binary numbers: ");
27.     while (i >= 0)
28.         printf("%d", sum[i--]);
29.     return 0;
30. }
```

```

1. /*
2.  * C Program to Find Multiplication of two Binary Numbers
3.  */
4. #include <stdio.h>
5.
6. int binaryproduct(int, int);
7.
8. int main()
9. {
10.
11.     long binary1, binary2, multiply = 0;
12.     int digit, factor = 1;
13.
14.     printf("Enter the first binary number: ");
15.     scanf("%ld", &binary1);
16.     printf("Enter the second binary number: ");
17.     scanf("%ld", &binary2);
18.     while (binary2 != 0)
19.     {
20.         digit = binary2 % 10;
21.         if (digit == 1)
22.         {
23.             binary1 = binary1 * factor;
24.             multiply = binaryproduct(binary1, multiply);
25.         }
26.         else
27.             binary1 = binary1 * factor;
28.         binary2 = binary2 / 10;
29.         factor = 10;
30.     }
31.     printf("Product of two binary numbers: %ld", multiply);
32.     return 0;
33. }
```

```

34.
35. int binaryproduct(int binary1, int binary2)
36. {
37.     int i = 0, remainder = 0, sum[20];
38.     int binaryprod = 0;
39.
40.     while (binary1 != 0 || binary2 != 0)
41.     {
42.         sum[i++] =(binary1 % 10 + binary2 % 10 + remainder) % 2;
43.         remainder =(binary1 % 10 + binary2 % 10 + remainder) / 2;
44.         binary1 = binary1 / 10;
45.         binary2 = binary2 / 10;
46.     }
47.     if (remainder != 0)
48.         sum[i++] = remainder;
49.     --i;
50.     while (i >= 0)
51.         binaryprod = binaryprod * 10 + sum[i--];
52.     return binaryprod;
53. }
```

```

1. /*
2.  * C Program to find Product of 2 Numbers without using Recursion
3.  */
4.
5. #include <stdio.h>
6.
7. int product(int, int);
8.
9. int main()
10. {
11.     int a, b, result;
12.
13.     printf("Enter two numbers to find their product: ");
14.     scanf("%d%d", &a, &b);
15.     result = product(a, b);
16.     printf("Product of %d and %d is %d\n", a, b, result);
17.     return 0;
18. }
19.
20. int product(int a, int b)
21. {
22.     int temp = 0;
23.
24.     while (b != 0)
25.     {
```

```

26.         temp += a;
27.         b--;
28.     }
29.     return temp;
30. }

31. /*
32. * C Program to Check whether a given Number is Armstrong
33. */
34. #include <stdio.h>
35. #include <math.h>
36.
37. void main()
38. {
39.     int number, sum = 0, rem = 0, cube = 0, temp;
40.
41.     printf ("enter a number");
42.     scanf("%d", &number);
43.     temp = number;
44.     while (number != 0)
45.     {
46.         rem = number % 10;
47.         cube = pow(rem, 3);
48.         sum = sum + cube;
49.         number = number / 10;
50.     }
51.     if (sum == temp)
52.         printf ("The given no is armstrong no");
53.     else
54.         printf ("The given no is not a armstrong no");
55. }

```

```

1. /*
2. * C Program to Check whether a given Number is Armstrong
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. void main()
8. {
9.     int number, sum = 0, rem = 0, cube = 0, temp;
10.
11.    printf ("enter a number");
12.    scanf("%d", &number);
13.    temp = number;
14.    while (number != 0)
15.    {

```

```

16.         rem = number % 10;
17.         cube = pow(rem, 3);
18.         sum = sum + cube;
19.         number = number / 10;
20.     }
21.     if (sum == temp)
22.         printf ("The given no is armstrong no");
23.     else
24.         printf ("The given no is not a armstrong no");
25. }
```

```

1. /*
2.  * C Program to Check whether a given Number is Perfect Number
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int number, rem, sum = 0, i;
9.
10.    printf("Enter a Number\n");
11.    scanf("%d", &number);
12.    for (i = 1; i <= (number - 1); i++)
13.    {
14.        rem = number % i;
15.        if (rem == 0)
16.        {
17.            sum = sum + i;
18.        }
19.    }
20.    if (sum == number)
21.        printf("Entered Number is perfect number");
22.    else
23.        printf("Entered Number is not a perfect number");
24.    return 0;
25. }
```

```

1. /*
2.  * C Program to Add two Complex Numbers
3. */
4. #include <stdio.h>
5.
6. struct complex
7. {
8.     int realpart, imaginary;
```

```

9.  };
10.
11. main()
12. {
13.     struct complex a, b, c;
14.
15.     printf("Enter value of a and b complex number a + ib.\n");
16.     printf("value of complex number a is = ");
17.     scanf("%d", &a.realpart);
18.     printf("value of complex number b is = ");
19.     scanf("%d", &a.imaginary);
20.     printf("Enter value of c and d complex number c + id.\n");
21.     printf("value of complex number c is = ");
22.     scanf("%d", &b.realpart);
23.     printf("value of complex number d is = ");
24.     scanf("%d", &b.imaginary);
25.     c.realpart = a.realpart + b.realpart;
26.     c.imaginary = a.imaginary + b.imaginary;
27.     if (c.imaginary >= 0)
28.         printf("complex numbers sum is = %d + %di\n", c.realpart, c.imaginary);
29.     else
30.         printf("complex numbers sum = %d %di\n", c.realpart, c.imaginary);
31.     return 0;
32. }
```

```

1. /*
2.  * C program to generate Fibonacci Series. Fibonacci Series
3.  * is 0 1 1 2 3 5 8 13 21 ...
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int fib1 = 0, fib2 = 1, fib3, limit, count = 0;
10.
11.    printf("Enter the limit to generate the Fibonacci Series \n");
12.    scanf("%d", &limit);
13.    printf("Fibonacci Series is ... \n");
14.    printf("%d\n", fib1);
15.    printf("%d\n", fib2);
16.    count = 2;
17.    while (count < limit)
18.    {
19.        fib3 = fib1 + fib2;
20.        count++;
21.        printf("%d\n", fib3);
```

```

22.         fib1 = fib2;
23.         fib2 = fib3;
24.     }
25. }
```

```

1. /*
2.  * C Program to Compute First N Fibonacci Numbers using Command Line Arguments
3.  */
4. #include <stdio.h>
5.
6. /* Global Variable Declaration */
7. int first = 0;
8. int second = 1;
9. int third;
10./* Function Prototype */
11.void rec_fibonacci(int);
12.
13.void main(int argc, char *argv[])/* Command Line Arguments*/
14.{
15.    int number = atoi(argv[1]);
16.    printf("%d\t%d", first, second); /* To print first and second number of fibonacci series
   */
17.    rec_fibonacci(number);
18.    printf("\n");
19.}
20.
21./* Code to print fibonacci series using recursive function */
22.void rec_fibonacci(int num)
23.{
24.    if (num == 2) /* To exit the function as the first two numbers are already printed */
25.    {
26.        return;
27.    }
28.    third = first + second;
29.    printf("\t%d", third);
30.    first = second;
31.    second = third;
32.    num--;
33.    rec_fibonacci(num);
34.}
```

```

1. /*
2.  * C program to find the sum of first 50 natural numbers
3.  * using for Loop
4. */
```

```

5. #include <stdio.h>
6.
7. void main()
8. {
9.     int num, sum = 0;
10.
11.    for (num = 1; num <= 50; num++)
12.    {
13.        sum = sum + num;
14.    }
15.    printf("Sum = %4d\n", sum);
16. }
```

```

1. /*
2. * C program to swap the contents of two numbers using bitwise XOR
3. * operation. Don't use either the temporary variable or arithmetic
4. * operators
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.    long i, k;
11.
12.    printf("Enter two integers \n");
13.    scanf("%ld %ld", &i, &k);
14.    printf("\n Before swapping i= %ld and k = %ld", i, k);
15.    i = i ^ k;
16.    k = i ^ k;
17.    i = i ^ k;
18.    printf("\n After swapping i= %ld and k = %ld", i, k);
19. }
```

```

1. /*
2. * C program to swap the contents of two numbers using bitwise XOR
3. * operation. Don't use either the temporary variable or arithmetic
4. * operators
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.    long i, k;
11.
12.    printf("Enter two integers \n");
```

```

13.     scanf("%ld %ld", &i, &k);
14.     printf("\n Before swapping i= %ld and k = %ld", i, k);
15.     i = i ^ k;
16.     k = i ^ k;
17.     i = i ^ k;
18.     printf("\n After swapping i= %ld and k = %ld", i, k);
19. }
```

```

1. /*
2. * C program to multiply given number by 4 using bitwise operators
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     long number, tempnum;
9.
10.    printf("Enter an integer \n");
11.    scanf("%ld", &number);
12.    tempnum = number;
13.    /* left shift by two bits */
14.    number = number << 2;
15.    printf("%ld x 4 = %ld\n", tempnum, number);
16. }
```

## 5. C Examples to illustrate Functions of a Computer

```

1. /*
2. * C program is to illustrate how user authentication is done.
3. * Program asks for the user name and password and displays
4. * the password as '*' character
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     char password[10], username[10], ch;
11.     int i;
12.
13.     printf("Enter User name: ");
14.     gets(username);
15.     printf("Enter the password < any 8 characters>: ");
16.     for (i = 0; i < 8; i++)
17.     {
18.         ch = getchar();
19.         password[i] = ch;
```

```

20.         ch = '*' ;
21.         printf("%c", ch);
22.     }
23.     password[i] = '\0';
24.     /* Original password can be printed, if needed */
25.     printf("\n Your password is :");
26.     for (i = 0; i < 8; i++)
27.     {
28.         printf("%c", password[i]);
29.     }
30. }
```

```

1. /*
2. * C Program to Get IP Address
3. */
4. #include <stdio.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/socket.h>
8. #include <sys/ioctl.h>
9. #include <netinet/in.h>
10. #include <net/if.h>
11. #include <unistd.h>
12. #include <arpa/inet.h>
13.
14. int main()
15. {
16.     int n;
17.     struct ifreq ifr;
18.     char array[] = "eth0";
19.
20.     n = socket(AF_INET, SOCK_DGRAM, 0);
21.     //Type of address to retrieve - IPv4 IP address
22.     ifr.ifr_addr.sa_family = AF_INET;
23.     //Copy the interface name in the ifreq structure
24.     strncpy(ifr.ifr_name , array , IFNAMSIZ - 1);
25.     ioctl(n, SIOCGIFADDR, &ifr);
26.     close(n);
27.     //display result
28.     printf("IP Address is %s - %s\n" , array , inet_ntoa(( (struct sockaddr_in *)&ifr.ifr_addr
    )->sin_addr) );
29.     return 0;
30. }
```

```
1. /*
```

```

2.  * C Program to Shutdown or Turn Off the Computer in Linux.
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     system("shutdown -P now");
9.     return 0;
10. }

```

## 6. C Examples on day-to-day Activities

```

1. /*
2.  * C program to find whether a given year is Leap year or not
3.  */
4. void main()
5. {
6.     int year;
7.
8.     printf("Enter a year \n");
9.     scanf("%d", &year);
10.    if ((year % 400) == 0)
11.        printf("%d is a leap year \n", year);
12.    else if ((year % 100) == 0)
13.        printf("%d is a not leap year \n", year);
14.    else if ((year % 4) == 0)
15.        printf("%d is a leap year \n", year);
16.    else
17.        printf("%d is not a leap year \n", year);
18. }

```

```

1. /*
2.  * C Program to Extract Last two Digits of a given Year
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int year, yr;
9.
10.    printf("Enter the year ");
11.    scanf("%d", &year);
12.    yr = year % 100;
13.    printf("Last two digits of year is: %02d", yr);
14.    return 0;
15. }

```

```
1. /*
2. * C program to display the inventory of items in a store / shop
3. * The inventory maintains details such as name, price, quantity
4. * and manufacturing date of each item.
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     struct date
11.     {
12.         int day;
13.         int month;
14.         int year;
15.     };
16.     struct details
17.     {
18.         char name[20];
19.         int price;
20.         int code;
21.         int qty;
22.         struct date mfg;
23.     };
24.     struct details item[50];
25.     int n, i;
26.
27.     printf("Enter number of items:");
28.     scanf("%d", &n);
29.     fflush(stdin);
30.     for (i = 0; i < n; i++)
31.     {
32.         fflush(stdin);
33.         printf("Item name: \n");
34.         scanf("%s", item[i].name);
35.         fflush(stdin);
36.         printf("Item code: \n");
37.         scanf("%d", &item[i].code);
38.         fflush(stdin);
39.         printf("Quantity: \n");
40.         scanf("%d", &item[i].qty);
41.         fflush(stdin);
42.         printf("price: \n");
43.         scanf("%d", &item[i].price);
44.         fflush(stdin);
45.         printf("Manufacturing date(dd-mm-yyyy): \n");
```

```

46.     scanf("%d-%d-%d", &item[i].mfg.day,
47.            &item[i].mfg.month, &item[i].mfg.year);
48. }
49. printf("***** INVENTORY ***** \n");
50. printf("-----\n");
51. printf("-----\n");
52. printf("S.N. | NAME | CODE | QUANTITY | PRICE\n");
53. printf(" | MFG.DATE |\n");
54. printf("-----\n");
55. printf("-----\n");
56. for (i = 0; i < n; i++)
57. {
58.     printf("%d %15s %d %5d %5d\n",
59.            i + 1, item[i].name, item[i].code, item[i].qty,
60.            item[i].price, item[i].mfg.day, item[i].mfg.month,
61.            item[i].mfg.year);
62. }
63. 
```

```

1. /*
2. * C Program to Display the ATM Transaction
3. */
4. #include <stdio.h>
5.
6. unsigned long amount=1000, deposit, withdraw;
7. int choice, pin, k;
8. char transaction ='y';
9.
10. void main()
11. {
12.     while (pin != 1520)
13.     {
14.         printf("ENTER YOUR SECRET PIN NUMBER:");
15.         scanf("%d", &pin);
16.         if (pin != 1520)
17.             printf("PLEASE ENTER VALID PASSWORD\n");
18.     }
19.     do
20.     {
21.         printf("*****Welcome to ATM Service*****\n");
22.         printf("1. Check Balance\n");
23.         printf("2. Withdraw Cash\n");
24.         printf("3. Deposit Cash\n");
25.         printf("4. Quit\n");
26.         printf("*****\n");
27.         printf("Enter your choice: ");
```

```

28.         scanf("%d", &choice);
29.         switch (choice)
30.         {
31.             case 1:
32.                 printf("\n YOUR BALANCE IN Rs : %lu ", amount);
33.                 break;
34.             case 2:
35.                 printf("\n ENTER THE AMOUNT TO WITHDRAW: ");
36.                 scanf("%lu", &withdraw);
37.                 if (withdraw % 100 != 0)
38.                 {
39.                     printf("\n PLEASE ENTER THE AMOUNT IN MULTIPLES OF 100");
40.                 }
41.                 else if (withdraw >(amount - 500))
42.                 {
43.                     printf("\n INSUFFICIENT BALANCE");
44.                 }
45.                 else
46.                 {
47.                     amount = amount - withdraw;
48.                     printf("\n\n PLEASE COLLECT CASH");
49.                     printf("\n YOUR CURRENT BALANCE IS%lu", amount);
50.                 }
51.                 break;
52.             case 3:
53.                 printf("\n ENTER THE AMOUNT TO DEPOSIT");
54.                 scanf("%lu", &deposit);
55.                 amount = amount + deposit;
56.                 printf("YOUR BALANCE IS %lu", amount);
57.                 break;
58.             case 4:
59.                 printf("\n THANK U USING ATM");
60.                 break;
61.             default:
62.                 printf("\n INVALID CHOICE");
63.             }
64.             printf("\n\n\n DO U WISH TO HAVE ANOTHER TRANSCATION?(y/n): \n");
65.             fflush(stdin);
66.             scanf("%c", &transaction);
67.             if (transaction == 'n'|| transaction == 'N')
68.                 k = 1;
69.         } while (!k);
70.         printf("\n\n THANKS FOR USING OUT ATM SERVICE");
71.     }

```

## 7. C Examples on Union

```

1. /*
2. * C program to accept the height of a person in centimeter and
3. * categorize the person based on height as taller, dwarf and
4. * average height person
5. */
6.
7. #include <stdio.h>
8. void main()
9. {
10.     float height;
11.
12.     printf("Enter the Height (in centimetres) \n");
13.     scanf("%f", &height);
14.     if (height < 150.0)
15.         printf("Dwarf \n");
16.     else if ((height >= 150.0) && (height <= 165.0))
17.         printf(" Average Height \n");
18.     else if ((height >= 165.0) && (height <= 195.0))
19.         printf("Taller \n");
20.     else
21.         printf("Abnormal height \n");
22. }

```

```

1. /*
2. * C Program to accept a grade and declare the equivalent description
3. * if code is S, then print SUPER
4. * if code is A, then print VERY GOOD
5. * if code is B, then print FAIR
6. * if code is Y, then print ABSENT
7. * if code is F, then print FAILS
8. */
9. #include <stdio.h>
10. #include <ctype.h>
11. #include <string.h>
12.
13. void main()
14. {
15.     char remark[15];
16.     char grade;
17.
18.     printf("Enter the grade \n");
19.     scanf("%c", &grade);
20.     /* Lower case letter to upper case */
21.     grade = toupper(grade);
22.     switch(grade)
23.     {

```

```

24.     case 'S':
25.         strcpy(remark, " SUPER");
26.         break;
27.     case 'A':
28.         strcpy(remark, " VERY GOOD");
29.         break;
30.     case 'B':
31.         strcpy(remark, " FAIR");
32.         break;
33.     case 'Y':
34.         strcpy(remark, " ABSENT");
35.         break;
36.     case 'F':
37.         strcpy(remark, " FAILS");
38.         break;
39.     default :
40.         strcpy(remark, "ERROR IN GRADE \n");
41.         break;
42.     }
43.     printf("RESULT : %s\n", remark);
44. }
```

```

1. /*
2. * C program to illustrate the concept of unions
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     union number
9.     {
10.         int n1;
11.         float n2;
12.     };
13.     union number x;
14.
15.     printf("Enter the value of n1: ");
16.     scanf("%d", &x.n1);
17.     printf("Value of n1 = %d", x.n1);
18.     printf("\nEnter the value of n2: ");
19.     scanf("%f", &x.n2);
20.     printf("Value of n2 = %f\n", x.n2);
21. }
```

```
1. /*
```

```

2.  * C Program to display function without using the Main Function
3.  */
4. #include <stdio.h>
5. #define decode(s,t,u,m,p,e,d) m##s##u##t
6. #define begin decode(a,n,i,m,a,t,e)
7.
8. int begin()
9. {
10.     printf(" helloworld ");
11. }
```

```

1. /*
2.  * C Program to Print a Semicolon without using a Semicolon
3.  * anywhere in the code
4. */
5. #include <stdio.h>
6.
7. int main(void)
8. {
9.     //59 is the ascii value of semicolon
10.    if (printf("%c ", 59))
11.    {
12.    }
13.    return 0;
14. }
```

```

1. /*
2.  * C program to Increase 1 to all of the given Integer Digit
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int number, sum = 0, remainder, count;
9.
10.    printf("Enter a number: ");
11.    scanf("%d", &number);
12.    while (number)
13.    {
14.        remainder = number % 10;
15.        sum = sum + (remainder + 1);
16.        number /= 10;
17.    }
18.    printf("increasing 1 to all digits: %d", sum);
19.    return 0;
```

20. }

## 8. C Examples to display Special Patterns

```

1. /*
2.  * C Program to Print Diamond Pattern
3.  */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int number, i, k, count = 1;
9.
10.    printf("Enter number of rows\n");
11.    scanf("%d", &number);
12.    count = number - 1;
13.    for (k = 1; k <= number; k++)
14.    {
15.        for (i = 1; i <= count; i++)
16.            printf(" ");
17.        count--;
18.        for (i = 1; i <= 2 * k - 1; i++)
19.            printf("*");
20.        printf("\n");
21.    }
22.    count = 1;
23.    for (k = 1; k <= number - 1; k++)
24.    {
25.        for (i = 1; i <= count; i++)
26.            printf(" ");
27.        count++;
28.        for (i = 1 ; i <= 2 * (number - k) - 1; i++)
29.            printf("*");
30.        printf("\n");
31.    }
32.    return 0;
33. }
```

```

1. /*
2.  * C Program to Print any Print Statement without using Semicolon
3.  */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     if_printf("Hi.. Welcome to sanfoundry"))
```

```

9.      {
10.    }
11. }
```

```

1. /*
2.  * C Program to Display its own Source Code as its Output
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     FILE *fp;
9.     char ch;
10.
11.    fp = fopen(__FILE__, "r");
12.    do
13.    {
14.        ch = getc(fp);
15.        putchar(ch);
16.    }
17.    while (ch != EOF);
18.    fclose(fp);
19.    return 0;
20. }
```

## 9. C Examples to illustrate Call by Value and Call by Reference

```

1. /*
2.  * C Program to Illustrate Pass by Reference
3. */
4. #include <stdio.h>
5.
6. void cube( int *x);
7.
8. int main()
9. {
10.    int num = 10;
11.
12.    cube(&num);
13.    printf("the cube of the given number is %d", num);
14.    return 0;
15. }
16.
17. void cube(int *x)
18. {
19.    *x = (*x) * (*x) * (*x);
```

20. }

```

1. /*
2.  * C Program to Illustrate Pass by Value.
3.  */
4. #include <stdio.h>
5.
6. void swap(int a, int b)
7. {
8.     int temp;
9.     temp = a;
10.    a = b;
11.    b = temp;
12. }
13.
14. int main()
15. {
16.     int num1 = 10, num2 = 20;
17.
18.     printf("Before swapping num1 = %d num2 = %d\n", num1, num2);
19.     swap(num1, num2);
20.     printf("After swapping num1 = %d num2 = %d \n", num2, num1);
21.     return 0;
22. }
```

## 10. C programs to illustrate the use of Arguments

```

1. /*
2.  * C Program to Input 3 Arguments and Operate Appropriately on the
3.  * Numbers
4.  */
5. #include <stdio.h>
6.
7. void main(int argc, char * argv[])
8. {
9.     int a, b, result;
10.    char ch;
11.
12.    printf("arguments entered: \n");
13.    a = atoi(argv[1]);
14.    b = atoi(argv[2]);
15.    ch = *argv[3];
16.    printf("%d %d %c", a, b, ch);
17.    switch (ch)
18.    {
19.        case '+':
```

```

20.         result = a + b;
21.         break;
22.     case '-':
23.         result = a - b;
24.         break;
25.     case 'x':
26.         result = a * b;
27.         break;
28.     case '/':
29.         result = a / b;
30.         break;
31.     default:
32.         printf("Enter a valid choice");
33.     }
34.     printf("\nThe result of the operation is %d", result);
35.     printf("\n");
36. }
```

```

1. /*
2.  * C Program to Print the Program Name and ALL its Arguments
3.  */
4. #include <stdio.h>
5.
6. void main(int argc, char *argv[]) /* command Line Arguments */
7. {
8.     int i;
9.     for (i = 0;i < argc;i++)
10.    {
11.        printf("%s ", argv[i]); /* Printing the string */
12.    }
13.    printf("\n");
14. }
```

## (2). C Programming Examples on Arrays

### 1. C Examples on Mathematical Operations on an Array

```

1. /*
2.  * C program to read N integers into an array A and
3.  * a) Find the sum of negative numbers
4.  * b) Find the sum of positive numbers
5.  * c) Find the average of all numbers
6.  * Display the results with suitable headings
7.  */
8. #include <stdio.h>
9. #define MAXSIZE 10
```

```

10.
11. void main()
12. {
13.     int array[MAXSIZE];
14.     int i, num, negative_sum = 0, positive_sum = 0;
15.     float total = 0.0, average;
16.
17.     printf ("Enter the value of N \n");
18.     scanf("%d", &num);
19.     printf("Enter %d numbers (-ve, +ve and zero) \n", num);
20.     for (i = 0; i < num; i++)
21.     {
22.         scanf("%d", &array[i]);
23.     }
24.     printf("Input array elements \n");
25.     for (i = 0; i < num; i++)
26.     {
27.         printf("%+3d\n", array[i]);
28.     }
29.     /* Summation starts */
30.     for (i = 0; i < num; i++)
31.     {
32.         if (array[i] < 0)
33.         {
34.             negative_sum = negative_sum + array[i];
35.         }
36.         else if (array[i] > 0)
37.         {
38.             positive_sum = positive_sum + array[i];
39.         }
40.         else if (array[i] == 0)
41.         {
42.             ;
43.         }
44.         total = total + array[i] ;
45.     }
46.     average = total / num;
47.     printf("\n Sum of all negative numbers = %d\n", negative_sum);
48.     printf("Sum of all positive numbers = %d\n", positive_sum);
49.     printf("\n Average of all input numbers = %.2f\n", average);
50. }
```

```

1. /*
2. * C program to read N integers and store them in an array A.
3. * Find the sum of all these elements using pointer.
4. */
```

```

5. #include <stdio.h>
6. #include <malloc.h>
7.
8. void main()
9. {
10.     int i, n, sum = 0;
11.     int *a;
12.
13.     printf("Enter the size of array A \n");
14.     scanf("%d", &n);
15.     a = (int *) malloc(n * sizeof(int));
16.     printf("Enter Elements of First List \n");
17.     for (i = 0; i < n; i++)
18.     {
19.         scanf("%d", a + i);
20.     }
21.     /* Compute the sum of all elements in the given array */
22.     for (i = 0; i < n; i++)
23.     {
24.         sum = sum + *(a + i);
25.     }
26.     printf("Sum of all elements in array = %d\n", sum);
27. }
```

```

1. /*
2.  * C program to find the sum of all elements of an array using
3.  * pointers as arguments.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     static int array[5] = { 200, 400, 600, 800, 1000 };
10.    int sum;
11.
12.    int addnum(int *ptr);
13.
14.    sum = addnum(array);
15.    printf("Sum of all array elements = %5d\n", sum);
16. }
17. int addnum(int *ptr)
18. {
19.     int index, total = 0;
20.     for (index = 0; index < 5; index++)
21.     {
22.         total += *(ptr + index);
```

```

23.     }
24.     return(total);
25. }
```

```

1. /*
2.  * C program to find the sum of all elements of an array using
3.  * pointers as arguments.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     static int array[5] = { 200, 400, 600, 800, 1000 };
10.    int sum;
11.
12.    int addnum(int *ptr);
13.
14.    sum = addnum(array);
15.    printf("Sum of all array elements = %5d\n", sum);
16. }
17. int addnum(int *ptr)
18. {
19.     int index, total = 0;
20.     for (index = 0; index < 5; index++)
21.     {
22.         total += *(ptr + index);
23.     }
24.     return(total);
25. }
```

```

1. /*
2.  * C program to find the sum of two one-dimensional arrays using
3.  * Dynamic Memory Allocation
4. */
5. #include <stdio.h>
6. #include <malloc.h>
7. #include <stdlib.h>
8.
9. void main()
10. {
11.     int i, n;
12.     int *a, *b, *c;
13.
14.     printf("How many Elements in each array...\\n");
15.     scanf("%d", &n);
```

```

16.     a = (int *)malloc(n * sizeof(int));
17.     b = (int *)malloc(n * sizeof(int));
18.     c = (int *)malloc(n * sizeof(int));
19.     printf("Enter Elements of First List\n");
20.     for (i = 0; i < n; i++)
21.     {
22.         scanf("%d", a + i);
23.     }
24.     printf("Enter Elements of Second List\n");
25.     for (i = 0; i < n; i++)
26.     {
27.         scanf("%d", b + i);
28.     }
29.     for (i = 0; i < n; i++)
30.     {
31.         *(c + i) = *(a + i) + *(b + i);
32.     }
33.     printf("Resultant List is\n");
34.     for (i = 0; i < n; i++)
35.     {
36.         printf("%d\n", *(c + i));
37.     }
38. }
```

```

1. /*
2.  * C Program to Find the Sum of Contiguous Subarray within a 1 - D Array of Numbers which has
3.  * the Largest Sum
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. int maxSubArraySum(int a[], int size, int *begin, int *end)
9. {
10.     int max_so_far = 0, max_end = 0;
11.     int i, current_index = 0;
12.
13.     for (i = 0; i < size; i++)
14.     {
15.         max_end = max_end + a[i];
16.         if (max_end <= 0)
17.         {
18.             max_end = 0;
19.             current_index = i + 1;
20.         }
21.         else if (max_so_far < max_end)
22.         {
```

```

22.         max_so_far = max_end;
23.         *begin = current_index;
24.         *end = i;
25.     }
26. }
27. return max_so_far;
28. }
29.
30. int main()
31. {
32.     int arr[] = {10, -2, 15, 9, -8, 12, 20, -5};
33.     int start = 0, end = 0;
34.     int size = sizeof(arr) / sizeof(arr[0]);
35.
36.     printf(" The max sum is %d", maxSubArraySum(arr, size, &start, &end));
37.     printf(" The begin and End are %d & %d", start, end);
38.     getchar();
39.     return 0;
40. }
```

## 2. C Examples on finding the Largest and Smallest numbers in an array

```

1. /*
2.  * C program to read in four integer numbers into an array and find the
3.  * average of largest two of the given numbers without sorting the array.
4.  * The program should output the given four numbers and the average.
5. */
6. #include <stdio.h>
7. #define MAX 4
8.
9. void main()
10. {
11.     int array[MAX], i, largest1, largest2, temp;
12.
13.     printf("Enter %d integer numbers \n", MAX);
14.     for (i = 0; i < MAX; i++)
15.     {
16.         scanf("%d", &array[i]);
17.     }
18.
19.     printf("Input interger are \n");
20.     for (i = 0; i < MAX; i++)
21.     {
22.         printf("%5d", array[i]);
23.     }
24.     printf("\n");
```

```

25.  /* assume first element of array is the first largest */
26.  largest1 = array[0];
27.  /* assume first element of array is the second largest */
28.  largest2 = array[1];
29.  if (largest1 < largest2)
30.  {
31.      temp = largest1;
32.      largest1 = largest2;
33.      largest2 = temp;
34.  }
35.  for (i = 2; i < 4;      i++)
36.  {
37.      if (array[i] >= largest1)
38.      {
39.          largest2 = largest1;
40.          largest1 = array[i];
41.      }
42.      else if (array[i] > largest2)
43.      {
44.          largest2 = array[i];
45.      }
46.  }
47.  printf("n%d is the first largest \n", largest1);
48.  printf("%d is the second largest \n", largest2);
49.  printf("nAverage of %d and %d = %d \n", largest1, largest2,
50. (largest1 + largest2) / 2);
51. }
```

```

1. /*
2. * C program to accept a list of data items and find the second largest
3. * and smallest elements in it. Compute the average of both and search
4. * for the average value if it is present in the array.
5. * Display appropriate message on successful search.
6. */
7. #include <stdio.h>
8.
9. void main ()
10. {
11.     int number[30];
12.     int i, j, a, n, counter, average;
13.
14.     printf("Enter the value of N\n");
15.     scanf("%d", &n);
16.     printf("Enter the numbers \n");
17.     for (i = 0; i < n; ++i)
18.         scanf("%d", &number[i]);
```

```

19.     for (i = 0; i < n; ++i)
20.     {
21.         for (j = i + 1; j < n; ++j)
22.         {
23.             if (number[i] < number[j])
24.             {
25.                 a = number[i];
26.                 number[i] = number[j];
27.                 number[j] = a;
28.             }
29.         }
30.     }
31.     printf("The numbers arranged in descending order are given below \n");
32.     for (i = 0; i < n; ++i)
33.     {
34.         printf("%d\n", number[i]);
35.     }
36.     printf("The 2nd largest number is = %d\n", number[1]);
37.     printf("The 2nd smallest number is = %d\n", number[n - 2]);
38.     average = (number[1] + number[n - 2]) / 2;
39.     counter = 0;
40.     for (i = 0; i < n; ++i)
41.     {
42.         if (average == number[i])
43.         {
44.             ++counter;
45.         }
46.     }
47.     if (counter == 0 )
48.         printf("The average of %d and %d is = %d is not in the array \n",
49.             number[1], number[n - 2], average);
50.     else
51.         printf("The average of %d and %d in array is %d in numbers \n",
52.             number[1], number[n - 2], counter);
53. }
```

```

1. /*
2. * C Program to Find the Largest Number in an Array
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int array[50], size, i, largest;
9.     printf("\n Enter the size of the array: ");
10.    scanf("%d", &size);
```

```

11.     printf("\n Enter %d elements of the array: ", size);
12.     for (i = 0; i < size; i++)
13.         scanf("%d", &array[i]);
14.     largest = array[0];
15.     for (i = 1; i < size; i++)
16.     {
17.         if (largest < array[i])
18.             largest = array[i];
19.     }
20.     printf("\n largest element present in the given array is : %d", largest);
21.     return 0;
22. }
```

### 3. C Examples on inserting and deleting elements to and from an Array

```

1. /*
2.  * C Program to accept N integer number and store them in an array AR.
3.  * The odd elements in the AR are copied into OAR and other elements
4.  * are copied into EAR. Display the contents of OAR and EAR.
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     long int ARR[10], OAR[10], EAR[10];
11.     int i, j = 0, k = 0, n;
12.
13.     printf("Enter the size of array AR \n");
14.     scanf("%d", &n);
15.     printf("Enter the elements of the array \n");
16.     for (i = 0; i < n; i++)
17.     {
18.         scanf("%ld", &ARR[i]);
19.         fflush(stdin);
20.     }
21.     /* Copy odd and even elements into their respective arrays */
22.     for (i = 0; i < n; i++)
23.     {
24.         if (ARR[i] % 2 == 0)
25.         {
26.             EAR[j] = ARR[i];
27.             j++;
28.         }
29.         else
30.         {
31.             OAR[k] = ARR[i];
```

```

32.         k++;
33.     }
34. }
35. printf("The elements of OAR are \n");
36. for (i = 0; i < j; i++)
37. {
38.     printf("%ld\n", OAR[i]);
39. }
40. printf("The elements of EAR are \n");
41. for (i = 0; i < k; i++)
42. {
43.     printf("%ld\n", EAR[i]);
44. }
45. }
```

```

1. /*
2. * C program to insert a particular element in a specified position
3. * in a given array
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int array[10];
10.    int i, j, n, m, temp, key, pos;
11.
12.    printf("Enter how many elements \n");
13.    scanf("%d", &n);
14.    printf("Enter the elements \n");
15.    for (i = 0; i < n; i++)
16.    {
17.        scanf("%d", &array[i]);
18.    }
19.    printf("Input array elements are \n");
20.    for (i = 0; i < n; i++)
21.    {
22.        printf("%d\n", array[i]);
23.    }
24.    for (i = 0; i < n; i++)
25.    {
26.        for (j = i + 1; j < n; j++)
27.        {
28.            if (array[i] > array[j])
29.            {
30.                temp = array[i];
31.                array[i] = array[j];
32.                array[j] = temp;
33.            }
34.        }
35.    }
36. }
```

```

32.             array[j] = temp;
33.         }
34.     }
35. }
36. printf("Sorted list is \n");
37. for (i = 0; i < n; i++)
38. {
39.     printf("%d\n", array[i]);
40. }
41. printf("Enter the element to be inserted \n");
42. scanf("%d", &key);
43. for (i = 0; i < n; i++)
44. {
45.     if (key < array[i])
46.     {
47.         pos = i;
48.         break;
49.     }
50.     if (key > array[n-1])
51.     {
52.         pos = n;
53.         break;
54.     }
55. }
56. if (pos != n)
57. {
58.     m = n - pos + 1 ;
59.     for (i = 0; i <= m; i++)
60.     {
61.         array[n - i + 2] = array[n - i + 1] ;
62.     }
63. }
64. array[pos] = key;
65. printf("Final list is \n");
66. for (i = 0; i < n + 1; i++)
67. {
68.     printf("%d\n", array[i]);
69. }
70. }
```

```

1. /*
2.  * C program to accept an array of integers and delete the
3.  * specified integer from the list
4.  */
5. #include <stdio.h>
6.
```

```

7. void main()
8. {
9.     int vectorx[10];
10.    int i, n, pos, element, found = 0;
11.
12.    printf("Enter how many elements\n");
13.    scanf("%d", &n);
14.    printf("Enter the elements\n");
15.    for (i = 0; i < n; i++)
16.    {
17.        scanf("%d", &vectorx[i]);
18.    }
19.    printf("Input array elements are\n");
20.    for (i = 0; i < n; i++)
21.    {
22.        printf("%d\n", vectorx[i]);
23.    }
24.    printf("Enter the element to be deleted\n");
25.    scanf("%d", &element);
26.    for (i = 0; i < n; i++)
27.    {
28.        if (vectorx[i] == element)
29.        {
30.            found = 1;
31.            pos = i;
32.            break;
33.        }
34.    }
35.    if (found == 1)
36.    {
37.        for (i = pos; i < n - 1; i++)
38.        {
39.            vectorx[i] = vectorx[i + 1];
40.        }
41.        printf("The resultant vector is \n");
42.        for (i = 0; i < n - 1; i++)
43.        {
44.            printf("%d\n", vectorx[i]);
45.        }
46.    }
47.    else
48.        printf("Element %d is not found in the vector\n", element);
49. }
```

```

1. /*
2. * C program to cyclically permute the elements of an array A.
```

```

3.   * i.e. the content of A1 become that of A2. And A2 contains
4.   * that of A3 & so on as An contains A1
5.   */
6. #include <stdio.h>
7.
8. void main ()
9. {
10.    int i, n, number[30];
11.    printf("Enter the value of the n = ");
12.    scanf("%d", &n);
13.    printf("Enter the numbers\n");
14.    for (i = 0; i < n; ++i)
15.    {
16.        scanf("%d", &number[i]);
17.    }
18.    number[n] = number[0];
19.    for (i = 0; i < n; ++i)
20.    {
21.        number[i] = number[i + 1];
22.    }
23.    printf("Cyclically permuted numbers are given below \n");
24.    for (i = 0; i < n; ++i)
25.        printf("%d\n", number[i]);
26. }
```

#### 4. C Examples on Sort and Merge Operations on an Array

```

1. /*
2.  * C program to accept N numbers and arrange them in an ascending order
3.  */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int i, j, a, n, number[30];
9.
10.    printf("Enter the value of N \n");
11.    scanf("%d", &n);
12.    printf("Enter the numbers \n");
13.    for (i = 0; i < n; ++i)
14.        scanf("%d", &number[i]);
15.    for (i = 0; i < n; ++i)
16.    {
17.        for (j = i + 1; j < n; ++j)
18.        {
19.            if (number[i] > number[j])
```

```

20.         {
21.             a = number[i];
22.             number[i] = number[j];
23.             number[j] = a;
24.         }
25.     }
26. }
27. printf("The numbers arranged in ascending order are given below \n");
28. for (i = 0; i < n; ++i)
29.     printf("%d\n", number[i]);
30. }
```

```

1. /*
2. * C program to accept a set of numbers and arrange them
3. * in a descending order
4. */
5. #include <stdio.h>
6.
7. void main ()
8. {
9.     int number[30];
10.    int i, j, a, n;
11.
12.    printf("Enter the value of N\n");
13.    scanf("%d", &n);
14.    printf("Enter the numbers \n");
15.    for (i = 0; i < n; ++i)
16.        scanf("%d", &number[i]);
17.    /* sorting begins ... */
18.    for (i = 0; i < n; ++i)
19.    {
20.        for (j = i + 1; j < n; ++j)
21.        {
22.            if (number[i] < number[j])
23.            {
24.                a = number[i];
25.                number[i] = number[j];
26.                number[j] = a;
27.            }
28.        }
29.    }
30.    printf("The numbers arranged in descending order are given below\n");
31.    for (i = 0; i < n; ++i)
32.    {
33.        printf("%d\n", number[i]);
34.    }
```

35. }

```
1. /*
2.  * C program to read N names, store them in the form of an array
3.  * and sort them in alphabetical order. Output the given names and
4.  * the sorted names in two columns side by side.
5. */
6. #include <stdio.h>
7. #include <string.h>
8.
9. void main()
10. {
11.     char name[10][8], tname[10][8], temp[8];
12.     int i, j, n;
13.
14.     printf("Enter the value of n \n");
15.     scanf("%d", &n);
16.     printf("Enter %d names n", \n);
17.     for (i = 0; i < n; i++)
18.     {
19.         scanf("%s", name[i]);
20.         strcpy(tname[i], name[i]);
21.     }
22.     for (i = 0; i < n - 1 ; i++)
23.     {
24.         for (j = i + 1; j < n; j++)
25.         {
26.             if (strcmp(name[i], name[j]) > 0)
27.             {
28.                 strcpy(temp, name[i]);
29.                 strcpy(name[i], name[j]);
30.                 strcpy(name[j], temp);
31.             }
32.         }
33.     }
34.     printf("\n-----\n");
35.     printf("Input NamesSorted names\n");
36.     printf("-----\n");
37.     for (i = 0; i < n; i++)
38.     {
39.         printf("%s\t\t%s\n", tname[i], name[i]);
40.     }
41.     printf("-----\n");
42. }
```

```

1. /*
2.  * C Program to Merge and Sort Elements of 2 different arrays
3. */
4. #include <stdio.h>
5.
6. void Merge(int * , int , int , int );
7.
8. void MergeSort(int *array, int left, int right)
9. {
10.     int middle = (left+right)/2;
11.     /* We have to sort only when left<right because when Left=right it is anyhow sorted*/
12.     if(left<right)
13.     {
14.         /* Sort the Left part */
15.         MergeSort(array, left, middle);
16.         /* Sort the right part */
17.         MergeSort(array, middle + 1, right);
18.         /* Merge the two sorted parts */
19.         Merge(array, left, middle, right);
20.     }
21. }
22. /* Merge functions merges the two sorted parts */
23. void Merge(int *array, int left, int middle, int right)
24. {
25.     /*to store sorted array*/
26.     int tmp[right - left + 1];
27.     int pos = 0, leftposition = left, rightposition = middle + 1;
28.     while (leftposition <= middle && rightposition <= right)
29.     {
30.         if (array[leftposition] < array[rightposition])
31.         {
32.             tmp[pos++] = array[leftposition++];
33.         }
34.         else
35.         {
36.             tmp[pos++] = array[rightposition++];
37.         }
38.     }
39.     while (leftposition <= middle)
40.         tmp[pos++] = array[leftposition++];
41.     while (rightposition <= right)
42.         tmp[pos++] = array[rightposition++];
43.     int i;
44.     /* Copy back the sorted array to the original array */
45.     for (i = 0; i < pos; i++)
46.     {
47.         array[i + left] = tmp[i];

```

```

48.     }
49.     return;
50. }
51. int main()
52. {
53.     int size;
54.     printf("\n enter the size of an array");
55.     scanf("%d", &size);
56.     int array[size];
57.     int i, j, k;
58.     printf("\n enter the array elements");
59.     for (i = 0; i < size; i++)
60.     {
61.         scanf("%d", &array[i]);
62.     }
63. /* Calling this functions sorts the array */
64. MergeSort(array, 0, size - 1);
65. for (i = 0; i < size; i++)
66. {
67.     printf("%d ", array[i]);
68. }
69. printf("\n");
70. return 0;
71. }
```

```

1. /*
2. * C Program to Merge the Elements of 2 Sorted Array
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array1[50], array2[50], array3[100], m, n, i, j, k = 0;
9.
10.    printf("\n Enter size of array Array 1: ");
11.    scanf("%d", &m);
12.    printf("\n Enter sorted elements of array 1: \n");
13.    for (i = 0; i < m; i++)
14.    {
15.        scanf("%d", &array1[i]);
16.    }
17.    printf("\n Enter size of array 2: ");
18.    scanf("%d", &n);
19.    printf("\n Enter sorted elements of array 2: \n");
20.    for (i = 0; i < n; i++)
21.    {
```

```

22.         scanf("%d", &array2[i]);
23.     }
24.     i = 0;
25.     j = 0;
26.     while (i < m && j < n)
27.     {
28.         if (array1[i] < array2[j])
29.         {
30.             array3[k] = array1[i];
31.             i++;
32.         }
33.         else
34.         {
35.             array3[k] = array2[j];
36.             j++;
37.         }
38.         k++;
39.     }
40.     if (i >= m)
41.     {
42.         while (j < n)
43.         {
44.             array3[k] = array2[j];
45.             j++;
46.             k++;
47.         }
48.     }
49.     if (j >= n)
50.     {
51.         while (i < m)
52.         {
53.             array3[k] = array1[i];
54.             i++;
55.             k++;
56.         }
57.     }
58.     printf("\n After merging: \n");
59.     for (i = 0; i < m + n; i++)
60.     {
61.         printf("\n%d", array3[i]);
62.     }
63. }
```

```

1. /*
2. * C Program to Merge the Elements of 2 Sorted Array
3. */
```

```
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array1[50], array2[50], array3[100], m, n, i, j, k = 0;
9.
10.    printf("\n Enter size of array Array 1: ");
11.    scanf("%d", &m);
12.    printf("\n Enter sorted elements of array 1: \n");
13.    for (i = 0; i < m; i++)
14.    {
15.        scanf("%d", &array1[i]);
16.    }
17.    printf("\n Enter size of array 2: ");
18.    scanf("%d", &n);
19.    printf("\n Enter sorted elements of array 2: \n");
20.    for (i = 0; i < n; i++)
21.    {
22.        scanf("%d", &array2[i]);
23.    }
24.    i = 0;
25.    j = 0;
26.    while (i < m && j < n)
27.    {
28.        if (array1[i] < array2[j])
29.        {
30.            array3[k] = array1[i];
31.            i++;
32.        }
33.        else
34.        {
35.            array3[k] = array2[j];
36.            j++;
37.        }
38.        k++;
39.    }
40.    if (i >= m)
41.    {
42.        while (j < n)
43.        {
44.            array3[k] = array2[j];
45.            j++;
46.            k++;
47.        }
48.    }
49.    if (j >= n)
50.    {
```

```

51.     while (i < m)
52.     {
53.         array3[k] = array1[i];
54.         i++;
55.         k++;
56.     }
57. }
58. printf("\n After merging: \n");
59. for (i = 0; i < m + n; i++)
60. {
61.     printf("\n%d", array3[i]);
62. }
63. }
```

```

1. /*
2. * C program to sort N numbers in ascending order using Bubble sort
3. * and print both the given and the sorted array
4. */
5. #include <stdio.h>
6. #define MAXSIZE 10
7.
8. void main()
9. {
10.     int array[MAXSIZE];
11.     int i, j, num, temp;
12.
13.     printf("Enter the value of num \n");
14.     scanf("%d", &num);
15.     printf("Enter the elements one by one \n");
16.     for (i = 0; i < num; i++)
17.     {
18.         scanf("%d", &array[i]);
19.     }
20.     printf("Input array is \n");
21.     for (i = 0; i < num; i++)
22.     {
23.         printf("%d\n", array[i]);
24.     }
25.     /* Bubble sorting begins */
26.     for (i = 0; i < num; i++)
27.     {
28.         for (j = 0; j < (num - i - 1); j++)
29.         {
30.             if (array[j] > array[j + 1])
31.             {
32.                 temp = array[j];
```

```

33.         array[j] = array[j + 1];
34.         array[j + 1] = temp;
35.     }
36. }
37. }
38. printf("Sorted array is...\n");
39. for (i = 0; i < num; i++)
40. {
41.     printf("%d\n", array[i]);
42. }
43. }
```

```

1. /*
2.  * C program to read an array, accept a key & split the array.
3.  * Add the first half to the end of second half.
4. */
5. #include <stdio.h>
6.
7. void main ()
8. {
9.     int number[30];
10.    int i, n, a, j;
11.
12.    printf("Enter the value of n\n");
13.    scanf("%d", &n);
14.    printf("enter the numbers\n");
15.    for (i = 0; i < n; ++i)
16.        scanf("%d", &number[i]);
17.    printf("Enter the position of the element to split the array \n");
18.    scanf("%d", &a);
19.    for (i = 0; i < a; ++i)
20.    {
21.        number[n] = number[0];
22.        for (j = 0; j < n; ++j)
23.        {
24.            number[j] = number[j + 1];
25.        }
26.    }
27.    printf("The resultant array is\n");
28.    for (i = 0; i < n; ++i)
29.    {
30.        printf("%d\n", number[i]);
31.    }
32. }
```

```

1. /*
2. * C program to accept an array of 10 elements and swap 3rd element
3. * with 4th element using pointers and display the results.
4. */
5. #include <stdio.h>
6. void swap34(float *ptr1, float *ptr2);
7.
8. void main()
9. {
10.     float x[10];
11.     int i, n;
12.
13.     printf("How many Elements...\\n");
14.     scanf("%d", &n);
15.     printf("Enter Elements one by one\\n");
16.     for (i = 0; i < n; i++)
17.     {
18.         scanf("%f", x + i);
19.     }
20.     /* Function call:Interchanging 3rd element by 4th */
21.     swap34(x + 2, x + 3);
22.     printf("\\nResultant Array...\\n");
23.     for (i = 0; i < n; i++)
24.     {
25.         printf("X[%d] = %f\\n", i, x[i]);
26.     }
27. }
28. /* Function to swap the 3rd element with the 4th element in the array */
29. void swap34(float *ptr1, float *ptr2 )
30. {
31.     float temp;
32.     temp = *ptr1;
33.     *ptr1 = *ptr2;
34.     *ptr2 = temp;
35. }
```

## 5. C Examples on Search operation on an Array

```

1. /*
2. * C program accept an array of N elements and a key to search.
3. * If the search is successful, it displays "SUCCESSFUL SEARCH".
4. * Otherwise, a message "UNSUCCESSFUL SEARCH" is displayed.
5. */
6. #include <stdio.h>
7.
8. void main()
```

```

9. {
10.     int array[20];
11.     int i, low, mid, high, key, size;
12.
13.     printf("Enter the size of an array\n");
14.     scanf("%d", &size);
15.     printf("Enter the array elements\n");
16.     for (i = 0; i < size; i++)
17.     {
18.         scanf("%d", &array[i]);
19.     }
20.     printf("Enter the key\n");
21.     scanf("%d", &key);
22.     /* search begins */
23.     low = 0;
24.     high = (size - 1);
25.     while (low <= high)
26.     {
27.         mid = (low + high) / 2;
28.         if (key == array[mid])
29.         {
30.             printf("SUCCESSFUL SEARCH\n");
31.             return;
32.         }
33.         if (key < array[mid])
34.             high = mid - 1;
35.         else
36.             low = mid + 1;
37.     }
38.     printf("UNSUCCESSFUL SEARCH\n");
39. }
```

```

1. /*
2. * C program to accept N numbers sorted in ascending order
3. * and to search for a given number using binary search.
4. * Report success or failure.
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     int array[10];
11.     int i, j, num, temp, keynum;
12.     int low, mid, high;
13.
14.     printf("Enter the value of num \n");
```

```

15.     scanf("%d", &num);
16.     printf("Enter the elements one by one \n");
17.     for (i = 0; i < num; i++)
18.     {
19.         scanf("%d", &array[i]);
20.     }
21.     printf("Input array elements \n");
22.     for (i = 0; i < num; i++)
23.     {
24.         printf("%d\n", array[i]);
25.     }
26. /* Bubble sorting begins */
27. for (i = 0; i < num; i++)
28. {
29.     for (j = 0; j < (num - i - 1); j++)
30.     {
31.         if (array[j] > array[j + 1])
32.         {
33.             temp = array[j];
34.             array[j] = array[j + 1];
35.             array[j + 1] = temp;
36.         }
37.     }
38. }
39. printf("Sorted array is...\n");
40. for (i = 0; i < num; i++)
41. {
42.     printf("%d\n", array[i]);
43. }
44. printf("Enter the element to be searched \n");
45. scanf("%d", &keynum);
46. /* Binary searching begins */
47. low = 1;
48. high = num;
49. do
50. {
51.     mid = (low + high) / 2;
52.     if (keynum < array[mid])
53.         high = mid - 1;
54.     else if (keynum > array[mid])
55.         low = mid + 1;
56. } while (keynum != array[mid] && low <= high);
57. if (keynum == array[mid])
58. {
59.     printf("SEARCH SUCCESSFUL \n");
60. }
61. else

```

```

62.     {
63.         printf("SEARCH FAILED \n");
64.     }
65. }
```

```

1.      * C Program to find the Biggest Number in an Array of Numbers using
2.      * Recursion
3.      */
4. #include <stdio.h>
5.
6. int large(int[], int, int);
7.
8. int main()
9. {
10.     int size;
11.     int largest;
12.     int list[20];
13.     int i;
14.
15.     printf("Enter size of the list:");
16.     scanf("%d", &size);
17.     printf("Printing the list:\n");
18.     for (i = 0; i < size ; i++)
19.     {
20.         list[i] = rand() % size;
21.         printf("%d\t", list[i]);
22.     }
23.     if (size == 0)
24.     {
25.         printf("Empty list\n");
26.     }
27.     else
28.     {
29.         largest = list[0];
30.         largest = large(list, size - 1, largest);
31.         printf("\nThe largest number in the list is: %d\n", largest);
32.     }
33. }
34. int large(int list[], int size, int largest)
35. {
36.     if (size == 1)
37.         return largest;
38.
39.     if (size > -1)
40.     {
```

```

41.         if (list[size] > largest)
42.         {
43.             largest = list[size];
44.         }
45.         return(largest = large(list, size - 1, largest));
46.     }
47. else
48. {
49.     return largest;
50. }
51. }
```

## 6. C Examples on Queue and Stack Implementation of an Array

```

1. /*
2. * C Program to Implement a Queue using an Array
3. */
4. #include <stdio.h>
5.
6. #define MAX 50
7. int queue_array[MAX];
8. int rear = - 1;
9. int front = - 1;
10. main()
11. {
12.     int choice;
13.     while (1)
14.     {
15.         printf("1.Insert element to queue \n");
16.         printf("2.Delete element from queue \n");
17.         printf("3.Display all elements of queue \n");
18.         printf("4.Quit \n");
19.         printf("Enter your choice : ");
20.         scanf("%d", &choice);
21.         switch (choice)
22.         {
23.             case 1:
24.                 insert();
25.                 break;
26.             case 2:
27.                 delete();
28.                 break;
29.             case 3:
30.                 display();
31.                 break;
32.             case 4:
```

```
33.         exit(1);
34.     default:
35.         printf("Wrong choice \n");
36.     } /*End of switch*/
37. } /*End of while*/
38. } /*End of main()*/
39. insert()
40. {
41.     int add_item;
42.     if (rear == MAX - 1)
43.         printf("Queue Overflow \n");
44.     else
45.     {
46.         if (front == - 1)
47.             /*If queue is initially empty */
48.             front = 0;
49.         printf("Inset the element in queue : ");
50.         scanf("%d", &add_item);
51.         rear = rear + 1;
52.         queue_array[rear] = add_item;
53.     }
54. } /*End of insert()*/
55.
56. delete()
57. {
58.     if (front == - 1 || front > rear)
59.     {
60.         printf("Queue Underflow \n");
61.         return ;
62.     }
63.     else
64.     {
65.         printf("Element deleted from queue is : %d\n", queue_array[front]);
66.         front = front + 1;
67.     }
68. } /*End of delete() */
69. display()
70. {
71.     int i;
72.     if (front == - 1)
73.         printf("Queue is empty \n");
74.     else
75.     {
76.         printf("Queue is : \n");
77.         for (i = front; i <= rear; i++)
78.             printf("%d ", queue_array[i]);
79.         printf("\n");
```

```
80.     }
81. } /*End of display() */
```

```
1. #include <stdio.h>
2. #define SIZE 10
3.
4.
5. int ar[SIZE];
6. int top1 = -1;
7. int top2 = SIZE;
8.
9. //Functions to push data
10. void push_stack1 (int data)
11. {
12.     if (top1 < top2 - 1)
13.     {
14.         ar[++top1] = data;
15.     }
16.     else
17.     {
18.         printf ("Stack Full! Cannot Push\n");
19.     }
20. }
21. void push_stack2 (int data)
22. {
23.     if (top1 < top2 - 1)
24.     {
25.         ar[--top2] = data;
26.     }
27.     else
28.     {
29.         printf ("Stack Full! Cannot Push\n");
30.     }
31. }
32.
33. //Functions to pop data
34. void pop_stack1 ()
35. {
36.     if (top1 >= 0)
37.     {
38.         int popped_value = ar[top1--];
39.         printf ("%d is being popped from Stack 1\n", popped_value);
40.     }
41.     else
42.     {
```

```
43.     printf ("Stack Empty! Cannot Pop\n");
44. }
45. }
46. void pop_stack2 ()
47. {
48.     if (top2 < SIZE)
49.     {
50.         int popped_value = ar[top2++];
51.         printf ("%d is being popped from Stack 2\n", popped_value);
52.     }
53.     else
54.     {
55.         printf ("Stack Empty! Cannot Pop\n");
56.     }
57. }
58.
59. //Functions to Print Stack 1 and Stack 2
60. void print_stack1 ()
61. {
62.     int i;
63.     for (i = top1; i >= 0; --i)
64.     {
65.         printf ("%d ", ar[i]);
66.     }
67.     printf ("\n");
68. }
69. void print_stack2 ()
70. {
71.     int i;
72.     for (i = top2; i < SIZE; ++i)
73.     {
74.         printf ("%d ", ar[i]);
75.     }
76.     printf ("\n");
77. }
78.
79. int main()
80. {
81.     int ar[SIZE];
82.     int i;
83.     int num_of_ele;
84.
85.     printf ("We can push a total of 10 values\n");
86.
87.     //Number of elements pushed in stack 1 is 6
88.     //Number of elements pushed in stack 2 is 4
89.
```

```

90.     for (i = 1; i <= 6; ++i)
91.     {
92.         push_stack1 (i);
93.         printf ("Value Pushed in Stack 1 is %d\n", i);
94.     }
95.     for (i = 1; i <= 4; ++i)
96.     {
97.         push_stack2 (i);
98.         printf ("Value Pushed in Stack 2 is %d\n", i);
99.     }
100.
101.        //Print Both Stacks
102.        print_stack1 ();
103.        print_stack2 ();
104.
105.        //Pushing on Stack Full
106.        printf ("Pushing Value in Stack 1 is %d\n", 11);
107.        push_stack1 (11);
108.
109.        //Popping All Elements From Stack 1
110.        num_of_ele = top1 + 1;
111.        while (num_of_ele)
112.        {
113.            pop_stack1 ();
114.            --num_of_ele;
115.        }
116.
117.        //Trying to Pop From Empty Stack
118.        pop_stack1 ();
119.
120.        return 0;
121.    }

```

## 7. C Examples on Printing the elements of an Array

```

1. /*
2. * C Program to Generate Pascal Triangle 1 D Array
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array[30], temp[30], i, j, k, l, num;           //using 2 arrays
9.
10.    printf("Enter the number of lines to be printed: ");
11.    scanf("%d", &num);

```

```

12.     temp[0] = 1;
13.     array[0] = 1;
14.     for (j = 0; j < num; j++)
15.         printf(" ");
16.     printf(" 1\n");
17.     for (i = 1; i < num; i++)
18.     {
19.         for (j = 0; j < i; j++)
20.             printf(" ");
21.         for (k = 1; k < num; k++)
22.         {
23.             array[k] = temp[k - 1] + temp[k];
24.         }
25.         array[i] = 1;
26.         for (l = 0; l <= i; l++)
27.         {
28.             printf("%3d", array[l]);
29.             temp[l] = array[l];
30.         }
31.         printf("\n");
32.     }
33. }
```

```

1. /*
2.  * C Program to Print the Number of Odd & Even Numbers in an Array
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array[100], i, num;
9.
10.    printf("Enter the size of an array \n");
11.    scanf("%d", &num);
12.    printf("Enter the elements of the array \n");
13.    for (i = 0; i < num; i++)
14.    {
15.        scanf("%d", &array[i]);
16.    }
17.    printf("Even numbers in the array are - ");
18.    for (i = 0; i < num; i++)
19.    {
20.        if (array[i] % 2 == 0)
21.        {
22.            printf("%d \t", array[i]);
23.        }
24.    }
25. }
```

```

24.     }
25.     printf("\n Odd numbers in the array are -");
26.     for (i = 0; i < num; i++)
27.     {
28.         if (array[i] % 2 != 0)
29.         {
30.             printf("%d \t", array[i]);
31.         }
32.     }
33. }
```

```

1. /*
2.  * C Program to Print all the Repeated Numbers with Frequency in an Array
3. */
4. #include <stdio.h>
5. #include <malloc.h>
6.
7. void duplicate(int array[], int num)
8. {
9.     int *count = (int *)calloc(sizeof(int), (num - 2));
10.    int i;
11.
12.    printf("duplicate elements present in the given array are ");
13.    for (i = 0; i < num; i++)
14.    {
15.        if (count[array[i]] == 1)
16.            printf(" %d ", array[i]);
17.        else
18.            count[array[i]]++;
19.    }
20. }
21.
22. int main()
23. {
24.     int array[] = {5, 10, 10, 2, 1, 4, 2};
25.     int array_freq = sizeof(array) / sizeof(array[0]);
26.     duplicate(array, array_freq);
27.     getchar();
28.     return 0;
29. }
```

```

1. /*
2.  * C Program to Print all the Repeated Numbers with Frequency in an Array
3. */
4. #include <stdio.h>
```

```

5. #include <malloc.h>
6.
7. void duplicate(int array[], int num)
8. {
9.     int *count = (int *)calloc(sizeof(int), (num - 2));
10.    int i;
11.
12.    printf("duplicate elements present in the given array are ");
13.    for (i = 0; i < num; i++)
14.    {
15.        if (count[array[i]] == 1)
16.            printf(" %d ", array[i]);
17.        else
18.            count[array[i]]++;
19.    }
20. }
21.
22. int main()
23. {
24.     int array[] = {5, 10, 10, 2, 1, 4, 2};
25.     int array_freq = sizeof(array) / sizeof(array[0]);
26.     duplicate(array, array_freq);
27.     getchar();
28.     return 0;
29. }
```

## 8. C Examples on Operations on Individual elements of an Array

```

1. /*
2.  * C Program to Find the Number of Elements in an Array
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <unistd.h>
7.
8. int main()
9. {
10.     int array[] = {15, 50, 34, 20, 10, 79, 100};
11.     int n;
12.
13.     n = sizeof(array);
14.     printf("Size of the given array is %d\n", n/sizeof(int));
15.     return 0;
16. }
```

```

1. /*
2. * C Program to Check Array bounds while Inputing Elements into the Array
3. */
4. #include <stdio.h>
5.
6. int main(void)
7. {
8.     int array[5], b, c;
9.     for (b = 0; b < 10 && (scanf("%d", &c)); b++)
10.         array[b] = c;
11.     for (b = 0; b < 15; b++)
12.         printf("%d ", array[b]);
13.     return 0;
14. }

```

```

1. /*
2. * C Program to Print the Alternate Elements in an Array
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array[10];
9.     int i, j, temp;
10.    printf("enter the element of an array \n");
11.    for (i = 0; i < 10; i++)
12.        scanf("%d", &array[i]);
13.    printf("Alternate elements of a given array \n");
14.    for (i = 0; i < 10; i += 2)
15.        printf( "%d\n", array[i]) ;
16. }

```

```

1. /*
2. * C Program to Find the Odd Element given an Array with only two Different Element
3. */
4. #include <stdio.h>
5.
6. void printodd(int array[], int size)
7. {
8.     int xor2 = array[0]; /* Will hold XOR of two odd occurring elements */
9.     int set;
10.    int i;
11.    int n = size - 2;
12.    int x = 0, y = 0;
13. }

```

```

14.     /* The xor will basically be xor of two odd occurring elements */
15.     for (i = 1; i < size; i++)
16.         xor2 = xor2 ^ array[i];
17.     /* Get one set rightmost bit in the xor2. */
18.     set = xor2 & ~(xor2 - 1);
19.     /* Now divide elements in two sets: */
20.     for (i = 0; i < size; i++)
21.     {
22.         /* XOR of first set is finally going to hold one odd occurring number x */
23.         if (array[i] & set)
24.             x = x ^ array[i];
25.         /* XOR of second set is finally going to hold the other odd occurring number y */
26.         else
27.             y = y ^ array[i];
28.     }
29.     printf("\n The ODD elements are %d & %d ", x, y);
30. }
31.
32. int main()
33. {
34.     int array[] = {10, 3, 2, 10, 2, 8, 8, 7};
35.     int arr_size = sizeof(array) / sizeof(array[0]);
36.     printodd(array, arr_size);
37.     getchar();
38.     return 0;
39. }
```

```

1. /*
2.  * C Program to Increment every Element of the Array by one & Print Incremented Array
3.  */
4. #include <stdio.h>
5.
6. void incrementArray(int[]);
7.
8. void main()
9. {
10.     int i;
11.     int array[4] = {10, 20, 30, 40};
12.
13.     incrementArray(array);
14.     for (i = 0; i < 4; i++)
15.         printf("%d\t", array[i]); // Prints 2, 3, 4, 5
16. }
17.
18. void incrementArray(int arr[])
19. {
```

```

20.     int i;
21.
22.     for (i = 0; i < 4; i++)
23.         arr[i]++;      // this alters values in array in main()
24. }
25. /*
26. * C Program to Find the Number of Non Repeated Elements in an Array
27. */
28. #include <stdio.h>
29. int main()
30. {
31.     int array[50];
32.     int *ptr;
33.     int i, j, k, size, n;
34.
35.     printf("\n Enter size of the array: ");
36.     scanf("%d", &n);
37.     printf("\n Enter %d elements of an array: ", n);
38.     for (i = 0; i < n; i++)
39.         scanf("%d", &array[i]);
40.     size = n;
41.     ptr = array;
42.     for (i = 0; i < size; i++)
43.     {
44.         for (j = 0; j < size; j++)
45.         {
46.             if (i == j)
47.             {
48.                 continue;
49.             }
50.             else if (*(ptr + i) == *(ptr + j))
51.             {
52.                 k = j;
53.                 size--;
54.                 while (k < size)
55.                 {
56.                     *(ptr + k) = *(ptr + k + 1);
57.                     k++;
58.                 }
59.                 j = 0;
60.             }
61.         }
62.     }
63.     printf("\n The array after removing duplicates is: ");
64.     for (i = 0; i < size; i++)
65.     {
66.         printf(" %d", array[i]);

```

```

67.     }
68.     return 0;
69. }
```

```

1. /*
2.  * C Program to identify missing numbers in a given array
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int n, i, j, c, t, b;
9.
10.    printf("Enter size of array : ");
11.    scanf("%d", &n);
12.    int array[n - 1]; /* array size-1 */
13.    printf("Enter elements into array : \n");
14.    for (i = 0; i < n - 1; i++)
15.        scanf("%d", &array[i]);
16.    b = array[0];
17.    for (i = 1; i < n - 1; i++)
18.        b = b ^ array[i];
19.    for (i = 2, c = 1; i <= n; i++)
20.        c = c ^ i;
21.    c = c ^ b;
22.    printf("Missing element is : %d \n", c);
23. }
```

```

1. /*
2.  * C Program to Segregate 0s on Left Side & 1s on right side of the Array (Traverse Array only
3.  once)
4. */
5.
6. /*Function to segregate all 0s on left and all 1s on right*/
7. void segregate0and1(int array[], int size)
8. {
9.     int left = 0, right = size-1;
10.
11.    while (left < right)
12.    {
13.        /* Increment Left index while we see 0 at left */
14.        while (array[left] == 0 && left < right)
15.            left++;
16.        /* Decrement right index while we see 1 at right */
```

```

17.         while (array[right] == 1 && left < right)
18.             right--;
19.         /* If left is smaller than right then there is a 1 at left and a 0 at right. Exchange
   it */
20.         if (left < right)
21.         {
22.             array[left] = 0;
23.             array[right] = 1;
24.             left++;
25.             right--;
26.         }
27.     }
28. }
29.
30. int main()
31. {
32.     int arr[] = {0, 1, 0, 1, 1, 0};
33.     int array_size = 6, i = 0;
34.
35.     segregate0and1(arr, array_size);
36.     printf("segregated array is ");
37.     for (i = 0; i < 6; i++)
38.         printf("%d ", arr[i]);
39.     getchar();
40.     return 0;
41. }
```

```

1. /*
2. * C Program to Find 2 Elements in the Array such that Difference between them is Largest
3. */
4. #include <stdio.h>
5.
6. int maximum_difference(int array[], int arr_size)
7. {
8.     int max_diff = array[1] - array[0];
9.     int i, j;
10.    for (i = 0; i < arr_size; i++)
11.    {
12.        for (j = i + 1; j < arr_size; j++)
13.        {
14.            if (array[j] - array[i] > max_diff)
15.                max_diff = array[j] - array[i];
16.        }
17.    }
18.    return max_diff;
19. }
```

```

20.
21. int main()
22. {
23.     int array[] = {10, 15, 90, 200, 110};
24.     printf("Maximum difference is %d", maximum_difference(array, 5));
25.     getchar();
26.     return 0;
27. }

```

## 9. C Examples on Mathematical Functions and their applications on Arrays

```

1. /*
2.  * C Program to Input a String & Store their Ascii Values in an Integer Array & Print the
3.  * Array
4. */
5.
6. void main()
7. {
8.     char string[20];
9.     int n, count = 0;
10.    printf("Enter the no of characters present in an array \n ");
11.    scanf("%d", &n);
12.    printf(" Enter the string of %d characters \n ", n);
13.    scanf("%s", string);
14.    while (count < n)
15.    {
16.        printf(" %c = %d\n", string[count], string[count] );
17.        ++ count ;
18.    }
19. }

```

```

1. /*
2.  * C Program to Input an Array, Store the Squares of these Elements in an Array & Print it
3.  */
4. #include <stdio.h>
5. #define MAX_ROWS 3
6. #define MAX_COLS 4
7.
8. void print_square(int [ ] );
9.
10. void main (void)
11. {
12.     int i;

```

```

13.     int num[MAX_ROWS][MAX_COLS] = { {10, 20, 30, 40}, {50, 60, 70, 80}, {90, 100, 110, 120}
14. };
15.     for (i = 0; i < MAX_ROWS; i++)
16.         print_square(num[i]);
17. }
18. void print_square(int x[])
19. {
20.     int j;
21.     for (j = 0; j < MAX_COLS; j++)
22.         printf ("%d\t", x[j] * x[j]);
23.     printf("\n");
24. }
```

```

1. /*
2.  * C Program to Find the two Elements such that their Sum is Closest to Zero
3. */
4. # include <stdio.h>
5. # include <stdlib.h>
6. # include <math.h>
7.
8. void minabsvaluepair(int array[], int array_size)
9. {
10.     int count = 0;
11.     int l, r, min_sum, sum, min_l, min_r;
12.
13.     /* Array should have at Least two elements*/
14.     if (array_size < 2)
15.     {
16.         printf("Invalid Input");
17.         return;
18.     }
19.
20.     /* Initialization of values */
21.     min_l = 0;
22.     min_r = 1;
23.     min_sum = array[0] + array[1];
24.     for (l = 0; l < array_size - 1; l++)
25.     {
26.         for (r = l + 1; r < array_size; r++)
27.         {
28.             sum = array[l] + array[r];
29.             if (abs(min_sum) > abs(sum))
30.             {
31.                 min_sum = sum;
32.                 min_l = l;
```

```

33.             min_r = r;
34.         }
35.     }
36. }
37. printf(" The two elements whose sum is minimum are %d and %d", array[min_l],
   array[min_r]);
38. }
39.
40. int main()
41. {
42.     int array[] = {42, 15, -25, 30, -10, 35};
43.     minabsvaluepair(array, 6);
44.     getchar();
45.     return 0;
46. }
```

```

1. /*
2.  * C Program to Find if a given Integer X appears more than N/2 times in a Sorted Array of N
   Integers
3. */
4. # include <stdio.h>
5. # define bool int
6.
7. bool Morenooftimes(int array[], int n, int x)
8. {
9.     int i;
10.    int final_index = n % 2 ? n / 2 : (n / 2 + 1);
11.
12.    for (i = 0; i < final_index; i++)
13.    {
14.        /* check if x is presents more than n/2 times */
15.        if (array[i] == x && array[i + n / 2] == x)
16.            return 1;
17.    }
18.    return 0;
19. }
20.
21. int main()
22. {
23.     int array[] = {10, 15, 15, 12, 17, 15};
24.     int n = sizeof(array) / sizeof(array[0]);
25.     int x = 15;
26.     if (Morenooftimes(array, n, x))
27.         printf("The given no %d appears more than %d times in array[]", x, n/2);
28.     else
29.         printf("The given no %d does not appear more than %d times in array[]", x, n/2);
```

```
30.     getchar();
31.     return 0;
32. }
```

```
1. /*
2.  * C Program to Find the Median of the Elements after Merging these 2 Sorted Arrays with Same
3.  * Size
4. */
5.
6. int getMedian(int array1[], int array2[], int n)
7. {
8.     int i = 0; /* Current index of i/p array array1[] */
9.     int j = 0; /* Current index of i/p array array2[] */
10.    int count;
11.    int m1 = -1, m2 = -1;
12.
13.    for (count = 0; count <= n; count++)
14.    {
15.        if (i == n)
16.        {
17.            m1 = m2;
18.            m2 = array2[0];
19.            break;
20.        }
21.        else if (j == n)
22.        {
23.            m1 = m2;
24.            m2 = array1[0];
25.            break;
26.        }
27.        if (array1[i] < array2[j])
28.        {
29.            m1 = m2; /* Store the prev median */
30.            m2 = array1[i];
31.            i++;
32.        }
33.        else
34.        {
35.            m1 = m2; /* Store the prev median */
36.            m2 = array2[j];
37.            j++;
38.        }
39.    }
40.    return (m1 + m2)/2;
41. }
```

```

42.
43. int main()
44. {
45.     int array1[] = {20, 25, 35, 30, 38};
46.     int array2[] = {22, 53, 65, 72, 45};
47.
48.     int n1 = sizeof(array1) / sizeof(array1[0]);
49.     int n2 = sizeof(array2) / sizeof(array2[0]);
50.     if (n1 == n2)
51.         printf("Median is %d", getMedian(array1, array2, n1));
52.     else
53.         printf("not possible to findout");
54.     getchar();
55.     return 0;
56. }
```

```

1. /*
2.  * C Program to Find Union & Intersection of 2 Arrays
3.  */
4. #include <stdio.h>
5. #define SIZE 5
6.
7. void get_value(int arr[]);
8. void print_value(int arr[], int n);
9. void function_sort(int arr[]);
10. int find_intersection(int array1[], int array2[], int intersection_array[]);
11. int find_union(int array1[], int array2[], int union_array[]);
12.
13. void main()
14. {
15.     int array1[SIZE], array2[SIZE], intersection_array[SIZE], union_array[SIZE*2];
16.     int num_elements;
17.
18.     //input elements of Array1
19.     printf("\n Enter the elements of Array 1: n");
20.     get_value(array1);
21.     printf("\n\n Elements of Array 1: ");
22.     print_value(array1, SIZE);
23.
24.     //Sort array 1
25.     function_sort(array1);
26.     printf("nnSorted elements of Array 1: ");
27.     print_value(array1, SIZE);
28.
29.     //input elements of Array2
30.     printf("nEnter the elements of Array 2: n");
```

```
31.     get_value(array2);
32.     printf("\n\n Elements of Array 2: ");
33.     print_value(array2, SIZE);
34.
35.     //Sort array 2
36.     function_sort(array2);
37.     printf("\n\nSorted elements of Array 2: ");
38.     print_value(array2, SIZE);
39.
40.     //Find Intersection
41.     num_elements = find_intersection(array1, array2, intersection_array);
42.     printf("\n\n Intersection is: ");
43.     print_value(intersection_array, num_elements);
44.
45.     //Find Union
46.     num_elements = find_union(array1, array2, union_array);
47.     printf("\n\n Union is: ");
48.     print_value(union_array, num_elements);
49. }
50.
51. void get_value(int arr[])
52. {
53.     int i, j;
54.     for (i = 0; i < SIZE; i++)
55.     {
56.         j = i + 1;
57.         printf("\n Enter element %d: ", j);
58.         scanf("%d", &arr[i]);
59.     }
60. }
61.
62. void print_value(int arr[], int n)
63. {
64.     int i;
65.     printf("{ ");
66.     for (i = 0; i < n; i++)
67.     {
68.         printf("%d ", arr[i]);
69.     }
70.     printf("}");
71. }
72.
73. void function_sort(int arr[])
74. {
75.     int i, j, temp, swapping;
76.
77.     for (i = 1; i < size; i++)
```

```

78.     {
79.         swapping = 0;
80.         for (j = 0; j < size-i; j++)
81.         {
82.             if (arr[j] > arr[j+1])
83.             {
84.                 temp = arr[j];
85.                 arr[j] = arr[j + 1];
86.                 arr[j + 1] = temp;
87.                 swapping = 1;
88.             }
89.         }
90.         if (swapping == 0)
91.         {
92.             break;
93.         }
94.     }
95. }
96.
97. int find_intersection(int array1[], int array2[], int intersection_array[])
98. {
99.     int i = 0, j = 0, k = 0;
100.    while ((i < size) && (j < size))
101.    {
102.        if (array1[i] < array2[j])
103.        {
104.            i++;
105.        }
106.        else if (array1[i] > array2[j])
107.        {
108.            j++;
109.        }
110.        else
111.        {
112.            intersection_array[k] = array1[i];
113.            i++;
114.            j++;
115.            k++;
116.        }
117.    }
118.    return(k);
119. }
120.
121. int find_union(int array1[], int array2[], int union_array[])
122. {
123.     int i = 0, j = 0, k = 0;
124.     while ((i < SIZE) && (j < SIZE))

```

```

125.     {
126.         if (array1[i] < array2[j])
127.         {
128.             union_array[k] = array1[i];
129.             i++;
130.             k++;
131.         }
132.         else if (array1[i] > array2[j])
133.         {
134.             union_array[k] = array2[j];
135.             j++;
136.             k++;
137.         }
138.         else
139.         {
140.             union_array[k] = array1[i];
141.             i++;
142.             j++;
143.             k++;
144.         }
145.     }
146.     if (i == SIZE)
147.     {
148.         while (j < SIZE)
149.         {
150.             union_array[k] = array2[j];
151.             j++;
152.             k++;
153.         }
154.     }
155.     else
156.     {
157.         while (i < SIZE)
158.         {
159.             union_array[k] = array1[i];
160.             i++;
161.             k++;
162.         }
163.     }
164.     return(k);
165. }

```

```

1. /*
2. * C Program to Find Ceiling & Floor of X given a Sorted Array & a value X
3. */
4. #include <stdio.h>

```

```

5.
6. /* Function to get index of ceiling of x in arr[low..high] */
7. int ceilSearch(int arr[], int low, int high, int x)
8. {
9.     int i;
10.
11.    /* If x is smaller than or equal to first element, then return the first element */
12.    if (x <= arr[low])
13.        return low;
14.
15.    /* Otherwise, linearly search for ceil value */
16.    for (i = low; i < high; i++)
17.    {
18.        if (arr[i] == x)
19.            return i;
20.
21.        /* if x lies between arr[i] and arr[i+1] including arr[i+1], then return arr[i+1] */
22.        if (arr[i] < x && arr[i + 1] >= x)
23.            return i + 1;
24.    }
25.
26.    /* If we reach here then x is greater than the last element of the array, return -1 in
   this case */
27.    return -1;
28. }
29.
30. int main()
31. {
32.     int arr[] = {1, 2, 8, 10, 10, 12, 19};
33.     int n = sizeof(arr)/sizeof(arr[0]);
34.     int x = 3;
35.     int index = ceilSearch(arr, 0, n-1, x);
36.     if (index == -1)
37.         printf("Ceiling of %d doesn't exist in array ", x);
38.     else
39.         printf("ceiling of %d is %d", x, arr[index]);
40.     getchar();
41.     return 0;
42. }
```

```

1. /*
2. * C Program to Find the Summation of Node values at Level/row and print it
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
```

```
7. struct btnode
8. {
9.     int value;
10.    struct btnode *r,*l;
11. }*root = NULL, *temp = NULL;
12.
13. void create();
14. void insert();
15. void add(struct btnode *t);
16. void computesum(struct btnode *t);
17. void display();
18.
19. int count = 0, sum[100] = {0}, max = 0;
20.
21. void main()
22. {
23.     int ch;
24.
25.     printf("\n OPERATIONS ---");
26.     printf("\n 1] Insert an element into tree");
27.     printf("\n 2] Display the sum of the elements at the same level");
28.     printf("\n 3] Exit ");
29.     while (1)
30.     {
31.         printf("\nEnter your choice : ");
32.         scanf("%d", &ch);
33.         switch (ch)
34.         {
35.             case 1:
36.                 insert();
37.                 break;
38.             case 2:
39.                 count = 0;
40.                 max = 0;
41.                 computesum(root);
42.                 display();
43.                 break;
44.             case 3:
45.                 exit(0);
46.             default :
47.                 printf("Wrong choice, Please enter correct choice   ");
48.                 break;
49.         }
50.     }
51. }
52.
53. /* To create a new node with the data from the user */
```

```

54. void create()
55. {
56.     int data;
57.
58.     printf("Enter the data of node : ");
59.     scanf("%d", &data);
60.     temp = (struct btnode* ) malloc(1*(sizeof(struct btnode)));
61.     temp->value = data;
62.     temp->l = temp->r = NULL;
63. }
64.
65. /* To check for root node and then create it */
66. void insert()
67. {
68.     create();
69.
70.     if (root == NULL)
71.         root = temp;
72.     else
73.         add(root);
74. }
75.
76. /* Search for the appropriate position to insert the new node */
77. void add(struct btnode *t)
78. {
79.     if ((temp->value > t->value) && (t->r != NULL))           /* value more than root node value
   insert at right */
80.         add(t->r);
81.     else if ((temp->value > t->value) && (t->r == NULL))
82.         t->r = temp;
83.     else if ((temp->value < t->value) && (t->l != NULL))           /* value less than root node
   value insert at left */
84.         add(t->l);
85.     else if ((temp->value < t->value) && (t->l==NULL))
86.         t->l = temp;
87. }
88.
89. /* Function to find the sum of nodes at same distance */
90. void computesum(struct btnode *t)
91. {
92.     if (root == NULL)
93.     {
94.         printf("Tree is empty ");
95.         return;
96.     }
97.     if (t->l != NULL)
98.     {

```

```

99.         count++;
100.        computesum(t->l);
101.    }
102.    sum[count] = sum[count] + t->value; /* addition of element by row wise */
103.    if (max < count)
104.        max = count;
105.    if (t->r != NULL)
106.    {
107.        count++;
108.        computesum(t->r);
109.    }
110.    count--;
111. }
112.
113. /* To display the sum of the nodes at the same distance */
114. void display()
115. {
116.     int i;
117.
118.     printf("Sum of nodes : \n Level \t Sum ");
119.     for (i = 0; i <= max; i++)
120.         printf("\n %d \t: %d ", i, sum[i]);
121. }
```

### (3). C Programming Examples on Matrix

#### 1. C Examples on Matrix Operations

```

1. /*
2. * C program to read two matrices A(MxN) and B(MxN) and perform addition
3. * OR subtraction of A and B. Also, find the trace of the resultant
4. * matrix. Display the given matrices, their sum or differences and
5. * the trace.
6. */
7. #include <stdio.h>
8. void trace(int arr[][10], int m, int n);
9.
10. void main()
11. {
12.     int array1[10][10], array2[10][10], arraysum[10][10],
13.     arraydiff[10][10];
14.     int i, j, m, n, option;
15.
16.     printf("Enter the order of the matrix array1 and array2 \n");
17.     scanf("%d %d", &m, &n);
18.     printf("Enter the elements of matrix array1 \n");
19.     for (i = 0; i < m; i++)
20.     {
```

```
21.         for (j = 0; j < n; j++)
22.         {
23.             scanf("%d", &array1[i][j]);
24.         }
25.     }
26.     printf("MATRIX array1 is \n");
27.     for (i = 0; i < m; i++)
28.     {
29.         for (j = 0; j < n; j++)
30.         {
31.             printf("%3d", array1[i][j]);
32.         }
33.         printf("\n");
34.     }
35.     printf("Enter the elements of matrix array2 \n");
36.     for (i = 0; i < m; i++)
37.     {
38.         for (j = 0; j < n; j++)
39.         {
40.             scanf("%d", &array2[i][j]);
41.         }
42.     }
43.     printf("MATRIX array2 is \n");
44.     for (i = 0; i < m; i++)
45.     {
46.         for (j = 0; j < n; j++)
47.         {
48.             printf("%3d", array2[i][j]);
49.         }
50.         printf("\n");
51.     }
52.     printf("Enter your option: 1 for Addition and 2 for Subtraction \n");
53.     scanf("%d", &option);
54.     switch (option)
55.     {
56.     case 1:
57.         for (i = 0; i < m; i++)
58.         {
59.             for (j = 0; j < n; j++)
60.             {
61.                 arraysum[i][j] = array1[i][j] + array2[i][j];
62.             }
63.         }
64.         printf("Sum matrix is \n");
65.         for (i = 0; i < m; i++)
66.         {
67.             for (j = 0; j < n; j++)
```

```

68.         {
69.             printf("%3d", arraysum[i][j]) ;
70.         }
71.         printf("\n");
72.     }
73.     trace (arraysum, m, n);
74.     break;
75. case 2:
76.     for (i = 0; i < m; i++)
77.     {
78.         for (j = 0; j < n; j++)
79.         {
80.             arraydiff[i][j] = array1[i][j] - array2[i][j];
81.         }
82.     }
83.     printf("Difference matrix is \n");
84.     for (i = 0; i < m; i++)
85.     {
86.         for (j = 0; j < n; j++)
87.         {
88.             printf("%3d", arraydiff[i][j]) ;
89.         }
90.         printf("\n");
91.     }
92.     trace (arraydiff, m, n);
93.     break;
94. }
95.
96. }
97. /* Function to find the trace of a given matrix and print it */
98. void trace (int arr[][10], int m, int n)
99. {
100.     int i, j, trace = 0;
101.     for (i = 0; i < m; i++)
102.     {
103.         for (j = 0; j < n; j++)
104.         {
105.             if (i == j)
106.             {
107.                 trace = trace + arr[i][j];
108.             }
109.         }
110.     }
111.     printf("Trace of the resultant matrix is = %d\n", trace);
112. }
```

```

1. /*
2.  * C program to accept a matrix of order MxN and find its transpose
3.  */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     static int array[10][10];
9.     int i, j, m, n;
10.
11.    printf("Enter the order of the matrix \n");
12.    scanf("%d %d", &m, &n);
13.    printf("Enter the coefficients of the matrix\n");
14.    for (i = 0; i < m; ++i)
15.    {
16.        for (j = 0; j < n; ++j)
17.        {
18.            scanf("%d", &array[i][j]);
19.        }
20.    }
21.    printf("The given matrix is \n");
22.    for (i = 0; i < m; ++i)
23.    {
24.        for (j = 0; j < n; ++j)
25.        {
26.            printf(" %d", array[i][j]);
27.        }
28.        printf("\n");
29.    }
30.    printf("Transpose of matrix is \n");
31.    for (j = 0; j < n; ++j)
32.    {
33.        for (i = 0; i < m; ++i)
34.        {
35.            printf(" %d", array[i][j]);
36.        }
37.        printf("\n");
38.    }
39. }

```

```

1. /*
2.  * Develop functions to read a matrix, display a matrix and compute
3.  * product of two matrices.
4.  * Use these functions to read two MxN matrices and compute their
5.  * product & display the result
6.  */

```

```
7. #include <stdio.h>
8. #define MAXROWS 10
9. #define MAXCOLS 10
10.
11. void readMatrix(int arr[][MAXCOLS], int m, int n);
12. void printMatrix(int arr[][MAXCOLS], int m, int n);
13. void productMatrix(int array1[][MAXCOLS], int array2[][MAXCOLS],
14. int array3[][MAXCOLS], int m, int n);
15.
16. void main()
17. {
18.     int array1[MAXROWS][MAXCOLS], array2[MAXROWS][MAXCOLS],
19.     array3[MAXROWS][MAXCOLS];
20.     int m, n;
21.
22.     printf("Enter the value of m and n \n");
23.     scanf("%d %d", &m, &n);
24.     printf("Enter Matrix array1 \n");
25.     readMatrix(array1, m, n);
26.     printf("Matrix array1 \n");
27.     printMatrix(array1, m, n);
28.     printf("Enter Matrix array2 \n");
29.     readMatrix(array2, m, n);
30.     printf("Matrix B \n");
31.     printMatrix(array2, m, n);
32.     productMatrix(array1, array2, array3, m, n);
33.     printf("The product matrix is \n");
34.     printMatrix(array3, m, n);
35. }
36. /* Input Matrix array1 */
37. void readMatrix(int arr[][MAXCOLS], int m, int n)
38. {
39.     int i, j;
40.     for (i = 0; i < m; i++)
41.     {
42.         for (j = 0; j < n; j++)
43.         {
44.             scanf("%d", &arr[i][j]);
45.         }
46.     }
47. }
48. void printMatrix(int arr[][MAXCOLS], int m, int n)
49. {
50.     int i, j;
51.     for (i = 0; i < m; i++)
52.     {
53.         for (j = 0; j < n; j++)
```

```

54.         {
55.             printf("%3d", arr[i][j]);
56.         }
57.         printf("\n");
58.     }
59. }
60. /* Multiplication of matrices */
61. void productMatrix(int array1[][MAXCOLS], int array2[][MAXCOLS],
62. int array3[][MAXCOLS], int m, int n)
63. {
64.     int i, j, k;
65.     for (i = 0; i < m; i++)
66.     {
67.         for (j = 0; j < n; j++)
68.         {
69.             array3[i][j] = 0;
70.             for (k = 0; k < n; k++)
71.             {
72.                 array3[i][j] = array3[i][j] + array1[i][k] *
73.                     array2[k][j];
74.             }
75.         }
76.     }
77. }
```

```

1. /*
2. * C program to accept two matrices and find the sum
3. * and difference of the matrices
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void readmatA();
9. void printmatA();
10. void readmatB();
11. void printmatB();
12. void sum();
13. void diff();
14.
15. int a[10][10], b[10][10], sumarray[10][10], arraydiff[10][10];
16. int i, j, row1, column1, row2, column2;
17.
18. void main()
19. {
20.     printf("Enter the order of the matrix A \n");
21.     scanf("%d %d", &row1, &column1);
```

```
22.     printf("Enter the order of the matrix B \n");
23.     scanf("%d %d", &row2, &column2);
24.     if (row1 != row2 && column1 != column2)
25.     {
26.         printf("Addition and subtraction are possible \n");
27.         exit(1);
28.     }
29. else
30. {
31.     printf("Enter the elements of matrix A \n");
32.     readmatA();
33.     printf("MATRIX A is \n");
34.     printmatA();
35.     printf("Enter the elements of matrix B \n");
36.     readmatB();
37.     printf("MATRIX B is \n");
38.     printmatB();
39.     sum();
40.     diff();
41. }
42. */
43. /* Function to read a matrix A */
44. void readmatA()
45. {
46.     for (i = 0; i < row1; i++)
47.     {
48.         for (j = 0; j < column1; j++)
49.         {
50.             scanf("%d", &a[i][j]);
51.         }
52.     }
53.     return;
54. }
55. /* Function to read a matrix B */
56. void readmatB()
57. {
58.     for (i = 0; i < row2; i++)
59.     {
60.         for (j = 0; j < column2; j++)
61.         {
62.             scanf("%d", &b[i][j]);
63.         }
64.     }
65. }
66. /* Function to print a matrix A */
67. void printmatA()
68. {
```

```

69.     for (i = 0; i < row1; i++)
70.     {
71.         for (j = 0; j < column1; j++)
72.         {
73.             printf("%3d", a[i][j]);
74.         }
75.         printf("\n");
76.     }
77. }
78. /* Function to print a matrix B */
79. void printmatB()
80. {
81.     for (i = 0; i < row2; i++)
82.     {
83.         for (j = 0; j < column2; j++)
84.         {
85.             printf("%3d", b[i][j]);
86.         }
87.         printf("\n");
88.     }
89. }
90. /* Function to do the sum of elements of matrix A and Matrix B */
91. void sum()
92. {
93.     for (i = 0; i < row1; i++)
94.     {
95.         for (j = 0; j < column2; j++)
96.         {
97.             sumarray[i][j] = a[i][j] + b[i][j];
98.         }
99.     }
100.    printf("Sum matrix is \n");
101.    for (i = 0; i < row1; i++)
102.    {
103.        for (j = 0; j < column2; j++)
104.        {
105.            printf("%3d", sumarray[i][j]) ;
106.        }
107.        printf("\n");
108.    }
109.    return;
110. }
111. /* Function to do the difference of elements of matrix A and Matrix B */
112. void diff()
113. {
114.     for (i = 0; i < row1; i++)
115.     {

```

```

116.         for (j = 0; j < column2; j++)
117.         {
118.             arraydiff[i][j] = a[i][j] - b[i][j];
119.         }
120.     }
121.     printf("Difference matrix is \n");
122.     for (i = 0; i < row1; i++)
123.     {
124.         for (j = 0; j < column2; j++)
125.         {
126.             printf("%3d", arraydiff[i][j]);
127.         }
128.         printf("\n");
129.     }
130.     return;
131. }
```

```

1. /*
2.  * C Program to Perform Matrix Multiplication using Recursion
3. */
4. #include <stdio.h>
5.
6. void multiply(int, int, int [][][10], int, int, int [][][10], int [][][10]);
7. void display(int, int, int[][][10]);
8.
9. int main()
10. {
11.     int a[10][10], b[10][10], c[10][10] = {0};
12.     int m1, n1, m2, n2, i, j, k;
13.
14.     printf("Enter rows and columns for Matrix A respectively: ");
15.     scanf("%d%d", &m1, &n1);
16.     printf("Enter rows and columns for Matrix B respectively: ");
17.     scanf("%d%d", &m2, &n2);
18.     if (n1 != m2)
19.     {
20.         printf("Matrix multiplication not possible.\n");
21.     }
22.     else
23.     {
24.         printf("Enter elements in Matrix A:\n");
25.         for (i = 0; i < m1; i++)
26.             for (j = 0; j < n1; j++)
27.             {
28.                 scanf("%d", &a[i][j]);
29.             }

```

```

30.         printf("\nEnter elements in Matrix B:\n");
31.         for (i = 0; i < m2; i++)
32.             for (j = 0; j < n2; j++)
33.             {
34.                 scanf("%d", &b[i][j]);
35.             }
36.             multiply(m1, n1, a, m2, n2, b, c);
37.         }
38.     printf("On matrix multiplication of A and B the result is:\n");
39.     display(m1, n2, c);
40. }
41.
42. void multiply (int m1, int n1, int a[10][10], int m2, int n2, int b[10][10], int c[10][10])
43. {
44.     static int i = 0, j = 0, k = 0;
45.
46.     if (i >= m1)
47.     {
48.         return;
49.     }
50.     else if (i < m1)
51.     {
52.         if (j < n2)
53.         {
54.             if (k < n1)
55.             {
56.                 c[i][j] += a[i][k] * b[k][j];
57.                 k++;
58.                 multiply(m1, n1, a, m2, n2, b, c);
59.             }
60.             k = 0;
61.             j++;
62.             multiply(m1, n1, a, m2, n2, b, c);
63.         }
64.         j = 0;
65.         i++;
66.         multiply(m1, n1, a, m2, n2, b, c);
67.     }
68. }
69.
70. void display(int m1, int n2, int c[10][10])
71. {
72.     int i, j;
73.
74.     for (i = 0; i < m1; i++)
75.     {
76.         for (j = 0; j < n2; j++)

```

```

77.     {
78.         printf("%d  ", c[i][j]);
79.     }
80.     printf("\n");
81. }
82. }

83. /*
84. * C Program to Perform Matrix Multiplication using Recursion
85. */
86. #include <stdio.h>
87.
88. void multiply(int, int, int [][][10], int, int, int [][][10], int [][][10]);
89. void display(int, int, int[][][10]);
90.
91. int main()
92. {
93.     int a[10][10], b[10][10], c[10][10] = {0};
94.     int m1, n1, m2, n2, i, j, k;
95.
96.     printf("Enter rows and columns for Matrix A respectively: ");
97.     scanf("%d%d", &m1, &n1);
98.     printf("Enter rows and columns for Matrix B respectively: ");
99.     scanf("%d%d", &m2, &n2);
100.    if (n1 != m2)
101.    {
102.        printf("Matrix multiplication not possible.\n");
103.    }
104.    else
105.    {
106.        printf("Enter elements in Matrix A:\n");
107.        for (i = 0; i < m1; i++)
108.            for (j = 0; j < n1; j++)
109.            {
110.                scanf("%d", &a[i][j]);
111.            }
112.        printf("\nEnter elements in Matrix B:\n");
113.        for (i = 0; i < m2; i++)
114.            for (j = 0; j < n2; j++)
115.            {
116.                scanf("%d", &b[i][j]);
117.            }
118.        multiply(m1, n1, a, m2, n2, b, c);
119.    }
120.    printf("On matrix multiplication of A and B the result is:\n");
121.    display(m1, n2, c);
122. }
123.

```

```

124. void multiply (int m1, int n1, int a[10][10], int m2, int n2, int b[10][10], int
125.   c[10][10])
126. {
127.     static int i = 0, j = 0, k = 0;
128.     if (i >= m1)
129.     {
130.         return;
131.     }
132.     else if (i < m1)
133.     {
134.         if (j < n2)
135.         {
136.             if (k < n1)
137.             {
138.                 c[i][j] += a[i][k] * b[k][j];
139.                 k++;
140.                 multiply(m1, n1, a, m2, n2, b, c);
141.             }
142.             k = 0;
143.             j++;
144.             multiply(m1, n1, a, m2, n2, b, c);
145.         }
146.         j = 0;
147.         i++;
148.         multiply(m1, n1, a, m2, n2, b, c);
149.     }
150. }
151.
152. void display(int m1, int n2, int c[10][10])
153. {
154.     int i, j;
155.
156.     for (i = 0; i < m1; i++)
157.     {
158.         for (j = 0; j < n2; j++)
159.         {
160.             printf("%d ", c[i][j]);
161.         }
162.         printf("\n");
163.     }
164. }
```

## 2. C Examples on Matrix Types

```
1. /*
```

```
2.  * C Program to accept two matrices and check if they are equal
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void main()
8. {
9.     int a[10][10], b[10][10];
10.    int i, j, row1, column1, row2, column2, flag = 1;
11.
12.    printf("Enter the order of the matrix A \n");
13.    scanf("%d %d", &row1, &column1);
14.    printf("Enter the order of the matrix B \n");
15.    scanf("%d %d", &row2, &column2);
16.    printf("Enter the elements of matrix A \n");
17.    for (i = 0; i < row1; i++)
18.    {
19.        for (j = 0; j < column1; j++)
20.        {
21.            scanf("%d", &a[i][j]);
22.        }
23.    }
24.    printf("Enter the elements of matrix B \n");
25.    for (i = 0; i < row2; i++)
26.    {
27.        for (j = 0; j < column2; j++)
28.        {
29.            scanf("%d", &b[i][j]);
30.        }
31.    }
32.    printf("MATRIX A is \n");
33.    for (i = 0; i < row1; i++)
34.    {
35.        for (j = 0; j < column1; j++)
36.        {
37.            printf("%3d", a[i][j]);
38.        }
39.        printf("\n");
40.    }
41.    printf("MATRIX B is \n");
42.    for (i = 0; i < row2; i++)
43.    {
44.        for (j = 0; j < column2; j++)
45.        {
46.            printf("%3d", b[i][j]);
47.        }
48.        printf("\n");
}
```

```

49.     }
50.     /* Comparing two matrices for equality */
51.     if (row1 == row2 && column1 == column2)
52.     {
53.         printf("Matrices can be compared \n");
54.         for (i = 0; i < row1; i++)
55.         {
56.             for (j = 0; j < column2; j++)
57.             {
58.                 if (a[i][j] != b[i][j])
59.                 {
60.                     flag = 0;
61.                     break;
62.                 }
63.             }
64.         }
65.     }
66.     else
67.     {
68.         printf(" Cannot be compared\n");
69.         exit(1);
70.     }
71.     if (flag == 1)
72.         printf("Two matrices are equal \n");
73.     else
74.         printf("But, two matrices are not equal \n");
75. }
```

```

1. /*
2.  * C Program to check if a given matrix is an identity matrix
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int a[10][10];
9.     int i, j, row, column, flag = 1;
10.
11.    printf("Enter the order of the matrix A \n");
12.    scanf("%d %d", &row, &column);
13.    printf("Enter the elements of matrix A \n");
14.    for (i = 0; i < row; i++)
15.    {
16.        for (j = 0; j < column; j++)
17.        {
18.            scanf("%d", &a[i][j]);
```

```

19.     }
20. }
21. printf("MATRIX A is \n");
22. for (i = 0; i < row; i++)
23. {
24.     for (j = 0; j < column; j++)
25.     {
26.         printf("%3d", a[i][j]);
27.     }
28.     printf("\n");
29. }
30. /* Check for unit (or identity) matrix */
31. for (i = 0; i < row; i++)
32. {
33.     for (j = 0; j < column; j++)
34.     {
35.         if (a[i][j] != 1 && a[j][i] != 0)
36.         {
37.             flag = 0;
38.             break;
39.         }
40.     }
41. }
42. if (flag == 1 )
43.     printf("It is identity matrix \n");
44. else
45.     printf("It is not a identity matrix \n");
46. }
```

```

1. /*
2. * C program to determine if a given matrix is a sparse matrix.
3. * Sparse matrix has more zero elements than nonzero elements.
4. */
5. #include <stdio.h>
6.
7. void main ()
8. {
9.     static int array[10][10];
10.    int i, j, m, n;
11.    int counter = 0;
12.
13.    printf("Enter the order of the matix \n");
14.    scanf("%d %d", &m, &n);
15.    printf("Enter the co-efficients of the matix \n");
16.    for (i = 0; i < m; ++i)
17.    {
```

```

18.         for (j = 0; j < n; ++j)
19.         {
20.             scanf("%d", &array[i][j]);
21.             if (array[i][j] == 0)
22.             {
23.                 ++counter;
24.             }
25.         }
26.     }
27.     if (counter > ((m * n) / 2))
28.     {
29.         printf("The given matrix is sparse matrix \n");
30.     }
31.     else
32.         printf("The given matrix is not a sparse matrix \n");
33.     printf("There are %d number of zeros", counter);
34. }
```

### 3. C Examples on Matrix Rows and Columns

```

1. /*
2.  * C program to accept a matrix of given order and interchange
3.  * any two rows and columns in the original matrix
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     static int array1[10][10], array2[10][10];
10.    int i, j, m, n, a, b, c, p, q, r;
11.
12.    printf("Enter the order of the matrix \n");
13.    scanf("%d %d", &m, &n);
14.    printf("Enter the co-efficients of the matrix \n");
15.    for (i = 0; i < m; ++i)
16.    {
17.        for (j = 0; j < n; ++j)
18.        {
19.            scanf("%d,", &array1[i][j]);
20.            array2[i][j] = array1[i][j];
21.        }
22.    }
23.    printf("Enter the numbers of two rows to be exchanged \n");
24.    scanf("%d %d", &a, &b);
25.    for (i = 0; i < m; ++i)
26.    {
```

```

27.     /* first row has index is 0 */
28.     c = array1[a - 1][i];
29.     array1[a - 1][i] = array1[b - 1][i];
30.     array1[b - 1][i] = c;
31. }
32. printf("Enter the numbers of two columns to be exchanged \n");
33. scanf("%d %d", &p, &q);
34. printf("The given matrix is \n");
35. for (i = 0; i < m; ++i)
36. {
37.     for (j = 0; j < n; ++j)
38.         printf(" %d", array2[i][j]);
39.     printf("\n");
40. }
41. for (i = 0; i < n; ++i)
42. {
43.     /* first column index is 0 */
44.     r = array2[i][p - 1];
45.     array2[i][p - 1] = array2[i][q - 1];
46.     array2[i][q - 1] = r;
47. }
48. printf("The matix after interchanging the two rows(in the original matrix) \n");
49. for (i = 0; i < m; ++i)
50. {
51.     for (j = 0; j < n; ++j)
52.     {
53.         printf(" %d", array1[i][j]);
54.     }
55.     printf("\n");
56. }
57. printf("The matix after interchanging the two columns(in the original matrix) \n");
58. for (i = 0; i < m; ++i)
59. {
60.     for (j = 0; j < n; ++j)
61.         printf(" %d", array2[i][j]);
62.     printf("\n");
63. }
64. }

```

```

1. /*
2. * C program to accept a matrices of order MxN and sort all rows of the
3. * matrix in ascending order and all columns in descending order
4. */
5. #include <stdio.h>
6.
7. void main()

```

```
8.  {
9.      static int array1[10][10], array2[10][10];
10.     int i, j, k, a, m, n;
11.
12.    printf("Enter the order of the matrix \n");
13.    scanf("%d %d", &m, &n);
14.    printf("Enter co-efficients of the matrix \n");
15.    for (i = 0; i < m; ++i)
16.    {
17.        for (j = 0; j < n; ++j)
18.        {
19.            scanf("%d", &array1[i][j]);
20.            array2[i][j] = array1[i][j];
21.        }
22.    }
23.    printf("The given matrix is \n");
24.    for (i = 0; i < m; ++i)
25.    {
26.        for (j = 0; j < n; ++j)
27.        {
28.            printf(" %d", array1[i][j]);
29.        }
30.        printf("\n");
31.    }
32.    printf("After arranging rows in ascending order\n");
33.    for (i = 0; i < m; ++i)
34.    {
35.        for (j = 0; j < n; ++j)
36.        {
37.            for (k =(j + 1); k < n; ++k)
38.            {
39.                if (array1[i][j] > array1[i][k])
40.                {
41.                    a = array1[i][j];
42.                    array1[i][j] = array1[i][k];
43.                    array1[i][k] = a;
44.                }
45.            }
46.        }
47.    }
48.    for (i = 0; i < m; ++i)
49.    {
50.        for (j = 0; j < n; ++j)
51.        {
52.            printf(" %d", array1[i][j]);
53.        }
54.        printf("\n");
```

```

55.     }
56.     printf("After arranging the columns in descending order \n");
57.     for (j = 0; j < n; ++j)
58.     {
59.         for (i = 0; i < m; ++i)
60.         {
61.             for (k = i + 1; k < m; ++k)
62.             {
63.                 if (array2[i][j] < array2[k][j])
64.                 {
65.                     a = array2[i][j];
66.                     array2[i][j] = array2[k][j];
67.                     array2[k][j] = a;
68.                 }
69.             }
70.         }
71.     }
72.     for (i = 0; i < m; ++i)
73.     {
74.         for (j = 0; j < n; ++j)
75.         {
76.             printf(" %d", array2[i][j]);
77.         }
78.         printf("\n");
79.     }
80. }
```

#### 4. C Examples on Matrix Diagonals

```

1. /*
2. * C program to read a matrix A (MxN) & find the following using
3. * functions a) Sum of the elements of each row
4. * b) Sum of the elements of each column
5. * c) Find the sum of all the elements of the matrix
6. * Output the computed results
7. */
8. #include <stdio.h>
9. int Addrow(int array1[10][10], int k, int c);
10. int Addcol(int array1[10][10], int k, int r);
11.
12. void main()
13. {
14.     int arr[10][10];
15.     int i, j, row, col, rowsum, colsum, sumall=0;
16.
17.     printf("Enter the order of the matrix \n");
```

```

18.     scanf("%d %d", &row, &col);
19.     printf("Enter the elements of the matrix \n");
20.     for (i = 0; i < row; i++)
21.     {
22.         for (j = 0; j < col; j++)
23.         {
24.             scanf("%d", &arr[i][j]);
25.         }
26.     }
27.     printf("Input matrix is \n");
28.     for (i = 0; i < row; i++)
29.     {
30.         for (j = 0; j < col; j++)
31.         {
32.             printf("%3d", arr[i][j]);
33.         }
34.         printf("\n");
35.     }
36. /* computing row sum */
37. for (i = 0; i < row; i++)
38. {
39.     rowsum = Addrow(arr, i, col);
40.     printf("Sum of row %d = %d\n", i + 1, rowsum);
41. }
42. /* computing col sum */
43. for (j = 0; j < col; j++)
44. {
45.     colsum = Addcol(arr, j, row);
46.     printf("Sum of column %d = %d\n", j + 1, colsum);
47. }
48. /* computation of all elements */
49. for (j = 0; j < row; j++)
50. {
51.     sumall = sumall + Addrow(arr, j, col);
52. }
53. printf("Sum of all elements of matrix = %d\n", sumall);
54. }
55. /* Function to add each row */
56. int Addrow(int array1[10][10], int k, int c)
57. {
58.     int rsum = 0, i;
59.     for (i = 0; i < c; i++)
60.     {
61.         rsum = rsum + array1[k][i];
62.     }
63.     return(rsum);
64. }
```

```

65. /* Function to add each column */
66. int Addcol(int array1[10][10], int k, int r)
67. {
68.     int csum = 0, j;
69.     for (j = 0; j < r; j++)
70.     {
71.         csum = csum + array1[j][k];
72.     }
73.     return(csum);
74. }
```

```

1. /*
2. * C program to find the frequency of odd numbers
3. * and even numbers in the input of a matrix
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     static int array[10][10];
10.    int i, j, m, n, even = 0, odd = 0;
11.
12.    printf("Enter the order of the matrix \n");
13.    scanf("%d %d", &m, &n);
14.    printf("Enter the coefficients of matrix \n");
15.    for (i = 0; i < m; ++i)
16.    {
17.        for (j = 0; j < n; ++j)
18.        {
19.            scanf("%d", &array[i][j]);
20.            if ((array[i][j] % 2) == 0)
21.            {
22.                ++even;
23.            }
24.            else
25.                ++odd;
26.        }
27.    }
28.    printf("The given matrix is \n");
29.    for (i = 0; i < m; ++i)
30.    {
31.        for (j = 0; j < n; ++j)
32.        {
33.            printf(" %d", array[i][j]);
34.        }
35.        printf("\n");
```

```

36.     }
37.     printf("\n The frequency of occurrence of odd number = %d \n", odd);
38.     printf("The frequency of occurrence of even number = %d\n", even);
39. }
```

```

1. /*
2.  * C program to accept a matrix of order M x N and store its elements
3.  * and interchange the main diagonal elements of the matrix
4.  * with that of the secondary diagonal elements
5. */
6. #include <stdio.h>
7.
8. void main ()
9. {
10.     static int array[10][10];
11.     int i, j, m, n, a;
12.
13.     printf("Enter the order of the matix \n");
14.     scanf("%d %d", &m, &n);
15.     if (m == n)
16.     {
17.         printf("Enter the co-efficients of the matrix\n");
18.         for (i = 0; i < m; ++i)
19.         {
20.             for (j = 0; j < n; ++j)
21.             {
22.                 scanf("%dx%d", &array[i][j]);
23.             }
24.         }
25.         printf("The given matrix is \n");
26.         for (i = 0; i < m; ++i)
27.         {
28.             for (j = 0; j < n; ++j)
29.             {
30.                 printf(" %d", array[i][j]);
31.             }
32.             printf("\n");
33.         }
34.         for (i = 0; i < m; ++i)
35.         {
36.             a = array[i][i];
37.             array[i][i] = array[i][m - i - 1];
38.             array[i][m - i - 1] = a;
39.         }
40.         printf("The matrix after changing the \n");
41.         printf("main diagonal & secondary diagonal\n");
```

```

42.         for (i = 0; i < m; ++i)
43.         {
44.             for (j = 0; j < n; ++j)
45.             {
46.                 printf(" %d", array[i][j]);
47.             }
48.             printf("\n");
49.         }
50.     }
51. else
52.     printf("The given order is not square matrix\n");
53. }
```

```

1. /*
2. * C program to accept a matrix of order M x N and find the sum
3. * of each row and each column of a matrix
4. */
5. #include <stdio.h>
6.
7. void main ()
8. {
9.     static int array[10][10];
10.    int i, j, m, n, sum = 0;
11.
12.    printf("Enter the order of the matrix\n");
13.    scanf("%d %d", &m, &n);
14.    printf("Enter the co-efficients of the matrix\n");
15.    for (i = 0; i < m; ++i)
16.    {
17.        for (j = 0; j < n; ++j)
18.        {
19.            scanf("%d", &array[i][j]);
20.        }
21.    }
22.    for (i = 0; i < m; ++i)
23.    {
24.        for (j = 0; j < n; ++j)
25.        {
26.            sum = sum + array[i][j] ;
27.        }
28.        printf("Sum of the %d row is = %d\n", i, sum);
29.        sum = 0;
30.    }
31.    sum = 0;
32.    for (j = 0; j < n; ++j)
33.    {
```

```

34.         for (i = 0; i < m; ++i)
35.     {
36.         sum = sum + array[i][j];
37.     }
38.     printf("Sum of the %d column is = %d\n", j, sum);
39.     sum = 0;
40. }
41. }
```

```

1. /*
2.  * C program to find accept a matrix of order M x N and find
3.  * the sum of the main diagonal and off diagonal elements
4. */
5. #include <stdio.h>
6.
7. void main ()
8. {
9.     static int array[10][10];
10.    int i, j, m, n, a = 0, sum = 0;
11.
12.    printf("Enter the order of the matrix \n");
13.    scanf("%d %d", &m, &n);
14.    if (m == n )
15.    {
16.        printf("Enter the co-efficients of the matrix\n");
17.        for (i = 0; i < m; ++i)
18.        {
19.            for (j = 0; j < n; ++j)
20.            {
21.                scanf("%d", &array[i][j]);
22.            }
23.        }
24.        printf("The given matrix is \n");
25.        for (i = 0; i < m; ++i)
26.        {
27.            for (j = 0; j < n; ++j)
28.            {
29.                printf(" %d", array[i][j]);
30.            }
31.            printf("\n");
32.        }
33.        for (i = 0; i < m; ++i)
34.        {
35.            sum = sum + array[i][i];
36.            a = a + array[i][m - i - 1];
37.        }
}
```

```

38.         printf("\nThe sum of the main diagonal elements is = %d\n", sum);
39.         printf("The sum of the off diagonal elemets is    = %d\n", a);
40.     }
41. else
42.     printf("The given order is not square matrix\n");
43. }

44. /*
45. * C program to find accept a matrix of order M x N and find
46. * the sum of the main diagonal and off diagonal elements
47. */
48. #include <stdio.h>
49.
50. void main ()
51. {
52.     static int array[10][10];
53.     int i, j, m, n, a = 0, sum = 0;
54.
55.     printf("Enetr the order of the matix \n");
56.     scanf("%d %d", &m, &n);
57.     if (m == n )
58.     {
59.         printf("Enter the co-efficients of the matrix\n");
60.         for (i = 0; i < m; ++i)
61.         {
62.             for (j = 0; j < n; ++j)
63.             {
64.                 scanf("%d", &array[i][j]);
65.             }
66.         }
67.         printf("The given matrix is \n");
68.         for (i = 0; i < m; ++i)
69.         {
70.             for (j = 0; j < n; ++j)
71.             {
72.                 printf(" %d", array[i][j]);
73.             }
74.             printf("\n");
75.         }
76.         for (i = 0; i < m; ++i)
77.         {
78.             sum = sum + array[i][i];
79.             a = a + array[i][m - i - 1];
80.         }
81.         printf("\nThe sum of the main diagonal elements is = %d\n", sum);
82.         printf("The sum of the off diagonal elemets is    = %d\n", a);
83.     }
84. else

```

```

85.         printf("The given order is not square matrix\n");
86. }

```

```

1. /*
2. * C program to find the trace and normal of a matrix
3. *
4. * Trace is defined as the sum of main diagonal elements and
5. * Normal is defined as square root of the sum of all the elements
6. */
7. #include <stdio.h>
8. #include <math.h>
9.
10. void main ()
11. {
12.     static int array[10][10];
13.     int i, j, m, n, sum = 0, sum1 = 0, a = 0, normal;
14.
15.     printf("Enter the order of the matrix\n");
16.     scanf("%d %d", &m, &n);
17.     printf("Enter the n coefficients of the matrix \n");
18.     for (i = 0; i < m; ++i)
19.     {
20.         for (j = 0; j < n; ++j)
21.         {
22.             scanf("%d", &array[i][j]);
23.             a = array[i][j] * array[i][j];
24.             sum1 = sum1 + a;
25.         }
26.     }
27.     normal = sqrt(sum1);
28.     printf("The normal of the given matrix is = %d\n", normal);
29.     for (i = 0; i < m; ++i)
30.     {
31.         sum = sum + array[i][i];
32.     }
33.     printf("Trace of the matrix is = %d\n", sum);
34. }

```

```

1. /*
2. * C Program to Display Upper Triangular Matrix
3. */
4. #include <stdio.h>
5.
6. void main()
7. {

```

```
8.     int i, j, r, c, array[10][10];
9.
10.    printf("Enter the r and c value:");
11.    scanf("%d%d", &r, &c);
12.    for (i = 1; i <= r; i++)
13.    {
14.        for (j = 1; j <= c; j++)
15.        {
16.            printf("array[%d][%d] = ", i, j);
17.            scanf("%d", &array[i][j]);
18.        }
19.    }
20.    printf("matrix is");
21.    for (i = 1; i <= r; i++)
22.    {
23.        for (j = 1; j <= c; j++)
24.        {
25.            printf("%d", array[i][j]);
26.        }
27.        printf("\n");
28.    }
29.    for (i = 1; i <= r; i++)
30.    {
31.        printf("\n");
32.        for (j = 1; j <= c; j++)
33.        {
34.            if (i >= j)
35.            {
36.                printf("%d", array[i][j]);
37.            }
38.            else
39.            {
40.                printf("\t");
41.            }
42.        }
43.    }
44.    printf("\n\n");
45.    for (i = 1; i <= r; i++)
46.    {
47.        printf("\n");
48.        for (j = 1; j <= c; j++)
49.        {
50.            if (j >= i)
51.            {
52.                printf("%d", array[i][j]);
53.            }
54.            else
```

```

55.     {
56.         //printf("\t");
57.     }
58.     // printf("\n");
59. }
60. }
```

```

1. /*
2. * C Program to Display Lower Triangular Matrix
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array[3][3], i, j, flag = 0 ;
9.     printf("\n\t Enter the value of Matrix : ");
10.    for (i = 0; i < 3; i++)
11.    {
12.        for (j = 0; j < 3; j++)
13.        {
14.            scanf("%d", &array[i][j]);
15.        }
16.    }
17.    for (i = 0; i < 3; i++)
18.    {
19.        for (j = 0; j < 3; j++)
20.        {
21.            if (array[i] < array[j] && array[i][j] == 0)
22.            {
23.                flag = flag + 1;
24.            }
25.        }
26.    }
27.    if (flag == 3)
28.        printf("\n\n Matrix is a Lower triangular matrix");
29.    else
30.        printf("\n\n Matrix is not a lower triangular matrix");
31. }
```

## (4.) C Programming Examples on Strings

### 1. C Examples on Palindrome

```

1. /*
2. * C program to read a string and check if it's a palindrome, without
```

```

3.   * using library functions. Display the result.
4.   */
5. #include <stdio.h>
6. #include <string.h>
7.
8. void main()
9. {
10.     char string[25], reverse_string[25] = {'\0'};
11.     int i, length = 0, flag = 0;
12.
13.     fflush(stdin);
14.     printf("Enter a string \n");
15.     gets(string);
16.     /* keep going through each character of the string till its end */
17.     for (i = 0; string[i] != '\0'; i++)
18.     {
19.         length++;
20.     }
21.     for (i = length - 1; i >= 0; i--)
22.     {
23.         reverse_string[length - i - 1] = string[i];
24.     }
25.     /*
26.      * Compare the input string and its reverse. If both are equal
27.      * then the input string is palindrome.
28.      */
29.     for (i = 0; i < length; i++)
30.     {
31.         if (reverse_string[i] == string[i])
32.             flag = 1;
33.         else
34.             flag = 0;
35.     }
36.     if (flag == 1)
37.         printf("%s is a palindrome \n", string);
38.     else
39.         printf("%s is not a palindrome \n", string);
40. }
```

```

1. /*
2.  * C program to find the length of a string without using the
3.  * built-in function also check whether it is a palindrome
4.  */
5. #include <stdio.h>
6. #include <string.h>
7.
```

```

8. void main()
9. {
10.     char string[25], reverse_string[25] = {'\0'};
11.     int i, length = 0, flag = 0;
12.
13.     printf("Enter a string \n");
14.     gets(string);
15.     /* keep going through each character of the string till its end */
16.     for (i = 0; string[i] != '\0'; i++)
17.     {
18.         length++;
19.     }
20.     printf("The length of the string '%s' = %d\n", string, length);
21.     for (i = length - 1; i >= 0 ; i--)
22.     {
23.         reverse_string[length - i - 1] = string[i];
24.     }
25.     /* Check if the string is a Palindrome */
26.
27.     for (flag = 1, i = 0; i < length ; i++)
28.     {
29.         if (reverse_string[i] != string[i])
30.             flag = 0;
31.     }
32.     if (flag == 1)
33.         printf ("%s is a palindrome \n", string);
34.     else
35.         printf("%s is not a palindrome \n", string);
36. }
```

```

1. /*
2.  * C Program to Check whether a given String is Palindrome or not
3.  * using Recursion
4.  */
5. #include <stdio.h>
6. #include <string.h>
7.
8. void check(char [], int);
9.
10. int main()
11. {
12.     char word[15];
13.
14.     printf("Enter a word to check if it is a palindrome\n");
15.     scanf("%s", word);
16.     check(word, 0);
```

```

17.
18.     return 0;
19. }
20.
21. void check(char word[], int index)
22. {
23.     int len = strlen(word) - (index + 1);
24.     if (word[index] == word[len])
25.     {
26.         if (index + 1 == len || index == len)
27.         {
28.             printf("The entered word is a palindrome\n");
29.             return;
30.         }
31.         check(word, index + 1);
32.     }
33.     else
34.     {
35.         printf("The entered word is not a palindrome\n");
36.     }
37. }

```

```

1. /*
2.  * C Program To Print Smallest and Biggest possible Word
3.  * which is Palindrome in a given String
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #include <stdlib.h>
8.
9. int main()
10. {
11.     int i = 0, l = 0, j, k, space = 0, count = 0, init = 0, min = 0, max = 0, len = 0, flag;
12.     char a[100], b[30][20], c[30], d[30], minP[30], maxP[30];
13.
14.     printf("Read a string:\n");
15.     fflush(stdin);
16.     scanf("%[^\\n]s", a);
17.     for (i = 0; a[i] != '\0'; i++)
18.     {
19.         if (a[i] == ' ')
20.             space++;
21.     }
22.     i = 0;
23.     for (j = 0; j < (space+1); i++, j++)
24.     {

```

```

25.         k = 0;
26.         while (a[i] != '\0')
27.         {
28.             if (a[i] == ' ')
29.             {
30.                 break;
31.             }
32.             else
33.             {
34.                 b[j][k++] = a[i];
35.                 i++;
36.             }
37.         }
38.         b[j][k] = '\0';
39.     }
40.     for (j = 0;j < space + 1;j++)
41.     printf("%s ", b[j]);
42.     printf("\n");
43.     for (i = 0;i < space + 1;i++)
44.     {
45.         strcpy(c, b[i]);
46.         count = strlen(b[i]);
47.         k = 0;
48.         for (l = count - 1;l >= 0;l--)
49.             d[k++] = b[i][l];
50.         d[k] = '\0';
51.         if (strcmp(d, c) == 0)           {
52.             flag = 1;
53.             if (init < 1)
54.             {
55.                 strcpy(minP, d);
56.                 strcpy(maxP, d);
57.                 min = strlen(minP);
58.                 max = strlen(maxP);
59.                 init++;
60.             }
61.             printf("String %s is a Palindrome\n", d);
62.             len = strlen(d);
63.             if (len >= max)
64.                 strcpy(maxP, d);
65.             else if (len <= min)
66.                 strcpy(minP, d);
67.             else
68.                 printf("");
69.         }
70.     }
71.     if (flag == 1)

```

```

72.     {
73.         printf("The minimum palindrome is %s\n", minP);
74.         printf("The maximum palindrome is %s\n", maxP);
75.     }
76. else
77.     printf("given string has no pallindrome\n");
78. }

79. /*
80. * C Program to Find the Largest & Smallest Word in a String
81. */
82. #include <stdio.h>
83. #include <string.h>
84. #include <ctype.h>
85.
86. int main()
87. {
88.     char string[100], word[20], max[20], min[20], c;
89.     int i = 0, j = 0, flag = 0;
90.
91.     printf("Enter string: ");
92.     i = 0;
93.     do
94.     {
95.         fflush(stdin);
96.         c = getchar();
97.         string[i++] = c;
98.
99.     } while (c != '\n');
100.    string[i - 1] = '\0';
101.    for (i = 0; i < strlen(string); i++)
102.    {
103.        while (i < strlen(string) && !isspace(string[i]) && isalnum(string[i]))
104.        {
105.            word[j++] = string[i++];
106.        }
107.        if (j != 0)
108.        {
109.            word[j] = '\0';
110.            if (!flag)
111.            {
112.                flag = !flag;
113.                strcpy(max, word);
114.                strcpy(min, word);
115.            }
116.            if (strlen(word) > strlen(max))
117.            {
118.                strcpy(max, word);

```

```

119.         }
120.         if (strlen(word) < strlen(min))
121.         {
122.             strcpy(min, word);
123.         }
124.         j = 0;
125.     }
126. }
127. printf("The largest word is '%s' and smallest word is '%s' in '%s'.\n", max, min,
   string);
128.
129.     return 0;
130. }
```

## 2. C Examples on String Operations

```

1. /*
2. * C program to read two strings and concatenate them, without using
3. * library functions. Display the concatenated string.
4. */
5. #include <stdio.h>
6. #include <string.h>
7.
8. void main()
9. {
10.     char string1[20], string2[20];
11.     int i, j, pos;
12.
13.     /* Initialize the string to NULL values */
14.     memset(string1, 0, 20);
15.     memset(string2, 0, 20);
16.
17.     printf("Enter the first string : ");
18.     scanf("%s", string1);
19.     printf("Enter the second string: ");
20.     scanf("%s", string2);
21.     printf("First string = %s\n", string1);
22.     printf("Second string = %s\n", string2);
23.
24.     /* Concatenate the second string to the end of the first string */
25.     for (i = 0; string1[i] != '\0'; i++)
26.     {
27.         /* null statement: simply traversing the string1 */
28.         ;
29.     }
30.     pos = i;
```

```

31.     for (j = 0; string2[j] != '\0'; i++)
32.     {
33.         string1[i] = string2[j++];
34.     }
35.     /* set the last character of string1 to NULL */
36.     string1[i] = '\0';
37.     printf("Concatenated string = %s\n", string1);
38. }
```

```

1. /*
2.  * C Program to Concatenate the given two Strings Lexically
3.  */
4. #include <string.h>
5. #include <stdio.h>
6.
7. void sort(char *p);
8.
9. void main()
10. {
11.     char string1[100], string2[100];
12.     int i, len, j;
13.
14.     printf("\nEnter a string : ");
15.     scanf("%[^\\n]s", string1);
16.     printf("\nEnter another string to concat : ");
17.     scanf(" %[^\n]s", string2);
18.     len = strlen(string1);
19.     string1[len] = ' ';
20.     for(i = 0, j = len + 1; i < strlen(string2); i++, j++)
21.         string1[j] = string2[i];
22.     string1[j] = '\0';
23.     sort(string1);
24. }
25.
26. /* Sorting to make concatenation Lexical */
27. void sort(char *p)
28. {
29.     char temp[100];
30.     char a[100][100];
31.     int t1, i, j = 0, k = 0, l = strlen(p), x = 0, y = 0, z = 0, count, l1, l2;
32.
33.     for (i = 0; i < l; i++)
34.     {
35.         if (p[i] != ' ')
36.         {
37.             a[k][j++] = p[i];

```

```

38.         }
39.     else
40.     {
41.         a[k][j] = '\0';
42.         k++;
43.         j = 0;
44.     }
45. }
46.
47. t1 = k;
48. k = 0;
49.
50. for (i = 0; i < t1; i++)
51. {
52.     for (j = i + 1; j <= t1; j++)
53.     {
54.         l1 = strlen(a[i]);
55.         l2 = strlen(a[j]);
56.         if (l1 > l2)
57.             count = l1;
58.         else
59.             count = l2;
60.         x = 0, y = 0;
61.         while ((x < count) || (y < count))
62.         {
63.             if (a[i][x] == a[j][y])
64.             {
65.                 x++;
66.                 y++;
67.                 continue;
68.             }
69.             else
70.                 if (a[i][x] < a[j][y]) break;
71.             else
72.                 if (a[i][x] > a[j][y])
73.                 {
74.                     for (z = 0; z < l2; z++)
75.                     {
76.                         temp[z] = a[j][z];
77.                         a[j][z] = '\0';
78.                     }
79.                     temp[z] = '\0';
80.
81.                     for (z = 0; z < l1; z++)
82.                     {
83.                         a[j][z] = a[i][z];
84.                         a[i][z] = '\0';

```

```

85.         }
86.         a[j][z] = '\0';
87.
88.         for (z = 0; z < strlen(temp); z++)
89.         {
90.             a[i][z] = temp[z];
91.         }
92.         break;
93.     }
94. }
95. }
96. }
97.
98. for (i = 0; i < l; i++)
99. p[i] = '\0';
100. k = 0;
101. j = 0;
102. for (i = 0; i < l; i++)
103. {
104.     if (a[k][j] != '\0')
105.     {
106.         p[i] = a[k][j++];
107.     }
108.     else
109.     {
110.         k++;
111.         j = 0;
112.         p[i] = ' ';
113.     }
114. }
115. puts(p);
116. }
```

```

1. /*
2.  * C program to accept a string and a substring and
3.  * check if the substring is present in the given string
4. */
5. #include<stdio.h>
6.
7. void main()
8. {
9.     char str[80], search[10];
10.    int count1 = 0, count2 = 0, i, j, flag;
11.
12.    printf("Enter a string:");
13.    gets(str);
```

```

14.     printf("Enter search substring:");
15.     gets(search);
16.     while (str[count1] != '\0')
17.         count1++;
18.     while (search[count2] != '\0')
19.         count2++;
20.     for (i = 0; i <= count1 - count2; i++)
21.     {
22.         for (j = i; j < i + count2; j++)
23.         {
24.             flag = 1;
25.             if (str[j] != search[j - i])
26.             {
27.                 flag = 0;
28.                 break;
29.             }
30.         }
31.         if (flag == 1)
32.             break;
33.     }
34.     if (flag == 1)
35.         printf("SEARCH SUCCESSFUL!");
36.     else
37.         printf("SEARCH UNSUCCESSFUL!");
38. }
```

```

1. /*
2.  * C Program to accepts two strings and compare them. Display
3.  * the result whether both are equal, or first string is greater
4.  * than the second or the first string is Less than the second string
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     int count1 = 0, count2 = 0, flag = 0, i;
11.     char string1[10], string2[10];
12.
13.     printf("Enter a string:");
14.     gets(string1);
15.     printf("Enter another string:");
16.     gets(string2);
17.     /* Count the number of characters in string1 */
18.     while (string1[count1] != '\0')
19.         count1++;
20.     /* Count the number of characters in string2 */
```

```

21.     while (string2[count2] != '\0')
22.         count2++;
23.     i = 0;
24.
25.     while ((i < count1) && (i < count2))
26.     {
27.         if (string1[i] == string2[i])
28.         {
29.             i++;
30.             continue;
31.         }
32.         if (string1[i] < string2[i])
33.         {
34.             flag = -1;
35.             break;
36.         }
37.         if (string1[i] > string2[i])
38.         {
39.             flag = 1;
40.             break;
41.         }
42.     }
43.     if (flag == 0)
44.         printf("Both strings are equal \n");
45.     if (flag == 1)
46.         printf("String1 is greater than string2 \n", string1, string2);
47.     if (flag == -1)
48.         printf("String1 is less than string2 \n", string1, string2);
49. }
```

```

1. /*
2.  * C program to find the Length of a string without using the
3.  * built-in function
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     char string[50];
10.    int i, length = 0;
11.
12.    printf("Enter a string \n");
13.    gets(string);
14.    /* keep going through each character of the string till its end */
15.    for (i = 0; string[i] != '\0'; i++)
16.    {
```

```

17.     length++;
18. }
19. printf("The length of a string is the number of characters in it \n");
20. printf("So, the length of %s = %d\n", string, length);
21. }
```

### 3. C Examples on Replace, Remove and Reverse Functions

```

1. /*
2. * C program to read an English sentence and replace
3. * Lowercase characters by uppercase and vice-versa.
4. * Output the given sentence as well as the converted
5. * sentence on two different lines.
6. */
7. #include <stdio.h>
8. #include <ctype.h>
9.
10. void main()
11. {
12.     char sentence[100];
13.     int count, ch, i;
14.
15.     printf("Enter a sentence \n");
16.     for (i = 0; (sentence[i] = getchar()) != '\n'; i++)
17.     {
18.         ;
19.     }
20.     sentence[i] = '\0';
21.     /* shows the number of chars accepted in a sentence */
22.     count = i;
23.     printf("The given sentence is : %s", sentence);
24.     printf("\n Case changed sentence is: ");
25.     for (i = 0; i < count; i++)
26.     {
27.         ch = islower(sentence[i])? toupper(sentence[i]) :
28. tolower(sentence[i]);
29.         putchar(ch);
30.     }
31. }
```

```

1. /*
2. * C Program to Remove given Word from a String
3. */
4. #include <stdio.h>
5. #include <string.h>
```

```
6.
7. void main()
8. {
9.     int i, j = 0, k = 0, count = 0;
10.    char str[100], key[20];
11.    char str1[10][20];
12.
13.    printf("enter string:");
14.    scanf("%[^\\n]s",str);
15.
16. /* Converts the string into 2D array */
17.    for (i = 0; str[i]!='\\0'; i++)
18.    {
19.        if (str[i]==' ')
20.        {
21.            str1[k][j] = '\\0';
22.            k++;
23.            j = 0;
24.        }
25.        else
26.        {
27.            str1[k][j] = str[i];
28.            j++;
29.        }
30.    }
31.    str1[k][j] = '\\0';
32.    printf("enter key:");
33.    scanf("%s", key);
34.
35. /* Compares the string with given word */
36.    for (i = 0;i < k + 1; i++)
37.    {
38.        if (strcmp(str1[i], key) == 0)
39.        {
40.            for (j = i; j < k + 1; j++)
41.                strcpy(str1[j], str1[j + 1]);
42.            k--;
43.        }
44.    }
45.    for (i = 0;i < k + 1; i++)
46.    {
47.        printf("%s ", str1[i]);
48.    }
49. }
50. }{
```

```

1. /*
2. * C Program to Remove all Characters in Second String which are
3. * present in First String
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #include <ctype.h>
8. #include <stdlib.h>
9. #define CHAR_SIZE 26
10.
11. void alphacheck(char *, int []);
12. void create(char [], char [], int[]);
13.
14. int main()
15. {
16.     char str1[50], str2[50];
17.     int a1[CHAR_SIZE] = {0};
18.     char str2_rem[50];
19.
20.     printf("Enter string1: ");
21.     scanf("%s", str1);
22.     printf("Enter string2: ");
23.     scanf("%s", str2);
24.     alphacheck(str1, a1);
25.     create(str2_rem, str2, a1);
26.     printf("On removing characters from second string we get: %s\n", str2_rem);
27.
28.     return 0;
29. }
30.
31. void alphacheck(char *str, int a[])
32. {
33.     int i, index;
34.
35.     for (i = 0; i < strlen(str); i++)
36.     {
37.         str[i] = tolower(str[i]);
38.         index = str[i] - 'a';
39.         if (!a[index])
40.         {
41.             a[index] = 1;
42.         }
43.     }
44.     printf("\n");
45. }
46.
47. void create(char str_rem[], char str[], int list[])

```

```

48. {
49.     int i, j = 0, index;
50.
51.     for (i = 0; i < strlen(str); i++)
52.     {
53.         index = str[i] - 'a';
54.         if (!list[index])
55.         {
56.             str_rem[j++] = str[i];
57.         }
58.     }
59.     str_rem[j] = '\0';
60. }
```

```

1. /*
2.  * C Program to Replace all the Characters by Lowercase
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void main(int argc, char* argv[])
8. {
9.     FILE *fp1;
10.    int ch;
11.
12.    if ((fp1 = fopen(argv[1], "r+")) == NULL)
13.    {
14.        printf("\nfile cant be opened");
15.        exit(0);
16.    }
17.    ch = fgetc(fp1);
18.    while (ch != EOF)
19.    {
20.        if (ch >= 65 && ch <= 90)
21.        {
22.            fseek(fp1, -1L, 1);
23.            fputc(ch + 32, fp1);
24.        }
25.        ch = fgetc(fp1);
26.    }
27. }
```

```

1. /*
2.  * C Program to Reverse the String using Recursion
3. */
```

```

4. #include <stdio.h>
5. #include <string.h>
6.
7. void reverse(char [], int, int);
8. int main()
9. {
10.     char str1[20];
11.     int size;
12.
13.     printf("Enter a string to reverse: ");
14.     scanf("%s", str1);
15.     size = strlen(str1);
16.     reverse(str1, 0, size - 1);
17.     printf("The string after reversing is: %s\n", str1);
18.     return 0;
19. }
20.
21. void reverse(char str1[], int index, int size)
22. {
23.     char temp;
24.     temp = str1[index];
25.     str1[index] = str1[size - index];
26.     str1[size - index] = temp;
27.     if (index == size / 2)
28.     {
29.         return;
30.     }
31.     reverse(str1, index + 1, size);
32. }
```

```

1. /*
2.  * C Program to Reverse every Word of given String
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void main()
8. {
9.     int i, j = 0, k = 0, x, len;
10.    char str[100], str1[10][20], temp;
11.
12.    printf("enter the string :");
13.    scanf("%[^\\n]s", str);
14.
15. /* reads into 2d character array */
16.    for (i = 0; str[i] != '\\0'; i++)
```

```

17.     {
18.         if (str[i] == ' ')
19.         {
20.             str1[k][j]='\0';
21.             k++;
22.             j=0;
23.         }
24.     else
25.     {
26.         str1[k][j]=str[i];
27.         j++;
28.     }
29. }
30. str1[k][j] = '\0';
31.
32. /* reverses each word of a given string */
33. for (i = 0;i <= k;i++)
34. {
35.     len = strlen(str1[i]);
36.     for (j = 0, x = len - 1;j < x;j++,x--)
37.     {
38.         temp = str1[i][j];
39.         str1[i][j] = str1[i][x];
40.         str1[i][x] = temp;
41.     }
42. }
43. for (i = 0;i <= k;i++)
44. {
45.     printf("%s ", str1[i]);
46. }
47. }
```

```

1. /*
2. * C program to read a sentence and count the total number of vowels
3. * and consonants in the sentence.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     char sentence[80];
10.    int i, vowels = 0, consonants = 0, special = 0;
11.
12.    printf("Enter a sentence \n");
13.    gets(sentence);
14.    for (i = 0; sentence[i] != '\0'; i++)
```

```

15.    {
16.        if ((sentence[i] == 'a' || sentence[i] == 'e' || sentence[i] ==
17.            'i' || sentence[i] == 'o' || sentence[i] == 'u') ||
18.                (sentence[i] == 'A' || sentence[i] == 'E' || sentence[i] ==
19.                'I' || sentence[i] == 'O' || sentence[i] == 'U'))
20.        {
21.            vowels = vowels + 1;
22.        }
23.        else
24.        {
25.            consonants = consonants + 1;
26.        }
27.        if (sentence[i] == 't' || sentence[i] == '\0' || sentence[i] == ' ')
28.        {
29.            special = special + 1;
30.        }
31.    }
32.    consonants = consonants - special;
33.    printf("No. of vowels in %s = %d\n", sentence, vowels);
34.    printf("No. of consonants in %s = %d\n", sentence, consonants);
35. }
```

```

1. /*
2. * C Program to Delete All Repeated Words in String
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void main()
8. {
9.     char a[100], b[20][20];
10.    int i, j = 0, k = 0, n, m;
11.
12.    printf("enter the string\n");
13.    scanf("%[^\\n]s", a);
14.    for (i = 0; a[i] != '\0'; i++)
15.    {
16.        if (a[i] == ' ')
17.        {
18.            b[k][j] = '\0';
19.            k++;
20.            j = 0;
21.        }
22.        else
23.        {
24.            b[k][j] = a[i];
```

```

25.         j++;
26.     }
27. }
28. b[k][j] = '\0';
29. for (i = 0;i <= k;i++)
30. {
31.     for (j = i + 1;j <= k;j++)
32.     {
33.         if (strcmp(b[i], b[j]) == 0)
34.         {
35.             for (m = j;m <= k;m++)
36.                 strcpy(b[m], b[m + 1]);
37.             k--;
38.         }
39.     }
40. }
41. for (n = 0;n <= k;n++)
42. {
43.     printf("%s\n", b[n]);
44. }
45. }
```

```

1. /*
2. * C Program to Reverse the String using Both Recursion and Iteration
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. /* Function Prototype */
8. void disp_str1_rec(char *);
9.
10. void main()
11. {
12.     char str1[100], *ptr;
13.     int len1 = 0, i;
14.     char ch;
15.     printf("Enter the string:\n");
16.     scanf("%[^\\n]s", str1);
17.     ptr = str1;
18.     len1 = strlen(str1);
19.     printf("Using iteration:\n");
20.     for (i = len1 - 1; i >= 0;i--)           /* Iterative Loop */
21.     {
22.
23.         ch = str1[i];
24.         printf("%c", ch);
```

```

25.     }
26.     printf("Using recursion:\n");
27.     disp_str1_rec(ptr);
28. }
29.
30. /* Code to reverse the string using Recursion */
31. void disp_str1_rec(char *stng)
32. {
33.     char ch;
34.     if (*stng != '\0')
35.     {
36.         ch = *stng;
37.         stng++;
38.         disp_str1_rec(stng);
39.         printf("%c", ch);
40.     }
41.     else
42.     return;
43. }
```

```

1. /*
2.  * C Program to Count Number of Words in a given Text Or Sentence
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void main()
8. {
9.     char s[200];
10.    int count = 0, i;
11.
12.    printf("enter the string\n");
13.    scanf("%[^\\n]s", s);
14.    for (i = 0;s[i] != '\0';i++)
15.    {
16.        if (s[i] == ' ')
17.            count++;
18.    }
19.    printf("number of words in given string are: %d\n", count + 1);
20. }
```

## 5. C Programming Examples on Bitwise Operations

### 1. C Examples on Mathematical Applications using Bitwise Operations

```

1. /*
2. * C Program to Check whether the given Integer has an Alternate
3. * Pattern
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int num, x, y, count = 0;
10.
11.    printf("enter the number:");
12.    scanf("%d", &num);
13.    x = num << 1;
14.    y = x ^ num;
15.    y = y + 1;
16.
17.    /* Checks if the number is in powers of 2 */
18.    while ((y / 2) != 0)
19.    {
20.        if (y % 2 != 0)
21.        {
22.            count++;
23.            break;
24.        }
25.        else
26.        {
27.            y = y / 2;
28.        }
29.    }
30.    if (count)
31.    {
32.        printf("false");
33.    }
34.    else
35.    {
36.        printf("true");
37.    }
38. }
```

```

1. /*
2. * C Program to round floor of integer to next lower power of 2
3. */
4. #include <stdio.h>
5. #define NUM_BITS_INT 32
6. int count = 0;
7.
```

```

8. void main()
9. {
10.     int temp, n, bit, i = 0;
11.
12.     printf("Enter a number : ");
13.     scanf("%d", &n);
14.     temp = n;
15.     while (i < NUM_BITS_INT)
16.     {
17.         bit = temp & 0x80000000;
18.         if (bit == -0x80000000)
19.         {
20.             bit = 1;
21.         }
22.         printf("%d", bit);
23.         temp = temp << 1;
24.         i++;
25.     }
26. }
```

```

1. /*
2.  * C Program that uses Function to return MSB position of unsigned Integer
3.  */
4. #include <stdio.h>
5. #define NUM_BITS_INT 32
6. int int_msb_position(int n);
7.
8. void main()
9. {
10.     int n, pos;
11.
12.     printf("Enter a number : ");
13.     scanf("%d", &n);
14.     pos = int_msb_position(n);
15.     printf("\nPosition of MSB bit = %d\n", NUM_BITS_INT - (pos + 1));
16. }
17.
18. /* Function to find the MSB bit position */
19. int int_msb_position(int n)
20. {
21.     int i = 0, bit;
22.     while (i < NUM_BITS_INT)
23.     {
24.         bit = n & 0x80000000;
25.         if (bit == -0x80000000)
26.         {
```

```

27.         bit = 1;
28.     }
29.     if (bit == 1)
30.         break;
31.     n = n << 1;
32.     i++;
33. }
34. return i;
35. }
```

```

1. /*
2. * C Program to use Bitwise Operations to Round(floor of) an Integer
3. * to next Lower Multiple of 2
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int x = 1, i, n;
10.
11.    printf("enter the number :");
12.    scanf("%d", &n);
13.    /* for positive values */
14.    if (n > 0)
15.    {
16.        for (; x <= n >> 1;)
17.        {
18.            x = x << 1;
19.        }
20.        n = x;
21.    }
22.    /* for negative values */
23.    else
24.    {
25.        n = ~n;
26.        n = n + 1;
27.        for (; x <= n >> 1;)
28.        {
29.            x = x << 1;
30.        }
31.        x = x << 1;
32.        x = ~x;
33.        x = x + 1;
34.        n = x;
35.    }
36.    printf("%d", n);
```

37. }

```

1. /*
2.  * C Program that uses Function to return MSB position of unsigned Integer
3.  */
4. #include <stdio.h>
5. #define NUM_BITS_INT 32
6. int int_msb_position(int n);
7.
8. void main()
9. {
10.     int n, pos;
11.
12.     printf("Enter a number : ");
13.     scanf("%d", &n);
14.     pos = int_msb_position(n);
15.     printf("\nPosition of MSB bit = %d\n", NUM_BITS_INT - (pos + 1));
16. }
17.
18. /* Function to find the MSB bit position */
19. int int_msb_position(int n)
20. {
21.     int i = 0, bit;
22.     while (i < NUM_BITS_INT)
23.     {
24.         bit = n & 0x80000000;
25.         if (bit == -0x80000000)
26.         {
27.             bit = 1;
28.         }
29.         if (bit == 1)
30.             break;
31.         n = n << 1;
32.         i++;
33.     }
34.     return i;
35. }

```

```

1. /*
2.  * C Program to use Bitwise Operations to Round(floor of) an Integer
3.  * to next Lower Multiple of 2
4.  */
5. #include <stdio.h>
6.
7. void main()

```

```

8.  {
9.      int x = 1, i, n;
10.
11.     printf("enter the number :");
12.     scanf("%d", &n);
13.     /* for positive values */
14.     if (n > 0)
15.     {
16.         for (; x <= n >> 1;)
17.         {
18.             x = x << 1;
19.         }
20.         n = x;
21.     }
22.     /* for negative values */
23.     else
24.     {
25.         n = ~n;
26.         n = n + 1;
27.         for (; x <= n >> 1;)
28.         {
29.             x = x << 1;
30.         }
31.         x = x << 1;
32.         x = ~x;
33.         x = x + 1;
34.         n = x;
35.     }
36.     printf("%d", n);
37. }
```

```

1. /*
2.  * C Program to Print the Range
3. */
4. #include <stdio.h>
5. #define SIZE(x) sizeof(x)*8
6.
7. void signed_one(int);
8. void unsigned_one(int);
9.
10. void main()
11. {
12.     printf("\nrange of int");
13.     signed_one(SIZE(int));
14.     printf("\nrange of unsigned int");
15.     unsigned_one(SIZE(unsigned int));
```

```

16.     printf("\nrange of char");
17.     signed_one(SIZE(char));
18.     printf("\nrange of unsigned char");
19.     unsigned_one(SIZE(unsigned char));
20.     printf("\nrange of short");
21.     signed_one(SIZE(short));
22.     printf("\nrange of unsigned short");
23.     unsigned_one(SIZE(unsigned short));
24.
25. }
26. /* RETURNS THE RANGE SIGNED*/
27. void signed_one(int count)
28. {
29.     int min, max, pro;
30.     pro = 1;
31.     while (count != 1)
32.     {
33.         pro = pro << 1;
34.         count--;
35.     }
36.     min = ~pro;
37.     min = min + 1;
38.     max = pro - 1;
39.     printf("\n%d to %d", min, max);
40. }
41. /* RETURNS THE RANGE UNSIGNED */
42. void unsigned_one(int count)
43. {
44.     unsigned int min, max, pro = 1;
45.
46.     while (count != 0)
47.     {
48.         pro = pro << 1;
49.         count--;
50.     }
51.     min = 0;
52.     max = pro - 1;
53.     printf("\n%u to %u", min, max);
54. }

```

## 2. C Examples on Integer Bits

```

1. /*
2. * C Program to check if all the bits of a given integer is one(1)
3. */
4. #include <stdio.h>

```

```

5.
6. int all_bits_one(int);
7. int count = 0;
8.
9. void main()
10. {
11.     int num;
12.     printf("enter the number:");
13.     scanf("%d", &num);
14.     num++;
15.     all_bits_one(num);
16.     if (count)
17.     {
18.         printf("false");
19.     }
20.     else
21.     {
22.         printf("true");
23.     }
24. }
25.
26./* checks whether all bits are 1 */
27.int all_bits_one(int x)
28.{
29.    if (x == 1)
30.        return 0;
31.    if (x % 2 != 0)
32.    {
33.        count++;
34.    }
35.    else
36.    {
37.        x = x / 2;
38.        all_bits_one(x);
39.    }
40.}

```

```

1. /*
2. * C Program to next higher value of n with same 1's
3. */
4. #define NUM_BITS_INT 32
5. #include <stdio.h>

```

```

6. int newcount(int);
7.
8. void main()
9. {
10.     int count1 = 0, k = 0, j, t, n, bit, i = 1, count = 0;
11.
12.     printf("Enter a number : ");
13.     scanf("%d", &n);
14.     t = n;
15.     while(t != 0)
16.     {
17.         bit = t & 0x80000000;
18.         if (bit == -0x80000000)
19.         {
20.             bit = 1;
21.         }
22.         if (bit == 1)
23.             count++;
24.         t = t << 1;
25.
26.     }
27.     for (k = n + 1;;k++)
28.     {
29.         count1 = newcount(k);
30.         if (count1 == count)
31.         {
32.             printf("The next highest number is : %d ", k);
33.             break;
34.         }
35.     }
36. }
37.
38./* To count the no. of 1's in the no. */
39.int newcount(int k)
40.{ 
41.     int bit, count = 0;
42.     while (k != 0)
43.     {
44.         bit = k & 0x80000000;
45.         if (bit == -0x80000000)
46.         {
47.             bit = 1;
48.         }

```

```

49.         if (bit == 1)
50.             count++;
51.         k = k << 1;
52.     }
53.     return(count);
54. }

55./*
56. * C Program to Count the Number of Trailing Zeroes in Integer
57. */
58.#include <stdio.h>
59.
60.void main()
61.{
62.    int j = 31, i, count = 0;
63.    unsigned int num;
64.    int b[32] = {0};
65.
66.    printf("enter the number:");
67.    scanf("%d", &num);
68.    while (num != 0)
69.    {
70.        if (num & 1 == 1)
71.        {
72.            break;
73.        }
74.        else
75.        {
76.            count++;
77.            num = num >> 1;
78.
79.        }
80.    }
81.    printf("\n%d", count);
82.}

```

```

1. /*
2. * C Program to Use Bitwise Operations to Count the Number of
3. * Leading Zero's in a Number x
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7. #define NUM_BITS_INT (sizeof(int)*8)

```

```

8. int find(int);
9.
10. void main()
11. {
12.     int n, i, a, count = 0, flag = 1, m = 1, j, cmp;
13.
14.     printf("Enter the number\n");
15.     scanf("%d", &n);
16.     a = n >> 31 & 1;
17.     if (a == 0)
18.     {
19.         for (i = (NUM_BITS_INT)-1;i >= 0;i--)
20.         {
21.             a = (n >> i)& 1;
22.             if (a == 0)
23.             {
24.                 count++;
25.             }
26.             else
27.             {
28.                 for (j = n + 1;;j++)
29.                 {
30.                     cmp = find(j);
31.                     if (cmp == (((NUM_BITS_INT)-1) - count) + 1)
32.                     {
33.                         printf("next higher power -> %d\n", j);
34.                         break;
35.                     }
36.                 }
37.                 break;
38.             }
39.         }
40.     }
41.     else
42.     {
43.         for (i = (NUM_BITS_INT)-1;i >= 0;i--)
44.         {
45.             a = (n >> i)& 1;
46.             if (a == 1)
47.             {
48.                 count++;
49.             }
50.             else

```

```

51.         {
52.             for (j = n + 1;j++)
53.             {
54.                 cmp = find(j);
55.                 if (cmp == (((NUM_BITS_INT)- 1) - count))
56.                 {
57.                     printf("next higher power -> %d\n", j);
58.                     break;
59.                 }
60.             }
61.             break;
62.         }
63.     }
64. }
65. }

66.

67./* To find trailing zero's */
68.int find(int n)
69.{
70.    int count = 0, a, flag = 1, i;
71.
72.    for (i = 0;i <= (NUM_BITS_INT) - 1;i++)
73.    {
74.        a = (n >> i) & 1;
75.        if (a == 1 && flag == 1)
76.        {
77.            return count;
78.        }
79.        else
80.        {
81.            count++;
82.            flag = 1;
83.        }
84.    }
85.}

```

```

1. /*
2. * C Program to find the Highest Bit Set for any given Integer
3. */
4. #include <stdio.h>
5. #define NUM_BITS sizeof(int)*8
6.

```

```

7. int highest_bit_set(int);
8. void display(int);
9. int i = NUM_BITS;
10.
11. void main()
12. {
13.     int num, pos;
14.
15.     printf("\nEnter the number:");
16.     scanf("%d", &num);
17.
18.     display(num);
19.     pos = highest_bit_set(num);
20.     printf("\nThe position of the highest bit set is %d", pos);
21. }
22./* RETURNS THE POSITION */
23.int highest_bit_set(int num)
24.{
25.    int count = 0;
26.    while (num >> 1 != 0)
27.    {
28.        count++;
29.        num = num >> 1;
30.    }
31.    return(count);
32.}
33./* DISPLAYS THE NUMBER IN BINARY REPRESENTATION */
34.void display(int num)
35.{
36.    int c;
37.    c = num & 1;
38.    if (i > 0)
39.    {
40.        i--;
41.        display(num >> 1);
42.    }
43.    printf("%d", c);
44.}

```

```

1. /*
2. * C Program to Count Number of bits set to 0 in a Integer x
3. */

```

```

4. #include <stdio.h>
5. #define NUM_BITS_INT (8*sizeof(int))
6.
7. int count_unset(int);
8.
9. int main()
10. {
11.     int i, num, snum, res, count = 0;
12.
13.     printf("\nEnter the number");
14.     scanf("%d", &num);
15.     /*
16.      * Check each bit whether the bit is set or unset
17.      * Uses >> and & operator for checking individual bits
18.      */
19.     for (i = 0;i <= NUM_BITS_INT;i++)
20.     {
21.         snum = num >> i;
22.         res = snum & 1;
23.         if (res == 0)
24.             count++;
25.     }
26.     printf("%d", count);
27. }
```

### 3. C Examples on Swapping and Replacing Integers

```

1. /*
2.  * C Program to Replace Bits in Integer from Specified Positions from
3.  * Another Integer
4.  */
5. #include <stdio.h>
6.
7. void replace_bits(int, int, int, int);
8.
9. int main()
10. {
11.     int number_x, number_y, start_pos, end_pos;
12.
13.     printf("\nEnter the number x in hexa decimal ");
```

```

14.     scanf("%x", &number_x);
15.     printf("\nEnter the number y in hexa decimal");
16.     scanf(" %x", &number_y);
17.     printf("\nEnter value for a");
18.     scanf("%d", &start_pos);
19.     printf("\nEnter value for b");
20.     scanf("%d", &end_pos);
21.     replace_bits(number_x, number_y, start_pos, end_pos);
22. }
23. /*
24. * Replace bits in first number from specified position with bits in second number
25. */
26. void replace_bits(int number_x, int number_y, int start_pos, int end_pos)
27. {
28.     int i, shift_y, ybit;
29.     long int temp, t;
30.
31. /*
32. * Replace the corresponding x bits by y bits
33. */
34.     for (i = start_pos;i <= end_pos;i++)
35.     {
36.         shift_y = number_y >> i;
37.         ybit = shift_y & 1;
38.         if (ybit == 1)
39.         {
40.             temp = 1 << i;
41.             number_x = number_x | temp;
42.
43.         }
44.         if (ybit == 0)
45.         {
46.             t = 0xFFFFFFFF;
47.             temp = 1 << i;
48.             start_pos = t ^ temp;
49.             number_x = number_x & start_pos;
50.         }
51.
52.     }
53.     printf("%x", number_x);
54. }
```

```

1. /*
2.  * C Program to Replace Bits in Integer x from Bit Position a to b from another
3.  * Integer y
4. #include <stdio.h>
5.
6. void changebits(int, int, int, int);
7.
8. int main()
9. {
10.     int num1, num2, pos1, pos2;
11.
12.     printf("**Replacing the bits in integer x from bit position a to b from another
13.     integer y**\n");
14.     printf("read number 1\n");
15.     scanf("%x", &num1);
16.     printf("Read number 2:\n");
17.     scanf("%x", &num2);
18.     printf("Read LSB position:\n");
19.     scanf("%d", &pos1);
20.     printf("Read MSB position:\n");
21.     scanf("%d", &pos2);
22.     changebits(num1, num2, pos1, pos2);
23.     return 0;
24. }
25.
26./*Function to swap bits in given positions*/
27.
28.void changebits(int num1, int num2, int pos1, int pos2)
29.{
30.    int temp1, temp_1, buffer2, bit1 = 0, bit2 = 0, counter = 0, a = 1;
31.
32.    temp1 = num1;
33.    temp_1 = num1;
34.    buffer2 = num2;
35.    for (;pos1 <= pos2;pos1++)
36.    {
37.        a = 1;
38.        num1 = temp_1;
39.        num2 = buffer2;
40.        while (counter <= pos1)
41.        {

```

```

42.         if (counter == pos1)
43.             bit1 = (num1&1);      //placing the bit of position 1 in bit1
44.             counter++;
45.             num1>> = 1;
46.     }
47.     counter = 0;
48.     while (counter <= pos1)
49.     {
50.         if (counter == pos1)
51.             bit2 = (num2&1);      //placing the bit of position 2 in bit2
52.             counter++;
53.             num2 >>= 1;
54.     }
55.     counter = 0;
56.     if (bit1 == bit2);
57.     else
58.     {
59.         while (counter++<pos1)
60.             a = a << 1;
61.             temp1 ^= a;      //placing the replaced bit integer into temp1 variable
62.     }
63.     counter = 0;
64. }
65. printf("the number num1 after shifting the bits is 0x%x\n", temp1);
66. }
```

```

1. /*
2. * C Program takes Byte as Input and returns all the Bits between
3. * given Positions
4. */
5. #include <stdio.h>
6.
7. int number_between_bit_positions(int,int,int);
8. int result = 0;
9.
10.int main()
11.{
```

12. int number, start\_pos, end\_pos;
- 13.
14. printf("\nEnter the number");
- 15. scanf("%d", &number);
- 16. printf("\nEnter the position of a and b");

```

17.     scanf("%d %d", &start_pos, &end_pos);
18.     result = number_between_bit_positions(number, start_pos, end_pos);
19.     printf("Byte Equivalent of bits between %d and %d positions %d", start_pos,
20.            end_pos, result);
21. }
22. int number_between_bit_positions(int number, int start_pos , int end_pos)
23. {
24.     int i, j, shift_num, res_val;
25.
26.     /*
27.      * Right shift to the specified start position,take the corresponding bits using
28.      * Left shift to locate the bits in their respective positions
29.      */
30.     for (i = start_pos, j = 0;i <= end_pos;i++,j++)
31.     {
32.         shift_num = number >> i;
33.         res_val = shift_num & 1;
34.         res_val = res_val << j;
35.         result += res_val;
36.     }
37.     return result;
38. }
```

```

1. /*
2.  * C Program to Swap the ith and jth Bits for a 32-Bit Integer
3. */
4. #include <stdio.h>
5.
6. int swap(int ,int);
7. int number, pos1, pos2;
8.
9. int main()
10. {
11.     int result, shift_pos1, shift_pos2;
12.
13.     printf("\nEnter Number");
14.     scanf("%d", &number);
15.     printf("\nEnter bit positions to swap");
16.     scanf("%d %d", &pos1, &pos2);
17.     shift_pos1 = number >> pos1;
```

```

18.     shift_pos2 = number >> pos2;
19.     result = swap(shift_pos1&1, shift_pos2&1);
20.     printf("%d\n", result);
21. }
22.
23. int swap(int pos1_val, int pos2_val)
24. {
25.     int temp1, temp2;
26.     long int base, base1;
27.     /*
28.      * If the pos1_val value is 1 then only pos2_val th bit is set to 1 by using <<
29.      and + operators
30.      */
31.     if (pos1_val == 1)
32.     {
33.         base1 = 1 << pos2;
34.         number = number + base1;
35.     }
36.     /*
37.      *If the pos2_val value is 1 then only pos2_val th bit is set to 1 by using <<
38.      and + operators
39.      */
40.     if (pos2_val == 1)
41.     {
42.         base1 = 1 << pos2;
43.         number = number + base1;
44.     }
45.     /*
46.      *If the pos1_val value is 0 then only pos2_val th bit is set to 0 using <<, ^
47.      and & operators
48.      */
49.     if (pos1_val == 0)
50.     {
51.         base = 0xFFFFFFFF;
52.         base1 = 1 << pos2;
53.         temp1 = base ^ base1;
54.         number = number & temp1;
55.     }
56.     /*
57.      *If the pos2_val value is 0 then only pos1_val th bit is set to 0 using <<, ^
58.      and & operators
59.      */

```

```

57.     if (pos2_val == 0)
58.     {
59.         base = 0xFFFFFFFF;
60.         base1 = 1 << pos1;
61.         temp2 = base ^ base1;
62.         number = number & temp2;
63.     }
64.     return number;
65. }
```

```

1. /*
2. * C Program to Check if a given Integer is Power of 2 using Bitwise Operators
3. */
4. #include <stdio.h>
5. #define NUM_BITS_INT (8*sizeof(int))
6.
7. int power_of_2(unsigned int);
8.
9. int main()
10. {
11.     unsigned int num;
12.
13.     printf("\nEnter Number");
14.     scanf("%d", &num);
15.     power_of_2(num);
16. }
17.
18. /*
19. * Finding the power of 2 using bit wise operators
20. */
21. int power_of_2(unsigned int x)
22. {
23.     int i, count = 0, result, shift_num;
24.
25.     for (i = 0;i <= NUM_BITS_INT;i++)
26.     {
27.         shift_num = x >> i;
28.         result = shift_num & 1;
29.         if (res == 1)
30.             count++;
31.     }
32.     /*
```

```

33.     *If number of bits set to 1 are odd then the number is power of 2
34.     *If number of bits set to 0 are even then the number is not power of 2
35.     */
36.     if (count % 2 == 1)
37.         printf("YES");
38.     else
39.         printf("NO");
40. }

```

```

1. /*
2.  * C Program to Swap two Integers without using Temporary Variables
3.  * and Bitwise Operations
4.  */
5. #include <stdio.h>
6.
7. // Function Prototype
8. void swap(int *, int *);
9.
10. void main()
11. {
12.     int x, y;
13.     printf("Enter 2 nos: \n");
14.     scanf("%d %d", &x, &y);
15.     printf("\nYou have entered x = %d y = %d \n", x, y);
16.     swap(&x,&y);      // passing the 2 nos to the swap function
17. }
18.
19.// function to swap the two numbers
20.void swap(int *a, int *b)
21.{
22.    *a = *a + *b;
23.    *b = *a - *b;
24.    *a = *a - *b;
25.    printf("Swapped . . .\n"); // printing the swapped numbers
26.    printf("x = %d y = %d\n", *a, *b);
27. }

```

#### 4. C Examples on String Positions

```

1. /*
2.  * C Program to Find the Position of String of 1-bits in a Number

```

```

3.  * for a given Length
4.  */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int n, len, pos = 0, i = 0, count = 0;
10.
11.    printf("**Finding the position of 1-bits in a number for given length**\n");
12.    printf("enter a number\n");
13.    scanf("%d", &n);
14.    printf("enter the length\n");
15.    scanf("%d", &len);
16.    while (i <= 32)
17.    {
18.        if ((n & 1) == 1)      //checking whether there is a 1-bit in the current
position
19.        {
20.            count++; //counting the consecutive 1's in the integer
21.            pos = i;
22.            if (count == len)    //checking whether the length matches
23.            {
24.                break;
25.            }
26.        }
27.        if ((n & 1) == 0)
28.        {
29.            count = 0;
30.        }
31.        n = n>>1;
32.        i++;
33.    }
34.    printf("the position of 1 in the string : %d\n", pos);
35. }
```

```

1. /*
2.  * C Program to Check if nth Bit in a 32-bit Integer is Set or not
3.  */
4. #include <stdio.h>
5.
6. /* gloabal varaibles */
7. int result,position;
```

```

8. /* function prototype */
9. int n_bit_position(int x,int position);
10.
11. void main()
12. {
13.     unsigned int number;
14.
15.     printf("Enter the unsigned integer:\n");
16.     scanf("%d", &number);
17.     printf("enter position\n");
18.     scanf("%d", &position);
19.     n_bit_position(i, position);
20.     if (result & 1)
21.         printf("YES\n");
22.     else
23.         printf("NO\n");
24. }
25.
26./* function to check whether the position is set to 1 or not */
27.int n_bit_position(int number,int position)
28.{
29.    result = (number>>(position));
30.}

```

```

1. /*
2. * C Program to Check if a given Bit Position is set to One or not
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     unsigned int number;
9.     int result, position;
10.
11.    printf("Enter the unsigned integer:\n");
12.    scanf("%d", &number);
13.    printf("enter position to be searched\n");
14.    scanf("%d", &position);
15.    result = (number >> (position));
16.    if (result & 1)
17.        printf("TRUE\n");
18.    else

```

```

19.         printf("FALSE\n");
20. }
```

## 5. C Examples on Binary Addition and Reversal of Bits

```

1. /*
2. *  C Program to Perform Binary Addition of Strings and Print it
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. /* global variables */
8. char s1[10], s2[10], s3[10];
9. int i, k;
10. char carry = '0';
11./* function prototype */
12.void binary_add(char *,char *);
13.
14.void main()
15.{
16.    printf("enter string1\n");
17.    scanf(" %[^\n]s", s1);
18.    printf("enter string2\n");
19.    scanf(" %[^\n]s", s2);
20.    binary_add(s1, s2);
21.    printf("binary addition of number is\n");
22.    if (carry == '1')
23.    {
24.        s3[i] = '1';
25.        for (i = 1;i <= k + 1;i++)
26.            printf("%c", s3[i]);
27.        printf("\n");
28.    }
29.    else
30.    {
31.        for (i = 1;i <= k + 1;i++)
32.            printf("%c", s3[i]);
33.        printf("\n");
34.    }
35.}
36.
37./* function to add two binary numbers in a string */
```

```

38. void binary_add(char *s1, char *s2)
39. {
40.     char *p1, *p2;
41.     p1 = s1;
42.     p2 = s2;
43.     k = strlen(s1);
44.
45.     for (; *p1 != '\0' && *p2 != '\0'; p1++, p2++);
46.     p1--;
47.     p2--;
48.     s3[k+1] = '\0';
49.     for (i = k + 1; i >= 1; i--, p1--, p2--)
50.     {
51.         if (*p1 == '0' && *p2 == '0' && carry == '0')
52.         {
53.             s3[i] = (*p1 ^ *p2) ^ carry;
54.             carry = '0';
55.         }
56.         else if (*p1 == '0' && *p2 == '0' && carry == '1')
57.         {
58.             s3[i] = (*p1 ^ *p2)^ carry;
59.             carry = '0';
60.         }
61.         else if (*p1 == '0' && *p2 == '1' && carry == '0')
62.         {
63.             s3[i] = (*p1 ^ *p2)^ carry;
64.             carry = '0';
65.         }
66.         else if (*p1 == '0' && *p2 == '1' && carry == '1')
67.         {
68.             s3[i] = (*p1 ^ *p2)^ carry;
69.             carry = '1';
70.         }
71.         else if (*p1 == '1' && *p2 == '0' && carry == '0')
72.         {
73.             s3[i] = (*p1 ^ *p2)^ carry;
74.             carry = '0';
75.         }
76.         else if (*p1 == '1' && *p2 == '0' && carry == '1')
77.         {
78.             s3[i] = (*p1 ^ *p2)^ carry;
79.             carry = '1';
80.         }

```

```

81.     else if (*p1 == '1' && *p2 == '1' && carry == '0')
82.     {
83.         s3[i] = (*p1 ^ *p2)^ carry;
84.         carry = '1';
85.     }
86.     else
87.     {
88.         s3[i] = (*p1 ^ *p2)^ carry;
89.         carry = '1';
90.     }
91. }
92. }
```

```

1. /*
2.  * C Program to Reverse all the Bits of an 32-bit Integer using
3.  * Bitwise
4. */
5. #include <stdio.h>
6. #define NUM_BITS_INT sizeof(int)*8
7.
8. void main()
9. {
10.     unsigned int number;
11.     int i = 0, hexadecimal, rev = 0, bit;
12.
13.     printf("enter the hexdecimal value\n");
14.     scanf("0x%number", &hexadecimal);
15.     while (i++ < NUM_BITS_INT)
16.     {
17.         bit = hexadecimal & 1;
18.         hexadecimal = hexadecimal >> 1;
19.         rev = rev ^ bit;
20.         if (i < NUM_BITS_INT)
21.             rev = rev << 1;
22.     }
23.     printf("reverse of hexdecimal value is 0x%number", rev);
24. }
```

## 6. C Examples on Counting and Swapping the contents of strings using Bitwise Operations

```

1. /*
2.  * C Program to Count the Number of Bits needed to be Flipped
3.  * to Integer X to Generate Integer Y
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7. #define NUM_BITS_INT (sizeof(int)*8)
8.
9. void main()
10. {
11.     int n, m, i, count = 0, a, b;
12.
13.     printf("Enter the number\n");
14.     scanf("%d", &n);
15.     printf("Enter another number\n");
16.     scanf("%d", &m);
17.     for (i = NUM_BITS_INT-1;i >= 0;i--)
18.     {
19.         a = (n >> i)& 1;
20.         b = (m >> i)& 1;
21.         if (a != b)
22.             count++;
23.     }
24.     printf("flip count = %d\n", count);
25. }
```

```

1. /*
2.  * C Program to Count the Number of Bits set to One using
3.  * Bitwise Operations
4. */
5. #include <stdio.h>
6.
7. int main()
8. {
9.     unsigned int number;
10.    int count = 0;
11.
12.    printf("Enter the unsigned integer:\n");
13.    scanf("%d", &number);
14.    while (number != 0)
15.    {
16.        if ((number & 1) == 1)
```

```

17.         count++;
18.     number = number >> 1;
19. }
20. printf("number of one's are :\n%d\n", count);
21. return 0;
22. }

23./*
24. * C Program To Identify the Missing Number in an Integer
25. * Array of Size N-1 with Numbers[1,N]
26. */
27.#include <stdio.h>
28.#define MAX 15
29.int missing_number_array(int [],int);
30.
31.int main()
32.{
33.    int a[MAX], num, i, n;
34.
35.    printf("enter the range of array\n");
36.    scanf("%d", &n);
37.    for (i = 0;i < n;i++)
38.    {
39.        printf("enter a[%d]element into the array:", i);
40.        scanf("%d", &a[i]);
41.    }
42.    num = missing_number_array(a, n);
43.    printf("The missing number -> %d\n", num);
44.}

45.

46./* To find the missing number in array */
47.int missing_number_array(int a[], int n)
48.{
49.    int i;
50.    int s1 = 0;
51.    int s2 = 0;
52.
53.    for (i = 0;i < n;i++)
54.        s1 = s1 ^ a[i];
55.    for (i = 1;i <= n + 1;i++)
56.        s2 = s2 ^ i;
57.    return (s1 ^ s2);
58.}

```

```
1. /*
2.  * C Program to Check whether the given Number is Palindrome
3.  * or not using Bitwise Operator
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #define SIZE 8
8. /* Function Prototype */
9. int is_palindrome(unsigned char[]);
10.
11 void main()
12 {
13     int num, num1 = 0, i = 0, j = SIZE - 1, res;
14     unsigned char c[SIZE];
15     printf("Enter a number(max 255)");
16     scanf("%d", &num);
17     num1 = num;
18     while (num != 0)
19     {
20         c[j] = num&1;
21         j--;
22         num = num>>1; /* Shifting right the given number by 1 bit */
23     }
24     printf("The number %d in binary is:", num1);
25     for (i = 0;i < SIZE;i++)
26     {
27         printf("%d", c[i]);
28     }
29     res = is_palindrome(c);    /* Calling Function */
30     if (res == 0)
31     {
32         printf("\nNUMBER IS PALINDROME\n");
33     }
34     else
35     {
36         printf("\nNUMBER IS NOT PALINDROME\n");
37     }
38 }
39.
40 /* Code to check if the number is palindrome or not */
41 int is_palindrome(unsigned char c[])
```

```

42. {
43.     char temp[SIZE];
44.     int i, j, flag = 0;
45.     for (i = 0, j = SIZE - 1; i < SIZE, j >= 0; i++, j--)
46.     {
47.         temp[j] = c[i];
48.     }
49.     for (i = 0; i < SIZE; i++)
50.     {
51.         if (temp[i] != c[i])
52.         {
53.             flag = 1;
54.         }
55.     }
56.     return flag;
57. }
```

```

1. /*
2. * C Program to Swap two Numbers using Bitwise operators
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. /* Function Prototype */
8. void swap(int*, int *);
9.
10. void main()
11. {
12.     int num1, num2;
13.     printf("\nEnter two numbers:");
14.     scanf("%d %d", &num1, &num2);
15.     printf("\nThe numbers before swapping are Number1= %d Number2 = %d", num1,
16.            num2);
16.     swap(&num1, &num2);           /* Call by Reference to function swap */
17.     printf("\nThe numbers after swapping are Number1= %d Number2 = %d", num1, num2);
18. }
19.
20. /* Code to swap two numbers using bitwise operator */
21. void swap(int *x, int *y)
22. {
23.     *x = *x ^ *y;
24.     *y = *x ^ *y;
```

```

25.     *x = *x ^ *y;
26. }
```

## 6. C Programming Examples on Linked List

### 1. C Examples on Creating and Displaying the elements of a Linked List

```

1. /*
2.  * C program to create a linked list and display the elements in the list
3. */
4. #include <stdio.h>
5. #include <malloc.h>
6. #include <stdlib.h>
7.
8. void main()
9. {
10.     struct node
11.     {
12.         int num;
13.         struct node *ptr;
14.     };
15.     typedef struct node NODE;
16.
17.     NODE *head, *first, *temp = 0;
18.     int count = 0;
19.     int choice = 1;
20.     first = 0;
21.
22.     while (choice)
23.     {
24.         head = (NODE *)malloc(sizeof(NODE));
25.         printf("Enter the data item\n");
26.         scanf("%d", &head-> num);
27.         if (first != 0)
28.         {
29.             temp->ptr = head;
30.             temp = head;
31.         }
32.         else
33.         {
34.             first = temp = head;
35.         }
36.     }
37. }
```

```

36.         fflush(stdin);
37.         printf("Do you want to continue(Type 0 or 1)?\n");
38.         scanf("%d", &choice);
39.
40.     }
41.     temp->ptr = 0;
42.     /* reset temp to the beginning */
43.     temp = first;
44.     printf("\n status of the linked list is\n");
45.     while (temp != 0)
46.     {
47.         printf("%d=>", temp->num);
48.         count++;
49.         temp = temp -> ptr;
50.     }
51.     printf("NULL\n");
52.     printf("No. of nodes in the list = %d\n", count);
53. }
```

```

1. /*
2.  * C Program to Read a Linked List in Reverse
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void reversedisplay(struct node *);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     struct node_occur *head = NULL;
22.     int n;
```

```
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     printf("Displaying the nodes in the list:\n");
27.     display(p);
28.     printf("Displaying the list in reverse:\n");
29.     reversedisplay(p);
30.     release(&p);
31.
32.     return 0;
33. }
34.
35. void reversedisplay(struct node *head)
36. {
37.     if (head != NULL)
38.     {
39.         reversedisplay(head->next);
40.         printf("%d\t", head->num);
41.     }
42. }
43.
44. void create(struct node **head)
45. {
46.     int c, ch;
47.     struct node *temp, *rear;
48.
49.     do
50.     {
51.         printf("Enter number: ");
52.         scanf("%d", &c);
53.         temp = (struct node *)malloc(sizeof(struct node));
54.         temp->num = c;
55.         temp->next = NULL;
56.         if (*head == NULL)
57.         {
58.             *head = temp;
59.         }
60.         else
61.         {
62.             rear->next = temp;
63.         }
64.         rear = temp;
65.         printf("Do you wish to continue [1/0]: ");
66.         scanf("%d", &ch);
```

```

67.     } while (ch != 0);
68.     printf("\n");
69. }
70.
71. void display(struct node *p)
72. {
73.     while (p != NULL)
74.     {
75.         printf("%d\t", p->num);
76.         p = p->next;
77.     }
78.     printf("\n");
79. }
80.
81. void release(struct node **head)
82. {
83.     struct node *temp = *head;
84.     *head = (*head)->next;
85.     while ((*head) != NULL)
86.     {
87.         free(temp);
88.         temp = *head;
89.         (*head) = (*head)->next;
90.     }
91. }
```

## 2. C Examples on Search and Display Functions of a Linked List

```

1. /*
2.  * C program to search for an element in Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13. void generate(struct node **, int);
14. void search(struct node *, int, int);
```

```

15. void delete(struct node **);
16.
17. int main()
18. {
19.     struct node *head;
20.     int key, num;
21.
22.     printf("Enter the number of nodes: ");
23.     scanf("%d", &num);
24.     generate(&head, num);
25.     printf("\nEnter key to search: ");
26.     scanf("%d", &key);
27.     search(head, key, num);
28.     delete(&head);
29. }
30.
31. void generate(struct node **head, int num)
32. {
33.     int i;
34.     struct node *temp;
35.
36.     for (i = 0; i < num; i++)
37.     {
38.         temp = (struct node *)malloc(sizeof(struct node));
39.         temp->a = rand() % num;
40.         printf("%d    ", temp->a);
41.         if (*head == NULL)
42.         {
43.             *head = temp;
44.             (*head)->next = NULL;
45.         }
46.         else
47.         {
48.             temp->next = *head;
49.             *head = temp;
50.         }
51.     }
52. }
53.
54. void search(struct node *head, int key, int index)
55. {
56.     if (head->a == key)
57.     {

```

```

58.         printf("Key found at Position: %d\n", index);
59.     }
60.     if (head->next == NULL)
61.     {
62.         return;
63.     }
64.     search(head->next, key, index - 1);
65. }
66.
67. void delete(struct node **head)
68. {
69.     struct node *temp;
70.     while (*head != NULL)
71.     {
72.         temp = *head;
73.         *head = (*head)->next;
74.         free(temp);
75.     }
76. }
```

```

1. /*
2.  * C Program to Search for an Element in the Linked List without
3.  * using Recursion
4. */
5.
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. struct node
10. {
11.     int a;
12.     struct node *next;
13. };
14.
15. void generate(struct node **, int);
16. void search(struct node *, int);
17. void delete(struct node **);
18.
19. int main()
20. {
21.     struct node *head = NULL;
22.     int key, num;
```

```

23.
24.     printf("Enter the number of nodes: ");
25.     scanf("%d", &num);
26.     printf("\nDisplaying the list\n");
27.     generate(&head, num);
28.     printf("\nEnter key to search: ");
29.     scanf("%d", &key);
30.     search(head, key);
31.     delete(&head);
32.
33.     return 0;
34. }
35.
36. void generate(struct node **head, int num)
37. {
38.     int i;
39.     struct node *temp;
40.
41.     for (i = 0; i < num; i++)
42.     {
43.         temp = (struct node *)malloc(sizeof(struct node));
44.         temp->a = rand() % num;
45.         if (*head == NULL)
46.         {
47.             *head = temp;
48.             temp->next = NULL;
49.         }
50.         else
51.         {
52.             temp->next = *head;
53.             *head = temp;
54.         }
55.         printf("%d ", temp->a);
56.     }
57. }
58.
59. void search(struct node *head, int key)
60. {
61.     while (head != NULL)
62.     {
63.         if (head->a == key)
64.         {
65.             printf("key found\n");

```

```

66.         return;
67.     }
68.     head = head->next;
69. }
70. printf("Key not found\n");
71. }
72.
73. void delete(struct node **head)
74. {
75.     struct node *temp;
76.
77.     while (*head != NULL)
78.     {
79.         temp = *head;
80.         *head = (*head)->next;
81.         free(temp);
82.     }
83. }
```

```

1. /*
2.  * C Program to Display the Nodes of a Linked List in Reverse without
3.  * using Recursion
4. */
5.
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. struct node
10. {
11.     int visited;
12.     int a;
13.     struct node *next;
14. };
15.
16. void generate(struct node **);
17. void display(struct node *);
18. void linear(struct node *);
19. void delete(struct node **);
20.
21. int main()
22. {
23.     struct node *head = NULL;
```

```
24.
25.     generate(&head);
26.     printf("\nPrinting the list in linear order\n");
27.     linear(head);
28.     printf("\nPrinting the list in reverse order\n");
29.     display(head);
30.     delete(&head);
31.
32.     return 0;
33. }
34.
35. void display(struct node *head)
36. {
37.     struct node *temp = head, *prev = head;
38.
39.     while (temp->visited == 0)
40.     {
41.         while (temp->next != NULL && temp->next->visited == 0)
42.         {
43.             temp = temp->next;
44.         }
45.         printf("%d ", temp->a);
46.         temp->visited = 1;
47.         temp = head;
48.     }
49. }
50.
51. void linear(struct node *head)
52. {
53.     while (head != NULL)
54.     {
55.         printf("%d ", head->a);
56.         head = head->next;
57.     }
58.     printf("\n");
59. }
60.
61. void generate(struct node **head)
62. {
63.     int num, i;
64.     struct node *temp;
65.
66.     printf("Enter length of list: ");
```

```

67.     scanf("%d", &num);
68.     for (i = num; i > 0; i--)
69.     {
70.         temp = (struct node *)malloc(sizeof(struct node));
71.         temp->a = i;
72.         temp->visited = 0;
73.         if (*head == NULL)
74.         {
75.             *head = temp;
76.             (*head)->next = NULL;
77.         }
78.         else
79.         {
80.             temp->next = *head;
81.             *head = temp;
82.         }
83.     }
84. }
85.
86. void delete(struct node **head)
87. {
88.     struct node *temp;
89.     while (*head != NULL)
90.     {
91.         temp = *head;
92.         *head = (*head)->next;
93.         free(temp);
94.     }
95. }
```

```

1. /*
2.  * Recursive C program to display members of a linked list
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
```

```
13. void generate(struct node **);
14. void display(struct node* );
15. void delete(struct node **);
16.
17. int main()
18. {
19.     struct node *head = NULL;
20.
21.     generate(&head);
22.     display(head);
23.     delete(&head);
24.     return 0;
25. }
26.
27. void generate(struct node **head)
28. {
29.     int num = 10, i;
30.     struct node *temp;
31.
32.     for (i = 0; i < num; i++)
33.     {
34.         temp = (struct node *)malloc(sizeof(struct node));
35.         temp->a = i;
36.         if (*head == NULL)
37.         {
38.             *head = temp;
39.             (*head)->next = NULL;
40.         }
41.         else
42.         {
43.             temp->next = *head;
44.             *head = temp;
45.         }
46.     }
47. }
48.
49. void display(struct node *head)
50. {
51.     printf("%d    ", head->a);
52.     if (head->next == NULL)
53.     {
54.         return;
55.     }
```

```

56.     display(head->next);
57. }
58.
59. void delete(struct node **head)
60. {
61.     struct node *temp;
62.     while (*head != NULL)
63.     {
64.         temp = *head;
65.         *head = (*head)->next;
66.         free(temp);
67.     }
68. }
```

```

1. /*
2.  * C Program to Display all the Nodes in a Linked List without using
3.  * Recursion
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *next;
12. };
13.
14. void generate(struct node **);
15. void display(struct node* );
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     display(head);
24.     delete(&head);
25.     return 0;
26. }
27.
28. void generate(struct node **head)
```

```
29. {
30.     int num = 10, i;
31.     struct node *temp;
32.
33.     for (i = 0; i < num; i++)
34.     {
35.         temp = (struct node *)malloc(sizeof(struct node));
36.         temp->a = i;
37.         if (*head == NULL)
38.         {
39.             *head = temp;
40.             (*head)->next = NULL;
41.         }
42.         else
43.         {
44.             temp->next = *head;
45.             *head = temp;
46.         }
47.     }
48. }
49.
50. void display(struct node *head)
51. {
52.     while (head != NULL)
53.     {
54.         printf("%d    ", head->a);
55.         head = head->next;
56.     }
57.     printf("\n");
58. }
59.
60. void delete(struct node **head)
61. {
62.     struct node *temp;
63.     while (*head != NULL)
64.     {
65.         temp = *head;
66.         *head = (*head)->next;
67.         free(temp);
68.     }
69. }
```

```
1. /*
2.  * C Program to Display the Nodes of a Linked List in Reverse without
3.  * using Recursion
4. */
5.
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. struct node
10. {
11.     int visited;
12.     int a;
13.     struct node *next;
14. };
15.
16. void generate(struct node **);
17. void display(struct node *);
18. void linear(struct node *);
19. void delete(struct node **);
20.
21. int main()
22. {
23.     struct node *head = NULL;
24.
25.     generate(&head);
26.     printf("\nPrinting the list in linear order\n");
27.     linear(head);
28.     printf("\nPrinting the list in reverse order\n");
29.     display(head);
30.     delete(&head);
31.
32.     return 0;
33. }
34.
35. void display(struct node *head)
36. {
37.     struct node *temp = head, *prev = head;
38.
39.     while (temp->visited == 0)
40.     {
41.         while (temp->next != NULL && temp->next->visited == 0)
42.         {
43.             temp = temp->next;
```

```
44.     }
45.     printf("%d ", temp->a);
46.     temp->visited = 1;
47.     temp = head;
48. }
49. }
50.
51. void linear(struct node *head)
52. {
53.     while (head != NULL)
54.     {
55.         printf("%d ", head->a);
56.         head = head->next;
57.     }
58.     printf("\n");
59. }
60.
61. void generate(struct node **head)
62. {
63.     int num, i;
64.     struct node *temp;
65.
66.     printf("Enter length of list: ");
67.     scanf("%d", &num);
68.     for (i = num; i > 0; i--)
69.     {
70.         temp = (struct node *)malloc(sizeof(struct node));
71.         temp->a = i;
72.         temp->visited = 0;
73.         if (*head == NULL)
74.         {
75.             *head = temp;
76.             (*head)->next = NULL;
77.         }
78.         else
79.         {
80.             temp->next = *head;
81.             *head = temp;
82.         }
83.     }
84. }
85.
86. void delete(struct node **head)
```

```

87. {
88.     struct node *temp;
89.     while (*head != NULL)
90.     {
91.         temp = *head;
92.         *head = (*head)->next;
93.         free(temp);
94.     }
95. }
```

### 3. C Examples on Implementation of Count, Length and Print Operations on a Linked List

```

1. /*
2.  * C program to find the number of occurrences of a given number in a
3.  * list
4. */
5. #include <stdio.h>
6.
7. void occur(int [], int, int, int, int *);
8.
9. int main()
10. {
11.     int size, key, count = 0;
12.     int list[20];
13.     int i;
14.
15.     printf("Enter the size of the list: ");
16.     scanf("%d", &size);
17.     printf("Printing the list:\n");
18.     for (i = 0; i < size; i++)
19.     {
20.         list[i] = rand() % size;
21.         printf("%d    ", list[i]);
22.     }
23.     printf("\nEnter the key to find its occurrence: ");
24.     scanf("%d", &key);
25.     occur(list, size, 0, key, &count);
26.     printf("%d occurs for %d times.\n", key, count);
27.     return 0;
28. }
29.
```

```

30. void occur(int list[], int size, int index, int key, int *count)
31. {
32.     if (size == index)
33.     {
34.         return;
35.     }
36.     if (list[index] == key)
37.     {
38.         *count += 1;
39.     }
40.     occur(list, size, index + 1, key, count);
41. }
```

```

1. /*
2.  * C Program Count the Number of Occurrences of an Element in the Linked List
3.  * without using Recursion
4. */
5. #include <stdio.h>
6.
7. int occur(int [], int, int);
8.
9. int main()
10. {
11.     int size, key, count;
12.     int list[20];
13.     int i;
14.
15.     printf("Enter the size of the list: ");
16.     scanf("%d", &size);
17.     printf("Printing the list:\n");
18.     for (i = 0; i < size; i++)
19.     {
20.         list[i] = rand() % size;
21.         printf("%d    ", list[i]);
22.     }
23.     printf("\nEnter the key to find it's occurrence: ");
24.     scanf("%d", &key);
25.     count = occur(list, size, key);
26.     printf("%d occurs for %d times.\n", key, count);
27.     return 0;
28. }
```

```

30. int occur(int list[], int size, int key)
31. {
32.     int i, count = 0;
33.
34.     for (i = 0; i < size; i++)
35.     {
36.         if (list[i] == key)
37.         {
38.             count += 1;
39.         }
40.     }
41.     return count;
42. }
```

```

1. /*
2.  * C program to find the length of a string
3. */
4. #include <stdio.h>
5.
6. int length(char [], int);
7. int main()
8. {
9.     char word[20];
10.    int count;
11.
12.    printf("Enter a word to count it's length: ");
13.    scanf("%s", word);
14.    count = length(word, 0);
15.    printf("The number of characters in %s are %d.\n", word, count);
16.    return 0;
17. }
18.
19. int length(char word[], int index)
20. {
21.     if (word[index] == '\0')
22.     {
23.         return 0;
24.     }
25.     return (1 + length(word, index + 1));
26. }
```

```

1. /*
2.  * C Program find the Length of the Linked List without using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13.
14. void generate(struct node **);
15. int length(struct node* );
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int count;
22.
23.     generate(&head);
24.     count = length(head);
25.     printf("The number of nodes are: %d\n", count);
26.     delete(&head);
27.
28.     return 0;
29. }
30.
31. void generate(struct node **head)
32. {
33.     /* for unknown number of nodes use num = rand() % 20; */
34.     int num = 10, i;
35.     struct node *temp;
36.
37.     for (i = 0; i < num; i++)
38.     {
39.         temp = (struct node *)malloc(sizeof(struct node));
40.         temp->a = i;
41.         if (*head == NULL)
42.         {
43.             *head = temp;

```

```

44.         (*head)->next = NULL;
45.     }
46. else
47. {
48.     temp->next = *head;
49.     *head = temp;
50. }
51. }
52. }

53.

54. int length(struct node *head)
55. {
56.     int num = 0;
57.     while (head != NULL)
58.     {
59.         num += 1;
60.         head = head->next;
61.     }
62.     return num;
63. }
64.

65. void delete(struct node **head)
66. {
67.     struct node *temp;
68.     while (*head != NULL)
69.     {
70.         temp = *head;
71.         *head = (*head)->next;
72.         free(temp);
73.     }
74. }

```

```

1. /*
2.  * C Program to Print the Alternate Nodes in a Linked List using Recursion
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;

```

```
11. };
12.
13. void generate(struct node **);
14. void display(struct node *);
15. void delete(struct node **);
16.
17. int main()
18. {
19.     struct node *head = NULL;
20.
21.     generate(&head);
22.     printf("\nDisplaying the alternate nodes\n");
23.     display(head);
24.     delete(&head);
25.
26.     return 0;
27. }
28.
29. void display(struct node *head)
30. {
31.     static flag = 0;
32.     if(head != NULL)
33.     {
34.         if (!(flag % 2))
35.         {
36.             printf("%d ", head->a);
37.         }
38.         flag++;
39.         display(head->next);
40.     }
41. }
42.
43. void generate(struct node **head)
44. {
45.     int num, i;
46.     struct node *temp;
47.
48.     printf("Enter length of list: ");
49.     scanf("%d", &num);
50.     for (i = num; i > 0; i--)
51.     {
52.         temp = (struct node *)malloc(sizeof(struct node));
53.         temp->a = i;
```

```

54.     if (*head == NULL)
55.     {
56.         *head = temp;
57.         (*head)->next = NULL;
58.     }
59.     else
60.     {
61.         temp->next = *head;
62.         *head = temp;
63.     }
64. }
65. }
66.
67. void delete(struct node **head)
68. {
69.     struct node *temp;
70.     while (*head != NULL)
71.     {
72.         temp = *head;
73.         *head = (*head)->next;
74.         free(temp);
75.     }
76. }
```

```

1. /*
2.  * C Program to Print the Alternate Nodes in a Linked List without
3.  * using Recursion
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *next;
12. };
13.
14. void generate(struct node **);
15. void display(struct node *);
16. void delete(struct node **);
17.
18. int main()
```

```
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     printf("\nDisplaying the alternate nodes\n");
24.     display(head);
25.     delete(&head);
26.
27.     return 0;
28. }
29.
30. void display(struct node *head)
31. {
32.     int flag = 0;
33.
34.     while(head != NULL)
35.     {
36.         if (!(flag % 2))
37.         {
38.             printf("%d ", head->a);
39.         }
40.         flag++;
41.         head = head->next;
42.     }
43. }
44.
45. void generate(struct node **head)
46. {
47.     int num, i;
48.     struct node *temp;
49.
50.     printf("Enter length of list: ");
51.     scanf("%d", &num);
52.     for (i = num; i > 0; i--)
53.     {
54.         temp = (struct node *)malloc(sizeof(struct node));
55.         temp->a = i;
56.         if (*head == NULL)
57.         {
58.             *head = temp;
59.             (*head)->next = NULL;
60.         }
61.         else
```

```

62.      {
63.          temp->next = *head;
64.          *head = temp;
65.      }
66.  }
67. }
68.
69. void delete(struct node **head)
70. {
71.     struct node *temp;
72.     while (*head != NULL)
73.     {
74.         temp = *head;
75.         *head = (*head)->next;
76.         free(temp);
77.     }
78. }

```

#### 4. C Examples on Implementation of other Data Structures using Linked List

```

/*
 * C Program to Implement a Stack using Linked List
 */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
void display();
void destroy();
void stack_count();
void create();

int count = 0;

```

```
void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                push(no);
                break;
            case 2:
                pop();
                break;
            case 3:
                if (top == NULL)
                    printf("No elements in stack");
                else
                {
                    e = topelement();
                    printf("\n Top element : %d", e);
                }
                break;
            case 4:
                empty();
                break;
        }
    }
}
```

```

        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            stack_count();
            break;
        case 8:
            destroy();
            break;
        default :
            printf(" Wrong choice, Please enter correct choice   ");
            break;
    }
}
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        top = temp;
    }
}

```

```

        temp->info = data;
        top = temp;
    }
    count++;
}

/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
}

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}

/* Return top element */

```

```

int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}

```

```

/*
 * C Program to Implement Queue Data Structure using Linked List
 */
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int info;

```

```

    struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Enque");
    printf("\n 2 - Deque");
    printf("\n 3 - Front element");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Display");
    printf("\n 7 - Queue size");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                enq(no);
                break;
            case 2:
                deq();
                break;
            case 3:
                e = frontelement();
                if (e != 0)
                    printf("Front element : %d", e);
        }
    }
}

```

```

        else
            printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice   ");
            break;
    }
}
}

/* Create an empty queue */
void create()
{
    front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
    printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
}

```

```
else
{
    temp=(struct node *)malloc(1*sizeof(struct node));
    rear->ptr = temp;
    temp->info = data;
    temp->ptr = NULL;

    rear = temp;
}
count++;
}

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeueing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty queue");
        return;
    }
    else
        if (front1->ptr != NULL)
```

```

    {
        front1 = front1->ptr;
        printf("\n Dequeued value : %d", front->info);
        free(front);
        front = front1;
    }
    else
    {
        printf("\n Dequeued value : %d", front->info);
        free(front);
        front = NULL;
        rear = NULL;
    }
    count--;
}

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
        printf("\n Queue empty");
    else
        printf("Queue not empty");
}

/*
 * C Program to Implement a Doubly Linked List & provide Insertion, Deletion &
Display Operations
*/
#include <stdio.h>
#include <stdlib.h>

struct node

```

```

{
    struct node *prev;
    int n;
    struct node *next;
}*h,*temp,*temp1,*temp2,*temp4;

void insert1();
void insert2();
void insert3();
void traversebeg();
void traverseend(int);
void sort();
void search();
void update();
void delete();

int count = 0;

void main()
{
    int ch;

    h = NULL;
    temp = temp1 = NULL;

    printf("\n 1 - Insert at beginning");
    printf("\n 2 - Insert at end");
    printf("\n 3 - Insert at position i");
    printf("\n 4 - Delete at i");
    printf("\n 5 - Display from beginning");
    printf("\n 6 - Display from end");
    printf("\n 7 - Search for element");
    printf("\n 8 - Sort the list");
    printf("\n 9 - Update an element");
    printf("\n 10 - Exit");

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:

```

```

        insert1();
        break;
    case 2:
        insert2();
        break;
    case 3:
        insert3();
        break;
    case 4:
        delete();
        break;
    case 5:
        traversebeg();
        break;
    case 6:
        temp2 = h;
        if (temp2 == NULL)
            printf("\n Error : List empty to display ");
        else
        {
            printf("\n Reverse order of linked list is : ");
            traverseend(temp2->n);
        }
        break;
    case 7:
        search();
        break;
    case 8:
        sort();
        break;
    case 9:
        update();
        break;
    case 10:
        exit(0);
    default:
        printf("\n Wrong choice menu");
    }
}
}

/* TO create an empty node */
void create()

```

```

{
    int data;

    temp =(struct node *)malloc(1*sizeof(struct node));
    temp->prev = NULL;
    temp->next = NULL;
    printf("\n Enter value to node : ");
    scanf("%d", &data);
    temp->n = data;
    count++;
}

/* To insert at beginning */
void insert1()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp->next = h;
        h->prev = temp;
        h = temp;
    }
}

/* To insert at end */
void insert2()
{
    if (h == NULL)
    {
        create();
        h = temp;
        temp1 = h;
    }
    else
    {
        create();
        temp1->next = temp;
    }
}

```

```

        temp->prev = temp1;
        temp1 = temp;
    }
}

/* To insert at any position */
void insert3()
{
    int pos, i = 2;

    printf("\n Enter position to be inserted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Position out of range to insert");
        return;
    }
    if ((h == NULL) && (pos != 1))
    {
        printf("\n Empty list cannot insert other than 1st position");
        return;
    }
    if ((h == NULL) && (pos == 1))
    {
        create();
        h = temp;
        temp1 = h;
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        create();
        temp->prev = temp2;
        temp->next = temp2->next;
        temp2->next->prev = temp;
        temp2->next = temp;
    }
}

```

```

    }

}

/* To delete an element */
void delete()
{
    int i = 1, pos;

    printf("\n Enter position to be deleted : ");
    scanf("%d", &pos);
    temp2 = h;

    if ((pos < 1) || (pos >= count + 1))
    {
        printf("\n Error : Position out of range to delete");
        return;
    }
    if (h == NULL)
    {
        printf("\n Error : Empty list no elements to delete");
        return;
    }
    else
    {
        while (i < pos)
        {
            temp2 = temp2->next;
            i++;
        }
        if (i == 1)
        {
            if (temp2->next == NULL)
            {
                printf("Node deleted from list");
                free(temp2);
                temp2 = h = NULL;
                return;
            }
        }
        if (temp2->next == NULL)
        {
            temp2->prev->next = NULL;
            free(temp2);
        }
    }
}

```

```

        printf("Node deleted from list");
        return;
    }
    temp2->next->prev = temp2->prev;
    if (i != 1)
        temp2->prev->next = temp2->next; /* Might not need this statement if i
== 1 check */
    if (i == 1)
        h = temp2->next;
    printf("\n Node deleted");
    free(temp2);
}
count--;
}

/* Traverse from beginning */
void traversebeg()
{
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("List empty to display \n");
        return;
    }
    printf("\n Linked list elements from begining : ");

    while (temp2->next != NULL)
    {
        printf(" %d ", temp2->n);
        temp2 = temp2->next;
    }
    printf(" %d ", temp2->n);
}

/* To traverse from end recursively */
void traverseend(int i)
{
    if (temp2 != NULL)
    {
        i = temp2->n;
        temp2 = temp2->next;
        traverseend(i);
    }
}
```

```

        printf(" %d ", i);
    }
}

/* To search for an element in the list */
void search()
{
    int data, count = 0;
    temp2 = h;

    if (temp2 == NULL)
    {
        printf("\n Error : List empty to search for data");
        return;
    }
    printf("\n Enter value to search : ");
    scanf("%d", &data);
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {
            printf("\n Data found in %d position", count + 1);
            return;
        }
        else
            temp2 = temp2->next;
        count++;
    }
    printf("\n Error : %d not found in list", data);
}

/* To update a node value in the list */
void update()
{
    int data, data1;

    printf("\n Enter node data to be updated : ");
    scanf("%d", &data);
    printf("\n Enter new data : ");
    scanf("%d", &data1);
    temp2 = h;
    if (temp2 == NULL)
    {

```

```

        printf("\n Error : List empty no node to update");
        return;
    }
    while (temp2 != NULL)
    {
        if (temp2->n == data)
        {

            temp2->n = data1;
            traversebeg();
            return;
        }
        else
            temp2 = temp2->next;
    }

    printf("\n Error : %d not found in list to update", data);
}

/* To sort the Linked List */
void sort()
{
    int i, j, x;

    temp2 = h;
    temp4 = h;

    if (temp2 == NULL)
    {
        printf("\n List empty to sort");
        return;
    }

    for (temp2 = h; temp2 != NULL; temp2 = temp2->next)
    {
        for (temp4 = temp2->next; temp4 != NULL; temp4 = temp4->next)
        {
            if (temp2->n > temp4->n)
            {
                x = temp2->n;
                temp2->n = temp4->n;
                temp4->n = x;
            }
        }
    }
}

```

```

        }
    }
    traversebeg();
}

1. /*
2.  * C Program to Implement Binary Tree using Linked List
3. */
4. #include <stdio.h>
5. #include <malloc.h>
6.
7. struct node {
8.     struct node * left;
9.     char data;
10.    struct node * right;
11.};
12.
13. struct node *constructTree( int );
14. void inorder(struct node *);
15.
16. char array[ ] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', '\0', '\0', 'H' };
17. int leftcount[ ] = { 1, 3, 5, -1, 9, -1, -1, -1, -1, -1 };
18. int rightcount[ ] = { 2, 4, 6, -1, -1, -1, -1, -1, -1, -1 };
19.
20. void main() {
21.     struct node *root;
22.     root = constructTree( 0 );
23.     printf("In-order Traversal: \n");
24.     inorder(root);
25. }
26.
27. struct node * constructTree( int index ) {
28.     struct node *temp = NULL;
29.     if (index != -1) {
30.         temp = (struct node *)malloc( sizeof( struct node ) );
31.         temp->left = constructTree( leftcount[index] );
32.         temp->data = array[index];
33.         temp->right = constructTree( rightcount[index] );
34.     }
35.     return temp;
36. }
37.

```

```
38. void inorder( struct node *root ) {
39.     if (root != NULL) {
40.         inorder(root->left);
41.         printf("%c\t", root->data);
42.         inorder(root->right);
43.     }
44. }

45. /*
46. * C Program to Implement Circular Doubly Linked List
47. */
48. #include <stdio.h>
49. #include <stdlib.h>
50.
51. struct node
52. {
53.     int val;
54.     struct node *next;
55.     struct node *prev;
56. };
57. typedef struct node n;
58.
59. n* create_node(int);
60. void add_node();
61. void insert_at_first();
62. void insert_at_end();
63. void insert_at_position();
64. void delete_node_position();
65. void sort_list();
66. void update();
67. void search();
68. void display_from_beg();
69. void display_in_rev();
70.
71. n *new, *ptr, *prev;
72. n *first = NULL, *last = NULL;
73. int number = 0;
74.
75. void main()
76. {
77.     int ch;
78.
79.     printf("\n linked list\n");
```

```
80.     printf("1.insert at beginning \n 2.insert at end\n 3.insert at
position\n4.sort linked list\n      5.delete node at position\n
6.updatenodevalue\n7.search element \n8.displaylist from beginning\n9.display
list from end\n10.exit ");
81.
82.     while (1)
83.     {
84.
85.         printf("\n enter your choice:");
86.         scanf("%d", &ch);
87.         switch (ch)
88.         {
89.             case 1 :
90.                 insert_at_first();
91.                 break;
92.             case 2 :
93.                 insert_at_end();
94.                 break;
95.             case 3 :
96.                 insert_at_position();
97.                 break;
98.             case 4 :
99.                 sort_list();
100.                break;
101.            case 5 :
102.                delete_node_position();
103.                break;
104.            case 6 :
105.                update();
106.                break;
107.            case 7 :
108.                search();
109.                break;
110.            case 8 :
111.                display_from_beg();
112.                break;
113.            case 9 :
114.                display_in_rev();
115.                break;
116.            case 10 :
117.                exit(0);
118.            case 11 :
119.                add_node();
```

```
120.         break;
121.     default:
122.         printf("\ninvalid choice");
123.     }
124. }
125. */
126. /*MEMORY ALLOCATED FOR NODE DYNAMICALLY
127. */
128. */
129. n* create_node(int info)
130. {
131.     number++;
132.     new = (n *)malloc(sizeof(n));
133.     new->val = info;
134.     new->next = NULL;
135.     new->prev = NULL;
136.     return new;
137. }
138. /*
139. *ADDS NEW NODE
140. */
141. void add_node()
142. {
143.
144.     int info;
145.
146.     printf("\nEnter the value you would like to add:");
147.     scanf("%d", &info);
148.     new = create_node(info);
149.
150.     if (first == last && first == NULL)
151.     {
152.
153.         first = last = new;
154.         first->next = last->next = NULL;
155.         first->prev = last->prev = NULL;
156.     }
157.     else
158.     {
159.         last->next = new;
160.         new->prev = last;
161.         last = new;
162.         last->next = first;
```

```
163.         first->prev = last;
164.     }
165. }
166. /*
167. *INSERTS ELEMENT AT FIRST
168. */
169. void insert_at_first()
170. {
171.
172.     int info;
173.
174.     printf("\nEnter the value to be inserted at first:");
175.     scanf("%d", &info);
176.     new = create_node(info);
177.
178.     if (first == last && first == NULL)
179.     {
180.         printf("\nInitially it is empty linked list later insertion is
done");
181.         first = last = new;
182.         first->next = last->next = NULL;
183.         first->prev = last->prev = NULL;
184.     }
185.     else
186.     {
187.         new->next = first;
188.         first->prev = new;
189.         first = new;
190.         first->prev = last;
191.         last->next = first;
192.         printf("\n the value is inserted at begining");
193.     }
194. }
195. /*
196. *INSERTS ELEMENT AT END
197. */
198. void insert_at_end()
199. {
200.
201.     int info;
202.
203.     printf("\nEnter the value that has to be inserted at last:");
204.     scanf("%d", &info);
```

```

205.     new = create_node(info);
206.
207.     if (first == last && first == NULL)
208.     {
209.         printf("\ninitially the list is empty and now new node is inserted
210. but at first");
211.         first = last = new;
212.         first->next = last->next = NULL;
213.         first->prev = last->prev = NULL;
214.     }
215.     else
216.     {
217.         last->next = new;
218.         new->prev = last;
219.         last = new;
220.         first->prev = last;
221.         last->next = first;
222.     }
223. /*
224. *INSERTS THE ELEMENT AT GIVEN POSITION
225. */
226. void insert_at_position()
227. {
228.     int info, pos, len = 0, i;
229.     n *prevnode;
230.
231.     printf("\n enter the value that you would like to insert:");
232.     scanf("%d", &info);
233.     printf("\n enter the position where you have to enter:");
234.     scanf("%d", &pos);
235.     new = create_node(info);
236.
237.     if (first == last && first == NULL)
238.     {
239.         if (pos == 1)
240.         {
241.             first = last = new;
242.             first->next = last->next = NULL;
243.             first->prev = last->prev = NULL;
244.         }
245.     }

```

```

246.             printf("\n empty linked list you cant insert at that particular
   position");
247.         }
248.     else
249.     {
250.         if (number < pos)
251.             printf("\n node cant be inserted as position is exceeding the
   linkedlist length");
252.
253.         else
254.         {
255.             for (ptr = first, i = 1;i <= number;i++)
256.             {
257.                 prevnode = ptr;
258.                 ptr = ptr->next;
259.                 if (i == pos-1)
260.                 {
261.                     prevnode->next = new;
262.                     new->prev = prevnode;
263.                     new->next = ptr;
264.                     ptr->prev = new;
265.                     printf("\ninserted at position %d successfully", pos);
266.                     break;
267.                 }
268.             }
269.         }
270.     }
271. }
272. /*
273. *SORTING IS DONE OF ONLY NUMBERS NOT LINKS
274. */
275. void sort_list()
276. {
277.     n *temp;
278.     int tempval, i, j;
279.
280.     if (first == last && first == NULL)
281.         printf("\nlinked list is empty no elements to sort");
282.     else
283.     {
284.         for (ptr = first,i = 0;i < number;ptr = ptr->next,i++)
285.         {
286.             for (temp = ptr->next,j=i;j<number;j++)

```



```

329.             ptr->prev = prevnode->prev;
330.             first = ptr;
331.             printf("%d is deleted", prevnode->val);
332.             free(prevnode);
333.             break;
334.         }
335.     else if (i == pos - 1)
336.     {
337.         number--;
338.         prevnode->next = ptr->next;
339.         ptr->next->prev = prevnode;
340.         printf("%d is deleted", ptr->val);
341.         free(ptr);
342.         break;
343.     }
344. }
345. }
346. }
347. }
348. /*
349. *UPDATION IS DONE FRO GIVEN OLD VAL
350. */
351. void update()
352. {
353.     int oldval, newval, i, f = 0;
354.     printf("\n enter the value old value:");
355.     scanf("%d", &oldval);
356.     printf("\n enter the value new value:");
357.     scanf("%d", &newval);
358.     if (first == last && first == NULL)
359.         printf("\n list is empty no elemnts for updation");
360.     else
361.     {
362.         for (ptr = first, i = 0; i < number; ptr = ptr->next, i++)
363.         {
364.             if (ptr->val == oldval)
365.             {
366.                 ptr->val = newval;
367.                 printf("value is updated to %d", ptr->val);
368.                 f = 1;
369.             }
370.         }
371.     if (f == 0)

```

```

372.             printf("\n no such old value to be get updated");
373.         }
374.     }
375. /*
376. *SEARCHING USING SINGLE KEY
377. */
378. void search()
379. {
380.     int count = 0, key, i, f = 0;
381.
382.     printf("\nEnter the value to be searched:");
383.     scanf("%d", &key);
384.
385.     if (first == last && first == NULL)
386.         printf("\nlist is empty no elemnets in list to search");
387.     else
388.     {
389.         for (ptr = first,i = 0;i < number;i++,ptr = ptr->next)
390.         {
391.             count++;
392.             if (ptr->val == key)
393.             {
394.                 printf("\n the value is found at position at %d", count);
395.                 f = 1;
396.             }
397.         }
398.         if (f == 0)
399.             printf("\n the value is not found in linkedlist");
400.     }
401. }
402. /*
403. *DISPLAYING IN BEGINNING
404. */
405. void display_from_beg()
406. {
407.     int i;
408.     if (first == last && first == NULL)
409.         printf("\nlist is empty no elemnts to print");
410.     else
411.     {
412.         printf("\n%d number of nodes are there", number);
413.         for (ptr = first, i = 0;i < number;i++,ptr = ptr->next)
414.             printf("\n %d", ptr->val);

```

```

415.      }
416.  }
417. /*
418. * DISPLAYING IN REVERSE
419. */
420. void display_in_rev()
421. {
422.     int i;
423.     if (first == last && first == NULL)
424.         printf("\nlist is empty there are no elements");
425.     else
426.     {
427.         for (ptr = last, i = 0;i < number;i++,ptr = ptr->prev)
428.         {
429.             printf("\n%d", ptr->val);
430.         }
431.     }
432. }
```

```

linked list
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 30

struct emp_data
{
    int empno;
    char empName[MAX];
    char designation[MAX];
    struct emp_data *next;
};

/* ****
/* Function to insert a node at the front of the Linked List. */
/* front: front pointer, id: employee ID, name: employee name */
/* desg: Employee designation */
/* Returns the new front pointer. */
/* ****

struct emp_data *insert(struct emp_data *front, int id, char name[],
char desg[])
{
```

```

{
    struct emp_data *newnode;

    newnode = (struct emp_data*)malloc(sizeof(struct emp_data));

    if (newnode == NULL)
    {
        printf("\n Allocation failed \n");
        exit(2);
    }
    newnode->empno = id;
    strcpy(newnode->empName, name);
    strcpy(newnode->designation, desg);
    newnode->next = front;
    front = newnode;
    return(front);
}
/* End of insert() */

/* Function to display a node in a Linked List */
void printNode(struct emp_data *p)
{
    printf("\n Employee Details...\n");
    printf("\n Emp No      : %d", p->empno);
    printf("\n Name       : %s", p->empName);
    printf("\n Designation   : %s\n", p->designation);
    printf("-----\n");
}
/* End of printNode() */

/*
***** Function to delete a node based on employee number *****
/* front: front pointer, id: Key value */
/* Returns: the modified list. */
***** */

struct emp_data* deleteNode(struct emp_data *front, int id)
{
    struct emp_data *ptr;
    struct emp_data *bptr;

    if (front->empno == id)
    {
        ptr = front;

```

```

        printf("\n Node deleted:");
        printNode(front);
        front = front->next;
        free(ptr);
        return(front);
    }
    for (ptr = front->next, bptr = front; ptr != NULL; ptr = ptr->next,
bptr = bptr->next)
    {
        if (ptr->empno == id)
        {
            printf("\n Node deleted:");
            printNode(ptr);
            bptr->next = ptr->next;
            free(ptr);
            return(front);
        }
    }
    printf("\n Employee Number %d not found ", id);
    return(front);
}
/* End of deleteNode() */

/*
 * Function to search the nodes in a Linear fashion based emp ID
 * front: front pointer, key: key ID.
 */
void search(struct emp_data *front, int key)
{
    struct emp_data *ptr;

    for (ptr = front; ptr != NULL; ptr = ptr -> next)
    {
        if (ptr->empno == key)
        {
            printf("\n Key found:");
            printNode(ptr);
            return;
        }
    }
    printf("\n Employee Number %d not found ", key);
}
/* End of search() */

```

```
/* Function to display the Linked List */
void display(struct emp_data *front)
{
    struct emp_data *ptr;

    for (ptr = front; ptr != NULL; ptr = ptr->next)
    {
        printNode(ptr);
    }
}
/* End of display() */

/* Function to display the menu of operations on a Linked List */
void menu()
{
    printf("-----\n");
    printf("Press 1 to INSERT a node into the list      \n");
    printf("Press 2 to DELETE a node from the list       \n");
    printf("Press 3 to DISPLAY the list                  \n");
    printf("Press 4 to SEARCH the list                 \n");
    printf("Press 5 to EXIT                           \n");
    printf("-----\n");
}
/* End of menu() */

/* Function to select the option */
char option()
{
    char choice;

    printf("\n\n>> Enter your choice: ");
    switch(choice=getche())
    {
        case '1':
        case '2':
        case '3':
        case '4':
        case '5':  return(choice);
        default :  printf("\n Invalid choice.");
    }
    return choice;
}
```

```

/* End of option() */

/* The main() program begins */
void main()
{
    struct emp_data *linkList;
    char name[21], desig[51];
    char choice;
    int eno;

    linkList = NULL;
    printf("\n Welcome to demonstration of singly linked list \n");
    menu();
    do
    {
        /* choose oeration to be performed */
        choice = option();
        switch(choice)
        {
            case '1':
                printf("\n Enter the Employee Number : ");
                scanf("%d", &eno);
                printf("Enter the Employee name : ");
                fflush(stdin);
                gets(name);
                printf("Enter the Employee Designation : ");
                gets(desig);
                linkList = insert(linkList, eno, name, desig);
                break;
            case '2':
                printf("\n\n Enter the employee number to be deleted: ");
                scanf("%d", &eno);
                linkList = deleteNode(linkList, eno);
                break;
            case '3':
                if (linkList == NULL)
                {
                    printf("\n List empty.");
                    break;
                }
                display(linkList);
                break;
            case '4':

```

```

        printf("\n\n Enter the employee number to be searched: ");
        scanf("%d", &eno);
        search(linkList, eno);
        break;
    case '5': break;
}
} while (choice != '5');
}

```

```

1. /*
2. * C Program to Implement Singly Linked List using Dynamic Memory Allocation
3. */
4. #include <stdio.h>
5. #include <malloc.h>
6. #define ISEMPY printf("\nEMPTY LIST:");
7. /*
8. * Node Declaration
9. */
10. struct node
11. {
12.     int value;
13.     struct node *next;
14. };
15.
16. snode* create_node(int);
17. void insert_node_first();
18. void insert_node_last();
19. void insert_node_pos();
20. void sorted_ascend();
21. void delete_pos();
22. void search();
23. void update_val();
24. void display();
25. void rev_display(snode *);
26.
27. typedef struct node snode;
28. snode *newnode, *ptr, *prev, *temp;
29. snode *first = NULL, *last = NULL;
30.
31. /*
32. * Main :contains menu
33. */

```

```

34.
35. int main()
36. {
37.     int ch;
38.     char ans = 'Y';
39.
40.     while (ans == 'Y' || ans == 'y')
41.     {
42.         printf("\n-----\n");
43.         printf("Operations on singly linked list\n");
44.         printf("\n-----\n");
45.         printf("\n1.Insert node at first");
46.         printf("\n2.Insert node at last");
47.         printf("\n3.Insert node at position");
48.         printf("\n4.Sorted Linked List in Ascending Order");
49.         printf("\n5.Delete Node from any Position");
50.         printf("\n6.Update Node Value");
51.         printf("\n7.Search Element in the linked list");
52.         printf("\n8.Display List from Beginning to end");
53.         printf("\n9.Display List from end using Recursion");
54.         printf("\n10.Exit\n");
55.         printf("\n~~~~~\n");
56.         printf("\nEnter your choice");
57.         scanf("%d", &ch);
58.
59.         switch (ch)
60.         {
61.             case 1:
62.                 printf("\n...Inserting node at first...\n");
63.                 insert_node_first();
64.                 break;
65.             case 2:
66.                 printf("\n...Inserting node at last...\n");
67.                 insert_node_last();
68.                 break;
69.             case 3:
70.                 printf("\n...Inserting node at position...\n");
71.                 insert_node_pos();
72.                 break;
73.             case 4:
74.                 printf("\n...Sorted Linked List in Ascending Order...\n");
75.                 sorted_ascend();
76.                 break;

```

```

77.     case 5:
78.         printf("\n...Deleting Node from any Position...\n");
79.         delete_pos();
80.         break;
81.     case 6:
82.         printf("\n...Updating Node Value...\n");
83.         update_val();
84.         break;
85.     case 7:
86.         printf("\n...Searching Element in the List...\n");
87.         search();
88.         break;
89.     case 8:
90.         printf("\n...Displaying List From Beginning to End...\n");
91.         display();
92.         break;
93.     case 9:
94.         printf("\n...Displaying List From End using Recursion...\n");
95.         rev_display(first);
96.         break;
97.     case 10:
98.         printf("\n...Exiting...\n");
99.         return 0;
100.        break;
101.    default:
102.        printf("\n...Invalid Choice...\n");
103.        break;
104.    }
105.    printf("\nYOU WANT TO CONTINUE (Y/N)");
106.    scanf(" %c", &ans);
107. }
108. return 0;
109. }

110.

111. /*
112. * Creating Node
113. */
114. snode* create_node(int val)
115. {
116.     newnode = (snode *)malloc(sizeof(snode));
117.     if (newnode == NULL)
118.     {
119.         printf("\nMemory was not allocated");

```

```

120.         return 0;
121.     }
122.     else
123.     {
124.         newnode->value = val;
125.         newnode->next = NULL;
126.         return newnode;
127.     }
128. }
129.
130. /*
131. * Inserting Node at First
132. */
133. void insert_node_first()
134. {
135.     int val;
136.
137.     printf("\nEnter the value for the node:");
138.     scanf("%d", &val);
139.     newnode = create_node(val);
140.     if (first == last && first == NULL)
141.     {
142.         first = last = newnode;
143.         first->next = NULL;
144.         last->next = NULL;
145.     }
146.     else
147.     {
148.         temp = first;
149.         first = newnode;
150.         first->next = temp;
151.     }
152.     printf("\n----INSERTED----");
153. }
154.
155. /*
156. * Inserting Node at Last
157. */
158. void insert_node_last()
159. {
160.     int val;
161.
162.     printf("\nEnter the value for the Node:");

```

```

163.     scanf("%d", &val);
164.     newnode = create_node(val);
165.     if (first == last && last == NULL)
166.     {
167.         first = last = newnode;
168.         first->next = NULL;
169.         last->next = NULL;
170.     }
171.     else
172.     {
173.         last->next = newnode;
174.         last = newnode;
175.         last->next = NULL;
176.     }
177.     printf("\n----INSERTED---");
178. }
179.
180. /*
181. * Inserting Node at position
182. */
183. void insert_node_pos()
184. {
185.     int pos, val, cnt = 0, i;
186.
187.     printf("\nEnter the value for the Node:");
188.     scanf("%d", &val);
189.     newnode = create_node(val);
190.     printf("\nEnter the position ");
191.     scanf("%d", &pos);
192.     ptr = first;
193.     while (ptr != NULL)
194.     {
195.         ptr = ptr->next;
196.         cnt++;
197.     }
198.     if (pos == 1)
199.     {
200.         if (first == last && first == NULL)
201.         {
202.             first = last = newnode;
203.             first->next = NULL;
204.             last->next = NULL;
205.         }

```

```

206.         else
207.         {
208.             temp = first;
209.             first = newnode;
210.             first->next = temp;
211.         }
212.         printf("\nInserted");
213.     }
214.     else if (pos>1 && pos<=cnt)
215.     {
216.         ptr = first;
217.         for (i = 1;i < pos;i++)
218.         {
219.             prev = ptr;
220.             ptr = ptr->next;
221.         }
222.         prev->next = newnode;
223.         newnode->next = ptr;
224.         printf("\n----INSERTED----");
225.     }
226.     else
227.     {
228.         printf("Position is out of range");
229.     }
230. }
231.
232. /*
233. * Sorted Linked List
234. */
235. void sorted_ascend()
236. {
237.     snode *nxt;
238.     int t;
239.
240.     if (first == NULL)
241.     {
242.         ISEMPTY;
243.         printf(":No elements to sort\n");
244.     }
245.     else
246.     {
247.         for (ptr = first;ptr != NULL;ptr = ptr->next)
248.         {

```

```

249.         for (nxt = ptr->next;nxt != NULL;nxt = nxt->next)
250.         {
251.             if (ptr->value > nxt->value)
252.             {
253.                 t = ptr->value;
254.                 ptr->value = nxt->value;
255.                 nxt->value = t;
256.             }
257.         }
258.     }
259.     printf("\n---Sorted List---");
260.     for (ptr = first;ptr != NULL;ptr = ptr->next)
261.     {
262.         printf("%d\t", ptr->value);
263.     }
264. }
265. }
266.
267. /*
268. * Delete Node from specified position in a non-empty List
269. */
270. void delete_pos()
271. {
272.     int pos, cnt = 0, i;
273.
274.     if (first == NULL)
275.     {
276.         ISEMPY;
277.         printf(":No node to delete\n");
278.     }
279.     else
280.     {
281.         printf("\nEnter the position of value to be deleted:");
282.         scanf(" %d", &pos);
283.         ptr = first;
284.         if (pos == 1)
285.         {
286.             first = ptr->next;
287.             printf("\nElement deleted");
288.         }
289.         else
290.         {
291.             while (ptr != NULL)

```

```

292.         {
293.             ptr = ptr->next;
294.             cnt = cnt + 1;
295.         }
296.         if (pos > 0 && pos <= cnt)
297.         {
298.             ptr = first;
299.             for (i = 1;i < pos;i++)
300.             {
301.                 prev = ptr;
302.                 ptr = ptr->next;
303.             }
304.             prev->next = ptr->next;
305.         }
306.         else
307.         {
308.             printf("Position is out of range");
309.         }
310.         free(ptr);
311.         printf("\nElement deleted");
312.     }
313. }
314. /*
315. * Updating Node value in a non-empty list
316. */
317.
318. void update_val()
319. {
320.     int oldval, newval, flag = 0;
321.
322.     if (first == NULL)
323.     {
324.         ISEMPTY;
325.         printf(":No nodes in the list to update\n");
326.     }
327.     else
328.     {
329.         printf("\nEnter the value to be updated:");
330.         scanf("%d", &oldval);
331.         printf("\nEnter the newvalue:");
332.         scanf("%d", &newval);
333.         for (ptr = first;ptr != NULL;ptr = ptr->next)
334.     {

```

```

335.         if (ptr->value == oldval)
336.         {
337.             ptr->value = newval;
338.             flag = 1;
339.             break;
340.         }
341.     }
342.     if (flag == 1)
343.     {
344.         printf("\nUpdated Successfully");
345.     }
346.     else
347.     {
348.         printf("\nValue not found in List");
349.     }
350.     }
351. }
352.
353. /*
354. * searching an element in a non-empty list
355. */
356. void search()
357. {
358.     int flag = 0, key, pos = 0;
359.
360.     if (first == NULL)
361.     {
362.         ISEMPY;
363.         printf(":No nodes in the list\n");
364.     }
365.     else
366.     {
367.         printf("\nEnter the value to search");
368.         scanf("%d", &key);
369.         for (ptr = first;ptr != NULL;ptr = ptr->next)
370.         {
371.             pos = pos + 1;
372.             if (ptr->value == key)
373.             {
374.                 flag = 1;
375.                 break;
376.             }
377.         }

```

```
378.         if (flag == 1)
379.         {
380.             printf("\nElement %d found at %d position\n", key, pos);
381.         }
382.         else
383.         {
384.             printf("\nElement %d not found in list\n", key);
385.         }
386.     }
387. }
388. /*
389. * Displays non-empty List from Beginning to End
390. */
391. void display()
392. {
393.     if (first == NULL)
394.     {
395.         ISEMPY;
396.         printf(":No nodes in the list to display\n");
397.     }
398.     else
399.     {
400.         for (ptr = first;ptr != NULL;ptr = ptr->next)
401.         {
402.             printf("%d\t", ptr->value);
403.         }
404.     }
405. }
406.
407. /*
408. * Display non-empty List in Reverse Order
409. */
410. void rev_display(snode *ptr)
411. {
412.     int val;
413.
414.     if (ptr == NULL)
415.     {
416.         ISEMPY;
417.         printf(":No nodes to display\n");
418.     }
419.     else
420.     {
```

```
421.         if (ptr != NULL)
422.         {
423.             val = ptr->value;
424.             rev_display(ptr->next);
425.             printf("%d\t", val);
426.         }
427.
428.     }
429. }
430. /*
431. * C Program to Implement Singly Linked List using Dynamic Memory Allocation
432. */
433. #include <stdio.h>
434. #include <malloc.h>
435. #define ISEMPY printf("\nEMPTY LIST:");
436. /*
437. * Node Declaration
438. */
439. struct node
440. {
441.     int value;
442.     struct node *next;
443. };
444.
445. snode* create_node(int);
446. void insert_node_first();
447. void insert_node_last();
448. void insert_node_pos();
449. void sorted_ascend();
450. void delete_pos();
451. void search();
452. void update_val();
453. void display();
454. void rev_display(snode *);
455.
456. typedef struct node snode;
457. snode *newnode, *ptr, *prev, *temp;
458. snode *first = NULL, *last = NULL;
459.
460. /*
461. * Main :contains menu
462. */
463.
```

```

464. int main()
465. {
466.     int ch;
467.     char ans = 'Y';
468.
469.     while (ans == 'Y' || ans == 'y')
470.     {
471.         printf("\n-----\n");
472.         printf("\nOperations on singly linked list\n");
473.         printf("\n-----\n");
474.         printf("\n1.Insert node at first");
475.         printf("\n2.Insert node at last");
476.         printf("\n3.Insert node at position");
477.         printf("\n4.Sorted Linked List in Ascending Order");
478.         printf("\n5.Delete Node from any Position");
479.         printf("\n6.Update Node Value");
480.         printf("\n7.Search Element in the linked list");
481.         printf("\n8.Display List from Beginning to end");
482.         printf("\n9.Display List from end using Recursion");
483.         printf("\n10.Exit\n");
484.         printf("\n~~~~~\n");
485.         printf("\nEnter your choice");
486.         scanf("%d", &ch);
487.
488.         switch (ch)
489.         {
490.             case 1:
491.                 printf("\n...Inserting node at first...\n");
492.                 insert_node_first();
493.                 break;
494.             case 2:
495.                 printf("\n...Inserting node at last...\n");
496.                 insert_node_last();
497.                 break;
498.             case 3:
499.                 printf("\n...Inserting node at position...\n");
500.                 insert_node_pos();
501.                 break;
502.             case 4:
503.                 printf("\n...Sorted Linked List in Ascending Order...\n");
504.                 sorted_ascend();
505.                 break;
506.             case 5:

```

```

507.         printf("\n...Deleting Node from any Position...\n");
508.         delete_pos();
509.         break;
510.     case 6:
511.         printf("\n...Updating Node Value...\n");
512.         update_val();
513.         break;
514.     case 7:
515.         printf("\n...Searching Element in the List...\n");
516.         search();
517.         break;
518.     case 8:
519.         printf("\n...Displaying List From Beginning to End...\n");
520.         display();
521.         break;
522.     case 9:
523.         printf("\n...Displaying List From End using Recursion...\n");
524.         rev_display(first);
525.         break;
526.     case 10:
527.         printf("\n...Exiting...\n");
528.         return 0;
529.         break;
530.     default:
531.         printf("\n...Invalid Choice...\n");
532.         break;
533.     }
534.     printf("\nYOU WANT TO CONTINUE (Y/N)");
535.     scanf(" %c", &ans);
536. }
537. return 0;
538. }
539.
540. /*
541. * Creating Node
542. */
543. snode* create_node(int val)
544. {
545.     newnode = (snode *)malloc(sizeof(snode));
546.     if (newnode == NULL)
547.     {
548.         printf("\nMemory was not allocated");
549.         return 0;

```

```
550.     }
551.     else
552.     {
553.         newnode->value = val;
554.         newnode->next = NULL;
555.         return newnode;
556.     }
557. }
558.
559. /*
560. * Inserting Node at First
561. */
562. void insert_node_first()
563. {
564.     int val;
565.
566.     printf("\nEnter the value for the node:");
567.     scanf("%d", &val);
568.     newnode = create_node(val);
569.     if (first == last && first == NULL)
570.     {
571.         first = last = newnode;
572.         first->next = NULL;
573.         last->next = NULL;
574.     }
575.     else
576.     {
577.         temp = first;
578.         first = newnode;
579.         first->next = temp;
580.     }
581.     printf("\n----INSERTED----");
582. }
583.
584. /*
585. * Inserting Node at Last
586. */
587. void insert_node_last()
588. {
589.     int val;
590.
591.     printf("\nEnter the value for the Node:");
592.     scanf("%d", &val);
```

```

593.     newnode = create_node(val);
594.     if (first == last && last == NULL)
595.     {
596.         first = last = newnode;
597.         first->next = NULL;
598.         last->next = NULL;
599.     }
600.     else
601.     {
602.         last->next = newnode;
603.         last = newnode;
604.         last->next = NULL;
605.     }
606.     printf("\n----INSERTED---");
607. }
608.
609. /*
610. * Inserting Node at position
611. */
612. void insert_node_pos()
613. {
614.     int pos, val, cnt = 0, i;
615.
616.     printf("\nEnter the value for the Node:");
617.     scanf("%d", &val);
618.     newnode = create_node(val);
619.     printf("\nEnter the position ");
620.     scanf("%d", &pos);
621.     ptr = first;
622.     while (ptr != NULL)
623.     {
624.         ptr = ptr->next;
625.         cnt++;
626.     }
627.     if (pos == 1)
628.     {
629.         if (first == last && first == NULL)
630.         {
631.             first = last = newnode;
632.             first->next = NULL;
633.             last->next = NULL;
634.         }
635.     else

```

```

636.         {
637.             temp = first;
638.             first = newnode;
639.             first->next = temp;
640.         }
641.         printf("\nInserted");
642.     }
643.     else if (pos>1 && pos<=cnt)
644.     {
645.         ptr = first;
646.         for (i = 1;i < pos;i++)
647.         {
648.             prev = ptr;
649.             ptr = ptr->next;
650.         }
651.         prev->next = newnode;
652.         newnode->next = ptr;
653.         printf("\n----INSERTED----");
654.     }
655.     else
656.     {
657.         printf("Position is out of range");
658.     }
659. }
660.
661. /*
662. * Sorted Linked List
663. */
664. void sorted_ascend()
665. {
666.     snode *nxt;
667.     int t;
668.
669.     if (first == NULL)
670.     {
671.         ISEMPY;
672.         printf(":No elements to sort\n");
673.     }
674.     else
675.     {
676.         for (ptr = first;ptr != NULL;ptr = ptr->next)
677.         {
678.             for (nxt = ptr->next;nxt != NULL;nxt = nxt->next)

```

```

679.         {
680.             if (ptr->value > nxt->value)
681.             {
682.                 t = ptr->value;
683.                 ptr->value = nxt->value;
684.                 nxt->value = t;
685.             }
686.         }
687.     }
688.     printf("\n---Sorted List---");
689.     for (ptr = first;ptr != NULL;ptr = ptr->next)
690.     {
691.         printf("%d\t", ptr->value);
692.     }
693. }
694. }
695.
696. /*
697. * Delete Node from specified position in a non-empty List
698. */
699. void delete_pos()
700. {
701.     int pos, cnt = 0, i;
702.
703.     if (first == NULL)
704.     {
705.         ISEMPY;
706.         printf(":No node to delete\n");
707.     }
708.     else
709.     {
710.         printf("\nEnter the position of value to be deleted:");
711.         scanf(" %d", &pos);
712.         ptr = first;
713.         if (pos == 1)
714.         {
715.             first = ptr->next;
716.             printf("\nElement deleted");
717.         }
718.         else
719.         {
720.             while (ptr != NULL)
721.             {

```

```

722.                 ptr = ptr->next;
723.                 cnt = cnt + 1;
724.             }
725.             if (pos > 0 && pos <= cnt)
726.             {
727.                 ptr = first;
728.                 for (i = 1;i < pos;i++)
729.                 {
730.                     prev = ptr;
731.                     ptr = ptr->next;
732.                 }
733.                 prev->next = ptr->next;
734.             }
735.             else
736.             {
737.                 printf("Position is out of range");
738.             }
739.             free(ptr);
740.             printf("\nElement deleted");
741.         }
742.     }
743. }
744. /*
745. * Updating Node value in a non-empty list
746. */
747. void update_val()
748. {
749.     int oldval, newval, flag = 0;
750.
751.     if (first == NULL)
752.     {
753.         ISEMPTY;
754.         printf(":No nodes in the list to update\n");
755.     }
756.     else
757.     {
758.         printf("\nEnter the value to be updated:");
759.         scanf("%d", &oldval);
760.         printf("\nEnter the newvalue:");
761.         scanf("%d", &newval);
762.         for (ptr = first;ptr != NULL;ptr = ptr->next)
763.         {
764.             if (ptr->value == oldval)

```

```
765.         {
766.             ptr->value = newval;
767.             flag = 1;
768.             break;
769.         }
770.     }
771.     if (flag == 1)
772.     {
773.         printf("\nUpdated Successfully");
774.     }
775.     else
776.     {
777.         printf("\nValue not found in List");
778.     }
779. }
780. }
781.
782. /*
783. * searching an element in a non-empty list
784. */
785. void search()
786. {
787.     int flag = 0, key, pos = 0;
788.
789.     if (first == NULL)
790.     {
791.         ISEMPY;
792.         printf(":No nodes in the list\n");
793.     }
794.     else
795.     {
796.         printf("\nEnter the value to search");
797.         scanf("%d", &key);
798.         for (ptr = first; ptr != NULL; ptr = ptr->next)
799.         {
800.             pos = pos + 1;
801.             if (ptr->value == key)
802.             {
803.                 flag = 1;
804.                 break;
805.             }
806.         }
807.         if (flag == 1)
```

```
808.         {
809.             printf("\nElement %d found at %d position\n", key, pos);
810.         }
811.     else
812.     {
813.         printf("\nElement %d not found in list\n", key);
814.     }
815. }
816. /*
817. * Displays non-empty List from Beginning to End
818. */
819. void display()
820. {
821.     if (first == NULL)
822.     {
823.         ISEMPY;
824.         printf(":No nodes in the list to display\n");
825.     }
826.     else
827.     {
828.         for (ptr = first;ptr != NULL;ptr = ptr->next)
829.         {
830.             printf("%d\t", ptr->value);
831.         }
832.     }
833. }
834. /*
835. */
836. /*
837. * Display non-empty list in Reverse Order
838. */
839. void rev_display(snode *ptr)
840. {
841.     int val;
842.
843.     if (ptr == NULL)
844.     {
845.         ISEMPY;
846.         printf(":No nodes to display\n");
847.     }
848.     else
849.     {
850.         if (ptr != NULL)
```

```

851.         {
852.             val = ptr->value;
853.             rev_display(ptr->next);
854.             printf("%d\t", val);
855.         }
856.
857.     }
858. }
```

```

1. /*
2. * C Program to Implement Doubly Linked List using Singly Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11. };
12.
13. void create(struct node **);
14. void move (struct node *);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL, *q = NULL;
21.     int result, count;
22.
23.     printf("Enter data into the list\n");
24.     create(&p);
25.     printf("Displaying list:\n");
26.     display(p);
27.     move(p);
28.     release (&p);
29.
30.     return 0;
31. }
32.
33. void move(struct node *head)
```

```

34. {
35.     struct node *p, *q;
36.     int ch;
37.
38.     p = q = head;
39.     printf("\nPointer at %d\n", head->num);
40.     do
41.     {
42.         printf("Select option:\n1. Move front\n2. Move back\n3. Exit\nYour choice:");
43.         scanf("%d", &ch);
44.         switch(ch)
45.         {
46.             case 1: if(q->next != NULL)
47.                         {
48.                             q = q->next;
49.                             printf("\nPointer at %d\n", q->num);
50.                         }
51.                         else
52.                         {
53.                             printf("\nPointer at last node %d. Cannot move ahead.\n", q->num);
54.                         }
55.                         break;
56.             case 2: while (p->next != q)
57.                         {
58.                             p = p->next;
59.                         }
60.                         if (p == q)
61.                         {
62.                             printf("\nPointer at first node %d. Cannot move behind.\n", q->num);
63.                         }
64.                         else
65.                         {
66.                             q = p;
67.                             p = head;
68.                             printf("\nPointer at %d\n", q->num);
69.                         }
70.                         break;
71.             case 3: return;
72.             default: printf("\nInvalid choice entered. Try again\n");
73.         }
    
```

```
74.     } while (1);
75. }
76.
77. void create(struct node **head)
78. {
79.     int c, ch;
80.     struct node *temp, *rear;
81.
82.     do
83.     {
84.         printf("Enter number: ");
85.         scanf("%d", &c);
86.         temp = (struct node *)malloc(sizeof(struct node));
87.         temp->num = c;
88.         temp->next = NULL;
89.         if (*head == NULL)
90.         {
91.             *head = temp;
92.         }
93.         else
94.         {
95.             rear->next = temp;
96.         }
97.         rear = temp;
98.         printf("Do you wish to continue [1/0]: ");
99.         scanf("%d", &ch);
100.        } while (ch != 0);
101.        printf("\n");
102.    }
103.
104.    void display(struct node *head)
105.    {
106.        while (head != NULL)
107.        {
108.            printf("%d\t", head->num);
109.            head = head->next;
110.        }
111.        printf("\n");
112.    }
113.
114.    void release(struct node **head)
115.    {
116.        struct node *temp;
```

```

117.         while ((*head) != NULL)
118.         {
119.             temp = *head;
120.             (*head) = (*head)->next;
121.             free(temp);
122.         }
123.     }

```

```

1. /*
2.  * C Program to Demonstrate Circular Single Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int data;
10.    struct node *link;
11.};
12.
13. struct node *head = NULL, *x, *y, *z;
14.
15. void create();
16. void ins_at_beg();
17. void ins_at_pos();
18. void del_at_beg();
19. void del_at_pos();
20. void traverse();
21. void search();
22. void sort();
23. void update();
24. void rev_traverse(struct node *p);
25.
26. void main()
27. {
28.     int ch;
29.
30.     printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at remaining");
31.     printf("\n4.Deletion at beginning \n5.Deletion at remaining \n6.traverse");
32.     printf("\n7.Search\n8.sort\n9.update\n10.Exit\n");

```

```
33.     while (1)
34.     {
35.         printf("\n Enter your choice:");
36.         scanf("%d", &ch);
37.         switch(ch)
38.         {
39.             case 1:
40.                 create();
41.                 break;
42.             case 2:
43.                 ins_at_beg();
44.                 break;
45.             case 3:
46.                 ins_at_pos();
47.                 break;
48.             case 4:
49.                 del_at_beg();
50.                 break;
51.             case 5:
52.                 del_at_pos();
53.                 break;
54.             case 6:
55.                 traverse();
56.                 break;
57.             case 7:
58.                 search();
59.                 break;
60.             case 8:
61.                 sort();
62.                 break;
63.             case 9:
64.                 update();
65.                 break;
66.             case 10:
67.                 rev_traverse(head);
68.                 break;
69.             default:
70.                 exit(0);
71.         }
72.     }
73. }
74.
75. /*Function to create a new circular Linked List*/
```

```

76. void create()
77. {
78.     int c;
79.
80.     x = (struct node*)malloc(sizeof(struct node));
81.     printf("\n Enter the data:");
82.     scanf("%d", &x->data);
83.     x->link = x;
84.     head = x;
85.     printf("\n If you wish to continue press 1 otherwise 0:");
86.     scanf("%d", &c);
87.     while (c != 0)
88.     {
89.         y = (struct node*)malloc(sizeof(struct node));
90.         printf("\n Enter the data:");
91.         scanf("%d", &y->data);
92.         x->link = y;
93.         y->link = head;
94.         x = y;
95.         printf("\n If you wish to continue press 1 otherwise 0:");
96.         scanf("%d", &c);
97.     }
98. }
99.
100. /*Function to insert an element at the begining of the list*/
101.
102. void ins_at_beg()
103. {
104.     x = head;
105.     y = (struct node*)malloc(sizeof(struct node));
106.     printf("\n Enter the data:");
107.     scanf("%d", &y->data);
108.     while (x->link != head)
109.     {
110.         x = x->link;
111.     }
112.     x->link = y;
113.     y->link = head;
114.     head = y;
115. }
116.
117. /*Function to insert an element at any position the list*/
118.

```

```

119.     void ins_at_pos()
120.     {
121.         struct node *ptr;
122.         int c = 1, pos, count = 1;
123.
124.         y = (struct node*)malloc(sizeof(struct node));
125.         if (head == NULL)
126.         {
127.             printf("cannot enter an element at this place");
128.         }
129.         printf("\n Enter the data:");
130.         scanf("%d", &y->data);
131.         printf("\n Enter the position to be inserted:");
132.         scanf("%d", &pos);
133.         x = head;
134.         ptr = head;
135.         while (ptr->link != head)
136.         {
137.             count++;
138.             ptr = ptr->link;
139.         }
140.         count++;
141.         if (pos > count)
142.         {
143.             printf("OUT OF BOUND");
144.             return;
145.         }
146.         while (c < pos)
147.         {
148.             z = x;
149.             x = x->link;
150.             c++;
151.         }
152.         y->link = x;
153.         z->link = y;
154.     }
155.
156.     /*Function to delete an element at any begining of the list*/
157.
158.     void del_at_beg()
159.     {
160.         if (head == NULL)
161.             printf("\n List is empty");

```

```

162.         else
163.         {
164.             x = head;
165.             y = head;
166.             while (x->link != head)
167.             {
168.                 x = x->link;
169.             }
170.             head = y->link;
171.             x->link = head;
172.             free(y);
173.         }
174.     }
175.
176. /*Function to delete an element at any position the List*/
177.
178. void del_at_pos()
179. {
180.     if (head == NULL)
181.         printf("\n List is empty");
182.     else
183.     {
184.         int c = 1, pos;
185.         printf("\n Enter the position to be deleted:");
186.         scanf("%d", &pos);
187.         x = head;
188.         while (c < pos)
189.         {
190.             y = x;
191.             x = x->link;
192.             c++;
193.         }
194.         y->link = x->link;
195.         free(x);
196.     }
197. }
198.
199. /*Function to display the elements in the List*/
200.
201. void traverse()
202. {
203.     if (head == NULL)
204.         printf("\n List is empty");

```

```

205.     else
206.     {
207.         x = head;
208.         while (x->link != head)
209.         {
210.             printf("%d->", x->data);
211.             x = x->link;
212.         }
213.         printf("%d", x->data);
214.     }
215. }
216.
217. /*Function to search an element in the list*/
218.
219. void search()
220. {
221.     int search_val, count = 0, flag = 0;
222.     printf("\nEnter the element to search\n");
223.     scanf("%d", &search_val);
224.     if (head == NULL)
225.         printf("\nList is empty nothing to search");
226.     else
227.     {
228.         x = head;
229.         while (x->link != head)
230.         {
231.             if (x->data == search_val)
232.             {
233.                 printf("\nthe element is found at %d", count);
234.                 flag = 1;
235.                 break;
236.             }
237.             count++;
238.             x = x->link;
239.         }
240.         if (x->data == search_val)
241.         {
242.             printf("element found at position %d", count);
243.         }
244.         if (flag == 0)
245.         {
246.             printf("\nelement not found");
247.         }

```

```
248.
249.         }
250.     }
251.
252.     /*Function to sort the list in ascending order*/
253.
254.     void sort()
255.     {
256.         struct node *ptr, *nxt;
257.         int temp;
258.
259.         if (head == NULL)
260.         {
261.             printf("empty linkedlist");
262.         }
263.         else
264.         {
265.             ptr = head;
266.             while (ptr->link != head)
267.             {
268.                 nxt = ptr->link;
269.                 while (nxt != head)
270.                 {
271.                     if (nxt != head)
272.                     {
273.                         if (ptr->data > nxt->data)
274.                         {
275.                             temp = ptr->data;
276.                             ptr->data = nxt->data;
277.                             nxt->data = temp;
278.                         }
279.                     }
280.                     else
281.                     {
282.                         break;
283.                     }
284.                     nxt = nxt->link;
285.                 }
286.                 ptr = ptr->link;
287.             }
288.         }
289.     }
```

```
291.     /*Function to update an element at any position the list*/
292.     void update()
293.     {
294.         struct node *ptr;
295.         int search_val;
296.         int replace_val;
297.         int flag = 0;
298.
299.         if (head == NULL)
300.         {
301.             printf("\n empty list");
302.         }
303.         else
304.         {
305.             printf("enter the value to be edited\n");
306.             scanf("%d", &search_val);
307.             fflush(stdin);
308.             printf("enter the value to be replace\n");
309.             scanf("%d", &replace_val);
310.             ptr = head;
311.             while (ptr->link != head)
312.             {
313.                 if (ptr->data == search_val)
314.                 {
315.                     ptr->data = replace_val;
316.                     flag = 1;
317.                     break;
318.                 }
319.                 ptr = ptr->link;
320.             }
321.             if (ptr->data == search_val)
322.             {
323.                 ptr->data = replace_val;
324.                 flag = 1;
325.             }
326.             if (flag == 1)
327.             {
328.                 printf("\nUpdate sucessful");
329.             }
330.             else
331.             {
332.                 printf("\n update not successful");
333.             }
```

```

334.         }
335.     }
336.
337.     /*Function to display the elements of the list in reverse order*/
338.
339.     void rev_traverse(struct node *p)
340.     {
341.         int i = 0;
342.
343.         if (head == NULL)
344.         {
345.             printf("empty linked list");
346.         }
347.         else
348.         {
349.             if (p->link != head)
350.             {
351.                 i = p->data;
352.                 rev_traverse(p->link);
353.                 printf(" %d", i);
354.             }
355.             if (p->link == head)
356.             {
357.                 printf(" %d", p->data);
358.             }
359.         }
360.     }

```

## 5. C Examples dealing with the Operations on the elements of a Linked List

```

1. /*
2.  * C Program to Add Corresponding Positioned Elements of 2 Linked Lists
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <ctype.h>
7.
8. struct node
9. {
10.     int num;
11.     struct node *next;
12. };

```

```

13.
14. int feednumber(struct node **);
15. struct node *addlist(struct node *, struct node *, int, int);
16. void release(struct node **);
17. void display(struct node *);
18.
19. int main()
20. {
21.     struct node *p = NULL;
22.     struct node *q = NULL;
23.     struct node *res = NULL;
24.     int pcount = 0, qcount = 0;
25.
26.     printf("Enter first number\n");
27.     pcount = feednumber(&p);
28.     printf("Enter second number\n");
29.     qcount = feednumber(&q);
30.     printf("Displaying list1: ");
31.     display(p);
32.     printf("Displaying list2: ");
33.     display(q);
34.     res = addlist(p, q, pcount, qcount);
35.     printf("Displaying the resulting list: ");
36.     display(res);
37.     release(&p);
38.     release(&q);
39.     release(&res);
40.
41.     return 0;
42. }
43.
44. /*Function to create nodes of numbers*/
45. int feednumber(struct node **head)
46. {
47.     char ch, dig;
48.     int count = 0;
49.     struct node *temp, *rear = NULL;
50.
51.     ch = getchar();
52.     while (ch != '\n')
53.     {
54.         dig = atoi(&ch);
55.         temp = (struct node *)malloc(sizeof(struct node));

```

```
56.     temp->num = dig;
57.     temp->next = NULL;
58.     count++;
59.     if ((*head) == NULL)
60.     {
61.         *head = temp;
62.         rear = temp;
63.     }
64.     else
65.     {
66.         rear->next = temp;
67.         rear = rear->next;
68.     }
69.     ch = getchar();
70. }
71.
72. return count;
73. }
74.
75. /*Function to display the list of numbers*/
76. void display (struct node *head)
77. {
78.     while (head != NULL)
79.     {
80.         printf("%d", head->num);
81.         head = head->next;
82.     }
83.     printf("\n");
84. }
85.
86. /*Function to free the allocated list of numbers*/
87. void release (struct node **head)
88. {
89.     struct node *temp = *head;
90.
91.     while ((*head) != NULL)
92.     {
93.         (*head) = (*head)->next;
94.         free(temp);
95.         temp = *head;
96.     }
97. }
98.
```

```

99. /*Function to add the list of numbers and store them in 3rd List*/
100.     struct node *addlist(struct node *p, struct node *q, int pcount, int qcount)
101.     {
102.         struct node *ptemp, *qtemp, *result = NULL, *temp;
103.         int i, carry = 0;
104.
105.         while (pcount != 0 && qcount != 0)
106.         {
107.             ptemp = p;
108.             qtemp = q;
109.             for (i = 0; i < pcount - 1; i++)
110.             {
111.                 ptemp = ptemp->next;
112.             }
113.             for (i = 0; i < qcount - 1; i++)
114.             {
115.                 qtemp = qtemp->next;
116.             }
117.             temp = (struct node *) malloc (sizeof(struct node));
118.             temp->num = ptemp->num + qtemp->num + carry;
119.             carry = temp->num / 10;
120.             temp->num = temp->num % 10;
121.             temp->next = result;
122.             result = temp;
123.             pcount--;
124.             qcount--;
125.         }
126.         /*both or one of the 2 lists have been read completely by now*/
127.         while (pcount != 0)
128.         {
129.             ptemp = p;
130.             for (i = 0; i < pcount - 1; i++)
131.             {
132.                 ptemp = ptemp->next;
133.             }
134.             temp = (struct node *) malloc (sizeof(struct node));
135.             temp->num = ptemp->num + carry;
136.             carry = temp->num / 10;
137.             temp->num = temp->num % 10;
138.             temp->next = result;
139.             result = temp;
140.             pcount--;
141.         }

```

```

142.         while (qcount != 0)
143.         {
144.             qtemp = q;
145.             for (i = 0; i < qcount - 1; i++)
146.             {
147.                 qtemp = qtemp->next;
148.             }
149.             temp = (struct node *) malloc (sizeof(struct node));
150.             temp->num = qtemp->num + carry;
151.             carry = temp->num / 10;
152.             temp->num = temp->num % 10;
153.             temp->next = result;
154.             result = temp;
155.             qcount--;
156.         }
157.
158.         return result;
159.     }

```

```

1. /*
2.  * C Program to Check whether 2 Lists are Same
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void feedmember(struct node **);
14. int compare (struct node *, struct node *);
15. void release(struct node **);
16.
17. int main()
18. {
19.     struct node *p = NULL;
20.     struct node *q = NULL;
21.     int result;
22.
23.     printf("Enter data into first list\n");

```

```

24.     feedmember(&p);
25.     printf("Enter data into second list\n");
26.     feedmember(&q);
27.     result = compare(p, q);
28.     if (result == 1)
29.     {
30.         printf("The 2 list are equal.\n");
31.     }
32.     else
33.     {
34.         printf("The 2 lists are unequal.\n");
35.     }
36.     release (&p);
37.     release (&q);
38.
39.     return 0;
40. }
41.
42.int compare (struct node *p, struct node *q)
43.
44. while (p != NULL && q != NULL)
45. {
46.     if (p->num != q-> num)
47.     {
48.         return 0;
49.     }
50.     else
51.     {
52.         p = p->next;
53.         q = q->next;
54.     }
55. }
56. if (p != NULL || q != NULL)
57. {
58.     return 0;
59. }
60. else
61. {
62.     return 1;
63. }
64. }
65.
66. void feedmember (struct node **head)

```

```

67. {
68.     int c, ch;
69.     struct node *temp;
70.
71.     do
72.     {
73.         printf("Enter number: ");
74.         scanf("%d", &c);
75.         temp = (struct node *)malloc(sizeof(struct node));
76.         temp->num = c;
77.         temp->next = *head;
78.         *head = temp;
79.         printf("Do you wish to continue [1/0]: ");
80.         scanf("%d", &ch);
81.     }while (ch != 0);
82.     printf("\n");
83. }
84.
85. void release (struct node **head)
86. {
87.     struct node *temp = *head;
88.
89.     while ((*head) != NULL)
90.     {
91.         (*head) = (*head)->next;
92.         free(temp);
93.         temp = *head;
94.     }
95. }
```

```

1. /*
2.  * C Program to Check whether a Singly Linked List is a Palindrome
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
```

```
13. int create(struct node **);
14. int palin_check (struct node *, int);
15. void release(struct node **);
16.
17. int main()
18. {
19.     struct node *p = NULL;
20.     int result, count;
21.
22.     printf("Enter data into the list\n");
23.     count = create(&p);
24.     result = palin_check(p, count);
25.     if (result == 1)
26.     {
27.         printf("The linked list is a palindrome.\n");
28.     }
29.     else
30.     {
31.         printf("The linked list is not a palindrome.\n");
32.     }
33.     release (&p);
34.
35.     return 0;
36. }
37.
38. int palin_check (struct node *p, int count)
39. {
40.     int i = 0, j;
41.     struct node *front, *rear;
42.
43.     while (i != count / 2)
44.     {
45.         front = rear = p;
46.         for (j = 0; j < i; j++)
47.         {
48.             front = front->next;
49.         }
50.         for (j = 0; j < count - (i + 1); j++)
51.         {
52.             rear = rear->next;
53.         }
54.         if (front->num != rear->num)
55.     {
```

```
56.         return 0;
57.     }
58.     else
59.     {
60.         i++;
61.     }
62. }
63.
64. return 1;
65. }
66.
67.int create (struct node **head)
68.{
69.    int c, ch, count = 0;
70.    struct node *temp;
71.
72.    do
73.    {
74.        printf("Enter number: ");
75.        scanf("%d", &c);
76.        count++;
77.        temp = (struct node *)malloc(sizeof(struct node));
78.        temp->num = c;
79.        temp->next = *head;
80.        *head = temp;
81.        printf("Do you wish to continue [1/0]: ");
82.        scanf("%d", &ch);
83.    }while (ch != 0);
84.    printf("\n");
85.
86.    return count;
87. }
88.
89 void release (struct node **head)
90.{
91.    struct node *temp = *head;
92.
93.    while ((*head) != NULL)
94.    {
95.        (*head) = (*head)->next;
96.        free(temp);
97.        temp = *head;
98.    }
}
```

99. }

```
1. /*
2.  * C Program to Detect the Cycle in a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13.void create(struct node **);
14.void makecycle(struct node **);
15.void release(struct node **);
16.int detectcycle(struct node *);
17.
18.int main()
19.{
20.    struct node *p = NULL;
21.    int result;
22.
23.    printf("Enter data into the list\n");
24.    create(&p);
25.    makecycle(&p); //comment it to avoid cycle creation
26.    printf("Identifying if a cycle exists.\n");
27.    result = detectcycle(p);
28.    if (result)
29.    {
30.        printf("Cycle detected in the linked list.\n");
31.    }
32.    else
33.    {
34.        printf("No cycle detected in the linked list.\n");
35.    }
36.    release (&p);
37.
38.    return 0;
39.}
```

```

41. void makecycle(struct node **p)
42. {
43.     struct node *rear, *front;
44.     int n, count = 0, i;
45.
46.     front = rear = *p;
47.     while (rear->next != NULL)
48.     {
49.         rear = rear->next;
50.         count++;
51.     }
52.     if (count)
53.     {
54.         n = rand() % count;
55.     }
56.     else
57.     {
58.         n = 1;
59.     }
60.     for (i = 0; i < n - 1; i++)
61.     {
62.         front = front->next;
63.     }
64.     rear->next = front;
65.     /*At this point a cycle is generated in the list*/
66. }
67.
68. int detectcycle(struct node *head)
69. {
70.     int flag = 1, count = 1, i;
71.     struct node *p, *q;
72.
73.     p = q = head;
74.     q = q->next;
75.     while (1)
76.     {
77.         q = q->next;
78.         if (flag)
79.         {
80.             p = p->next;
81.         }
82.         if (q == p)
83.         {

```

```

84.         /*Deleting the Loop to deallocate the list*/
85.         q = q->next;
86.         while (q != p)
87.         {
88.             count++;
89.             q = q->next;
90.         }
91.         q = p = head;
92.         for (i = 0; i < count; i++)
93.         {
94.             q = q->next;
95.         }
96.         while (p != q)
97.         {
98.             p = p->next;
99.             q = q->next;
100.            }
101.            q->next = NULL;
102.
103.            return 1;
104.        }
105.        else if (q->next == NULL)
106.        {
107.            return 0;
108.        }
109.        flag = !flag;
110.    }
111.}
112.
113. void create(struct node **head)
114.{
115.    int c, ch;
116.    struct node *temp, *rear;
117.
118.    do
119.    {
120.        printf("Enter number: ");
121.        scanf("%d", &c);
122.        temp = (struct node *)malloc(sizeof(struct node));
123.        temp->num = c;
124.        temp->next = NULL;
125.        if (*head == NULL)
126.    {

```

```

127.             *head = temp;
128.         }
129.     else
130.     {
131.         rear->next = temp;
132.     }
133.     rear = temp;
134.     printf("Do you wish to continue [1/0]: ");
135.     scanf("%d", &ch);
136.     } while (ch != 0);
137.     printf("\n");
138. }
139.
140. void release(struct node **head)
141. {
142.     struct node *temp = *head;
143.     temp = temp->next;
144.     while ((*head) != NULL)
145.     {
146.         free(temp);
147.         temp = *head;
148.         (*head) = (*head)->next;
149.     }
150. }
```

```

1. /*
2.  * C Program to Find Number of Occurrences of All Elements in a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. struct node_occur
14. {
15.     int num;
16.     int times;
17.     struct node_occur *next;
```

```

18. };
19.
20. void create(struct node **);
21. void occur(struct node *, struct node_occur **);
22. void release(struct node **);
23. void release_2(struct node_occur **);
24. void display(struct node *);
25. void disp_occur(struct node_occur *);
26.
27. int main()
28. {
29.     struct node *p = NULL;
30.     struct node_occur *head = NULL;
31.     int n;
32.
33.     printf("Enter data into the list\n");
34.     create(&p);
35.     printf("Displaying the occurrence of each node in the list:\n");
36.     display(p);
37.     occur(p, &head);
38.     disp_occur(head);
39.     release(&p);
40.     release_2(&head);
41.
42.     return 0;
43. }
44.
45. void occur(struct node *head, struct node_occur **result)
46. {
47.     struct node *p;
48.     struct node_occur *temp, *prev;
49.
50.     p = head;
51.     while (p != NULL)
52.     {
53.         temp = *result;
54.         while (temp != NULL && temp->num != p->num)
55.         {
56.             prev = temp;
57.             temp = temp->next;
58.         }
59.         if (temp == NULL)
60.         {

```

```

61.         temp = (struct node_occur *)malloc(sizeof(struct node_occur));
62.         temp->num = p->num;
63.         temp->times = 1;
64.         temp->next = NULL;
65.         if (*result != NULL)
66.         {
67.             prev->next = temp;
68.         }
69.         else
70.         {
71.             *result = temp;
72.         }
73.     }
74.     else
75.     {
76.         temp->times += 1;
77.     }
78.     p = p->next;
79. }
80. }
81.
82. void create(struct node **head)
83. {
84.     int c, ch;
85.     struct node *temp, *rear;
86.
87.     do
88.     {
89.         printf("Enter number: ");
90.         scanf("%d", &c);
91.         temp = (struct node *)malloc(sizeof(struct node));
92.         temp->num = c;
93.         temp->next = NULL;
94.         if (*head == NULL)
95.         {
96.             *head = temp;
97.         }
98.         else
99.         {
100.             rear->next = temp;
101.         }
102.         rear = temp;
103.         printf("Do you wish to continue [1/0]: ");

```

```

104.         scanf("%d", &ch);
105.     } while (ch != 0);
106.     printf("\n");
107. }
108.
109. void display(struct node *p)
110. {
111.     while (p != NULL)
112.     {
113.         printf("%d\t", p->num);
114.         p = p->next;
115.     }
116.     printf("\n");
117. }
118.
119. void disp_occur(struct node_occur *p)
120. {
121.     printf("*****\nNumber\tOccurrence\n*****\n*****\n");
122.     while (p != NULL)
123.     {
124.         printf("    %d\t%d\n", p->num, p->times);
125.         p = p->next;
126.     }
127. }
128.
129. void release(struct node **head)
130. {
131.     struct node *temp = *head;
132.     *head = (*head)->next;
133.     while ((*head) != NULL)
134.     {
135.         free(temp);
136.         temp = *head;
137.         (*head) = (*head)->next;
138.     }
139. }
140.
141. void release_2(struct node_occur **head)
142. {
143.     struct node_occur *temp = *head;
144.     *head = (*head)->next;
145.     while ((*head) != NULL)

```

```

146.         {
147.             free(temp);
148.             temp = *head;
149.             (*head) = (*head)->next;
150.         }
151.     }

```

```

1. /*
2.  * C Program to Find the first Common Element between the 2 given Linked Lists
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13.void create(struct node **);
14.int find(struct node *, struct node *);
15 void release(struct node **);
16 void display(struct node *);
17.
18.int main()
19.{
20.    struct node *p = NULL, *q = NULL;
21.    int result;
22.
23.    printf("Enter data into the list\n");
24.    create(&p);
25.    printf("Enter data into the list\n");
26.    create(&q);
27.    printf("Displaying list1:\n");
28.    display(p);
29.    printf("Displaying list2:\n");
30.    display(q);
31.    result = find(p, q);
32.    if (result)
33.    {
34.        printf("The first matched element is %d.\n", result);
35.    }

```

```
36.     else
37.     {
38.         printf("No matching element found.\n");
39.     }
40.     release (&p);
41.
42.     return 0;
43. }
44.
45. int find(struct node *p, struct node *q)
46. {
47.     struct node *temp;
48.
49.     while (p != NULL)
50.     {
51.         temp = q;
52.         while (temp != NULL)
53.         {
54.             if (temp->num == p->num)
55.             {
56.                 return p->num;
57.             }
58.             temp = temp->next;
59.         }
60.         p = p->next;
61.     }
62.
63. /*Assuming 0 is not used in the list*/
64. return 0;
65. }
66.
67. void create(struct node **head)
68. {
69.     int c, ch;
70.     struct node *temp, *rear;
71.
72.     do
73.     {
74.         printf("Enter number: ");
75.         scanf("%d", &c);
76.         temp = (struct node *)malloc(sizeof(struct node));
77.         temp->num = c;
78.         temp->next = NULL;
```

```

79.     if (*head == NULL)
80.     {
81.         *head = temp;
82.     }
83.     else
84.     {
85.         rear->next = temp;
86.     }
87.     rear = temp;
88.     printf("Do you wish to continue [1/0]: ");
89.     scanf("%d", &ch);
90. } while (ch != 0);
91. printf("\n");
92. }
93.
94. void display(struct node *head)
95. {
96.     while (head != NULL)
97.     {
98.         printf("%d\t", head->num);
99.         head = head->next;
100.    }
101.    printf("\n");
102. }
103.
104. void release(struct node **head)
105. {
106.     struct node *temp;
107.     while ((*head) != NULL)
108.     {
109.         temp = *head;
110.         (*head) = (*head)->next;
111.         free(temp);
112.     }
113. }
```

```

1. /*
2. * C Program to Find the Largest Element in a Doubly Linked List
3. */
4.
5. #include <stdio.h>
6. #include <stdlib.h>
```

```
7.
8. struct node
9. {
10.     int num;
11.     struct node *next;
12.     struct node *prev;
13. };
14.
15. void create(struct node **);
16. int max(struct node *);
17. void release(struct node **);
18.
19. int main()
20. {
21.     struct node *p = NULL;
22.     int n;
23.
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     n = max(p);
27.     printf("The maximum number entered in the list is %.d.\n", n);
28.     release (&p);
29.
30.     return 0;
31. }
32.
33. int max(struct node *head)
34. {
35.     struct node *max, *q;
36.
37.     q = max = head;
38.     while (q != NULL)
39.     {
40.         if (q->num > max->num)
41.         {
42.             max = q;
43.         }
44.         q = q->next;
45.     }
46.
47.     return (max->num);
48. }
49.
```

```

50. void create(struct node **head)
51. {
52.     int c, ch;
53.     struct node *temp, *rear;
54.
55.     do
56.     {
57.         printf("Enter number: ");
58.         scanf("%d", &c);
59.         temp = (struct node *)malloc(sizeof(struct node));
60.         temp->num = c;
61.         temp->next = NULL;
62.         temp->prev = NULL;
63.         if (*head == NULL)
64.         {
65.             *head = temp;
66.         }
67.         else
68.         {
69.             rear->next = temp;
70.             temp->prev = rear;
71.         }
72.         rear = temp;
73.         printf("Do you wish to continue [1/0]: ");
74.         scanf("%d", &ch);
75.     } while (ch != 0);
76.     printf("\n");
77. }
78.
79. void release(struct node **head)
80. {
81.     struct node *temp = *head;
82.     *head = (*head)->next;
83.     while ((*head) != NULL)
84.     {
85.         free(temp);
86.         temp = *head;
87.         (*head) = (*head)->next;
88.     }
89. }
```

## 6. C Examples on Print, Reverse and Read Operations

```
1. /*
2.  * C Program to Print Middle most Node of a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void middlenode(struct node *);
15. void release(struct node **);
16.
17. int main()
18. {
19.     struct node *p = NULL;
20.
21.     printf("Enter data into the list\n");
22.     create(&p);
23.     middlenode(p);
24.     release (&p);
25.
26.     return 0;
27. }
28.
29. void middlenode(struct node *head)
30. {
31.     struct node *p, *q;
32.     int flag = 0;
33.
34.     q = p = head;
35.     /*for every two hops of q, one hop for p*/
36.     while (q->next != NULL)
37.     {
38.         q = q->next;
39.         if (flag)
40.         {
41.             p = p->next;
42.         }
43.         flag = !flag;
```

```

44.     }
45.     if (flag)
46.     {
47.         printf("List contains even number of nodes\nThe middle two node's values
48.             are: %d %d\n", p->next->num, p->num);
49.     }
50.     else
51.     {
52.         printf("The middle node of the list is: %d\n", p->num);
53.     }
54.
55. void create(struct node **head)
56. {
57.     int c, ch;
58.     struct node *temp;
59.
60.     do
61.     {
62.         printf("Enter number: ");
63.         scanf("%d", &c);
64.         temp = (struct node *)malloc(sizeof(struct node));
65.         temp->num = c;
66.         temp->next = *head;
67.         *head = temp;
68.         printf("Do you wish to continue [1/0]: ");
69.         scanf("%d", &ch);
70.     } while (ch != 0);
71.     printf("\n");
72. }
73.
74. void release(struct node **head)
75. {
76.     struct node *temp = *head;
77.     *head = (*head)->next;
78.     while ((*head) != NULL)
79.     {
80.         free(temp);
81.         temp = *head;
82.         (*head) = (*head)->next;
83.     }
84. }
```

```
1. /*
2.  * C Program to Read a Linked List in Reverse
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void reversedisplay(struct node *);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     struct node_occur *head = NULL;
22.     int n;
23.
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     printf("Displaying the nodes in the list:\n");
27.     display(p);
28.     printf("Displaying the list in reverse:\n");
29.     reversedisplay(p);
30.     release(&p);
31.
32.     return 0;
33. }
34.
35. void reversedisplay(struct node *head)
36. {
37.     if (head != NULL)
38.     {
39.         reversedisplay(head->next);
40.         printf("%d\t", head->num);
41.     }
}
```

```

42. }
43.
44. void create(struct node **head)
45. {
46.     int c, ch;
47.     struct node *temp, *rear;
48.
49.     do
50.     {
51.         printf("Enter number: ");
52.         scanf("%d", &c);
53.         temp = (struct node *)malloc(sizeof(struct node));
54.         temp->num = c;
55.         temp->next = NULL;
56.         if (*head == NULL)
57.         {
58.             *head = temp;
59.         }
60.         else
61.         {
62.             rear->next = temp;
63.         }
64.         rear = temp;
65.         printf("Do you wish to continue [1/0]: ");
66.         scanf("%d", &ch);
67.     } while (ch != 0);
68.     printf("\n");
69. }
70.
71. void display(struct node *p)
72. {
73.     while (p != NULL)
74.     {
75.         printf("%d\t", p->num);
76.         p = p->next;
77.     }
78.     printf("\n");
79. }
80.
81. void release(struct node **head)
82. {
83.     struct node *temp = *head;
84.     *head = (*head)->next;

```

```

85.     while ((*head) != NULL)
86.     {
87.         free(temp);
88.         temp = *head;
89.         (*head) = (*head)->next;
90.     }
91. }
```

```

1. /*
2.  * C Program to Remove Duplicates from a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void dup_delete(struct node **);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     struct node_occur *head = NULL;
22.     int n;
23.
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     printf("Displaying the nodes in the list:\n");
27.     display(p);
28.     printf("Deleting duplicate elements in the list...\n");
29.     dup_delete(&p);
30.     printf("Displaying non-deleted nodes in the list:\n");
31.     display(p);
32.     release(&p);
33.
34.     return 0;
```

```

35. }
36.
37. void dup_delete(struct node **head)
38. {
39.     struct node *p, *q, *prev, *temp;
40.
41.     p = q = prev = *head;
42.     q = q->next;
43.     while (p != NULL)
44.     {
45.         while (q != NULL && q->num != p->num)
46.         {
47.             prev = q;
48.             q = q->next;
49.         }
50.         if (q == NULL)
51.         {
52.             p = p->next;
53.             if (p != NULL)
54.             {
55.                 q = p->next;
56.             }
57.         }
58.         else if (q->num == p->num)
59.         {
60.             prev->next = q->next;
61.             temp = q;
62.             q = q->next;
63.             free(temp);
64.         }
65.     }
66. }
67.
68. void create(struct node **head)
69. {
70.     int c, ch;
71.     struct node *temp, *rear;
72.
73.     do
74.     {
75.         printf("Enter number: ");
76.         scanf("%d", &c);
77.         temp = (struct node *)malloc(sizeof(struct node));

```

```

78.         temp->num = c;
79.         temp->next = NULL;
80.         if (*head == NULL)
81.         {
82.             *head = temp;
83.         }
84.         else
85.         {
86.             rear->next = temp;
87.         }
88.         rear = temp;
89.         printf("Do you wish to continue [1/0]: ");
90.         scanf("%d", &ch);
91.     } while (ch != 0);
92.     printf("\n");
93. }
94.

95. void display(struct node *p)
96. {
97.     while (p != NULL)
98.     {
99.         printf("%d\t", p->num);
100.        p = p->next;
101.    }
102.    printf("\n");
103. }
104.

105. void release(struct node **head)
106. {
107.     struct node *temp = *head;
108.     *head = (*head)->next;
109.     while ((*head) != NULL)
110.     {
111.         free(temp);
112.         temp = *head;
113.         (*head) = (*head)->next;
114.     }
115. }
```

```

1. /*
2.  * C Program to Reverse a Linked List
3. */
```

```
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11. };
12.
13. void create(struct node **);
14. void reverse(struct node **);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     int n;
22.
23.     printf("Enter data into the list\n");
24.     create(&p);
25.     printf("Displaying the nodes in the list:\n");
26.     display(p);
27.     printf("Reversing the list...\n");
28.     reverse(&p);
29.     printf("Displaying the reversed list:\n");
30.     display(p);
31.     release(&p);
32.
33.     return 0;
34. }
35.
36. void reverse(struct node **head)
37. {
38.     struct node *p, *q, *r;
39.
40.     p = q = r = *head;
41.     p = p->next->next;
42.     q = q->next;
43.     r->next = NULL;
44.     q->next = r;
45.
46.     while (p != NULL)
```

```
47.     {
48.         r = q;
49.         q = p;
50.         p = p->next;
51.         q->next = r;
52.     }
53.     *head = q;
54. }
55.
56. void create(struct node **head)
57. {
58.     int c, ch;
59.     struct node *temp, *rear;
60.
61.     do
62.     {
63.         printf("Enter number: ");
64.         scanf("%d", &c);
65.         temp = (struct node *)malloc(sizeof(struct node));
66.         temp->num = c;
67.         temp->next = NULL;
68.         if (*head == NULL)
69.         {
70.             *head = temp;
71.         }
72.         else
73.         {
74.             rear->next = temp;
75.         }
76.         rear = temp;
77.         printf("Do you wish to continue [1/0]: ");
78.         scanf("%d", &ch);
79.     } while (ch != 0);
80.     printf("\n");
81. }
82.
83. void display(struct node *p)
84. {
85.     while (p != NULL)
86.     {
87.         printf("%d\t", p->num);
88.         p = p->next;
89.     }
```

```

90.     printf("\n");
91. }
92.
93. void release(struct node **head)
94. {
95.     struct node *temp = *head;
96.     *head = (*head)->next;
97.     while ((*head) != NULL)
98.     {
99.         free(temp);
100.        temp = *head;
101.        (*head) = (*head)->next;
102.    }
103. }
```

```

1. /*
2.  * C Program to Reverse only First N Elements of a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void reverse(struct node **, int);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     int n;
22.
23.     printf("Enter data into the list\n");
24.     create(&p);
25.     printf("Displaying the nodes in the list:\n");
26.     display(p);
27.     printf("Enter the number N to reverse first N node: ");
```

```
28.     scanf("%d", &n);
29.     printf("Reversing the list...\\n");
30.     if (n > 1)
31.     {
32.         reverse(&p, n - 2);
33.     }
34.     printf("Displaying the reversed list:\\n");
35.     display(p);
36.     release(&p);
37.
38.     return 0;
39. }
40.
41. void reverse(struct node **head, int n)
42. {
43.     struct node *p, *q, *r, *rear;
44.
45.     p = q = r = *head;
46.     if (n == 0)
47.     {
48.         q = q->next;
49.         p->next = q->next;
50.         q->next = p;
51.         *head = q;
52.     }
53.     else
54.     {
55.         p = p->next->next;
56.         q = q->next;
57.         r->next = NULL;
58.         rear = r;
59.         q->next = r;
60.
61.         while (n > 0 && p != NULL)
62.         {
63.             r = q;
64.             q = p;
65.             p = p->next;
66.             q->next = r;
67.             n--;
68.         }
69.         *head = q;
70.         rear->next = p;
```

```

71.     }
72. }
73.
74. void create(struct node **head)
75. {
76.     int c, ch;
77.     struct node *temp, *rear;
78.
79.     do
80.     {
81.         printf("Enter number: ");
82.         scanf("%d", &c);
83.         temp = (struct node *)malloc(sizeof(struct node));
84.         temp->num = c;
85.         temp->next = NULL;
86.         if (*head == NULL)
87.         {
88.             *head = temp;
89.         }
90.         else
91.         {
92.             rear->next = temp;
93.         }
94.         rear = temp;
95.         printf("Do you wish to continue [1/0]: ");
96.         scanf("%d", &ch);
97.     } while (ch != 0);
98.     printf("\n");
99. }
100.
101.    void display(struct node *p)
102.    {
103.        while (p != NULL)
104.        {
105.            printf("%d\t", p->num);
106.            p = p->next;
107.        }
108.        printf("\n");
109.    }
110.
111.    void release(struct node **head)
112.    {
113.        struct node *temp = *head;

```

```

114.         *head = (*head)->next;
115.         while ((*head) != NULL)
116.         {
117.             free(temp);
118.             temp = *head;
119.             (*head) = (*head)->next;
120.         }
121.     }

```

## 7. C Examples on Binary Tree Implementation using Linked List

```

1. /*
2.  * C Program that takes an Ordered Binary tree & Rearranges the
3.  * Internal Pointers to make a Circular Doubly Linked List out
4.  * of the Tree Nodes
5. */
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. struct node
10. {
11.     int num;
12.     struct node *left;
13.     struct node *right;
14.     int used;
15. };
16.
17. void create(struct node **);
18. void release(struct node **);
19. void display(struct node *, int);
20. struct node * transformdet(struct node *);
21. struct node * transform(struct node *);
22.
23. int main()
24. {
25.     struct node *root = NULL, *head;
26.
27.     printf("Creating binary tree:\n");
28.     create (&root);
29.     printf("Displaying binary tree:\n");
30.     display(root, 0);
31.     head = transform(root);

```

```
32.     printf("\nDisplaying circular linked list:\n");
33.     display(head, 1);
34.     root->left->right = NULL;
35.     release(&root);
36.
37.     return 0;
38. }
39.
40. struct node * transformdet(struct node *root)
41. {
42.     struct node *left, *right;
43.
44.     if (root == NULL)
45.     {
46.         return root;
47.     }
48.     if (root->left != NULL)
49.     {
50.         left = transformdet(root->left);
51.         while (left->right != NULL)
52.         {
53.             left = left->right;
54.         }
55.         left->right = root;
56.         root->left = left;
57.     }
58.     if (root->right != NULL)
59.     {
60.         right = transformdet(root->right);
61.         while (right->left != NULL)
62.         {
63.             right = right->left;
64.         }
65.         right->left = root;
66.         root->right = right;
67.     }
68.
69.     return root;
70. }
71.
72. struct node * transform(struct node *root)
73. {
74.     struct node *rear;
```

```

75.     if (root == NULL)
76.     {
77.         return root;
78.     }
79.     root = transformdet(root);
80.     rear = root;
81.     while (root->left != NULL)
82.     {
83.         root = root->left;
84.     }
85.     while (rear->right != NULL)
86.     {
87.         rear = rear->right;
88.     }
89.     root->left = rear;
90.     rear->right = root;
91.
92.     return (root);
93. }
94.
95. void create(struct node **root)
96. {
97.     struct node *temp, *p, *q;
98.     int a, ch;
99.
100.    do
101.    {
102.        p = *root;
103.        printf("Enter a number in the tree: ");
104.        scanf("%d", &a);
105.        temp = (struct node *)malloc(sizeof(struct node));
106.        temp->num = a;
107.        temp->used = 0;
108.        temp->left = temp->right = NULL;
109.        if (*root == NULL)
110.        {
111.            *root = temp;
112.        }
113.        else
114.        {
115.            while (p != NULL)
116.            {
117.                q = p;

```

```

118.             if (p->num >= temp->num)
119.             {
120.                 p = p->right;
121.             }
122.             else
123.             {
124.                 p = p->left;
125.             }
126.         }
127.         if (q->num >= temp->num)
128.         {
129.             q->right = temp;
130.         }
131.         else
132.         {
133.             q->left = temp;
134.         }
135.     }
136.     printf("Do you want to add more numbers? [1/0]\n");
137.     scanf("%d", &ch);
138. } while (ch != 0);
139.
140.
141. void display(struct node *root, int n)
142. {
143.     struct node *temp;
144.
145.     if (root != NULL && !n)
146.     {
147.         display(root->left, 0);
148.         printf("%d ", root->num);
149.         display(root->right, 0);
150.     }
151.     else if (root != NULL && n)
152.     {
153.         temp = root;
154.         printf("%d ", temp->num);
155.         temp = temp->right;
156.         while (temp != root)
157.         {
158.             printf("%d ", temp->num);
159.             temp = temp->right;
160.         }

```

```
161.         printf("\n");
162.     }
163. }
164.
165. void release(struct node **root)
166. {
167.     if (*root != NULL)
168.     {
169.         release(&(*root)->right);
170.         free(*root);
171.     }
172. }
173. /*
174. * C Program to Construct a Balanced Binary Search Tree
175. * which has same data members as the given Doubly Linked List
176. */
177. #include <stdio.h>
178. #include <stdlib.h>
179.
180. struct node
181. {
182.     int num;
183.     struct node *left;
184.     struct node *right;
185. };
186.
187. void create(struct node **);
188. void treemaker(struct node **, int);
189. void display(struct node *);
190. void displayTree(struct node *);
191. void delete(struct node **);
192.
193. int main()
194. {
195.     struct node *headList = NULL, *rootTree, *p;
196.     int count = 1, flag = 0;
197.
198.     create(&headList);
199.     printf("Displaying the doubly linked list:\n");
200.     display(headList);
201.     rootTree = p = headList;
202.     while (p->right != NULL)
203.     {
```

```

204.         p = p->right;
205.         count = count + 1;
206.         if (flag)
207.         {
208.             rootTree = rootTree->right;
209.         }
210.         flag = !flag;
211.     }
212.     treemaker(&rootTree, count / 2);
213.     printf("Displaying the tree: (Inorder)\n");
214.     displayTree(rootTree);
215.     printf("\n");
216.
217.     return 0;
218. }
219.
220. void create(struct node **head)
221. {
222.     struct node *rear, *temp;
223.     int a, ch;
224.
225.     do
226.     {
227.         printf("Enter a number: ");
228.         scanf("%d", &a);
229.         temp = (struct node *)malloc(sizeof(struct node));
230.         temp->num = a;
231.         temp->right = NULL;
232.         temp->left = NULL;
233.         if (*head == NULL)
234.         {
235.             *head = temp;
236.         }
237.         else
238.         {
239.             rear->right = temp;
240.             temp->left = rear;
241.         }
242.         rear = temp;
243.         printf("Do you wish to continue [1/0] ?: ");
244.         scanf("%d", &ch);
245.     } while (ch != 0);
246. }
```

```
247.  
248. void treemaker(struct node **root, int count)  
249. {  
250.     struct node *quarter, *thirdquarter;  
251.     int n = count, i = 0;  
252.  
253.     if ((*root)->left != NULL)  
254.     {  
255.         quarter = (*root)->left;  
256.         for (i = 1; (i < count / 2) && (quarter->left != NULL); i++)  
257.         {  
258.             quarter = quarter->left;  
259.         }  
260.         (*root)->left->right = NULL;  
261.         (*root)->left = quarter;  
262.         /*  
263.          * Uncomment the following line to see when the pointer changes  
264.          */  
265.         //printf("%d's Left child is now %d\n", (*root)->num, quarter->num);  
266.         if (quarter != NULL)  
267.         {  
268.             treemaker(&quarter, count / 2);  
269.         }  
270.     }  
271.     if ((*root)->right != NULL)  
272.     {  
273.         thirdquarter = (*root)->right;  
274.         for (i = 1; (i < count / 2) && (thirdquarter->right != NULL); i++)  
275.         {  
276.             thirdquarter = thirdquarter->right;  
277.         }  
278.         (*root)->right->left = NULL;  
279.         (*root)->right = thirdquarter;  
280.         /*  
281.          * Uncomment the following line to see when the pointer changes  
282.          */  
283.         //printf("%d's right child is now %d\n", (*root)->num, thirdquarter-  
284.             >num);  
285.         if (thirdquarter != NULL)  
286.         {  
287.             treemaker(&thirdquarter, count / 2);  
288.         }  
289.     }
```

```

289. }
290.
291. void display(struct node *head)
292. {
293.     while (head != NULL)
294.     {
295.         printf("%d ", head->num);
296.         head = head->right;
297.     }
298.     printf("\n");
299. }
300.
301. /*DisplayTree performs inorder traversal*/
302. void displayTree(struct node *root)
303. {
304.     if (root != NULL)
305.     {
306.         displayTree(root->left);
307.         printf("%d ", root->num);
308.         displayTree(root->right);
309.     }
310. }
311.
312. void delete(struct node **root)
313. {
314.     if (*root != NULL)
315.     {
316.         displayTree((*root)->left);
317.         displayTree((*root)->right);
318.         free(*root);
319.     }
320. }

```

```

1. /*
2.  * C Program to Convert a Binary Tree into a Singly Linked List by Traversing Level
3.   by Level
4. */
5. <pre>
6. #include <stdio.h>
7. #include <stdlib.h>
8. /*structure type to create a tree*/

```

```
9. struct node
10. {
11.     int num;
12.     struct node *left;
13.     struct node *right;
14. };
15. /*
16. * structure type to point to the nodes of a tree
17. * and also create self-referential list used for
18. * queueing.
19. */
20. struct queue
21. {
22.     struct node *nodeptr;
23.     struct queue *next;
24. };
25. /* resulting singly linked list */
26. struct list
27. {
28.     int num;
29.     struct list *next;
30. };
31.
32. void createTree(struct node **);
33. void createlistbfs(struct node *, struct list **);
34. void delete(struct node **);
35. void display(struct list *);
36. void deleteList(struct list **);
37.
38. int main()
39. {
40.     struct node *root = NULL;
41.     struct list *head = NULL;
42.
43.     createTree(&root);
44.     createlistbfs(root, &head);
45.     printf("Displaying the list generated at node by node level of the tree: ");
46.     display(head);
47.     deleteList(&head);
48.     delete(&root);
49.
50.     return 0;
51. }
```

```
52.  
53. void createTree(struct node **root)  
54. {  
55.     struct node *temp, *p, *q;  
56.     int a, ch;  
57.  
58.     do  
59.     {  
60.         printf("Enter a number for a node: ");  
61.         scanf("%d", &a);  
62.         temp = (struct node *)malloc(sizeof(struct node));  
63.         temp->num = a;  
64.         temp->left = NULL;  
65.         temp->right = NULL;  
66.         p = q = *root;  
67.         if (*root == NULL)  
68.         {  
69.             *root = temp;  
70.         }  
71.         else  
72.         {  
73.             while (1)  
74.             {  
75.                 q = p;  
76.                 if (p->num >= temp->num)  
77.                 {  
78.                     p = p->left;  
79.                 }  
80.                 else  
81.                 {  
82.                     p = p->right;  
83.                 }  
84.                 if (p == NULL)  
85.                 {  
86.                     break;  
87.                 }  
88.             }  
89.             if (q->num >= temp->num)  
90.                 q->left = temp;  
91.             else  
92.                 q->right = temp;  
93.         }  
94.         printf("Do you want to continue? [1/0]: ");
```

```
95.         scanf("%d", &ch);
96.     } while (ch != 0);
97. }
98.
99. void createlistbfs(struct node *root, struct list **head)
100. {
101.     struct queue *qhead, *qrear, *qtemp, *qrelease;
102.     struct list *temp, *rear;
103.
104.     if (root == NULL)
105.     {
106.         return;
107.     }
108.     qhead = (struct queue *)malloc(sizeof(struct queue));
109.     qhead->nodeptr = root;
110.     qhead->next = NULL;
111.     qrear = qhead;
112.     while (qhead != NULL)
113.     {
114.
115.         temp = (struct list *)malloc(sizeof(struct list));
116.         temp->num = qhead->nodeptr->num;
117.         temp->next = NULL;
118.         if (*head == NULL)
119.         {
120.             *head = temp;
121.         }
122.         else
123.         {
124.             rear->next = temp;
125.         }
126.         rear = temp;
127.         if (qhead->nodeptr->left != NULL)
128.         {
129.             qtemp = (struct queue *)malloc(sizeof(struct queue));
130.             qtemp->nodeptr = qhead->nodeptr->left;
131.             qtemp->next = NULL;
132.             qrear->next = qtemp;
133.             qrear = qtemp;
134.         }
135.         if (qhead->nodeptr->right != NULL)
136.         {
137.             qtemp = (struct queue *)malloc(sizeof(struct queue));
```

```
138.         qtemp->nodeptr = qhead->nodeptr->right;
139.         qtemp->next = NULL;
140.         qrear->next = qtemp;
141.         qrear = qtemp;
142.     }
143.     qrelease = qhead;
144.     qhead = qhead->next;
145.     free(qrelease);
146. }
147. }
148.
149. void delete(struct node **root)
150. {
151.     if (*root == NULL)
152.     {
153.         return;
154.     }
155.     else
156.     {
157.         if ((*root)->left != NULL)
158.         {
159.             delete(&((*root)->left));
160.         }
161.         if ((*root)->right != NULL)
162.         {
163.             delete(&((*root)->right));
164.         }
165.     }
166. }
167.
168. void display(struct list *head)
169. {
170.     while (head != NULL)
171.     {
172.         printf("%d ", head->num);
173.         head = head->next;
174.     }
175. }
176.
177. void deleteList(struct list **head)
178. {
179.     struct list *temp;
```

```

181.         temp = *head;
182.         while (temp != NULL)
183.         {
184.             *head = (*head)->next;
185.             free(temp);
186.             temp = *head;
187.         }
188.     }

```

```

1. /*
2.  * C Program to Convert a given Singly Linked List to a Circular List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void tocircular(struct node **);
15. void release(struct node **);
16. void display(struct node *);
17.
18. int main()
19. {
20.     struct node *p = NULL;
21.     int result, count;
22.
23.     printf("Enter data into the list\n");
24.     create(&p);
25.     tocircular(&p);
26.     printf("Circular list generated\n");
27.     display(p);
28.     release (&p);
29.
30.     return 0;
31. }
32.
33. void tocircular(struct node **p)

```

```

34. {
35.     struct node *rear;
36.
37.     rear = *p;
38.     while (rear->next != NULL)
39.     {
40.         rear = rear->next;
41.     }
42.     rear->next = *p;
43.     /*After this the singly linked List is now circular*/
44. }
45.
46. void create(struct node **head)
47. {
48.     int c, ch;
49.     struct node *temp;
50.
51.     do
52.     {
53.         printf("Enter number: ");
54.         scanf("%d", &c);
55.         temp = (struct node *)malloc(sizeof(struct node));
56.         temp->num = c;
57.         temp->next = *head;
58.         *head = temp;
59.         printf("Do you wish to continue [1/0]: ");
60.         scanf("%d", &ch);
61.     } while (ch != 0);
62.     printf("\n");
63. }
64.
65. void display(struct node *head)
66. {
67.     struct node *temp = head;
68.     printf("Displaying the list elements\n");
69.     printf("%d\t", temp->num);
70.     temp = temp->next;
71.     while (head != temp)
72.     {
73.         printf("%d\t", temp->num);
74.         temp = temp->next;
75.     }
76.     printf("and back to %d\t%d ..\n", temp->num, temp->next->num);

```

```

77. }
78.
79. void release(struct node **head)
80. {
81.     struct node *temp = *head;
82.     temp = temp->next;
83.     (*head)->next = NULL;
84.     (*head) = temp->next;
85.     while ((*head) != NULL)
86.     {
87.         free(temp);
88.         temp = *head;
89.         (*head) = (*head)->next;
90.     }
91. }
```

```

1. /*
2.  * C Program to Find Intersection & Union of 2 Linked Lists
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13. void create(struct node **);
14. void findunion(struct node *, struct node *, struct node **);
15. void findintersect(struct node *, struct node *, struct node **);
16. void display(struct node *);
17. void release(struct node **);
18.
19. int main()
20. {
21.     struct node *phead, *qhead, *intersect, *unionlist;
22.
23.     phead = qhead = intersect = unionlist = NULL;
24.     printf("Enter elements in the list 1\n");
25.     create(&phead);
26.     printf("\nEnter elements in the list 2\n");
```

```

27.     create(&qhead);
28.     findunion(phead, qhead, &unionlist);
29.     findintersect(phead, qhead, &intersect);
30.     printf("\nDisplaying list 1:\n");
31.     display(phead);
32.     printf("Displaying list 2:\n");
33.     display(qhead);
34.     printf("Displaying the union of the 2 lists:\n");
35.     display(unionlist);
36.     printf("Displaying the intersection of the 2 lists:\n");
37.     if (intersect == NULL)
38.     {
39.         printf("Null\n");
40.     }
41.     else
42.     {
43.         display(intersect);
44.     }
45.     release(&phead);
46.     release(&qhead);
47.     release(&unionlist);
48.     release(&intersect);
49.
50.     return 0;
51. }
52.
53. void findintersect(struct node *p, struct node *q, struct node **intersect)
54. {
55.     struct node *ptemp, *qtemp, *itemp, *irear, *ifront;
56.
57.     ptemp = p;
58.     while (ptemp != NULL)
59.     {
60.         qtemp = q;
61.         ifront = *intersect;
62.         while (qtemp != NULL && ptemp->num != qtemp->num)
63.         {
64.             qtemp = qtemp->next;
65.         }
66.         if (qtemp != NULL)
67.         {
68.             if (ifront != NULL)
69.             {

```

```

70.             if (ifront->num == qtemp->num)
71.             {
72.                 ptemp = ptemp->next;
73.                 continue;
74.             }
75.             ifront = ifront->next;
76.         }
77.         itemp = (struct node *)malloc(sizeof(struct node));
78.         itemp->num = qtemp->num;
79.         itemp->next = NULL;
80.         if (*intersect == NULL)
81.         {
82.             *intersect = itemp;
83.         }
84.         else
85.         {
86.             irear->next = itemp;
87.         }
88.         irear = itemp;
89.     }
90.     ptemp = ptemp->next;
91. }
92. }

93.

94. void findunion(struct node *p, struct node *q, struct node **unionlist)
95. {
96.     struct node *utemp, *ufront, *urear;
97.     int flag = 0;
98.

99.     while (p != NULL)
100.     {
101.         upfront = *unionlist;
102.         while (ufront != NULL)
103.         {
104.             if (ufront->num == p->num)
105.             {
106.                 flag = 1;
107.             }
108.             upfront = upfront->next;
109.         }
110.         if (flag)
111.         {
112.             flag = 0;

```

```
113.     }
114.     else
115.     {
116.         utemp = (struct node *)malloc(sizeof(struct node));
117.         utemp->num = p->num;
118.         utemp->next = NULL;
119.         if (*unionlist == NULL)
120.         {
121.             *unionlist = utemp;
122.         }
123.         else
124.         {
125.             urear->next = utemp;
126.         }
127.         urear = utemp;
128.     }
129.     p = p->next;
130. }
131. while (q != NULL)
132. {
133.     ufront = *unionlist;
134.     while (ufront != NULL)
135.     {
136.         if (ufront->num == q->num)
137.         {
138.             flag = 1;
139.         }
140.         ufront = ufront->next;
141.     }
142.     if (flag)
143.     {
144.         flag = 0;
145.     }
146.     else
147.     {
148.         utemp = (struct node *)malloc(sizeof(struct node));
149.         utemp->num = q->num;
150.         utemp->next = NULL;
151.         if (*unionlist == NULL)
152.         {
153.             *unionlist = utemp;
154.         }
155.         else
```

```

156.          {
157.              urear->next = utemp;
158.          }
159.          urear = utemp;
160.      }
161.      q = q->next;
162.  }
163. }
164.
165. void create(struct node **head)
166. {
167.     struct node *temp, *rear;
168.     int ch, a;
169.
170.     do
171.     {
172.         printf("Enter a number: ");
173.         scanf("%d", &a);
174.         temp = (struct node *)malloc(sizeof(struct node));
175.         temp->num = a;
176.         temp->next = NULL;
177.         if (*head == NULL)
178.         {
179.             *head = temp;
180.         }
181.         else
182.         {
183.             rear->next = temp;
184.         }
185.         rear = temp;
186.         printf("Do you want to continue [1/0] ? ");
187.         scanf("%d", &ch);
188.     } while (ch != 0);
189. }
190.
191. void display(struct node *head)
192. {
193.     while (head != NULL)
194.     {
195.         printf("%d    ", head->num);
196.         head = head->next;
197.     }
198.     printf("\n");

```

```

199.     }
200.
201.     void release(struct node **head)
202.     {
203.         struct node *temp = *head;
204.         while ((*head) != NULL)
205.         {
206.             (*head) = (*head)->next;
207.             free (temp);
208.             temp = *head;
209.         }
210.     }

```

## 8. C Examples on Interchange and Modify Operations

```

1. /*
2.  * C Program to Interchange the two Adjacent Nodes given a circular
3.  * Linked List
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int num;
11.     struct node *next;
12. };
13.
14. void create(struct node **);
15. void tocircular(struct node **);
16. void release(struct node **);
17. void change(struct node **, int);
18. void display(struct node *);
19.
20. int main()
21. {
22.     struct node *p = NULL;
23.     int num;
24.
25.     printf("Enter data into the list\n");
26.     create(&p);
27.     tocircular(&p);
28.     printf("Circular list generated\n");

```

```
29.     display(p);
30.     printf("Enter node position to interchange with it's adjacent: ");
31.     scanf("%d", &num);
32.     change(&p, num - 2);
33.     printf("After interchanging, ");
34.     display(p);
35.     release (&p);
36.
37.     return 0;
38. }
39.
40. void tocircular(struct node **p)
41. {
42.     struct node *rear;
43.
44.     rear = *p;
45.     while (rear->next != NULL)
46.     {
47.         rear = rear->next;
48.     }
49.     rear->next = *p;
50.     /*After this the singly linked list is now circular*/
51. }
52.
53. void change(struct node **head, int num)
54. {
55.     struct node *p, *q, *r;
56.
57.     p = q = r = *head;
58.     p = p->next->next;
59.     q = q->next;
60.     while (num != 0)
61.     {
62.         r = q;
63.         q = p;
64.         p = p->next;
65.         num--;
66.     }
67.     r->next = p;
68.     q->next = p->next;
69.     p->next = q;
70. }
71.
```

```
72. void create(struct node **head)
73. {
74.     int c, ch;
75.     struct node *temp, *rear;
76.
77.     do
78.     {
79.         printf("Enter number: ");
80.         scanf("%d", &c);
81.         temp = (struct node *)malloc(sizeof(struct node));
82.         temp->num = c;
83.         temp->next = NULL;
84.         if (*head == NULL)
85.         {
86.             *head = temp;
87.         }
88.         else
89.         {
90.             rear->next = temp;
91.         }
92.         rear = temp;
93.         printf("Do you wish to continue [1/0]: ");
94.         scanf("%d", &ch);
95.     } while (ch != 0);
96.     printf("\n");
97. }
98.
99. void display(struct node *head)
100. {
101.     struct node *temp = head;
102.     printf("Displaying the list elements\n");
103.     printf("%d\t", temp->num);
104.     temp = temp->next;
105.     while (head != temp)
106.     {
107.         printf("%d\t", temp->num);
108.         temp = temp->next;
109.     }
110.     printf("\n");
111. }
112.
113. void release(struct node **head)
114. {
```

```

115.         struct node *temp = *head;
116.         temp = temp->next;
117.         (*head)->next = NULL;
118.         (*head) = temp->next;
119.         while ((*head) != NULL)
120.         {
121.             free(temp);
122.             temp = *head;
123.             (*head) = (*head)->next;
124.         }
125.     }

```

```

1. /*
2.  * C Program to Interchange two Elements of the List without
3.  * touching the Key Field
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int num;
11.     struct node *next;
12. };
13.
14. void create(struct node **);
15. void release(struct node **);
16. void change(struct node **, int, int);
17. void display(struct node *);
18.
19. int main()
20. {
21.     struct node *p = NULL;
22.     int num1, num2;
23.
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     printf("Circular list generated\n");
27.     display(p);
28.     printf("Enter node position: ");
29.     scanf("%d", &num1);
30.     printf("Enter node position to exchange with: ");

```

```

31.     scanf("%d", &num2);
32.     change(&p, num1 - 2, num2 - 2);
33.     printf("After interchanging, ");
34.     display(p);
35.     release (&p);
36.
37.     return 0;
38. }
39.
40. void change(struct node **head, int num1, int num2)
41. {
42.     struct node *p1, *q1, *r1;
43.     struct node *p2, *q2, *r2;
44.
45.     p1 = q1 = r1 = *head;
46.     p2 = q2 = r2 = *head;
47.     if (num1 == num2)
48.     {
49.         return;
50.     }
51.     else if ((p1->next == NULL && num1 > 0) || (p1->next->next == NULL && num1 > 1))
52.     {
53.         printf("List smaller than entered node position.\n");
54.     }
55.     else if ((p2->next == NULL && num2 > 0) || (p2->next->next == NULL && num2 > 1))
56.     {
57.         printf("List smaller than entered node position.\n");
58.     }
59.     else
60.     {
61.         if (num1 >=0 && num2 >= 0)
62.         {
63.             p1 = p1->next->next;
64.             q1 = q1->next;
65.             while (num1 > 0)
66.             {
67.                 r1 = q1;
68.                 q1 = p1;
69.                 p1 = p1->next;
70.                 num1--;
71.             }
72.             p2 = p2->next->next;
73.             q2 = q2->next;

```

```
74.         while (num2 > 0)
75.        {
76.            r2 = q2;
77.            q2 = p2;
78.            p2 = p2->next;
79.            num2--;
80.        }
81.        r2->next = q1;
82.        q2->next = p1;
83.        r1->next = q2;
84.        q1->next = p2;
85.    }
86.    else if (num1 == -1)
87.    {
88.        p2 = p2->next->next;
89.        q2 = q2->next;
90.        while (num2 > 0)
91.        {
92.            r2 = q2;
93.            q2 = p2;
94.            p2 = p2->next;
95.            num2--;
96.        }
97.        if (p1->next != q2)
98.        {
99.            q2->next = p1->next;
100.           p1->next = p2;
101.           r2->next = p1;
102.       }
103.       else
104.       {
105.           p1->next = q2->next;
106.           q2->next = p1;
107.       }
108.       *head = q2;
109.    }
110.    else if (num2 == -1)
111.    {
112.        p1 = p1->next->next;
113.        q1 = q1->next;
114.        while (num1 > 0)
115.        {
116.            r1 = q1;
```

```

117.                 q1 = p1;
118.                 p1 = p1->next;
119.                 num1--;
120.             }
121.             if (p2->next != q1)
122.             {
123.                 q1->next = p2->next;
124.                 p2->next = p1;
125.                 r1->next = p2;
126.             }
127.             else
128.             {
129.                 p2->next = q1->next;
130.                 q1->next = p2;
131.             }
132.             *head = q1;
133.         }
134.     }
135. }
136.
137. void create(struct node **head)
138. {
139.     int c, ch;
140.     struct node *temp, *rear;
141.
142.     do
143.     {
144.         printf("Enter number: ");
145.         scanf("%d", &c);
146.         temp = (struct node *)malloc(sizeof(struct node));
147.         temp->num = c;
148.         temp->next = NULL;
149.         if (*head == NULL)
150.         {
151.             *head = temp;
152.         }
153.         else
154.         {
155.             rear->next = temp;
156.         }
157.         rear = temp;
158.         printf("Do you wish to continue [1/0]: ");
159.         scanf("%d", &ch);

```

```

160.         } while (ch != 0);
161.         printf("\n");
162.     }
163.
164.     void display(struct node *head)
165.     {
166.         struct node *temp = head;
167.         printf("Displaying the list elements\n");
168.         while (temp != NULL)
169.         {
170.             printf("%d\t", temp->num);
171.             temp = temp->next;
172.         }
173.         printf("\n");
174.     }
175.
176.     void release(struct node **head)
177.     {
178.         struct node *temp = *head;
179.         *head = (*head)->next;
180.         while ((*head) != NULL)
181.         {
182.             free(temp);
183.             temp = *head;
184.             (*head) = (*head)->next;
185.         }
186.     }

```

```

1. /*
2.  * C Program to Modify the Linked List such that All Even Numbers
3.  * appear before all the Odd Numbers in the Modified Linked List
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int num;
11.     struct node *next;
12. };
13.
14. void create(struct node **);

```

```
15. void generate_evenodd(struct node *, struct node**);
16. void release(struct node **);
17. void display(struct node *);
18.
19. int main()
20. {
21.     struct node *p = NULL, *q = NULL;
22.     int key, result;
23.
24.     printf("Enter data into the list\n");
25.     create(&p);
26.     printf("Displaying the nodes in the list:\n");
27.     display(p);
28.     generate_evenodd(p, &q);
29.     printf("Displaying the list with even and then odd:\n");
30.     display(q);
31.     release(&p);
32.
33.     return 0;
34. }
35.
36. void generate_evenodd(struct node *list, struct node **head)
37. {
38.     struct node *even = NULL, *odd = NULL, *temp;
39.     struct node *reven, *rodd;
40.     while (list != NULL)
41.     {
42.         temp = (struct node *)malloc(sizeof(struct node));
43.         temp->num = list->num;
44.         temp->next = NULL;
45.         if (list->num % 2 == 0)
46.         {
47.             if (even == NULL)
48.             {
49.                 even = temp;
50.             }
51.             else
52.             {
53.                 reven->next = temp;
54.             }
55.             reven = temp;
56.         }
57.         else
```

```

58.     {
59.         if (odd == NULL)
60.         {
61.             odd = temp;
62.         }
63.         else
64.         {
65.             rodd->next = temp;
66.         }
67.         rodd = temp;
68.     }
69.     list = list->next;
70. }
71. reven->next = odd;
72. *head = even;
73. }
74.
75. void create(struct node **head)
76. {
77.     int c, ch;
78.     struct node *temp, *rear;
79.
80.     do
81.     {
82.         printf("Enter number: ");
83.         scanf("%d", &c);
84.         temp = (struct node *)malloc(sizeof(struct node));
85.         temp->num = c;
86.         temp->next = NULL;
87.         if (*head == NULL)
88.         {
89.             *head = temp;
90.         }
91.         else
92.         {
93.             rear->next = temp;
94.         }
95.         rear = temp;
96.         printf("Do you wish to continue [1/0]: ");
97.         scanf("%d", &ch);
98.     } while (ch != 0);
99.     printf("\n");
100. }
```

```

101.
102.     void display(struct node *p)
103.     {
104.         while (p != NULL)
105.         {
106.             printf("%d\t", p->num);
107.             p = p->next;
108.         }
109.         printf("\n");
110.     }
111.
112.     void release(struct node **head)
113.     {
114.         struct node *temp = *head;
115.         *head = (*head)->next;
116.         while ((*head) != NULL)
117.         {
118.             free(temp);
119.             temp = *head;
120.             (*head) = (*head)->next;
121.         }
122.     }

```

```

1. /*
2.  * C Program to Print Nth Node from the Last of a Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int num;
10.    struct node *next;
11.};
12.
13.void create(struct node **);
14.void nthnode(struct node *, int);
15.void release(struct node **);
16.
17.int main()
18.{
19.    struct node *p = NULL;

```

```

20.     int n;
21.
22.     printf("Enter data into the list\n");
23.     create(&p);
24.     printf("Enter the value n to find nth position from the last node: ");
25.     scanf("%d", &n);
26.     nthnode(p, n);
27.     release (&p);
28.
29.     return 0;
30. }
31.
32. void nthnode(struct node *head, int n)
33. {
34.     struct node *p, *q;
35.     int i;
36.
37.     q = p = head;
38.
39.     for (i = 0; i < n && q != NULL; i++)
40.     {
41.         q = q->next;
42.     }
43.     if (i < n)
44.     {
45.         printf("Entered n = %d is larger than the number of elements = %d in list.
Please try again.\n", n, i);
46.     }
47.     else
48.     {
49.         while (q->next != NULL)
50.         {
51.             q = q->next;
52.             p = p->next;
53.         }
54.         printf("%d is %d nodes from the last node.\n", p->num, n);
55.     }
56. }
57.
58. void create(struct node **head)
59. {
60.     int c, ch;
61.     struct node *temp, *rear;

```

```

62.
63.     do
64. {
65.     printf("Enter number: ");
66.     scanf("%d", &c);
67.     temp = (struct node *)malloc(sizeof(struct node));
68.     temp->num = c;
69.     temp->next = NULL;
70.     if (*head == NULL)
71.     {
72.         *head = temp;
73.     }
74.     else
75.     {
76.         rear->next = temp;
77.     }
78.     rear = temp;
79.     printf("Do you wish to continue [1/0]: ");
80.     scanf("%d", &ch);
81. } while (ch != 0);
82. printf("\n");
83. }
84.
85. void release(struct node **head)
86. {
87.     struct node *temp = *head;
88.     *head = (*head)->next;
89.     while ((*head) != NULL)
90.     {
91.         free(temp);
92.         temp = *head;
93.         (*head) = (*head)->next;
94.     }
95. }
```

```

1. /*
2.  * C Program to Solve Josephus Problem using Linked List
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
```

```
8. {
9.     int num;
10.    struct node *next;
11. };
12.
13. void create(struct node **);
14. void display(struct node *);
15. int survivor(struct node **, int);
16.
17. int main()
18. {
19.     struct node *head = NULL;
20.     int survive, skip;
21.
22.     create(&head);
23.     printf("The persons in circular list are:\n");
24.     display(head);
25.     printf("Enter the number of persons to be skipped: ");
26.     scanf("%d", &skip);
27.     survive = survivor(&head, skip);
28.     printf("The person to survive is : %d\n", survive);
29.     free(head);
30.
31.     return 0;
32. }
33.
34. int survivor(struct node **head, int k)
35. {
36.     struct node *p, *q;
37.     int i;
38.
39.     q = p = *head;
40.     while (p->next != p)
41.     {
42.         for (i = 0; i < k - 1; i++)
43.         {
44.             q = p;
45.             p = p->next;
46.         }
47.         q->next = p->next;
48.         printf("%d has been killed.\n", p->num);
49.         free(p);
50.         p = q->next;
```

```
51.     }
52.     *head = p;
53.
54.     return (p->num);
55. }
56.
57. void create (struct node **head)
58. {
59.     struct node *temp, *rear;
60.     int a, ch;
61.
62.     do
63.     {
64.         printf("Enter a number: ");
65.         scanf("%d", &a);
66.         temp = (struct node *)malloc(sizeof(struct node));
67.         temp->num = a;
68.         temp->next = NULL;
69.         if (*head == NULL)
70.         {
71.             *head = temp;
72.         }
73.         else
74.         {
75.             rear->next = temp;
76.         }
77.         rear = temp;
78.         printf("Do you want to add a number [1/0]? ");
79.         scanf("%d", &ch);
80.     } while (ch != 0);
81.     rear->next = *head;
82. }
83.
84. void display(struct node *head)
85. {
86.     struct node *temp;
87.
88.     temp = head;
89.     printf("%d    ", temp->num);
90.     temp = temp->next;
91.     while (head != temp)
92.     {
93.         printf("%d    ", temp->num);
```

```

94.         temp = temp->next;
95.     }
96.     printf("\n");
97. }
```

```

1. /*
2. * C Program to Support Infinite Precision Arithmetic & Store a
3. * Number as a List of Digits
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <ctype.h>
8.
9. struct node
10. {
11.     int num;
12.     struct node *next;
13. };
14.
15. int feednumber(struct node **);
16. void release(struct node **);
17. void display(struct node *);
18.
19. int main()
20. {
21.     struct node *p = NULL;
22.     int pcount = 0, qcount = 0;
23.
24.     printf("Enter number of any length\n");
25.     pcount = feednumber(&p);
26.     printf("Number of integers entered are: %d\n", pcount);
27.     printf("Displaying the number entered:\n");
28.     display(p);
29.     release(&p);
30.
31.     return 0;
32. }
33.
34. /*Function to create nodes of numbers*/
35. int feednumber(struct node **head)
36. {
37.     char ch, dig;
```

```
38.     int count = 0;
39.     struct node *temp, *rear = NULL;
40.
41.     ch = getchar();
42.     while (ch != '\n')
43.     {
44.         dig = atoi(&ch);
45.         temp = (struct node *)malloc(sizeof(struct node));
46.         temp->num = dig;
47.         temp->next = NULL;
48.         count++;
49.         if ((*head) == NULL)
50.         {
51.             *head = temp;
52.             rear = temp;
53.         }
54.         else
55.         {
56.             rear->next = temp;
57.             rear = rear->next;
58.         }
59.         ch = getchar();
60.     }
61.
62.     return count;
63. }
64.
65./*Function to display the list of numbers*/
66.void display (struct node *head)
67.{
68.    while (head != NULL)
69.    {
70.        printf("%d", head->num);
71.        head = head->next;
72.    }
73.    printf("\n");
74.}
75.
76./*Function to free the allocated list of numbers*/
77.void release (struct node **head)
78.{
79.    struct node *temp = *head;
80.
```

```

81.     while ((*head) != NULL)
82.     {
83.         (*head) = (*head)->next;
84.         free(temp);
85.         temp = *head;
86.     }
87. }
```

## (7.) C Programming Examples on Stacks & Queues

### 1. C Examples on Stack Implementation

```

/*
 * C program to implement stack. Stack is a LIFO data structure.
 * Stack operations: PUSH(insert operation), POP(Delete operation)
 * and Display stack.
 */
#include <stdio.h>
#define MAXSIZE 5

struct stack
{
    int stk[MAXSIZE];
    int top;
};

typedef struct stack STACK;
STACK s;

void push(void);
int pop(void);
void display(void);

void main ()
{
    int choice;
    int option = 1;
    s.top = -1;

    printf ("STACK OPERATION\n");
    while (option)
    {
        printf ("-----\n");
```

```

printf ("      1    -->    PUSH          \n");
printf ("      2    -->    POP           \n");
printf ("      3    -->    DISPLAY        \n");
printf ("      4    -->    EXIT          \n");
printf ("-----\n");

printf ("Enter your choice\n");
scanf ("%d", &choice);
switch (choice)
{
case 1:
    push();
    break;
case 2:
    pop();
    break;
case 3:
    display();
    break;
case 4:
    return;
}
fflush (stdin);
printf ("Do you want to continue(Type 0 or 1)?\n");
scanf ("%d", &option);
}

/* Function to add an element to the stack */
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("Enter the element to be pushed\n");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
}

```

```

        return;
    }
/* Function to delete an element from the stack */
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %dn", s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
/* Function to display the status of the stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\n The status of the stack is \n");
        for (i = s.top; i >= 0; i--)
        {
            printf ("%d\n", s.stk[i]);
        }
    }
    printf ("\n");
}

```

1. /\*
2. \* C Program to Reverse a Stack using Recursion
3. \*/

```
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13. void generate(struct node **);
14. void display(struct node *);
15. void stack_reverse(struct node **, struct node **);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     printf("\nThe sequence of contents in stack\n");
24.     display(head);
25.     printf("\nInversing the contents of the stack\n");
26.     if (head != NULL)
27.     {
28.         stack_reverse(&head, &(head->next));
29.     }
30.     printf("\nThe contents in stack after reversal\n");
31.     display(head);
32.     delete(&head);
33.
34.     return 0;
35. }
36.
37. void stack_reverse(struct node **head, struct node **head_next)
38. {
39.     struct node *temp;
40.
41.     if (*head_next != NULL)
42.     {
43.         temp = (*head_next)->next;
44.         (*head_next)->next = (*head);
45.         *head = *head_next;
46.         *head_next = temp;
```

```
47.         stack_reverse(head, head_next);
48.     }
49. }
50.
51. void display(struct node *head)
52. {
53.     if (head != NULL)
54.     {
55.         printf("%d ", head->a);
56.         display(head->next);
57.     }
58. }
59.
60. void generate(struct node **head)
61. {
62.     int num, i;
63.     struct node *temp;
64.
65.     printf("Enter length of list: ");
66.     scanf("%d", &num);
67.     for (i = num; i > 0; i--)
68.     {
69.         temp = (struct node *)malloc(sizeof(struct node));
70.         temp->a = i;
71.         if (*head == NULL)
72.         {
73.             *head = temp;
74.             (*head)->next = NULL;
75.         }
76.         else
77.         {
78.             temp->next = *head;
79.             *head = temp;
80.         }
81.     }
82. }
83.
84. void delete(struct node **head)
85. {
86.     struct node *temp;
87.     while (*head != NULL)
88.     {
89.         temp = *head;
```

```

90.         *head = (*head)->next;
91.         free(temp);
92.     }
93. }
```

```

1. /*
2.  * C Program to Reverse a Stack without using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13. void generate(struct node **);
14. void display(struct node *);
15. void stack_reverse(struct node **);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     printf("\nThe sequence of contents in stack\n");
24.     display(head);
25.     printf("\nInversing the contents of the stack\n");
26.     stack_reverse(&head);
27.     printf("\nThe contents in stack after reversal\n");
28.     display(head);
29.     delete(&head);
30.     return 0;
31. }
32.
33. void stack_reverse(struct node **head)
34. {
35.     struct node *temp, *prev;
36.
37.     if (*head == NULL)
```

```
38.     {
39.         printf("Stack does not exist\n");
40.     }
41.     else if ((*head)->next == NULL)
42.     {
43.         printf("Single node stack reversal brings no difference\n");
44.     }
45.     else if ((*head)->next->next == NULL)
46.     {
47.         (*head)->next->next = *head;
48.         *head = (*head)->next;
49.         (*head)->next->next = NULL;
50.     }
51.     else
52.     {
53.         prev = *head;
54.         temp = (*head)->next;
55.         *head = (*head)->next->next;
56.         prev->next = NULL;
57.         while ((*head)->next != NULL)
58.         {
59.             temp->next = prev;
60.             prev = temp;
61.             temp = *head;
62.             *head = (*head)->next;
63.         }
64.         temp->next = prev;
65.         (*head)->next = temp;
66.     }
67. }
68.
69. void display(struct node *head)
70. {
71.     if (head != NULL)
72.     {
73.         printf("%d ", head->a);
74.         display(head->next);
75.     }
76. }
77.
78. void generate(struct node **head)
79. {
80.     int num, i;
```

```

81.     struct node *temp;
82.
83.     printf("Enter length of list: ");
84.     scanf("%d", &num);
85.     for (i = num; i > 0; i--)
86.     {
87.         temp = (struct node *)malloc(sizeof(struct node));
88.         temp->a = i;
89.         if (*head == NULL)
90.         {
91.             *head = temp;
92.             (*head)->next = NULL;
93.         }
94.         else
95.         {
96.             temp->next = *head;
97.             *head = temp;
98.         }
99.     }
100. }
101.
102. void delete(struct node **head)
103. {
104.     struct node *temp;
105.     while (*head != NULL)
106.     {
107.         temp = *head;
108.         *head = (*head)->next;
109.         free(temp);
110.     }
111. }
```

```

1. //This is a C Program to Implement two Stacks using a Single Array & Check for
2. //Overflow & Underflow
3. #include <stdio.h>
4.
5.
6. int ar[SIZE];
7. int top1 = -1;
8. int top2 = SIZE;
9.
```

```
10.//Functions to push data
11.void push_stack1 (int data)
12.{
13.    if (top1 < top2 - 1)
14.    {
15.        ar[++top1] = data;
16.    }
17.    else
18.    {
19.        printf ("Stack Full! Cannot Push\n");
20.    }
21.}
22.void push_stack2 (int data)
23.{
24.    if (top1 < top2 - 1)
25.    {
26.        ar[--top2] = data;
27.    }
28.    else
29.    {
30.        printf ("Stack Full! Cannot Push\n");
31.    }
32.}
33.
34.//Functions to pop data
35.void pop_stack1 ()
36.{
37.    if (top1 >= 0)
38.    {
39.        int popped_value = ar[top1--];
40.        printf ("%d is being popped from Stack 1\n", popped_value);
41.    }
42.    else
43.    {
44.        printf ("Stack Empty! Cannot Pop\n");
45.    }
46.}
47.void pop_stack2 ()
48.{
49.    if (top2 < SIZE)
50.    {
51.        int popped_value = ar[top2++];
52.        printf ("%d is being popped from Stack 2\n", popped_value);
```

```
53. }
54. else
55. {
56.     printf ("Stack Empty! Cannot Pop\n");
57. }
58. }
59.

60.//Functions to Print Stack 1 and Stack 2
61.void print_stack1 ()
62.{
63.    int i;
64.    for (i = top1; i >= 0; --i)
65.    {
66.        printf ("%d ", ar[i]);
67.    }
68.    printf ("\n");
69.}

70.void print_stack2 ()
71.{
72.    int i;
73.    for (i = top2; i < SIZE; ++i)
74.    {
75.        printf ("%d ", ar[i]);
76.    }
77.    printf ("\n");
78.}

79.

80.int main()
81.{
82.    int ar[SIZE];
83.    int i;
84.    int num_of_ele;
85.

86.    printf ("We can push a total of 10 values\n");
87.

88.    //Number of elements pushed in stack 1 is 6
89.    //Number of elements pushed in stack 2 is 4
90.

91.    for (i = 1; i <= 6; ++i)
92.    {
93.        push_stack1 (i);
94.        printf ("Value Pushed in Stack 1 is %d\n", i);
95.    }
```

```

96.  for (i = 1; i <= 4; ++i)
97.  {
98.      push_stack2 (i);
99.      printf ("Value Pushed in Stack 2 is %d\n", i);
100.     }
101.
102.     //Print Both Stacks
103.     print_stack1 ();
104.     print_stack2 ();
105.
106.     //Pushing on Stack Full
107.     printf ("Pushing Value in Stack 1 is %d\n", 11);
108.     push_stack1 (11);
109.
110.    //Popping All Elements From Stack 1
111.    num_of_ele = top1 + 1;
112.    while (num_of_ele)
113.    {
114.        pop_stack1 ();
115.        --num_of_ele;
116.    }
117.
118.    //Trying to Pop From Empty Stack
119.    pop_stack1 ();
120.
121.    return 0;
122. }
```

```

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

struct node
{
    int info;
    struct node *ptr;
}*top,*top1,*temp;

int topelement();
void push(int data);
void pop();
void empty();
```

```

void display();
void destroy();
void stack_count();
void create();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Push");
    printf("\n 2 - Pop");
    printf("\n 3 - Top");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Dipslay");
    printf("\n 7 - Stack Count");
    printf("\n 8 - Destroy stack");

    create();

    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);

        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                push(no);
                break;
            case 2:
                pop();
                break;
            case 3:
                if (top == NULL)
                    printf("No elements in stack");
                else
                {
                    e = topelement();

```

```

        printf("\n Top element : %d", e);
    }
    break;
case 4:
    empty();
    break;
case 5:
    exit(0);
case 6:
    display();
    break;
case 7:
    stack_count();
    break;
case 8:
    destroy();
    break;
default :
    printf(" Wrong choice, Please enter correct choice  ");
    break;
}
}
}

/* Create empty stack */
void create()
{
    top = NULL;
}

/* Count stack elements */
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}

/* Push data into stack */
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
    }
    else
        top->ptr = (struct node *)malloc(1*sizeof(struct node));
    top->data = data;
}
```

```

        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}

/* Display stack elements */
void display()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }

    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
}

/* Pop Operation on stack */
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
}

```

```

    free(top);
    top = top1;
    count--;
}

/* Return top element */
int topelement()
{
    return(top->info);
}

/* Check if stack is empty or not */
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}

/* Destroy entire stack */
void destroy()
{
    top1 = top;

    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;

    printf("\n All stack elements destroyed");
    count = 0;
}

```

## 2. C Examples on Queue Implementation

1. /\*

```
2.  * C Program to Implement a Queue using an Array
3.  */
4. #include <stdio.h>
5.
6. #define MAX 50
7. int queue_array[MAX];
8. int rear = - 1;
9. int front = - 1;
10.main()
11.{
12.    int choice;
13.    while (1)
14.    {
15.        printf("1.Insert element to queue \n");
16.        printf("2.Delete element from queue \n");
17.        printf("3.Display all elements of queue \n");
18.        printf("4.Quit \n");
19.        printf("Enter your choice : ");
20.        scanf("%d", &choice);
21.        switch (choice)
22.        {
23.            case 1:
24.                insert();
25.                break;
26.            case 2:
27.                delete();
28.                break;
29.            case 3:
30.                display();
31.                break;
32.            case 4:
33.                exit(1);
34.            default:
35.                printf("Wrong choice \n");
36.        } /*End of switch*/
37.    } /*End of while*/
38. } /*End of main()*/
39.insert()
40.{
41.    int add_item;
42.    if (rear == MAX - 1)
43.        printf("Queue Overflow \n");
44.    else
```

```

45.     {
46.         if (front == - 1)
47.             /*If queue is initially empty */
48.             front = 0;
49.             printf("Inset the element in queue : ");
50.             scanf("%d", &add_item);
51.             rear = rear + 1;
52.             queue_array[rear] = add_item;
53.     }
54. } /*End of insert()*/
55.
56. delete()
57. {
58.     if (front == - 1 || front > rear)
59.     {
60.         printf("Queue Underflow \n");
61.         return ;
62.     }
63.     else
64.     {
65.         printf("Element deleted from queue is : %d\n", queue_array[front]);
66.         front = front + 1;
67.     }
68. } /*End of delete() */
69. display()
70. {
71.     int i;
72.     if (front == - 1)
73.         printf("Queue is empty \n");
74.     else
75.     {
76.         printf("Queue is : \n");
77.         for (i = front; i <= rear; i++)
78.             printf("%d ", queue_array[i]);
79.         printf("\n");
80.     }
81. } /*End of display() */

```

*Linked List*

```

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

```

```
struct node
{
    int info;
    struct node *ptr;
}*front,*rear,*temp,*front1;

int frontelement();
void enq(int data);
void deq();
void empty();
void display();
void create();
void queuesize();

int count = 0;

void main()
{
    int no, ch, e;

    printf("\n 1 - Enque");
    printf("\n 2 - Deque");
    printf("\n 3 - Front element");
    printf("\n 4 - Empty");
    printf("\n 5 - Exit");
    printf("\n 6 - Display");
    printf("\n 7 - Queue size");
    create();
    while (1)
    {
        printf("\n Enter choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("Enter data : ");
                scanf("%d", &no);
                enq(no);
                break;
            case 2:
                deq();
                break;
        }
    }
}
```

```

        case 3:
            e = frontelement();
            if (e != 0)
                printf("Front element : %d", e);
            else
                printf("\n No front element in Queue as queue is empty");
            break;
        case 4:
            empty();
            break;
        case 5:
            exit(0);
        case 6:
            display();
            break;
        case 7:
            queuesize();
            break;
        default:
            printf("Wrong choice, Please enter correct choice   ");
            break;
    }
}
}

/* Create an empty queue */
void create()
{
    front = rear = NULL;
}

/* Returns queue size */
void queuesize()
{
    printf("\n Queue size : %d", count);
}

/* Enqueing the queue */
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear->data = data;
        rear->next = NULL;
    }
    else
    {
        struct node *temp = (struct node *)malloc(sizeof(struct node));
        temp->data = data;
        temp->next = rear;
        rear = temp;
    }
}

```

```

        rear->ptr = NULL;
        rear->info = data;
        front = rear;
    }
else
{
    temp=(struct node *)malloc(1*sizeof(struct node));
    rear->ptr = temp;
    temp->info = data;
    temp->ptr = NULL;

    rear = temp;
}
count++;
}

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue is empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d ", front1->info);
        front1 = front1->ptr;
    }
    if (front1 == rear)
        printf("%d", front1->info);
}

/* Dequeueing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
        printf("\n Error: Trying to display elements from empty queue");
    }
}

```

```

        return;
    }
    else
    {
        if (front1->ptr != NULL)
        {
            front1 = front1->ptr;
            printf("\n Dequeued value : %d", front->info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n Dequeued value : %d", front->info);
            free(front);
            front = NULL;
            rear = NULL;
        }
        count--;
    }

/* Returns the front element of queue */
int frontelement()
{
    if ((front != NULL) && (rear != NULL))
        return(front->info);
    else
        return 0;
}

/* Display if queue is empty or not */
void empty()
{
    if ((front == NULL) && (rear == NULL))
        printf("\n Queue empty");
    else
        printf("Queue not empty");
}

```

1. /\*
2. \* C Program to Implement Priority Queue to Add and Delete Elements
3. \*/
4. #include <stdio.h>

```
5. #include <stdlib.h>
6.
7. #define MAX 5
8.
9. void insert_by_priority(int);
10. void delete_by_priority(int);
11. void create();
12. void check(int);
13. void display_pqueue();
14.
15. int pri_que[MAX];
16. int front, rear;
17.
18. void main()
19. {
20.     int n, ch;
21.
22.     printf("\n1 - Insert an element into queue");
23.     printf("\n2 - Delete an element from queue");
24.     printf("\n3 - Display queue elements");
25.     printf("\n4 - Exit");
26.
27.     create();
28.
29.     while (1)
30.     {
31.         printf("\nEnter your choice : ");
32.         scanf("%d", &ch);
33.
34.         switch (ch)
35.         {
36.             case 1:
37.                 printf("\nEnter value to be inserted : ");
38.                 scanf("%d", &n);
39.                 insert_by_priority(n);
40.                 break;
41.             case 2:
42.                 printf("\nEnter value to delete : ");
43.                 scanf("%d", &n);
44.                 delete_by_priority(n);
45.                 break;
46.             case 3:
47.                 display_pqueue();
```

```

48.         break;
49.     case 4:
50.         exit(0);
51.     default:
52.         printf("\nChoice is incorrect, Enter a correct choice");
53.     }
54. }
55. }
56.
57./* Function to create an empty priority queue */
58.void create()
59.{
60.    front = rear = -1;
61.}
62.
63./* Function to insert value into priority queue */
64.void insert_by_priority(int data)
65.{
66.    if (rear >= MAX - 1)
67.    {
68.        printf("\nQueue overflow no more elements can be inserted");
69.        return;
70.    }
71.    if ((front == -1) && (rear == -1))
72.    {
73.        front++;
74.        rear++;
75.        pri_que[rear] = data;
76.        return;
77.    }
78.    else
79.        check(data);
80.    rear++;
81.}
82.
83./* Function to check priority and place element */
84.void check(int data)
85.{
86.    int i,j;
87.
88.    for (i = 0; i <= rear; i++)
89.    {
90.        if (data >= pri_que[i])

```

```

91.         {
92.             for (j = rear + 1; j > i; j--)
93.             {
94.                 pri_que[j] = pri_que[j - 1];
95.             }
96.             pri_que[i] = data;
97.             return;
98.         }
99.     }
100.    pri_que[i] = data;
101. }
102.

103. /* Function to delete an element from queue */
104. void delete_by_priority(int data)
105. {
106.     int i;

107.     if ((front == -1) && (rear == -1))
108.     {
109.         printf("\nQueue is empty no elements to delete");
110.         return;
111.     }

112.     for (i = 0; i <= rear; i++)
113.     {
114.         if (data == pri_que[i])
115.         {
116.             for (; i < rear; i++)
117.             {
118.                 pri_que[i] = pri_que[i + 1];
119.             }
120.             pri_que[i] = -99;
121.             rear--;
122.

123.             if (rear == -1)
124.                 front = -1;
125.             return;
126.         }
127.     }
128.     printf("\n%d not found in queue to delete", data);
129. }
130. }
131. }
132. }
133.

```

```

134.     /* Function to display queue elements */
135.     void display_pqueue()
136.     {
137.         if ((front == -1) && (rear == -1))
138.         {
139.             printf("\nQueue is empty");
140.             return;
141.         }
142.
143.         for (; front <= rear; front++)
144.         {
145.             printf(" %d ", pri_que[front]);
146.         }
147.
148.         front = 0;
149.     }

```

### 3. C Examples on String Implementation

```

1. /*
2.  * C Program to Check String is Palindrome using Stack.
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void push(char);
8. char pop();
9.
10. char stack[100];
11. int top = -1;
12.
13. void main()
14. {
15.     char str[100];
16.     int i, count = 0, len;
17.
18.     printf("Enter string to check it is palindrome or not : ");
19.     scanf("%s", str);
20.
21.     len = strlen(str);
22.
23.     for (i = 0; i < len; i++)

```

```

24.     {
25.         push(str[i]);
26.     }
27.
28.     for (i = 0; i < len; i++)
29.     {
30.         if (str[i] == pop())
31.             count++;
32.     }
33.
34.     if (count == len)
35.         printf("%s is a Palindrome string\n", str);
36.     else
37.         printf("%s is not a palindrome string\n", str);
38. }
39.
40./* Function to push character into stack */
41 void push(char c)
42 {
43     stack[++top] = c;
44. }
45.
46./* Function to pop the top character from stack */
47 char pop()
48 {
49     return(stack[top--]);
50. }
```

```

1. /*
2. * C Program to Check if Expression is correctly Parenthesized
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7.
8. int top = -1;
9. char stack[100];
10.
11.// function prototypes
12 void push(char);
13 void pop();
14 void find_top();
```

```
15.  
16. void main()  
17. {  
18.     int i;  
19.     char a[100];  
20.     printf("enter expression\n");  
21.     scanf("%s", &a);  
22.     for (i = 0; a[i] != '\0'; i++)  
23.     {  
24.         if (a[i] == '(')  
25.         {  
26.             push(a[i]);  
27.         }  
28.         else if (a[i] == ')')  
29.         {  
30.             pop();  
31.         }  
32.     }  
33.     find_top();  
34. }  
35.  
36.// to push elements in stack  
37. void push(char a)  
38. {  
39.     stack[top] = a;  
40.     top++;  
41. }  
42.  
43.// to pop elements from stack  
44. void pop()  
45. {  
46.     if (top == -1)  
47.     {  
48.         printf("expression is invalid\n");  
49.         exit(0);  
50.     }  
51.     else  
52.     {  
53.         top--;  
54.     }  
55. }  
56.  
57.// to find top element of stack
```

```

58. void find_top()
59. {
60.     if (top == -1)
61.         printf("\nexpression is valid\n");
62.     else
63.         printf("\nexpression is invalid\n");
64. }
```

```

1. /*
2.  * C Program to Identify whether the String is Palindrome or not using Stack
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #define MAX 50
8.
9. int top = -1, front = 0;
10. int stack[MAX];
11. void push(char);
12. void pop();
13.
14. void main()
15. {
16.     int i, choice;
17.     char s[MAX], b;
18.     while (1)
19.     {
20.         printf("1-enter string\n2-exit\n");
21.         printf("enter your choice\n");
22.         scanf("%d", &choice);
23.         switch (choice)
24.         {
25.             case 1:
26.                 printf("Enter the String\n");
27.                 scanf("%s", s);
28.                 for (i = 0;s[i] != '\0';i++)
29.                 {
30.                     b = s[i];
31.                     push(b);
32.                 }
33.                 for (i = 0;i < (strlen(s) / 2);i++)
34.                 {
```

```

35.         if (stack[top] == stack[front])
36.         {
37.             pop();
38.             front++;
39.         }
40.     else
41.     {
42.         printf("%s is not a palindrome\n", s);
43.         break;
44.     }
45. }
46. if ((strlen(s) / 2) == front)
47.     printf("%s is palindrome\n", s);
48. front = 0;
49. top = -1;
50. break;
51. case 2:
52.     exit(0);
53. default:
54.     printf("enter correct choice\n");
55. }
56. }
57. }
58.
59. /* to push a character into stack */
60. void push(char a)
61. {
62.     top++;
63.     stack[top] = a;
64. }
65.
66. /* to delete an element in stack */
67. void pop()
68. {
69.     top--;
70. }

```

#### 4. C Examples on Queue Implementation using Other Data Structures

```

1. /*
2. * C Program to Implement various Queue Functions using Dynamic Memory Allocation
3. */

```

```
4. #include <stdio.h>
5. #include <stdlib.h>
6. #define MAX 10
7.
8. struct node
9. {
10.     int data;
11.     struct node *link;
12. }*front, *rear;
13.
14. // function prototypes
15. void insert();
16. void delete();
17. void queue_size();
18. void check();
19. void first_element();
20.
21. void main()
22. {
23.     int choice, value;
24.
25.     while(1)
26.     {
27.         printf("enter the choice \n");
28.         printf("1 : create an empty queue \n2 : Insert element\n");
29.         printf("3 : Dequeue an element \n4 : Check if empty\n");
30.         printf("5. Get the first element of the queue\n");
31.         printf("6. Get the number of entries in the queue\n");
32.         printf("7. Exit\n");
33.         scanf("%d", &choice);
34.         switch (choice)      // menu driven program
35.         {
36.             case 1:
37.                 printf("Empty queue is created with a capacity of %d\n", MAX);
38.                 break;
39.             case 2:
40.                 insert();
41.                 break;
42.             case 3:
43.                 delete();
44.                 break;
45.             case 4:
46.                 check();
```

```
47.         break;
48.     case 5:
49.         first_element();
50.         break;
51.     case 6:
52.         queue_size();
53.         break;
54.     case 7:
55.         exit(0);
56.     default:
57.         printf("wrong choice\n");
58.         break;
59.     }
60. }
61. }
62.
63.// to insert elements in queue
64.void insert()
65.{
66.    struct node *temp;
67.
68.    temp = (struct node*)malloc(sizeof(struct node));
69.    printf("Enter value to be inserted \n");
70.    scanf("%d", &temp->data);
71.    temp->link = NULL;
72.    if (rear == NULL)
73.    {
74.        front = rear = temp;
75.    }
76.    else
77.    {
78.        rear->link = temp;
79.        rear = temp;
80.    }
81. }
82.
83.// delete elements from queue
84.void delete()
85.{
86.    struct node *temp;
87.
88.    temp = front;
89.    if (front == NULL)
```

```

90.    {
91.        printf("queue is empty \n");
92.        front = rear = NULL;
93.    }
94. else
95. {
96.     printf("deleted element is %d\n", front->data);
97.     front = front->link;
98.     free(temp);
99. }
100. }

101.

102. // check if queue is empty or not
103. void check()
104. {
105.     if (front == NULL)
106.         printf("\nQueue is empty\n");
107.     else
108.         printf("***** Elements are present in the queue
*****\n");
109. }

110.

111. // returns first element of queue
112. void first_element()
113. {
114.     if (front == NULL)
115.     {
116.         printf("***** The queue is empty *****\n");
117.     }
118.     else
119.         printf("***** The front element is %d *****\n",
front->data);
120. }

121.

122. // returns number of entries and displays the elements in queue
123. void queue_size()
124. {
125.     struct node *temp;
126.
127.     temp = front;
128.     int cnt = 0;
129.     if (front == NULL)
130.     {

```

```

131.         printf(" queue empty \n");
132.     }
133.     while (temp)
134.     {
135.         printf("%d ", temp->data);
136.         temp = temp->link;
137.         cnt++;
138.     }
139.     printf("***** size of queue is %d ***** \n", cnt);
140. }
```

```

1. /*
2. * C Program to Implement Queue Functions Using Arrays and Macros
3. */
4. #include <stdio.h>
5. #include<stdlib.h>
6.
7. /* Macro Definition */
8. #define MAX 10
9. #define EMPTY "QUEUE EMPTY"
10.#define ISFULL rear >= MAX - 1
11.#define FULL "QUEUE FULL"
12.#define ISEMPY rear == -1
13.
14./* Global Variable Declaration */
15.int queue[MAX], front = 0, rear = -1;
16.
17./* Function Prototypes */
18.void insert_rear();
19.void delete_front();
20.void display_queue();
21.void empty_queue();
22.void front_ele();
23.int queue_size();
24.void destroy();
25.
26.void main()
27.{
28.    int choice, n, flag = 0;
29.    char ch;
30.
31.    do
```

```
32.     {
33.         printf("MENU\n");
34.         printf("Enter 1 to INSERT an element in the queue\n");
35.         printf("Enter 2 to DELETE an element in the queue\n");
36.         printf("Enter 3 to DISPLAY the elements of the queue\n");
37.         printf("Enter 4 to CHECK if the queue is EMPTY\n");
38.         printf("Enter 5 to KNOW the FIRST element of the queue\n");
39.         printf("Enter 6 to KNOW the queue SIZE\n");
40.         printf("Enter 7 to Destroy the Queue\n");
41.         printf("Enter 8 to EXIT the program\n");
42.         printf("Enter your Choice:");
43.         scanf("%d", &choice);
44.         switch(choice)
45.         {
46.             case 1:
47.                 insert_rear();
48.                 break;
49.             case 2:
50.                 delete_front();
51.                 break;
52.             case 3:
53.                 display_queue();
54.                 break;
55.             case 4:
56.                 empty_queue();
57.                 break;
58.             case 5:
59.                 front_ele();
60.                 break;
61.             case 6:
62.                 n = queue_size();
63.                 printf("\nthe queue size is: %d", n);
64.                 break;
65.             case 7:
66.                 destroy();
67.                 flag = 1;
68.                 break;
69.             case 8:
70.                 exit(0);
71.                 break;
72.             default:
73.                 printf("WRONG CHOICE\n");
74.         }
```

```

75.         printf("\nDo you want to continue:");
76.         scanf(" %c", &ch);
77.     } while(ch == 'y' || ch == 'Y');
78.     if (flag == 0)
79.     {
80.         destroy();
81.     }
82. }
83.
84./* Code to Insert the element in Queue */
85.void insert_rear()
86.{
87.    int val;
88.
89.    if (ISFULL)
90.    {
91.        printf(FULL);
92.    }
93.    else
94.    {
95.        printf("\nEnter the value you want to insert in the queue:");
96.        scanf("%d", &val);
97.        rear++;
98.        queue[rear] = val;
99.        printf("\nElement successfully inserted in the queue");
100.       }
101.   }
102.
103. /* Code to Delete the element in Queue */
104. void delete_front()
105. {
106.     if (ISEMPTY)
107.     {
108.         printf(EMPTY);
109.     }
110.     else
111.     {
112.         printf("\nThe deleted element is: %d", queue[front]);
113.         front++;
114.     }
115.   }
116.
117. /* Code to Display the Elements of Queue */

```

```
118.     void display_queue()
119.     {
120.         int i;
121.
122.         if (ISEMPTY)
123.         {
124.             printf(EMPTY);
125.         }
126.         else
127.         {
128.             for (i = front;i <= rear;i++)
129.             {
130.                 printf("%d->", queue[i]);
131.             }
132.         }
133.     }
134.
135. /* Code to Check the Queue is Empty or Not */
136. void empty_queue()
137. {
138.     if (ISEMPTY)
139.     {
140.         printf(EMPTY);
141.     }
142.     else
143.     {
144.         printf("\nTHE QUEUE has elements\n");
145.     }
146. }
147.
148.
149. /* Code to Check the First element of Queue */
150. void front_ele()
151. {
152.     if (ISEMPTY)
153.     {
154.         printf(EMPTY);
155.     }
156.     else
157.     {
158.         printf("The first element of the queue is: %d", queue[front]);
159.     }
160. }
```

```
161.
162.     /* Code to Check the Size of Queue */
163.     int queue_size()
164.     {
165.         int i = 0, count = 0;
166.
167.         if (ISEMPTY)
168.         {
169.             printf(EMPTY);
170.         }
171.         else
172.         {
173.             for (i = front;i <= rear;i++)
174.             {
175.                 count++;
176.             }
177.         }
178.         return count;
179.     }
180.
181.     /* Code to destroy the queue */
182.     void destroy()
183.     {
184.         int size, i;
185.
186.         if (ISEMPTY)
187.         {
188.             printf("EMPTY QUEUE CANNOT BE DESTROYED");
189.         }
190.         else
191.         {
192.             size = queue_size();
193.
194.             for (i = 0;i < size;i++)
195.             {
196.                 front++;
197.             }
198.             front = 0;
199.             rear = -1;
200.             printf("\n\nQUEUE DESTROYED");
201.         }
202.     }
```

```
1. /*
2.  * C Program to Implement Queues using Stacks
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void push1(int);
8. void push2(int);
9. int pop1();
10.int pop2();
11 void enqueue();
12 void dequeue();
13 void display();
14 void create();
15.
16.int st1[100], st2[100];
17.int top1 = -1, top2 = -1;
18.int count = 0;
19.
20 void main()
21.{
22.    int ch;
23.
24.    printf("\n1 - Enqueue element into queue");
25.    printf("\n2 - Dequeue element from queue");
26.    printf("\n3 - Display from queue");
27.    printf("\n4 - Exit");
28.    create();
29.    while (1)
30.    {
31.        printf("\nEnter choice");
32.        scanf("%d", &ch);
33.        switch (ch)
34.        {
35.            case 1:
36.                enqueue();
37.                break;
38.            case 2:
39.                dequeue();
40.                break;
41.            case 3:
```

```
42.         display();
43.         break;
44.     case 4:
45.         exit(0);
46.     default:
47.         printf("Wrong choice");
48.     }
49. }
50. }
51.
52./*Function to create a queue*/
53.void create()
54.{
55.    top1 = top2 = -1;
56.}
57.
58./*Function to push the element on to the stack*/
59:void push1(int data)
60.{
61.    st1[++top1] = data;
62.}
63.
64./*Function to pop the element from the stack*/
65.int pop1()
66.{
67.    return(st1[top1--]);
68.}
69.
70./*Function to push an element on to stack*/
71.void push2(int data)
72.{
73.    st2[++top2] = data;
74.}
75.
76./*Function to pop an element from th stack*/
77.
78.int pop2()
79.{
80.    return(st2[top2--]);
81.}
82.
83./*Function to add an element into the queue using stack*/
84.void enqueue()
```

```

85. {
86.     int data, i;
87.
88.     printf("Enter data into queue");
89.     scanf("%d", &data);
90.     push1(data);
91.     count++;
92. }
93.
94. /*Function to delete an element from the queue using stack*/
95.
96. void dequeue()
97. {
98.     int i;
99.
100.    for (i = 0;i <= count;i++)
101.    {
102.        push2(pop1());
103.    }
104.    pop2();
105.    count--;
106.    for (i = 0;i <= count;i++)
107.    {
108.        push1(pop2());
109.    }
110. }
111.
112. /*Function to display the elements in the stack*/
113.
114. void display()
115. {
116.     int i;
117.
118.     for (i = 0;i <= top1;i++)
119.     {
120.         printf(" %d ", st1[i]);
121.     }
122. }
```

```

1. /*
2. * C Program to Implement Queues using Stacks
3. */
```

```
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void push1(int);
8. void push2(int);
9. int pop1();
10. int pop2();
11. void enqueue();
12. void dequeue();
13. void display();
14. void create();
15.
16. int st1[100], st2[100];
17. int top1 = -1, top2 = -1;
18. int count = 0;
19.
20. void main()
21. {
22.     int ch;
23.
24.     printf("\n1 - Enqueue element into queue");
25.     printf("\n2 - Dequeue element from queue");
26.     printf("\n3 - Display from queue");
27.     printf("\n4 - Exit");
28.     create();
29.     while (1)
30.     {
31.         printf("\nEnter choice");
32.         scanf("%d", &ch);
33.         switch (ch)
34.         {
35.             case 1:
36.                 enqueue();
37.                 break;
38.             case 2:
39.                 dequeue();
40.                 break;
41.             case 3:
42.                 display();
43.                 break;
44.             case 4:
45.                 exit(0);
46.             default:
```

```
47.         printf("Wrong choice");
48.     }
49. }
50. }
51.
52./*Function to create a queue*/
53.void create()
54.{
55.    top1 = top2 = -1;
56.}
57.
58./*Function to push the element on to the stack*/
59.void push1(int data)
60.{
61.    st1[++top1] = data;
62.}
63.
64./*Function to pop the element from the stack*/
65.int pop1()
66.{
67.    return(st1[top1--]);
68.}
69.
70./*Function to push an element on to stack*/
71.void push2(int data)
72.{
73.    st2[++top2] = data;
74.}
75.
76./*Function to pop an element from th stack*/
77.
78.int pop2()
79.{
80.    return(st2[top2--]);
81.}
82.
83./*Function to add an element into the queue using stack*/
84.void enqueue()
85.{
86.    int data, i;
87.
88.    printf("Enter data into queue");
89.    scanf("%d", &data);
```

```

90.     push1(data);
91.     count++;
92. }
93.
94. /*Function to delete an element from the queue using stack*/
95.
96. void dequeue()
97. {
98.     int i;
99.
100.    for (i = 0;i <= count;i++)
101.    {
102.        push2(pop1());
103.    }
104.    pop2();
105.    count--;
106.    for (i = 0;i <= count;i++)
107.    {
108.        push1(pop2());
109.    }
110. }
111.
112. /*Function to display the elements in the stack*/
113.
114. void display()
115. {
116.     int i;
117.
118.     for (i = 0;i <= top1;i++)
119.     {
120.         printf(" %d ", st1[i]);
121.     }
122. }
```

## 5. C Examples on Other Stack Implementations

```

1. /*
2. * C Program to Illustrate Stack Operations using MACROS
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
```

```
7. #define MAX 5
8. #define EMPTY "Stack is Empty"
9. #define OVERFLOW "Stack Overflow"
10. #define ISOVERFLOW top >= MAX - 1 /*Macro to find whether the stack is full*/
11. #define ISEMPTY top == -1      /*Macro to find whether the stack is empty*/
12.
13. void push(int);
14. void pop();
15. void display();
16. void stacksize();
17. void destroy();
18. void stackfull();
19.
20. int top = -1;
21. int stack[MAX];
22.
23. void main()
24. {
25.     int choice, value;
26.
27.     while (1)
28.     {
29.         printf("Enter Your choice:\n");
30.         printf("1.PUSH\n2.POP\n3.DISPLAY\n4.STACKSIZE\n5.DESTROY\n6.SATCKFULL
CHECK\n7.EXIT");
31.         scanf("%d", &choice);
32.         switch (choice)
33.         {
34.             case 1:
35.                 printf("enter the value to be pushed on to the stack");
36.                 scanf("%d", &value);
37.                 push(value);
38.                 continue;
39.             case 2:
40.                 pop();
41.                 continue;
42.             case 3:
43.                 display();
44.                 continue;
45.             case 4:
46.                 stacksize();
47.                 continue;
48.             case 5:
```

```
49.         destroy();
50.         continue;
51.     case 6:
52.         stackfull();
53.         continue;
54.     case 7:
55.         exit(0);
56.     default:
57.         printf("YOU HAVE ENTERD A WRONG CHOICE");
58.     }
59. }
60. }

61.

62./*Function to add an element into the stack*/
63.void push(int value)
64.{
65.    if (ISOVERFLOW)
66.    {
67.        printf(OVERFLOW);
68.        return;
69.    }
70.    top++;
71.    stack[top] = value;
72. }

73.

74./*Function to delete an element from the stack*/
75.void pop()
76.{
77.    if (ISEMPTY)
78.    {
79.        printf(EMPTY);
80.        return;
81.    }
82.    printf("the popped element is %d", stack[top]);
83.    top--;
84. }

85.

86./*Function to display all the elements in the stack*/
87.

88.void display()
89.{
90.    int temp = top;
```

```

92.     if (ISEMPTY)
93.     {
94.         printf(EMPTY);
95.         return;
96.     }
97.     while (temp + 1)
98.     {
99.         printf("%d\n", stack[temp]);
100.            temp--;
101.        }
102.    }
103.
104. /* Function to check whether the stack is full using macro */
105. void stackfull()
106. {
107.     int temp = top, count = 0;
108.
109.     if (ISEMPTY)
110.     {
111.         printf(EMPTY);
112.         return;
113.     }
114.     while (temp + 1)
115.     {
116.         printf("%d\n", stack[temp]);
117.         temp--;
118.         count++;
119.     }
120.     if (count >= MAX)
121.     {
122.         printf("Stack is full");
123.     }
124.     else
125.     {
126.         printf("there are %d more spaces in the stack", (MAX-count));
127.     }
128. }
129.
130. /* Function to return the size of the stack */
131. void stacksize()
132. {
133.     int temp = top, count = 0;
134.     if (ISEMPTY)

```

```

135.         {
136.             printf(EMPTY);
137.             return;
138.         }
139.         while (temp + 1)
140.         {
141.             temp--;
142.             count++;
143.         }
144.         printf("the size of the stack is %d\n", count);
145.     }
146.
147.     /* Function to delete all the elements in the stack */
148.
149.     void destroy()
150.     {
151.         if (ISEMPTY)
152.         {
153.             printf("nothing is there to destroy");
154.         }
155.         while (top != -1)
156.         {
157.             pop();
158.         }
159.     }

```

```

1. /*
2.  * C Program to Implement Stack Operations using Dynamic Memory
3.  * Allocation
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int data;
11.     struct node *link;
12. }*top = NULL;
13.
14. #define MAX 5
15.
16. // function prototypes

```

```
17. void push();
18. void pop();
19. void empty();
20. void stack_full();
21. void stack_count();
22. void destroy();
23. void print_top();
24.
25. void main()
26. {
27.     int choice;
28.
29.     while (1)
30.     {
31.         printf("1. push an element \n");
32.         printf("2. pop an element \n");
33.         printf("3. check if stack is empty \n");
34.         printf("4. check if stack is full \n");
35.         printf("5. count/display elements present in stack \n");
36.         printf("6. empty and destroy stack \n");
37.         printf("7. Print top of the stack \n");
38.         printf("8. exit \n");
39.         printf("Enter your choice \n");
40.         scanf("%d",&choice);
41.         switch (choice)
42.         {
43.             case 1:
44.                 push();
45.                 break;
46.             case 2:
47.                 pop();
48.                 break;
49.             case 3:
50.                 empty();
51.                 break;
52.             case 4:
53.                 stack_full();
54.                 break;
55.             case 5:
56.                 stack_count();
57.                 break;
58.             case 6:
59.                 destroy();
```

```

60.         break;
61.     case 7:
62.         print_top();
63.         break;
64.     case 8:
65.         exit(0);
66.     default:
67.         printf("wrong choice\n");
68.     }
69. }
70. }

71.

72.// to insert elements in stack
73.void push()
74.{
75.    int val,count;
76.    struct node *temp;
77.    temp = (struct node*)malloc(sizeof(struct node));
78.
79.    count = st_count();
80.    if (count <= MAX - 1)
81.    {
82.        printf("\nEnter value which you want to push into the stack :\n");
83.        scanf("%d",&val);
84.        temp->data = val;
85.        temp->link = top;
86.        top = temp;
87.    }
88.    else
89.        printf("WARNING: STACK FULL\n");
90. }

91.

92.// to delete elements from stack
93.void pop()
94.{
95.    struct node *temp;
96.    if (top == NULL)
97.        printf("**Stack is empty**\n");
98.    else
99.    {
100.        temp = top;
101.        printf("Value popped out is %d \n",temp->data);
102.        top = top->link;

```

```
103.         free(temp);
104.     }
105. }
106.
107. // to check if stack is empty
108. void empty()
109. {
110.     if (top == NULL)
111.         printf("STACK IS EMPTY\n");
112.     else
113.         printf("elements are present, stack is not empty \n");
114. }
115.
116. // to check if stack is full
117. void stack_full()
118. {
119.     int count;
120.
121.     count = st_count();
122.     if (count == MAX)
123.     {
124.         printf("stack is full\n");
125.     }
126.     else
127.         printf("stack is not full \n");
128. }
129.
130. // to count the number of elements
131. void stack_count()
132. {
133.     int count = 0;
134.     struct node *temp;
135.
136.     temp = top;
137.     while (temp != NULL)
138.     {
139.         printf(" %d\n",temp->data);
140.         temp = temp->link;
141.         count++;
142.     }
143.     printf("size of stack is %d \n",count);
144. }
```

```

146.     int st_count()
147.     {
148.         int count = 0;
149.         struct node *temp;
150.         temp = top;
151.         while (temp != NULL)
152.         {
153.             temp = temp->link;
154.             count++;
155.         }
156.         return count;
157.     }
158.
159.     // to empty and destroy the stack
160.     void destroy()
161.     {
162.         struct node *temp;
163.         temp = top;
164.         while (temp != NULL)
165.         {
166.             pop();
167.             temp = temp->link;
168.         }
169.         printf("stack destroyed\n");
170.     }
171.
172.     // to print top element of stack
173.     void print_top()
174.     {
175.         if (top == NULL)
176.             printf("\n**Top is not available for an EMPTY stack**\n");
177.         else
178.             printf("\nTop of the stack is %d \n",top->data);
179.     }

```

## (8.) C Programming Examples on Searching and Sorting

### 1. C Examples on Basic Sorting Algorithms

```

1. /*
2.  * C program to sort N numbers in ascending order using Bubble sort
3.  * and print both the given and the sorted array

```

```
4.  /*
5. #include <stdio.h>
6. #define MAXSIZE 10
7.
8. void main()
9. {
10.     int array[MAXSIZE];
11.     int i, j, num, temp;
12.
13.     printf("Enter the value of num \n");
14.     scanf("%d", &num);
15.     printf("Enter the elements one by one \n");
16.     for (i = 0; i < num; i++)
17.     {
18.         scanf("%d", &array[i]);
19.     }
20.     printf("Input array is \n");
21.     for (i = 0; i < num; i++)
22.     {
23.         printf("%d\n", array[i]);
24.     }
25.     /* Bubble sorting begins */
26.     for (i = 0; i < num; i++)
27.     {
28.         for (j = 0; j < (num - i - 1); j++)
29.         {
30.             if (array[j] > array[j + 1])
31.             {
32.                 temp = array[j];
33.                 array[j] = array[j + 1];
34.                 array[j + 1] = temp;
35.             }
36.         }
37.     }
38.     printf("Sorted array is...\n");
39.     for (i = 0; i < num; i++)
40.     {
41.         printf("%d\n", array[i]);
42.     }
43. }
```

```
1. /*
```

```

2.  * C program to accept N numbers and arrange them in an ascending order
3.  */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int i, j, a, n, number[30];
9.
10.    printf("Enter the value of N \n");
11.    scanf("%d", &n);
12.    printf("Enter the numbers \n");
13.    for (i = 0; i < n; ++i)
14.        scanf("%d", &number[i]);
15.    for (i = 0; i < n; ++i)
16.    {
17.        for (j = i + 1; j < n; ++j)
18.        {
19.            if (number[i] > number[j])
20.            {
21.                a = number[i];
22.                number[i] = number[j];
23.                number[j] = a;
24.            }
25.        }
26.    }
27.    printf("The numbers arranged in ascending order are given below \n");
28.    for (i = 0; i < n; ++i)
29.        printf("%d\n", number[i]);
30. }
```

```

1. /*
2.  * C program to read N names, store them in the form of an array
3.  * and sort them in alphabetical order. Output the given names and
4.  * the sorted names in two columns side by side.
5. */
6. #include <stdio.h>
7. #include <string.h>
8.
9. void main()
10. {
11.     char name[10][8], tname[10][8], temp[8];
12.     int i, j, n;
```

```

13.
14.     printf("Enter the value of n \n");
15.     scanf("%d", &n);
16.     printf("Enter %d names n", \n);
17.     for (i = 0; i < n; i++)
18.     {
19.         scanf("%s", name[i]);
20.         strcpy(tname[i], name[i]);
21.     }
22.     for (i = 0; i < n - 1 ; i++)
23.     {
24.         for (j = i + 1; j < n; j++)
25.         {
26.             if (strcmp(name[i], name[j]) > 0)
27.             {
28.                 strcpy(temp, name[i]);
29.                 strcpy(name[i], name[j]);
30.                 strcpy(name[j], temp);
31.             }
32.         }
33.     }
34.     printf("\n-----\n");
35.     printf("Input NamesSorted names\n");
36.     printf("-----\n");
37.     for (i = 0; i < n; i++)
38.     {
39.         printf("%s\t\t%s\n", tname[i], name[i]);
40.     }
41.     printf("-----\n");
42. }
```

```

1. /*
2. * C Program to Implement Selection Sort Recursively
3. */
4. #include <stdio.h>
5.
6. void selection(int [], int, int, int, int);
7.
8. int main()
9. {
10.     int list[30], size, temp, i, j;
```

```

12.     printf("Enter the size of the list: ");
13.     scanf("%d", &size);
14.     printf("Enter the elements in list:\n");
15.     for (i = 0; i < size; i++)
16.     {
17.         scanf("%d", &list[i]);
18.     }
19.     selection(list, 0, 0, size, 1);
20.     printf("The sorted list in ascending order is\n");
21.     for (i = 0; i < size; i++)
22.     {
23.         printf("%d ", list[i]);
24.     }
25.
26.     return 0;
27. }
28.
29. void selection(int list[], int i, int j, int size, int flag)
30. {
31.     int temp;
32.
33.     if (i < size - 1)
34.     {
35.         if (flag)
36.         {
37.             j = i + 1;
38.         }
39.         if (j < size)
40.         {
41.             if (list[i] > list[j])
42.             {
43.                 temp = list[i];
44.                 list[i] = list[j];
45.                 list[j] = temp;
46.             }
47.             selection(list, i, j + 1, size, 0);
48.         }
49.         selection(list, i + 1, 0, size, 1);
50.     }
51. }
```

1. /\*

```

2.  * C program for SELECTION sort which uses following functions
3.  * a) To find maximum of elements
4.  * b) To swap two elements
5.  */
6. #include <stdio.h>
7.
8. int findmax(int b[10], int k);
9. void exchang(int b[10], int k);
10. void main()
11. {
12.     int array[10];
13.     int i, j, n, temp;
14.
15.     printf("Enter the value of n \n");
16.     scanf("%d", &n);
17.     printf("Enter the elements one by one \n");
18.     for (i = 0; i < n; i++)
19.     {
20.         scanf("%d", &array[i]);
21.     }
22.     printf("Input array elements \n");
23.     for (i = 0; i < n ; i++)
24.     {
25.         printf("%d\n", array[i]);
26.     }
27. /* Selection sorting begins */
28. exchang(array, n);
29. printf("Sorted array is...\n");
30. for (i = 0; i < n; i++)
31. {
32.     printf("%d\n", array[i]);
33. }
34. }
35./* function to find the maximum value */
36. int findmax(int b[10], int k)
37. {
38.     int max = 0, j;
39.     for (j = 1; j <= k; j++)
40.     {
41.         if (b[j] > b[max])
42.         {
43.             max = j;
44.         }

```

```

45.     }
46.     return(max);
47. }
48. void exchang(int b[10], int k)
49. {
50.     int temp, big, j;
51.     for (j = k - 1; j >= 1; j--)
52.     {
53.         big = findmax(b, j);
54.         temp = b[big];
55.         b[big] = b[j];
56.         b[j] = temp;
57.     }
58.     return;
59. }
```

```

1. /*
2.  * C Program to Input Few Numbers & Perform Merge Sort on them using Recursion
3. */
4.
5. #include <stdio.h>
6.
7. void mergeSort(int [], int, int, int);
8. void partition(int [],int, int);
9.
10.int main()
11.{
12.    int list[50];
13.    int i, size;
14.
15.    printf("Enter total number of elements:");
16.    scanf("%d", &size);
17.    printf("Enter the elements:\n");
18.    for(i = 0; i < size; i++)
19.    {
20.        scanf("%d", &list[i]);
21.    }
22.    partition(list, 0, size - 1);
23.    printf("After merge sort:\n");
24.    for(i = 0;i < size; i++)
25.    {
26.        printf("%d    ",list[i]);
```

```
27.     }
28.
29.     return 0;
30. }
31.
32. void partition(int list[],int low,int high)
33. {
34.     int mid;
35.
36.     if(low < high)
37.     {
38.         mid = (low + high) / 2;
39.         partition(list, low, mid);
40.         partition(list, mid + 1, high);
41.         mergeSort(list, low, mid, high);
42.     }
43. }
44.
45. void mergeSort(int list[],int low,int mid,int high)
46. {
47.     int i, mi, k, lo, temp[50];
48.
49.     lo = low;
50.     i = low;
51.     mi = mid + 1;
52.     while ((lo <= mid) && (mi <= high))
53.     {
54.         if (list[lo] <= list[mi])
55.         {
56.             temp[i] = list[lo];
57.             lo++;
58.         }
59.         else
60.         {
61.             temp[i] = list[mi];
62.             mi++;
63.         }
64.         i++;
65.     }
66.     if (lo > mid)
67.     {
68.         for (k = mi; k <= high; k++)
69.         {
```

```

70.         temp[i] = list[k];
71.         i++;
72.     }
73. }
74. else
75. {
76.     for (k = lo; k <= mid; k++)
77.     {
78.         temp[i] = list[k];
79.         i++;
80.     }
81. }
82.
83. for (k = low; k <= high; k++)
84. {
85.     list[k] = temp[k];
86. }
87. }
```

```

1. /*
2. * C Program to Perform Quick Sort on a set of Entries from a File
3. * using Recursion
4. */
5. #include <stdio.h>
6.
7. void quicksort (int [], int, int);
8.
9. int main()
10. {
11.     int list[50];
12.     int size, i;
13.
14.     printf("Enter the number of elements: ");
15.     scanf("%d", &size);
16.     printf("Enter the elements to be sorted:\n");
17.     for (i = 0; i < size; i++)
18.     {
19.         scanf("%d", &list[i]);
20.     }
21.     quicksort(list, 0, size - 1);
22.     printf("After applying quick sort\n");
23.     for (i = 0; i < size; i++)
```

```
24.     {
25.         printf("%d ", list[i]);
26.     }
27.     printf("\n");
28.
29.     return 0;
30. }
31. void quicksort(int list[], int low, int high)
32. {
33.     int pivot, i, j, temp;
34.     if (low < high)
35.     {
36.         pivot = low;
37.         i = low;
38.         j = high;
39.         while (i < j)
40.         {
41.             while (list[i] <= list[pivot] && i <= high)
42.             {
43.                 i++;
44.             }
45.             while (list[j] > list[pivot] && j >= low)
46.             {
47.                 j--;
48.             }
49.             if (i < j)
50.             {
51.                 temp = list[i];
52.                 list[i] = list[j];
53.                 list[j] = temp;
54.             }
55.         }
56.         temp = list[j];
57.         list[j] = list[pivot];
58.         list[pivot] = temp;
59.         quicksort(list, low, j - 1);
60.         quicksort(list, j + 1, high);
61.     }
62. }
```

## 2. C Examples on Searching Algorithms

```

1. /*
2.  * C program accept an array of N elements and a key to search.
3.  * If the search is successful, it displays "SUCCESSFUL SEARCH".
4.  * Otherwise, a message "UNSUCCESSFUL SEARCH" is displayed.
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     int array[20];
11.     int i, low, mid, high, key, size;
12.
13.     printf("Enter the size of an array\n");
14.     scanf("%d", &size);
15.     printf("Enter the array elements\n");
16.     for (i = 0; i < size; i++)
17.     {
18.         scanf("%d", &array[i]);
19.     }
20.     printf("Enter the key\n");
21.     scanf("%d", &key);
22.     /* search begins */
23.     low = 0;
24.     high = (size - 1);
25.     while (low <= high)
26.     {
27.         mid = (low + high) / 2;
28.         if (key == array[mid])
29.         {
30.             printf("SUCCESSFUL SEARCH\n");
31.             return;
32.         }
33.         if (key < array[mid])
34.             high = mid - 1;
35.         else
36.             low = mid + 1;
37.     }
38.     printf("UNSUCCESSFUL SEARCH\n");
39. }
```

```

1. /*
2.  * C program to accept N numbers sorted in ascending order
```

```
3.  * and to search for a given number using binary search.
4.  * Report success or failure.
5.  */
6. #include <stdio.h>
7.
8. void main()
9. {
10.    int array[10];
11.    int i, j, num, temp, keynum;
12.    int low, mid, high;
13.
14.    printf("Enter the value of num \n");
15.    scanf("%d", &num);
16.    printf("Enter the elements one by one \n");
17.    for (i = 0; i < num; i++)
18.    {
19.        scanf("%d", &array[i]);
20.    }
21.    printf("Input array elements \n");
22.    for (i = 0; i < num; i++)
23.    {
24.        printf("%d\n", array[i]);
25.    }
26.    /* Bubble sorting begins */
27.    for (i = 0; i < num; i++)
28.    {
29.        for (j = 0; j < (num - i - 1); j++)
30.        {
31.            if (array[j] > array[j + 1])
32.            {
33.                temp = array[j];
34.                array[j] = array[j + 1];
35.                array[j + 1] = temp;
36.            }
37.        }
38.    }
39.    printf("Sorted array is...\n");
40.    for (i = 0; i < num; i++)
41.    {
42.        printf("%d\n", array[i]);
43.    }
44.    printf("Enter the element to be searched \n");
45.    scanf("%d", &keynum);
```

```

46.     /* Binary searching begins */
47.     low = 1;
48.     high = num;
49.     do
50.     {
51.         mid = (low + high) / 2;
52.         if (keynum < array[mid])
53.             high = mid - 1;
54.         else if (keynum > array[mid])
55.             low = mid + 1;
56.     } while (keynum != array[mid] && low <= high);
57.     if (keynum == array[mid])
58.     {
59.         printf("SEARCH SUCCESSFUL \n");
60.     }
61.     else
62.     {
63.         printf("SEARCH FAILED \n");
64.     }
65. }
```

```

1. /*
2.  * C Program to search for an element in a List using
3.  */
4. #include <stdio.h>
5.
6. int search(int [], int, int);
7. int main()
8. {
9.     int size, index, key;
10.    int list[20];
11.    int count = 0;
12.    int i;
13.
14.    printf("Enter the size of the list: ");
15.    scanf("%d", &size);
16.    index = size;
17.    printf("Printing the list:\n");
18.    for (i = 0; i < size; i++)
19.    {
20.        list[i] = rand() % size;
21.        printf("%d\t", list[i]);
```

```

22. }
23. printf("\nEnter the key to search: ");
24. scanf("%d", &key);
25. while (index > 0)
26. {
27.     index = search(list, index - 1, key);
28.     /* In an array first position is indexed by 0 */
29.     printf("Key found at position: %d\n", index + 1);
30.     count++;
31. }
32. if (!count)
33.     printf("Key not found.\n");
34. return 0;
35. }
36. int search(int array[], int size, int key)
37. {
38.     int location;
39.     if (array[size] == key)
40.     {
41.         return size;
42.     }
43.     else if (size == -1)
44.     {
45.         return -1;
46.     }
47.     else
48.     {
49.         return (location = search(array, size - 1, key));
50.     }
51. }
```

```

1. /*
2. * C Program to Perform Binary Search using Recursion
3. */
4. #include <stdio.h>
5.
6. void binary_search(int [], int, int, int);
7. void bubble_sort(int [], int);
8.
9. int main()
10. {
11.     int key, size, i;
```

```
12.     int list[25];
13.
14.     printf("Enter size of a list: ");
15.     scanf("%d", &size);
16.     printf("Generating random numbers\n");
17.     for(i = 0; i < size; i++)
18.     {
19.         list[i] = rand() % 100;
20.         printf("%d ", list[i]);
21.     }
22.     bubble_sort(list, size);
23.     printf("\n\n");
24.     printf("Enter key to search\n");
25.     scanf("%d", &key);
26.     binary_search(list, 0, size, key);
27.
28. }
29.
30. void bubble_sort(int list[], int size)
31. {
32.     int temp, i, j;
33.     for (i = 0; i < size; i++)
34.     {
35.         for (j = i; j < size; j++)
36.         {
37.             if (list[i] > list[j])
38.             {
39.                 temp = list[i];
40.                 list[i] = list[j];
41.                 list[j] = temp;
42.             }
43.         }
44.     }
45. }
46.
47. void binary_search(int list[], int lo, int hi, int key)
48. {
49.     int mid;
50.
51.     if (lo > hi)
52.     {
53.         printf("Key not found\n");
54.         return;
```

```

55.     }
56.     mid = (lo + hi) / 2;
57.     if (list[mid] == key)
58.     {
59.         printf("Key found\n");
60.     }
61.     else if (list[mid] > key)
62.     {
63.         binary_search(list, lo, mid - 1, key);
64.     }
65.     else if (list[mid] < key)
66.     {
67.         binary_search(list, mid + 1, hi, key);
68.     }
69. }

```

```

1. /*
2. * C program to input N numbers and store them in an array.
3. * Do a linear search for a given key and report success
4. * or failure.
5. */
6. #include <stdio.h>
7.
8. void main()
9. {
10.     int array[10];
11.     int i, num, keynum, found = 0;
12.
13.     printf("Enter the value of num \n");
14.     scanf("%d", &num);
15.     printf("Enter the elements one by one \n");
16.     for (i = 0; i < num; i++)
17.     {
18.         scanf("%d", &array[i]);
19.     }
20.     printf("Input array is \n");
21.     for (i = 0; i < num; i++)
22.     {
23.         printf("%dn", array[i]);
24.     }
25.     printf("Enter the element to be searched \n");
26.     scanf("%d", &keynum);

```

```

27. /* Linear search begins */
28. for (i = 0; i < num ; i++)
29. {
30.     if (keynum == array[i] )
31.     {
32.         found = 1;
33.         break;
34.     }
35. }
36. if (found == 1)
37.     printf("Element is present in the array\n");
38. else
39.     printf("Element is not present in the array\n");
40. }
```

#### 41. 3. C Examples on Special Sorting Algorithms

```

42. /*
43. * C Program to Implement Insertion Sort
44. */
45. #include <stdio.h>
46. #define MAX 7
47.
48. void insertion_sort(int *);
49.
50. void main()
51. {
52.     int a[MAX], i;
53.
54.     printf("enter elements to be sorted:");
55.     for (i = 0;i < MAX;i++)
56.     {
57.         scanf("%d", &a[i]);
58.     }
59.     insertion_sort(a);
60.     printf("sorted elements:\n");
61.     for (i = 0;i < MAX; i++)
62.     {
63.         printf(" %d", a[i]);
64.     }
65. }
66.
67. /* sorts the input */
```

```

68. void insertion_sort(int * x)
69. {
70.     int temp, i, j;
71.
72.     for (i = 1; i < MAX; i++)
73.     {
74.         temp = x[i];
75.         j = i - 1;
76.         while (temp < x[j] && j >= 0)
77.         {
78.             x[j + 1] = x[j];
79.             j = j - 1;
80.         }
81.         x[j + 1] = temp;
82.     }
83. }
```

```

1. /*
2.  * C Program to Implement Postman Sort Algorithm
3.  */
4. #include <stdio.h>
5.
6. void arrange(int, int);
7. int array[100], array1[100];
8. int i, j, temp, max, count, maxdigits = 0, c = 0;
9.
10. void main()
11. {
12.     int t1, t2, k, t, n = 1;
13.
14.     printf("Enter size of array :");
15.     scanf("%d", &count);
16.     printf("Enter elements into array :");
17.     for (i = 0; i < count; i++)
18.     {
19.         scanf("%d", &array[i]);
20.         array1[i] = array[i];
21.     }
22.     for (i = 0; i < count; i++)
23.     {
24.         t = array[i];           /*first element in t */
25.         while(t > 0)
```

```

26.     {
27.         c++;
28.         t = t / 10;           /* Find MSB */
29.     }
30.     if (maxdigits < c)
31.         maxdigits = c;    /* number of digits of a each number */
32.     c = 0;
33. }
34. while(--maxdigits)
35.     n = n * 10;
36.
37. for (i = 0; i < count; i++)
38. {
39.     max = array[i] / n;           /* MSB - Dividnng by perticular base */
40.     t = i;
41.     for (j = i + 1; j < count;j++)
42.     {
43.         if (max > (array[j] / n))
44.         {
45.             max = array[j] / n;   /* greatest MSB */
46.             t = j;
47.         }
48.     }
49.     temp = array1[t];
50.     array1[t] = array1[i];
51.     array1[i] = temp;
52.     temp = array[t];
53.     array[t] = array[i];
54.     array[i] = temp;
55. }
56. while (n >= 1)
57. {
58.     for (i = 0; i < count;)
59.     {
60.         t1 = array[i] / n;
61.         for (j = i + 1; t1 == (array[j] / n); j++);
62.             arrange(i, j);
63.         i = j;
64.     }
65.     n = n / 10;
66. }
67. printf("\nSorted Array (Postman sort) :");
68. for (i = 0; i < count; i++)

```

```

69.         printf("%d ", array1[i]);
70.     printf("\n");
71. }
72.
73./* Function to arrange the sequence having same base */
74.void arrange(int k,int n)
75.{
76.    for (i = k; i < n - 1; i++)
77.    {
78.        for (j = i + 1; j < n; j++)
79.        {
80.            if (array1[i] > array1[j])
81.            {
82.                temp = array1[i];
83.                array1[i] = array1[j];
84.                array1[j] = temp;
85.                temp = (array[i] % 10);
86.                array[i] = (array[j] % 10);
87.                array[j] = temp;
88.            }
89.        }
90.    }
91.}

```

```

1. /*
2. * C Program to Sort an Integer Array using LSDRadix Sort Algorithm
3. */
4. #include <stdio.h>
5.
6. int min = 0, count = 0, array[100] = {0}, array1[100] = {0};
7.
8. void main()
9. {
10.     int k, i, j, temp, t, n;
11.
12.     printf("Enter size of array :");
13.     scanf("%d", &count);
14.     printf("Enter elements into array :");
15.     for (i = 0; i < count; i++)
16.     {
17.         scanf("%d", &array[i]);
18.         array1[i] = array[i];

```

```

19.    }
20.    for (k = 0; k < 3; k++)
21.    {
22.        for (i = 0; i < count; i++)
23.        {
24.            min = array[i] % 10;           /* To find minimum Lsd */
25.            t = i;
26.            for (j = i + 1; j < count; j++)
27.            {
28.                if (min > (array[j] % 10))
29.                {
30.                    min = array[j] % 10;
31.                    t = j;
32.                }
33.            }
34.            temp = array1[t];
35.            array1[t] = array1[i];
36.            array1[i] = temp;
37.            temp = array[t];
38.            array[t] = array[i];
39.            array[i] = temp;
40.
41.        }
42.        for (j = 0; j < count; j++)      /*to find MSB */
43.            array[j] = array[j] / 10;
44.    }
45.    printf("Sorted Array (1Sdradix sort) : ");
46.    for (i = 0; i < count; i++)
47.        printf("%d ", array1[i]);
48. }
```

```

1. /*
2. * C Program to sort an array based on heap sort algorithm(MAX heap)
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int heap[10], no, i, j, c, root, temp;
9.
10.    printf("\n Enter no of elements :");
11.    scanf("%d", &no);
```

```

12.     printf("\n Enter the nos : ");
13.     for (i = 0; i < no; i++)
14.         scanf("%d", &heap[i]);
15.     for (i = 1; i < no; i++)
16.     {
17.         c = i;
18.         do
19.         {
20.             root = (c - 1) / 2;
21.             if (heap[root] < heap[c]) /* to create MAX heap array */
22.             {
23.                 temp = heap[root];
24.                 heap[root] = heap[c];
25.                 heap[c] = temp;
26.             }
27.             c = root;
28.         } while (c != 0);
29.     }
30.
31.     printf("Heap array : ");
32.     for (i = 0; i < no; i++)
33.         printf("%d\t ", heap[i]);
34.     for (j = no - 1; j >= 0; j--)
35.     {
36.         temp = heap[0];
37.         heap[0] = heap[j] /* swap max element with rightmost leaf element */
38.         heap[j] = temp;
39.         root = 0;
40.         do
41.         {
42.             c = 2 * root + 1; /* Left node of root element */
43.             if ((heap[c] < heap[c + 1]) && c < j-1)
44.                 c++;
45.             if (heap[root]<heap[c] && c<j) /* again rearrange to max heap array
*/
46.             {
47.                 temp = heap[root];
48.                 heap[root] = heap[c];
49.                 heap[c] = temp;
50.             }
51.             root = c;
52.         } while (c < j);
53.     }

```

```

54.     printf("\n The sorted array is : ");
55.     for (i = 0; i < no; i++)
56.         printf("\t %d", heap[i]);
57.     printf("\n Complexity : \n Best case = Avg case = Worst case = O(n logn) \n");
58. }
```

```

1. /*
2.  * C Program to Sort the Array Elements using Gnome Sort
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int i, temp, ar[10], n;
9.
10.    printf("\nEnter the elemnts number u would like to enter:");
11.    scanf("%d", &n);
12.    printf("\nEnter the elements to be sorted through gnome sort:\n");
13.    for (i = 0; i < n; i++)
14.        scanf("%d", &ar[i]);
15.    i = 0;
16.    while (i < n)
17.    {
18.        if (i == 0 || ar[i - 1] <= ar[i])
19.            i++;
20.        else
21.        {
22.            temp = ar[i-1];
23.            ar[i - 1] = ar[i];
24.            ar[i] = temp;
25.            i = i - 1;
26.        }
27.    }
28.    for (i = 0;i < n;i++)
29.        printf("%d\t", ar[i]);
30. }
```

#### 4. C Examples on another category of Special Sorting Algorithms

```

1. /*
2.  * C Program to Implement qsort using function pointers
3. */
4. #include <stdio.h>
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <string.h>
8.
9. struct s
10. {
11.     char empname[5];
12.     int empid;
13. };
14.
15./* To sort array elemets */
16.int int_call(const void *a1,const void *b1)
17.{
18.    const int *a = (const int *)a1;
19.    const int *b = (const int *)b1;
20.
21.    if (*a > *b)
22.        return 1;
23.    else
24.    {
25.        if (*a == *b)
26.            return 0;
27.        else
28.            return -1;
29.    }
30.}
31.
32./* To sort structure elemets */
33.int string_call(const void *a1, const void *b1)
34.{
35.    const char *a = (const char *)a1;
36.    const char *b = (const char *)b1;
37.    return(strcmp(a, b));
38.}
39.
40.void main()
41.{
42.    int array1[5]={20, 30, 50, 60, 10};
43.    struct s emprec[5];

```

```

44.     int i, j;
45.
46.     strcpy(emprec[0].empname, "bbb");
47.     emprec[0].empid = 100;
48.     strcpy(emprec[1].empname, "ccc");
49.     emprec[1].empid = 200;
50.     strcpy(emprec[2].empname, "eee");
51.     emprec[2].empid = 300;
52.     strcpy(emprec[3].empname, "aaa");
53.     emprec[3].empid = 400;
54.     strcpy(emprec[4].empname, "ddd");
55.     emprec[4].empid = 500;
56.     qsort(array1, 5, sizeof(int), int_call);
57.     qsort(emprec, 5, sizeof(struct s), string_call);
58.     for (i = 0; i < 5; i++)
59.         printf("%d\t", array1[i]);
60.     printf("\nSorting of Structure elements ");
61.     for (i = 0; i < 5; i++)
62.         printf("\n%s\t%d", emprec[i].empname, emprec[i].empid);
63.     printf("\n");
64. }
```

```

1. /*
2.  * C Program to Implement Pigeonhole Sort
3. */
4. #include <stdio.h>
5.
6. #define MAX 7
7.
8. void pigeonhole_sort(int, int, int *);
9. void main()
10. {
11.     int a[MAX], i, min, max;
12.     printf("enter the values into the matrix :");
13.     for (i = 0; i < MAX; i++)
14.     {
15.         scanf("%d", &a[i]);
16.     }
17.     min = a[0];
18.     max = a[0];
19.     for (i = 1; i < MAX; i++)
20.     {
```

```
21.     if (a[i] < min)
22.     {
23.         min = a[i];
24.     }
25.     if (a[i] > max)
26.     {
27.         max = a[i];
28.     }
29. }
30. pigeonhole_sort(min, max, a);
31. printf("Sorted order is :\n");
32. for (i = 0; i < MAX; i++)
33. {
34.     printf("%d", a[i]);
35. }
36. }
37.
38. /* sorts the array using pigeonhole algorithm */
39. void pigeonhole_sort(int mi, int ma, int * a)
40. {
41.
42.     int size, count = 0, i;
43.     int *current;
44.     current = a;
45.     size = ma - mi + 1;
46.     int holes[size];
47.     for (i = 0; i < size; i++)
48.     {
49.         holes[i] = 0;
50.     }
51.     for (i = 0; i < size; i++, current++)
52.     {
53.         holes[*current-mi] += 1;
54.     }
55.     for (count = 0, current = &a[0]; count < size; count++)
56.     {
57.         while (holes[count--] > 0)
58.         {
59.             *current++ = count + mi;
60.         }
61.     }
62. }
```

```
1. /*
2.  * C Program to Implement Cyclesort
3. */
4. #include <stdio.h>
5.
6. #define MAX 8
7.
8. void cycle_sort(int *);
9.
10. void main()
11. {
12.     int a[MAX], i;
13.
14.     printf("enter the elements into array :");
15.     for (i = 0; i < MAX; i++)
16.     {
17.         scanf("%d", &a[i]);
18.     }
19.     cycle_sort(a);
20.     printf("sorted elements are :\n");
21.     for (i = 0; i < MAX; i++)
22.     {
23.         printf("%d", a[i]);
24.     }
25. }
26.
27. /* sorts elements using cycle sort algorithm */
28. void cycle_sort(int * a)
29. {
30.     int temp, item, pos, i, j, k;
31.
32.     for (i = 0; i < MAX; i++)
33.     {
34.         item = a[i];
35.         pos = i;
36.         do
37.         {
38.             k = 0;
39.             for (j = 0; j < MAX; j++)
40.             {
41.                 if (pos != j && a[j] < item)
```

```

42.         {
43.             k++;
44.         }
45.     }
46.     if (pos != k)
47.     {
48.         while (pos != k && item == a[k])
49.         {
50.             k++;
51.         }
52.         temp = a[k];
53.         a[k] = item;
54.         item = temp;
55.         pos = k;
56.     }
57. }while (pos != i);
58.
59. }
```

```

1. /*
2. * C Program to Implement Oddeven Sort
3. */
4. #include <stdio.h>
5. #define MAX 7
6.
7. void swap(int *,int *);
8. void oddeven_sort(int *);
9.
10. void main()
11. {
12.     int a[MAX], i;
13.
14.     printf("enter the elements in to the matrix :");
15.     for (i = 0;i < MAX;i++)
16.     {
17.         scanf("%d", &a[i]);
18.     }
19.     printf("sorted elements are :\n");
20.     oddeven_sort(a);
21.     for (i = 0;i < MAX;i++)
22.     {
23.         printf(" %d", a[i]);
```

```

24.     }
25. }
26.
27./* swaps the elements */
28.void swap(int * x, int * y)
29.{
30.    int temp;
31.
32.    temp = *x;
33.    *x = *y;
34.    *y = temp;
35.}
36.
37./* sorts the array using oddeven algorithm */
38.void oddeven_sort(int * x)
39.{
40.    int sort = 0, i;
41.
42.    while (!sort)
43.    {
44.        sort = 1;
45.        for (i = 1; i < MAX; i += 2)
46.        {
47.            if (x[i] > x[i+1])
48.            {
49.                swap(&x[i], &x[i+1]);
50.                sort = 0;
51.            }
52.        }
53.        for (i = 0; i < MAX - 1; i += 2)
54.        {
55.            if (x[i] > x[i + 1])
56.            {
57.                swap(&x[i], &x[i + 1]);
58.                sort = 0;
59.            }
60.        }
61.    }
62.}

```

```

1. /*
2. * C Program to Implement CockTail Sort

```

```
3.  */
4. #include <stdio.h>
5. #define MAX 8
6.
7. int main()
8. {
9.     int data[MAX];
10.    int i, j, n, c;
11.
12.    printf("\nEnter the data");
13.    for (i = 0; i < MAX; i++)
14.    {
15.        scanf("%d", &data[i]);
16.    }
17.    n = MAX;
18.    do
19.    {
20.        /*
21.             * Rightward pass will shift the largest element to its correct place at
22.             * the end
23.        */
24.        for (i = 0; i < n - 1; i++)
25.        {
26.            if (data[i] > data[i + 1])
27.            {
28.                data[i] = data[i] + data[i + 1];
29.                data[i + 1] = data[i] - data[i + 1];
30.                data[i] = data[i] - data[i + 1];
31.            }
32.
33.        }
34.        n = n - 1;
35.        /*
36.             * Leftward pass will shift the smallest element to its correct place at
37.             * the beginning
38.        */
39.        for (i= MAX - 1, c = 0; i >= c; i--)
40.        {
41.            if(data[i] < data[i - 1])
42.            {
43.                data[i] = data[i] + data[i - 1];
44.                data[i - 1] = data[i] - data[i - 1];
```

```

44.             data[i] = data[i] - data[i - 1];
45.         }
46.     }
47.     c = c + 1;
48.
49. } while (n != 0 && c != 0);
50. printf("The sorted elements are:");
51. for (i = 0; i < MAX; i++)
52. {
53.     printf("%d\t", data[i]);
54. }
55. }
```

```

1. /*
2. * C Program to Implement Bitonic sort
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #define MAX 8
7. #define SWAP(x,y) t = x; x = y; y = t;
8.
9. void compare();
10. void bitonicmerge(int, int, int);
11. void recbitonic(int, int, int);
12. void sort();
13.
14. int data[MAX];
15. int up = 1;
16. int down = 0;
17.
18. int main()
19. {
20.     int i;
21.
22.     printf("\nEnter the data");
23.     for (i = 0;i < MAX ;i++)
24.     {
25.         scanf("%d", &data[i]);
26.     }
27.     sort();
28.     for (i = 0;i < MAX;i++)
29.     {
```

```

30.         printf("%d ", data[i]);
31.     }
32. }
33./*
34. * compare and swap based on dir
35. */
36.void compare(int i, int j, int dir)
37.{
38.    int t;
39.
40.    if (dir == (data[i] > data[j]))
41.    {
42.        SWAP(data[i], data[j]);
43.    }
44.}
45./*
46. * Sorts a bitonic sequence in ascending order if dir=1
47. * otherwise in descending order
48. */
49.void bitonicmerge(int low, int c, int dir)
50.{
51.    int k, i;
52.
53.    if (c > 1)
54.    {
55.        k = c / 2;
56.        for (i = low;i < low+k ;i++)
57.            compare(i, i+k, dir);
58.        bitonicmerge(low, k, dir);
59.        bitonicmerge(low+k, k, dir);
60.    }
61.}
62./*
63. * Generates bitonic sequence by sorting recursively
64. * two halves of the array in opposite sorting orders
65. * bitonicmerge will merge the resultant data
66. */
67.void recbitonic(int low, int c, int dir)
68.{
69.    int k;
70.
71.    if (c > 1)
72.    {

```

```

73.         k = c / 2;
74.         recbitonic(low, k, up);
75.         recbitonic(low + k, k, down);
76.         bitonicmerge(low, c, dir);
77.     }
78. }
79.
80./*
81. * Sorts the entire array
82. */
83. void sort()
84. {
85.     recbitonic(0, MAX, up);
86. }
```

```

1. /*
2.  * C Program to Perform Comb Sort on Array of Integers
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. /*Function to find the new gap between the elements*/
8. int newgap(int gap)
9. {
10.     gap = (gap * 10) / 13;
11.     if (gap == 9 || gap == 10)
12.         gap = 11;
13.     if (gap < 1)
14.         gap = 1;
15.     return gap;
16. }
17.
18./*Function to implement the combsort*/
19.void combsort(int a[], int aSize)
20.{
21.    int gap = aSize;
22.    int temp, i;
23.    for (;;)
24.    {
25.        gap = newgap(gap);
26.        int swapped = 0;
27.        for (i = 0; i < aSize - gap; i++)
```

```

28.     {
29.         int j = i + gap;
30.         if (a[i] > a[j])
31.         {
32.             temp = a[i];
33.             a[i] = a[j];
34.             a[j] = temp;
35.             swapped = 1;
36.         }
37.     }
38.     if (gap == 1 && !swapped)
39.         break;
40. }
41. }
42. int main ()
43. {
44.     int n, i;
45.     int *a;
46.     printf("Please insert the number of elements to be sorted: ");
47.     scanf("%d", &n);           // The total number of elements
48.     a = (int *)calloc(n, sizeof(int));
49.     for (i = 0;i< n;i++)
50.     {
51.         printf("Input element %d :", i);
52.         scanf("%d", &a[i]); // Adding the elements to the array
53.     }
54.     printf("unsorted list");      // Displaying the unsorted array
55.     for(i = 0;i < n;i++)
56.     {
57.         printf("%d", a[i]);
58.     }
59.     combsort(a, n);
60.     printf("Sorted list:\n");      // Display the sorted array
61.     for(i = 0;i < n;i++)
62.     {
63.         printf("%d ", (a[i]));
64.     }
65.     return 0;
66. }
```

```

1. /*
2. * C Program to Implement Stooge Sort
```

```

3.  /*
4. #include <stdio.h>
5.
6. // Function Prototype
7. void stoogesort(int [], int, int);
8.
9. void main()
10. {
11.     int b[7], i;
12.
13.     printf("Enter the values you want to sort using STOOGESORT!!!:\n");
14.     for (i = 0; i < 7; i++)
15.         scanf(" %d", &b[i]);
16.     stoogesort(b, 0, 6);
17.     printf("sorted by STOOGESORT\n");
18.     for (i = 0; i < 7; i++)
19.     {
20.         printf("%d ", b[i]);
21.     }
22.     printf("\n");
23. }
24.
25. // Function to implement STOOGESORT
26. void stoogesort(int a[], int i, int j)
27. {
28.     int temp, k;
29.
30.     if (a[i] > a[j])
31.     {
32.         temp = a[i];
33.         a[i] = a[j];
34.         a[j] = temp;
35.     }
36.     if ((i + 1) >= j)
37.         return;
38.     k = (int)((j - i + 1) / 3);
39.     stoogesort(a, i, j - k);
40.     stoogesort(a, i + k, j);
41.     stoogesort(a, i, j - k);
42. }
```

## 5. C Examples on Pancake Sort, Bogo Sort and Shell Sort

```
1. /*
2.  * C Program to Implement Pancake Sort on Array of Integers
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void do_flip(int *, int, int);
8.
9. /*Function to implement the pancake sort*/
10.int pancake_sort(int *list, unsigned int length)
11.{  
12.    if (length < 2)  
13.        return 0;  
14.    int i, a, max_num_pos, moves;  
15.  
16.    moves = 0;  
17.    for (i = length;i > 1;i--)  
18.    {  
19.        max_num_pos = 0;  
20.        for (a = 0;a < i;a++)  
21.        {  
22.            if (list[a] > list[max_num_pos])  
23.                max_num_pos = a;  
24.        }  
25.        if (max_num_pos == i - 1)  
26.            continue;  
27.        if (max_num_pos)  
28.        {  
29.            moves++;  
30.            do_flip(list, length, max_num_pos + 1);  
31.        }  
32.        do_flip(list, length, i);  
33.    }  
34.    return moves;  
35.}  
36.  
37.//*Function to do flips in the elements*/
38 void do_flip(int *list, int length, int num)
39.{  
40.    int swap;  
41.    int i = 0;
```

```

42.     for (i;i < --num;i++)
43.     {
44.         swap = list[i];
45.         list[i] = list[num];
46.         list[num] = swap;
47.     }
48. }
49.
50./*Function to print the array*/
51.void print_array(int list[], int length)
52.{
53.    int i;
54.    for (i = 0;i < length;i++)
55.    {
56.        printf("%d ", list[i]);
57.    }
58.}
59.
60.int main(int argc, char **argv)
61.{
62.    int list[9];
63.    int i;
64.    printf("enter the 9 elements of array:\n");
65.    for (i = 0;i < 9;i++)
66.        scanf("%d", &list[i]);
67.    printf("\nOriginal: ");
68.    print_array(list, 9);
69.    int moves = pancake_sort(list, 9);
70.    printf("\nSorted: ");
71.    print_array(list, 9);
72.    printf(" - with a total of %d moves\n", moves);
73.}
```

```

1. /*
2. * C Program to Implement BogoSort in an Integer Array
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. #define size 7
8. /* Function Prototypes */
9.
```

```

10. int is_sorted(int *, int);
11. void shuffle(int *, int);
12. void bogosort(int *, int);
13.
14. int main()
15. {
16.     int numbers[size];
17.     int i;
18.
19.     printf("Enter the elements of array:");
20.     for (i = 0; i < size;i++)
21.     {
22.         scanf("%d", &numbers[i]);
23.     }
24.     bogosort(numbers, size);
25.     printf("The array after sorting is:");
26.     for (i = 0;i < size;i++)
27.     {
28.         printf("%d\n", numbers[i]);
29.     }
30.     printf("\n");
31. }
32.
33./* Code to check if the array is sorted or not */
34. int is_sorted(int *a, int n)
35. {
36.     while (--n >= 1)
37.     {
38.         if (a[n] < a[n - 1])
39.         {
40.             return 0;
41.         }
42.     }
43.     return 1;
44. }
45.
46./* Code to shuffle the array elements */
47. void shuffle(int *a, int n)
48. {
49.     int i, t, temp;
50.     for (i = 0;i < n;i++)
51.     {
52.         t = a[i];

```

```
53.     temp = rand() % n; /* Shuffles the given array using Random function */
54.     a[i] = a[temp];
55.     a[temp] = t;
56. }
57. }
58.
59./* Code to check if the array is sorted or not and if not sorted calls the shuffle
   function to shuffle the array elements */
60.void bogosort(int *a, int n)
61.{
62.    while (!is_sorted(a, n))
63.    {
64.        shuffle(a, n);
65.    }
66.}
67./*
68. * C Program to Perform Shell Sort without using Recursion
69. */
70.#include <stdio.h>
71.#define size 7
72.
73./* Function Prototype */
74.int shell_sort(int []);
75.
76 void main()
77.{
78.    int arr[size], i;
79.    printf("Enter the elements to be sorted:");
80.    for (i = 0;i < size;i++)
81.    {
82.        scanf("%d", &arr[i]);
83.    }
84.    shell_sort(arr);
85.    printf("The array after sorting is:");
86.    for (i = 0;i < size;i++)
87.    {
88.        printf("\n%d", arr[i]);
89.    }
90.}
91.
92./* Code to sort array using shell sort */
93.int shell_sort(int array[])
94.{
```

```

95.     int i = 0, j = 0, k = 0, mid = 0;
96.     for (k = size / 2;k > 0;k /= 2)
97.     {
98.         for (j = k;j < size;j++)
99.         {
100.             for (i = j - k;i >= 0;i -= k)
101.             {
102.                 if (array[i + k] >= array[i])
103.                 {
104.                     break;
105.                 }
106.                 else
107.                 {
108.                     mid = array[i];
109.                     array[i] = array[i + k];
110.                     array[i + k] = mid;
111.                 }
112.             }
113.         }
114.     }
115.     return 0;
116. }
```

## (9.) C Programming Examples on Trees

### 1. C Examples on Tree Traversals

```

1. /*
2. * C Program for Depth First Binary Tree Search using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12.};
13.
14. void generate(struct node **, int);
```

```

15. void DFS(struct node * );
16. void delete(struct node ** );
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
   Exit\nChoice: ");
26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 DFS(head);
36.                 break;
37.             case 3:
38.                 delete(&head);
39.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
40.                 break;
41.             default:
42.                 printf("Not a valid input, try again\n");
43.         }
44.     } while (choice != 3);
45.     return 0;
46. }
47.
48. void generate(struct node **head, int num)
49. {
50.     struct node *temp = *head, *prev = *head;
51.
52.     if (*head == NULL)
53.     {
54.         *head = (struct node *)malloc(sizeof(struct node));
55.         (*head)->a = num;
56.         (*head)->left = (*head)->right = NULL;

```

```
57.     }
58. else
59. {
60.     while (temp != NULL)
61.     {
62.         if (num > temp->a)
63.         {
64.             prev = temp;
65.             temp = temp->right;
66.         }
67.         else
68.         {
69.             prev = temp;
70.             temp = temp->left;
71.         }
72.     }
73.     temp = (struct node *)malloc(sizeof(struct node));
74.     temp->a = num;
75.     if (num >= prev->a)
76.     {
77.         prev->right = temp;
78.     }
79.     else
80.     {
81.         prev->left = temp;
82.     }
83. }
84. }
85.
86. void DFS(struct node *head)
87. {
88.     if (head)
89.     {
90.         if (head->left)
91.         {
92.             DFS(head->left);
93.         }
94.         if (head->right)
95.         {
96.             DFS(head->right);
97.         }
98.         printf("%d  ", head->a);
99.     }
}
```

```

100.      }
101.
102.      void delete(struct node **head)
103.      {
104.          if (*head != NULL)
105.          {
106.              if ((*head)->left)
107.              {
108.                  delete(&(*head)->left);
109.              }
110.              if ((*head)->right)
111.              {
112.                  delete(&(*head)->right);
113.              }
114.              free(*head);
115.          }
116.      }

```

```

1. /*
2.  * C Program to Traverse the Tree Recursively
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. void infix(struct node *);
16. void postfix(struct node *);
17. void prefix(struct node *);
18. void delete(struct node **);
19.
20. int main()
21. {
22.     struct node *head = NULL;
23.     int choice = 0, num, flag = 0, key;
24.

```

```

25.     do
26.     {
27.         printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse
via prefix\n4. Traverse via postfix\n5. Exit\nChoice: ");
28.         scanf("%d", &choice);
29.         switch(choice)
30.         {
31.             case 1:
32.                 printf("Enter element to insert: ");
33.                 scanf("%d", &num);
34.                 generate(&head, num);
35.                 break;
36.             case 2:
37.                 infix(head);
38.                 break;
39.             case 3:
40.                 prefix(head);
41.                 break;
42.             case 4:
43.                 postfix(head);
44.                 break;
45.             case 5:
46.                 delete(&head);
47.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
48.                 break;
49.             default: printf("Not a valid input, try again\n");
50.         }
51.     } while (choice != 5);
52.     return 0;
53. }
54.
55. void generate(struct node **head, int num)
56. {
57.     struct node *temp = *head, *prev = *head;
58.
59.     if (*head == NULL)
60.     {
61.         *head = (struct node *)malloc(sizeof(struct node));
62.         (*head)->a = num;
63.         (*head)->left = (*head)->right = NULL;
64.     }
65.     else
66.     {

```

```
67.         while (temp != NULL)
68.         {
69.             if (num > temp->a)
70.             {
71.                 prev = temp;
72.                 temp = temp->right;
73.             }
74.             else
75.             {
76.                 prev = temp;
77.                 temp = temp->left;
78.             }
79.         }
80.         temp = (struct node *)malloc(sizeof(struct node));
81.         temp->a = num;
82.         if (num >= prev->a)
83.         {
84.             prev->right = temp;
85.         }
86.         else
87.         {
88.             prev->left = temp;
89.         }
90.     }
91. }
92.
93. void infix(struct node *head)
94. {
95.     if (head)
96.     {
97.         infix(head->left);
98.         printf("%d    ", head->a);
99.         infix(head->right);
100.        }
101.    }
102.
103. void prefix(struct node *head)
104. {
105.     if (head)
106.     {
107.         printf("%d    ", head->a);
108.         prefix(head->left);
109.         prefix(head->right);
```

```

110.         }
111.     }
112.
113.     void postfix(struct node *head)
114.     {
115.         if (head)
116.         {
117.             postfix(head->left);
118.             postfix(head->right);
119.             printf("%d    ", head->a);
120.         }
121.     }
122.
123.     void delete(struct node **head)
124.     {
125.         if (*head != NULL)
126.         {
127.             if ((*head)->left)
128.             {
129.                 delete(&(*head)->left);
130.             }
131.             if ((*head)->right)
132.             {
133.                 delete(&(*head)->right);
134.             }
135.             free(*head);
136.         }
137.     }

```

```

1. /*
2.  * C Program to Search an Element in a Tree Recursively
3. */
4.
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *left;
12.     struct node *right;
13. };

```

```
14.  
15. void generate(struct node **, int);  
16. int search(struct node *, int);  
17. void delete(struct node **);  
18.  
19. int main()  
20. {  
21.     struct node *head = NULL;  
22.     int choice = 0, num, flag = 0, key;  
23.  
24.     do  
25.     {  
26.         printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");  
27.         scanf("%d", &choice);  
28.         switch(choice)  
29.         {  
30.             case 1:  
31.                 printf("Enter element to insert: ");  
32.                 scanf("%d", &num);  
33.                 generate(&head, num);  
34.                 break;  
35.             case 2:  
36.                 printf("Enter key to search: ");  
37.                 scanf("%d", &key);  
38.                 flag = search(head, key);  
39.                 if (flag)  
40.                 {  
41.                     printf("Key found in tree\n");  
42.                 }  
43.                 else  
44.                 {  
45.                     printf("Key not found\n");  
46.                 }  
47.                 break;  
48.             case 3:  
49.                 delete(&head);  
50.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");  
51.                 break;  
52.             default:  
53.                 printf("Not a valid input, try again\n");  
54.             }  
55.         } while (choice != 3);  
56.
```

```
57.     return 0;
58. }
59.
60. void generate(struct node **head, int num)
61. {
62.     struct node *temp = *head, *prev = *head;
63.
64.     if (*head == NULL)
65.     {
66.         *head = (struct node *)malloc(sizeof(struct node));
67.         (*head)->a = num;
68.         (*head)->left = (*head)->right = NULL;
69.     }
70.     else
71.     {
72.         while (temp != NULL)
73.         {
74.             if (num > temp->a)
75.             {
76.                 prev = temp;
77.                 temp = temp->right;
78.             }
79.             else
80.             {
81.                 prev = temp;
82.                 temp = temp->left;
83.             }
84.         }
85.         temp = (struct node *)malloc(sizeof(struct node));
86.         temp->a = num;
87.         if (num >= prev->a)
88.         {
89.             prev->right = temp;
90.         }
91.         else
92.         {
93.             prev->left = temp;
94.         }
95.     }
96. }
97.
98. int search(struct node *head, int key)
99. {
```

```

100.         while (head != NULL)
101.         {
102.             if (key > head->a)
103.             {
104.                 return search(head->right, key);
105.             }
106.             else if (key < head->a)
107.             {
108.                 return search(head->left, key);
109.             }
110.             else
111.             {
112.                 return 1;
113.             }
114.         }
115.
116.         return 0;
117.     }
118.
119.     void delete(struct node **head)
120.     {
121.         if (*head != NULL)
122.         {
123.             if ((*head)->left)
124.             {
125.                 delete(&(*head)->left);
126.             }
127.             if ((*head)->right)
128.             {
129.                 delete(&(*head)->right);
130.             }
131.             free(*head);
132.         }
133.     }

```

```

1. /*
2.  * C Program to Traverse the Tree Non-Recursively
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node

```

```

8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. int search(struct node *, int);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 printf("Enter key to search: ");
36.                 scanf("%d", &key);
37.                 flag = search(head, key);
38.                 if (flag)
39.                 {
40.                     printf("Key found in tree\n");
41.                 }
42.                 else
43.                 {
44.                     printf("Key not found\n");
45.                 }
46.                 break;
47.             case 3:
48.                 delete(&head);
49.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
50.                 break;
}

```

```
51.         default: printf("Not a valid input, try again\n");
52.     }
53. } while (choice != 3);
54. return 0;
55. }
56.
57. void generate(struct node **head, int num)
58. {
59.     struct node *temp = *head, *prev = *head;
60.
61.     if (*head == NULL)
62.     {
63.         *head = (struct node *)malloc(sizeof(struct node));
64.         (*head)->a = num;
65.         (*head)->left = (*head)->right = NULL;
66.     }
67.     else
68.     {
69.         while (temp != NULL)
70.         {
71.             if (num > temp->a)
72.             {
73.                 prev = temp;
74.                 temp = temp->right;
75.             }
76.             else
77.             {
78.                 prev = temp;
79.                 temp = temp->left;
80.             }
81.         }
82.         temp = (struct node *)malloc(sizeof(struct node));
83.         temp->a = num;
84.         if (num >= prev->a)
85.         {
86.             prev->right = temp;
87.         }
88.         else
89.         {
90.             prev->left = temp;
91.         }
92.     }
93. }
```

```

94.
95. int search(struct node *head, int key)
96. {
97.     while (head != NULL)
98.     {
99.         if (key > head->a)
100.             {
101.                 head = head->right;
102.             }
103.         else if (key < head->a)
104.             {
105.                 head = head->left;
106.             }
107.         else
108.             {
109.                 return 1;
110.             }
111.     }
112.     return 0;
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }
129. }
```

```

1. /*
2.  * C Program to Traverse the Tree Non-Recursively
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
```

```
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. int search(struct node *, int);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 printf("Enter key to search: ");
36.                 scanf("%d", &key);
37.                 flag = search(head, key);
38.                 if (flag)
39.                 {
40.                     printf("Key found in tree\n");
41.                 }
42.                 else
43.                 {
44.                     printf("Key not found\n");
45.                 }
46.                 break;
47.             case 3:
48.                 delete(&head);
```

```
49.         printf("Memory Cleared\nPROGRAM TERMINATED\n");
50.         break;
51.     default: printf("Not a valid input, try again\n");
52. }
53. } while (choice != 3);
54. return 0;
55. }
56.
57. void generate(struct node **head, int num)
58. {
59.     struct node *temp = *head, *prev = *head;
60.
61.     if (*head == NULL)
62.     {
63.         *head = (struct node *)malloc(sizeof(struct node));
64.         (*head)->a = num;
65.         (*head)->left = (*head)->right = NULL;
66.     }
67.     else
68.     {
69.         while (temp != NULL)
70.         {
71.             if (num > temp->a)
72.             {
73.                 prev = temp;
74.                 temp = temp->right;
75.             }
76.             else
77.             {
78.                 prev = temp;
79.                 temp = temp->left;
80.             }
81.         }
82.         temp = (struct node *)malloc(sizeof(struct node));
83.         temp->a = num;
84.         if (num >= prev->a)
85.         {
86.             prev->right = temp;
87.         }
88.         else
89.         {
90.             prev->left = temp;
91.         }
92.     }
93. }
```

```

92.     }
93. }
94.
95. int search(struct node *head, int key)
96. {
97.     while (head != NULL)
98.     {
99.         if (key > head->a)
100.             {
101.                 head = head->right;
102.             }
103.         else if (key < head->a)
104.             {
105.                 head = head->left;
106.             }
107.         else
108.             {
109.                 return 1;
110.             }
111.     }
112.     return 0;
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }
129. }
```

```

1. /*
2.  * C Program for Depth First Binary Tree Search without using
3.  * Recursion
```

```

4.  /*
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *left;
12.     struct node *right;
13.     int visited;
14. };
15.
16. void generate(struct node **, int);
17. void DFS(struct node *);
18. void delete(struct node **);
19.
20. int main()
21. {
22.     struct node *head = NULL;
23.     int choice = 0, num, flag = 0, key;
24.
25.     do
26.     {
27.         printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
Exit\nChoice: ");
28.         scanf("%d", &choice);
29.         switch(choice)
30.         {
31.             case 1:
32.                 printf("Enter element to insert: ");
33.                 scanf("%d", &num);
34.                 generate(&head, num);
35.                 break;
36.             case 2:
37.                 DFS(head);
38.                 break;
39.             case 3:
40.                 delete(&head);
41.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
42.                 break;
43.             default:
44.                 printf("Not a valid input, try again\n");
45.         }
}

```

```
46.     } while (choice != 3);
47.
48.     return 0;
49. }
50.
51. void generate(struct node **head, int num)
52. {
53.     struct node *temp = *head, *prev = *head;
54.
55.     if (*head == NULL)
56.     {
57.         *head = (struct node *)malloc(sizeof(struct node));
58.         (*head)->a = num;
59.         (*head)->visited = 0;
60.         (*head)->left = (*head)->right = NULL;
61.     }
62.     else
63.     {
64.         while (temp != NULL)
65.         {
66.             if (num > temp->a)
67.             {
68.                 prev = temp;
69.                 temp = temp->right;
70.             }
71.             else
72.             {
73.                 prev = temp;
74.                 temp = temp->left;
75.             }
76.         }
77.         temp = (struct node *)malloc(sizeof(struct node));
78.         temp->a = num;
79.         temp->visited = 0;
80.         if (temp->a >= prev->a)
81.         {
82.             prev->right = temp;
83.         }
84.         else
85.         {
86.             prev->left = temp;
87.         }
88.     }
```

```
89. }
90.
91. void DFS(struct node *head)
92. {
93.     struct node *temp = head, *prev;
94.
95.     printf("On DFS traversal we get:\n");
96.     while (temp && !temp->visited)
97.     {
98.         if (temp->left && !temp->left->visited)
99.         {
100.             temp = temp->left;
101.         }
102.         else if (temp->right && !temp->right->visited)
103.         {
104.             temp = temp->right;
105.         }
106.         else
107.         {
108.             printf("%d ", temp->a);
109.             temp->visited = 1;
110.             temp = head;
111.         }
112.     }
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }
129. }
```

```

1. /*
2.  * C Program to Find Nth Node in the Inorder Traversal of a Tree
3. */
4.
5. typedef struct node
6. {
7.     int value;
8.     struct node *left;
9.     struct node *right;
10. }newnode;
11.
12. newnode *root;
13. static ctr;
14.
15. void nthnode(newnode *root, int n, newnode **nthnode);
16. int main()
17. {
18.     newnode *temp;
19.     root=0;
20.
21.     // Construct the tree
22.     add(19);
23.     add(20);
24.     add(11);
25.     inorder(root);
26.     // Get the pointer to the nth Inorder node
27.     nthinorder(root, 6, &temp);
28.     printf("\n[%d]\n", temp->value);
29.     return(0);
30. }
31.
32. // Get the pointer to the nth inorder node in "nthnode"
33. void nthinorder(newnode *root, int n, newnode **nthnode)
34. {
35.     static whichnode;
36.     static found;
37.
38.     if (!found)
39.     {
40.         if (root)
41.         {
42.             nthinorder(root->left, n , nthnode);
43.             if (++whichnode == n)

```

```
44.         {
45.             printf("\n Found %dth node\n", n);
46.             found = 1;
47.             *nthnode = root;
48.         }
49.         nthinorder(root->right, n, nthnode);
50.     }
51. }
52. }
53.
54.inorder(newnode *root)
55.{ 
56.}
57.// Add value to a Binary Search Tree
58.add(int value)
59.{
60.    newnode *temp, *prev, *cur;
61.
62.    temp = malloc(sizeof(newnode));
63.    temp->value = value;
64.    temp->left = 0;
65.    temp->right = 0;
66.    if (root == 0)
67.    {
68.        root = temp;
69.    }
70.    else
71.    {
72.        prev = 0;
73.        cur = root;
74.        while(cur)
75.        {
76.            prev = cur;
77.            cur = (value < cur->value)? cur->left : cur->right;
78.        }
79.        if (value > prev->value)
80.            prev->right = temp;
81.        else
82.            prev->left = temp;
83.    }
84.}
```

```

1. /*
2.  * C Program for Depth First Binary Tree Search without using
3.  * Recursion
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *left;
12.     struct node *right;
13.     int visited;
14. };
15.
16. void generate(struct node **, int);
17. void DFS(struct node *);
18. void delete(struct node **);
19.
20. int main()
21. {
22.     struct node *head = NULL;
23.     int choice = 0, num, flag = 0, key;
24.
25.     do
26.     {
27.         printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
Exit\nChoice: ");
28.         scanf("%d", &choice);
29.         switch(choice)
30.         {
31.             case 1:
32.                 printf("Enter element to insert: ");
33.                 scanf("%d", &num);
34.                 generate(&head, num);
35.                 break;
36.             case 2:
37.                 DFS(head);
38.                 break;
39.             case 3:
40.                 delete(&head);
41.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
42.                 break;

```

```
43.     default:
44.         printf("Not a valid input, try again\n");
45.     }
46. } while (choice != 3);
47.
48. return 0;
49. }
50.
51. void generate(struct node **head, int num)
52. {
53.     struct node *temp = *head, *prev = *head;
54.
55.     if (*head == NULL)
56.     {
57.         *head = (struct node *)malloc(sizeof(struct node));
58.         (*head)->a = num;
59.         (*head)->visited = 0;
60.         (*head)->left = (*head)->right = NULL;
61.     }
62.     else
63.     {
64.         while (temp != NULL)
65.         {
66.             if (num > temp->a)
67.             {
68.                 prev = temp;
69.                 temp = temp->right;
70.             }
71.             else
72.             {
73.                 prev = temp;
74.                 temp = temp->left;
75.             }
76.         }
77.         temp = (struct node *)malloc(sizeof(struct node));
78.         temp->a = num;
79.         temp->visited = 0;
80.         if (temp->a >= prev->a)
81.         {
82.             prev->right = temp;
83.         }
84.         else
85.         {
```

```

86.         prev->left = temp;
87.     }
88. }
89. }
90.
91. void DFS(struct node *head)
92. {
93.     struct node *temp = head, *prev;
94.
95.     printf("On DFS traversal we get:\n");
96.     while (temp && !temp->visited)
97.     {
98.         if (temp->left && !temp->left->visited)
99.         {
100.             temp = temp->left;
101.         }
102.         else if (temp->right && !temp->right->visited)
103.         {
104.             temp = temp->right;
105.         }
106.         else
107.         {
108.             printf("%d ", temp->a);
109.             temp->visited = 1;
110.             temp = head;
111.         }
112.     }
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }

```

129. }

```
1. /*
2. * C Program to Find Nth Node in the Inorder Traversal of a Tree
3. */
4.
5. typedef struct node
6. {
7.     int value;
8.     struct node *left;
9.     struct node *right;
10. }newnode;
11.
12. newnode *root;
13. static ctr;
14.
15. void nthnode(newnode *root, int n, newnode **nthnode);
16. int main()
17. {
18.     newnode *temp;
19.     root=0;
20.
21.     // Construct the tree
22.     add(19);
23.     add(20);
24.     add(11);
25.     inorder(root);
26.     // Get the pointer to the nth Inorder node
27.     nthinorder(root, 6, &temp);
28.     printf("\n[%d]\n", temp->value);
29.     return(0);
30. }
31.
32. // Get the pointer to the nth inorder node in "nthnode"
33. void nthinorder(newnode *root, int n, newnode **nthnode)
34. {
35.     static whichnode;
36.     static found;
37.
38.     if (!found)
39.     {
40.         if (root)
```

```

41.     {
42.         nthinorder(root->left, n , nthnode);
43.         if (++whichnode == n)
44.         {
45.             printf("\n Found %dth node\n", n);
46.             found = 1;
47.             *nthnode = root;
48.         }
49.         nthinorder(root->right, n , nthnode);
50.     }
51. }
52. }
53.
54. inorder(newnode *root)
55. {
56. }
57. // Add value to a Binary Search Tree
58. add(int value)
59. {
60.     newnode *temp, *prev, *cur;
61.
62.     temp = malloc(sizeof(newnode));
63.     temp->value = value;
64.     temp->left = 0;
65.     temp->right = 0;
66.     if (root == 0)
67.     {
68.         root = temp;
69.     }
70.     else
71.     {
72.         prev = 0;
73.         cur = root;
74.         while(cur)
75.         {
76.             prev = cur;
77.             cur = (value < cur->value)? cur->left : cur->right;
78.         }
79.         if (value > prev->value)
80.             prev->right = temp;
81.         else
82.             prev->left = temp;
83.     }

```

84. }

```
1. /*
2. * C Program to Find the Largest value in a Tree using
3. * Inorder Traversal
4. *          40
5. *         /\
6. *        20 60
7. *       / \ \
8. *      10 30 80
9. *     /
10. *    90
11. */
12.#include <stdio.h>
13.#include <stdlib.h>
14.
15.struct btnode
16.{
17.    int value;
18.    struct btnode *left, *right;
19.};
20.typedef struct btnode node;
21.
22.* function prototypes */
23.
24.void insert(node *, node *);
25.void inorder(node *);
26.void largest(node *);
27.
28.void main()
29.{
30.    node *root = NULL, *new = NULL ;
31.    int num = 1;
32.
33.    printf("Enter the elements of the tree(enter 0 to exit)\n");
34.    while (1)
35.    {
36.        scanf("%d", &num);
37.        if (num == 0)
38.            break;
39.        new = malloc(sizeof(node));
40.        new->left = new->right = NULL;
```

```
41.         new->value = num;
42.         if (root == NULL)
43.             root = new;
44.         else
45.         {
46.             insert(new, root);
47.         }
48.     }
49.     printf("elements in a tree in inorder are\n");
50.     inorder(root);
51.     largest(root);
52. }
53.
54./* displaying nodes of a tree using inorder */
55.
56.void inorder(node *root)
57.{
58.    if (root != NULL)
59.    {
60.        inorder(root->left);
61.        printf("%d -> ", root->value);
62.        inorder(root->right);
63.    }
64.}
65.
66./* inserting nodes into the tree */
67.
68.void insert(node * new , node *root)
69.{
70.    if (new->value > root->value)
71.    {
72.        if (root->right == NULL)
73.            root->right = new;
74.        else
75.            insert (new, root->right);
76.    }
77.    if (new->value < root->value)
78.    {
79.        if (root->left == NULL)
80.            root->left = new;
81.        else
82.            insert(new, root->left);
83.    }
}
```

```

84. }
85.
86./* finding Largest node in a tree */
87.void largest(node *root)
88.{
89.    if (root->right == NULL)
90.    {
91.        printf("largest element is %d", root->value);
92.    }
93.    while (root != NULL && root->right != NULL)
94.    {
95.        root = root->right;
96.    }
97.    printf("\nlargest value is %d\n", root->value);
98.}

99./*
100. * C Program to Implement Depth First Search Traversal using Post Order
101.      50
102.      / \
103.     20   30
104.    / \
105.   70   80
106.  / \   \
107. 10  40   60
108. (50, 20, 30, 70, 80, 10, 40, 60)
109. */
110. #include <stdio.h>
111. #include <stdlib.h>
112.
113. struct btnode {
114.     int value;
115.     struct btnode *l;
116.     struct btnode *r;
117. };
118.
119. typedef struct btnode bt;
120. bt *root;
121. bt *new, *list;
122. bt *create_node();
123. void display(bt *);
124. void construct_tree();
125. void dfs(bt *);
126.

```

```
127. void main()
128. {
129.     construct_tree();
130.     display(root);
131.     printf("\n");
132.     printf("Depth first traversal\n ");
133.     dfs(root);
134. }
135.
136. /* Creates an empty node */
137. bt * create_node()
138. {
139.     new=(bt *)malloc(sizeof(bt));
140.     new->l = NULL;
141.     new->r = NULL;
142. }
143.
144. /* Constructs a tree */
145. void construct_tree()
146. {
147.     root = create_node();
148.     root->value = 50;
149.     root->l = create_node();
150.     root->l->value = 20;
151.     root->r = create_node();
152.     root->r->value = 30;
153.     root->l->l = create_node();
154.     root->l->l->value = 70;
155.     root->l->r = create_node();
156.     root->l->r->value = 80;
157.     root->l->r->r = create_node();
158.     root->l->r->r->value = 60;
159.     root->l->l->l = create_node();
160.     root->l->l->l->value = 10;
161.     root->l->l->r = create_node();
162.     root->l->l->r->value = 40;
163. }
164.
165. /* Display the elements in a tree using inorder */
166. void display(bt * list)
167. {
168.     if (list == NULL)
169.     {
```

```

170.         return;
171.     }
172.     display(list->l);
173.     printf("->%d", list->value);
174.     display(list->r);
175. }
176.
177. /* Dfs traversal using post order */
178. void dfs(bt * list)
179. {
180.     if (list == NULL)
181.     {
182.         return;
183.     }
184.     dfs(list->l);
185.     dfs(list->r);
186.     printf("->%d ", list->value);
187. }
```

```

1. /*
2. * C Program to Find the Largest value in a Tree using
3. * Inorder Traversal
4. *          40
5. *         /\
6. *        20 60
7. *       / \ \
8. *      10 30 80
9. *      \
10. *      90
11. */
12. #include <stdio.h>
13. #include <stdlib.h>
14.
15. struct btnode
16. {
17.     int value;
18.     struct btnode *left, *right;
19. };
20. typedef struct btnode node;
21.
22. /* function prototypes */
23.
```

```
24. void insert(node *, node *);  
25. void inorder(node *);  
26. void largest(node *);  
27.  
28. void main()  
29. {  
30.     node *root = NULL, *new = NULL ;  
31.     int num = 1;  
32.  
33.     printf("Enter the elements of the tree(enter 0 to exit)\n");  
34.     while (1)  
35.     {  
36.         scanf("%d", &num);  
37.         if (num == 0)  
38.             break;  
39.         new = malloc(sizeof(node));  
40.         new->left = new->right = NULL;  
41.         new->value = num;  
42.         if (root == NULL)  
43.             root = new;  
44.         else  
45.         {  
46.             insert(new, root);  
47.         }  
48.     }  
49.     printf("elements in a tree in inorder are\n");  
50.     inorder(root);  
51.     largest(root);  
52. }  
53.  
54. /* displaying nodes of a tree using inorder */  
55.  
56. void inorder(node *root)  
57. {  
58.     if (root != NULL)  
59.     {  
60.         inorder(root->left);  
61.         printf("%d -> ", root->value);  
62.         inorder(root->right);  
63.     }  
64. }  
65.  
66. /* inserting nodes into the tree */
```

```

67.
68. void insert(node * new , node *root)
69. {
70.     if (new->value > root->value)
71.     {
72.         if (root->right == NULL)
73.             root->right = new;
74.         else
75.             insert (new, root->right);
76.     }
77.     if (new->value < root->value)
78.     {
79.         if (root->left == NULL)
80.             root->left = new;
81.         else
82.             insert(new, root->left);
83.     }
84. }
85.
86./* finding Largest node in a tree */
87.void largest(node *root)
88.{
89.    if (root->right == NULL)
90.    {
91.        printf("largest element is %d", root->value);
92.    }
93.    while (root != NULL && root->right != NULL)
94.    {
95.        root = root->right;
96.    }
97.    printf("\nlargest value is %d\n", root->value);
98. }
```

```

1. /*
2.  * C Program to Build Binary Tree if Inorder or Postorder Traversal as Input
3. *
4. *          40
5. *          / \
6. *          20   60
7. *          / \   \
8. *          10  30   80
9. *                  \

```

```

10. *
11. *          90
12. *      (Given Binary Search Tree)
13. */
14. #include <stdio.h>
15. #include <stdlib.h>
16.
17. struct btnode
18. {
19.     int value;
20.     struct btnode *l;
21.     struct btnode *r;
22. }*root = NULL, *temp = NULL;
23.
24. typedef struct btnode N;
25. void insert();
26. N* bt(int arr[],int,int);
27. N* new(int);
28. void inorder(N *t);
29. void create();
30. void search(N *t);
31. void preorder(N *t);
32. void postorder(N *t);
33.
34. void main()
35. {
36.     int ch, i, n;
37.     int arr[] = {10, 20, 30, 40, 60, 80, 90};
38.     n = sizeof(arr) / sizeof(arr[0]);
39.     printf("\n1- Inorder\n");
40.     printf("2 - postorder\n");
41.     printf("\nEnter choice : ");
42.     scanf("%d", &ch);
43.     switch (ch)
44.     {
45.     case 1:
46.         root = bt(arr, 0, n-1);
47.         printf("Given inorder traversal as input\n");
48.         for (i = 0;i<= 6;i++)
49.             printf("%d->", arr[i]);
50.         printf("\npreorder traversal of tree\n");
51.         preorder(root);
52.         printf("\ninorder traversal of tree\n");

```

```

53.     inorder(root);
54.     printf("\npostorder traversal of tree\n");
55.     postorder(root);
56.     break;
57. case 2:
58.     insert();
59.     printf("\npreorder traversal of tree\n");
60.     preorder(root);
61.     printf("\nInorder traversal of tree\n");
62.     inorder(root);
63.     printf("\npostorder traversal of tree\n");
64.     postorder(root);
65.     break;
66. default:printf("enter correct choice");
67. }
68. }
69.

70./* To create a new node */
71.N* new(int val)
72.{
73.    N* node = (N*)malloc(sizeof(N));
74.
75.    node->value = val;
76.    node->l = NULL;
77.    node->r = NULL;
78.    return node;
79.}
80.

81./* To create a balanced binary search tree */
82.N* bt(int arr[], int first, int last)
83.{
84.    int mid;
85.
86.    N* root = (N*)malloc(sizeof(N));
87.    if (first > last)
88.        return NULL;
89.    mid = (first + last) / 2;
90.    root = new(arr[mid]);
91.    root->l = bt(arr, first, mid - 1);
92.    root->r = bt(arr, mid + 1, last);
93.    return root;
94.}
95.

```

```

96./* Insert a node in the tree */
97.void insert()
98.{
99.    int arr1[] = {10, 30, 20, 90, 80, 60, 40}, i;
100.
101.        printf("Given post order traversal array\n");
102.        for (i = 0;i <= 6;i++)
103.        {
104.            printf("%d->", arr1[i]);
105.        }
106.        for (i = 6;i >= 0;i--)
107.        {
108.            create(arr1[i]);
109.            if (root == NULL)
110.                root = temp;
111.            else
112.                search(root);
113.        }
114.    }
115.
116.    /*Create a node */
117.    void create(int data)
118.    {
119.        temp = (N *)malloc(1*sizeof(N));
120.
121.        temp->value = data;
122.        temp->l = temp->r = NULL;
123.    }
124.
125.    /* Search for the appropriate position to insert the new node */
126.    void search(N *t)
127.    {
128.        if ((temp->value>t->value)&&(t->r != NULL))
129.            search(t->r);
130.        else if ((temp->value>t->value)&&(t->r == NULL))
131.            t->r = temp;
132.        else if ((temp->value<t->value)&&(t->l != NULL))
133.            search(t->l);
134.        else if ((temp->value<t->value)&&(t->l == NULL))
135.            t->l = temp;
136.    }
137.
138.    /* to display inorder of tree */

```

```

139.     void inorder(N *t)
140.     {
141.         if (t->l != NULL)
142.             inorder(t->l);
143.         printf("%d->", t->value);
144.         if (t->r != NULL)
145.             inorder(t->r);
146.     }
147.
148.     /* to display preorder traversal of tree */
149.     void preorder(N *t)
150.     {
151.         printf("%d->", t->value);
152.         if (t->l != NULL)
153.             inorder(t->l);
154.         if (t->r != NULL)
155.             inorder(t->r);
156.     }
157.
158.     /* to display postorder traversal of tree */
159.     void postorder(N *t)
160.     {
161.         if (t->l != NULL)
162.             inorder(t->l);
163.         if (t->r != NULL)
164.             inorder(t->r);
165.         printf("%d->", t->value);
166.     }

```

## 2. C Examples on Heap and Binary Tree Implementation

```

1. /*
2.  * C Program to Implement Binary Tree using Linked List
3.  */
4. #include <stdio.h>
5. #include <malloc.h>
6.
7. struct node {
8.     struct node * left;
9.     char data;

```

```

10.     struct node * right;
11. };
12.
13. struct node *constructTree( int );
14. void inorder(struct node *);
15.
16. char array[ ] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', '\0', '\0', 'H' };
17. int leftcount[ ] = { 1, 3, 5, -1, 9, -1, -1, -1, -1, -1 };
18. int rightcount[ ] = { 2, 4, 6, -1, -1, -1, -1, -1, -1, -1 };
19.
20. void main() {
21.     struct node *root;
22.     root = constructTree( 0 );
23.     printf("In-order Traversal: \n");
24.     inorder(root);
25. }
26.
27. struct node * constructTree( int index ) {
28.     struct node *temp = NULL;
29.     if (index != -1) {
30.         temp = (struct node *)malloc( sizeof ( struct node ) );
31.         temp->left = constructTree( leftcount[index] );
32.         temp->data = array[index];
33.         temp->right = constructTree( rightcount[index] );
34.     }
35.     return temp;
36. }
37.
38. void inorder( struct node *root ) {
39.     if (root != NULL) {
40.         inorder(root->left);
41.         printf("%c\t", root->data);
42.         inorder(root->right);
43.     }
44. }
```

```

1. /*
2.  * C Program to Implement a Heap & provide Insertion & Deletion Operation
3.  */
4. #include <stdio.h>
5.
6. int array[100], n;
```

```

7. main()
8. {
9.     int choice, num;
10.    n = 0; /*Represents number of nodes in the heap*/
11.    while(1)
12.    {
13.        printf("1.Insert the element \n");
14.        printf("2.Delete the element \n");
15.        printf("3.Display all elements \n");
16.        printf("4.Quit \n");
17.        printf("Enter your choice : ");
18.        scanf("%d", &choice);
19.        switch(choice)
20.        {
21.            case 1:
22.                printf("Enter the element to be inserted to the list : ");
23.                scanf("%d", &num);
24.                insert(num, n);
25.                n = n + 1;
26.                break;
27.            case 2:
28.                printf("Enter the elements to be deleted from the list: ");
29.                scanf("%d", &num);
30.                delete(num);
31.                break;
32.            case 3:
33.                display();
34.                break;
35.            case 4:
36.                exit(0);
37.            default:
38.                printf("Invalid choice \n");
39.        }/*End of switch */
40.    }/*End of while */
41. }/*End of main()*/
42.
43.display()
44.{
45.    int i;
46.    if (n == 0)
47.    {
48.        printf("Heap is empty \n");
49.        return;

```

```

50.     }
51.     for (i = 0; i < n; i++)
52.         printf("%d ", array[i]);
53.     printf("\n");
54. }/*End of display()*/
55.
56.insert(int num, int location)
57.{
58.    int parentnode;
59.    while (location > 0)
60.    {
61.        parentnode =(location - 1)/2;
62.        if (num <= array[parentnode])
63.        {
64.            array[location] = num;
65.            return;
66.        }
67.        array[location] = array[parentnode];
68.        location = parentnode;
69.    }/*End of while*/
70.    array[0] = num; /*assign number to the root node */
71.}/*End of insert()*/
72.
73.delete(int num)
74.{
75.    int left, right, i, temp, parentnode;
76.
77.    for (i = 0; i < num; i++) {
78.        if (num == array[i])
79.            break;
80.    }
81.    if (num != array[i])
82.    {
83.        printf("%d not found in heap list\n", num);
84.        return;
85.    }
86.    array[i] = array[n - 1];
87.    n = n - 1;
88.    parentnode =(i - 1) / 2; /*find parentnode of node i */
89.    if (array[i] > array[parentnode])
90.    {
91.        insert(array[i], i);
92.        return;

```

```

93.     }
94.     left = 2 * i + 1; /*left child of i*/
95.     right = 2 * i + 2; /* right child of i*/
96.     while (right < n)
97.     {
98.         if (array[i] >= array[left] && array[i] >= array[right])
99.             return;
100.            if (array[right] <= array[left])
101.            {
102.                temp = array[i];
103.                array[i] = array[left];
104.                array[left] = temp;
105.                i = left;
106.            }
107.            else
108.            {
109.                temp = array[i];
110.                array[i] = array[right];
111.                array[right] = temp;
112.                i = right;
113.            }
114.            left = 2 * i + 1;
115.            right = 2 * i + 2;
116.        }/*End of while*/
117.        if (left == n - 1 && array[i])    {
118.            temp = array[i];
119.            array[i] = array[left];
120.            array[left] = temp;
121.        }
122.    }

```

```

1. /*
2.  * C Program to Construct a B Tree
3.  */
4.
5. ****
6. * binarytree.h
7. ****
8.
9. typedef char DATA;
10.
11. struct node

```

```
12. {
13.     DATA d;
14.     struct node *left;
15.     struct node *right;
16. };
17.
18. typedef struct node NODE;
19. typedef NODE *BTREE;
20.
21. BTREE newnode(void);
22. BTREE init_node(DATA d, BTREE p1, BTREE p2);
23. BTREE create_tree(DATA a[], int i, int size);
24. void preorder (BTREE root);
25. void inorder (BTREE root);
26. void postorder (BTREE root);
27.
28. ****
29. * binarytree.c:
30. ****
31. #include <assert.h>
32. #include <stdio.h>
33. #include <stdlib.h>
34. #include "binarytree.h"
35.
36. BTREE new_node()
37. {
38.     return ((BTREE)malloc(sizeof(NODE)));
39. }
40.
41. BTREE init_node(DATA d1, BTREE p1, BTREE p2)
42. {
43.     BTREE t;
44.
45.     t = new_node();
46.     t->d = d1;
47.     t->left = p1;
48.     t->right = p2;
49.     return t;
50. }
51.
52. /* create a linked binary tree from an array */
53. BTREE create_tree(DATA a[], int i, int size)
54. {
```

```

55.     if (i >= size)
56.         return NULL;
57.     else
58.         return(init_node(a[i],
59.             create_tree(a, 2*i+1, size),
60.             create_tree(a, 2*i+2, size)));
61. }
62.
63./* preorder traversal */
64.void preorder (BTREE root)
65.{
66.    if (root != NULL) {
67.        printf("%c ", root->d);
68.        preorder(root -> left);
69.        preorder(root -> right);
70.    }
71. }
72.
73./* Inorder traversal */
74.void inorder (BTREE root)
75.{
76.    if (root != NULL) {
77.        inorder(root -> left);
78.        printf("%c ", root->d);
79.        inorder(root -> right);
80.    }
81. }
82.
83./* postorder binary tree traversal */
84.
85.void postorder (BTREE root)
86.{
87.    if (root != NULL) {
88.        postorder(root -> left);
89.        postorder(root -> right);
90.        printf("%c ", root->d);
91.    }
92. }
93.
94. ****
95. * pgm.c
96. ****
97.#include <assert.h>
```

```

98. #include <stdio.h>
99. #include <stdlib.h>
100.
101.     #include "binarytree.c"
102.     #define ARRAY_SIZE 10
103.     int main(void)
104.     {
105.         char a[ARRAY_SIZE] = {'g','d','i','b','f','h','j','a','c','e'};
106.         BTREE root;
107.
108.         root = create_tree(a, 0, ARRAY_SIZE) ;
109.         assert(root != NULL);
110.         printf("PREORDER\n");
111.         preorder(root);
112.         printf("\n");
113.         printf("INORDER\n");
114.         inorder(root);
115.         printf("\n");
116.
117.         printf("POSTORDER\n");
118.         postorder(root);
119.         printf("\n");
120.     }

```

```

1. /*
2.  * C Program to Construct a Binary Search Tree and perform deletion, inorder
3.  traversal on it
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct btnode
9. {
10.     int value;
11.     struct btnode *l;
12.     struct btnode *r;
13. }*root = NULL, *temp = NULL, *t2, *t1;
14. void delete1();
15. void insert();
16. void delete();
17. void inorder(struct btnode *t);

```

```
18. void create();
19. void search(struct btnode *t);
20. void preorder(struct btnode *t);
21. void postorder(struct btnode *t);
22. void search1(struct btnode *t,int data);
23. int smallest(struct btnode *t);
24. int largest(struct btnode *t);
25.
26. int flag = 1;
27.
28. void main()
29. {
30.     int ch;
31.
32.     printf("\nOPERATIONS ---");
33.     printf("1 - Insert an element into tree\n");
34.     printf("2 - Delete an element from the tree\n");
35.     printf("3 - Inorder Traversal\n");
36.     printf("4 - Preorder Traversal\n");
37.     printf("5 - Postorder Traversal\n");
38.     printf("6 - Exit\n");
39.     while(1)
40.     {
41.         printf("\nEnter your choice : ");
42.         scanf("%d", &ch);
43.         switch (ch)
44.         {
45.             case 1:
46.                 insert();
47.                 break;
48.             case 2:
49.                 delete();
50.                 break;
51.             case 3:
52.                 inorder(root);
53.                 break;
54.             case 4:
55.                 preorder(root);
56.                 break;
57.             case 5:
58.                 postorder(root);
59.                 break;
60.             case 6:
```

```

61.         exit(0);
62.     default :
63.         printf("Wrong choice, Please enter correct choice  ");
64.         break;
65.     }
66. }
67. }
68.

69./* To insert a node in the tree */
70.void insert()
71.{
72.    create();
73.    if (root == NULL)
74.        root = temp;
75.    else
76.        search(root);
77.}
78.

79./* To create a node */
80.void create()
81.{
82.    int data;
83.

84.    printf("Enter data of node to be inserted : ");
85.    scanf("%d", &data);
86.    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
87.    temp->value = data;
88.    temp->l = temp->r = NULL;
89.}
90.

91./* Function to search the appropriate position to insert the new node */
92.void search(struct btnode *t)
93.{
94.    if ((temp->value > t->value) && (t->r != NULL))          /* value more than root
   node value insert at right */
95.        search(t->r);
96.    else if ((temp->value > t->value) && (t->r == NULL))
97.        t->r = temp;
98.    else if ((temp->value < t->value) && (t->l != NULL))      /* value less than root
   node value insert at left */
99.        search(t->l);
100.       else if ((temp->value < t->value) && (t->l == NULL))
101.           t->l = temp;

```

```
102.     }
103.
104.     /* recursive function to perform inorder traversal of tree */
105.     void inorder(struct btnode *t)
106.     {
107.         if (root == NULL)
108.         {
109.             printf("No elements in a tree to display");
110.             return;
111.         }
112.         if (t->l != NULL)
113.             inorder(t->l);
114.         printf("%d -> ", t->value);
115.         if (t->r != NULL)
116.             inorder(t->r);
117.     }
118.
119.     /* To check for the deleted node */
120.     void delete()
121.     {
122.         int data;
123.
124.         if (root == NULL)
125.         {
126.             printf("No elements in a tree to delete");
127.             return;
128.         }
129.         printf("Enter the data to be deleted : ");
130.         scanf("%d", &data);
131.         t1 = root;
132.         t2 = root;
133.         search1(root, data);
134.     }
135.
136.     /* To find the preorder traversal */
137.     void preorder(struct btnode *t)
138.     {
139.         if (root == NULL)
140.         {
141.             printf("No elements in a tree to display");
142.             return;
143.         }
144.         printf("%d -> ", t->value);
```

```

145.         if (t->l != NULL)
146.             preorder(t->l);
147.         if (t->r != NULL)
148.             preorder(t->r);
149.     }
150.
151.     /* To find the postorder traversal */
152.     void postorder(struct btnode *t)
153.     {
154.         if (root == NULL)
155.         {
156.             printf("No elements in a tree to display ");
157.             return;
158.         }
159.         if (t->l != NULL)
160.             postorder(t->l);
161.         if (t->r != NULL)
162.             postorder(t->r);
163.         printf("%d -> ", t->value);
164.     }
165.
166.     /* Search for the appropriate position to insert the new node */
167.     void search1(struct btnode *t, int data)
168.     {
169.         if ((data>t->value))
170.         {
171.             t1 = t;
172.             search1(t->r, data);
173.         }
174.         else if ((data < t->value))
175.         {
176.             t1 = t;
177.             search1(t->l, data);
178.         }
179.         else if ((data==t->value))
180.         {
181.             delete1(t);
182.         }
183.     }
184.
185.     /* To delete a node */
186.     void delete1(struct btnode *t)
187.     {

```

```
188.     int k;
189.
190.     /* To delete Leaf node */
191.     if ((t->l == NULL) && (t->r == NULL))
192.     {
193.         if (t1->l == t)
194.         {
195.             t1->l = NULL;
196.         }
197.         else
198.         {
199.             t1->r = NULL;
200.         }
201.         t = NULL;
202.         free(t);
203.         return;
204.     }
205.
206.     /* To delete node having one Left hand child */
207.     else if ((t->r == NULL))
208.     {
209.         if (t1 == t)
210.         {
211.             root = t->l;
212.             t1 = root;
213.         }
214.         else if (t1->l == t)
215.         {
216.             t1->l = t->l;
217.
218.         }
219.         else
220.         {
221.             t1->r = t->l;
222.         }
223.         t = NULL;
224.         free(t);
225.         return;
226.     }
227.
228.     /* To delete node having right hand child */
229.     else if (t->l == NULL)
230.     {
```

```

231.         if (t1 == t)
232.         {
233.             root = t->r;
234.             t1 = root;
235.         }
236.         else if (t1->r == t)
237.             t1->r = t->r;
238.         else
239.             t1->l = t->r;
240.         t == NULL;
241.         free(t);
242.         return;
243.     }
244.
245. /* To delete node having two child */
246. else if ((t->l != NULL) && (t->r != NULL))
247. {
248.     t2 = root;
249.     if (t->r != NULL)
250.     {
251.         k = smallest(t->r);
252.         flag = 1;
253.     }
254.     else
255.     {
256.         k = largest(t->l);
257.         flag = 2;
258.     }
259.     search1(root, k);
260.     t->value = k;
261. }
262.
263. }
264.
265. /* To find the smallest element in the right sub tree */
266. int smallest(struct btnode *t)
267. {
268.     t2 = t;
269.     if (t->l != NULL)
270.     {
271.         t2 = t;
272.         return(smallest(t->l));
273.     }

```

```

274.         else
275.             return (t->value);
276.     }
277.
278.     /* To find the largest element in the left sub tree */
279.     int largest(struct btnode *t)
280.     {
281.         if (t->r != NULL)
282.         {
283.             t2 = t;
284.             return(largest(t->r));
285.         }
286.         else
287.             return(t->value);
288.     }
289. /*
290.  * C Program To Find the Smallest and Largest Elements
291.  * in the Binary Search Tree
292.  *          40
293.  *         /   \
294.  *        20   60
295.  *       /   \   \
296.  *      10   30   80
297.  *           \
298.  *            90
299.  *      (Given Binary Search Tree)
300.  *
301. */
302. #include <stdio.h>
303. #include <stdlib.h>
304.
305. struct btnode
306. {
307.     int value;
308.     struct btnode *l;
309.     struct btnode *r;
310. }*root = NULL;
311.
312. typedef struct btnode N;
313. N* new(int);
314. void create();
315. void preorder(N *t);
316. void min_max(N *t);

```

```
317.  
318. void main()  
319. {  
320.     int choice;  
321.  
322.     create();  
323.     while (1)  
324.     {  
325.         printf("Enter the choice\n");  
326.         printf("1-Display : 2-Min & Max element : 3-Exit\n");  
327.         scanf("%d", &choice);  
328.         switch (choice)  
329.         {  
330.  
331.             case 1:  
332.                 printf("preorder preorder of tree elements\n");  
333.                 preorder(root);  
334.                 printf("\n");  
335.                 break;  
336.             case 2:  
337.                 min_max(root);  
338.                 break;  
339.             case 3:  
340.                 exit(0);  
341.             default:  
342.                 printf("Enter the right choice\n");  
343.             }  
344.         }  
345.     }  
346.  
347. /* creating temporary node */  
348. N* new(int data)  
349. {  
350.     N* temp = (N*)malloc(sizeof(N));  
351.     temp->value = data;  
352.     temp->l = NULL;  
353.     temp->r = NULL;  
354.  
355.     return(temp);  
356. }  
357.  
358. /* Creating the binary search tree */  
359. void create()
```

```
360. {
361.     root = new(40);
362.     root->l = new(20);
363.     root->r = new(60);
364.     root->l->l = new(10);
365.     root->l->r = new(30);
366.     root->r->r = new(80);
367.     root->r->r->r = new(90);
368. }
369.
370. /* To display preorder traversal of the tree */
371. void preorder(N *temp)
372. {
373.     printf("%d->", temp->value);
374.     if (temp->l != NULL)
375.         preorder(temp->l);
376.     if (temp->r != NULL)
377.         preorder(temp->r);
378. }
379.
380. /* TO find the minimum and maximum values in the given tree */
381. void min_max(N *temp)
382. {
383.     while (temp->l != NULL)
384.         temp = temp->l;
385.     printf("Minimum value = %d\n", temp->value);
386.     temp = root;
387.     while (temp->r != NULL)
388.         temp = temp->r;
389.     printf("Maximum value = %d\n", temp->value);
390. }
```

```
1. /*
2. * C Program to Construct a Balanced Binary Tree using Sorted Array
3. *
4. *           40
5. *         /   \
6. *        20   60
7. *       /   \   \
8. *      10  30  80
9. *             \
9. *             90
10. * (Given Binary Search Tree)
```

```

11. /*
12. #include <stdio.h>
13. #include <stdlib.h>
14.
15. struct btnode
16. {
17.     int value;
18.     struct btnode *l;
19.     struct btnode *r;
20. };
21.
22. typedef struct btnode N;
23. N* bst(int arr[], int first, int last);
24. N* new(int val);
25. void display(N *temp);
26.
27. int main()
28. {
29.     int arr[] = {10, 20, 30, 40, 60, 80, 90};
30.     N *root = (N*)malloc(sizeof(N));
31.     int n = sizeof(arr) / sizeof(arr[0]), i;
32.
33.     printf("Given sorted array is\n");
34.     for (i = 0;i < n;i++)
35.         printf("%d\t", arr[i]);
36.     root = bst(arr, 0, n - 1);
37.     printf("\n The preorder traversal of binary search tree is as follows\n");
38.     display(root);
39.     printf("\n");
40.     return 0;
41. }
42.
43. /* To create a new node */
44. N* new(int val)
45. {
46.     N* node = (N*)malloc(sizeof(N));
47.
48.     node->value = val;
49.     node->l = NULL;
50.     node->r = NULL;
51.     return node;
52. }
53.
```

```

54./* To create a balanced binary search tree */
55.N* bst(int arr[], int first, int last)
56.{
57.    int mid;
58.    N* temp = (N*)malloc(sizeof(N));
59.    if (first > last)
60.        return NULL;
61.    mid = (first + last) / 2;
62.    temp = new(arr[mid]);
63.    temp->l = bst(arr, first, mid - 1);
64.    temp->r = bst(arr, mid + 1, last);
65.    return temp;
66.}
67.
68./* To display the preorder */
69.void display(N *temp)
70.{
71.    printf("%d->", temp->value);
72.    if (temp->l != NULL)
73.        display(temp->l);
74.    if (temp->r != NULL)
75.        display(temp->r);
76.}

```

```

1. /*
2.  * C Program to Find the Common Ancestor and Print the Path
3. *
4. *          10
5. *         /   \
6. *        7    15
7. *       / \   / \
8. *      6  8 12  18
9. *     /     \
10.*    5       9
11.*          (Given Binary tree)
12.*/
13.#include <stdio.h>
14.#include <stdlib.h>
15.
16.struct btnode
17.{
18.    int value;

```

```
19.     struct btnode *l;
20.     struct btnode *r;
21. };
22.
23. typedef struct btnode N;
24.
25. N* new(int);
26. int count;
27.
28. void create();
29. void preorder(N *t);
30. void ancestor(N *t);
31. int search(N *t, int, int);
32. void path(int, int, int);
33.
34. N *root = NULL;
35.
36. void main()
37. {
38.     int choice;
39.
40.     create();
41.     while (1)
42.     {
43.         printf("Enter the choice\n");
44.         printf("1-Display : 2-path : 3-Exit\n");
45.         scanf("%d", &choice);
46.         switch (choice)
47.         {
48.             case 1:
49.                 printf("preorder display of tree elements\n");
50.                 preorder(root);
51.                 printf("\n");
52.                 break;
53.             case 2:
54.                 ancestor(root);
55.                 break;
56.             case 3:
57.                 exit(0);
58.             default:
59.                 printf("Enter the right choice\n");
60.         }
61.     }
}
```

```

62. }
63.
64./* creating temporary node */
65.N* new(int data)
66.{ 
67.    N* temp = (N*)malloc(sizeof(N));
68.    temp->value = data;
69.    temp->l = NULL;
70.    temp->r = NULL;
71.
72.    return(temp);
73. }
74.
75./* Creating the binary search tree */
76.void create()
77.{ 
78.    root = new(10);
79.    root->l = new(7);
80.    root->r = new(15);
81.    root->l->l = new(6);
82.    root->l->r = new(8);
83.    root->r->l = new(12);
84.    root->r->r = new(18);
85.    root->r->r->r = new(20);
86.    root->l->l->l = new(5);
87.    root->l->r->r = new(9);
88. }
89.
90./* To display the preorder traversal of the tree */
91.void preorder(N *temp)
92.{ 
93.    printf("%d->", temp->value);
94.    if (temp->l != NULL)
95.        preorder(temp->l);
96.    if (temp->r != NULL)
97.        preorder(temp->r);
98. }
99.
100.   /* to find common ancestor for given nodes */
101. void ancestor(N *temp)
102. {
103.     int a, b, anc = 0;
104.     count = 0;

```

```

105.
106.     printf("enter two node values to find common ancestor\n");
107.     scanf("%d", &a);
108.     scanf("%d", &b);
109.     count = search(root, a, b);
110.     if (count == 2)
111.     {
112.         while (temp->value != a && temp->value != b)
113.         {
114.             if ((temp->value > a)&&(temp->value > b))
115.             {
116.                 anc = temp->value;
117.                 temp = temp->l;
118.             }
119.             else if ((temp->value < a)&&(temp->value < b))
120.             {
121.                 anc = temp->value;
122.                 temp = temp->r;
123.             }
124.             else if ((temp->value > a)&&(temp->value < b))
125.             {
126.                 anc = temp->value;
127.                 printf("anc = %d\n", anc);
128.                 break;
129.             }
130.             else if ((temp->value < a)&&(temp->value > b))
131.             {
132.                 anc = temp->value;
133.                 temp = temp->r;
134.             }
135.             else
136.             {
137.                 printf("common ancestor = %d\n", anc);
138.                 break;
139.             }
140.         }
141.         path(anc, a, b);
142.     }
143.     else
144.         printf("enter correct node values & do not enter root value\n");
145.     }
146.
147.     /* to find whether given nodes are present in tree or not */

```

```

148.     int search(N *temp, int a, int b)
149.     {
150.         if ((temp->value == a || temp->value == b)&& (root->value != a&&root-
>value != b))
151.         {
152.             count++;
153.         }
154.         if (temp->l != NULL)
155.             search(temp->l, a, b);
156.         if (temp->r != NULL)
157.             search(temp->r, a, b);
158.         return count;
159.     }
160.
161.     /* to print the path ancestor to given nodes */
162.     void path(int anc, int c, int b)
163.     {
164.         N *temp = NULL;
165.         int i = 0, a[2];
166.         a[0] = c;
167.         a[1] = b;
168.
169.         for (;i < 2;i++)
170.         {
171.             if (anc == root->value) // If ancestor is root
172.             {
173.                 temp = root;
174.                 while (1)
175.                 {
176.                     printf("%d", temp->value);
177.                     if (a[i] < temp->value)
178.                         temp = temp->l;
179.                     else if (a[i] > temp->value)
180.                         temp = temp->r;
181.                     else
182.                     {
183.                         if (a[i] == temp->value)
184.                         {
185.                             break;
186.                         }
187.                     }
188.                     printf("->");
189.                 }

```

```

190.         printf("\n");
191.     }
192.     else if (anc < root->value)      //If ancestor is less than the root
193.     {
194.         temp = root;
195.         while (temp != NULL)
196.         {
197.             if (anc < temp->value)
198.                 temp = temp->l;
199.             else if (anc > temp->value)
200.                 temp = temp->r;
201.             else
202.             {
203.                 while (1)
204.                 {
205.                     if (a[i] < temp->value)
206.                     {
207.                         printf("%d->", temp->value);
208.                         temp = temp->l;
209.                     }
210.                     else if (a[i] > temp->value)
211.                     {
212.                         printf("%d->", temp->value);
213.                         temp = temp->r;
214.                     }
215.                     else
216.                     {
217.                         printf("%d\n", temp->value);
218.                         break;
219.                     }
220.                 }
221.             }
222.         }
223.     }
224.     else //If ancestor greater than the root value
225.     {
226.         temp = root;
227.         while (temp != NULL)
228.         {
229.             if (anc > temp->value)
230.                 temp = temp->r;
231.             else if (anc < temp->value)

```

```

232.         temp = temp->l;
233.     else
234.     {
235.         while (1)
236.         {
237.             if (a[i] < temp->value)
238.             {
239.                 printf("%d->", temp->value);
240.                 temp = temp->l;
241.             }
242.             else if (a[i] > temp->value)
243.             {
244.                 printf("%d->", temp->value);
245.                 temp = temp->r;
246.             }
247.             else
248.             {
249.                 printf("%d\n", temp->value);
250.                 break;
251.             }
252.         }
253.     }
254. }
255. }
256. }
257. }
```

### 3. C Examples dealing with the Nodes of a Tree

```

1. /*
2. * C Program to Count Number of Leaf Nodes in a Tree
3.
4.      50
5.      / \
6.     20   30
7.     / \
8.    70  80
9.   / \   \
10. 10   40   60
11. (50,20,30,70,80,10,40,60)
12. */
13. #include <stdio.h>
```

```
14. #include <stdlib.h>
15.
16. struct btnode {
17.     int value;
18.     struct btnode * l;
19.     struct btnode * r;
20. };
21.
22. typedef struct btnode bt;
23.
24. bt *root;
25. bt *new, *list;
26. int count = 0;
27.
28. bt * create_node();
29. void display(bt *);
30. void construct_tree();
31. void count_leaf(bt *);
32.
33. void main()
34. {
35.     construct_tree();
36.     display(root);
37.     count_leaf(root);
38.     printf("\n leaf nodes are : %d",count);
39. }
40.
41. /* To create a empty node */
42. bt * create_node()
43. {
44.     new = (bt *)malloc(sizeof(bt));
45.     new->l = NULL;
46.     new->r = NULL;
47. }
48.
49. /* To construct a tree */
50. void construct_tree()
51. {
52.     root = create_node();
53.     root->value = 50;
54.     root->l = create_node();
55.     root->l->value = 20;
56.     root->r = create_node();
```

```

57.     root->r->value = 30;
58.     root->l->l = create_node();
59.     root->l->l->value = 70;
60.     root->l->r = create_node();
61.     root->l->r->value = 80;
62.     root->l->r->r = create_node();
63.     root->l->r->r->value = 60;
64.     root->l->l->l = create_node();
65.     root->l->l->l->value = 10;
66.     root->l->l->r = create_node();
67.     root->l->l->r->value = 40;
68. }
69.
70./* To display the elements in a tree using inorder */
71.void display(bt * list)
72.{
73.    if (list == NULL)
74.    {
75.        return;
76.    }
77.    display(list->l);
78.    printf("->%d", list->value);
79.    display(list->r);
80.}
81.
82./* To count the number of Leaf nodes */
83.void count_leaf(bt * list)
84.{
85.    if (list == NULL)
86.    {
87.        return;
88.    }
89.    if (list->l == NULL && list->r == NULL)
90.    {
91.        count++;
92.    }
93.    count_leaf(list->l);
94.    count_leaf(list->r);
95.}

```

```

1. /*
2. * C Program to Find Nodes which are at Maximum Distance in Binary Tree

```

```

3.  /*
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct btnode
8. {
9.     int value;
10.    struct btnode *r,*l;
11. } *root = NULL, *temp = NULL;
12.
13. void create();
14. void insert();
15. void add(struct btnode *t);
16. void maxdistance(struct btnode *t);
17.
18. int count = 0, max = 0, v[100] = {0}, z = 0, max2, max1[100] = {0};
19.
20. void main()
21. {
22.     int ch, i;
23.
24.     printf("Program to find nodes at maximum distance");
25.     printf("\n OPERATIONS ----");
26.     printf("\n1] Insert ");
27.     printf("\n2] Display nodes ");
28.     printf("\n3] Exit ");
29.     while (1)
30.     {
31.         printf("\nEnter your choice : ");
32.         scanf("%d", &ch);
33.         switch (ch)
34.         {
35.             case 1:
36.                 insert();
37.                 break;
38.             case 2:
39.                 max = 0;
40.                 count = 0;
41.                 maxdistance(root);
42.                 for (i = 1; i < z; i++)
43.                 {
44.                     max2 = max1[0];
45.                     if (max2 < max1[i])

```

```

46.           max2 = max1[i];
47.       }
48.       printf("Maximum distance nodes \nNodes\t Distance ");
49.       for (i = 0; i < z; i++)
50.     {
51.         if (max2 == max1[i])
52.           printf("\n %d\t %d ",v[i],max2);
53.     }
54.     break;
55.   case 3:
56.     exit(0);
57.   default :
58.     printf("Wrong choice, Please enter correct choice  ");
59.     break;
60.   }
61. }
62. }

63.

64./* To create a new node with the data from the user */
65.void create()
66.{
67.  int data;
68.
69.  printf("Enter the data of node : ");
70.  scanf("%d", &data);
71.  temp = (struct btnode* ) malloc(1*(sizeof(struct btnode)));
72.  temp->value = data;
73.  temp->l = temp->r = NULL;
74. }
75.

76./* To check for root node and then create it */
77.void insert()
78.{
79.  create();
80.
81.  if (root == NULL)
82.    root = temp;
83.  else
84.    add(root);
85. }
86.

87./* Search for the appropriate position to insert the new node */
88.void add(struct btnode *t)

```

```

89. {
90.     if ((temp->value > t->value) && (t->r!=NULL))      /* value more than root node
   value insert at right */
91.         add(t->r);
92.     else if ((temp->value > t->value) && (t->r==NULL))
93.         t->r = temp;
94.     else if ((temp->value < t->value) && (t->l!=NULL))      /* value less than root
   node value insert at left */
95.         add(t->l);
96.     else if ((temp->value < t->value) && (t->l==NULL))
97.         t->l = temp;
98. }
99.

100.    /* Function to find the max distance nodes */
101.    void maxdistance(struct btnode *t)
102.    {
103.        if (t->l!=NULL)
104.        {
105.            count++;           /* to count the number of nodes in between root
   and Leaf */
106.            maxdistance(t->l);
107.        }
108.        if (max < count)
109.            max = count;
110.        if (max == count)
111.        {
112.            max1[z] = max;
113.            v[z] = t->value;
114.            z++;
115.        }
116.        if (t->r != NULL)
117.        {
118.            count++;
119.            maxdistance(t->r);
120.        }
121.        count--;
122.    }

```

```

1. *
2. * C Program to Find the Number of Nodes in a Binary Tree
3. */
4. #include <stdio.h>

```

```
5. #include <stdlib.h>
6.
7. /*
8.  * Structure of node
9. */
10. struct btnode
11. {
12.     int value;
13.     struct btnode *l;
14.     struct btnode *r;
15. };
16.
17. void createbinary();
18. void preorder(node *);
19. int count(node*);
20. node* add(int);
21.
22. typedef struct btnode node;
23. node *ptr, *root = NULL;
24.
25. int main()
26. {
27.     int c;
28.
29.     createbinary();
30.     preorder(root);
31.     c = count(root);
32.     printf("\nNumber of nodes in binary tree are:%d\n", c);
33. }
34. /*
35. * constructing the following binary tree
36. *      50
37. *      / \
38. *     20 30
39. *    / \
40. *   70 80
41. *  / \   \
42. * 10 40     60
43. */
44. void createbinary()
45. {
46.     root = add(50);
47.     root->l = add(20);
```

```
48.     root->r = add(30);
49.     root->l->l = add(70);
50.     root->l->r = add(80);
51.     root->l->l->l = add(10);
52.     root->l->l->r = add(40);
53.     root->l->r->r = add(60);
54. }
55.
56. /*
57. * Add the node to binary tree
58. */
59. node* add(int val)
60. {
61.     ptr = (node*)malloc(sizeof(node));
62.     if (ptr == NULL)
63.     {
64.         printf("Memory was not allocated");
65.         return;
66.     }
67.     ptr->value = val;
68.     ptr->l = NULL;
69.     ptr->r = NULL;
70.     return ptr;
71. }
72.
73. /*
74. * counting the number of nodes in a tree
75. */
76. int count(node *n)
77. {
78.     int c = 1;
79.
80.     if (n == NULL)
81.         return 0;
82.     else
83.     {
84.         c += count(n->l);
85.         c += count(n->r);
86.         return c;
87.     }
88. }
89.
90. /*
```

```

91. * Displaying the nodes of tree in preorder
92. */
93. void preorder(node *t)
94. {
95.     if (t != NULL)
96.     {
97.         printf("%d->", t->value);
98.         preorder(t->l);
99.         preorder(t->r);
100.    }
101. }
```

```

1. /*
2. * C Program to Print Border of given Tree in Anticlockwise Direction
3. *          50
4. *        /   \
5. *       20   30
6. *      /   \
7. *     70   80
8. *    / \   \
9. *   10  40   60
10. */
11. #include <stdio.h>
12. #include <stdlib.h>
13.
14. struct btnode {
15.     int value;
16.     struct btnode *l;
17.     struct btnode *r;
18. };
19. struct btnode *root;
20. typedef struct btnode bt;
21. bt *new, *ptr, *ptr1, *ptr2;
22.
23. void print();
24. void print_leaf_nodes(bt* );
25. void print_right_recursive(bt* );
26. bt* create();
27. void construct_binary_tree();
28.
29. void main()
30. {
```

```
31.     construct_binary_tree();
32.     printf("\nprinting the border elements anticlockwise direction:\n");
33.     print();
34.     printf("\n");
35. }
36.
37. bt* create()
38. {
39.     new = (bt *)malloc(sizeof(bt));
40.     new->l = NULL;
41.     new->r = NULL;
42.     return new;
43. }
44.
45. void construct_binary_tree()
46. {
47.     root = create();
48.     root->value = 50;
49.
50.     ptr = create();
51.     root->l = ptr;
52.     ptr->value = 20;
53.     ptr1 = create();
54.     ptr->l = ptr1;
55.     ptr1->value = 70;
56.     ptr2 = create();
57.     ptr1->l = ptr2;
58.     ptr2->value = 10;
59.     ptr2 = create();
60.     ptr1->r = ptr2;
61.     ptr2->value = 40;
62.     ptr1 = create();
63.     ptr->r = ptr1;
64.     ptr1->value = 80;
65.     ptr2 = create();
66.     ptr1->r = ptr2;
67.     ptr2->value = 60;
68.     ptr = create();
69.     root->r = ptr;
70.     ptr->value = 30;
71. }
72.
73. void print()
```

```
74. {
75.     ptr = root;
76.     while (ptr->l != NULL)
77.     {
78.         printf("->%d", ptr->value);
79.         ptr = ptr->l;
80.     }
81.     ptr = root;
82.     print_leaf_nodes(ptr);
83.     ptr = root;
84.     print_right_recursive(ptr);
85. }
86.
87. void print_leaf_nodes(bt* ptr)
88. {
89.     if (ptr != NULL)
90.     {
91.         if (ptr->l == NULL && ptr->r == NULL)
92.         {
93.             printf("->%d", ptr->value);
94.         }
95.         else
96.         {
97.             print_leaf_nodes(ptr->l);
98.             print_leaf_nodes(ptr->r);
99.         }
100.        }
101.        else
102.            return;
103.    }
104.
105.    void print_right_recursive(bt* ptr)
106.    {
107.        int val;
108.        ptr = ptr->r;
109.        if (ptr->r != NULL)
110.        {
111.            print_right_recursive(ptr->r);
112.            printf("->%d", ptr->value);
113.        }
114.        else
115.        {
116.            return;
```

```
117.         }
118.     }
```

```
1. /*
2.  * C Program to Print Height and Depth of given Binary Tree
3.  *
4.  *           50
5.  *           / \
6.  *          20   30
7.  *          / \
8.  *         70   80
9.  *         / \   \
10. *        10  40   60
10. */
11.#include <stdio.h>
12.#include <stdlib.h>
13.
14. struct btnode {
15.     int value;
16.     struct btnode *l;
17.     struct btnode *r;
18. };
19. struct btnode *root;
20. typedef struct btnode bt;
21. bt *new,*ptr,*ptr1,*ptr2;
22.
23. bt* create()
24. {
25.     new = (bt *)malloc(sizeof(bt));
26.     new->l = NULL;
27.     new->r = NULL;
28.     return new;
29. }
30.
31. void construct_binary_tree()
32. {
33.     root = create();
34.     root->value = 50;
35.
36.     ptr = create();
37.     root->l = ptr;
38.     ptr->value = 20;
39.
```

```
40.     ptr1 = create();
41.     ptr->l = ptr1;
42.     ptr1->value = 70;
43.
44.     ptr2 = create();
45.     ptr1->l = ptr2;
46.     ptr2->value = 10;
47.
48.     ptr2 = create();
49.     ptr1->r = ptr2;
50.     ptr2->value = 40;
51.
52.     ptr1 = create();
53.     ptr->r = ptr1;
54.     ptr1->value = 80;
55.
56.     ptr2 = create();
57.     ptr1->r = ptr2;
58.     ptr2->value = 60;
59.
60.     ptr = create();
61.     root->r = ptr;
62.     ptr->value = 30;
63. }
64.
65. void main()
66. {
67.     int depth1 = 0, depth2 = 0;
68.
69.     construct_binary_tree();
70.     ptr = root;
71.     while (ptr->l != NULL || ptr->r != NULL)
72.     {
73.         depth1++;
74.         if (ptr->l == NULL)
75.             ptr = ptr->r;
76.         else
77.             ptr = ptr->l;
78.     }
79.     ptr = root;
80.     while (ptr->l != NULL || ptr->r != NULL)
81.     {
82.         depth2++;
```

```

83.     if (ptr->r == NULL)
84.         ptr = ptr->l;
85.     else
86.         ptr = ptr->r;
87. }
88. /*
89. *DEPTH IS CONSIDERED FROM LEVEL-0 ALSO HEIGHT IS CONSIDERED AS NUMBER OF EDGES
90. */
91. if (depth1 > depth2)
92.     printf("height of the tree is %d\ndepth of the tree is %d",depth1,depth1);
93. else
94.     printf("height of the tree is %d\ndepth of the tree is %d",depth2,depth2);
95. }
```

```

1. /*
2. * C Program to Count Number of Non Leaf Nodes of a given Tree
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct btnode
8. {
9.     int value;
10.    struct btnode *r,*l;
11. } *root = NULL, *temp = NULL;
12.
13. void create();
14. void insert();
15. void add(struct btnode *t);
16. void inorder(struct btnode *t);
17.
18. int count = 0;
19.
20. void main()
21. {
22.     int ch;
23.
24.     printf("\nOPERATIONS ---");
25.     printf("\n1] Insert ");
26.     printf("\n2] Display");
27.     printf("\n3] Exit ");
28. }
```

```

29.     while (1)
30.     {
31.         printf("\nEnter your choice : ");
32.         scanf("%d", &ch);
33.         switch (ch)
34.         {
35.             case 1:
36.                 insert();
37.                 break;
38.             case 2:
39.                 inorder(root);
40.                 printf("\nNumber of non leaf nodes: %d", count);
41.                 break;
42.             case 3:
43.                 exit(0);
44.             default :
45.                 printf("Wrong choice, Please enter correct choice   ");
46.                 break;
47.         }
48.     }
49. }
50.
51./* To create a new node with the data from the user */
52.void create()
53.{
54.    int data;
55.
56.    printf("Enter the data of node : ");
57.    scanf("%d", &data);
58.    temp = (struct btnode* ) malloc(1*(sizeof(struct btnode)));
59.    temp->value = data;
60.    temp->l = temp->r = NULL;
61.}
62.
63./* To check for root node and then create it */
64.void insert()
65.{
66.    create();
67.
68.    if (root == NULL)
69.        root = temp;
70.    else
71.        add(root);

```

```

72. }
73.
74./* Search for the appropriate position to insert the new node */
75.void add(struct btnode *t)
76.{
77.    if ((temp->value > t->value) && (t->r != NULL))           /* value more than root
   node value insert at right */
78.        add(t->r);
79.    else if ((temp->value > t->value) && (t->r == NULL))
80.        t->r = temp;
81.    else if ((temp->value < t->value) && (t->l != NULL))           /* value less than
   root node value insert at left */
82.        add(t->l);
83.    else if ((temp->value < t->value) && (t->l == NULL))
84.        t->l = temp;
85.}
86.
87./* To display and count the sum of non leaf nodes */
88.void inorder(struct btnode *t)
89.{
90.    if (t->l != NULL)
91.        inorder(t->l);
92.    if ((t->l != NULL) || (t->r != NULL))
93.    {
94.        count++;                                /* To count number of non leaf nodes */
95.        printf("%d ->",t->value);
96.    }
97.    if (t->r != NULL)
98.        inorder(t->r);
99.}
```

#### 4. C Examples on Special Properties of Binary Trees

```

1. /*
2.  * C Program to Check whether a Tree and its Mirror Image are same
3.  *
4.  *          50
5.  *          / \           50
6.  *          20   30         / \
7.  *      Sample Tree<----- / \           30   20
8.  *      Mirror image           / \   ----->
9.  *
10. *          70     80
11. *          / \     \           80     70
12. *          / \     \           /   / \
```

```
9.      *          10  40    60          60  40    10
10.     *          (50,20,30,70,80,10,40,60)
11.    */
12. #include <stdio.h>
13. #include <stdlib.h>
14.
15. struct btnode {
16.     int value;
17.     struct btnode * l;
18.     struct btnode * r;
19. };
20.
21. typedef struct btnode bt;
22.
23. bt *root,*temp;
24. bt *new;
25. int c;
26.
27. bt * create_node();
28. void display(bt *);
29. bt * construct_tree();
30. void mirror_image(bt *);
31. int compare(bt *,bt *);
32.
33. void main()
34. {
35.     root = construct_tree();
36.     display(root);
37.     temp = construct_tree();
38.     mirror_image(temp);
39.     printf("\n mirror image:\n");
40.     display(temp);
41.     c = compare(root,temp);
42.     if (c)
43.     {
44.         printf("\nsame");
45.     }
46.     else
47.     {
48.         printf("\nnot same");
49.     }
50. }
```

```
52./* creates new node */
53.bt * create_node()
54.{  
55.    new=(bt *)malloc(sizeof(bt));  
56.    new->l = NULL;  
57.    new->r = NULL;  
58.}  
59.  
60./* constructs the tree */
61.bt * construct_tree()  
62.{  
63.    bt *list;  
64.  
65.    list = create_node();  
66.    list->value = 50;  
67.    list->l = create_node();  
68.    list->l->value = 20;  
69.    list->r = create_node();  
70.    list->r->value = 30;  
71.    list->l->l = create_node();  
72.    list->l->l->value = 70;  
73.    list->l->r = create_node();  
74.    list->l->r->value = 80;  
75.    list->l->r->r = create_node();  
76.    list->l->r->r->value = 60;  
77.    list->l->l->l = create_node();  
78.    list->l->l->l->value = 10;  
79.    list->l->l->r = create_node();  
80.    list->l->l->r->value = 40;  
81.  
82.    return list;  
83.}  
84.  
85./* displays the tree using inorder */
86.void display(bt * list)
87.{  
88.    if (list == NULL)
89.    {
90.        return;
91.    }
92.    display(list->l);
93.    printf("->%d", list->value);
94.    display(list->r);
```

```

95. }
96.
97./* creates mirror image of a tree */
98.void mirror_image(bt * list)
99.{
100.    bt * temp1;
101.
102.    if (list == NULL)
103.    {
104.        return;
105.    }
106.    temp1 = list->l;
107.    list->l = list->r;
108.    list->r = temp1;
109.    mirror_image(list->l);
110.    mirror_image(list->r);
111.}
112.
113./* compares tree and its mirror image */
114.int compare(bt *list, bt * list1)
115.{
116.    int d;
117.    if (list == NULL && list1 == NULL)
118.    {
119.        return 1;
120.    }
121.    else if (list != NULL && list1 != NULL)
122.    {
123.        return(list->value == list1->value &&
124.            compare(list->l, list1->l) &&
125.            compare(list->r, list1->r));
126.    }
127.    else
128.    {
129.        return 0;
130.    }
131.}

```

```

1. /*
2.  * C Program to Create a Mirror Copy of a Tree and Display using
3.  * BFS Traversal
4. *

```

```

5.   *                  /\
6.   *                  20 60
7.   *                  /\  \
8.   *                  10 30  80
9.   *
10.  *                  \
11. /*                  90
12. #include <stdio.h>
13. #include <stdlib.h>
14.
15. struct btnode
16. {
17.     int value;
18.     struct btnode *left, *right;
19. };
20. typedef struct btnode node;
21.
22. /* function prototypes */
23. void insert(node *, node *);
24. void mirror(node *);
25.
26. /* global variables */
27. node *root = NULL;
28. int val, front = 0, rear = -1, i;
29. int queue[20];
30.
31. void main()
32. {
33.     node *new = NULL ;
34.     int num = 1;
35.     printf("Enter the elements of the tree(enter 0 to exit)\n");
36.     while (1)
37.     {
38.         scanf("%d", &num);
39.         if (num == 0)
40.             break;
41.         new = malloc(sizeof(node));
42.         new->left = new->right = NULL;
43.         new->value = num;
44.         if (root == NULL)
45.             root = new;
46.         else
47.         {

```

```

48.         insert(new, root);
49.     }
50. }
51. printf("mirror image of tree is\n");
52. queue[++rear] = root->value;
53. mirror(root);
54. for (i = 0;i <= rear;i++)
55.     printf("%d -> ", queue[i]);
56. printf("%d\n", root->right->right->right->value);
57. }
58.
59./* inserting nodes into the tree */
60 void insert(node * new , node *root)
61 {
62.     if (new->value > root->value)
63.     {
64.         if (root->right == NULL)
65.             root->right = new;
66.         else
67.             insert (new, root->right);
68.     }
69.     if (new->value < root->value)
70.     {
71.         if (root->left == NULL)
72.             root->left = new;
73.         else
74.             insert (new, root->left);
75.     }
76. }
77.
78./* mirror image of nodes */
79 void mirror(node *root)
80 {
81.     val = root->value;
82.     if ((front <= rear) && (root->value == queue[front]))
83.     {
84.         if (root->right != NULL || root->right == NULL)
85.             queue[++rear] = root->right->value;
86.         if (root->left != NULL)
87.             queue[++rear] = root->left->value;
88.         front++;
89.     }
90.     if (root->right != NULL)

```

```

91.     {
92.         mirror(root->right);
93.     }
94.     if (root->left != NULL)
95.     {
96.         mirror(root->left);
97.     }
98. }
```

```

1. /*
2. * C Program to Find the Nearest Common Ancestor in the Given set
3. * of Nodes
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. /*
9. * Structure of binary tree node
10.*/
11. struct btnode
12. {
13.     int value;
14.     struct btnode *l;
15.     struct btnode *r;
16. };
17.
18. void createbinary();
19. node* add(int val);
20. int height(node *);
21. int nearest_common_ancestor(node*,  int,  int);
22. typedef struct btnode node;
23. node *root = NULL, *ptr;
24.
25. int main()
26. {
27.     int c, n1, n2;
28.
29.     createbinary();
30.     printf("\nEnter nodes having common ancestor");
31.     scanf("%d %d", &n1, &n2);
32.     c = nearestcommonancestor(root, n1, n2);
33.     if (c == -1)
```

```

34.     {
35.         printf("No common ancestor");
36.     }
37. else
38. {
39.     printf("The common ancestor is %d", c);
40. }
41. }
42. /*
43. * constructing the following binary tree
44. *      50
45. *      / \
46. *     20 30
47. *     / \
48. *    70 80
49. *   / \     \
50. *10 40      60
51. */
52. void createbinary()
53. {
54.     root = add(50);
55.     root->l = add(20);
56.     root->r = add(30);
57.     root->l->l = add(70);
58.     root->l->r = add(80);
59.     root->l->l->l = add(10);
60.     root->l->l->r = add(40);
61.     root->l->r->r = add(60);
62. }
63.
64. /*
65. * Add node to binary tree
66. */
67. node* add(int val)
68. {
69.     ptr = (node*)malloc(sizeof(node));
70.     if (ptr == NULL)
71.     {
72.         printf("Memory was not allocated");
73.         return;
74.     }
75.     ptr->value = val;
76.     ptr->l = NULL;

```



```

119.         prev = temp;
120.         for (j = 1, temp = root;j < h;j++)
121.         {
122.             if (temp != NULL)
123.             {
124.                 if (temp->r->value == n2 || temp->r->value == n1 ||
125.                     temp->l->value == n1 || temp->l->value == n2)
126.                 {
127.                     /* If the parent of n1 and parent of n2 are same
128.                      then the value of parent is returned
129.                     */
130.                     if (prev->value == temp->value)
131.                         return prev->value;
132.                     /*
133.                      * otherwise from parents of two nodes is traversed
134.                      upward if any node matches with other's node parent then that is
135.                      * considered as common ancestor
136.                      */
137.                     else
138.                         for (k = 0, prev = temp;k < h;k++)
139.                         {
140.                             if (temp->l->value == prev->l->value)
141.                                 return temp->value;
142.                             else
143.                                 temp = temp->l;
144.                         }
145.                     temp = temp->l;
146.                 }
147.             }
148.             temp = temp->l;
149.         }
150.     }

```

```

1. /*
2.  * C Program to Find the Common Ancestor and Print the Path
3.  *
4.  *          10
5.  *          /   \
6.  *          7    15

```

```

7.      *          / \   / \
8.      *          6   8 12  18
9.      *          /     \
10.     *          5       9
11.     *          (Given Binary tree)
12.    */
13. #include <stdio.h>
14. #include <stdlib.h>
15.
16. struct btnode
17. {
18.     int value;
19.     struct btnode *l;
20.     struct btnode *r;
21. };
22.
23. typedef struct btnode N;
24.
25. N* new(int);
26. int count;
27.
28. void create();
29. void preorder(N *t);
30. void ancestor(N *t);
31. int search(N *t, int, int);
32. void path(int, int, int);
33.
34. N *root = NULL;
35.
36. void main()
37. {
38.     int choice;
39.
40.     create();
41.     while (1)
42.     {
43.         printf("Enter the choice\n");
44.         printf("1-Display : 2-path : 3-Exit\n");
45.         scanf("%d", &choice);
46.         switch (choice)
47.         {
48.             case 1:
49.                 printf("preorder display of tree elements\n");

```

```
50.         preorder(root);
51.         printf("\n");
52.         break;
53.     case 2:
54.         ancestor(root);
55.         break;
56.     case 3:
57.         exit(0);
58.     default:
59.         printf("Enter the right choice\n");
60.     }
61. }
62. }
63.
64./* creating temporary node */
65.N* new(int data)
66.{
67.    N* temp = (N*)malloc(sizeof(N));
68.    temp->value = data;
69.    temp->l = NULL;
70.    temp->r = NULL;
71.
72.    return(temp);
73.}
74.
75./* Creating the binary search tree */
76.void create()
77.{
78.    root = new(10);
79.    root->l = new(7);
80.    root->r = new(15);
81.    root->l->l = new(6);
82.    root->l->r = new(8);
83.    root->r->l = new(12);
84.    root->r->r = new(18);
85.    root->r->r->r = new(20);
86.    root->l->l->l = new(5);
87.    root->l->r->r = new(9);
88.}
89.
90./* To display the preorder traversal of the tree */
91.void preorder(N *temp)
92.{
```

```

93.         printf("%d->", temp->value);
94.         if (temp->l != NULL)
95.             preorder(temp->l);
96.         if (temp->r != NULL)
97.             preorder(temp->r);
98.     }
99.
100.    /* to find common ancestor for given nodes */
101.    void ancestor(N *temp)
102.    {
103.        int a, b, anc = 0;
104.        count = 0;
105.
106.        printf("enter two node values to find common ancestor\n");
107.        scanf("%d", &a);
108.        scanf("%d", &b);
109.        count = search(root, a, b);
110.        if (count == 2)
111.        {
112.            while (temp->value != a && temp->value != b)
113.            {
114.                if ((temp->value > a)&&(temp->value > b))
115.                {
116.                    anc = temp->value;
117.                    temp = temp->l;
118.                }
119.                else if ((temp->value < a)&&(temp->value < b))
120.                {
121.                    anc = temp->value;
122.                    temp = temp->r;
123.                }
124.                else if ((temp->value > a)&&(temp->value < b))
125.                {
126.                    anc = temp->value;
127.                    printf("anc = %d\n", anc);
128.                    break;
129.                }
130.                else if ((temp->value < a)&&(temp->value > b))
131.                {
132.                    anc = temp->value;
133.                    temp = temp->r;
134.                }
135.            }

```

```

136.         {
137.             printf("common ancestor = %d\n", anc);
138.             break;
139.         }
140.     }
141.     path(anc, a, b);
142. }
143. else
144.     printf("enter correct node values & do not enter root value\n");
145. }
146.
147. /* to find whether given nodes are present in tree or not */
148. int search(N *temp, int a, int b)
149. {
150.     if ((temp->value == a || temp->value == b)&& (root->value != a&&root-
>value != b))
151.     {
152.         count++;
153.     }
154.     if (temp->l != NULL)
155.         search(temp->l, a, b);
156.     if (temp->r != NULL)
157.         search(temp->r, a, b);
158.     return count;
159. }
160.
161. /* to print the path ancestor to given nodes */
162. void path(int anc, int c, int b)
163. {
164.     N *temp = NULL;
165.     int i = 0, a[2];
166.     a[0] = c;
167.     a[1] = b;
168.
169.     for (;i < 2;i++)
170.     {
171.         if (anc == root->value) // If ancestor is root
172.         {
173.             temp = root;
174.             while (1)
175.             {
176.                 printf("%d", temp->value);
177.                 if (a[i] < temp->value)

```

```

178.             temp = temp->l;
179.         else if (a[i] > temp->value)
180.             temp = temp->r;
181.         else
182.         {
183.             if (a[i] == temp->value)
184.             {
185.                 break;
186.             }
187.         }
188.         printf("->");
189.     }
190.     printf("\n");
191. }
192. else if (anc < root->value)      //If ancestor is less than the root
193. value
194. {
195.     temp = root;
196.     while (temp != NULL)
197.     {
198.         if (anc < temp->value)
199.             temp = temp->l;
200.         else if (anc > temp->value)
201.             temp = temp->r;
202.         else
203.         {
204.             while (1)
205.             {
206.                 if (a[i] < temp->value)
207.                 {
208.                     printf("%d->", temp->value);
209.                     temp = temp->l;
210.                 }
211.                 else if (a[i] > temp->value)
212.                 {
213.                     printf("%d->", temp->value);
214.                     temp = temp->r;
215.                 }
216.                 else
217.                 {
218.                     printf("%d\n", temp->value);
219.                     break;
220.                 }
221.             }
222.         }
223.     }
224. }

```

```

220.                     }
221.                 }
222.             }
223.         }
224.     //If ancestor greater than the root value
225.     {
226.         temp = root;
227.         while (temp != NULL)
228.         {
229.             if (anc > temp->value)
230.                 temp = temp->r;
231.             else if (anc < temp->value)
232.                 temp = temp->l;
233.             else
234.             {
235.                 while (1)
236.                 {
237.                     if (a[i] < temp->value)
238.                     {
239.                         printf("%d->", temp->value);
240.                         temp = temp->l;
241.                     }
242.                     else if (a[i] > temp->value)
243.                     {
244.                         printf("%d->", temp->value);
245.                         temp = temp->r;
246.                     }
247.                     else
248.                     {
249.                         printf("%d\n", temp->value);
250.                         break;
251.                     }
252.                 }
253.             }
254.         }
255.     }
256. }
257. }
```

```

1. /*
2.  * C Program to Print all the Elements of Nth Level in Single Line
3. */
```

```
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. /*
8.  * structure of node
9. */
10. struct btnode
11. {
12.     int value;
13.     struct btnode *l;
14.     struct btnode *r;
15. };
16.
17. void createbinary();
18. node* add(int val);
19. int height(node *);
20. void printlevel(node *, int, int);
21. void print();
22.
23. typedef struct btnode node;
24. node *root = NULL, *ptr;
25.
26. int main()
27. {
28.     int c;
29.
30.     createbinary();
31.     print();
32. }
33. /*
34. * constructing the following binary tree
35. *      50
36. *      / \
37. *     20 30
38. *    / \
39. *   70 80
40. *  / \     \
41. * 10 40     60
42. */
43. void createbinary()
44. {
45.     root = add(50);
46.     root->l = add(20);
```

```
47.     root->r = add(30);
48.     root->l->l = add(70);
49.     root->l->r = add(80);
50.     root->l->l->l = add(10);
51.     root->l->l->r = add(40);
52.     root->l->r->r = add(60);
53. }
54.
55./*
56. * Adding node to binary tree
57. */
58.node* add(int val)
59.{
60.    ptr = (node*)malloc(sizeof(node));
61.    if (ptr == NULL)
62.    {
63.        printf("Memory was not allocated");
64.        return;
65.    }
66.    ptr->value = val;
67.    ptr->l = NULL;
68.    ptr->r = NULL;
69.    return ptr;
70.}
71.
72./*
73. * Prints all the nodes of all levels of the binary tree
74. */
75.void print()
76.{
77.    int h, i;
78.
79.    h = height(root);
80.    for (i = 0;i < h;i++)
81.    {
82.        printf("\nLEVEL %d : ", i);
83.        printlevel(root, i, 0);
84.        printf("\n");
85.    }
86.}
87./*
88. *Prints the nodes of a particular level
89. */
```

```

90. void printlevel(node *n, int desired, int current)
91. {
92.     if (n)
93.     {
94.         if (desired == current)
95.             printf("%d\t", n->value);
96.         else
97.         {
98.             printlevel(n->l, desired, current + 1);
99.             printlevel(n->r, desired, current + 1);
100.            }
101.        }
102.    }
103.
104. /*
105. * Height of the binary tree
106. */
107. int height(node *n)
108. {
109.     int lheight, rheight;
110.     if (n != NULL)
111.     {
112.         lheight = height(n->l);
113.         rheight = height(n->r);
114.         if (lheight > rheight)
115.             return(lheight+1);
116.         else
117.             return(rheight+1);
118.     }
119. }
```

```

1. /*
2. * C Program to Convert Binary Tree to Binary Search Tree
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. /*
8. * Structure of the binary tree
9. */
10. struct btnode
11. {
```

```

12.     int value;
13.     struct btnode *l;
14.     struct btnode *r;
15. };
16.
17. void createbinary();
18. void inorder(node *);
19. int count(node* );
20. node* add(int );
21. void sort();
22. void binary_to_bst(node *);
23. int compare(const void *,const void *);
24. void inorder_to_array(node*,int[],int *);
25. void array_to_bst(int *arr,node *,int *);
26. void display_bst(node *);
27. void print();
28. void print_level(node*,int,int);
29. int height(node* );
30.
31. typedef struct btnode node;
32. node *root = NULL,*ptr;
33.
34. int data[10];
35. int i = 0;
36.
37. int main()
38. {
39.     createbinary();
40.     binary_to_bst(root);
41.     printf("\n~~~~~\n");
42.     printf("\nThe inorder of binary search tree\n");
43.     inorder(root);
44.     printf("\n~~~~~\n");
45.     printf("\n=====\n");
46.     printf("\nThe nodes of a binary search tree (LEVEL WISE)\n");
47.     printf("\n=====\n");
48.     print();
49. }
50. /*
51. * constructing the following binary tree
52. *      50
53. *      / \
54. *     20 30

```

```

55. *   / \
56. * 70 80
57. * / \     \
58. *10 40      60
59. */
60.void createbinary()
61.{  

62.    root = add(50);
63.    root->l = add(20);
64.    root->r = add(30);
65.    root->l->l = add(70);
66.    root->l->r = add(80);
67.    root->l->l->l = add(10);
68.    root->l->l->r = add(40);
69.    root->l->r->r = add(60);
70. }
71.
72./*
73. * Adds a node to the tree
74. */
75.node* add(int val)
76.{  

77.    ptr = (node*)malloc(sizeof(node));
78.    if (ptr == NULL)
79.    {
80.        printf("Memory was not allocated");
81.        return 0;
82.    }
83.    ptr->value = val;
84.    ptr->l = NULL;
85.    ptr->r = NULL;
86.    return ptr;
87. }
88.
89./*
90. * Store the inorder of binary tree
91. */
92.void inorder_to_array(node *n,int data[],int *ptr)
93.{  

94.    if (n != NULL)
95.    {
96.        inorder_to_array(n->l,data,ptr);
97.        data[*ptr] = n->value;

```

```
98.         (*ptr)++;
99.         inorder_to_array(n->r,data,ptr);
100.     }
101. }
102.
103. /*
104. * counting the number of nodes in a tree
105. */
106. int count(node *n)
107. {
108.     int c = 1;
109.
110.     if (n == NULL)
111.         return 0;
112.     else
113.     {
114.         c += count(n->l);
115.         c += count(n->r);
116.         return c;
117.     }
118. }
119. /*
120. *Display inorder of the BST
121. */
122. void inorder(node *root)
123. {
124.     if (root != NULL)
125.     {
126.         inorder(root->l);
127.         printf("%d->", root->value);
128.         inorder(root->r);
129.     }
130.
131. }
132. /*
133. * print the nodes of the BST for all levels until height of the tree is
reached
134. */
135.
136. void print()
137. {
138.     int h, i;
```

```
140.         h = height(root);
141.         for(i = 0;i < h;i++)
142.         {
143.             printf("\nLEVEL %d : ", i);
144.             print_level(root, i, 0);
145.             printf("\n");
146.         }
147.     }
148.
149. /*
150. * print the nodes of the BST for a particular level
151. */
152. void print_level(node *n, int desired, int current)
153. {
154.     if (n)
155.     {
156.         if (desired == current)
157.         {
158.             printf(" %5d", n->value);
159.         }
160.         else
161.         {
162.             print_level(n->l, desired, current + 1);
163.             print_level(n->r, desired, current + 1);
164.         }
165.     }
166. }
167.
168. /*
169. * Height of binary tree
170. */
171. int height(node *n)
172. {
173.     int lheight, rheight;
174.
175.     if (n != NULL)
176.     {
177.         lheight = height(n->l);
178.         rheight = height(n->r);
179.         if (lheight > rheight)
180.             return(lheight + 1);
181.         else
182.             return(rheight + 1);
```

```

183.         }
184.     else
185.     {
186.         return 0;
187.     }
188. }
189. int compare(const void *a, const void *b)
190. {
191.     return *(int*)a-*(int*)b;
192. }
193.
194. /*
195. * copies the elements of array to binary tree
196. */
197. void array_to_bst(int *arr, node *root, int *indexptr)
198. {
199.     if (root != NULL)
200.     {
201.         array_to_bst(arr,root->l, indexptr);
202.         root->value = arr[i++];
203.         array_to_bst(arr,root->r, indexptr);
204.     }
205. }
206.
207. /*
208. * Converting binary tree to binary search tree
209. * storeinorder() function stores the inorder traversal of binary tree
210. * qsort() sorts the inorder of binary tree
211. * arraytobst() copies the elements of array to binary tree
212. * Then binary tree converted to binary search tree
213. */
214.
215. void binary_to_bst(node *root)
216. {
217.     int n, i;
218.
219.     if (root == NULL)
220.         return;
221.     n = count(root);
222.     i = 0;
223.     inorder_to_array(root, data, &i);
224.     qsort(&data, n, sizeof(data[0]), compare);
225.     i = 0;

```

```

226.         array_to_bst(data, root, &i);
227.     }

```

## 6. C Examples on Inorder Traversal of a Binary Tree

```

1. /*
2. * C Program to Search for a Particular Value in a Binary Tree
3. *           50
4. *           /\
5. *           20  30
6. *           /\
7. *           70  80
8. *           / \   \
9. *           10 40  60
10. */
11.
12. #include <stdio.h>
13. #include <malloc.h>
14. /* Structure to create the binary tree */
15.
16. struct btnode
17. {
18.     int value;
19.     struct btnode *l;
20.     struct btnode *r;
21. };
22.
23. struct btnode *root = NULL;
24. int flag;
25.
26. /* Function Prototypes */
27. void in_order_traversal(struct btnode *);
28. void in_order_search(struct btnode *,int);
29. struct btnode *newnode(int);
30.
31. void main()
32. {
33.     /* Inserting elements in the binary tree */
34.     int search_val;
35.     root = newnode(50);
36.     root->l = newnode(20);
37.     root->r = newnode(30);

```

```

38.     root->l->l = newnode(70);
39.     root->l->r = newnode(80);
40.     root->l->l->l = newnode(10);
41.     root->l->l->r = newnode(40);
42.     root->l->r->r = newnode(60);
43.
44.     printf("The elements of Binary tree are:");
45.     in_order_traversal(root);
46.     printf("Enter the value to be searched:");
47.     scanf("%d", &search_val);
48.     in_order_search(root, search_val);
49.     if (flag == 0) // flag to check if the element is present in the tree or
not
50.     {
51.         printf("Element not present in the binary tree\n");
52.     }
53. }
54.
55./* Code to dynamically create new nodes */
56. struct btnode* newnode(int value)
57. {
58.     struct btnode *temp = (struct btnode *)malloc(sizeof(struct btnode));
59.     temp->value = value;
60.     temp->l = NULL;
61.     temp->r = NULL;
62.     return temp;
63. }
64.
65./* Code to display the elements of the binary tree */
66.
67. void in_order_traversal(struct btnode *p)
68. {
69.     if (!p)
70.     {
71.         return;
72.     }
73.     in_order_traversal(p->l);
74.     printf("%d->", p->value);
75.     in_order_traversal(p->r);
76. }
77.
78./* Code to search for a particular element in the tree */
79. void in_order_search(struct btnode *p, int val)

```

```
80. {
81.     if (!p)
82.     {
83.         return;
84.     }
85.     in_order_search(p->l, val);
86.     if(p->value == val)
87.     {
88.         printf("\nElement present in the binary tree.\n");
89.         flag = 1;
90.     }
91.     in_order_search(p->r, val);
92. }

93. /*
94. * C Program to Find the Sum of All Nodes in a Binary Tree
95. *          50
96. *        / \
97. *      20  30
98. *    / \
99. *   70  80
100. *  /   \
101. * 10  40  60
102. */
103. #include <stdio.h>
104. #include <malloc.h>
105.
106. /* Structure to create the binary tree */
107. struct btnode
108. {
109.     int value;
110.     struct btnode *l;
111.     struct btnode *r;
112. };
113. struct btnode *root = NULL;
114. int sum;
115.
116. /* Function Prototypes */
117.
118. void in_order_traversal(struct btnode *);
119. void in_order_sum(struct btnode *);
120. struct btnode *newnode(int);
121.
122. void main()
```

```

123. {
124.
125.     /* Inserting elements in the binary tree */
126.     root = newnode(50);
127.     root->l = newnode(20);
128.     root->r = newnode(30);
129.     root->l->l = newnode(70);
130.     root->l->r = newnode(80);
131.     root->l->l->l = newnode(10);
132.     root->l->l->r = newnode(40);
133.     root->l->r->r = newnode(60);
134.     printf("The elements of Binary tree are:");
135.     in_order_traversal(root);
136.     in_order_sum(root);
137.     printf("\n\nThe sum of all the elements are:%d", sum);
138. }
139.
140. /* Code to dynamically create new nodes */
141. struct btnode* newnode(int value)
142. {
143.     struct btnode *temp = (struct btnode *)malloc(sizeof(struct btnode));
144.     temp->value = value;
145.     temp->l = NULL;
146.     temp->r = NULL;
147.     return temp;
148. }
149.
150. /* Code to display the elements of the binary tree */
151. void in_order_traversal(struct btnode *p)
152. {
153.     if (!p)
154.     {
155.         return;
156.     }
157.     in_order_traversal(p->l);
158.     printf("%d->", p->value);
159.     in_order_traversal(p->r);
160. }
161.
162. /* Code to find the sum of all elements in the tree */
163. void in_order_sum(struct btnode *p)
164. {
165.     if (!p)

```

```

166.      {
167.          return;
168.      }
169.      in_order_sum(p->l);
170.      sum = sum + p->value;
171.      in_order_sum(p->r);
172.  }

```

```

1. *
2. * C Program to Construct a Binary Search Tree and perform deletion, inorder
traversal on it
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct btnode
8. {
9.     int value;
10.    struct btnode *l;
11.    struct btnode *r;
12. }*root = NULL, *temp = NULL, *t2, *t1;
13.
14. void delete1();
15. void insert();
16. void delete();
17. void inorder(struct btnode *t);
18. void create();
19. void search(struct btnode *t);
20. void preorder(struct btnode *t);
21. void postorder(struct btnode *t);
22. void search1(struct btnode *t,int data);
23. int smallest(struct btnode *t);
24. int largest(struct btnode *t);
25.
26. int flag = 1;
27.
28. void main()
29. {
30.     int ch;
31.
32.     printf("\nOPERATIONS ---");
33.     printf("\n1 - Insert an element into tree\n");

```

```

34.     printf("2 - Delete an element from the tree\n");
35.     printf("3 - Inorder Traversal\n");
36.     printf("4 - Preorder Traversal\n");
37.     printf("5 - Postorder Traversal\n");
38.     printf("6 - Exit\n");
39.     while(1)
40.     {
41.         printf("\nEnter your choice : ");
42.         scanf("%d", &ch);
43.         switch (ch)
44.         {
45.             case 1:
46.                 insert();
47.                 break;
48.             case 2:
49.                 delete();
50.                 break;
51.             case 3:
52.                 inorder(root);
53.                 break;
54.             case 4:
55.                 preorder(root);
56.                 break;
57.             case 5:
58.                 postorder(root);
59.                 break;
60.             case 6:
61.                 exit(0);
62.             default :
63.                 printf("Wrong choice, Please enter correct choice   ");
64.                 break;
65.         }
66.     }
67. }
68.
69./* To insert a node in the tree */
70.void insert()
71.{
72.    create();
73.    if (root == NULL)
74.        root = temp;
75.    else
76.        search(root);

```

```

77. }
78.
79./* To create a node */
80.void create()
81.{
82.    int data;
83.
84.    printf("Enter data of node to be inserted : ");
85.    scanf("%d", &data);
86.    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
87.    temp->value = data;
88.    temp->l = temp->r = NULL;
89.}
90.
91./* Function to search the appropriate position to insert the new node */
92.void search(struct btnode *t)
93.{
94.    if ((temp->value > t->value) && (t->r != NULL))           /* value more than root
   node value insert at right */
95.        search(t->r);
96.    else if ((temp->value > t->value) && (t->r == NULL))
97.        t->r = temp;
98.    else if ((temp->value < t->value) && (t->l != NULL))      /* value Less than root
   node value insert at left */
99.        search(t->l);
100.       else if ((temp->value < t->value) && (t->l == NULL))
101.           t->l = temp;
102.    }
103.
104.    /* recursive function to perform inorder traversal of tree */
105.    void inorder(struct btnode *t)
106.    {
107.        if (root == NULL)
108.        {
109.            printf("No elements in a tree to display");
110.            return;
111.        }
112.        if (t->l != NULL)
113.            inorder(t->l);
114.        printf("%d -> ", t->value);
115.        if (t->r != NULL)
116.            inorder(t->r);
117.    }

```

```

118.
119.     /* To check for the deleted node */
120.     void delete()
121.     {
122.         int data;
123.
124.         if (root == NULL)
125.         {
126.             printf("No elements in a tree to delete");
127.             return;
128.         }
129.         printf("Enter the data to be deleted : ");
130.         scanf("%d", &data);
131.         t1 = root;
132.         t2 = root;
133.         search1(root, data);
134.     }
135.
136.     /* To find the preorder traversal */
137.     void preorder(struct btnode *t)
138.     {
139.         if (root == NULL)
140.         {
141.             printf("No elements in a tree to display");
142.             return;
143.         }
144.         printf("%d -> ", t->value);
145.         if (t->l != NULL)
146.             preorder(t->l);
147.         if (t->r != NULL)
148.             preorder(t->r);
149.     }
150.
151.     /* To find the postorder traversal */
152.     void postorder(struct btnode *t)
153.     {
154.         if (root == NULL)
155.         {
156.             printf("No elements in a tree to display ");
157.             return;
158.         }
159.         if (t->l != NULL)
160.             postorder(t->l);

```

```

161.         if (t->r != NULL)
162.             postorder(t->r);
163.         printf("%d -> ", t->value);
164.     }
165.
166.     /* Search for the appropriate position to insert the new node */
167.     void search1(struct btnode *t, int data)
168.     {
169.         if ((data>t->value))
170.         {
171.             t1 = t;
172.             search1(t->r, data);
173.         }
174.         else if ((data < t->value))
175.         {
176.             t1 = t;
177.             search1(t->l, data);
178.         }
179.         else if ((data==t->value))
180.         {
181.             delete1(t);
182.         }
183.     }
184.
185.     /* To delete a node */
186.     void delete1(struct btnode *t)
187.     {
188.         int k;
189.
190.         /* To delete Leaf node */
191.         if ((t->l == NULL) && (t->r == NULL))
192.         {
193.             if (t1->l == t)
194.             {
195.                 t1->l = NULL;
196.             }
197.             else
198.             {
199.                 t1->r = NULL;
200.             }
201.             t = NULL;
202.             free(t);
203.             return;

```

```
204.     }
205.
206.     /* To delete node having one Left hand child */
207.     else if ((t->r == NULL))
208.     {
209.         if (t1 == t)
210.         {
211.             root = t->l;
212.             t1 = root;
213.         }
214.         else if (t1->l == t)
215.         {
216.             t1->l = t->l;
217.
218.         }
219.         else
220.         {
221.             t1->r = t->l;
222.         }
223.         t = NULL;
224.         free(t);
225.         return;
226.     }
227.
228.     /* To delete node having right hand child */
229.     else if (t->l == NULL)
230.     {
231.         if (t1 == t)
232.         {
233.             root = t->r;
234.             t1 = root;
235.         }
236.         else if (t1->r == t)
237.             t1->r = t->r;
238.         else
239.             t1->l = t->r;
240.         t == NULL;
241.         free(t);
242.         return;
243.     }
244.
245.     /* To delete node having two child */
246.     else if ((t->l != NULL) && (t->r != NULL))
```

```

247.     {
248.         t2 = root;
249.         if (t->r != NULL)
250.         {
251.             k = smallest(t->r);
252.             flag = 1;
253.         }
254.         else
255.         {
256.             k = largest(t->l);
257.             flag = 2;
258.         }
259.         search1(root, k);
260.         t->value = k;
261.     }
262.
263. }
264.
265. /* To find the smallest element in the right sub tree */
266. int smallest(struct btnode *t)
267. {
268.     t2 = t;
269.     if (t->l != NULL)
270.     {
271.         t2 = t;
272.         return(smallest(t->l));
273.     }
274.     else
275.         return (t->value);
276. }
277.
278. /* To find the largest element in the left sub tree */
279. int largest(struct btnode *t)
280. {
281.     if (t->r != NULL)
282.     {
283.         t2 = t;
284.         return(largest(t->r));
285.     }
286.     else
287.         return(t->value);
288. }

```

```
1. /*
2.  * C Program to Count Number of Leaf Nodes in a Tree
3.
4.      50
5.      / \
6.     20   30
7.     / \
8.    70  80
9.   / \   \
10. 10   40   60
11. (50,20,30,70,80,10,40,60)
12. */
13. #include <stdio.h>
14. #include <stdlib.h>
15.
16. struct btnode {
17.     int value;
18.     struct btnode * l;
19.     struct btnode * r;
20. };
21.
22. typedef struct btnode bt;
23.
24. bt *root;
25. bt *new, *list;
26. int count = 0;
27.
28. bt * create_node();
29. void display(bt *);
30. void construct_tree();
31. void count_leaf(bt *);
32.
33. void main()
34. {
35.     construct_tree();
36.     display(root);
37.     count_leaf(root);
38.     printf("\n leaf nodes are : %d",count);
39. }
40.
41. /* To create a empty node */
```

```
42. bt * create_node()
43. {
44.     new = (bt *)malloc(sizeof(bt));
45.     new->l = NULL;
46.     new->r = NULL;
47. }
48.
49./* To construct a tree */
50.void construct_tree()
51.{
52.    root = create_node();
53.    root->value = 50;
54.    root->l = create_node();
55.    root->l->value = 20;
56.    root->r = create_node();
57.    root->r->value = 30;
58.    root->l->l = create_node();
59.    root->l->l->value = 70;
60.    root->l->r = create_node();
61.    root->l->r->value = 80;
62.    root->l->r->r = create_node();
63.    root->l->r->r->value = 60;
64.    root->l->l->l = create_node();
65.    root->l->l->l->value = 10;
66.    root->l->l->r = create_node();
67.    root->l->l->r->value = 40;
68. }
69.
70./* To display the elements in a tree using inorder */
71.void display(bt * list)
72.{
73.    if (list == NULL)
74.    {
75.        return;
76.    }
77.    display(list->l);
78.    printf("->%d", list->value);
79.    display(list->r);
80. }
81.
82./* To count the number of Leaf nodes */
83.void count_leaf(bt * list)
84.{

}
```

```

85.     if (list == NULL)
86.     {
87.         return;
88.     }
89.     if (list->l == NULL && list->r == NULL)
90.     {
91.         count++;
92.     }
93.     count_leaf(list->l);
94.     count_leaf(list->r);
95. }
```

```

1. /*
2. * C Program to Find Nodes which are at Maximum Distance in Binary Tree
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct btnode
8. {
9.     int value;
10.    struct btnode *r,*l;
11. } *root = NULL, *temp = NULL;
12.
13. void create();
14. void insert();
15. void add(struct btnode *t);
16. void maxdistance(struct btnode *t);
17.
18. int count = 0, max = 0, v[100] = {0}, z = 0, max2, max1[100] = {0};
19.
20. void main()
21. {
22.     int ch, i;
23.
24.     printf("Program to find nodes at maximum distance");
25.     printf("\n OPERATIONS ----");
26.     printf("\n1] Insert ");
27.     printf("\n2] Display nodes ");
28.     printf("\n3] Exit ");
29.     while (1)
30.     {
```

```

31.     printf("\nEnter your choice : ");
32.     scanf("%d", &ch);
33.     switch (ch)
34.     {
35.     case 1:
36.         insert();
37.         break;
38.     case 2:
39.         max = 0;
40.         count = 0;
41.         maxdistance(root);
42.         for (i = 1; i < z; i++)
43.         {
44.             max2 = max1[0];
45.             if (max2 < max1[i])
46.                 max2 = max1[i];
47.         }
48.         printf("Maximum distance nodes \nNodes\t Distance ");
49.         for (i = 0; i < z; i++)
50.         {
51.             if (max2 == max1[i])
52.                 printf("\n %d\t %d ", v[i], max2);
53.         }
54.         break;
55.     case 3:
56.         exit(0);
57.     default :
58.         printf("Wrong choice, Please enter correct choice   ");
59.         break;
60.     }
61. }
62. */
63.
64./* To create a new node with the data from the user */
65.void create()
66.{
67.    int data;
68.
69.    printf("Enter the data of node : ");
70.    scanf("%d", &data);
71.    temp = (struct btnode* ) malloc(1*(sizeof(struct btnode)));
72.    temp->value = data;
73.    temp->l = temp->r = NULL;

```

```

74. }
75.
76./* To check for root node and then create it */
77.void insert()
78.{
79.    create();
80.
81.    if (root == NULL)
82.        root = temp;
83.    else
84.        add(root);
85.}
86.
87./* Search for the appropriate position to insert the new node */
88.void add(struct btnode *t)
89.{
90.    if ((temp->value > t->value) && (t->r!=NULL))      /* value more than root node
   value insert at right */
91.        add(t->r);
92.    else if ((temp->value > t->value) && (t->r==NULL))
93.        t->r = temp;
94.    else if ((temp->value < t->value) && (t->l!=NULL))      /* value less than root
   node value insert at left */
95.        add(t->l);
96.    else if ((temp->value < t->value) && (t->l==NULL))
97.        t->l = temp;
98.}
99.
100.   /* Function to find the max distance nodes */
101.   void maxdistance(struct btnode *t)
102.   {
103.       if (t->l!=NULL)
104.       {
105.           count++;          /* to count the number of nodes in between root
   and Leaf */
106.           maxdistance(t->l);
107.       }
108.       if (max < count)
109.           max = count;
110.       if (max == count)
111.       {
112.           max1[z] = max;
113.           v[z] = t->value;

```

```

114.             z++;
115.         }
116.         if (t->r != NULL)
117.         {
118.             count++;
119.             maxdistance(t->r);
120.         }
121.         count--;
122.     }

```

## (10.) C Programming Examples on File Handling

### 1. C Examples on File Creation

```

1. /*
2.  * C program to create a file called emp.rec and store information
3.  * about a person, in terms of his name, age and salary.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     FILE *fptr;
10.    char name[20];
11.    int age;
12.    float salary;
13.
14.    /* open for writing */
15.    fptr = fopen("emp.rec", "w");
16.
17.    if (fptr == NULL)
18.    {
19.        printf("File does not exists \n");
20.        return;
21.    }
22.    printf("Enter the name \n");
23.    scanf("%s", name);
24.    fprintf(fptr, "Name      = %s\n", name);
25.    printf("Enter the age\n");
26.    scanf("%d", &age);
27.    fprintf(fptr, "Age      = %d\n", age);
28.    printf("Enter the salary\n");

```

```

29.     scanf("%f", &salary);
30.     fprintf(fptr, "Salary = %.2f\n", salary);
31.     fclose(fptr);
32. }
```

```

1. /*
2. * C program to illustrate how a file stored on the disk is read
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void main()
8. {
9.     FILE *fptr;
10.    char filename[15];
11.    char ch;
12.
13.    printf("Enter the filename to be opened \n");
14.    scanf("%s", filename);
15.    /* open the file for reading */
16.    fptr = fopen(filename, "r");
17.    if (fptr == NULL)
18.    {
19.        printf("Cannot open file \n");
20.        exit(0);
21.    }
22.    ch = fgetc(fptr);
23.    while (ch != EOF)
24.    {
25.        printf ("%c", ch);
26.        ch = fgetc(fptr);
27.    }
28.    fclose(fptr);
29. }
```

## 2. C Examples on File Content Deletion

```

1. /*
2. * C Program Delete a specific Line from a Text File
3. */
```

```
4. #include <stdio.h>
5.
6. int main()
7. {
8.     FILE *fileptr1, *fileptr2;
9.     char filename[40];
10.    char ch;
11.    int delete_line, temp = 1;
12.
13.    printf("Enter file name: ");
14.    scanf("%s", filename);
15.    //open file in read mode
16.    fileptr1 = fopen(filename, "r");
17.    ch = getc(fileptr1);
18.    while (ch != EOF)
19.    {
20.        printf("%c", ch);
21.        ch = getc(fileptr1);
22.    }
23.    //rewind
24.    rewind(fileptr1);
25.    printf("\n Enter line number of the line to be deleted:");
26.    scanf("%d", &delete_line);
27.    //open new file in write mode
28.    fileptr2 = fopen("replica.c", "w");
29.    ch = getc(fileptr1);
30.    while (ch != EOF)
31.    {
32.        ch = getc(fileptr1);
33.        if (ch == '\n')
34.            temp++;
35.        //except the line to be deleted
36.        if (temp != delete_line)
37.        {
38.            //copy all lines in file replica.c
39.            putc(ch, fileptr2);
40.        }
41.    }
42.    fclose(fileptr1);
43.    fclose(fileptr2);
44.    remove(filename);
45.    //rename the file replica.c to original name
46.    rename("replica.c", filename);
```

```

47.     printf("\n The contents of file after being modified are as follows:\n");
48.     fileptr1 = fopen(filename, "r");
49.     ch = getc(fileptr1);
50.     while (ch != EOF)
51.     {
52.         printf("%c", ch);
53.         ch = getc(fileptr1);
54.     }
55.     fclose(fileptr1);
56.     return 0;
57. }
```

```

1. /*
2.  * C Program to Replace a specified Line in a Text File
3. */
4. #include <stdio.h>
5.
6. int main(void)
7. {
8.     FILE *fileptr1, *fileptr2;
9.     char filechar[40];
10.    char c;
11.    int delete_line, temp = 1;
12.
13.    printf("Enter file name: ");
14.    scanf("%s", filechar);
15.    fileptr1 = fopen(filechar, "r");
16.    c = getc(fileptr1);
17.    //print the contents of file .
18.    while (c != EOF)
19.    {
20.        printf("%c", c);
21.        c = getc(fileptr1);
22.    }
23.    printf(" \n Enter line number to be deleted and replaced");
24.    scanf("%d", &delete_line);
25.    //take fileptr1 to start point.
26.    rewind(fileptr1);
27.    //open replica.c in write mode
28.    fileptr2 = fopen("replica.c", "w");
29.    c = getc(fileptr1);
```

```

30.     while (c != EOF)
31.     {
32.         if (c == '\n')
33.         {
34.             temp++;
35.         }
36.         //till the Line to be deleted comes, copy the content to other
37.         if (temp != delete_line)
38.         {
39.             putc(c, fileptr2);
40.         }
41.         else
42.         {
43.             while ((c = getc(fileptr1)) != '\n')
44.             {
45.             }
46.             //read and skip the line ask for new text
47.             printf("Enter new text");
48.             //flush the input stream
49.             fflush(stdin);
50.             putc('\n', fileptr2);
51.             //put '\n' in new file
52.             while ((c = getchar()) != '\n')
53.                 putc(c, fileptr2);
54.             //take the data from user and place it in new file
55.             fputs("\n", fileptr2);
56.             temp++;
57.         }
58.         //continue this till EOF is encountered
59.         c = getc(fileptr1);
60.     }
61.     fclose(fileptr1);
62.     fclose(fileptr2);
63.     remove(filechar);
64.     rename("replica.c", filechar);
65.     fileptr1 = fopen(filechar, "r");
66.     //reads the character from file
67.     c = getc(fileptr1);
68.     //until last character of file is encountered
69.     while (c != EOF)
70.     {
71.         printf("%c", c);
72.         //all characters are printed

```

```

73.         c = getc(fileptr1);
74.     }
75.     fclose(fileptr1);
76.     return 0;
77. }
```

```

1. /*
2. * C Program to Find the Number of Lines in a Text File
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     FILE *fileptr;
9.     int count_lines = 0;
10.    char filechar[40], chr;
11.
12.    printf("Enter file name: ");
13.    scanf("%s", filechar);
14.    fileptr = fopen(filechar, "r");
15.    //extract character from file and store in chr
16.    chr = getc(fileptr);
17.    while (chr != EOF)
18.    {
19.        //Count whenever new line is encountered
20.        if (chr == '\n')
21.        {
22.            count_lines = count_lines + 1;
23.        }
24.        //take next character from file.
25.        chr = getc(fileptr);
26.    }
27.    fclose(fileptr); //close file.
28.    printf("There are %d lines in %s in a file\n", count_lines, filechar);
29.    return 0;
30. }
```

### 3. C Examples on Appending and Merging the Contents of Files

```
1. /*
```

```
2.  * C Program to Append the Content of File at the end of Another
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. main()
8. {
9.     FILE *fsring1, *fsring2, *ftemp;
10.    char ch, file1[20], file2[20], file3[20];
11.
12.    printf("Enter name of first file ");
13.    gets(file1);
14.    printf("Enter name of second file ");
15.    gets(file2);
16.    printf("Enter name to store merged file ");
17.    gets(file3);
18.    fsring1 = fopen(file1, "r");
19.    fsring2 = fopen(file2, "r");
20.    if (fsring1 == NULL || fsring2 == NULL)
21.    {
22.        perror("Error has occurred");
23.        printf("Press any key to exit...\\n");
24.        exit(EXIT_FAILURE);
25.    }
26.    ftemp = fopen(file3, "w");
27.    if (ftemp == NULL)
28.    {
29.        perror("Error has occurred");
30.        printf("Press any key to exit...\\n");
31.        exit(EXIT_FAILURE);
32.    }
33.    while ((ch = fgetc(fsring1)) != EOF)
34.        fputc(ch, ftemp);
35.    while ((ch = fgetc(fsring2)) != EOF)
36.        fputc(ch, ftemp);
37.    printf("Two files merged %s successfully.\\n", file3);
38.    fclose(fsring1);
39.    fclose(fsring2);
40.    fclose(ftemp);
41.    return 0;
42. }
```

```

1. /*
2.  * C Program that Merges Lines Alternatively from 2 Files & Print Result
3. */
4. #include<stdio.h>
5. main()
6. {
7.     char file1[10], file2[10];
8.
9.     puts("enter the name of file 1");           /*getting the names of file to be
   concatenated*/
10.    scanf("%s", file1);
11.    puts("enter the name of file 2");
12.    scanf("%s", file2);
13.    FILE *fptr1, *fptr2, *fptr3;
14.    fptr1=fopen(file1, "r");                  /*opening the files in read only mode*/
15.    fptr2=fopen(file2, "r");
16.    fptr3=fopen("merge2.txt", "w+");        /*opening a new file in write,update mode*/
17.    char str1[200];
18.    char ch1, ch2;
19.    int n = 0, w = 0;
20.    while (((ch1=fgetc(fptr1)) != EOF) && ((ch2 = fgetc(fptr2)) != EOF))
21.    {
22.        if (ch1 != EOF)                   /*getting lines in alternately from two files*/
23.        {
24.            ungetc(ch1, fptr1);
25.            fgets(str1, 199, fptr1);
26.            fputs(str1, fptr3);
27.            if (str1[0] != '\n')
28.                n++;          /*counting no. of lines*/
29.        }
30.        if (ch2 != EOF)
31.        {
32.            ungetc(ch2, fptr2);
33.            fgets(str1, 199, fptr2);
34.            fputs(str1, fptr3);
35.            if (str1[0] != '\n')
36.                n++;          /*counting no.of lines*/
37.        }
38.    }
39.    rewind(fptr3);
40.    while ((ch1 = fgetc(fptr3)) != EOF)        /*countig no.of words*/
41.    {
42.        ungetc(ch1, fptr3);

```

```

43.     fscanf(fp3, "%s", str1);
44.     if (str1[0] != ' ' || str1[0] != '\n')
45.         w++;
46. }
47. fprintf(fp3, "\n\n number of lines = %d n number of words is = %d\n", n, w - 1);
48. /*appending comments in the concatenated file*/
49. fclose(fp1);
50. fclose(fp2);
51. fclose(fp3);
52. }
```

```

1. /*
2. * C Program to List Files in Directory
3. */
4. #include <dirent.h>
5. #include <stdio.h>
6.
7. int main(void)
8. {
9.     DIR *d;
10.    struct dirent *dir;
11.    d = opendir(".");
12.    if (d)
13.    {
14.        while ((dir = readdir(d)) != NULL)
15.        {
16.            printf("%s\n", dir->d_name);
17.        }
18.        closedir(d);
19.    }
20.    return(0);
21. }
```

```

1. /*
2. * C Program to Find Sum of Numbers given in Command Line Arguments
3. * Recursively
4. */
5. #include <stdio.h>
6.
```

```

7. int count, s = 0;
8. void sum(int *, int *);
9.
10. void main(int argc, char *argv[])
11. {
12.     int i, ar[argc];
13.     count = argc;
14.     for (i = 1; i < argc; i++)
15.     {
16.         ar[i - 1] = atoi(argv[i]);
17.     }
18.     sum(ar, ar + 1);
19.     printf("%d", s);
20. }
21.
22./* computes sum of two numbers recursively */
23. void sum(int *a, int * b)
24. {
25.     if (count == 1)
26.         return;
27.     s = s + *a + *b;
28.     count -= 2;
29.     sum(a + 2, b + 2);
30. }
```

```

1. /*
2.  * C program to Display the Function Names defined in C Source File
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void check(char *c,int p1, int p2);
8. void display(char *c, int p1);
9.
10. void main(int argc, char **argv)
11. {
12.     FILE *fp;
13.     char ch[100];
14.     char *pos1, *pos2, *pos3;
15.
16.     fp=fopen(argv[1], "r");
```

```

17.     if (fp == NULL)
18.     {
19.         printf("\nFile unable to open");
20.         return;
21.     }
22. else
23.     printf("\nFile Opened to display function names :\n");
24. while (1)
25. {
26.     if ((fgets(ch, 100, fp)) != NULL)
27.     {
28.         if ((strstr(ch, "/*")) == NULL)
29.         {
30.             pos1 = strchr(ch, '(');           /* check opening brace */
31.             if (pos1)
32.             {
33.                 pos2 = strchr(ch, ')');       /* check closing brace */
34.                 if (pos2)
35.                 {
36.                     pos3 = strchr(ch, ';');   /* check for semicolon */
37.                     if ((pos1 < pos2) && (pos3 == NULL) || (pos3 < pos1))
38.                     {
39.                         check(ch, pos1 - ch, pos2 - ch);
40.                     }
41.                     else    continue;
42.                 }
43.                 else    continue;
44.             }
45.             else    continue;
46.         }
47.         else    continue;
48.     }
49.     else    break;
50. }
51. fclose(fp);
52. }

54./* To check if it is a function */
55.void check(char *c, int p1, int p2)
56.{
57.    int i, flag = 0, temp = p1;
58.
59.    if ((c[p1 + 1] == ')'))

```

```

60.    {
61.        display(c, p1);
62.        return;
63.    }
64.    for (i = p1 + 1; i < p2; i++)
65.    {
66.        if ((c[i] != ' ') || (c[i] == ')'))
67.        {
68.            flag = 1;
69.
70.        }
71.        if (flag == 0)
72.        {
73.            display(c, p1);
74.            return;
75.        }
76.        else
77.        {
78.            flag = 0;
79.            while (c[--temp] != ' ');
80.            for (i = 0; i < temp; i++)
81.                if (c[i]==' ')
82.                {
83.                    flag = 1;
84.                }
85.                if (flag == 0)
86.                {
87.                    display(c, p1);
88.                    return;
89.                }
90.                else
91.                    return;
92.            }
93.        }
94.    }
95.
96./* To display function name */
97.void display(char *c,int p1)
98.{
99.    int temp = p1, i;
100.
101.    while (c[--temp] != ' ');

```

```

102.         for (i = temp + 1; i < p1; i++)
103.             printf("%c", c[i]);
104.         printf("\n");
105.     return;
106.
107. }

```

```

1. /*
2. * C Program to Find the Size of File using File Handling Function
3. */
4. #include <stdio.h>
5.
6. void main(int argc, char **argv)
7. {
8.     FILE *fp;
9.     char ch;
10.    int size = 0;
11.
12.    fp = fopen(argv[1], "r");
13.    if (fp == NULL)
14.        printf("\nFile unable to open ");
15.    else
16.        printf("\nFile opened ");
17.    fseek(fp, 0, 2); /* file pointer at the end of file */
18.    size = ftell(fp); /* take a position of file pointer un size variable */
19.    printf("The size of given file is : %d\n", size);
20.    fclose(fp);
21. }

```

#### 4. C Examples on Create, Copy, Comparison and Printing the Contents of the File

```

1. /*
2. * C Program to Capitalize First Letter of every Word in a File
3. */
4. #include <stdio.h>
5. #include <fcntl.h>
6. #include <stdlib.h>
7. int to_initcap_file(FILE *);

```

```
8.
9. void main(int argc, char * argv[])
10. {
11.     FILE *fp1;
12.     char fp[10];
13.     int p;
14.
15.     fp1 = fopen(argv[1], "r+");
16.     if (fp1 == NULL)
17.     {
18.         printf("cannot open the file ");
19.         exit(0);
20.     }
21.     p = to_initcap_file(fp1);
22.     if (p == 1)
23.     {
24.         printf("success");
25.     }
26.     else
27.     {
28.         printf("failure");
29.     }
30.     fclose(fp1);
31. }
32.
33./* capitalizes first letter of every word */
34.int to_initcap_file(FILE *fp)
35.{
36.    char c;
37.
38.    c = fgetc(fp);
39.    if (c >= 'a' && c <= 'z')
40.    {
41.        fseek(fp, -1L, 1);
42.        fputc(c - 32, fp);
43.    }
44.    while(c != EOF)
45.    {
46.        if (c == ' ' || c == '\n')
47.        {
48.            c = fgetc(fp);
49.            if (c >= 'a' && c <= 'z')
50.            {
```

```

51.             fseek(fp, -1L, 1);
52.             fputc(c - 32, fp);
53.         }
54.     }
55. else
56. {
57.     c = fgetc(fp);
58. }
59. }
60. return 1;
61. }
```

```

1. /*
2. * C Program to Print Environment variables
3. */
4. #include <stdio.h>
5.
6. void main(int argc, char *argv[], char * envp[])
7. {
8.     int i;
9.
10.    for (i = 0; envp[i] != NULL; i++)
11.    {
12.        printf("\n%s", envp[i]);
13.    }
14. }
```

```

1. /*
2. * C Program to Copy a File into Another File
3. */
4. #include <stdio.h>
5.
6. void main(int argc,char **argv)
7. {
8.     FILE *fp1, *fp2;
9.     char ch;
10.    int pos;
11.
12.    if ((fp1 = fopen(argv[1],"r")) == NULL)
13.    {
```

```

14.     printf("\nFile cannot be opened");
15.     return;
16. }
17. else
18. {
19.     printf("\nFile opened for copy...\n ");
20. }
21. fp2 = fopen(argv[2], "w");
22. fseek(fp1, 0L, SEEK_END); // file pointer at end of file
23. pos = ftell(fp1);
24. fseek(fp1, 0L, SEEK_SET); // file pointer set at start
25. while (pos--)
26. {
27.     ch = fgetc(fp1); // copying file character by character
28.     fputc(ch, fp2);
29. }
30. fcloseall();
31. }
```

```

1. /*
2. * C Program to Create Employee Record and Update it
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. #define size 200
8.
9. struct emp
10. {
11.     int id;
12.     char *name;
13. }*emp1, *emp3;
14.
15. void display();
16. void create();
17. void update();
18.
19. FILE *fp, *fp1;
20. int count = 0;
21.
22. void main(int argc, char **argv)
```

```

23. {
24.     int i, n, ch;
25.
26.     printf("1] Create a Record\n");
27.     printf("2] Display Records\n");
28.     printf("3] Update Records\n");
29.     printf("4] Exit");
30.     while (1)
31.     {
32.         printf("\nEnter your choice : ");
33.         scanf("%d", &ch);
34.         switch (ch)
35.         {
36.             case 1:
37.                 fp = fopen(argv[1], "a");
38.                 create();
39.                 break;
40.             case 2:
41.                 fp1 = fopen(argv[1],"rb");
42.                 display();
43.                 break;
44.             case 3:
45.                 fp1 = fopen(argv[1], "r+");
46.                 update();
47.                 break;
48.             case 4:
49.                 exit(0);
50.         }
51.     }
52. }
53.
54. /* To create an employee record */
55. void create()
56. {
57.     int i;
58.     char *p;
59.
60.     emp1 = (struct emp *)malloc(sizeof(struct emp));
61.     emp1->name = (char *)malloc((size)*(sizeof(char)));
62.     printf("Enter name of employee : ");
63.     scanf(" %[^\n]s", emp1->name);
64.     printf("Enter emp id : ");
65.     scanf(" %d", &emp1->id);

```

```

66.     fwrite(&emp1->id, sizeof(emp1->id), 1, fp);
67.     fwrite(emp1->name, size, 1, fp);
68.     count++; // count to number of entries of records
69.     fclose(fp);
70. }
71.
72. /* Display the records in the file */
73. void display()
74. {
75.     emp3=(struct emp *)malloc(1*sizeof(struct emp));
76.     emp3->name=(char *)malloc(size*sizeof(char));
77.     int i = 1;
78.
79.     if (fp1 == NULL)
80.         printf("\nFile not opened for reading");
81.     while (i <= count)
82.     {
83.         fread(&emp3->id, sizeof(emp3->id), 1, fp1);
84.         fread(emp3->name, size, 1, fp1);
85.         printf("\n%d %s",emp3->id,emp3->name);
86.         i++;
87.     }
88.     fclose(fp1);
89.     free(emp3->name);
90.     free(emp3);
91. }
92.
93. void update()
94. {
95.     int id, flag = 0, i = 1;
96.     char s[size];
97.
98.     if (fp1 == NULL)
99.     {
100.         printf("File cant be opened");
101.         return;
102.     }
103.     printf("Enter employee id to update : ");
104.     scanf("%d", &id);
105.     emp3 = (struct emp *)malloc(1*sizeof(struct emp));
106.     emp3->name=(char *)malloc(size*sizeof(char));
107.     while(i<=count)
108.     {

```

```

109.         fread(&emp3->id, sizeof(emp3->id), 1, fp1);
110.         fread(emp3->name, size, 1, fp1);
111.         if (id == emp3->id)
112.         {
113.             printf("Enter new name of employee to update : ");
114.             scanf(" %[^\n]s", s);
115.             fseek(fp1, -204L, SEEK_CUR);
116.             fwrite(&emp3->id, sizeof(emp3->id), 1, fp1);
117.             fwrite(s, size, 1, fp1);
118.             flag = 1;
119.             break;
120.         }
121.         i++;
122.     }
123.     if (flag != 1)
124.     {
125.         printf("No employee record found");
126.         flag = 0;
127.     }
128.     fclose(fp1);
129.     free(emp3->name); /* to free allocated memory */
130.     free(emp3);
131. }
```

```

1. /*
2. * C Program to Compare two Binary Files, Printing the First Byte
3. * Position where they Differ
4. */
5. #include <stdio.h>
6.
7. void compare_two_binary_files(FILE *,FILE *);
8.
9. int main(int argc, char *argv[])
10. {
11.     FILE *fp1, *fp2;
12.
13.     if (argc < 3)
14.     {
15.         printf("\nInsufficient Arguments: \n");
16.         printf("\nHelp:./executable <filename1> <filename2>\n");
17.         return;
```

```

18.     }
19. else
20. {
21.     fp1 = fopen(argv[1], "r");
22.     if (fp1 == NULL)
23.     {
24.         printf("\nError in opening file %s", argv[1]);
25.         return;
26.     }
27.
28.     fp2 = fopen(argv[2], "r");
29.
30.     if (fp2 == NULL)
31.     {
32.         printf("\nError in opening file %s", argv[2]);
33.         return;
34.     }
35.
36.     if ((fp1 != NULL) && (fp2 != NULL))
37.     {
38.         compare_two_binary_files(fp1, fp2);
39.     }
40. }
41. }
42.
43./*
44. * compare two binary files character by character
45. */
46.void compare_two_binary_files(FILE *fp1, FILE *fp2)
47.{
48.    char ch1, ch2;
49.    int flag = 0;
50.
51.    while (((ch1 = fgetc(fp1)) != EOF) && ((ch2 = fgetc(fp2)) != EOF))
52.    {
53.        /*
54.         * character by character comparision
55.         * if equal then continue by comparing till the end of files
56.         */
57.        if (ch1 == ch2)
58.        {
59.            flag = 1;
60.            continue;

```

```

61.      }
62.      /*
63.       * If not equal then returns the byte position
64.       */
65.     else
66.     {
67.       fseek(fp1, -1, SEEK_CUR);
68.       flag = 0;
69.       break;
70.     }
71.   }
72.
73.   if (flag == 0)
74.   {
75.     printf("Two files are not equal : byte position at which two files differ
76.           is %d\n", ftell(fp1)+1);
77.   }
78.   else
79.   {
80.     printf("Two files are Equal\n ", ftell(fp1)+1);
81.   }

```

## 5. C Examples on Convert, Replace, Count and Reverse Operations on the Contents of a File

```

1. /*
2.  * C Program to Convert the Content of File to UpperCase
3. */
4. #include <stdio.h>
5.
6. int to_upper_file(FILE *);
7.
8. int main(int argc,char *argv[])
9. {
10.   FILE *fp;
11.   int status;
12.
13.   if (argc == 1)
14.   {
15.     printf("Insufficient Arguments:");
16.     printf("No File name is provided at command line");

```

```
17.         return;
18.     }
19.     if (argc > 1)
20.     {
21.         fp = fopen(argv[1],"r+");
22.         status = to_upper_file(fp);
23.
24.         /*
25.             *If the status returned is 0 then the coversion of file content was
26.             completed successfully
27.         */
28.
29.         if (status == 0)
30.         {
31.             printf("\n The content of \"%s\" file was successfully converted to
32.             upper case\n",argv[1]);
33.             return;
34.         }
35.         /*
36.             * If the status returns is -1 then the conversion of file content was not
37.             done
38.         */
39.         if (status == -1)
40.         {
41.             printf("\n Failed to convert");
42.             return;
43.         }
44.     }
45.     /* convert the file content to uppercase
46.     */
47. int to_upper_file(FILE *fp)
48. {
49.     char ch;
50.
51.     if (fp == NULL)
52.     {
53.         perror("Unable to open file");
54.         return -1;
55.     }
56.     else
```

```

57.     {
58.         /*
59.             * Read the file content and convert to uppercase
60.             */
61.         while (ch != EOF)
62.         {
63.             ch = fgetc(fp);
64.             if ((ch >= 'a') && (ch <= 'z'))
65.             {
66.                 ch = ch - 32;
67.                 fseek(fp, -1, SEEK_CUR);
68.                 fputc(ch, fp);
69.             }
70.         }
71.         return 0;
72.     }
73. }
```

```

1. /*
2.  * C Program to replace first letter of every word with caps
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. void main(int argc, char *argv[])
8. {
9.     FILE *fp1;
10.    int return_val;
11.
12.    if ((fp1 = fopen(argv[1], "r+")) == NULL)
13.    {
14.        printf("file cant be opened");
15.        exit(0);
16.    }
17.    return_val = init_cap_file(fp1);
18.    if (return_val == 1)
19.    {
20.        printf("\nsuccess");
21.    }
22.    else
23.    {
```

```

24.         printf("\n failure");
25.     }
26. }
27.
28. int init_cap_file(FILE *fp1)
29. {
30.     char ch;
31.
32.     ch = fgetc(fp1);
33.     if (ch >= 97 && ch <= 122)
34.     {
35.         fseek(fp1, -1L, 1);
36.         fputc(ch - 32, fp1);
37.     }
38.     while (ch != EOF)
39.     {
40.         if (ch == ' ' || ch == '\n')
41.         {
42.             ch = fgetc(fp1);
43.             if (ch >= 97 && ch <= 122)
44.             {
45.                 fseek(fp1, -1L, 1);
46.                 fputc(ch - 32, fp1);
47.             }
48.         }
49.         else
50.             ch = fgetc(fp1);
51.     }
52.     return 1;
53. }
```

```

1. /*
2.  * C Program to Count No of Lines, BLank Lines, Comments in a given Program
3.  */
4. #include <stdio.h>
5.
6. void main(int argc, char* argv[])
7. {
8.     int line_count = 0, n_o_c_l = 0, n_o_n_b_l = 0, n_o_b_l = 0, n_e_c = 0;
9.     FILE *fp1;
10.    char ch;
```

```

11.     fp1 = fopen(argv[1], "r");
12.
13.     while ((ch = fgetc(fp1)) != EOF)
14.     {
15.         if (ch == '\n')
16.         {
17.             line_count++;
18.         }
19.         if (ch == '\n')
20.         {
21.             if ((ch = fgetc(fp1)) == '\n')
22.             {
23.                 fseek(fp1, -1, 1);
24.                 n_o_b_l++;
25.             }
26.         }
27.         if (ch == ';')
28.         {
29.             if ((ch = fgetc(fp1)) == '\n')
30.             {
31.                 fseek(fp1, -1, 1);
32.                 n_e_c++;
33.             }
34.         }
35.     }
36.     fseek(fp1, 0, 0);
37.     while ((ch = fgetc(fp1)) != EOF)
38.     {
39.         if (ch == '/')
40.         {
41.             if ((ch = fgetc(fp1)) == '/')
42.             {
43.                 n_o_c_l++;
44.             }
45.         }
46.     }
47.     printf("Total no of lines: %d\n", line_count);
48.     printf("Total no of comment line: %d\n", n_o_c_l);
49.     printf("Total no of blank lines: %d\n", n_o_b_l);
50.     printf("Total no of non blank lines: %d\n", line_count-n_o_b_l);
51.     printf("Total no of lines end with semicolon: %d\n", n_e_c);
52. }
```

```
1. *
2. * C Program to Reverse the Contents of a File and Print it
3. */
4. #include <stdio.h>
5. #include <errno.h>
6.
7. long count_characters(FILE *);
8.
9. void main(int argc, char * argv[])
10. {
11.     int i;
12.     long cnt;
13.     char ch, ch1;
14.     FILE *fp1, *fp2;
15.
16.     if (fp1 = fopen(argv[1], "r"))
17.     {
18.         printf("The FILE has been opened...\\n");
19.         fp2 = fopen(argv[2], "w");
20.         cnt = count_characters(fp1); // to count the total number of characters
21.                                     // inside the source file
22.         fseek(fp1, -1L, 2);        // makes the pointer fp1 to point at the last
23.                                     // character of the file
24.         printf("Number of characters to be copied %d\\n", ftell(fp1));
25.
26.         while (cnt)
27.         {
28.             ch = fgetc(fp1);
29.             fputc(ch, fp2);
30.             fseek(fp1, -2L, 1);    // shifts the pointer to the previous character
31.             cnt--;
32.         }
33.         printf("\\n**File copied successfully in reverse order**\\n");
34.     }
35.     perror("Error occurred\\n");
36. }
37. fclose(fp1);
38. fclose(fp2);
39. }
```

```

40. // count the total number of characters in the file that *f points to
41. long count_characters(FILE *f)
42. {
43.     fseek(f, -1L, 2);
44.     long last_pos = ftell(f); // returns the position of the last element of the
45.     file
46.     last_pos++;
47. }

```

```

1. /*
2.  * C Program to Convert the Content of File to LowerCase
3. */
4. #include <stdio.h>
5. #include <errno.h>
6.
7. int to_lower_file(FILE *);
8.
9. void main(int argc, char * argv[])
10. {
11.     int op = -1;
12.     char ch;
13.     FILE *fp;
14.     if (fp = fopen(argv[1], "r+"))
15.     {
16.         printf("FILE has been opened..!!!\n");
17.         op = to_lower_file(fp);
18.         printf(" %d \n", op);
19.         fclose(fp);
20.     }
21.     else
22.     {
23.         perror("Error Occured");
24.         printf(" %d\n ", op);
25.     }
26. }
27.
28. int to_lower_file(FILE *f)
29. {
30.     int c;
31.     char ch;

```

```

32.     while ((ch = fgetc(f)) != EOF)
33.     {
34.         c = (int)ch;
35.         if (c >= 65 && c <= 90)
36.         {
37.             ch = ch + 32;
38.             fseek(f, -1L, 1);
39.             fputc(ch, f);
40.         }
41.     }
42.     return 0;
43. }
```

#### 44. 6. C Examples on Creation and Updating Operations on the Contents of the File

```

1. /*
2. * C Program to Update Details of Employee using Files
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. struct emp
8. {
9.     int empid;
10.    char *name;
11. };
12.
13. int count = 0;
14. void add_rec(char *a);
15. void display(char *a);
16. void update_rec(char *a);
17.
18. void main(int argc, char *argv[])
19. {
20.     int choice;
21.     while (1)
22.     {
23.         printf("MENU:\n");
24.         printf("1.Add a record\n");
25.         printf("2.Display the file\n");
26.         printf("3.Update the record\n");
```

```
27.     printf("Enter your choice:");
28.     scanf("%d", &choice);
29.
30.     switch(choice)
31.     {
32.     case 1:
33.         add_rec(argv[1]);
34.         break;
35.     case 2:
36.         display(argv[1]);
37.         break;
38.     case 3:
39.         update_rec(argv[1]);
40.         break;
41.     case 4:
42.         exit(0);
43.     default:
44.         printf("Wrong choice!!!\nEnter the correct choice\n");
45.     }
46. }
47.
48.
49. void add_rec(char *a)
50. {
51.     FILE *fp;
52.     fp = fopen(a, "a+");
53.     struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
54.     temp->name = (char *)malloc(50*sizeof(char));
55.     if (fp == NULL)
56.         printf("Error!!!");
57.     else
58.     {
59.         printf("Enter the employee id\n");
60.         scanf("%d", &temp->empid);
61.         fwrite(&temp->empid, sizeof(int), 1, fp);
62.         printf("enter the employee name\n");
63.         scanf(" %[^\n]s", temp->name);
64.         fwrite(temp->name, 50, 1, fp);
65.         count++;
66.     }
67.     fclose(fp);
68.     free(temp);
69.     free(temp->name);
```

```

70. }
71.
72. void display(char *a)
73. {
74.     FILE *fp;
75.     char ch;
76.     int rec = count;
77.     fp = fopen(a, "r");
78.     struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
79.     temp->name = (char *)malloc(50*sizeof(char));
80.     if (fp == NULL)
81.         printf("Error!!!");
82.     else
83.     {
84.         while (rec)
85.         {
86.             fread(&temp->empid, sizeof(int), 1, fp);
87.             printf("%d", temp->empid);
88.             fread(temp->name, 50, 1, fp);
89.             printf(" %s\n", temp->name);
90.             rec--;
91.         }
92.     }
93.     fclose(fp);
94.     free(temp);
95.     free(temp->name);
96. }
97.
98. void update_rec(char *a)
99. {
100.     FILE *fp;
101.     char ch, name[5];
102.     int rec, id, c;
103.     fp = fopen(a, "r+");
104.     struct emp *temp = (struct emp *)malloc(sizeof(struct emp));
105.     temp->name = (char *)malloc(50*sizeof(char));
106.     printf("Enter the employee id to update:\n");
107.     scanf("%d", &id);
108.     fseek(fp, 0, 0);
109.     rec = count;
110.     while (rec)
111.     {
112.         fread(&temp->empid, sizeof(int), 1, fp);

```

```

113.         printf("%d", temp->empid);
114.         if (id == temp->empid)
115.         {
116.             printf("Enter the employee name to be updated");
117.             scanf(" %[^\n]s", name);
118.             c = fwrite(name, 50, 1, fp);
119.             break;
120.         }
121.         fread(temp->name, 50, 1, fp);
122.         rec--;
123.     }
124.     if (c == 1)
125.         printf("Record updated\n");
126.     else
127.         printf("Update not successful\n");
128.     fclose(fp);
129.     free(temp);
130.     free(temp->name);
131. }
```

```

1. /*
2.  * C Program to Create Employee File Name Record that is taken from the Command-Line
3.  Argument
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7. #include <errno.h>
8. #include <string.h>
9.
10. struct emprec
11. {
12.     int empid;
13.     char *name;
14. };
15.
16. void insert(char *a);
17. void display(char *a);
18. void update(char *a);
19. int count;
20. void main(int argc, char *argv[])

```

```

21. {
22.     int choice;
23.
24.     while (1)
25.     {
26.         printf("Enter the choice\n");
27.         printf("1-Insert a new record into file\n2-Display the records\n");
28.         printf("3-Update the record\n4-Exit\n");
29.         scanf("%d", &choice);
30.         switch (choice)
31.         {
32.             case 1:
33.                 insert(argv[1]);
34.                 break;
35.             case 2:
36.                 display(argv[1]);
37.                 break;
38.             case 3:
39.                 update(argv[1]);
40.                 break;
41.             case 4:
42.                 exit(0);
43.             default:
44.                 printf("Enter the correct choice\n");
45.         }
46.     }
47. }
48.
49./* To insert a new record into the file */
50.void insert(char *a)
51.{
52.    FILE *fp1;
53.    emp *temp1 = (emp *)malloc(sizeof(emp));
54.    temp1->name = (char *)malloc(200 * sizeof(char)); //allocating memory for
pointer
55.
56.    fp1 = fopen(a, "a+");
57.    if (fp1 == NULL)
58.        perror("");
59.    else
60.    {
61.        printf("Enter the employee id\n");
62.        scanf("%d", &temp1->empid);

```

```
63.     fwrite(&temp1->empid, sizeof(int), 1, fp1);
64.     printf("Enter the employee name\n");
65.     scanf(" %[^\n]s", temp1->name);
66.     fwrite(temp1->name, 200, 1, fp1);
67.     count++;
68. }
69. fclose(fp1);
70. free(temp1);
71. free(temp1->name);
72. }
73.
74. /* To display the records in the file */
75. void display(char *a)
76. {
77.     FILE *fp1;
78.     char ch;
79.     int var = count;
80.     emp *temp = (emp *)malloc(sizeof(emp));
81.     temp->name = (char *)malloc(200*sizeof(char));
82.
83.     fp1 = fopen(a, "r");
84.     if (count == 0)
85.     {
86.         printf("no records to display\n");
87.         return;
88.     }
89.     if (fp1 == NULL)
90.         perror("");
91.     else
92.     {
93.         while(var) // display the employee records
94.         {
95.             fread(&temp->empid, sizeof(int), 1, fp1);
96.             printf("%d", temp->empid);
97.             fread(temp->name, 200, 1, fp1);
98.             printf(" %s\n", temp->name);
99.             var--;
100.            }
101.        }
102.    fclose(fp1);
103.    free(temp);
104.    free(temp->name);
105. }
```

```

106.
107.     /* To Update the given record */
108.     void update(char *a)
109.     {
110.         FILE *fp1;
111.         char ch, name[200];
112.         int var = count, id, c;
113.         emp *temp = (emp *)malloc(sizeof(emp));
114.         temp->name = (char *)malloc(200*sizeof(char));
115.
116.         fp1 = fopen(a, "r+");
117.         if (fp1 == NULL)
118.             perror("");
119.         else
120.         {
121.             while (var)      //displaying employee records so that user enter
correct employee id
122.             {
123.                 fread(&temp->empid, sizeof(int), 1, fp1);
124.                 printf("%d", temp->empid);
125.                 fread(temp->name, 200, 1, fp1);
126.                 printf(" %s\n", temp->name);
127.                 var--;
128.             }
129.             printf("enter which employee id to be updated\n");
130.             scanf("%d", &id);
131.             fseek(fp1, 0, 0);
132.             var = count;
133.             while(var)    //Loop to update the name of entered employeeid
134.             {
135.                 fread(&temp->empid, sizeof(int), 1, fp1);
136.                 if (id == temp->empid)
137.                 {
138.                     printf("enter employee name for update:");
139.                     scanf(" %[^\n]s", name);
140.                     c = fwrite(name, 200, 1, fp1);
141.                     break;
142.                 }
143.                 fread(temp->name, 200, 1, fp1);
144.                 var--;
145.             }
146.             if (c == 1)
147.                 printf("update of the record successfully\n");

```

```

148.         else
149.             printf("update unsuccesful enter correct id\n");
150.             fclose(fp1);
151.             free(temp);
152.             free(temp->name);
153.         }
154.     }

```

```

1. /*
2. * C Program to Join Lines of Two given Files and
3. * Store them in a New file
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. /* Function Prototype */
9. int joinfiles(FILE *, FILE *, FILE *);
10.
11. char ch;
12. int flag;
13.
14. void main(int argc, char *argv[])
15. {
16.     FILE *file1, *file2, *target;
17.
18.     file1 = fopen(argv[1], "r");
19.     if (file1 == NULL)
20.     {
21.         perror("Error Occured!");
22.     }
23.     file2 = fopen(argv[2], "r");
24.     if (file2 == NULL)
25.     {
26.         perror("Error Occured!");
27.     }
28.     target = fopen(argv[3], "a");
29.     if (target == NULL)
30.     {
31.         perror("Error Occured!");
32.     }
33.

```

```
34.     joinfiles(file1, file2, target);           /* Calling Function */
35.
36.     if (flag == 1)
37.     {
38.         printf("The files have been successfully concatenated\n");
39.     }
40. }
41.
42./* Code join the two given files line by line into a new file */
43.
44. int joinfiles(FILE *file1, FILE *file2, FILE *target)
45. {
46.     while ((fgetc(file1) != EOF) || (fgetc(file2) != EOF))
47.     {
48.         fseek(file1, -1, 1);
49.         while ((ch = fgetc(file1)) != '\n')
50.         {
51.             if (ch == EOF)
52.             {
53.                 break;
54.             }
55.             else
56.             {
57.                 fputc(ch, target);
58.             }
59.         }
60.         while ((ch = fgetc(file2)) != '\n')
61.         {
62.             if (ch == EOF)
63.             {
64.                 break;
65.             }
66.             else
67.             {
68.                 fputc(ch, target);
69.             }
70.         }
71.         fputc('\n', target);
72.     }
73.     fclose(file1);
74.     fclose(file2);
75.     fclose(target);
76.     return flag = 1;
```

77. }

```
1. /*
2. * C Program to Collect Statistics of a Source File Like Total Lines,
3. * Total no. of Blank Lines, Total no. of Lines ending with Semicolon
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void main(int argc, char *argv[]) /* Command Line Arguments */
9. {
10.     int ncount = 0, ccount = 0, scount = 0, blank = 0;
11.     char ch;
12.     FILE *fp;
13.     fp = fopen(argv[1], "r");
14.     if (fp == NULL)
15.     {
16.         perror("Error Occured");
17.     }
18.     else
19.     {
20.         while(1)
21.         {
22.             ch = fgetc(fp);
23.             if (ch == EOF)
24.             {
25.                 break;
26.             }
27.             if (ch == 10)
28.             {
29.                 ncount++;
30.                 if (ch = fgetc(fp) == '\n')
31.                 {
32.                     fseek(fp, -1, 1); /* shifting offset of the file to
   previous position */
33.                     blank++;
34.                 }
35.             }
36.             else if (ch == 59)
37.             {
38.                 scount++;
```

```

39.     }
40.     else if (ch == '/' || ch == '*')
41.     {
42.         ccount++;
43.     }
44. }
45. }
46. printf("\nThe Total number of lines are %d", ncount);
47. printf("\nThe Total number of Commented lines are %d", ccount);
48. printf("\nThe Total number of blank lines are %d", blank);
49. printf("\nThe total number of lines that end with Semicolon %d", scount);
50. printf("\nThe length of Actual code is %d ", ncount-blank-ccount);
51. fclose(fp);
52. }
```

## (11. )C Programming Examples on Mathematical Functions

### 1. C Examples on Mathematical Calculations

```

1. /*
2. * C program to find the simple interest, given principal,
3. * rate of interest and time.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     float principal_amt, rate, simple_interest;
10.    int time;
11.
12.    printf("Enter the values of principal_amt, rate and time \n");
13.    scanf("%f %f %d", &principal_amt, &rate, &time);
14.    simple_interest = (principal_amt * rate * time) / 100.0;
15.    printf("Amount = Rs. %.2f\n", principal_amt);
16.    printf("Rate = Rs. %.2f%\n", rate);
17.    printf("Time = %d years\n", time);
18.    printf("Simple interest = %.2f\n", simple_interest);
19. }
```

```

1. /*
2.  * C program to find out the roots of a quadratic equation
3.  * for non-zero coefficients. In case of errors the program
4.  * should report suitable error message.
5. */
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <math.h>
9.
10. void main()
11. {
12.     float a, b, c, root1, root2;
13.     float realp, imagp, disc;
14.
15.     printf("Enter the values of a, b and c \n");
16.     scanf("%f %f %f", &a, &b, &c);
17.     /* If a = 0, it is not a quadratic equation */
18.     if (a == 0 || b == 0 || c == 0)
19.     {
20.         printf("Error: Roots cannot be determined \n");
21.         exit(1);
22.     }
23.     else
24.     {
25.         disc = b * b - 4.0 * a * c;
26.         if (disc < 0)
27.         {
28.             printf("Imaginary Roots\n");
29.             realp = -b / (2.0 * a) ;
30.             imagp = sqrt(abs(disc)) / (2.0 * a);
31.             printf("Root1 = %f +i %f\n", realp, imagp);
32.             printf("Root2 = %f -i %f\n", realp, imagp);
33.         }
34.         else if (disc == 0)
35.         {
36.             printf("Roots are real and equal\n");
37.             root1 = -b / (2.0 * a);
38.             root2 = root1;
39.             printf("Root1 = %f\n", root1);
40.             printf("Root2 = %f\n", root2);
41.         }
42.         else if (disc > 0 )
43.         {

```

```

44.         printf("Roots are real and distinct \n");
45.         root1 =(-b + sqrt(disc)) / (2.0 * a);
46.         root2 =(-b - sqrt(disc)) / (2.0 * a);
47.         printf("Root1 = %f \n", root1);
48.         printf("Root2 = %f \n", root2);
49.     }
50. }
51. }
```

```

1. /*
2.  * C program to find out the roots of a quadratic equation
3.  * for non-zero coefficients. In case of errors the program
4.  * should report suitable error message.
5. */
6. #include <stdio.h>
7. #include <stdlib.h>
8. #include <math.h>
9.
10. void main()
11. {
12.     float a, b, c, root1, root2;
13.     float realp, imagp, disc;
14.
15.     printf("Enter the values of a, b and c \n");
16.     scanf("%f %f %f", &a, &b, &c);
17.     /* If a = 0, it is not a quadratic equation */
18.     if (a == 0 || b == 0 || c == 0)
19.     {
20.         printf("Error: Roots cannot be determined \n");
21.         exit(1);
22.     }
23.     else
24.     {
25.         disc = b * b - 4.0 * a * c;
26.         if (disc < 0)
27.         {
28.             printf("Imaginary Roots\n");
29.             realp = -b / (2.0 * a) ;
30.             imagp = sqrt(abs(disc)) / (2.0 * a);
31.             printf("Root1 = %f +i %f\n", realp, imagp);
32.             printf("Root2 = %f -i %f\n", realp, imagp);
33.         }
34.     }
35. }
```

```

34.     else if (disc == 0)
35.     {
36.         printf("Roots are real and equal\n");
37.         root1 = -b / (2.0 * a);
38.         root2 = root1;
39.         printf("Root1 = %f\n", root1);
40.         printf("Root2 = %f\n", root2);
41.     }
42.     else if (disc > 0 )
43.     {
44.         printf("Roots are real and distinct \n");
45.         root1 =(-b + sqrt(disc)) / (2.0 * a);
46.         root2 =(-b - sqrt(disc)) / (2.0 * a);
47.         printf("Root1 = %f \n", root1);
48.         printf("Root2 = %f \n", root2);
49.     }
50. }
51. }
```

## 2. C Examples on Fibonacci Numbers

```

1. /*
2.  * C Program to find the nth number in Fibonacci series using recursion
3. */
4. #include <stdio.h>
5. int fibo(int);
6.
7. int main()
8. {
9.     int num;
10.    int result;
11.
12.    printf("Enter the nth number in fibonacci series: ");
13.    scanf("%d", &num);
14.    if (num < 0)
15.    {
16.        printf("Fibonacci of negative number is not possible.\n");
17.    }
18.    else
19.    {
20.        result = fibo(num);
21.        printf("The %d number in fibonacci series is %d\n", num, result);
```

```

22.     }
23.     return 0;
24. }
25. int fibo(int num)
26. {
27.     if (num == 0)
28.     {
29.         return 0;
30.     }
31.     else if (num == 1)
32.     {
33.         return 1;
34.     }
35.     else
36.     {
37.         return(fibo(num - 1) + fibo(num - 2));
38.     }
39. }
```

```

1. /*
2. * C Program to find the nth number in Fibonacci series using recursion
3. */
4. #include <stdio.h>
5. int fibo(int);
6.
7. int main()
8. {
9.     int num;
10.    int result;
11.
12.    printf("Enter the nth number in fibonacci series: ");
13.    scanf("%d", &num);
14.    if (num < 0)
15.    {
16.        printf("Fibonacci of negative number is not possible.\n");
17.    }
18.    else
19.    {
20.        result = fibo(num);
21.        printf("The %d number in fibonacci series is %d\n", num, result);
22.    }
23.    return 0;
```

```

24. }
25. int fibo(int num)
26. {
27.     if (num == 0)
28.     {
29.         return 0;
30.     }
31.     else if (num == 1)
32.     {
33.         return 1;
34.     }
35.     else
36.     {
37.         return(fibo(num - 1) + fibo(num - 2));
38.     }
39. }
```

```

1. /*
2. * C program to generate and print first N FIBONACCI numbers
3. * in the series.
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int fib1 = 0, fib2 = 1, fib3, num, count = 0;
10.
11.    printf("Enter the value of num \n");
12.    scanf("%d", &num);
13.    printf("First %d FIBONACCI numbers are ... \n", num);
14.    printf("%d\n", fib1);
15.    printf("%d\n", fib2);
16.    count = 2; /* fib1 and fib2 are already used */
17.    while (count < num)
18.    {
19.        fib3 = fib1 + fib2;
20.        count++;
21.        printf("%d\n", fib3);
22.        fib1 = fib2;
23.        fib2 = fib3;
24.    }
25. }
```

### 3. C Examples on GCD and LCM of Numbers

```

1. /*
2.  * C program to find the GCD and LCM of two integers using Euclid's algorithm
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int num1, num2, gcd, lcm, remainder, numerator, denominator;
9.
10.    printf("Enter two numbers\n");
11.    scanf("%d %d", &num1, &num2);
12.    if (num1 > num2)
13.    {
14.        numerator = num1;
15.        denominator = num2;
16.    }
17.    else
18.    {
19.        numerator = num2;
20.        denominator = num1;
21.    }
22.    remainder = num1 % num2;
23.    while (remainder != 0)
24.    {
25.        numerator = denominator;
26.        denominator = remainder;
27.        remainder = numerator % denominator;
28.    }
29.    gcd = denominator;
30.    lcm = num1 * num2 / gcd;
31.    printf("GCD of %d and %d = %d\n", num1, num2, gcd);
32.    printf("LCM of %d and %d = %d\n", num1, num2, lcm);
33. }
```

```

1. /*
2.  * C Program to find HCF of a given Number using Recursion
3. */
4. #include <stdio.h>
```

```

5.
6. int hcf(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their HCF: ");
13.     scanf("%d%d", &a, &b);
14.     result = hcf(a, b);
15.     printf("The HCF of %d and %d is %d.\n", a, b, result);
16. }
17.
18. int hcf(int a, int b)
19. {
20.     while (a != b)
21.     {
22.         if (a > b)
23.         {
24.             return hcf(a - b, b);
25.         }
26.         else
27.         {
28.             return hcf(a, b - a);
29.         }
30.     }
31.     return a;
32. }
```

```

1. /*
2.  * C Program to Find LCM of a Number using Recursion
3.  */
4. #include <stdio.h>
5.
6. int lcm(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.     int prime[100];
12.
13.     printf("Enter two numbers: ");
```

```

14.     scanf("%d%d", &a, &b);
15.     result = lcm(a, b);
16.     printf("The LCM of %d and %d is %d\n", a, b, result);
17.     return 0;
18. }
19.
20.int lcm(int a, int b)
21.{ 
22.    static int common = 1;
23.
24.    if (common % a == 0 && common % b == 0)
25.    {
26.        return common;
27.    }
28.    common++;
29.    lcm(a, b);
30.    return common;
31. }
```

```

1. /*
2.  * C Program to find GCD of given Numbers using Recursion
3.  */
4. #include <stdio.h>
5.
6. int gcd(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their GCD: ");
13.     scanf("%d%d", &a, &b);
14.     result = gcd(a, b);
15.     printf("The GCD of %d and %d is %d.\n", a, b, result);
16. }
17.
18.int gcd(int a, int b)
19.{
20.    while (a != b)
21.    {
22.        if (a > b)
23.        {
```

```

24.         return gcd(a - b, b);
25.     }
26.     else
27.     {
28.         return gcd(a, b - a);
29.     }
30. }
31. return a;
32. }}

```

```

1. /*
2. * C Program to find HCF of a given Number without using Recursion
3. */
4. #include <stdio.h>
5.
6. int hcf(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their HCF: ");
13.     scanf("%d%d", &a, &b);
14.     result = hcf(a, b);
15.     printf("The HCF of %d and %d is %.1f.\n", a, b, result);
16.
17.     return 0;
18. }
19.
20. int hcf(int a, int b)
21. {
22.     while (a != b)
23.     {
24.         if (a > b)
25.         {
26.             a = a - b;
27.         }
28.         else
29.         {
30.             b = b - a;
31.         }
32.     }

```

```
33.     return a;
34. }
```

```
1. /*
2.  * C program to find the value of sin(x) using the series
3.  * up to the given accuracy (without using user defined function)
4.  * also print sin(x) using Library function.
5. */
6. #include <stdio.h>
7. #include <math.h>
8. #include <stdlib.h>
9.
10. void main()
11. {
12.     int n, x1;
13.     float accuracy, term, denominator, x, sinx, sinval;
14.
15.     printf("Enter the value of x (in degrees) \n");
16.     scanf("%f", &x);
17.     x1 = x;
18.     /* Converting degrees to radians */
19.     x = x * (3.142 / 180.0);
20.     sinval = sin(x);
21.     printf("Enter the accuracy for the result \n");
22.     scanf("%f", &accuracy);
23.     term = x;
24.     sinx = term;
25.     n = 1;
26.     do
27.     {
28.         denominator = 2 * n * (2 * n + 1);
29.         term = -term * x * x / denominator;
30.         sinx = sinx + term;
31.         n = n + 1;
32.     } while (accuracy <= fabs(sinval - sinx));
33.     printf("Sum of the sine series = %f \n", sinx);
34.     printf("Using Library function sin(%d) = %f\n", x1, sin(x));
35. }
```

```
1. /*
2.  * C program to find the value of cos(x) using the series
```

```

3.  * up to the given accuracy (without using user defined function)
4.  * also print cos(x) using library function.
5.  */
6. #include <stdio.h>
7. #include <math.h>
8. #include <stdlib.h>
9.
10. void main()
11. {
12.     int n, x1;
13.     float accuracy, term, denominator, x, cosx, cosval;
14.
15.     printf("Enter the value of x (in degrees) \n");
16.     scanf("%f", &x);
17.     x1 = x;
18.     /* Converting degrees to radians */
19.     x = x * (3.142 / 180.0);
20.     cosval = cos(x);
21.     printf("Enter the accuracy for the result \n");
22.     scanf("%f", &accuracy);
23.     term = 1;
24.     cosx = term;
25.     n = 1;
26.     do
27.     {
28.         denominator = 2 * n * (2 * n - 1);
29.         term = -term * x * x / denominator;
30.         cosx = cosx + term;
31.         n = n + 1;
32.     } while (accuracy <= fabs(cosval - cosx));
33.     printf("Sum of the cosine series = %f\n", cosx);
34.     printf("Using Library function cos(%d) = %f\n", x1, cos(x));
35. }
```

```

1. /*
2.  * C program to find the sum of cos(x) series
3.  */
4. #include <stdio.h>
5. #include <math.h>
6.
7. void main()
8. {
```

```

9.     int n, x1, i, j;
10.    float x, sign, cosx, fact;
11.
12.    printf("Enter the number of the terms in a series\n");
13.    scanf("%d", &n);
14.    printf("Enter the value of x(in degrees)\n");
15.    scanf("%f", &x);
16.    x1 = x;
17.    /* Degrees to radians */
18.    x = x * (3.142 / 180.0);
19.    cosx = 1;
20.    sign = -1;
21.    for (i = 2; i <= n; i = i + 2)
22.    {
23.        fact = 1;
24.        for (j = 1; j <= i; j++)
25.        {
26.            fact = fact * j;
27.        }
28.        cosx = cosx + (pow(x, i) / fact) * sign;
29.        sign = sign * (-1);
30.    }
31.    printf("Sum of the cosine series = %7.2f\n", cosx);
32.    printf("The value of cos(%d) using library function = %f\n", x1,
33.           cos(x));
34. }
```

#### 4. C Examples on Statistical Properties

```

1. /*
2.  * C program to find the sum of 'N' natural numbers
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int i, num, sum = 0;
9.
10.    printf("Enter an integer number \n");
11.    scanf ("%d", &num);
12.    for (i = 1; i <= num; i++)
13.    {
```

```

14.         sum = sum + i;
15.     }
16.     printf ("Sum of first %d natural numbers = %d\n", num, sum);
17. }
```

```

1. /*
2. * C program to check whether a given number is prime or not
3. * and output the given number with suitable message.
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void main()
9. {
10.     int num, j, flag;
11.
12.     printf("Enter a number \n");
13.     scanf("%d", &num);
14.
15.     if (num <= 1)
16.     {
17.         printf("%d is not a prime numbers \n", num);
18.         exit(1);
19.     }
20.     flag = 0;
21.     for (j = 2; j <= num / 2; j++)
22.     {
23.         if ((num % j) == 0)
24.         {
25.             flag = 1;
26.             break;
27.         }
28.     }
29.     if (flag == 0)
30.         printf("%d is a prime number \n", num);
31.     else
32.         printf("%d is not a prime number \n", num);
33. }
```

```

1. /*
2. * C program to find prime numbers in a given range.
```

```
3. * Also print the number of prime numbers.
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void main()
9. {
10.     int num1, num2, i, j, flag, temp, count = 0;
11.
12.     printf("Enter the value of num1 and num2 \n");
13.     scanf("%d %d", &num1, &num2);
14.     if (num2 < 2)
15.     {
16.         printf("There are no primes upto %d\n", num2);
17.         exit(0);
18.     }
19.     printf("Prime numbers are \n");
20.     temp = num1;
21.     if ( num1 % 2 == 0)
22.     {
23.         num1++;
24.     }
25.     for (i = num1; i <= num2; i = i + 2)
26.     {
27.         flag = 0;
28.         for (j = 2; j <= i / 2; j++)
29.         {
30.             if ((i % j) == 0)
31.             {
32.                 flag = 1;
33.                 break;
34.             }
35.         }
36.         if (flag == 0)
37.         {
38.             printf("%d\n", i);
39.             count++;
40.         }
41.     }
42.     printf("Number of primes between %d & %d = %d\n", temp, num2, count);
43. }
```

```
1. /*
2.  * C program to find prime numbers in a given range.
3.  * Also print the number of prime numbers.
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void main()
9. {
10.     int num1, num2, i, j, flag, temp, count = 0;
11.
12.     printf("Enter the value of num1 and num2 \n");
13.     scanf("%d %d", &num1, &num2);
14.     if (num2 < 2)
15.     {
16.         printf("There are no primes upto %d\n", num2);
17.         exit(0);
18.     }
19.     printf("Prime numbers are \n");
20.     temp = num1;
21.     if ( num1 % 2 == 0)
22.     {
23.         num1++;
24.     }
25.     for (i = num1; i <= num2; i = i + 2)
26.     {
27.         flag = 0;
28.         for (j = 2; j <= i / 2; j++)
29.         {
30.             if ((i % j) == 0)
31.             {
32.                 flag = 1;
33.                 break;
34.             }
35.         }
36.         if (flag == 0)
37.         {
38.             printf("%d\n", i);
39.             count++;
40.         }
41.     }
42.     printf("Number of primes between %d & %d = %d\n", temp, num2, count);
43. }
```

```
1. /*
2.  * C program to input real numbers and find the mean, variance
3.  * and standard deviation
4. */
5. #include <stdio.h>
6. #include <math.h>
7. #define MAXSIZE 10
8.
9. void main()
10. {
11.     float x[MAXSIZE];
12.     int i, n;
13.     float average, variance, std_deviation, sum = 0, sum1 = 0;
14.
15.     printf("Enter the value of N \n");
16.     scanf("%d", &n);
17.     printf("Enter %d real numbers \n", n);
18.     for (i = 0; i < n; i++)
19.     {
20.         scanf("%f", &x[i]);
21.     }
22.     /* Compute the sum of all elements */
23.     for (i = 0; i < n; i++)
24.     {
25.         sum = sum + x[i];
26.     }
27.     average = sum / (float)n;
28.     /* Compute variance and standard deviation */
29.     for (i = 0; i < n; i++)
30.     {
31.         sum1 = sum1 + pow((x[i] - average), 2);
32.     }
33.     variance = sum1 / (float)n;
34.     std_deviation = sqrt(variance);
35.     printf("Average of all elements = %.2f\n", average);
36.     printf("variance of all elements = %.2f\n", variance);
37.     printf("Standard deviation = %.2f\n", std_deviation);
38. }
```

```
1. /*
```

```

2.  * C program to evaluate a given polynomial by reading its coefficients
3.  * in an array.
4.  *  $P(x) = A_n X^n + A_{n-1} X^{n-1} + A_{n-2} X^{n-2} + \dots + A_1 X + A_0$ 
5.  *
6.  * The polynomial can be written as:
7.  *  $P(x) = A_0 + X(A_1 + X(A_2 + X(A_3 + X(Q_4 + X(\dots X(A_{n-1} + X A_n))))))$ 
8.  * and evaluated starting from the inner loop
9.  */
10. #include <stdio.h>
11. #include <stdlib.h>
12. #define MAXSIZE 10
13.
14. void main()
15. {
16.     int array[MAXSIZE];
17.     int i, num, power;
18.     float x, polySum;
19.
20.     printf("Enter the order of the polynomial \n");
21.     scanf("%d", &num);
22.     printf("Enter the value of x \n");
23.     scanf("%f", &x);
24.     /* Read the coefficients into an array */
25.     printf("Enter %d coefficients \n", num + 1);
26.     for (i = 0; i <= num; i++)
27.     {
28.         scanf("%d", &array[i]);
29.     }
30.     polySum = array[0];
31.     for (i = 1; i <= num; i++)
32.     {
33.         polySum = polySum * x + array[i];
34.     }
35.     power = num;
36.
37.     printf("Given polynomial is: \n");
38.     for (i = 0; i <= num; i++)
39.     {
40.         if (power < 0)
41.         {
42.             break;
43.         }
44.         /* printing proper polynomial function */

```

```

45.     if (array[i] > 0)
46.         printf(" + ");
47.     else if (array[i] < 0)
48.         printf(" - ");
49.     else
50.         printf(" ");
51.     printf("%dx^%d ", abs(array[i]), power--);
52. }
53. printf("\n Sum of the polynomial = %6.2f \n", polySum);
54. }
```

```

1. /*
2. * C program to accept a coordinate point in a XY coordinate system
3. * and determine its quadrant
4. */
5. #include <stdio.h>
6.
7. void main()
8. {
9.     int x, y;
10.
11.    printf("Enter the values for X and Y\n");
12.    scanf("%d %d", &x, &y);
13.    if (x > 0 && y > 0)
14.        printf("point (%d, %d) lies in the First quadrant\n");
15.    else if (x < 0 && y > 0)
16.        printf("point (%d, %d) lies in the Second quadrant\n");
17.    else if (x < 0 && y < 0)
18.        printf("point (%d, %d) lies in the Third quadrant\n");
19.    else if (x > 0 && y < 0)
20.        printf("point (%d, %d) lies in the Fourth quadrant\n");
21.    else if (x == 0 && y == 0)
22.        printf("point (%d, %d) lies at the origin\n");
23. }
```

## 5. C Examples on Power and Factorial of a Number

```

1. /*
2.  * C Program to find Power of a Number using Recursion
3. */
4. #include <stdio.h>
5.
6. long power (int, int);
7.
8. int main()
9. {
10.     int pow, num;
11.     long result;
12.
13.     printf("Enter a number: ");
14.     scanf("%d", &num);
15.     printf("Enter it's power: ");
16.     scanf("%d", &pow);
17.     result = power(num, pow);
18.     printf("%d^%d is %ld", num, pow, result);
19.     return 0;
20. }
21.
22.long power (int num, int pow)
23.{
24.    if (pow)
25.    {
26.        return (num * power(num, pow - 1));
27.    }
28.    return 1;
29. }
```

```

1. /*
2.  * C Program to find factorial of a given number using recursion
3. */
4. #include <stdio.h>
5.
6. int factorial(int);
7.
8. int main()
9. {
10.     int num;
11.     int result;
12.
```

```

13.     printf("Enter a number to find it's Factorial: ");
14.     scanf("%d", &num);
15.     if (num < 0)
16.     {
17.         printf("Factorial of negative number not possible\n");
18.     }
19.     else
20.     {
21.         result = factorial(num);
22.         printf("The Factorial of %d is %d.\n", num, result);
23.     }
24.     return 0;
25. }
26. int factorial(int num)
27. {
28.     if (num == 0 || num == 1)
29.     {
30.         return 1;
31.     }
32.     else
33.     {
34.         return(num * factorial(num - 1));
35.     }
36. }
```

```

1. /*
2.  * C program to compute the value of X ^ N given X and N as inputs
3.  */
4. #include <stdio.h>
5. #include <math.h>
6.
7. long int power(int x, int n);
8.
9. void main()
10. {
11.     long int x, n, xpown;
12.
13.     printf("Enter the values of X and N \n");
14.     scanf("%ld %ld", &x, &n);
15.     xpown = power(x, n);
16.     printf("X to the power N = %ld\n", xpown);
17. }
```

```

18./* Recursive function to computer the X to power N */
19.long int power(int x, int n)
20.{
21.    if (n == 1)
22.        return(x);
23.    else if (n % 2 == 0)
24.        /* if n is even */
25.        return (pow(power(x, n/2), 2));
26.    else
27.        /* if n is odd */
28.        return (x * power(x, n - 1));
29.}
```

```

1. /*
2.  * C program to find the factorial of a given number
3. */
4.
5. #include <stdio.h>
6. void main()
7. {
8.     int i, fact = 1, num;
9.
10.    printf("Enter the number \n");
11.    scanf("%d", &num);
12.    if (num <= 0)
13.        fact = 1;
14.    else
15.    {
16.        for (i = 1; i <= num; i++)
17.        {
18.            fact = fact * i;
19.        }
20.    }
21.    printf("Factorial of %d = %5d\n", num, fact);
22.}
```

```

1. /*
2.  * C program to Calculate the value of nCr
3. */
4. #include <stdio.h>
5.
```

```

6. int fact(int z);
7.
8. void main()
9. {
10.     int n, r, ncr;
11.
12.     printf("\n Enter the value for N and R \n");
13.     scanf("%d%d", &n, &r);
14.     ncr = fact(n) / (fact(r) * fact(n - r));
15.     printf("\n The value of ncr is: %d", ncr);
16. }
17.
18.int fact(int z)
19.{
20.    int f = 1, i;
21.    if (z == 0)
22.    {
23.        return(f);
24.    }
25.    else
26.    {
27.        for (i = 1; i <= z; i++)
28.        {
29.            f = f * i;
30.        }
31.    }
32.    return(f);
33.}
```

```

1. /*
2.  * C Program to Find & Display Multiplication table
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int number, i = 1;
9.
10.    printf(" Enter the Number:");
11.    scanf("%d", &number);
12.    printf("Multiplication table of %d:\n ", number);
13.    printf("-----\n");
```

```

14.     while (i <= 10)
15.     {
16.         printf(" %d x %d = %d \n ", number, i, number * i);
17.         i++;
18.     }
19.     return 0;
20. }
```

## 6. C Examples on Summation of a Series

```

1. /*
2.  * C Program to find the sum of series 1^2 + 2^2 + .... + n^2.
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     int number, i;
9.     int sum = 0;
10.
11.    printf("Enter maximum values of series number: ");
12.    scanf("%d", &number);
13.    sum = (number * (number + 1) * (2 * number + 1)) / 6;
14.    printf("Sum of the above given series : ");
15.    for (i = 1; i <= number; i++)
16.    {
17.        if (i != number)
18.            printf("%d^2 + ", i);
19.        else
20.            printf("%d^2 = %d ", i, sum);
21.    }
22.    return 0;
23. }
```

```

1. /*
2.  * C Program to find the Sum of Series 1 + 1/2 + 1/3 + 1/4 + ... + 1/N
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     double number, sum = 0, i;
```

```

9.
10.    printf("\n enter the number ");
11.    scanf("%lf", &number);
12.    for (i = 1; i <= number; i++)
13.    {
14.        sum = sum + (1 / i);
15.        if (i == 1)
16.            printf("\n 1 +");
17.        else if (i == number)
18.            printf(" (1 / %lf)", i);
19.        else
20.            printf(" (1 / %lf) + ", i);
21.    }
22.    printf("\n The sum of the given series is %.2lf", sum);
23. }
```

```

1. /*
2. * C Program to Find find Sum of the Series 1/1! + 2/2! + 3/3! + .....1/N!
3. */
4. #include <stdio.h>
5.
6. double sumseries(double);
7.
8. main()
9. {
10.     double number,sum;
11.     printf("\n Enter the value:  ");
12.     scanf("%lf", &number);
13.     sum = sumseries(number);
14.     printf("\n Sum of the above series = %lf ", sum);
15. }
16.
17. double sumseries(double m)
18. {
19.     double sum2 = 0, f = 1, i;
20.     for (i = 1; i <= m; i++)
21.     {
22.         f = f * i;
23.         sum2 = sum2 +(i / f);
24.     }
25.     return(sum2);
26. }
```

```

1. /*
2. * C Program to Find the Sum of A.P Series
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main()
8. {
9.     int a, d, n, i, tn;
10.    int sum = 0;
11.
12.    printf("Enter the first term value of the A.P. series: ");
13.    scanf("%d", &a);
14.    printf("Enter the total numbers in the A.P. series: ");
15.    scanf("%d", &n);
16.    printf("Enter the common difference of A.P. series: ");
17.    scanf("%d", &d);
18.    sum = (n * (2 * a + (n - 1)* d ))/ 2;
19.    tn = a + (n - 1) * d;
20.    printf("Sum of the A.P series is: ");
21.    for (i = a; i <= tn; i = i + d )
22.    {
23.        if (i != tn)
24.            printf("%d + ", i);
25.        else
26.            printf("%d = %d ", i, sum);
27.    }
28.    return 0;
29. }
```

```

1. /*
2. * C Program to Find the Sum of G.P Series
3. */
4. #include <stdio.h>
5. #include <math.h>
```

```

6.
7. int main()
8. {
9.     float a, r, i, last_term, sum = 0;
10.    int n;
11.
12.    printf("Enter the first term of the G.P. series: ");
13.    scanf("%f", &a);
14.    printf("Enter the total numbers in the G.P. series: ");
15.    scanf("%d", &n);
16.    printf("Enter the common ratio of G.P. series: ");
17.    scanf("%f", &r);
18.    sum = (a * (1 - pow(r, n + 1))) / (1 - r);
19.    last_term = a * (1 - pow(r, n - 1));
20.    printf("last_term term of G.P.: %f", last_term);
21.    printf("\n Sum of the G.P.: %f", sum);
22.    return 0;
23. }
```

```

1. /*
2.  * C Program to Find the Sum of H.P Series
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int n;
9.     float i, sum, term;
10.
11.    printf("1 + 1 / 2 + 1 / 3 +.....+1 / n \n");
12.    printf("Enter the value of n \n");
13.    scanf("%d", &n);
14.    sum = 0;
15.    for (i = 1; i <= n; i++)
16.    {
17.        term = 1 / i;
18.        sum = sum + term;
19.    }
20.    printf("the Sum of H.P Series is = %f", sum);
21. }
```

## 7. C Examples on finding the Area of Geometrical Figures

```

1. /*
2. * C program to find the area of a triangle, given three sides
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. void main()
8. {
9.     int s, a, b, c, area;
10.
11.    printf("Enter the values of a, b and c \n");
12.    scanf("%d %d %d", &a, &b, &c);
13.    /* compute s */
14.    s = (a + b + c) / 2;
15.    area = sqrt(s * (s - a) * (s - b) * (s - c));
16.    printf("Area of a triangle = %d \n", area);
17. }
```

```

1. /*
2. * C program to find the area of a circle, given the radius
3. */
4. #include <stdio.h>
5. #include <math.h>
6. #define PI 3.142
7.
8. void main()
9. {
10.    float radius, area;
11.
12.    printf("Enter the radius of a circle \n");
13.    scanf("%f", &radius);
14.    area = PI * pow(radius, 2);
15.    printf("Area of a circle = %5.2f\n", area);
16. }
```

```

1. /*
2. * C Program to Find Area of a Right Angled Triangle
3. */
4. #include <stdio.h>
5.
```

```

6. int main()
7. {
8.     float height, width;
9.     float area;
10.
11.    printf("Enter height and width of the given triangle:\n ");
12.    scanf("%f%f", &height, &width);
13.    area = 0.5 * height * width;
14.    printf("Area of right angled triangle is: %.3f\n", area);
15.    return 0;
16. }
```

```

1. /*
2.  * C Program to Find Area of Trapezium
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     float a, b, h;
9.     float area;
10.
11.    printf("Enter the value for two bases & height of the trapezium: \n");
12.    scanf("%f%f%f", &a, &b, &h);
13.    area = 0.5 * (a + b) * h ;
14.    printf("Area of the trapezium is: %.3f", area);
15.    return 0;
16. }
```

```

1. /*
2.  * C Program to Find Area of rhombus
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     float diagonal1, diagonal2;
9.     float area;
10.
11.    printf("Enter diagonals of the given rhombus: \n ");
12.    scanf("%f%f", &diagonal1, &diagonal2);
```

```

13.     area = 0.5 * diagonal1 * diagonal2;
14.     printf("Area of rhombus is: %.3f \n", area);
15.     return 0;
16. }
```

```

1. /*
2.  * C Program to Find Area of Parallelogram
3. */
4. #include <stdio.h>
5.
6. int main()
7. {
8.     float base, altitude;
9.     float area;
10.
11.    printf("Enter base and altitude of the given Parallelogram: \n ");
12.    scanf("%f%f", &base, &altitude);
13.    area = base * altitude;
14.    printf("Area of Parallelogram is: %.3f\n", area);
15.    return 0;
16. }
```

## 8. C Examples on finding the Volume and Surface Area of Geometrical Figures

```

1. /*
2.  * C program to compute the surface area and volume of a cube
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. void main()
8. {
9.     float side, surfacearea, volume;
10.
11.    printf("Enter the length of a side \n");
12.    scanf("%f", &side);
13.    surfacearea = 6.0 * side * side;
14.    volume = pow(side, 3);
15.    printf("Surface area = %.2f and Volume = %.2f \n", surfacearea,
```

```
16.     volume);  
17. }
```

```
1. /*  
2.  * C program to find the areas of different geometrical shapes such as  
3.  * circle, square, rectangle etc using switch statements.  
4. */  
5. #include <stdio.h>  
6.  
7. void main()  
8. {  
9.     int fig_code;  
10.    float side, base, length, breadth, height, area, radius;  
11.  
12.    printf("-----\n");  
13.    printf(" 1 --> Circle\n");  
14.    printf(" 2 --> Rectangle\n");  
15.    printf(" 3 --> Triangle\n");  
16.    printf(" 4 --> Square\n");  
17.    printf("-----\n");  
18.    printf("Enter the Figure code\n");  
19.    scanf("%d", &fig_code);  
20.    switch(fig_code)  
21.    {  
22.        case 1:  
23.            printf("Enter the radius\n");  
24.            scanf("%f", &radius);  
25.            area = 3.142 * radius * radius;  
26.            printf("Area of a circle = %f\n", area);  
27.            break;  
28.        case 2:  
29.            printf("Enter the breadth and length\n");  
30.            scanf("%f %f", &breadth, &length);  
31.            area = breadth * length;  
32.            printf("Area of a Reactangle = %f\n", area);  
33.            break;  
34.        case 3:  
35.            printf("Enter the base and height\n");  
36.            scanf("%f %f", &base, &height);  
37.            area = 0.5 * base * height;  
38.            printf("Area of a Triangle = %f\n", area);  
39.            break;
```

```

40.     case 4:
41.         printf("Enter the side\n");
42.         scanf("%f", &side);
43.         area = side * side;
44.         printf("Area of a Square=%f\n", area);
45.         break;
46.     default:
47.         printf("Error in figure code\n");
48.         break;
49.     }
50. }
```

```

1. /*
2.  * C Program to Find the Volume and Surface Area of cylinder
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main()
8. {
9.
10.    float radius, height;
11.    float surface_area, volume;
12.
13.    printf("Enter value for radius and height of a cylinder : \n");
14.    scanf("%f%f", &radius, &height);
15.    surface_area = 2 * (22 / 7) * radius * (radius + height);
16.    volume = (22 / 7) * radius * radius * height;
17.    printf("Surface area of cylinder is: %.3f", surface_area);
18.    printf("\n Volume of cylinder is : %.3f", volume);
19.    return 0;
20. }
```

```

1. /*
2.  * C Program to Find the Volume and Surface Area of Cuboids
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main()
8. {
```

```

9.     float width, length, height;
10.    float surfacearea, volume, space_diagonal;
11.
12.    printf("Enter value of width, length & height of the cuboids:\n");
13.    scanf("%f%f%f", &width, &length, &height);
14.    surfacearea = 2 *(width * length + length * height +
15.    height * width);
16.    volume = width * length * height;
17.    space_diagonal = sqrt(width * width + length * length +
18.    height * height);
19.    printf("Surface area of cuboids is: %.3f", surfacearea);
20.    printf("\n Volume of cuboids is : %.3f", volume);
21.    printf("\n Space diagonal of cuboids is : %.3f", space_diagonal);
22.    return 0;
23. }

24. /*
25.  * C Program to Find the volume and surface area of cone
26.  */
27.#include <stdio.h>
28.#include <math.h>
29.
30.int main()
31.{
32.
33.    float radius, height;
34.    float surface_area, volume;
35.
36.    printf("Enter value of radius and height of a cone :\n ");
37.    scanf("%f%f", &radius, &height);
38.    surface_area = (22 / 7) * radius * (radius + sqrt(radius * radius + height
* height));
39.    volume = (1.0/3) * (22 / 7) * radius * radius * height;
40.    printf("Surface area of cone is: %.3f", surface_area);
41.    printf("\n Volume of cone is : %.3f", volume);
42.    return 0;
43. }

```

```

1. /*
2.  * C Program to Find Volume and Surface Area of Sphere
3.  */
4. #include <stdio.h>
5. #include <math.h>

```

```
6.
7. int main()
8. {
9.
10.    float radius;
11.    float surface_area, volume;
12.
13.    printf("Enter radius of the sphere : \n");
14.    scanf("%f", &radius);
15.    surface_area = 4 * (22/7) * radius * radius;
16.    volume = (4.0/3) * (22/7) * radius * radius * radius;
17.    printf("Surface area of sphere is: %.3f", surface_area);
18.    printf("\n Volume of sphere is : %.3f", volume);
19.    return 0;
20. }

21./*
22. * C Program to Find the Perimeter of a Circle, Rectangle and Triangle
23. */
24.#include <stdio.h>
25.#include <math.h>
26.
27.int main()
28.{
29.    float radius, length, width, a, b, c, height;
30.    int n;
31.    float perimeter;
32.
33.    //Perimeter of rectangle
34.    printf("\n Perimeter of rectangle \n");
35.    printf("-----\n");
36.    printf("\n Enter width and length of the rectangle : ");
37.    scanf("%f%f", &width,& length);
38.    perimeter = 2 * (width + length);
39.    printf("Perimeter of rectangle is: %.3f", perimeter);
40.
41.    //Perimeter of triangle
42.    printf("\n Perimeter of triangle \n");
43.    printf("-----\n");
44.    printf("\n Enter the size of all sides of the triangle : ");
45.    scanf("%f%f%f", &a, &b, &c);
46.    perimeter = a + b + c;
47.    printf("Perimeter of triangle is: %.3f", perimeter);
48.
```

```

49. //Perimeter of circle
50. printf(" \n Perimeter of circle \n");
51. printf("-----\n");
52. printf("\n Enter the radius of the circle : ");
53. scanf("%f", &radius);
54. perimeter = 2 * (22 / 7) * radius;
55. printf("Perimeter of circle is: %.3f", perimeter);
56.
57. //Perimeter of equilateral triangle
58. printf(" \n Perimeter of equilateral triangle \n");
59. printf("-----\n");
60. printf("\n Enter any side of the equilateral triangle : ");
61. scanf("%f", &a);
62. perimeter = 3 * a;
63. printf("Perimeter of equilateral triangle is: %.3f", perimeter);
64.
65. //Perimeter of right angled triangle
66. printf(" \n Perimeter of right angled triangle \n");
67. printf("-----\n");
68. printf("\n Enter the width and height of the right angled triangle : ");
69. scanf("%f%f", &width, &height);
70. perimeter = width + height + sqrt(width * width + height * height);
71. printf("Perimeter of right angled triangle is: %.3f", perimeter);
72. return 0;
73. }

```

```

1. /*
2.  * C Program to Find the Perimeter of a Circle, Rectangle and Triangle
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int main()
8. {
9.     float radius, length, width, a, b, c, height;
10.    int n;
11.    float perimeter;
12.
13.    //Perimeter of rectangle
14.    printf(" \n Perimeter of rectangle \n");
15.    printf("-----\n");
16.    printf("\n Enter width and length of the rectangle : ");

```

```

17.     scanf("%f%f", &width, & length);
18.     perimeter = 2 * (width + length);
19.     printf("Perimeter of rectangle is: %.3f", perimeter);
20.
21. //Perimeter of triangle
22. printf("\n Perimeter of triangle \n");
23. printf("-----\n");
24. printf("\n Enter the size of all sides of the triangle : ");
25. scanf("%f%f%f", &a, &b, &c);
26. perimeter = a + b + c;
27. printf("Perimeter of triangle is: %.3f", perimeter);
28.
29. //Perimeter of circle
30. printf(" \n Perimeter of circle \n");
31. printf("-----\n");
32. printf("\n Enter the radius of the circle : ");
33. scanf("%f", &radius);
34. perimeter = 2 * (22 / 7) * radius;
35. printf("Perimeter of circle is: %.3f", perimeter);
36.
37. //Perimeter of equilateral triangle
38. printf(" \n Perimeter of equilateral triangle \n");
39. printf("-----\n");
40. printf("\n Enter any side of the equilateral triangle : ");
41. scanf("%f", &a);
42. perimeter = 3 * a;
43. printf("Perimeter of equilateral triangle is: %.3f", perimeter);
44.
45. //Perimeter of right angled triangle
46. printf(" \n Perimeter of right angled triangle \n");
47. printf("-----\n");
48. printf("\n Enter the width and height of the right angled triangle : ");
49. scanf("%f%f", &width, &height);
50. perimeter = width + height + sqrt(width * width + height * height);
51. printf("Perimeter of right angled triangle is: %.3f", perimeter);
52. return 0;
53. }

```

## 9. C Examples on implementation of Floyd and Pascal's Triangle

```

1. /*
2.  * C Program to Display Floyd's triangle

```

```

3. /*
4. #include <stdio.h>
5.
6. main( )
7. {
8.     int i, j, k = 1;
9.
10.    printf("floyds triangle is\n");
11.    for( i = 1; k <= 20; ++i )
12.    {
13.        for( j = 1; j <= i; ++j )
14.            printf( "%d ", k++ );
15.        printf( "\n\n" );
16.    }
17.    return 0;
18. }
```

```

1. /*
2. * C Program to Display Pascal triangle
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int array[15][15], i, j, rows, num = 25, k;
9.
10.    printf("\n enter the number of rows:");
11.    scanf("%d", &rows);
12.    for (i = 0; i < rows; i++)
13.    {
14.        for (k = num - 2 * i; k >= 0; k--)
15.            printf(" ");
16.        for (j = 0; j <= i; j++)
17.        {
18.            if (j == 0 || i == j)
19.            {
20.                array[i][j] = 1;
21.            }
22.            else
23.            {
24.                array[i][j] = array[i - 1][j - 1] + array[i - 1][j];
25.            }

```

```

26.         printf("%4d", array[i][j]);
27.     }
28.     printf("\n");
29. }
30.

```

```

1. /*
2. * C program to Calculate the Value of nPr
3. */
4. #include <stdio.h>
5.
6. void main(void)
7. {
8.     printf("%d\n", fact(8));
9.     int n, r;
10.    printf("Enter value for n and r\n");
11.    scanf("%d%d", &n, &r);
12.    int npr = fact(n) / fact(n - r);
13.    printf("\n Permutation values is = %d", npr);
14. }
15.
16. int fact(int x)
17. {
18.     if (x <= 1)
19.         return 1;
20.     return x * fact(x - 1);
21. }

```

## 12. C Programming Examples on Puzzles & Games

```

1. /*
2. * C program for Tower of Hanoi using Recursion
3. */
4. #include <stdio.h>
5.
6. void towers(int, char, char, char);
7.
8. int main()
9. {
10.     int num;
11.
12.     printf("Enter the number of disks : ");

```

```

13.     scanf("%d", &num);
14.     printf("The sequence of moves involved in the Tower of Hanoi are :\n");
15.     towers(num, 'A', 'C', 'B');
16.     return 0;
17. }
18. void towers(int num, char frompeg, char topeg, char auxpeg)
19. {
20.     if (num == 1)
21.     {
22.         printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
23.         return;
24.     }
25.     towers(num - 1, frompeg, auxpeg, topeg);
26.     printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
27.     towers(num - 1, auxpeg, topeg, frompeg);
28. }
```

```

1. /*
2.  * C Program to Solve the Magic Squares Puzzle without using
3.  * Recursion
4. */
5. #include <stdio.h>
6.
7. void magicsq(int, int [[10]]);
8.
9. int main( )
10. {
11.     int size;
12.     int a[10][10];
13.
14.     printf("Enter the size: ");
15.     scanf("%d", &size);
16.     if (size % 2 == 0)
17.     {
18.         printf("Magic square works for an odd numbered size\n");
19.     }
20.     else
21.     {
22.         magicsq(size, a);
23.     }
24.     return 0;
25. }
```

```
26.  
27. void magicsq(int size, int a[][][10])  
28. {  
29.     int sqr = size * size;  
30.     int i = 0, j = size / 2, k;  
31.  
32.     for (k = 1; k <= sqr; ++k)  
33.     {  
34.         a[i][j] = k;  
35.         i--;  
36.         j++;  
37.  
38.         if (k % size == 0)  
39.         {  
40.             i += 2;  
41.             --j;  
42.         }  
43.         else  
44.         {  
45.             if (j == size)  
46.             {  
47.                 j -= size;  
48.             }  
49.             else if (i < 0)  
50.             {  
51.                 i += size;  
52.             }  
53.         }  
54.     }  
55.     for (i = 0; i < size; i++)  
56.     {  
57.         for (j = 0; j < size; j++)  
58.         {  
59.             printf("%d ", a[i][j]);  
60.         }  
61.         printf("\n");  
62.     }  
63.     printf("\n");  
64. }
```

## (12)C Programming Examples on Puzzles &amp; Games

```

1. /*
2. * C program for Tower of Hanoi using Recursion
3. */
4. #include <stdio.h>
5.
6. void towers(int, char, char, char);
7.
8. int main()
9. {
10.     int num;
11.
12.     printf("Enter the number of disks : ");
13.     scanf("%d", &num);
14.     printf("The sequence of moves involved in the Tower of Hanoi are :\n");
15.     towers(num, 'A', 'C', 'B');
16.     return 0;
17. }
18. void towers(int num, char frompeg, char topeg, char auxpeg)
19. {
20.     if (num == 1)
21.     {
22.         printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
23.         return;
24.     }
25.     towers(num - 1, frompeg, auxpeg, topeg);
26.     printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
27.     towers(num - 1, auxpeg, topeg, frompeg);
28. }
```

```

1. /*
2. * C Program to Solve the Magic Squares Puzzle without using
3. * Recursion
4. */
5. #include <stdio.h>
6.
7. void magicsq(int, int [][10]);
8.
9. int main( )
10. {
```

```
11.     int size;
12.     int a[10][10];
13.
14.     printf("Enter the size: ");
15.     scanf("%d", &size);
16.     if (size % 2 == 0)
17.     {
18.         printf("Magic square works for an odd numbered size\n");
19.     }
20.     else
21.     {
22.         magicsq(size, a);
23.     }
24.     return 0;
25. }
26.
27. void magicsq(int size, int a[][10])
28. {
29.     int sqr = size * size;
30.     int i = 0, j = size / 2, k;
31.
32.     for (k = 1; k <= sqr; ++k)
33.     {
34.         a[i][j] = k;
35.         i--;
36.         j++;
37.
38.         if (k % size == 0)
39.         {
40.             i += 2;
41.             --j;
42.         }
43.         else
44.         {
45.             if (j == size)
46.             {
47.                 j -= size;
48.             }
49.             else if (i < 0)
50.             {
51.                 i += size;
52.             }
53.         }
```

```

54.     }
55.     for (i = 0; i < size; i++)
56.     {
57.         for (j = 0; j < size; j++)
58.         {
59.             printf("%d ", a[i][j]);
60.         }
61.         printf("\n");
62.     }
63.     printf("\n");
64. }
```

## (13). C Programming Examples using Recursion

### 1. C Examples on Traversal of a Tree using Recursion

```

1. /*
2. * C Program for Depth First Binary Tree Search using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. void DFS(struct node *);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
Exit\nChoice: ");
```

```

26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 DFS(head);
36.                 break;
37.             case 3:
38.                 delete(&head);
39.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
40.                 break;
41.             default:
42.                 printf("Not a valid input, try again\n");
43.             }
44.         } while (choice != 3);
45.     return 0;
46. }
47.

48. void generate(struct node **head, int num)
49. {
50.     struct node *temp = *head, *prev = *head;
51.
52.     if (*head == NULL)
53.     {
54.         *head = (struct node *)malloc(sizeof(struct node));
55.         (*head)->a = num;
56.         (*head)->left = (*head)->right = NULL;
57.     }
58.     else
59.     {
60.         while (temp != NULL)
61.         {
62.             if (num > temp->a)
63.             {
64.                 prev = temp;
65.                 temp = temp->right;
66.             }
67.             else
68.             {

```

```
69.             prev = temp;
70.             temp = temp->left;
71.         }
72.     }
73.     temp = (struct node *)malloc(sizeof(struct node));
74.     temp->a = num;
75.     if (num >= prev->a)
76.     {
77.         prev->right = temp;
78.     }
79.     else
80.     {
81.         prev->left = temp;
82.     }
83. }
84. }
85.
86. void DFS(struct node *head)
87. {
88.     if (head)
89.     {
90.         if (head->left)
91.         {
92.             DFS(head->left);
93.         }
94.         if (head->right)
95.         {
96.             DFS(head->right);
97.         }
98.         printf("%d ", head->a);
99.     }
100. }
101.
102. void delete(struct node **head)
103. {
104.     if (*head != NULL)
105.     {
106.         if ((*head)->left)
107.         {
108.             delete(&(*head)->left);
109.         }
110.         if ((*head)->right)
111.         {
```

```

112.             delete(&(*head)->right);
113.         }
114.         free(*head);
115.     }
116. }
```

```

1. /*
2. * C Program to Traverse the Tree Recursively
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. void infix(struct node *);
16. void postfix(struct node *);
17. void prefix(struct node *);
18. void delete(struct node **);
19.
20. int main()
21. {
22.     struct node *head = NULL;
23.     int choice = 0, num, flag = 0, key;
24.
25.     do
26.     {
27.         printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse
via prefix\n4. Traverse via postfix\n5. Exit\nChoice: ");
28.         scanf("%d", &choice);
29.         switch(choice)
30.         {
31.             case 1:
32.                 printf("Enter element to insert: ");
33.                 scanf("%d", &num);
34.                 generate(&head, num);
```

```
35.         break;
36.     case 2:
37.         infix(head);
38.         break;
39.     case 3:
40.         prefix(head);
41.         break;
42.     case 4:
43.         postfix(head);
44.         break;
45.     case 5:
46.         delete(&head);
47.         printf("Memory Cleared\nPROGRAM TERMINATED\n");
48.         break;
49.     default: printf("Not a valid input, try again\n");
50. }
51. } while (choice != 5);
52. return 0;
53. }

54.

55. void generate(struct node **head, int num)
56. {
57.     struct node *temp = *head, *prev = *head;
58.
59.     if (*head == NULL)
60.     {
61.         *head = (struct node *)malloc(sizeof(struct node));
62.         (*head)->a = num;
63.         (*head)->left = (*head)->right = NULL;
64.     }
65.     else
66.     {
67.         while (temp != NULL)
68.         {
69.             if (num > temp->a)
70.             {
71.                 prev = temp;
72.                 temp = temp->right;
73.             }
74.             else
75.             {
76.                 prev = temp;
77.                 temp = temp->left;
```

```
78.         }
79.     }
80.     temp = (struct node *)malloc(sizeof(struct node));
81.     temp->a = num;
82.     if (num >= prev->a)
83.     {
84.         prev->right = temp;
85.     }
86.     else
87.     {
88.         prev->left = temp;
89.     }
90. }
91. }
92.
93. void infix(struct node *head)
94. {
95.     if (head)
96.     {
97.         infix(head->left);
98.         printf("%d    ", head->a);
99.         infix(head->right);
100.        }
101.    }
102.
103. void prefix(struct node *head)
104. {
105.     if (head)
106.     {
107.         printf("%d    ", head->a);
108.         prefix(head->left);
109.         prefix(head->right);
110.        }
111.    }
112.
113. void postfix(struct node *head)
114. {
115.     if (head)
116.     {
117.         postfix(head->left);
118.         postfix(head->right);
119.         printf("%d    ", head->a);
120.        }
}
```

```

121.     }
122.
123.     void delete(struct node **head)
124.     {
125.         if (*head != NULL)
126.         {
127.             if ((*head)->left)
128.             {
129.                 delete(&(*head)->left);
130.             }
131.             if ((*head)->right)
132.             {
133.                 delete(&(*head)->right);
134.             }
135.             free(*head);
136.         }
137.     }

```

```

1. /*
2.  * C Program to Reverse a Stack using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13.void generate(struct node **);
14.void display(struct node *);
15.void stack_reverse(struct node **, struct node **);
16.void delete(struct node **);
17.
18.int main()
19.{
20.    struct node *head = NULL;
21.
22.    generate(&head);
23.    printf("\nThe sequence of contents in stack\n");

```

```
24.     display(head);
25.     printf("\nInversing the contents of the stack\n");
26.     if (head != NULL)
27.     {
28.         stack_reverse(&head, &(head->next));
29.     }
30.     printf("\nThe contents in stack after reversal\n");
31.     display(head);
32.     delete(&head);
33.
34.     return 0;
35. }
36.
37. void stack_reverse(struct node **head, struct node **head_next)
38. {
39.     struct node *temp;
40.
41.     if (*head_next != NULL)
42.     {
43.         temp = (*head_next)->next;
44.         (*head_next)->next = (*head);
45.         *head = *head_next;
46.         *head_next = temp;
47.         stack_reverse(head, head_next);
48.     }
49. }
50.
51. void display(struct node *head)
52. {
53.     if (head != NULL)
54.     {
55.         printf("%d ", head->a);
56.         display(head->next);
57.     }
58. }
59.
60. void generate(struct node **head)
61. {
62.     int num, i;
63.     struct node *temp;
64.
65.     printf("Enter length of list: ");
66.     scanf("%d", &num);
```

```

67.     for (i = num; i > 0; i--)
68.     {
69.         temp = (struct node *)malloc(sizeof(struct node));
70.         temp->a = i;
71.         if (*head == NULL)
72.         {
73.             *head = temp;
74.             (*head)->next = NULL;
75.         }
76.         else
77.         {
78.             temp->next = *head;
79.             *head = temp;
80.         }
81.     }
82. }
83.
84. void delete(struct node **head)
85. {
86.     struct node *temp;
87.     while (*head != NULL)
88.     {
89.         temp = *head;
90.         *head = (*head)->next;
91.         free(temp);
92.     }
93. }
```

```

1. /*
2.  * Recursive C program to find Length of a Linked List
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13. void generate(struct node **);
```

```
14. int length(struct node*);  
15. void delete(struct node **);  
16.  
17. int main()  
18. {  
19.     struct node *head = NULL;  
20.     int count;  
21.  
22.     generate(&head);  
23.     count = length(head);  
24.     printf("The number of nodes are: %d\n", count);  
25.     delete(&head);  
26.     return 0;  
27. }  
28.  
29. void generate(struct node **head)  
30. {  
31.     /* for unknown number of nodes use num = rand() % 20; */  
32.     int num = 10, i;  
33.     struct node *temp;  
34.  
35.     for (i = 0; i < num; i++)  
36.     {  
37.         temp = (struct node *)malloc(sizeof(struct node));  
38.         temp->a = i;  
39.         if (*head == NULL)  
40.         {  
41.             *head = temp;  
42.             (*head)->next = NULL;  
43.         }  
44.         else  
45.         {  
46.             temp->next = *head;  
47.             *head = temp;  
48.         }  
49.     }  
50. }  
51.  
52. int length(struct node *head)  
53. {  
54.     if (head->next == NULL)  
55.     {  
56.         return 1;
```

```

57.     }
58.     return (1 + length(head->next));
59. }
60.
61. void delete(struct node **head)
62. {
63.     struct node *temp;
64.     while (*head != NULL)
65.     {
66.         temp = *head;
67.         *head = (*head)->next;
68.         free(temp);
69.     }
70. }
```

## 2. C Examples on Sorting Algorithms using Recursion

```

1. /*
2.  * C Program to Implement Selection Sort Recursively
3. */
4. #include <stdio.h>
5.
6. void selection(int [], int, int, int, int);
7.
8. int main()
9. {
10.    int list[30], size, temp, i, j;
11.
12.    printf("Enter the size of the list: ");
13.    scanf("%d", &size);
14.    printf("Enter the elements in list:\n");
15.    for (i = 0; i < size; i++)
16.    {
17.        scanf("%d", &list[i]);
18.    }
19.    selection(list, 0, 0, size, 1);
20.    printf("The sorted list in ascending order is\n");
21.    for (i = 0; i < size; i++)
22.    {
23.        printf("%d ", list[i]);
24.    }
25. }
```

```

26.     return 0;
27. }
28.
29. void selection(int list[], int i, int j, int size, int flag)
30. {
31.     int temp;
32.
33.     if (i < size - 1)
34.     {
35.         if (flag)
36.         {
37.             j = i + 1;
38.         }
39.         if (j < size)
40.         {
41.             if (list[i] > list[j])
42.             {
43.                 temp = list[i];
44.                 list[i] = list[j];
45.                 list[j] = temp;
46.             }
47.             selection(list, i, j + 1, size, 0);
48.         }
49.     selection(list, i + 1, 0, size, 1);
50. }
51. }
```

```

1. /*
2.  * C Program to Input Few Numbers & Perform Merge Sort on them using Recursion
3. */
4.
5. #include <stdio.h>
6.
7. void mergeSort(int [], int, int, int);
8. void partition(int [],int, int);
9.
10.int main()
11.{
12.    int list[50];
13.    int i, size;
```

```
15.     printf("Enter total number of elements:");
16.     scanf("%d", &size);
17.     printf("Enter the elements:\n");
18.     for(i = 0; i < size; i++)
19.     {
20.         scanf("%d", &list[i]);
21.     }
22.     partition(list, 0, size - 1);
23.     printf("After merge sort:\n");
24.     for(i = 0;i < size; i++)
25.     {
26.         printf("%d    ",list[i]);
27.     }
28.
29.     return 0;
30. }
31.
32. void partition(int list[],int low,int high)
33. {
34.     int mid;
35.
36.     if(low < high)
37.     {
38.         mid = (low + high) / 2;
39.         partition(list, low, mid);
40.         partition(list, mid + 1, high);
41.         mergeSort(list, low, mid, high);
42.     }
43. }
44.
45. void mergeSort(int list[],int low,int mid,int high)
46. {
47.     int i, mi, k, lo, temp[50];
48.
49.     lo = low;
50.     i = low;
51.     mi = mid + 1;
52.     while ((lo <= mid) && (mi <= high))
53.     {
54.         if (list[lo] <= list[mi])
55.         {
56.             temp[i] = list[lo];
57.             lo++;
```

```

58.     }
59.     else
60.     {
61.         temp[i] = list[mi];
62.         mi++;
63.     }
64.     i++;
65. }
66. if (lo > mid)
67. {
68.     for (k = mi; k <= high; k++)
69.     {
70.         temp[i] = list[k];
71.         i++;
72.     }
73. }
74. else
75. {
76.     for (k = lo; k <= mid; k++)
77.     {
78.         temp[i] = list[k];
79.         i++;
80.     }
81. }
82.
83. for (k = low; k <= high; k++)
84. {
85.     list[k] = temp[k];
86. }
87. }
```

```

1. /*
2.  * C Program to find the nth number in Fibonacci series using recursion
3. */
4. #include <stdio.h>
5. int fibo(int);
6.
7. int main()
8. {
9.     int num;
10.    int result;
```

```

11.
12.     printf("Enter the nth number in fibonacci series: ");
13.     scanf("%d", &num);
14.     if (num < 0)
15.     {
16.         printf("Fibonacci of negative number is not possible.\n");
17.     }
18.     else
19.     {
20.         result = fibo(num);
21.         printf("The %d number in fibonacci series is %d\n", num, result);
22.     }
23.     return 0;
24. }
25. int fibo(int num)
26. {
27.     if (num == 0)
28.     {
29.         return 0;
30.     }
31.     else if (num == 1)
32.     {
33.         return 1;
34.     }
35.     else
36.     {
37.         return(fibo(num - 1) + fibo(num - 2));
38.     }
39. }
```

```

1. /*
2.  * C Program to find the Biggest Number in an Array of Numbers using
3.  * Recursion
4. */
5. #include <stdio.h>
6.
7. int large(int[], int, int);
8.
9. int main()
10. {
11.     int size;
```

```
12.     int largest;
13.     int list[20];
14.     int i;
15.
16.     printf("Enter size of the list:");
17.     scanf("%d", &size);
18.     printf("Printing the list:\n");
19.     for (i = 0; i < size ; i++)
20.     {
21.         list[i] = rand() % size;
22.         printf("%d\t", list[i]);
23.     }
24.     if (size == 0)
25.     {
26.         printf("Empty list\n");
27.     }
28.     else
29.     {
30.         largest = list[0];
31.         largest = large(list, size - 1, largest);
32.         printf("\nThe largest number in the list is: %d\n", largest);
33.     }
34. }
35.int large(int list[], int size, int largest)
36.{ 
37.     if (size == 1)
38.         return largest;
39.
40.     if (size > -1)
41.     {
42.         if (list[size] > largest)
43.         {
44.             largest = list[size];
45.         }
46.         return(largest = large(list, size - 1, largest));
47.     }
48.     else
49.     {
50.         return largest;
51.     }
52. }
```

```

1. /*
2.  * C Program to find Sum of Digits of a Number using Recursion
3. */
4. #include <stdio.h>
5.
6. int sum (int a);
7.
8. int main()
9. {
10.     int num, result;
11.
12.     printf("Enter the number: ");
13.     scanf("%d", &num);
14.     result = sum(num);
15.     printf("Sum of digits in %d is %d\n", num, result);
16.     return 0;
17. }
18.
19.int sum (int num)
20.{
21.    if (num != 0)
22.    {
23.        return (num % 10 + sum (num / 10));
24.    }
25.    else
26.    {
27.        return 0;
28.    }
29.}
```

```

1. /*
2.  * C Program to find Sum of N Numbers using Recursion
3. */
4. #include <stdio.h>
5.
6. void display(int);
7.
8. int main()
9. {
10.     int num, result;
11.
```

```

12.     printf("Enter the Nth number: ");
13.     scanf("%d", &num);
14.     display(num);
15.     return 0;
16. }
17.
18. void display(int num)
19. {
20.     static int i = 1;
21.
22.     if (num == i)
23.     {
24.         printf("%d\n", num);
25.         return;
26.     }
27.     else
28.     {
29.         printf("%d ", i);
30.         i++;
31.         display(num);
32.     }
33. }
```

```

1. /*
2. * C Program to Perform Matrix Multiplication using Recursion
3. */
4. #include <stdio.h>
5.
6. void multiply(int, int, int [[10]], int, int, int [[10]], int [[10]]);
7. void display(int, int, int [[10]]);
8.
9. int main()
10. {
11.     int a[10][10], b[10][10], c[10][10] = {0};
12.     int m1, n1, m2, n2, i, j, k;
13.
14.     printf("Enter rows and columns for Matrix A respectively: ");
15.     scanf("%d%d", &m1, &n1);
16.     printf("Enter rows and columns for Matrix B respectively: ");
17.     scanf("%d%d", &m2, &n2);
```

```

18.     if (n1 != m2)
19.     {
20.         printf("Matrix multiplication not possible.\n");
21.     }
22. else
23. {
24.     printf("Enter elements in Matrix A:\n");
25.     for (i = 0; i < m1; i++)
26.         for (j = 0; j < n1; j++)
27.         {
28.             scanf("%d", &a[i][j]);
29.         }
30.     printf("\nEnter elements in Matrix B:\n");
31.     for (i = 0; i < m2; i++)
32.         for (j = 0; j < n2; j++)
33.         {
34.             scanf("%d", &b[i][j]);
35.         }
36.     multiply(m1, n1, a, m2, n2, b, c);
37. }
38. printf("On matrix multiplication of A and B the result is:\n");
39. display(m1, n2, c);
40. }
41.
42. void multiply (int m1, int n1, int a[10][10], int m2, int n2, int b[10][10], int
   c[10][10])
43. {
44.     static int i = 0, j = 0, k = 0;
45.
46.     if (i >= m1)
47.     {
48.         return;
49.     }
50.     else if (i < m1)
51.     {
52.         if (j < n2)
53.         {
54.             if (k < n1)
55.             {
56.                 c[i][j] += a[i][k] * b[k][j];
57.                 k++;
58.                 multiply(m1, n1, a, m2, n2, b, c);
59.             }
}

```

```

60.         k = 0;
61.         j++;
62.         multiply(m1, n1, a, m2, n2, b, c);
63.     }
64.     j = 0;
65.     i++;
66.     multiply(m1, n1, a, m2, n2, b, c);
67. }
68. }
69.
70. void display(int m1, int n2, int c[10][10])
71. {
72.     int i, j;
73.
74.     for (i = 0; i < m1; i++)
75.     {
76.         for (j = 0; j < n2; j++)
77.         {
78.             printf("%d ", c[i][j]);
79.         }
80.         printf("\n");
81.     }
82. }
```

### 3. C Examples on Linked List Implementation using Recursion

```

84. /*
85. * Recursive C program to reverse nodes of a Linked List and display
86. * them
87. */
88.#include <stdio.h>
89.#include <stdlib.h>
90.
91. struct node
92. {
93.     int data;
94.     struct node *next;
95. };
96.
97. void print_reverse_recursive (struct node *);
98. void print (struct node *);
99. void create_new_node (struct node *, int );
100.
```

```

101. //Driver Function
102. int main ()
103. {
104.     struct node *head = NULL;
105.     insert_new_node (&head, 1);
106.     insert_new_node (&head, 2);
107.     insert_new_node (&head, 3);
108.     insert_new_node (&head, 4);
109.     printf ("LinkedList : ");
110.     print (head);
111.     printf ("\nLinkedList in reverse order : ");
112.     print_reverse_recursive (head);
113.     printf ("\n");
114.     return 0;
115. }
116.
117. //Recursive Reverse
118. void print_reverse_recursive (struct node *head)
119. {
120.     if (head == NULL)
121.     {
122.         return;
123.     }
124.
125.     //Recursive call first
126.     print_reverse (head -> next);
127.     //Print Later
128.     printf ("%d ", head -> data);
129. }
130.
131. //Print the Linkedlist normal
132. void print (struct node *head)
133. {
134.     if (head == NULL)
135.     {
136.         return;
137.     }
138.     printf ("%d ", head -> data);
139.     print (head -> next);
140. }
141.
142. //New data added in the start
143. void insert_new_node (struct node ** head_ref, int new_data)

```

```

144. {
145.     struct node * new_node = (struct node *) malloc (sizeof (struct node));
146.     new_node -> data = new_data;
147.     new_node -> next = (*head_ref);
148.     (*head_ref) = new_node;
149. }
```

```

1. /*
2.  * Recursive C program to display members of a linked list
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13. void generate(struct node **);
14. void display(struct node* );
15. void delete(struct node **);
16.
17. int main()
18. {
19.     struct node *head = NULL;
20.
21.     generate(&head);
22.     display(head);
23.     delete(&head);
24.     return 0;
25. }
26.
27. void generate(struct node **head)
28. {
29.     int num = 10, i;
30.     struct node *temp;
31.
32.     for (i = 0; i < num; i++)
33.     {
34.         temp = (struct node *)malloc(sizeof(struct node));
35.         temp->a = i;
```

```

36.     if (*head == NULL)
37.     {
38.         *head = temp;
39.         (*head)->next = NULL;
40.     }
41.     else
42.     {
43.         temp->next = *head;
44.         *head = temp;
45.     }
46. }
47. }
48.
49. void display(struct node *head)
50. {
51.     printf("%d    ", head->a);
52.     if (head->next == NULL)
53.     {
54.         return;
55.     }
56.     display(head->next);
57. }
58.
59. void delete(struct node **head)
60. {
61.     struct node *temp;
62.     while (*head != NULL)
63.     {
64.         temp = *head;
65.         *head = (*head)->next;
66.         free(temp);
67.     }
68. }
```

```

1. /*
2.  * C program to find the number of occurrences of a given number in a
3.  * list
4.  */
5. #include <stdio.h>
6.
7. void occur(int [], int, int, int, int *);
```

```

9. int main()
10. {
11.     int size, key, count = 0;
12.     int list[20];
13.     int i;
14.
15.     printf("Enter the size of the list: ");
16.     scanf("%d", &size);
17.     printf("Printing the list:\n");
18.     for (i = 0; i < size; i++)
19.     {
20.         list[i] = rand() % size;
21.         printf("%d    ", list[i]);
22.     }
23.     printf("\nEnter the key to find it's occurrence: ");
24.     scanf("%d", &key);
25.     occur(list, size, 0, key, &count);
26.     printf("%d occurs for %d times.\n", key, count);
27.     return 0;
28. }
29.
30. void occur(int list[], int size, int index, int key, int *count)
31. {
32.     if (size == index)
33.     {
34.         return;
35.     }
36.     if (list[index] == key)
37.     {
38.         *count += 1;
39.     }
40.     occur(list, size, index + 1, key, count);
41. }
```

```

1. /*
2.  * C program to find the length of a string
3.  */
4. #include <stdio.h>
5.
6. int length(char [], int);
7. int main()
8. {
```

```

9.     char word[20];
10.    int count;
11.
12.    printf("Enter a word to count it's length: ");
13.    scanf("%s", word);
14.    count = length(word, 0);
15.    printf("The number of characters in %s are %d.\n", word, count);
16.    return 0;
17. }
18.
19. int length(char word[], int index)
20. {
21.     if (word[index] == '\0')
22.     {
23.         return 0;
24.     }
25.     return (1 + length(word, index + 1));
26. }
```

#### 4. C Examples to illustrate Binary Search Algorithm and Tower of Hanoi Problem using Recursion

```

1. /*
2. * C Program to Perform Binary Search using Recursion
3. */
4. #include <stdio.h>
5.
6. void binary_search(int [], int, int, int);
7. void bubble_sort(int [], int);
8.
9. int main()
10. {
11.     int key, size, i;
12.     int list[25];
13.
14.     printf("Enter size of a list: ");
15.     scanf("%d", &size);
16.     printf("Generating random numbers\n");
17.     for(i = 0; i < size; i++)
18.     {
19.         list[i] = rand() % 100;
20.         printf("%d ", list[i]);
```

```
21.     }
22.     bubble_sort(list, size);
23.     printf("\n\n");
24.     printf("Enter key to search\n");
25.     scanf("%d", &key);
26.     binary_search(list, 0, size, key);
27.
28. }
29.
30. void bubble_sort(int list[], int size)
31. {
32.     int temp, i, j;
33.     for (i = 0; i < size; i++)
34.     {
35.         for (j = i; j < size; j++)
36.         {
37.             if (list[i] > list[j])
38.             {
39.                 temp = list[i];
40.                 list[i] = list[j];
41.                 list[j] = temp;
42.             }
43.         }
44.     }
45. }
46.
47. void binary_search(int list[], int lo, int hi, int key)
48. {
49.     int mid;
50.
51.     if (lo > hi)
52.     {
53.         printf("Key not found\n");
54.         return;
55.     }
56.     mid = (lo + hi) / 2;
57.     if (list[mid] == key)
58.     {
59.         printf("Key found\n");
60.     }
61.     else if (list[mid] > key)
62.     {
63.         binary_search(list, lo, mid - 1, key);
```

```

64.     }
65.     else if (list[mid] < key)
66.     {
67.         binary_search(list, mid + 1, hi, key);
68.     }
69. }

```

```

1. /*
2. * C Program to Perform Quick Sort on a set of Entries from a File
3. * using Recursion
4. */
5. #include <stdio.h>
6.
7. void quicksort (int [], int, int);
8.
9. int main()
10. {
11.     int list[50];
12.     int size, i;
13.
14.     printf("Enter the number of elements: ");
15.     scanf("%d", &size);
16.     printf("Enter the elements to be sorted:\n");
17.     for (i = 0; i < size; i++)
18.     {
19.         scanf("%d", &list[i]);
20.     }
21.     quicksort(list, 0, size - 1);
22.     printf("After applying quick sort\n");
23.     for (i = 0; i < size; i++)
24.     {
25.         printf("%d ", list[i]);
26.     }
27.     printf("\n");
28.
29.     return 0;
30. }
31. void quicksort(int list[], int low, int high)
32. {
33.     int pivot, i, j, temp;
34.     if (low < high)
35.     {

```

```

36.     pivot = low;
37.     i = low;
38.     j = high;
39.     while (i < j)
40.     {
41.         while (list[i] <= list[pivot] && i <= high)
42.         {
43.             i++;
44.         }
45.         while (list[j] > list[pivot] && j >= low)
46.         {
47.             j--;
48.         }
49.         if (i < j)
50.         {
51.             temp = list[i];
52.             list[i] = list[j];
53.             list[j] = temp;
54.         }
55.     }
56.     temp = list[j];
57.     list[j] = list[pivot];
58.     list[pivot] = temp;
59.     quicksort(list, low, j - 1);
60.     quicksort(list, j + 1, high);
61. }
62. }
```

```

1. /*
2. * C Program to Reverse the String using Recursion
3. */
4. #include <stdio.h>
5. #include <string.h>
6.
7. void reverse(char [], int, int);
8. int main()
9. {
10.     char str1[20];
11.     int size;
12.
13.     printf("Enter a string to reverse: ");
14.     scanf("%s", str1);
```

```

15.     size = strlen(str1);
16.     reverse(str1, 0, size - 1);
17.     printf("The string after reversing is: %s\n", str1);
18.     return 0;
19. }
20.
21. void reverse(char str1[], int index, int size)
22. {
23.     char temp;
24.     temp = str1[index];
25.     str1[index] = str1[size - index];
26.     str1[size - index] = temp;
27.     if (index == size / 2)
28.     {
29.         return;
30.     }
31.     reverse(str1, index + 1, size);
32. }
```

```

1. /*
2.  * C program to find the reverse of a number using recursion
3. */
4. #include <stdio.h>
5. #include <math.h>
6.
7. int rev(int, int);
8.
9. int main()
10. {
11.     int num, result;
12.     int length = 0, temp;
13.
14.     printf("Enter an integer number to reverse: ");
15.     scanf("%d", &num);
16.     temp = num;
17.     while (temp != 0)
18.     {
19.         length++;
20.         temp = temp / 10;
21.     }
22.     result = rev(num, length);
23.     printf("The reverse of %d is %d.\n", num, result);
```

```

24.     return 0;
25. }
26.
27. int rev(int num, int len)
28. {
29.     if (len == 1)
30.     {
31.         return num;
32.     }
33.     else
34.     {
35.         return (((num % 10) * pow(10, len - 1)) + rev(num / 10, --len));
36.     }
37. }
```

```

1. /*
2.  * C program for Tower of Hanoi using Recursion
3. */
4. #include <stdio.h>
5.
6. void towers(int, char, char, char);
7.
8. int main()
9. {
10.     int num;
11.
12.     printf("Enter the number of disks : ");
13.     scanf("%d", &num);
14.     printf("The sequence of moves involved in the Tower of Hanoi are :\n");
15.     towers(num, 'A', 'C', 'B');
16.     return 0;
17. }
18. void towers(int num, char frompeg, char topeg, char auxpeg)
19. {
20.     if (num == 1)
21.     {
22.         printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
23.         return;
24.     }
25.     towers(num - 1, frompeg, auxpeg, topeg);
26.     printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
27.     towers(num - 1, auxpeg, topeg, frompeg);
```

28. }

```

1. /*
2.  * C Program to Copy One String to Another using Recursion
3.  */
4. #include <stdio.h>
5.
6. void copy(char [], char [], int);
7.
8. int main()
9. {
10.    char str1[20], str2[20];
11.
12.    printf("Enter string to copy: ");
13.    scanf("%s", str1);
14.    copy(str1, str2, 0);
15.    printf("Copying success.\n");
16.    printf("The first string is: %s\n", str1);
17.    printf("The second string is: %s\n", str2);
18.    return 0;
19. }
20.
21. void copy(char str1[], char str2[], int index)
22. {
23.    str2[index] = str1[index];
24.    if (str1[index] == '\0')
25.        return;
26.    copy(str1, str2, index + 1);
27. }
```

```

1. /*
2.  * C Program to Check whether a given String is Palindrome or not
3.  * using Recursion
4.  */
5. #include <stdio.h>
6. #include <string.h>
7.
8. void check(char [], int);
9.
10. int main()
11. {
```

```

12.     char word[15];
13.
14.     printf("Enter a word to check if it is a palindrome\n");
15.     scanf("%s", word);
16.     check(word, 0);
17.
18.     return 0;
19. }
20.
21. void check(char word[], int index)
22. {
23.     int len = strlen(word) - (index + 1);
24.     if (word[index] == word[len])
25.     {
26.         if (index + 1 == len || index == len)
27.         {
28.             printf("The entered word is a palindrome\n");
29.             return;
30.         }
31.         check(word, index + 1);
32.     }
33.     else
34.     {
35.         printf("The entered word is not a palindrome\n");
36.     }
37. }

```

## 5. C Examples on Basic Mathematical Operations using Recursion

```

1. /*
2. * C Program to find whether a Number is Prime or Not using Recursion
3. */
4. #include <stdio.h>
5.
6. int primeno(int, int);
7.
8. int main()
9. {
10.     int num, check;
11.     printf("Enter a number: ");
12.     scanf("%d", &num);
13.     check = primeno(num, num / 2);

```

```

14.     if (check == 1)
15.     {
16.         printf("%d is a prime number\n", num);
17.     }
18.     else
19.     {
20.         printf("%d is not a prime number\n", num);
21.     }
22.     return 0;
23. }
24.
25.int primeno(int num, int i)
26.{ 
27.    if (i == 1)
28.    {
29.        return 1;
30.    }
31.    else
32.    {
33.        if (num % i == 0)
34.        {
35.            return 0;
36.        }
37.        else
38.        {
39.            return primeno(num, i - 1);
40.        }
41.    }
42. }
```

```

1. /*
2. * C Program to find factorial of a given number using recursion
3. */
4. #include <stdio.h>
5.
6. int factorial(int);
7.
8. int main()
9. {
10.    int num;
11.    int result;
```

```

13.     printf("Enter a number to find it's Factorial: ");
14.     scanf("%d", &num);
15.     if (num < 0)
16.     {
17.         printf("Factorial of negative number not possible\n");
18.     }
19.     else
20.     {
21.         result = factorial(num);
22.         printf("The Factorial of %d is %d.\n", num, result);
23.     }
24.     return 0;
25. }
26. int factorial(int num)
27. {
28.     if (num == 0 || num == 1)
29.     {
30.         return 1;
31.     }
32.     else
33.     {
34.         return(num * factorial(num - 1));
35.     }
36. }
```

```

1. /*
2.  * C Program to Find LCM of a Number using Recursion
3.  */
4. #include <stdio.h>
5.
6. int lcm(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.     int prime[100];
12.
13.     printf("Enter two numbers: ");
14.     scanf("%d%d", &a, &b);
15.     result = lcm(a, b);
16.     printf("The LCM of %d and %d is %d\n", a, b, result);
17.     return 0;
```

```

18. }
19.
20. int lcm(int a, int b)
21. {
22.     static int common = 1;
23.
24.     if (common % a == 0 && common % b == 0)
25.     {
26.         return common;
27.     }
28.     common++;
29.     lcm(a, b);
30.     return common;
31. }
```

```

1. /*
2. * C Program to find GCD of given Numbers using Recursion
3. */
4. #include <stdio.h>
5.
6. int gcd(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their GCD: ");
13.     scanf("%d%d", &a, &b);
14.     result = gcd(a, b);
15.     printf("The GCD of %d and %d is %d.\n", a, b, result);
16. }
17.
18. int gcd(int a, int b)
19. {
20.     while (a != b)
21.     {
22.         if (a > b)
23.         {
24.             return gcd(a - b, b);
25.         }
26.         else
27.         {
```

```

28.         return gcd(a, b - a);
29.     }
30. }
31. return a;
32. }}

```

```

1. /*
2. * C Program to find HCF of a given Number using Recursion
3. */
4. #include <stdio.h>
5.
6. int hcf(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their HCF: ");
13.     scanf("%d%d", &a, &b);
14.     result = hcf(a, b);
15.     printf("The HCF of %d and %d is %d.\n", a, b, result);
16. }
17.
18. int hcf(int a, int b)
19. {
20.     while (a != b)
21.     {
22.         if (a > b)
23.         {
24.             return hcf(a - b, b);
25.         }
26.         else
27.         {
28.             return hcf(a, b - a);
29.         }
30.     }
31.     return a;
32. }

```

```

1. /*
2. * C Program to find Product of 2 Numbers using Recursion

```

```

3. /*
4. #include <stdio.h>
5.
6. int product(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter two numbers to find their product: ");
13.     scanf("%d%d", &a, &b);
14.     result = product(a, b);
15.     printf("Product of %d and %d is %d\n", a, b, result);
16.     return 0;
17. }
18.
19. int product(int a, int b)
20. {
21.     if (a < b)
22.     {
23.         return product(b, a);
24.     }
25.     else if (b != 0)
26.     {
27.         return (a + product(a, b - 1));
28.     }
29.     else
30.     {
31.         return 0;
32.     }
33. }
```

```

1. /*
2. * C Program to find Power of a Number using Recursion
3. */
4. #include <stdio.h>
5.
6. long power (int, int);
7.
8. int main()
9. {
10.     int pow, num;
```

```

11.     long result;
12.
13.     printf("Enter a number: ");
14.     scanf("%d", &num);
15.     printf("Enter it's power: ");
16.     scanf("%d", &pow);
17.     result = power(num, pow);
18.     printf("%d^%d is %ld", num, pow, result);
19.     return 0;
20. }
21.
22. long power (int num, int pow)
23. {
24.     if (pow)
25.     {
26.         return (num * power(num, pow - 1));
27.     }
28.     return 1;
29. }
```

## 6. C Examples on Number System Conversion using Recursion

```

1. /*
2.  * C Program to Print Binary Equivalent of an Integer using Recursion
3.  */
4. #include <stdio.h>
5.
6. int binary_conversion(int);
7.
8. int main()
9. {
10.     int num, bin;
11.
12.     printf("Enter a decimal number: ");
13.     scanf("%d", &num);
14.     bin = binary_conversion(num);
15.     printf("The binary equivalent of %d is %d\n", num, bin);
16. }
17.
18. int binary_conversion(int num)
19. {
20.     if (num == 0)
```

```

21.     {
22.         return 0;
23.     }
24.     else
25.     {
26.         return (num % 2) + 10 * binary_conversion(num / 2);
27.     }
28. }

```

```

1. /*
2.  * C Program to Print the Alternate Nodes in a Linked List using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13.void generate(struct node **);
14.void display(struct node *);
15.void delete(struct node **);
16.
17.int main()
18.{
19.    struct node *head = NULL;
20.
21.    generate(&head);
22.    printf("\nDisplaying the alternate nodes\n");
23.    display(head);
24.    delete(&head);
25.
26.    return 0;
27.}
28.
29.void display(struct node *head)
30.{
31.    static flag = 0;
32.    if(head != NULL)
33.    {

```

```
34.         if (!(flag % 2))
35.         {
36.             printf("%d  ", head->a);
37.         }
38.         flag++;
39.         display(head->next);
40.     }
41. }
42.
43. void generate(struct node **head)
44. {
45.     int num, i;
46.     struct node *temp;
47.
48.     printf("Enter length of list: ");
49.     scanf("%d", &num);
50.     for (i = num; i > 0; i--)
51.     {
52.         temp = (struct node *)malloc(sizeof(struct node));
53.         temp->a = i;
54.         if (*head == NULL)
55.         {
56.             *head = temp;
57.             (*head)->next = NULL;
58.         }
59.         else
60.         {
61.             temp->next = *head;
62.             *head = temp;
63.         }
64.     }
65. }
66.
67. void delete(struct node **head)
68. {
69.     struct node *temp;
70.     while (*head != NULL)
71.     {
72.         temp = *head;
73.         *head = (*head)->next;
74.         free(temp);
75.     }
76. }
```

```

77. *
78. * C Program to Convert a Number Decimal System to Binary System using
    Recursion
79. */
80.#include <stdio.h>
81.
82.int convert(int);
83.
84.int main()
85.{
86.    int dec, bin;
87.
88.    printf("Enter a decimal number: ");
89.    scanf("%d", &dec);
90.    bin = convert(dec);
91.    printf("The binary equivalent of %d is %d.\n", dec, bin);
92.
93.    return 0;
94.}
95.
96.int convert(int dec)
97.{
98.    if (dec == 0)
99.    {
100.        return 0;
101.    }
102.    else
103.    {
104.        return (dec % 2 + 10 * convert(dec / 2));
105.    }
106.}

```

```

1. *
2. * C Program to find the first capital Letter in a string using
3. * Recursion
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #include <ctype.h>
8.
9. char caps_check(char *);
10.

```

```

11. int main()
12. {
13.     char string[20], letter;
14.
15.     printf("Enter a string to find it's first capital letter: ");
16.     scanf("%s", string);
17.     letter = caps_check(string);
18.     if (letter == 0)
19.     {
20.         printf("No capital letter is present in %s.\n", string);
21.     }
22.     else
23.     {
24.         printf("The first capital letter in %s is %c.\n", string, letter);      }
25.     return 0;
26. }
27. char caps_check(char *string)
28. {
29.     static int i = 0;
30.     if (i < strlen(string))
31.     {
32.         if (isupper(string[i]))
33.         {
34.             return string[i];
35.         }
36.         else
37.         {
38.             i = i + 1;
39.             return caps_check(string);
40.         }
41.     }
42.     else return 0;
43. }
```

## (14.) C Programming Examples without using Recursion

### 1. C Examples on Traversal of a Tree without using Recursion

```

1. /*
2.  * C Program to Traverse the Tree Non-Recursively
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
```

```
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. int search(struct node *, int);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 printf("Enter key to search: ");
36.                 scanf("%d", &key);
37.                 flag = search(head, key);
38.                 if (flag)
39.                 {
40.                     printf("Key found in tree\n");
41.                 }
42.                 else
43.                 {
44.                     printf("Key not found\n");
45.                 }
46.                 break;
47.             case 3:
48.                 delete(&head);
```

```
49.         printf("Memory Cleared\nPROGRAM TERMINATED\n");
50.         break;
51.     default: printf("Not a valid input, try again\n");
52. }
53. } while (choice != 3);
54. return 0;
55. }
56.
57. void generate(struct node **head, int num)
58. {
59.     struct node *temp = *head, *prev = *head;
60.
61.     if (*head == NULL)
62.     {
63.         *head = (struct node *)malloc(sizeof(struct node));
64.         (*head)->a = num;
65.         (*head)->left = (*head)->right = NULL;
66.     }
67.     else
68.     {
69.         while (temp != NULL)
70.         {
71.             if (num > temp->a)
72.             {
73.                 prev = temp;
74.                 temp = temp->right;
75.             }
76.             else
77.             {
78.                 prev = temp;
79.                 temp = temp->left;
80.             }
81.         }
82.         temp = (struct node *)malloc(sizeof(struct node));
83.         temp->a = num;
84.         if (num >= prev->a)
85.         {
86.             prev->right = temp;
87.         }
88.         else
89.         {
90.             prev->left = temp;
91.         }
92.     }
93. }
```

```

92.     }
93. }
94.
95. int search(struct node *head, int key)
96. {
97.     while (head != NULL)
98.     {
99.         if (key > head->a)
100.             {
101.                 head = head->right;
102.             }
103.         else if (key < head->a)
104.             {
105.                 head = head->left;
106.             }
107.         else
108.             {
109.                 return 1;
110.             }
111.     }
112.     return 0;
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }
129. }
```

```

1. /*
2.  * C Program to Traverse the Tree Non-Recursively
3. */
```

```
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *left;
11.    struct node *right;
12. };
13.
14. void generate(struct node **, int);
15. int search(struct node *, int);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int choice = 0, num, flag = 0, key;
22.
23.     do
24.     {
25.         printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
26.         scanf("%d", &choice);
27.         switch(choice)
28.         {
29.             case 1:
30.                 printf("Enter element to insert: ");
31.                 scanf("%d", &num);
32.                 generate(&head, num);
33.                 break;
34.             case 2:
35.                 printf("Enter key to search: ");
36.                 scanf("%d", &key);
37.                 flag = search(head, key);
38.                 if (flag)
39.                 {
40.                     printf("Key found in tree\n");
41.                 }
42.                 else
43.                 {
44.                     printf("Key not found\n");
45.                 }
46.         }
47.     }
48. }
```

```
47.     case 3:
48.         delete(&head);
49.         printf("Memory Cleared\nPROGRAM TERMINATED\n");
50.         break;
51.     default: printf("Not a valid input, try again\n");
52. }
53. } while (choice != 3);
54. return 0;
55. }

56.

57. void generate(struct node **head, int num)
58. {
59.     struct node *temp = *head, *prev = *head;
60.
61.     if (*head == NULL)
62.     {
63.         *head = (struct node *)malloc(sizeof(struct node));
64.         (*head)->a = num;
65.         (*head)->left = (*head)->right = NULL;
66.     }
67.     else
68.     {
69.         while (temp != NULL)
70.         {
71.             if (num > temp->a)
72.             {
73.                 prev = temp;
74.                 temp = temp->right;
75.             }
76.             else
77.             {
78.                 prev = temp;
79.                 temp = temp->left;
80.             }
81.         }
82.         temp = (struct node *)malloc(sizeof(struct node));
83.         temp->a = num;
84.         if (num >= prev->a)
85.         {
86.             prev->right = temp;
87.         }
88.         else
89.         {
```

```
90.             prev->left = temp;
91.         }
92.     }
93. }
94.
95. int search(struct node *head, int key)
96. {
97.     while (head != NULL)
98.     {
99.         if (key > head->a)
100.             {
101.                 head = head->right;
102.             }
103.         else if (key < head->a)
104.             {
105.                 head = head->left;
106.             }
107.         else
108.             {
109.                 return 1;
110.             }
111.     }
112.     return 0;
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }
129. }
```

```

1. /*
2.  * C Program for Depth First Binary Tree Search without using
3.  * Recursion
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *left;
12.     struct node *right;
13.     int visited;
14. };
15.
16. void generate(struct node **, int);
17. void DFS(struct node *);
18. void delete(struct node **);
19.
20. int main()
21. {
22.     struct node *head = NULL;
23.     int choice = 0, num, flag = 0, key;
24.
25.     do
26.     {
27.         printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
Exit\nChoice: ");
28.         scanf("%d", &choice);
29.         switch(choice)
30.         {
31.             case 1:
32.                 printf("Enter element to insert: ");
33.                 scanf("%d", &num);
34.                 generate(&head, num);
35.                 break;
36.             case 2:
37.                 DFS(head);
38.                 break;
39.             case 3:
40.                 delete(&head);
41.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
42.                 break;

```

```
43.     default:
44.         printf("Not a valid input, try again\n");
45.     }
46. } while (choice != 3);
47.
48. return 0;
49. }
50.
51. void generate(struct node **head, int num)
52. {
53.     struct node *temp = *head, *prev = *head;
54.
55.     if (*head == NULL)
56.     {
57.         *head = (struct node *)malloc(sizeof(struct node));
58.         (*head)->a = num;
59.         (*head)->visited = 0;
60.         (*head)->left = (*head)->right = NULL;
61.     }
62.     else
63.     {
64.         while (temp != NULL)
65.         {
66.             if (num > temp->a)
67.             {
68.                 prev = temp;
69.                 temp = temp->right;
70.             }
71.             else
72.             {
73.                 prev = temp;
74.                 temp = temp->left;
75.             }
76.         }
77.         temp = (struct node *)malloc(sizeof(struct node));
78.         temp->a = num;
79.         temp->visited = 0;
80.         if (temp->a >= prev->a)
81.         {
82.             prev->right = temp;
83.         }
84.         else
85.         {
```

```

86.             prev->left = temp;
87.         }
88.     }
89. }
90.
91. void DFS(struct node *head)
92. {
93.     struct node *temp = head, *prev;
94.
95.     printf("On DFS traversal we get:\n");
96.     while (temp && !temp->visited)
97.     {
98.         if (temp->left && !temp->left->visited)
99.         {
100.             temp = temp->left;
101.         }
102.         else if (temp->right && !temp->right->visited)
103.         {
104.             temp = temp->right;
105.         }
106.         else
107.         {
108.             printf("%d ", temp->a);
109.             temp->visited = 1;
110.             temp = head;
111.         }
112.     }
113. }
114.
115. void delete(struct node **head)
116. {
117.     if (*head != NULL)
118.     {
119.         if ((*head)->left)
120.         {
121.             delete(&(*head)->left);
122.         }
123.         if ((*head)->right)
124.         {
125.             delete(&(*head)->right);
126.         }
127.         free(*head);
128.     }

```

129. }

## 2. C Examples on solving Magic Square Puzzle and finding the HCF of a given number without using Recursion

```

1. /*
2. * C Program to find HCF of a given Number without using Recursion
3. */
4. #include <stdio.h>
5.
6. int hcf(int, int);
7.
8. int main()
9. {
10.     int a, b, result;
11.
12.     printf("Enter the two numbers to find their HCF: ");
13.     scanf("%d%d", &a, &b);
14.     result = hcf(a, b);
15.     printf("The HCF of %d and %d is %d.\n", a, b, result);
16.
17.     return 0;
18. }
19.
20.int hcf(int a, int b)
21.{
22.    while (a != b)
23.    {
24.        if (a > b)
25.        {
26.            a = a - b;
27.        }
28.        else
29.        {
30.            b = b - a;
31.        }
32.    }
33.    return a;
34.}
```

```

1. /*
2. * C Program to Solve the Magic Squares Puzzle without using
```

```
3.  * Recursion
4.  */
5. #include <stdio.h>
6.
7. void magicsq(int, int [][][10]);
8.
9. int main( )
10. {
11.     int size;
12.     int a[10][10];
13.
14.     printf("Enter the size: ");
15.     scanf("%d", &size);
16.     if (size % 2 == 0)
17.     {
18.         printf("Magic square works for an odd numbered size\n");
19.     }
20.     else
21.     {
22.         magicsq(size, a);
23.     }
24.     return 0;
25. }
26.
27. void magicsq(int size, int a[][][10])
28. {
29.     int sqr = size * size;
30.     int i = 0, j = size / 2, k;
31.
32.     for (k = 1; k <= sqr; ++k)
33.     {
34.         a[i][j] = k;
35.         i--;
36.         j++;
37.
38.         if (k % size == 0)
39.         {
40.             i += 2;
41.             --j;
42.         }
43.         else
44.         {
45.             if (j == size)
```

```

46.         {
47.             j -= size;
48.         }
49.         else if (i < 0)
50.         {
51.             i += size;
52.         }
53.     }
54. }
55. for (i = 0; i < size; i++)
56. {
57.     for (j = 0; j < size; j++)
58.     {
59.         printf("%d ", a[i][j]);
60.     }
61.     printf("\n");
62. }
63. printf("\n");
64. }
```

### 3. C Examples on Number Conversion without using Recursion

```

1. /*
2. * C Program to Convert Binary Code of a Number into its Equivalent
3. * Gray's Code without using Recursion
4. */
5. #include <stdio.h>
6. #include <math.h>
7.
8. int bintogray(int);
9.
10.int main ()
11.{
12.    int bin, gray;
13.
14.    printf("Enter a binary number: ");
15.    scanf("%d", &bin);
16.    gray = bintogray(bin);
17.    printf("The gray code of %d is %d\n", bin, gray);
18.    return 0;
19.}
20.
```

```

21. int bintogray(int bin)
22. {
23.     int a, b, result = 0, i = 0;
24.
25.     while (bin != 0)
26.     {
27.         a = bin % 10;
28.         bin = bin / 10;
29.         b = bin % 10;
30.         if ((a && !b) || (!a && b))
31.         {
32.             result = result + pow(10, i);
33.         }
34.         i++;
35.     }
36.     return result;
37. }
```

```

1. /*
2. * C Program to find Product of 2 Numbers without using Recursion
3. */
4.
5. #include <stdio.h>
6.
7. int product(int, int);
8.
9. int main()
10. {
11.     int a, b, result;
12.
13.     printf("Enter two numbers to find their product: ");
14.     scanf("%d%d", &a, &b);
15.     result = product(a, b);
16.     printf("Product of %d and %d is %d\n", a, b, result);
17.     return 0;
18. }
19.
20. int product(int a, int b)
21. {
22.     int temp = 0;
23.
24.     while (b != 0)
```

```

25.     {
26.         temp += a;
27.         b--;
28.     }
29.     return temp;
30. }
```

```

1. /*
2.  * C Program to Convert Binary Code of a Number into its Equivalent
3.  * Gray's Code using Recursion
4. */
5. #include <stdio.h>
6.
7. int bintogray(int);
8.
9. int main ()
10. {
11.     int bin, gray;
12.
13.     printf("Enter a binary number: ");
14.     scanf("%d", &bin);
15.     gray = bintogray(bin);
16.     printf("The gray code of %d is %d\n", bin, gray);
17.     return 0;
18. }
19.
20. int bintogray(int bin)
21. {
22.     int a, b, result = 0, i = 0;
23.
24.     if (!bin)
25.     {
26.         return 0;
27.     }
28.     else
29.     {
30.         a = bin % 10;
31.         bin = bin / 10;
32.         b = bin % 10;
33.         if ((a && !b) || (!a && b))
34.         {
35.             return (1 + 10 * bintogray(bin));

```

```

36.      }
37.      else
38.      {
39.          return (10 * bintogray(bin));
40.      }
41.  }
42. }
```

#### 4. C Examples on Linked List Implementation without using Recursion

```

1. /*
2.  * C Program to Print the Alternate Nodes in a Linked List without
3.  * using Recursion
4. */
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *next;
12. };
13.
14. void generate(struct node **);
15. void display(struct node *);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     printf("\nDisplaying the alternate nodes\n");
24.     display(head);
25.     delete(&head);
26.
27.     return 0;
28. }
29.
30. void display(struct node *head)
31. {
32.     int flag = 0;
```

```
33.
34.     while(head != NULL)
35.     {
36.         if (!(flag % 2))
37.         {
38.             printf("%d ", head->a);
39.         }
40.         flag++;
41.         head = head->next;
42.     }
43. }
44.
45. void generate(struct node **head)
46. {
47.     int num, i;
48.     struct node *temp;
49.
50.     printf("Enter length of list: ");
51.     scanf("%d", &num);
52.     for (i = num; i > 0; i--)
53.     {
54.         temp = (struct node *)malloc(sizeof(struct node));
55.         temp->a = i;
56.         if (*head == NULL)
57.         {
58.             *head = temp;
59.             (*head)->next = NULL;
60.         }
61.         else
62.         {
63.             temp->next = *head;
64.             *head = temp;
65.         }
66.     }
67. }
68.
69. void delete(struct node **head)
70. {
71.     struct node *temp;
72.     while (*head != NULL)
73.     {
74.         temp = *head;
75.         *head = (*head)->next;
```

```
76.         free(temp);
77.     }
78. }
```

```
1. /*
2. * C Program find the Length of the Linked List without using Recursion
3. */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
8. {
9.     int a;
10.    struct node *next;
11.};
12.
13.
14. void generate(struct node **);
15. int length(struct node*);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.     int count;
22.
23.     generate(&head);
24.     count = length(head);
25.     printf("The number of nodes are: %d\n", count);
26.     delete(&head);
27.
28.     return 0;
29. }
30.
31. void generate(struct node **head)
32. {
33.     /* for unknown number of nodes use num = rand() % 20; */
34.     int num = 10, i;
35.     struct node *temp;
36.
37.     for (i = 0; i < num; i++)
38.     {
```

```

39.         temp = (struct node *)malloc(sizeof(struct node));
40.         temp->a = i;
41.         if (*head == NULL)
42.         {
43.             *head = temp;
44.             (*head)->next = NULL;
45.         }
46.         else
47.         {
48.             temp->next = *head;
49.             *head = temp;
50.         }
51.     }
52. }
53.
54. int length(struct node *head)
55. {
56.     int num = 0;
57.     while (head != NULL)
58.     {
59.         num += 1;
60.         head = head->next;
61.     }
62.     return num;
63. }
64.
65. void delete(struct node **head)
66. {
67.     struct node *temp;
68.     while (*head != NULL)
69.     {
70.         temp = *head;
71.         *head = (*head)->next;
72.         free(temp);
73.     }
74. }
```

```

1. /*
2.  * C Program Count the Number of Occurrences of an Element in the Linked List
3.  * without using Recursion
4.  */
5. #include <stdio.h>
```

```

6.
7. int occur(int [], int, int);
8.
9. int main()
10. {
11.     int size, key, count;
12.     int list[20];
13.     int i;
14.
15.     printf("Enter the size of the list: ");
16.     scanf("%d", &size);
17.     printf("Printing the list:\n");
18.     for (i = 0; i < size; i++)
19.     {
20.         list[i] = rand() % size;
21.         printf("%d    ", list[i]);
22.     }
23.     printf("\nEnter the key to find it's occurrence: ");
24.     scanf("%d", &key);
25.     count = occur(list, size, key);
26.     printf("%d occurs for %d times.\n", key, count);
27.     return 0;
28. }
29.
30. int occur(int list[], int size, int key)
31. {
32.     int i, count = 0;
33.
34.     for (i = 0; i < size; i++)
35.     {
36.         if (list[i] == key)
37.         {
38.             count += 1;
39.         }
40.     }
41.     return count;
42. }
```

```

1. /*
2.  * C Program to Display all the Nodes in a Linked List without using
3.  * Recursion
4. */
```

```
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. struct node
9. {
10.     int a;
11.     struct node *next;
12. };
13.
14. void generate(struct node **);
15. void display(struct node* );
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     display(head);
24.     delete(&head);
25.     return 0;
26. }
27.
28. void generate(struct node **head)
29. {
30.     int num = 10, i;
31.     struct node *temp;
32.
33.     for (i = 0; i < num; i++)
34.     {
35.         temp = (struct node *)malloc(sizeof(struct node));
36.         temp->a = i;
37.         if (*head == NULL)
38.         {
39.             *head = temp;
40.             (*head)->next = NULL;
41.         }
42.         else
43.         {
44.             temp->next = *head;
45.             *head = temp;
46.         }
47.     }
```

```

48. }
49.
50. void display(struct node *head)
51. {
52.     while (head != NULL)
53.     {
54.         printf("%d    ", head->a);
55.         head = head->next;
56.     }
57.     printf("\n");
58. }
59.
60. void delete(struct node **head)
61. {
62.     struct node *temp;
63.     while (*head != NULL)
64.     {
65.         temp = *head;
66.         *head = (*head)->next;
67.         free(temp);
68.     }
69. }
```

```

1. /*
2.  * C Program to Display the Nodes of a Linked List in Reverse without
3.  * using Recursion
4. */
5.
6. #include <stdio.h>
7. #include <stdlib.h>
8.
9. struct node
10. {
11.     int visited;
12.     int a;
13.     struct node *next;
14. };
15.
16. void generate(struct node **);
17. void display(struct node *);
18. void linear(struct node *);
19. void delete(struct node **);
```

```
20.
21. int main()
22. {
23.     struct node *head = NULL;
24.
25.     generate(&head);
26.     printf("\nPrinting the list in linear order\n");
27.     linear(head);
28.     printf("\nPrinting the list in reverse order\n");
29.     display(head);
30.     delete(&head);
31.
32.     return 0;
33. }
34.
35. void display(struct node *head)
36. {
37.     struct node *temp = head, *prev = head;
38.
39.     while (temp->visited == 0)
40.     {
41.         while (temp->next != NULL && temp->next->visited == 0)
42.         {
43.             temp = temp->next;
44.         }
45.         printf("%d ", temp->a);
46.         temp->visited = 1;
47.         temp = head;
48.     }
49. }
50.
51. void linear(struct node *head)
52. {
53.     while (head != NULL)
54.     {
55.         printf("%d ", head->a);
56.         head = head->next;
57.     }
58.     printf("\n");
59. }
60.
61. void generate(struct node **head)
62. {
```

```

63.     int num, i;
64.     struct node *temp;
65.
66.     printf("Enter length of list: ");
67.     scanf("%d", &num);
68.     for (i = num; i > 0; i--)
69.     {
70.         temp = (struct node *)malloc(sizeof(struct node));
71.         temp->a = i;
72.         temp->visited = 0;
73.         if (*head == NULL)
74.         {
75.             *head = temp;
76.             (*head)->next = NULL;
77.         }
78.         else
79.         {
80.             temp->next = *head;
81.             *head = temp;
82.         }
83.     }
84. }
85.
86. void delete(struct node **head)
87. {
88.     struct node *temp;
89.     while (*head != NULL)
90.     {
91.         temp = *head;
92.         *head = (*head)->next;
93.         free(temp);
94.     }
95. }
```

```

1. /*
2.  * C Program to Search for an Element in the Linked List without
3.  * using Recursion
4. */
5.
6. #include <stdio.h>
7. #include <stdlib.h>
8.
```

```
9. struct node
10. {
11.     int a;
12.     struct node *next;
13. };
14.
15. void generate(struct node **, int);
16. void search(struct node *, int);
17. void delete(struct node **);
18.
19. int main()
20. {
21.     struct node *head = NULL;
22.     int key, num;
23.
24.     printf("Enter the number of nodes: ");
25.     scanf("%d", &num);
26.     printf("\nDisplaying the list\n");
27.     generate(&head, num);
28.     printf("\nEnter key to search: ");
29.     scanf("%d", &key);
30.     search(head, key);
31.     delete(&head);
32.
33.     return 0;
34. }
35.
36. void generate(struct node **head, int num)
37. {
38.     int i;
39.     struct node *temp;
40.
41.     for (i = 0; i < num; i++)
42.     {
43.         temp = (struct node *)malloc(sizeof(struct node));
44.         temp->a = rand() % num;
45.         if (*head == NULL)
46.         {
47.             *head = temp;
48.             temp->next = NULL;
49.         }
50.         else
51.         {
```

```

52.         temp->next = *head;
53.         *head = temp;
54.     }
55.     printf("%d ", temp->a);
56. }
57. }
58.
59. void search(struct node *head, int key)
60. {
61.     while (head != NULL)
62.     {
63.         if (head->a == key)
64.         {
65.             printf("key found\n");
66.             return;
67.         }
68.         head = head->next;
69.     }
70.     printf("Key not found\n");
71. }
72.
73. void delete(struct node **head)
74. {
75.     struct node *temp;
76.
77.     while (*head != NULL)
78.     {
79.         temp = *head;
80.         *head = (*head)->next;
81.         free(temp);
82.     }
83. }
```

## 5. C Examples on Stack Reversal without using Recursion

```

1. /*
2.  * C Program to Reverse a Stack without using Recursion
3.  */
4. #include <stdio.h>
5. #include <stdlib.h>
6.
7. struct node
```

```
8. {
9.     int a;
10.    struct node *next;
11. };
12.
13. void generate(struct node **);
14. void display(struct node *);
15. void stack_reverse(struct node **);
16. void delete(struct node **);
17.
18. int main()
19. {
20.     struct node *head = NULL;
21.
22.     generate(&head);
23.     printf("\nThe sequence of contents in stack\n");
24.     display(head);
25.     printf("\nInversing the contents of the stack\n");
26.     stack_reverse(&head);
27.     printf("\nThe contents in stack after reversal\n");
28.     display(head);
29.     delete(&head);
30.     return 0;
31. }
32.
33. void stack_reverse(struct node **head)
34. {
35.     struct node *temp, *prev;
36.
37.     if (*head == NULL)
38.     {
39.         printf("Stack does not exist\n");
40.     }
41.     else if ((*head)->next == NULL)
42.     {
43.         printf("Single node stack reversal brings no difference\n");
44.     }
45.     else if ((*head)->next->next == NULL)
46.     {
47.         (*head)->next->next = *head;
48.         *head = (*head)->next;
49.         (*head)->next->next = NULL;
50.     }
}
```

```

51.     else
52.     {
53.         prev = *head;
54.         temp = (*head)->next;
55.         *head = (*head)->next->next;
56.         prev->next = NULL;
57.         while ((*head)->next != NULL)
58.         {
59.             temp->next = prev;
60.             prev = temp;
61.             temp = *head;
62.             *head = (*head)->next;
63.         }
64.         temp->next = prev;
65.         (*head)->next = temp;
66.     }
67. }
68.
69. void display(struct node *head)
70. {
71.     if (head != NULL)
72.     {
73.         printf("%d ", head->a);
74.         display(head->next);
75.     }
76. }
77.
78. void generate(struct node **head)
79. {
80.     int num, i;
81.     struct node *temp;
82.
83.     printf("Enter length of list: ");
84.     scanf("%d", &num);
85.     for (i = num; i > 0; i--)
86.     {
87.         temp = (struct node *)malloc(sizeof(struct node));
88.         temp->a = i;
89.         if (*head == NULL)
90.         {
91.             *head = temp;
92.             (*head)->next = NULL;
93.         }

```

```

94.     else
95.     {
96.         temp->next = *head;
97.         *head = temp;
98.     }
99. }
100. }
101.
102. void delete(struct node **head)
103. {
104.     struct node *temp;
105.     while (*head != NULL)
106.     {
107.         temp = *head;
108.         *head = (*head)->next;
109.         free(temp);
110.     }
111. }
```

```

1. /*
2.  * C Program to find the First Capital Letter in a String without
3.  * using Recursion
4. */
5. #include <stdio.h>
6. #include <string.h>
7. #include <ctype.h>
8.
9. char caps_check(char *);
10.
11.int main()
12.{  

13.    char string[20], letter;  

14.
15.    printf("Enter a string to find it's first capital letter: ");
16.    scanf("%s", string);
17.    letter = caps_check(string);
18.    if (letter == 0)
19.    {
20.        printf("No capital letter is present in %s.\n", string);
21.    }
22.    else
23.    {
```

```

24.     printf("The first capital letter in %s is %c.\n", string, letter);    }
25.     return 0;
26. }
27. char caps_check(char *string)
28. {
29.     int i = 0;
30.     while (string[i] != '\0')
31.     {
32.         if (isupper(string[i]))
33.         {
34.             return string[i];
35.         }
36.         i++;
37.     }
38.     return 0;
39. }
```

```

1. /*
2.  * C Program to sort an array based on heap sort algorithm(MAX heap)
3. */
4. #include <stdio.h>
5.
6. void main()
7. {
8.     int heap[10], no, i, j, c, root, temp;
9.
10.    printf("\n Enter no of elements :");
11.    scanf("%d", &no);
12.    printf("\n Enter the nos : ");
13.    for (i = 0; i < no; i++)
14.        scanf("%d", &heap[i]);
15.    for (i = 1; i < no; i++)
16.    {
17.        c = i;
18.        do
19.        {
20.            root = (c - 1) / 2;
21.            if (heap[root] < heap[c]) /* to create MAX heap array */
22.            {
23.                temp = heap[root];
24.                heap[root] = heap[c];
25.                heap[c] = temp;
```

```

26.         }
27.         c = root;
28.     } while (c != 0);
29. }
30.
31. printf("Heap array : ");
32. for (i = 0; i < no; i++)
33.     printf("%d\t ", heap[i]);
34. for (j = no - 1; j >= 0; j--)
35. {
36.     temp = heap[0];
37.     heap[0] = heap[j] /* swap max element with rightmost leaf element */
38.     heap[j] = temp;
39.     root = 0;
40.     do
41.     {
42.         c = 2 * root + 1; /* Left node of root element */
43.         if ((heap[c] < heap[c + 1]) && c < j-1)
44.             c++;
45.         if (heap[root]<heap[c] && c<j) /* again rearrange to max heap array
*/
46.         {
47.             temp = heap[root];
48.             heap[root] = heap[c];
49.             heap[c] = temp;
50.         }
51.         root = c;
52.     } while (c < j);
53. }
54. printf("\n The sorted array is : ");
55. for (i = 0; i < no; i++)
56.     printf("\t %d", heap[i]);
57. printf("\n Complexity : \n Best case = Avg case = Worst case = O(n logn) \n");
58. }

```

## C Program to Create a Random Graph Using Random Edge Generation

```

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include <time.h>

```

```

4.
5. #define MAX_VERTICES 30
6. #define MAX_EDGES 10
7.
8. typedef unsigned char vertex;
9.
10.int main(){
11.
12.     /*number of nodes in a graph*/
13.     srand ( time(NULL) );
14.     int numberOfVertices = rand() % MAX_VERTICES;
15.
16.     /*number of maximum edges a vertex can have*/
17.     srand ( time(NULL) );
18.     int maxNumberOfEdges = rand() % MAX_EDGES;
19.     /*graphs is 2 dimensional array of pointers*/
20.     if( numberOfVertices == 0)
21.         numberOfVertices++;
22.     vertex ***graph;
23.     printf("Total Vertices = %d, Max # of Edges = %d\n",numberOfVertices,
24.     maxNumberOfEdges);
24.
25.     /*generate a dynamic array of random size*/
26.     if ((graph = (vertex ***) malloc(sizeof(vertex **) * numberOfVertices)) ==
27.         NULL){
27.         printf("Could not allocate memory for graph\n");
28.         exit(1);
29.     }
30.
31.     /*generate space for edges*/
32.     int vertexCounter = 0;
33.     /*generate space for vertices*/
34.     int edgeCounter = 0;
35.
36.     for (vertexCounter = 0; vertexCounter < numberOfVertices; vertexCounter++){
37.         if ((graph[vertexCounter] = (vertex **) malloc(sizeof(vertex *) *
38.             maxNumberOfEdges)) == NULL){
38.             printf("Could not allocate memory for edges\n");
39.             exit(1);
40.         }
41.         for (edgeCounter = 0; edgeCounter < maxNumberOfEdges; edgeCounter++){
42.             if ((graph[vertexCounter][edgeCounter] = (vertex *

```

```

43.             printf("Could not allocate memory for vertex\n");
44.             exit(1);
45.         }
46.     }
47. }
48.
49. /*start Linking the graph. All vetrices need not have same number of Links*/
50. vertexCounter = 0; edgeCounter = 0;
51. for (vertexCounter = 0; vertexCounter < numberOfVertices; vertexCounter++){
52.     printf("%d:\t", vertexCounter);
53.     for (edgeCounter=0; edgeCounter < maxNumberOfEdges; edgeCounter++){
54.         if (rand()%2 == 1){ /*Link the vertices*/
55.             int linkedVertex = rand() % numberOfVertices;
56.             graph[vertexCounter][edgeCounter] = graph[linkedVertex];
57.             printf("%d, ", linkedVertex);
58.         }
59.         else{ /*make the link NULL*/
60.             graph[vertexCounter][edgeCounter] = NULL;
61.         }
62.     }
63.     printf("\n");
64. }
65. return 1;
66. }
```

## C Program to Implement Hash Tables chaining with Singly Linked Lists

```

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4.
5. struct hash *hashTable = NULL;
6. int eleCount = 0;
7.
8. struct node {
9.     int key, age;
10.    char name[100];
11.    struct node *next;
12. };
13.
```

```

14. struct hash {
15.     struct node *head;
16.     int count;
17. };
18.
19. struct node * createNode(int key, char *name, int age) {
20.     struct node *newnode;
21.     newnode = (struct node *) malloc(sizeof(struct node));
22.     newnode->key = key;
23.     newnode->age = age;
24.     strcpy(newnode->name, name);
25.     newnode->next = NULL;
26.     return newnode;
27. }
28.
29. void insertToHash(int key, char *name, int age) {
30.     int hashIndex = key % eleCount;
31.     struct node *newnode = createNode(key, name, age);
32.     /* head of list for the bucket with index "hashIndex" */
33.     if (!hashTable[hashIndex].head) {
34.         hashTable[hashIndex].head = newnode;
35.         hashTable[hashIndex].count = 1;
36.         return;
37.     }
38.     /* adding new node to the list */
39.     newnode->next = (hashTable[hashIndex].head);
40.     /*
41.      * update the head of the list and no of
42.      * nodes in the current bucket
43.      */
44.     hashTable[hashIndex].head = newnode;
45.     hashTable[hashIndex].count++;
46.     return;
47. }
48.
49. void deleteFromHash(int key) {
50.     /* find the bucket using hash index */
51.     int hashIndex = key % eleCount, flag = 0;
52.     struct node *temp, *myNode;
53.     /* get the list head from current bucket */
54.     myNode = hashTable[hashIndex].head;
55.     if (!myNode) {
56.         printf("Given data is not present in hash Table! !\n");

```

```

57.         return;
58.     }
59.     temp = myNode;
60.     while (myNode != NULL) {
61.         /* delete the node with given key */
62.         if (myNode->key == key) {
63.             flag = 1;
64.             if (myNode == hashTable[hashIndex].head)
65.                 hashTable[hashIndex].head = myNode->next;
66.             else
67.                 temp->next = myNode->next;
68.
69.             hashTable[hashIndex].count--;
70.             free(myNode);
71.             break;
72.         }
73.         temp = myNode;
74.         myNode = myNode->next;
75.     }
76.     if (flag)
77.         printf("Data deleted successfully from Hash Table\n");
78.     else
79.         printf("Given data is not present in hash Table!!!!\n");
80.     return;
81. }
82.
83. void searchInHash(int key) {
84.     int hashIndex = key % eleCount, flag = 0;
85.     struct node *myNode;
86.     myNode = hashTable[hashIndex].head;
87.     if (!myNode) {
88.         printf("Search element unavailable in hash table\n");
89.         return;
90.     }
91.     while (myNode != NULL) {
92.         if (myNode->key == key) {
93.             printf("VoterID : %d\n", myNode->key);
94.             printf("Name    : %s\n", myNode->name);
95.             printf("Age     : %d\n", myNode->age);
96.             flag = 1;
97.             break;
98.         }
99.         myNode = myNode->next;

```

```

100.        }
101.        if (!flag)
102.            printf("Search element unavailable in hash table\n");
103.        return;
104.    }
105.
106.    void display() {
107.        struct node *myNode;
108.        int i;
109.        for (i = 0; i < eleCount; i++) {
110.            if (hashTable[i].count == 0)
111.                continue;
112.            myNode = hashTable[i].head;
113.            if (!myNode)
114.                continue;
115.            printf("\nData at index %d in Hash Table:\n", i);
116.            printf("VoterID      Name          Age   \n");
117.            printf("-----\n");
118.            while (myNode != NULL) {
119.                printf("%-12d", myNode->key);
120.                printf("%-15s", myNode->name);
121.                printf("%d\n", myNode->age);
122.                myNode = myNode->next;
123.            }
124.        }
125.        return;
126.    }
127.
128.    int main() {
129.        int n, ch, key, age;
130.        char name[100];
131.        printf("Enter the number of elements:");
132.        scanf("%d", &n);
133.        eleCount = n;
134.        /* create hash table with "n" no of buckets */
135.        hashTable = (struct hash *) calloc(n, sizeof(struct hash));
136.        while (1) {
137.            printf("\n1. Insertion\t2. Deletion\n");
138.            printf("3. Searching\t4. Display\n5. Exit\n");
139.            printf("Enter your choice:");
140.            scanf("%d", &ch);
141.            switch (ch) {
142.                case 1:

```

```

143.         printf("Enter the key value:");
144.         scanf("%d", &key);
145.         getchar();
146.         printf("Name:");
147.         fgets(name, 100, stdin);
148.         name[strlen(name) - 1] = '\0';
149.         printf("Age:");
150.         scanf("%d", &age);
151.         /*inserting new node to hash table */
152.         insertToHash(key, name, age);
153.         break;
154.
155.     case 2:
156.         printf("Enter the key to perform deletion:");
157.         scanf("%d", &key);
158.         /* delete node with "key" from hash table */
159.         deleteFromHash(key);
160.         break;
161.
162.     case 3:
163.         printf("Enter the key to search:");
164.         scanf("%d", &key);
165.         searchInHash(key);
166.         break;
167.     case 4:
168.         display();
169.         break;
170.     case 5:
171.         exit(0);
172.     default:
173.         printf("U have entered wrong option!!\n");
174.         break;
175.     }
176. }
177. return 0;
178. }
```

## C Program to Implement a Heap & provide Insertion & Deletion Operation

1. /\*

```
2.  * C Program to Implement a Heap & provide Insertion & Deletion Operation
3.  */
4. #include <stdio.h>
5.
6. int array[100], n;
7. main()
8. {
9.     int choice, num;
10.    n = 0; /*Represents number of nodes in the heap*/
11.    while(1)
12.    {
13.        printf("1.Insert the element \n");
14.        printf("2.Delete the element \n");
15.        printf("3.Display all elements \n");
16.        printf("4.Quit \n");
17.        printf("Enter your choice : ");
18.        scanf("%d", &choice);
19.        switch(choice)
20.        {
21.            case 1:
22.                printf("Enter the element to be inserted to the list : ");
23.                scanf("%d", &num);
24.                insert(num, n);
25.                n = n + 1;
26.                break;
27.            case 2:
28.                printf("Enter the elements to be deleted from the list: ");
29.                scanf("%d", &num);
30.                delete(num);
31.                break;
32.            case 3:
33.                display();
34.                break;
35.            case 4:
36.                exit(0);
37.            default:
38.                printf("Invalid choice \n");
39.        }/*End of switch */
40.    }/*End of while */
41. }/*End of main()*/
42.
43.display()
44.{
```

```

45.     int i;
46.     if (n == 0)
47.     {
48.         printf("Heap is empty \n");
49.         return;
50.     }
51.     for (i = 0; i < n; i++)
52.         printf("%d ", array[i]);
53.     printf("\n");
54. }/*End of display()*/
55.
56.insert(int num, int location)
57.{
58.    int parentnode;
59.    while (location > 0)
60.    {
61.        parentnode =(location - 1)/2;
62.        if (num <= array[parentnode])
63.        {
64.            array[location] = num;
65.            return;
66.        }
67.        array[location] = array[parentnode];
68.        location = parentnode;
69.    }/*End of while*/
70.    array[0] = num; /*assign number to the root node */
71.}/*End of insert()*/
72.
73.delete(int num)
74.{
75.    int left, right, i, temp, parentnode;
76.
77.    for (i = 0; i < num; i++) {
78.        if (num == array[i])
79.            break;
80.    }
81.    if (num != array[i])
82.    {
83.        printf("%d not found in heap list\n", num);
84.        return;
85.    }
86.    array[i] = array[n - 1];
87.    n = n - 1;

```

```
88.     parentnode = (i - 1) / 2; /*find parentnode of node i */
89.     if (array[i] > array[parentnode])
90.     {
91.         insert(array[i], i);
92.         return;
93.     }
94.     left = 2 * i + 1; /*Left child of i*/
95.     right = 2 * i + 2; /* right child of i*/
96.     while (right < n)
97.     {
98.         if (array[i] >= array[left] && array[i] >= array[right])
99.             return;
100.            if (array[right] <= array[left])
101.            {
102.                temp = array[i];
103.                array[i] = array[left];
104.                array[left] = temp;
105.                i = left;
106.            }
107.            else
108.            {
109.                temp = array[i];
110.                array[i] = array[right];
111.                array[right] = temp;
112.                i = right;
113.            }
114.            left = 2 * i + 1;
115.            right = 2 * i + 2;
116.        }/*End of while*/
117.        if (left == n - 1 && array[i])    {
118.            temp = array[i];
119.            array[i] = array[left];
120.            array[left] = temp;
121.        }
122.    }
```