**List of 50 C Coding Interview Questions and Answer**

Here is a list of 50 C coding interview questions and answers:

**1. Find the largest number among the three numbers.**

* C

|  |
| --- |
| // C Program to find  // Largest of three numbers  #include <stdio.h>    **int** main()  {  **int** a = 1, b = 2, c = 3;        // condition for a is greatest  **if** (a > b && a > c)  **printf**("%d", a);        // condition for b is greatest  **else** **if** (b > a && b > c)  **printf**("%d", b);        // remaining conditions      // c is greatest  **else**  **printf**("%d", c);    **return** 0;  } |

**Output**

3

**2. Write a Program to check whether a number is prime or not.**

* C

|  |
| --- |
| // C Program for  // Checking value is  // Prime or not  #include <stdio.h>    **int** main()  {  **int** N = 91;  **int** flag = 0;        // Iterate from 2 to N/2  **for** (**int** i = 2; i <= N / 2; i++) {            // If n is divisible by any number between 2 and          // n/2, it is not prime  **if** (N % i == 0) {              flag = 1;  **break**;          }      }    **if** (flag == 0)  **printf**("Not a Prime Number");  **else**  **printf**("Is a Prime Number");  **return** 0;  } |

**Output**

Is a Prime Number

**3. Write a C program to calculate Compound Interest.**

* C

|  |
| --- |
| // C program to calculate Compound Interest  #include <stdio.h>    // For using pow function we must  // include math.h  #include <math.h>    // Driver code  **int** main()  {      // Principal amount  **double** principal = 2300;        // Annual rate of interest  **double** rate = 7;        // Time  **double** **time** = 4;        // Calculating compound Interest  **double** amount          = principal \* ((**pow**((1 + rate / 100), **time**)));  **double** CI = amount - principal;    **printf**("Compound Interest is : %lf", CI);  **return** 0;  } |

**Output**

Compound Interest is : 714.830823

**4. Write a Program in C to Swap the values of two variables without using any extra variable.**

* C

|  |
| --- |
| // C Program to  // Swap two numbers  // No Extra Space  #include <stdio.h>    **int** main()  {    **int** x = 10;  **int** y = 20;    **printf**("x: %d , y: %d\n", x, y);        // Code to swap 'x' and 'y'      x = x + y;      y = x - y;      x = x - y;    **printf**("x: %d , y: %d\n", x, y);    **return** 0;  } |

**Output**

x: 10 , y: 20

x: 20 , y: 10

**5. Write a Program to Replace all 0’s with 1’s in a Number.**

* C

|  |
| --- |
| // C Program for  // Replacing 0 to 1  #include <math.h>  #include <stdio.h>    **int** main()  {  **int** N = 102301;    **int** ans = 0;  **int** i = 0;  **while** (N != 0) {          // Condition to change value  **if** (N % 10 == 0)              ans = ans + 1 \* **pow**(10, i);  **else**              ans = ans + (N % 10) \* **pow**(10, i);            N = N / 10;          i++;      }    **printf**("%d", ans);    **return** 0;  } |

**Output:**

112311

**6. Write a Program to convert the binary number into a decimal number.**

* C

|  |
| --- |
| // C Program for converting  // binary to decimal  #include <stdio.h>    **int** main()  {  **int** N = 11011;        // Initializing base value a to 1  **int** a = 1;  **int** ans = 0;  **while** (N != 0) {          ans = ans + (N % 10) \* a;          N = N / 10;          a = a \* 2;      }    **printf**("%d", ans);  **return** 0;  } |

**Output**

27

**7.  Write a Program to check if the year is a leap year or not.**

* C

|  |
| --- |
| // C Program to check  // Year is leap year or not  #include <stdio.h>    // Function Declaration to check leap year  **void** leap\_year(**int** year)  {      // If a year is multiple of 400, then leap year  **if** (year % 400 == 0)  **printf**("%d is a leap year.\n", year);        // If a year is multiple of 100, then not a leap year  **else** **if** (year % 100 == 0)  **printf**("%d is not a leap year.\n", year);        // If a year is multiple of 4, then leap year  **else** **if** (year % 4 == 0)  **printf**("%d is a leap year.\n", year);        // Not leap year  **else**  **printf**("%d is not a leap year.\n", year);  }    **int** main()  {      leap\_year(2000);      leap\_year(2002);      leap\_year(2008);    **return** 0;  } |

**Output**

2000 is a leap year.

2002 is not a leap year.

2008 is a leap year.

**8. Write a program to Factorial of a Number.**

* C

|  |
| --- |
| // C Program to calculate  // Factorial of a number  #include <stdio.h>    // Calculating factorial using iteration  **void** factorial\_iteration(**int** N)  {      unsigned **long** **long** **int** ans = 1;  **for** (**int** i = 1; i <= N; i++) {          ans = ans \* i;      }    **printf**("Factorial of %d is %lld\n", N, ans);  }    // Calculating factorial using recursion  **int** factorial(**int** N)  {  **if** (N == 0)  **return** 1;        // Recursive call  **return** N \* factorial(N - 1);  }    **int** main()  {  **int** n;      n = 13;      factorial\_iteration(n);        n = 9;  **printf**("Factorial of %d using recursion:%d\n", n,             factorial(n));    **return** 0;  } |

**Output**

Factorial of 13 is 6227020800

Factorial of 9 using recursion:362880

**9. Write a Program to Check if a number is an Armstrong number or not.**

* C

|  |
| --- |
| // C program to check if number  // is Armstrong number or not  #include <stdio.h>    // Function to calculate x raised to the power y  **int** power(**int** x, unsigned **int** y)  {  **if** (y == 0)  **return** 1;  **if** (y % 2 == 0)  **return** power(x, y / 2) \* power(x, y / 2);    **return** x \* power(x, y / 2) \* power(x, y / 2);  }    // Function to calculate order of the number  **int** order(**int** n)  {  **int** res = 0;  **while** (n) {          res++;          n = n / 10;      }  **return** res;  }    // Function to check whether the given number is  // Armstrong number or not  **int** isArmstrong(**int** x)  {      // Calling order function  **int** n = order(x);  **int** temp = x, sum = 0;  **while** (temp) {  **int** r = temp % 10;          sum += power(r, n);          temp = temp / 10;      }        // If satisfies Armstrong condition  **if** (sum == x)  **return** 1;  **else**  **return** 0;  }    // Driver Program  **int** main()  {  **int** x = 120;  **if** (isArmstrong(x) == 1)  **printf**("True\n");  **else**  **printf**("False\n");        x = 1634;  **if** (isArmstrong(x) == 1)  **printf**("True\n");  **else**  **printf**("False\n");    **return** 0;  } |

**Output**

False

True

**10. Write a program to Find all the roots of a quadratic equation in C.**

* C

|  |
| --- |
| // C program to find roots  // of a quadratic equation  #include <math.h>  #include <stdio.h>  #include <stdlib.h>    // Prints roots of quadratic equation ax\*2 + bx + x  **void** find\_roots(**int** a, **int** b, **int** c)  {      // If a is 0, then equation is not quadratic, but      // linear  **if** (a == 0) {  **printf**("Invalid");  **return**;      }    **int** d = (b \* b) - (4 \* a \* c);  **double** sqrt\_val = **sqrt**(**abs**(d));    **if** (d > 0) {  **printf**("Roots are real and different \n");  **printf**("%f\n%f", (**double**)(-b + sqrt\_val) / (2 \* a),                 (**double**)(-b - sqrt\_val) / (2 \* a));      }  **else** **if** (d == 0) {  **printf**("Roots are real and same \n");  **printf**("%f", -(**double**)b / (2 \* a));      }  **else** // d < 0      {  **printf**("Roots are complex \n");  **printf**("%f + i%f\n%f - i%f", -(**double**)b / (2 \* a),                 sqrt\_val / (2 \* a), -(**double**)b / (2 \* a),                 sqrt\_val / (2 \* a));      }  }    // Driver code  **int** main()  {  **int** a = 1, b = -16, c = 1;        // Function call      find\_roots(a, b, c);  **return** 0;  } |

**Output:**

Roots are real and different  
15.937254  
0.062746

**11. Write a Program to reverse a number.**

* C

|  |
| --- |
| // C Programs to Calculate  // reverse of a number  #include <stdio.h>    // Iterative approach  **int** reverse\_iteration(**int** N)  {  **int** ans = 0;  **while** (N != 0) {            ans = ans \* 10 + (N % 10);          N = N / 10;      }    **return** ans;  }    // recursive approach  **int** reverse(**int** n, **int** ans)  {  **if** (n == 0)  **return** ans;        ans = ans \* 10 + n % 10;  **return** reverse(n / 10, ans);  }    **int** main()  {  **int** N = 15942;  **printf**("Initial number:%d\n", N);        N = reverse\_iteration(N);  **printf**("%d after reverse using iteration\n", N);    **int** ans = 0;      ans = reverse(N, ans);  **printf**("%d after again reverse using recursion", ans);    **return** 0;  } |

**Output**

Initial number:15942

24951 after reverse using iteration

15942 after again reverse using recursion

**12. Check whether a number is a palindrome.**

* C

|  |
| --- |
| // C Program for  // Checking Palindrome  #include <stdio.h>    // Checking if the number is  // Palindrome number  **void** check\_palindrome(**int** N)  {  **int** T = N;  **int** rev = 0; // This variable stored reversed digit        // Execute a while loop to reverse digits of given      // number  **while** (T != 0) {          rev = rev \* 10 + T % 10;          T = T / 10;      }        // Compare original\_number with reversed number  **if** (rev == N)  **printf**("%d is palindrome\n", N);  **else**  **printf**("%d is not a palindrome\n", N);  }    **int** main()  {  **int** N = 13431;  **int** M = 12345;        // Function call      check\_palindrome(N);      check\_palindrome(M);    **return** 0;  } |

**Output**

13431 is palindrome

12345 is not a palindrome

**13. Write a C Program to check if two numbers are equal without using the bitwise operator.**

* C

|  |
| --- |
| // C Program for checking numbers  // are equal using bitwise operator  #include <stdio.h>    **int** main()  {  **int** x = 1;  **int** y = 2;        // Using XOR      // XOR of two equal numbers is 0  **if** (!(x ^ y))  **printf**(" %d is equal to %d ", x, y);  **else**  **printf**(" %d is not equal to %d ", x, y);    **return** 0;  } |

**Output**

1 is not equal to 2

**14. Write a  C program to find the GCD of two numbers.**

* C

|  |
| --- |
| // C program to find GCD of two numbers  #include <math.h>  #include <stdio.h>    // Function to return gcd of a and b  **int** gcd(**int** a, **int** b)  {      // Find Minimum of a and b  **int** result = ((a < b) ? a : b);  **while** (result > 0) {  **if** (a % result == 0 && b % result == 0) {  **break**;          }          result--;      }  **return** result; // return gcd of a and b  }    // Driver program to test above function  **int** main()  {  **int** a = 98, b = 56;  **printf**("GCD of %d and %d is %d ", a, b, gcd(a, b));  **return** 0;  } |

**Output**

GCD of 98 and 56 is 14

**15. Write a  C program to find the LCM of two numbers.**

* C

|  |
| --- |
| // C program to find  // LCM of two numbers  #include <stdio.h>    // minimum of two numbers  **int** Min(**int** Num1, **int** Num2)  {  **if** (Num1 >= Num2)  **return** Num2;  **else**  **return** Num1;  }    **int** LCM(**int** Num1, **int** Num2, **int** K)  {      // If either of the two numbers      // is 1, return their product  **if** (Num1 == 1 || Num2 == 1)  **return** Num1 \* Num2;        // If both the numbers are equal  **if** (Num1 == Num2)  **return** Num1;        // If K is smaller than the      // minimum of the two numbers  **if** (K <= Min(Num1, Num2)) {            // Checks if both numbers are          // divisible by K or not  **if** (Num1 % K == 0 && Num2 % K == 0) {                // Recursively call LCM() function  **return** K \* LCM(Num1 / K, Num2 / K, 2);          }            // Otherwise  **else**  **return** LCM(Num1, Num2, K + 1);      }        // If K exceeds minimum  **else**  **return** Num1 \* Num2;  }    **int** main()  {      // Given N & M  **int** N = 12, M = 9;        // Function Call  **int** ans = LCM(N, M, 2);    **printf**("%d", ans);    **return** 0;  } |

**Output**

36

**16. Write a C Program to find the Maximum and minimum of two numbers without using any loop or condition.**

* C

|  |
| --- |
| // C Program to check  // Maximum and Minimum  // Between two numbers  // Without any condition or loop  #include <stdio.h>  #include <stdlib.h>    **int** main()  {  **int** a = 55, b = 23;        // return maximum among the two numbers  **printf**("max = %d\n", ((a + b) + **abs**(a - b)) / 2);        // return minimum among the two numbers  **printf**("min = %d", ((a + b) - **abs**(a - b)) / 2);    **return** 0;  } |

**Output**

max = 55

min = 23

**17. Write a Program in C to Print all natural numbers up to N without using a semi-colon.**

* C

|  |
| --- |
| // C program to print  // all natural numbers  // upto N without using semi-colon  #include <stdio.h>  #define N 10    **int** main(**int** val)  {  **if** (val <= N && **printf**("%d ", val) && main(val + 1)) {      }  } |

**Output**

1 2 3 4 5 6 7 8 9 10

**18. Write a Program to find the area of a circle.**

* C

|  |
| --- |
| // C program to find area  // of circle  #include <math.h>  #include <stdio.h>  #define PI 3.142    **double** findArea(**int** r) { **return** PI \* **pow**(r, 2); }    **int** main()  {  **printf**("Area is %f", findArea(5));  **return** 0;  } |

**Output**

Area is 78.550000

**19.  Write a Program to create a pyramid pattern using C.**

* C

|  |
| --- |
| // C Program print Pyramid pattern  #include <stdio.h>    **int** main()  {  **int** N = 5;        // Outer Loop for number of rows  **for** (**int** i = 1; i <= N; i++) {            // inner Loop for space printing  **for** (**int** j = 1; j <= N - i; j++)  **printf**(" ");            // inner Loop for star printing  **for** (**int** j = 1; j < 2 \* i; j++)  **printf**("\*");  **printf**("\n");      }  **return** 0;  } |

**Output**

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

**20. Write a program to form Pascal Triangle using numbers.**

1   
 1 1   
 1 2 1   
 1 3 3 1   
1 4 6 4 1

* C

|  |
| --- |
| // C Program to print  // Pascal's Triangle  #include <stdio.h>    **int** main()  {  **int** n = 5;      **for** (**int** i = 1; i <= n; i++) {  **for** (**int** j = 1; j <= n - i; j++) {  **printf**("  ");          }    **int** x = 1;    **for** (**int** j = 1; j <= i; j++) {  **printf**("%d   ", x);              x = x \* (i - j) / j;          }  **printf**("\n");      }    **return** 0;  } |

**Output**

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

**21. Write a Program to return the nth row of Pascal’s triangle.**

* C

|  |
| --- |
| // C program to return the Nth row of pascal's triangle  #include <stdio.h>    // Print the N-th row of the Pascal's Triangle  **void** generateNthrow(**int** N)  {      // nC0 = 1  **int** prev = 1;  **printf**("%d", prev);    **for** (**int** i = 1; i <= N; i++) {          // nCr = (nCr-1 \* (n - r + 1))/r  **int** curr = (prev \* (N - i + 1)) / i;  **printf**(",%d ", curr);          prev = curr;      }  }    **int** main()  {  **int** n = 5;      generateNthrow(n);  **return** 0;  } |

**Output**

1,5 ,10 ,10 ,5 ,1

**22. Write a program to reverse an Array.**

* C

|  |
| --- |
| // C Program to reverse  // An array  #include <stdio.h>    **void** reverse(**int**\* arr, **int** n)  {      // Swapping front and back elements.  **for** (**int** i = 0, j = n - 1; i < j; i++, j--) {  **int** ele = arr[i];          arr[i] = arr[j];          arr[j] = ele;      }  }    **int** main()  {    **int** arr[] = { 1, 2, 3, 4, 5 };      // Function Call      reverse(arr, 5);        // reverse array element printing  **for** (**int** i = 0; i < 5; i++)  **printf**("%d ", arr[i]);    **return** 0;  } |

**Output**

5 4 3 2 1

**23. Write a program to check the repeating elements in C.**

* C

|  |
| --- |
| // C Program for  // checking duplicate  // values in a array  #include <stdio.h>    **int** Sort(**int** arr[], **int** size)  {  **for** (**int** i = 0; i < size - 1; i++) {    **for** (**int** j = 0; j < size - i - 1; j++) {  **if** (arr[j] > arr[j + 1]) {  **int** temp = arr[j];                  arr[j] = arr[j + 1];                  arr[j + 1] = temp;              }          }      }  }    // find repeating element  **void** findRepeating(**int** arr[], **int** n)  {  **int** count = 0;  **for** (**int** i = 0; i < n; i++) {    **int** flag = 0;  **while** (i < n - 1 && arr[i] == arr[i + 1]) {              flag = 1;              i++;          }  **if** (flag)  **printf**("%d ", (arr[i - 1]));      }    **return**;  }    **int** main()  {  **int** arr[] = { 1, 3, 4, 1, 2, 3, 5, 5 };    **int** n = **sizeof**(arr) / **sizeof**(arr[0]);          Sort(arr,n);          findRepeating(arr,n);        **return** 0;  } |

**Output**

1 3 5

**24. Write a Program to print the Maximum and Minimum elements in an array.**

* C

|  |
| --- |
| // C Program for calculating  // maximum and minimum element  #include <stdio.h>    **void** find\_small\_large(**int** arr[], **int** n)  {  **int** min, max;        // assign first element as minimum and maximum      min = arr[0];      max = arr[0];    **for** (**int** i = 1; i < n; i++) {            // finding smallest here  **if** (arr[i] < min)              min = arr[i]; // finding largest here  **if** (arr[i] > max)              max = arr[i];      }  **printf**("Maximum: %d and Minimum: %d\n", min, max);  }    **int** main()  {  **int** arr[] = { 15, 14, 35, 2, 11, 83 };  **int** len = **sizeof**(arr) / **sizeof**(arr[0]);        // Function call      find\_small\_large(arr, len);    **return** 0;  } |

**Output**

Smallest: 2 and Largest: 83

**25. Write a Program for the cyclic rotation of an array to k positions.**

* C

|  |
| --- |
| // C program to rotate  // Array by  k elements  #include <stdio.h>    // Print array  **void** printArray(**int** arr[], **int** n)  {  **int** i;  **for** (i = 0; i < n; i++)  **printf**("%d ", arr[i]);  }    // Caculates greatest common divisor  **int** gcd(**int** a, **int** b)  {  **if** (b == 0)  **return** a;  **else**  **return** gcd(b, a % b);  }    // Rotate array  **void** Rotate(**int** arr[], **int** k, **int** N)  {  **int** i, j, a, temp;      k = k % N;    **int** rotate = gcd(k, N);    **for** (i = 0; i < rotate; i++) {            temp = arr[i];          j = i;  **while** (1) {              a = j + k;  **if** (a >= N)                  a = a - N;  **if** (a == i)  **break**;              arr[j] = arr[a];              j = a;          }          arr[j] = temp;      }  }    **int** main()  {  **int** arr[] = { 1, 2, 3, 4, 5 };        // Rotating array      Rotate(arr, 2, 5);        // Printing array      printArray(arr, 5);    **return** 0;  } |

**Output**

3 4 5 1 2

**26. Write a Program to sort First half in Ascending order and the Second in Descending order.**

* C

|  |
| --- |
| // C Program for Sorting  // First half in Ascending order  // and Second Descending order  #include <stdio.h>    **void** Sort\_asc\_desc(**int** arr[], **int** n)  {  **int** temp;  **for** (**int** i = 0; i < n - 1; i++) {  **for** (**int** j = i + 1; j < n; j++) {  **if** (arr[i] > arr[j]) {                  temp = arr[i];                  arr[i] = arr[j];                  arr[j] = temp;              }          }      }        // printing first half in ascending order  **for** (**int** i = 0; i < n / 2; i++)  **printf**("%d ", arr[i]);        // printing second half in descending order  **for** (**int** j = n - 1; j >= n / 2; j--)  **printf**("%d ", arr[j]);  }    **int** main()  {  **int** arr[] = { 11, 23, 42, 16, 83, 73, 59 };  **int** N = **sizeof**(arr) / **sizeof**(arr[0]);        Sort\_asc\_desc(arr, N);    **return** 0;  } |

**Output**

11 16 23 83 73 59 42

**27. Write a Program to print sums of all subsets in an array.**

* C

|  |
| --- |
| // C Program to print sum of  // all subsets  #include <stdio.h>    // Function to print sum of subset  // Using recursion  **void** subset\_sum(**int** arr[], **int** i, **int** j, **int** sum)  {  **if** (i > j) {  **printf**("%d ", sum);  **return**;      }        subset\_sum(arr, i + 1, j, sum + arr[i]);      subset\_sum(arr, i + 1, j, sum);  }    // driver code  **int** main()  {  **int** arr[] = { 1, 2, 3 };  **int** n = **sizeof**(arr) / **sizeof**(arr[0]);        // Function calling to print subset sum      subset\_sum(arr, 0, n - 1, 0);  **return** 0;  } |

**Output**

6 3 4 1 5 2 3 0

**28. Write a Program to Find if there is any subarray with a sum equal to 0.**

* C

|  |
| --- |
| // C Program to check 0 sum  // subarray possible  #include <stdio.h>    **int** main()  {      // array  **int** arr[] = { -2, 2, 1, 1, 8 };  **int** n = **sizeof**(arr) / **sizeof**(arr[0]);    **int** flag = 0, sum;        // Traversing array to check  **for** (**int** i = 0; i < n; i++) {    **for** (**int** j = i; j < n; j++) {              sum += arr[j];    **if** (sum == 0) {                  flag = 1;  **printf**(                      "True subarray with 0 sum is possible");  **break**;              }          }      }    **if** (flag == 0)  **printf**("No such condition");  } |

**Output**

True subarray with 0 sum is possible

**29. Write a C program to Implement Kadane’s Algorithm**

* C

|  |
| --- |
| // C program to implement Kadane's Algorithm  #include <limits.h>  #include <stdio.h>    **int** main()  {  **int** a[] = { -2, -3, 4, -1, -2, 1, 5, -3 };  **int** n = **sizeof**(a) / **sizeof**(a[0]);    **int** max\_so\_far = INT\_MIN, max\_ending\_here = 0,          start = 0, end = 0, s = 0;    **for** (**int** i = 0; i < n; i++) {          max\_ending\_here += a[i];    **if** (max\_so\_far < max\_ending\_here) {              max\_so\_far = max\_ending\_here;              start = s;              end = i;          }    **if** (max\_ending\_here < 0) {              max\_ending\_here = 0;              s = i + 1;          }      }  **printf**("Maximum contiguous sum is %d\n", max\_so\_far);  **printf**("Starting index %d Ending index %d", start, end);    **return** 0;  } |

**Output**

Maximum contiguous sum is 7

Starting index 2 Ending index 6

**30. Write a Program to find the transpose of a matrix.**

* C

|  |
| --- |
| #include <stdio.h>    // This function stores transpose of A[][] in B[][]  **void** transpose(**int** N, **int** M, **int** A[M][N], **int** B[N][M])  {  **int** i, j;  **for** (i = 0; i < N; i++)  **for** (j = 0; j < M; j++)              B[i][j] = A[j][i];  }    **int** main()  {  **int** M = 3;  **int** N = 4;    **int** A[3][4] = { { 1, 1, 1, 1 },                      { 2, 2, 2, 2 },                      { 3, 3, 3, 3 } };        // Note dimensions of B[][]  **int** B[N][M], i, j;        transpose(N, M, A, B);    **printf**("Result matrix is \n");  **for** (i = 0; i < N; i++) {  **for** (j = 0; j < M; j++)  **printf**("%d ", B[i][j]);  **printf**("\n");      }    **return** 0;  } |

**Output**

Result matrix is

1 2 3

1 2 3

1 2 3

1 2 3

**31. Write a Program to Rotate a matrix by 90 degrees in the clockwise direction in C.**

* C

|  |
| --- |
| // C Program to rotate the array  // By 90 degree in clockwise direction  #include <stdio.h>    **void** swap(**int**\* a, **int**\* b){  **int** temp = \*a;      \*a = \*b;      \*b = temp;  }    **int** main()  {    **int** n = 4;  **int** arr[4][4] = { { 1, 2, 3, 4 },                        { 5, 6, 7, 8 },                        { 9, 10, 11, 12 },                        { 13, 14, 15, 16 } };        // Print Orignal Matrix  **printf**("Orignal Matrix:\n");  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < n; j++) {  **printf**("%d ", arr[i][j]);          }  **printf**("\n");      }        // Rotate the matrix about the main diagonal  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < i; j++)              swap(&arr[i][j], &arr[j][i]);      }        // Rotate the matrix about middle column  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < n / 2; j++) {              swap(&arr[i][j], &arr[i][n - j - 1]);          }      }        // Print the rotated matrix  **printf**("Matrix after rotation: \n");  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < n; j++) {  **printf**("%d ", arr[i][j]);          }  **printf**("\n");      }  } |

**Output**

Orignal Matrix:

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

Matrix after rotation:

13 9 5 1

14 10 6 2

15 11 7 3

16 12 8 4

**32. Write a Program to find the Spiral Traversal of a Matrix in C.**

* C

|  |
| --- |
| // C Program to find Spiral Traversal  // Of a matrix  #include <stdio.h>    **int** main()  {  **int** arr[4][4] = { { 1, 5, 9, 13 },                        { 2, 6, 10, 14 },                        { 3, 7, 11, 15 },                        { 4, 8, 12, 16 } };    **int** m = 4, n = 4;  **int** i, l = 0, right = m - 1, begin = 0, end = n - 1;    **while** (l <= right && begin <= end) {            // Print the first row          // from the remaining rows  **for** (i = l; i <= right; ++i) {  **printf**("%d ", arr[begin][i]);          }          begin++;            // Print the last column          // from the remaining columns  **for** (i = begin; i <= end; ++i) {  **printf**("%d ", arr[i][right]);          }          right--;            // Print the last row from          // the remaining rows  **if** (begin <= end) {  **for** (i = right; i >= l; --i) {  **printf**("%d ", arr[end][i]);              }              end--;          }            // Print the first column from          // the remaining columns  **if** (l <= right) {  **for** (i = end; i >= begin; --i) {  **printf**("%d ", arr[i][l]);              }              l++;          }      }    **return** 0;  } |

**Output**

1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7

**33. Write a program to count the sum of numbers in a string.**

* C

|  |
| --- |
| #include <stdio.h>    **int** main()  {    **char** s[] = "124259";    **int** ans = 0;      // iterate through all the number  **for** (**int** i = 0; s[i] != '\0'; i++) {  **int** ele = s[i] - 48;  **if** (ele <= 9)              ans += ele;      }        // print sum of the numbers  **printf**("%d", ans);    **return** 0;  } |

**Output**

23

**34. Program to calculate the length of the string.**

* C

|  |
| --- |
| // C Program to calculate  // length of a string  #include <stdio.h>  #include <string.h>    **int** length(**char** s[], **int** i)  {  **if** (s[i] == '\0')  **return** 0;    **return** length(s, i + 1) + 1;  }    **int** main()  {  **char** s[] = "GeeksforGeeks";        // Calculating using strlen  **int** len = **strlen**(s);  **printf**("length using strlen:%d\n", len);        // Calculating using iteration  **int** i;  **for** (i = 0; s[i] != '\0'; i++) {  **continue**;      }  **printf**("length using iteration:%d\n", i);        // Calculating using recursion  **int** ans = length(s, 0);  **printf**("length using recursion:%d\n", ans);  **return** 0;  } |

**Output**

length using strlen:13

length using iteration:13

length using recursion:13

**35. Write a program to check string is a palindrome.**

* C

|  |
| --- |
| // C implementation to check if a given  // string is palindrome or not  #include <stdio.h>  #include <string.h>  #include <stdbool.h>    **bool** is\_palindrome(**char**\* str, **int** i, **int** j)  {  **if** (i >= j) {  **return** **true**;      }  **if** (str[i] != str[j]) {  **return** **false**;      }  **return** is\_palindrome(str, i + 1, j - 1);  }    **void** check\_palindrome(**char**\* str)  {        // Start from leftmost and      // rightmost corners of str  **int** h = 0;  **int** flag = 0;  **int** l = **strlen**(str) - 1;        // Keep comparing characters      // while they are same  **while** (h > l) {  **if** (str[l++] != str[h--]) {  **printf**("%s is not a palindrome\n", str);              flag = 1;  **break**;              // will break from here          }      }    **if** (flag == 0)  **printf**("%s is a palindrome\n", str);  }    **int** main()  {  **char** str[] = { "GeekeeG" };  **char** str2[] = { "GeeksforGeeks" };        check\_palindrome(str);    **printf**("Checking %s using recursive approach\n", str2);  **bool** ans = is\_palindrome(str2,0,**strlen**(str2)-1);  **if** (ans)  **printf**("It is Palindrome\n");  **else**  **printf**("Not a Palindrome\n");    **return** 0;  } |

**Output**

GeekeeG is a palindrome

Checking GeeksforGeeks using recursive approach

Not a Palindrome

**36. Write a program to print all permutations of a given string in lexicographically sorted order in C.**

* C

|  |
| --- |
| // C Program to print all permutations of a string in sorted  // order.  #include <stdio.h>  #include <stdlib.h>  #include <string.h>    // function two compare two characters a and b  **int** compare(**const** **void**\* a, **const** **void**\* b)  {  **return** (\*(**char**\*)a - \*(**char**\*)b);  }    // function two swap two characters a and b  **void** swap(**char**\* a, **char**\* b)  {  **char** t = \*a;      \*a = \*b;      \*b = t;  }    // function finds the index of the smallest character  **int** findCeil(**char** str[], **char** first, **int** l, **int** h)  {  **int** ceilIndex = l;    **for** (**int** i = l + 1; i <= h; i++)  **if** (str[i] > first && str[i] < str[ceilIndex])              ceilIndex = i;    **return** ceilIndex;  }    // Print all permutations of str in sorted order  **void** sortedPermutations(**char** str[])  {  **int** size = **strlen**(str);    **qsort**(str, size, **sizeof**(str[0]), compare);    **int** isFinished = 0;  **while** (!isFinished) {  **printf**("%s \n", str);    **int** i;  **for** (i = size - 2; i >= 0; --i)  **if** (str[i] < str[i + 1])  **break**;    **if** (i == -1)              isFinished = 1;  **else** {    **int** ceilIndex                  = findCeil(str, str[i], i + 1, size - 1);              swap(&str[i], &str[ceilIndex]);  **qsort**(str + i + 1, size - i - 1, **sizeof**(str[0]),                    compare);          }      }  }    **int** main()  {  **char** str[] = "123";      sortedPermutations(str);  **return** 0;  } |

**Output**

123

132

213

231

312

321

**37. Write a program to calculate the Power of a Number using Recursion in C.**

* C

|  |
| --- |
| // C program  to calculate the Power of a Number using  // Recursion  #include <stdio.h>    **int** power(**int** a, **int** b)  {  **if** (b == 0)  **return** 1;    **return** power(a, b - 1) \* a;  }    **int** main()  {  **int** a = 4, b = 5;    **int** ans = power(a, b);    **printf**("%d", ans);  **return** 0;  } |

**Output**

1024

**38. Write a Code to print the Fibonacci series using recursion.**

* C

|  |
| --- |
| // C Program to illustrate  // Fibonacci Series using Recursion  #include <stdio.h>    **int** fibonacci(**int** n)  {  **if** (n <= 1)  **return** n;  **return** fibonacci(n - 1) + fibonacci(n - 2);  }    **int** fibonacci\_iteration(**int** n)  {  **if** (n <= 1)  **return** 1;    **int** arr[n + 1];      arr[0] = 1;      arr[1] = 1;    **for** (**int** i = 2; i < n + 1; i++)          arr[i] = arr[i - 1] + arr[i - 2];    **return** arr[n];  }    **int** main()  {  **int** n = 9;  **printf**("Fibonacci using recursion of %d:%d\n", n,             fibonacci(n));        n = 11;  **printf**("Fibonacci using iteration of %d:%d", n,             fibonacci\_iteration(n));  **return** 0;  } |

**Output**

Fibonacci using recursion of 9:34

Fibonacci using iteration of 11:144

**39. Write a Program to find the HCF of two Numbers using Recursion.**

* C

|  |
| --- |
| // C program to find  // GCD of two numbers  #include <stdio.h>    // Recursive function to  // Calculate and return gcd of a and b  **int** gcd(**int** a, **int** b)  {      // Everything divides 0  **if** (a == 0)  **return** b;  **if** (b == 0)  **return** a;        // base case  **if** (a == b)  **return** a;        // a is greater  **if** (a > b)  **return** gcd(a - b, b);  **return** gcd(a, b - a);  }    **int** main()  {  **int** a = 192, b = 36;  **printf**("GCD of %d and %d is %d ", a, b, gcd(a, b));  **return** 0;  } |

**Output**

GCD of 192 and 36 is 12

**40. Write a Program in C to reverse a string using recursion.**

* C

|  |
| --- |
| // C program to reverse  // String using recursion  #include <stdio.h>  #include <string.h>    // Using Iteration for reverse  **void** reverse\_iteration(**char**\* str)  {  **int** i = 0;  **int** j = **strlen**(str) - 1;    **for** (; i < j; i++, j--) {  **char** temp = str[i];          str[i] = str[j];          str[j] = temp;      }  }    // Using recursion for reverse  **void** reverse(**char**\* str)  {  **if** (\*str) {          reverse(str + 1);  **printf**("%c", \*str);      }  }    **int** main()  {  **char** a[] = "Geeks for Geeks";  **printf**("Orignal string:%s\n", a);        reverse\_iteration(a);  **printf**("Reverse the string(iteration):%s\n", a);    **printf**("Using recursion for reverse:");      reverse(a);    **return** 0;  } |

**Output**

Orignal string:Geeks for Geeks

Reverse the string(iteration):skeeG rof skeeG

Using recursion for reverse:Geeks for Geeks

**41.  Write a C Program to search elements in an array.**

* C

|  |
| --- |
| // C code to Search elements in array  #include <stdio.h>    **int** search(**int** arr[], **int** N, **int** x)  {  **int** i;        // iterate through all the element of array  **for** (i = 0; i < N; i++)  **if** (arr[i] == x)  **return** i;  **return** -1;  }    **int** main(**void**)  {  **int** arr[] = { 9, 3, 2, 1, 10, 4 };  **int** x = 10;  **int** N = **sizeof**(arr) / **sizeof**(arr[0]);        // Function Call  **int** result = search(arr, N, x);    **if** (result == -1) {  **printf**("Element is not present in array");      }  **else** {  **printf**("Element is present at index %d", result);      }    **return** 0;  } |

**Output**

Element is present at index 4

**42.  Write a C Program to search elements in an array using Binary Search.**

* C

|  |
| --- |
| // C program to Search element  // in Array using Binary Search  #include <stdio.h>    **int** binarySearch(**int** arr[], **int** l, **int** r, **int** x)  {  **if** (r >= l) {  **int** mid = l + (r - l) / 2;            // If the element is present at the middle          // itself  **if** (arr[mid] == x)  **return** mid;            // If element is smaller than mid, then          // it can only be present in left subarray  **if** (arr[mid] > x)  **return** binarySearch(arr, l, mid - 1, x);            // Else the element can only be present          // in right subarray  **return** binarySearch(arr, mid + 1, r, x);      }    **return** -1;  }    **int** main()  {  **int** arr[] = { 11, 14, 19, 23, 40 };  **int** n = **sizeof**(arr) / **sizeof**(arr[0]);  **int** x = 40;  **int** result = binarySearch(arr, 0, n - 1, x);  **if** (result == -1) {  **printf**("Element is not present in array");      }  **else** {  **printf**("Element is present at index %d", result);      }  **return** 0;  } |

**Output**

Element is present at index 4

**43. Write a C Program to sort arrays using Bubble, Selection, and Insertion Sort.**

* C

|  |
| --- |
| // C Program to implement  // Sorting Algorithms  #include <stdio.h>    // A function to implement bubble sort  **void** bubble\_sort(**int**\* arr, **int** n)  {  **for** (**int** j = 0; j < n - 1; j++) {            // Last j elements are already in place  **for** (**int** i = 0; i < n - j - 1; i++) {  **if** (arr[i] > arr[i + 1]) {  **int** temp = arr[i];                  arr[i] = arr[i + 1];                  arr[i + 1] = temp;              }          }      }  }    // A function to implement swaping  **void** swap(**int**\* xp, **int**\* yp)  {  **int** temp = \*xp;      \*xp = \*yp;      \*yp = temp;  }    // A function to implement selectionSort  **void** selectionSort(**int** arr[], **int** n)  {        // One by one move boundary of unsorted subarray  **for** (**int** i = 0; i < n - 1; i++) {          // Find the minimum element in unsorted array  **int** min\_idx = i;  **for** (**int** j = i + 1; j < n; j++)  **if** (arr[j] < arr[min\_idx])                  min\_idx = j;            // Swap the found minimum element          // with the first element  **if** (min\_idx != i)              swap(&arr[min\_idx], &arr[i]);      }  }    **void** insertionSort(**int** arr[], **int** n)  {    **for** (**int** i = 1; i < n; i++) {  **int** key = arr[i];  **int** j = i - 1;            // Move elements of arr that are          // greater than key, to one position ahead          // of their current position  **while** (j >= 0 && arr[j] > key) {              arr[j + 1] = arr[j];              j = j - 1;          }          arr[j + 1] = key;      }  }    **int** main()  {  **int** arr1[] = { 9, 4, 3, 11, 1, 5 };  **int** arr2[] = { 4, 3, 9, 1, 5, 11 };  **int** arr3[] = { 5, 1, 11, 3, 4, 9 };  **int** n = 6;    **printf**("Non-Sorted array: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr1[i]);  **printf**("\n");        // sort array      bubble\_sort(arr1, n);        // printing array  **printf**("Sorted array using Bubble sort: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr1[i]);  **printf**("\n");    **printf**("Non-Sorted array: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr2[i]);  **printf**("\n");        // sort array      insertionSort(arr2, n);        // printing array  **printf**("Sorted array using Insertion Sort: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr2[i]);  **printf**("\n");    **printf**("Non-Sorted array: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr3[i]);  **printf**("\n");        // sort array      selectionSort(arr3, n);        // printing array  **printf**("Sorted array using Selection Sort: ");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr3[i]);  **printf**("\n");    **return** 0;  } |

**Output**

Non-Sorted array: 9 4 3 11 1 5

Sorted array using Bubble sort: 1 3 4 5 9 11

Non-Sorted array: 4 3 9 1 5 11

Sorted array using Insertion Sort: 1 3 4 5 9 11

Non-Sorted array: 5 1 11 3 4 9

Sorted array using Selection Sort: 1 3 4 5 9 11

**44. Write a C Program to sort arrays using Merge Sort.**

* C

|  |
| --- |
| // C program for  // Sorting array  // using Merge Sort  #include <stdio.h>    **void** merge(**int** arr[], **int** l, **int** m, **int** r)  {  **int** i, j, k;  **int** n1 = m - l + 1;  **int** n2 = r - m;        // create temperary arrays  **int** L[n1], R[n2];        // Copy data to arrays from L[] and R[]  **for** (i = 0; i < n1; i++)          L[i] = arr[l + i];  **for** (j = 0; j < n2; j++)          R[j] = arr[m + 1 + j];        // Initial index of first ,second      // and merged subarray respectively      i = 0;      j = 0;      k = l;  **while** (i < n1 && j < n2) {  **if** (L[i] <= R[j]) {              arr[k] = L[i];              i++;          }  **else** {              arr[k] = R[j];              j++;          }          k++;      }        // Copy the remaining elements of L[]  **while** (i < n1) {          arr[k] = L[i];          i++;          k++;      }        // Copy the remaining elements of R[]  **while** (j < n2) {          arr[k] = R[j];          j++;          k++;      }  }    **void** mergeSort(**int** arr[], **int** l, **int** r)  {  **if** (l < r) {            // calculating middle term  **int** mid = l + (r - l) / 2;            // divide to sort both halves          mergeSort(arr, l, mid);          mergeSort(arr, mid + 1, r);            merge(arr, l, mid, r);      }  }    **int** main()  {  **int** arr[] = { 23, 9, 13, 15, 6, 7 };  **int** n = **sizeof**(arr) / **sizeof**(arr[0]);        // Printing orignal array  **printf**("Given array:");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr[i]);  **printf**("\n");        mergeSort(arr, 0, n - 1);        // Printing sorted array  **printf**("Sorted array :");  **for** (**int** i = 0; i < n; i++)  **printf**("%d ", arr[i]);  **printf**("\n");    **return** 0;  } |

**Output**

Given array:23 9 13 15 6 7

Sorted array :6 7 9 13 15 23

**45. Write a C Program to sort arrays using Quick Sort.**

* C

|  |
| --- |
| // C Program for  // sorting array using  // Quick sort  #include <stdio.h>    **void** swap(**int**\* a, **int**\* b)  {  **int** t = \*a;      \*a = \*b;      \*b = t;  }    **int** partition(**int** array[], **int** low, **int** high)  {  **int** pivot = array[high];    **int** i = (low - 1);        // compare elements with the pivot  **for** (**int** j = low; j < high; j++) {  **if** (array[j] <= pivot) {              i++;              swap(&array[i], &array[j]);          }      }        // swap the pivot element with the greater element at i      swap(&array[i + 1], &array[high]);    **return** (i + 1);  }    **void** quickSort(**int** array[], **int** low, **int** high)  {  **if** (low < high) {  **int** pi = partition(array, low, high);          quickSort(array, low, pi - 1);          quickSort(array, pi + 1, high);      }  }    **void** printArray(**int** array[], **int** n)  {  **for** (**int** i = 0; i < n; ++i) {  **printf**("%d  ", array[i]);      }  **printf**("\n");  }    **int** main()  {  **int** arr[] = { 28, 7, 20, 1, 10, 3 , 6 };    **int** n = **sizeof**(arr) / **sizeof**(arr[0]);    **printf**("Unsorted Array:");      printArray(arr, n);        quickSort(arr, 0, n - 1);    **printf**("Sorted array :");      printArray(arr, n);    **return** 0;  } |

**Output**

Unsorted Array:28 7 20 1 10 3 6

Sorted array :1 3 6 7 10 20 28

**46. Write a program to sort an array using pointers.**

* C

|  |
| --- |
| // C Program to implement  // sorting using pointers  #include <stdio.h>    // Function to sort the numbers using pointers  **void** sort(**int** n, **int**\* ptr)  {  **int** i, j;        // Sort the numbers using pointers  **for** (i = 0; i < n; i++) {    **for** (j = i + 1; j < n; j++) {    **if** (\*(ptr + j) < \*(ptr + i)) {    **int** temp = \*(ptr + i);                  \*(ptr + i) = \*(ptr + j);                  \*(ptr + j) = temp;              }          }      }        // print the numbers  **for** (i = 0; i < n; i++)  **printf**("%d ", \*(ptr + i));  }    // Driver code  **int** main()  {  **int** n = 5;  **int** arr[] = { 13, 22, 7, 12, 4 };        sort(n, arr);    **return** 0;  } |

**Output**

4 7 12 13 22

**47. Write a C program to Store Information about Students Using Structure**

* C

|  |
| --- |
| // C Program to Store  // Information about Students  // Using Structure  #include <stdio.h>  #include <stdlib.h>  #include <string.h>    // Create the student structure  **struct** Student {  **char**\* name;  **int** roll\_number;  **int** age;  };    // Driver code  **int** main()  {  **int** n = 3;        // Create the student's structure variable      // with n Student's records  **struct** Student student[n];        // Get the students data      student[0].roll\_number = 1;      student[0].name = "Geeks1";      student[0].age = 10;        student[1].roll\_number = 2;      student[1].name = "Geeks2";      student[1].age = 11;        student[2].roll\_number = 3;      student[2].name = "Geeks3";      student[2].age = 13;        // Printing  the Structers  **printf**("Student Records:\n\n");  **for** (**int** i = 0; i < n; i++) {  **printf**("\tName : %s", student[i].name);  **printf**("\tRoll Number : %d",                 student[i].roll\_number);  **printf**("\tAge : %d\n", student[i].age);      }    **return** 0;  } |

**Output**

Student Records:

Name : Geeks1 Roll Number : 1 Age : 10

Name : Geeks2 Roll Number : 2 Age : 11

Name : Geeks3 Roll Number : 3 Age : 13

**48.  Write a C Program To Add Two Complex Numbers Using Structures And Functions.**

* C

|  |
| --- |
| // C program to demonstrate  // addition of complex numbers  #include <stdio.h>    // define a structure for complex number  **typedef** **struct** complexNumber {  **int** real;  **int** img;  } complex;      complex add(complex x, complex y)  {      // define a new complex number.      complex add;        // add similar type together      add.real = x.real + y.real;      add.img = x.img + y.img;    **return** (add);  }    **int** main()  {        // define three complex type numbers      complex x, y, sum;        // first complex number      x.real = 4;      x.img = 5;        // second complex number      y.real = 7;      y.img = 11;        // printing both complex numbers  **printf**(" x = %d + %di\n", x.real, x.img);  **printf**(" y = %d + %di\n", y.real, y.img);        // call add(a,b) function and      // pass complex numbers a & b      // as an parameter.      sum = add(x, y);        // print result  **printf**("\n sum = %d + %di", sum.real, sum.img);    **return** 0;  } |

**Output**

x = 4 + 5i

y = 7 + 11i

sum = 11 + 16i

**49. Write a C Program to add Two Distance Given as Input in Feet and Inches**

* C

|  |
| --- |
| // C program for calculating sum of  // Distance in intches and feet  #include "stdio.h"    // Struct defined for the inch-feet system  **struct** InchFeet {  **int** feet;  **float** inch;  };    // Function to find the sum of all N  // set of Inch Feet distances  **void** findSum(**struct** InchFeet arr[], **int** N)  {        // Variable to store sum  **int** feet\_sum = 0;  **float** inch\_sum = 0.0;    **int** x;        // Traverse the InchFeet array  **for** (**int** i = 0; i < N; i++) {            // Find the total sum of          // feet and inch          feet\_sum += arr[i].feet;          inch\_sum += arr[i].inch;      }        // If inch sum is greater than 11      // convert it into feet      // as 1 feet = 12 inch  **if** (inch\_sum >= 12) {            // Find integral part of inch\_sum          x = (**int**)inch\_sum;            // Delete the integral part x          inch\_sum -= x;            // Add x%12 to inch\_sum          inch\_sum += x % 12;            // Add x/12 to feet\_sum          feet\_sum += x / 12;      }        // Print the corresponding sum of      // feet\_sum and inch\_sum  **printf**("Feet Sum: %d\n", feet\_sum);  **printf**("Inch Sum: %.2f", inch\_sum);  }    **int** main()  {  **struct** InchFeet arr[]          = { { 11, 5.1 }, { 13, 4.5 }, { 6, 8.1 } };    **int** N = **sizeof**(arr) / **sizeof**(arr[0]);        findSum(arr, N);    **return** 0;  } |

**Output**

Feet Sum: 31

Inch Sum: 5.70

**50. Write a C program to reverse a linked list iteratively**

* C

|  |
| --- |
| // C program to reverse a linked list iteratively  #include <stdio.h>  #include <stdlib.h>    /\* Link list node \*/  **struct** Node {  **int** data;  **struct** Node\* next;  };    /\* Function to reverse the linked list \*/  **static** **void** reverse(**struct** Node\*\* head\_ref)  {  **struct** Node\* prev = NULL;  **struct** Node\* current = \*head\_ref;  **struct** Node\* next = NULL;  **while** (current != NULL) {          // Store next          next = current->next;            // Reverse current node's pointer          current->next = prev;            // Move pointers one position ahead.          prev = current;          current = next;      }      \*head\_ref = prev;  }    /\* Function to push a node \*/  **void** push(**struct** Node\*\* head\_ref, **int** new\_data)  {  **struct** Node\* new\_node          = (**struct** Node\*)**malloc**(**sizeof**(**struct** Node));      new\_node->data = new\_data;      new\_node->next = (\*head\_ref);      (\*head\_ref) = new\_node;  }    /\* Function to print linked list \*/  **void** printList(**struct** Node\* head)  {  **struct** Node\* temp = head;  **while** (temp != NULL) {  **printf**("%d ", temp->data);          temp = temp->next;      }  }    /\* Driver code\*/  **int** main()  {      /\* Start with the empty list \*/  **struct** Node\* head = NULL;        push(&head, 10);      push(&head, 14);      push(&head, 19);      push(&head, 25);    **printf**("Given linked list\n");      printList(head);      reverse(&head);  **printf**("\nReversed linked list \n");      printList(head);  **getchar**();  } |

**Output**

Given linked list

25 19 14 10

Reversed linked list

10 14 19 25

**C Coding Interview Questions – FAQs**

**Q: What are the most common C coding interview questions?**

*The most common C coding interview questions are designed to test your knowledge of the following topics:*

* *C syntax and semantics*
* *Data structures and algorithms*
* *Memory management*
* *Pointers*
* *File I/O*

***Some specific examples of common C coding interview questions include:***

* *Reverse a linked list.*
* *Implement a binary search tree.*
* *Write a function to find the maximum element in an array.*
* *Explain the difference between a pointer and an array.*
* *What is the difference between a function declaration and a function definition?*
* *How do you allocate memory on the heap?*
* *How do you free memory that has been allocated on the heap?*
* *What is a dangling pointer?*
* *How do you read and write data to a file?*

**Q. Who can benefit from these C coding interview questions and answers?**

*These questions are designed to benefit anyone preparing for a C coding interview. Whether you’re a beginner looking to learn the fundamentals or an experienced programmer aiming to enhance your C skills, this resource can assist you in your preparation.*

**Q: How can I use these questions effectively in my interview preparation?**

*Start by assessing your current level of expertise in C programming language. Then, you can use these questions to gradually build your skills up and knowledge. Practice solving them on your own, and review the explanations to ensure a thorough understanding.*