

C Programming Topic Wise Question & Solutions

Table of Contents

<i>Introduction to C Programming</i>	<i>1</i>
<i>Variables in C</i>	<i>5</i>
<i>Conditional Execution in C (if else/switch)</i>	<i>7</i>
If Else Case	7
Switch Case	12
<i>Loops in C</i>	<i>18</i>
For Loop	18
While Loop	24
<i>Arrays in C</i>	<i>28</i>
Integer Array	28
Char Array	36
<i>Pointers in C</i>	<i>48</i>
Integer Array using pointers	48
Char Array using pointers	52
<i>Functions in C</i>	<i>59</i>
Function using integer array	63
Functions using char Array	70

Introduction to C Programming

- Q1. A newbie in C write the below given program to print integer output help him with rewriting program.

```
#include<stdio.h>

void main ()
{
    int 1_one = 25;
    printf("%d",1_one);
}
```

Sample Solution:

C Code:

```
// C program to print int output
/*Correction is in variable name, it will be valid only if it's first char
it's either alphabet or underscore*/
#include<stdio.h>
void main()
{
    int v1_one = 25;
    printf("%d",v1_one);
}
```

- Q2. Write a C program to take a one-line text as input in string s from user and print Hello, World! followed by input string provided by user.

Example

s=" Life is beautiful "

The required output is:

Hello, World!

Life is beautiful

Input:

Welcome to C programming.

Output:

Hello, World!

Welcome to C programming.

Sample Solution:

C Code:

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

void main()
{
    char s[100];
    scanf("%[^\n]%*c", s);
    printf("Hello, World!\n");
    printf("%s", s);
}
```

Q3. Write a C program to swap two numbers without using the third variable.

Input:

Enter two numbers:

x=50

y=60

Output:

x=60

y=50

Sample Solution:

C Code:

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

void main()
{
    int x=50, y=60;
    printf("Values before swap x=%d and y=%d\n",x,y);
    x=x+y; // x=110 (50+60)
    y=x-y; // y=110-60 =50
    x=x-y; // x=110-50 =60
    // so now x=60 and y=50.
    printf("Values after swap x=%d and y=%d\n",x,y);
}
```

Q4. Write a C program to add to number without using “+” operator.

Input:

Enter two numbers:

20

40

Output:

60

Sample Solution:

C Code:

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

void main()
{
    int x,y;
    printf("Enter two numbers:\n");
    scanf("%d%d",&x,&y);
    int sum = -(-x-y); // -(-x-y)= x+y, simple mathematics trick
    printf("\nSum is: %d",sum);
}
```

Variables in C

Q5. Write a C program to convert a string to an unsigned long integer.

Input: 25

Output: 25

Sample Solution:

C Code:

```
#include<stdio.h>

#include<stdlib.h>

void main ()
{
    char buffer[123];
    unsigned long ul;
    printf ("\nInput an unsigned number: ");
    fgets (buffer,123,stdin);

    ul = strtoul (buffer,NULL,0);
    printf ("Output: %lu\n\n",ul);
}
```

Q6. Write a C program to convert float number to string.

In program it should print with %s

Input: 23.34

Output:

The string for the num is 23.34

Sample Solution:

C Code:

```
#include<stdio.h>
void main()
{
    char result[50];
    float num = 23.34;
    sprintf(result, "%f", num);
    printf("\n The string for the num is %s", result);
    getchar();
}
```


Conditional Execution in C (if else/switch)

➤ If Else Case

Q11. Write a C program to print “Hello World” without using semicolon even once?

Hint: Think out of the box

Sample Solution:

C Code:

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    if(printf("Hello World"))
}
```

Q12. Given an integer N and a pizza which can be cut into pieces, each cut should be a straight line going from the center of the pizza to its border. Also, the angle between any two cuts must be a positive integer. Two pieces are equal if their appropriate angles are equal.

The given pizza can be cut in following three ways:

- Cut the pizza into **N equal pieces**.
- Cut the pizza into **N pieces of any size**.
- Cut the pizza into **N pieces** such that **no two of them are equal**.

The task is to find if it is possible to cut the pizza in the above ways for a given value of **N**.

Print **1** if **possible** else **0** for all the cases i.e. print 111 if all the cases are possible.

Input: N = 4

Output: 1 1 1

Case 1: All four pieces can have angle = 90

Case 2: Same cut as Case 1

Case 3: 1, 2, 3 and 354 are the respective angles of the four pieces cut.

Input: N = 7

Output: 0 1 1

Sample Solution:

C Code:

```
// C implementation of the approach
#include<stdio.h>
#include<stdlib.h>

// Function to check if it is possible
// to cut the pizza in the given way
void cutPizza(int n)
{
    // Case 1
    if((360 % n == 0))
    {
        printf("%d", 1);
    }
    else
    {
        printf("%d",0);
    }

    // Case 2
    if(n <= 360)
    {
        printf("%d", 1);
    }
    else
    {
        printf("%d",0);
    }

    // Case 3
    if(((n * (n + 1)) / 2) <= 360)
    {
        printf("%d", 1);
    }
    else
    {
        printf("%d",0);
    }
}
```

Q13. Write a C program to count total number of notes in given amount

Input

Input amount: 575

Output

Total number of notes:

500: 1

100: 0

50: 1

20: 1

10: 0

5: 1

2: 0

1: 0

Sample Solution:

C Code:

```
/**
 * C program to count minimum number of notes in an amount
 */

#include <stdio.h>

void main()
{
    int amount;
    int note500, note100, note50, note20, note10, note5, note2, note1;

    /* Initialize all notes to 0 */
    note500 = note100 = note50 = note20 = note10 = note5 = note2 = note1 =
    0;

    /* Input amount from user */
    printf("Enter amount: ");
    scanf("%d", &amount);

    if(amount >= 500)
    {
        note500 = amount/500;
        amount -= note500 * 500;
    }
    if(amount >= 100)
    {
        note100 = amount/100;
        amount -= note100 * 100;
    }
    if(amount >= 50)
    {
        note50 = amount/50;
        amount -= note50 * 50;
    }
    if(amount >= 20)
    {
        note20 = amount/20;
```

```

        amount -= note20 * 20;
    }
    if(amount >= 10)
    {
        note10 = amount/10;
        amount -= note10 * 10;
    }
    if(amount >= 5)
    {
        note5 = amount/5;
        amount -= note5 * 5;
    }
    if(amount >= 2)
    {
        note2 = amount /2;
        amount -= note2 * 2;
    }
    if(amount >= 1)
    {
        note1 = amount;
    }

    /* Print required notes */
    printf("Total number of notes = \n");
    printf("500 = %d\n", note500);
    printf("100 = %d\n", note100);
    printf("50 = %d\n", note50);
    printf("20 = %d\n", note20);
    printf("10 = %d\n", note10);
    printf("5 = %d\n", note5);
    printf("2 = %d\n", note2);
    printf("1 = %d\n", note1);
}

```

➤ Switch Case

Q14. Write a C program to input an alphabet and check whether it is vowel or consonant using switch case.

Input

Input alphabet: c

Output

'c' is consonant

Sample Solution:

C Code:

```
/*C program to check vowel or consonant using switch case*/
#include <stdio.h>

void main()
{
    char ch;
    /* Input an alphabet from user */
    printf("Enter any alphabet: ");
    scanf("%c", &ch);
    /* Switch value of ch */
    switch(ch)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
            printf("Vowel");
            break;
        default:
            printf("Consonant");
    }
}
```

Q15. Write a C program to input hex number and convert it into binary using switch case.

Input

Input Hex Number: 1AC5

Output

Sample Solution:

C Code:

```
// C program to convert
// Hexadecimal number to Binary

#include <stdio.h>

void main()
{
    // Get the Hexadecimal number
    char hexdec[100] = "1AC5";

    // Convert Hexadecimal to Binary
    printf("\nEquivalent Binary value is : ");
    long int i = 0;

    while (hexdec[i]) {

        switch (hexdec[i]) {
            case '0':
                printf("0000");
                break;
            case '1':
                printf("0001");
                break;
            case '2':
                printf("0010");
                break;
            case '3':
                printf("0011");
                break;
            case '4':
                printf("0100");
                break;
            case '5':
                printf("0101");
                break;
            case '6':
                printf("0110");
                break;
            case '7':
```

```

        printf("0111");
        break;
    case '8':
        printf("1000");
        break;
    case '9':
        printf("1001");
        break;
    case 'A':
    case 'a':
        printf("1010");
        break;
    case 'B':
    case 'b':
        printf("1011");
        break;
    case 'C':
    case 'c':
        printf("1100");
        break;
    case 'D':
    case 'd':
        printf("1101");
        break;
    case 'E':
    case 'e':
        printf("1110");
        break;
    case 'F':
    case 'f':
        printf("1111");
        break;
    default:
        printf("\nInvalid hexadecimal digit %c",
            hexdec[i]);
    }
    i++;
}
return 0;
}

```


Loops in C

➤ For Loop

Q16. In the below code, change/add only one character and print '*' exactly 20 times.

```
void main (){  
  
    int i, n = 20;  
  
    for (i = 0; i < n; i--)  
  
        printf ("*");  
  
    getchar ();  
  
}
```

Sample Solution:

C Code:

```
#include <stdio.h>  
void main()  
{  
    int i, n = 20;  
    for (i = 0; i < n; n--)  
        printf("*");  
    getchar();  
}
```

Q17. Write a C program to get the sum of digit of given number in single statement line.

Input : $n = 687$

Output: 21

Input : $n = 12$

Output: 3

Sample Solution:

C Code:

```
# include<stdio.h>
void main()
{
    int n = 687;
    int sum;
    /*Single line that calculates sum*/
    for(sum=0; n > 0; sum+=n%10,n/=10);
    printf(" %d ", sum);
    getchar();
}
```

Q18. Write a C program to create below pattern

```
* * * * * * * * * *
* * * *   * * * *
* * *     * * *
* *       * *
*         *
*         *
* *       * *
* * *     * * *
* * * *   * * * *
* * * * * * * * * *
```

Hint: Think out of the box

Sample Solution:

C Code:

```
#include <stdio.h>
void main()
{
    int i, j, n;
    scanf("%d", &n);
    // upper half of the pattern
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < (2 * n); j++)
        {
            if(i + j <= n - 1) // upper left triangle
                printf("*");
            else
                printf(" ");
            if((i + n) <= j) // upper right triangle
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    // bottom half of the pattern
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < (2 * n); j++)
        {
            if(i >= j) //bottom left triangle
                printf("*");
            else
                printf(" ");
            if(i >= (2 * n - 1) - j) // bottom right triangle
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
}
```

Q19. Write a C program to take row and column and print output pattern as shown

Input

Input rows: 5

Input columns: 5

Output

10001

01010

00100

01010

10001

Hint: Think out of the box

Sample Solution:

C Code:

```
/**
 * C program to print box number pattern with cross center
 */

#include <stdio.h>

void main()
{
    int rows, cols, i, j;

    /* Input rows and columns from user */
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    for(i=1; i<=rows; i++)
    {
        for(j=1; j<=cols; j++)
        {
            if(i == j || (j == (cols+1) - i))
            {
                printf("1");
            }
            else
            {
                printf("0");
            }
        }

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

➤ While Loop

Q20. Write a C Program to Check Whether a Number is Palindrome or Not

Input:

12121

Output:

True

Input:

12121

Output:

False

Sample Solution:

C Code:

```
//C Program to Check Whether a Number is Palindrome or Not
#include <stdio.h>
void main() {
    int n, reversedN = 0, remainder, originalN;
    printf("Enter an integer: ");
    scanf("%d", &n);
    originalN = n;

    // reversed integer is stored in reversedN
    while (n != 0) {
        remainder = n % 10;
        reversedN = reversedN * 10 + remainder;
        n /= 10;
    }

    // palindrome if originalN and reversedN are equal
    if (originalN == reversedN)
        printf("True\n");
    else
        printf("False\n");
}
```

Q21. Write a C program to convert integer to roman.

Input

Enter a number: 1996

Output

mmxii

Sample Solution:

C Code:

```
// Program to convert a decimal number to roman numerals

#include <stdio.h>
void convertRoman(int num)
{
    while(num != 0)
    {
        if (num >= 1000)        // 1000 - m
        {
            printf("m");
            num -= 1000;
        }

        else if (num >= 900)    // 900 - cm
        {
            printf("cm");
            num -= 900;
        }

        else if (num >= 500)    // 500 - d
        {
            printf("d");
            num -= 500;
        }

        else if (num >= 400)    // 400 - cd
        {
            printf("cd");
            num -= 400;
        }

        else if (num >= 100)    // 100 - c
        {
            printf("c");
            num -= 100;
        }

        else if (num >= 90)     // 90 - xc
```

```

{
    printf("xc");
    num -= 90;
}
else if (num >= 50)    // 50 - 1
{
    printf("l");
    num-=50;
}
else if (num >= 40)    // 40 - xl
{
    printf("xl");
    num -= 40;
}
else if (num >= 10)    // 10 - x
{
    printf("x");
    num -= 10;
}
else if (num >= 9)     // 9 - ix
{
    printf("ix");
    (num -= 9);
}
else if (num >= 5)     // 5 - v
{
    printf("v");
    num -= 5;
}
else if (num >= 4)     // 4 - iv
{
    printf("iv");
    num-= 4;
}
else if (num >= 1)     // 1 - i
{
    printf("i");
    num-=1;
}
}
}

```

Arrays in C

➤ Integer Array

- Q22. Given an array `nums` with `n` objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

Example 1:

Input: `nums = [2,0,2,1,1,0]`

Output: `[0,0,1,1,2,2]`

Example 2:

Input: `nums = [2,0,1]`

Output: `[0,1,2]`

Example 3:

Input: `nums = [0]`

Output: `[0]`

Example 4:

Input: `nums = [1]`

Output: `[1]`

Sample Solution:

C Code:

// A C program to sort an array with 0, 1 and 2
// Doing in one pass needs two flags pointer, one left one right.

```
#include <stdio.h>
void main()
{
    int arr[] = { 0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1 };
    int numsSize = sizeof(arr) / sizeof(int);
    int redSt=0, bluSt=numsSize-1;
    int i=0;
    while(i<bluSt+1)
    {
        if(arr[i]==0)
        {
            int tmp = arr[i];
            arr[i] = arr[redSt];
            arr[redSt] = tmp;
            redSt++;
            i++;
            continue;
        }
        if(arr[i] ==2){
            int tmp = arr[i];
            arr[i] = arr[bluSt];
            arr[bluSt] = tmp;
            bluSt--;
            continue;
        }
        i++;
    }
    // Print the sorted array
    for (int i = 0; i < numsSize; i++)
        printf("%d",arr[i]);
}
```

Q23. Write a C program to find the maximum amount of water that C program to find maximum amount of water that can be trapped within given set of bars. block height are given as input in an array and assuming each block of size 1m.

Input 0:

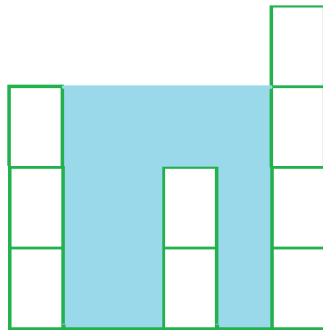
$N = 4$

$arr [] = \{7, 4, 0, 9\}$

Output 0:

10

Explanation:



Bars for input {3, 0, 0, 2, 0, 4}
Total trapped water = 3 + 3 + 1 + 3 = 10

Water trapped by above

block of height 4 is 3 units and above

block of height 0 is 7 units. So, the

total unit of water trapped is 10 units.

Input 1:

$N = 3$

$arr [] = \{6, 9, 9\}$

Output 1:

0

Explanation: No water trapped


```

int findWater(int arr[], int n)
{
    // left[i] contains height of tallest bar to the
    // left of i'th bar including itself
    int left[n];

    // Right [i] contains height of tallest bar to
    // the right of ith bar including itself
    int right[n];

    // Initialize result
    int water = 0;

    // Fill left array
    left[0] = arr[0];
    for (int i = 1; i < n; i++)
        left[i] = max(left[i - 1], arr[i]);

    // Fill right array
    right[n - 1] = arr[n - 1];
    for (int i = n - 2; i >= 0; i--)
        right[i] = max(right[i + 1], arr[i]);

    // Calculate the accumulated water element by element
    // consider the amount of water on i'th bar, the
    // amount of water accumulated on this particular
    // bar will be equal to min(left[i], right[i]) - arr[i] .
    for (int i = 0; i < n; i++)
        water += min(left[i], right[i]) - arr[i];

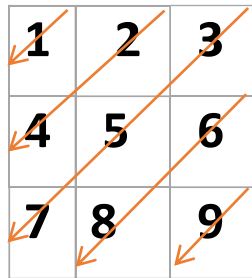
    return water;
}

// Driver program
void main()
{
    int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Maximum water that can be accumulated is %d"
        , findWater(arr, n));
}

```

Q24. Given a square matrix of size $N \times N$, return an array of its anti-diagonals. For better understanding let us look at the image given below

Input:



A 3x3 matrix with numbers 1 through 9. Orange arrows point from the top-right to the bottom-left, indicating the anti-diagonals. The arrows start at (1,3), (2,2), and (3,1) and point to (3,1), (2,2), and (1,3) respectively.

1	2	3
4	5	6
7	8	9

Output:

1
2 4
3 5 7
6 9 8
9

Sample Solution:

C Code:

```
// C implementation to return an array of its anti-diagonals
// when an N*N square matrix is given
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// function to print the diagonals
void diagonal(int A[3][3])
{
    int N = 3;
    // For each column start row is 0
    for (int col = 0; col < N; col++) {

        int startcol = col, startrow = 0;

        while (startcol >= 0 && startrow < N) {
            printf("%d\t", A[startrow][startcol]);

            startcol--;

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

    // For each row start column is N-1
    for (int row = 1; row < N; row++) {
        int startrow = row, startcol = N - 1;

        while (startrow < N && startcol >= 0) {
            printf("%d\t", A[startrow][startcol]);

            startcol--;

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

Q25. Write a C program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

Input: arr[] = {0, -1, 2, -3, 1}

sum = -2

Output: -3, 1

If we calculate the sum of the output,

$1 + (-3) = -2$

Input: arr[] = {1, -2, 1, 0, 5}

sum = 0

Output: -1

No valid pair exists.

Sample Solution:

C Code:

```
// C program to check if given array
// has 2 elements whose sum is equal
// to the given value

// Works only if range elements is limited
#include <stdio.h>
#include <stdbool.h>
#define MAX 100000

void main()
{
    int arr[] = { 1, 4, 45, 6, 10, 8 };
    int sum = 16;
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    //printPairs(A, arr_size, n);
    int i, temp;

    /*initialize hash set as 0*/
    bool s[MAX] = { 0 };

    for (i = 0; i < arr_size; i++)
    {
        temp = sum - arr[i];
        if (s[temp] == 1)
            printf(
                "Pair with given sum %d is (%d, %d) \n",
                sum, arr[i], temp);
        s[arr[i]] = 1;
    }

    getchar();
}
```

➤ Char Array

Q26. Write a C program to reverse a String in C

Input: ANU

Output: UNA

Sample Solution:

C Code:

```
// reverse of the string
#include<stdio.h>
#include<string.h>
void main()
{
    char str[100];
    printf("Enter string: ");
    fgets(str, sizeof(str), stdin);
    strrev(str);
    printf("The reverse of the string is: ");
    puts(str);
}
```

Q27. Write a C program to implement following string library operations on given string s1, s2 and character ch accordingly.

- a. strlen
- b. strcmp
- c. strcpy
- d. strcat
- e. strstr
- f. strchr

Sample Solution:

C Code:

```

/*****
C program for following string library operations
a.    strlen
b.    strcmp
c.    strcpy
d.    strcat
e.    strstr
f.    strchr
*****/
#include <stdio.h>
#include <string.h> // for sting operations

void main() {
    char str1[] = "abcd", str2[] = "abCd", ch='b';
    int result;
    char* ret;

    //a. strlen -- calculate lengths of given string
    //  strlen() function takes a string as an argument and re-
    //  turns its length.
    //  The returned value is of type size_t (the unsigned integer type).

    //  using the %zu format specifier to print size_t
    printf("Length of string a = %zu \n",strlen(str1));
    printf("Length of string b = %zu \n",strlen(str2));

    //b. strcmp --compares two strings
    //  strcmp() function takes two strings as an argument:str1 and str2
    //and returns 0 incase strings are equal and non-zero value
    //when strings are not equal

    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);

    //c. strcpy()-----copies a string to another
    //char* strcpy(char* destination, const char* source);
    //The strcpy() function copies the string pointed
    //by source (including the null character) to the destination.

```

```

//and returns the copied string.

// copying str1 to str2
strcpy(str2, str1);
printf("strcpy(str1, str2),copied str1 to str2 = %s\n", str1);

//d.strcat()----concatenates(joins) two strings
//char *strcat(char *destination, const char *source)
// The strcat() function concatenates the destination string and
// the source string, and the result is stored in the destination string.

// concatenates str1 and str2
// the resultant string is stored in str1.
strcat(str1, str2);
printf("strcat(str1, str2),result-
ant string is stored in str1 = %s\n", str1);

//e. strstr ----finds the first occurrence of the sub-
string s2 in the string s1
//The strstr() function takes two arguments: str and target.
//It searches for the first occurrence of tar-
get in the string pointed to by str.
//The terminating null characters are ignored.
//If the substring is found, the strstr() function re-
turns the pointer to the first
//character of the substring in dest.
//If the substring is not found, a null pointer is returned.
//If dest points to an empty string, str is returned
//char* strstr( char* str, const char* target );

// Find first occurrence of s2 in s1
printf("strstr operation:\n");
ret = strstr(str1, str2);

// Prints the result
if (ret) {
    printf("String found\n");
    printf("First occur-
rence of string '%s' in '%s' is '%s'", str2, str1, ret);
}
else
    printf("String not found\n");

```



```

//f. strchr( )----finds first occurrence of the charac-
ter in a given string.
// Syntax for strchr( ) function is given below.
//The terminating null byte is considered to be part of the string.
//int strchr(const char *str, int character);
    printf("strchr operation:\n");
    ret = strchr (str2,ch);
    if(ret)
        printf ("Character %c is found at posi-
tion %ld in string %s\n",ch,ret-str2+1,str2);
    else
        printf ("Character %c not found in string %s\n",ch,str2);
    return 0;
}

```

Q28. Write C program for the below pyramid string pattern.

Enter a string: PROGRAM

Enter number of rows: 5

P
RO
GRA
MPRO
GRAMP
ROGRAM

Sample Solution:

C Code:

```
//C program for the pyramid string pattern.
#include<stdio.h>
void main()
{
    // variables
    char str[20];
    int n, a=0;

    // take input values
    printf("Enter a string: ");
    scanf("%[^\n]", str);
    printf("Enter number of rows: ");
    scanf("%d", &n);

    // outer loop for row
    for(int i=0;i<=n;i++)
    {
        // for space
        for(int j=0;j<=n-i;j++)
            printf(" "); // print space

        for(int k=0;k<=i;k++)
        {
            // print character
            printf("%2c", str[a++]);

            // if index reach end of string then again
            // it should start from initial characters
            if(str[a]=='\0') a=0;
        }

        printf("\n"); // new line
    }
}
```

- Q29. Write a C program find the first occurrence of a character in a given string with the help of library function.

Input: *str[] = 'This is a string', ch = 'a'*

Output: *9*

Input: *str[] = 'My name is Anu', ch = 'a'*

Output : *5*

Sample Solution:

C Code:

```
// C program to find position of a character
// in a given string.
#include<stdio.h>
#include<stdlib.h>

void main ()
{
    char str[] = "My name is Anu";
    char* ch = strchr(str, 'a');
    printf("%d", ch - str + 1);
}
```

- Q30. Write a C program to check if strings are rotations of each other or not. Given a string s1 and a string s2, write a snippet to say whether s2 is a rotation of s1?

Input: *str1 =ABCD, str2= CDAB*

Output: *True*

Explanation: *ABCD and CDAB are rotation of each other*

Input: *str1 =ABCD, str2= ACBD*

Output: *False*

Sample Solution:

C Code:

```
// C program to check if two given strings are rotations of
// each other
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    char *str1 = "AACD";
    char *str2 = "ACDA";

    int size1 = strlen(str1);
    int size2 = strlen(str2);
    char temp[100];
    void *ptr;

    /* Check if sizes of two strings are same */
    if (size1 != size2)
    {
        printf("Strings lengths are not equal");
    }
    else
    {
        /* Create a temp string with value str1.str1 */
        temp[0] = '/0';
        strcat(temp, str1);
        strcat(temp, str2);
        /* Now check if str2 is a substring of temp */
        ptr = strstr(temp, str2);

        /* strstr returns NULL if the second string is NOT a
           substring of first string */
        if (ptr != NULL)
            printf("Strings are rotations of each other");
        else
            printf("Strings are not rotations of each other");
    }
    getchar();
}
```

Q31. Write a C program to check if strings are anagrams of each other or not. Given a string s1 and a string s2, write a snippet to say whether s2 is an anagrams of s1? An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, “abcd” and “dabc” are an anagram of each other.

Input: str1 =LISTEN, str2= SILENT

Output: True

Input: str1 =abcd, str2= pqrs

Output: False

Sample Solution:

C Code:

```
// C program to check if two strings
// are anagrams of each other
#include <stdio.h>

#define NO_OF_CHARS 256

void main()
{
    char str1[] = "listen";
    char str2[] = "silent";
    // Create 2 count arrays and initialize all values as 0
    int count1[NO_OF_CHARS] = { 0 };
    int count2[NO_OF_CHARS] = { 0 };
    int i;
    // For each character in input strings, increment count
    // in the corresponding count array
    for (i = 0; str1[i] && str2[i]; i++) {
        count1[str1[i]]++;
        count2[str2[i]]++;
    }
    // If both strings are of different length. Removing
    // this condition will make the program fail for strings
    // like "aaca" and "aca"
    if (str1[i] || str2[i])
    { printf("The two strings are of unequal length\n");
    }
    else
    {
        // Compare count arrays
        int count=0;
        for (i = 0; i < NO_OF_CHARS; i++)
        {
            if (count1[i] != count2[i])
            {
                count++;
            }
        }
    }
}
```

Q32. Write a C Program to Sort set of strings in alphabetical order string taken as input from user.

Input: 4

Nanu

Manu

Tanu

Anu

Output:

Anu

Manu

Nanu

Tanu

Sample Solution:

C Code:

```
/* C program would sort the input strings in
 * an ascending order and would display the same
 */
#include<stdio.h>
#include<string.h>

void main()
{
    int i,j,count;
    char str[25][25],temp[25];
    puts("How many strings u are going to enter?: ");
    scanf("%d",&count);

    puts("Enter Strings one by one: ");
    for(i=0;i<=count;i++)
        gets(str[i]);
    for(i=0;i<=count;i++)
        for(j=i+1;j<=count;j++){
            if(strcmp(str[i],str[j])>0){
                strcpy(temp,str[i]);
                strcpy(str[i],str[j]);
                strcpy(str[j],temp);
            }
        }
    printf("Order of Sorted Strings:");
    for(i=0;i<=count;i++)
        puts(str[i]);
}
```

Pointers in C

➤ Integer Array using pointers

Q33. Write a program in C to store n elements in an array and print the elements using pointer

Input the number of elements to store in the array :5

Input 5 number of elements in the array :

element - 0 : 5

element - 1 : 7

element - 2 : 2

element - 3 : 9

element - 4 : 8

Expected Output :

The elements you entered are :

element - 0 : 5

element - 1 : 7

element - 2 : 2

element - 3 : 9

element - 4 : 8

Sample Solution:

C Code:

```
#include <stdio.h>
void main()
{
    int arr1[25], i,n;
    printf("\n\n Pointer :Store and retrieve elements from an array :\n");
    printf("-----\n");
    printf(" Input the number of elements to store in the array :");
    scanf("%d",&n);

    printf(" Input %d number of elements in the array :\n",n);
    for(i=0;i<n;i++)
    {
        printf(" element - %d : ",i);
        scanf("%d",arr1+i);
    }
    printf(" The elements you entered are : \n");
    for(i=0;i<n;i++)
    {
        printf(" element - %d : %d \n",i,*(arr1+i));
    }
}
```

Q34. Given a square matrix, turn it by 90 degrees in anti-clockwise direction without using any extra space.

Input:

1 2 3

4 5 6

7 8 9

Output:

3 6 9

2 5 8

1 4 7

***Rotated the input matrix by
90 degrees in anti-clockwise direction.***

Sample Solution:

C Code:

```
// C program for left
// rotation of matrix by 90 degree without using extra space

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

#define R 4
#define C 4
// This function swaps 2 values
void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// After transpose we swap elements of column
// one by one for finding left
// rotation of matrix by 90 degree
void reverseColumns(int arr[R][C])
{
    for (int i = 0; i < C; i++)
        for (int j = 0, k = C - 1;
             j < k; j++, k--)
            swap(&arr[j][i], &arr[k][i]);
}

// Function for do transpose of matrix
void transpose(int arr[R][C])
{
    for (int i = 0; i < R; i++)
        for (int j = i; j < C; j++)
            swap(&arr[i][j], &arr[j][i]);
}
// Function to anticlockwise rotate matrix by 90 degree
void rotate90(int arr[R][C])
{
    transpose(arr);
    reverseColumns(arr);
}
```

```

// Function for print matrix
void printMatrix(int arr[R][C])
{
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++)
            printf("%d\t", arr[i][j]);
        printf("\n");
    }
}

// Driven code
int main(void)
{
    int arr[R][C] = { { 1, 2, 3, 4 },
                      { 5, 6, 7, 8 },
                      { 9, 10, 11, 12 },
                      { 13, 14, 15, 16 } };
    printMatrix(arr);
    printf("After Rotation...\n");
    rotate90(arr);
    printMatrix(arr);
    return 0;
}

```

➤ Char Array using pointers

Q35. Write a C program to find the occurrence of the word and replace a word in a text by another given word.

Given three strings 'str', 'oldW' and 'newW'. The task is find all occurrences of the word 'oldW' and replace them with word 'newW'.

Input: str [] = " All is well",

oldW [] = "is",

newW [] = "izzz"

Output: All izzz well

Sample Solution:

C Code:

```
// C program to search and replace
// all occurrences of a word with other word.
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to replace a string with another string
char* replaceWord(const char* s, const char* oldW,
                  const char* newW)
{
    char* result;
    int i, cnt = 0;
    int newWlen = strlen(newW);
    int oldWlen = strlen(oldW);
    // Counting the number of times old word
    // occur in the string
    for (i = 0; s[i] != '\0'; i++) {
        if (strstr(&s[i], oldW) == &s[i]) {
            cnt++;
            // Jumping to index after the old word.
            i += oldWlen - 1;
        }
    }
    // Making new string of enough length
    result = (char*)malloc(i + cnt * (newWlen - oldWlen) + 1);
    i = 0;
    while (*s) {
        // compare the substring with the result
        if (strstr(s, oldW) == s) {
            strcpy(&result[i], newW);
            i += newWlen;
            s += oldWlen;
        }
        else
            result[i++] = *s++;
    }
    result[i] = '\0';
    return result;
}
```

```

// Driver Program
void main()
{
    char str[] = "Today is holiday";
    char c[] = "is";
    char d[] = "izzz";

    char* result = NULL;

    // oldW string
    printf("Old string: %s\n", str);

    result = replaceWord(str, c, d);
    printf("New String: %s\n", result);

    free(result);
}

```

Q36. Write a C program to compare two string using pointers (take two string s1 and s2 print equal if both the strings are equal else print not equal)

Input: s1 = "for", s2 = "formula"

Output: not equal

Explanation:

Strings are not equal

Input: s1 = "learn", s2 = "learn"

Output: equal

Explanation:

Strings are equal

Sample Solution:

C Code:

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

void main()
{
    char *a = "this is good";
    char *b = "this is good";
    int cmp;

    while(*a == *b && *a != '\0' && *b != '\0')
    {
        cmp = 0;
        a++;
        b++;
    }
    if(*a == '\0' && *b == '\0' && cmp == 0)
    {
        printf("equal\n");
    }
    else
    {
        printf("not equal\n");
    }
}
```

Q37. Given a string *s* we need to tell minimum characters to be appended (insertion at end) to make a string palindrome.

Input: *s* = "abede"

Output: 2

We can make string palindrome as "abede**ba**"
by adding **ba** at the end of the string.

Input: *s* = "aabb"

Output: 2

We can make string palindrome as "aabb**aa**"
by adding **aa** at the end of the string.

Sample Solution:

C Code:

```
// C program to find minimum number of appends
// needed to make a string Palindrome
#include<stdio.h>
#include<string.h>
// Checking if the string is palindrome or not
int isPalindrome(char *str)
{
    int len = strlen(str);

    // single character is always palindrome
    if (len == 1)
        return 1; //true

    // pointing to first character
    char *ptr1 = str;

    // pointing to last character
    char *ptr2 = str+len-1;

    while (ptr2 > ptr1)
    {
        if (*ptr1 != *ptr2)
            return 0; //false
        ptr1++;
        ptr2--;
    }
    return 1; //true
}
// Recursive function to count number of appends
int noOfAppends(char s[])
{
    if (isPalindrome(s))
        return 0;
    // Removing first character of string by
    // incrementing base address pointer.
    s++;

    return 1 + noOfAppends(s);
}
```

```
// Driver program to test above functions
void main()
{
    char s[] = "abede";
    printf("%d\n", noOfAppends(s));
}
```

Functions in C

Q38. Write a one-line C function to round floating point numbers

Input: 1.67

Output: 2

Sample Solution:

C Code:

```
/* Program for rounding floating point numbers */
# include<stdio.h>

int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}

void main()
{
    printf("%d", roundNo(-1.777));
    getchar();
}
```

Q39. Write a C function to find the parity of an unsigned integer. Parity of a number refers to whether it contains an odd or even number of 1-bits. The number has “odd parity”, if it contains odd number of 1-bits and is “even parity” if it contains even number of 1-bits.

Example:

Input: $n = 7$

Output:

$parity = 1$

Sample Solution:

C Code:

```
// C program to find parity
// of an integer
# include <stdio.h>

/* Function to get parity of number n. It returns 1
if n has odd parity, and returns 0 if n has even
parity */
int getParity(unsigned int n)
{
    bool parity = 0;
    while (n)
    {
        parity = !parity;
        n      = n & (n - 1);
    }
    return parity;
}

/* Driver program to test getParity() */
void main()
{
    unsigned int n = 7;
    printf("Parity of no %d = %s", n,
        (getParity(n)? "odd": "even"));

    getchar();
}
```

Q40. Write a C program to print numbers from 1 to 100 without using a loop

Hint: Think out of the box

Sample Solution:

C Code:

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

void printNumbers(int);
void main ()
{
    int n=1;
    printNumbers(n);
}

void printNumbers(int n)
{
    if(n<=100)
    {
        printf("%d\n", n);
        printNumbers(n+1);
    }
}
```

- Q41. Write a C Program to illustrate the following global variables and local variables

Concept: Basic understanding

Sample Solution:

C Code:

```
//C program to illustrate global and local variable
#include<stdio.h>

int globalVar1 = 22, globalVar2 = 44;
void test();

void main()
{
    int mainVar1 = 22, mainVar2 = 44;
    // mainVar1, mainVar2 are local variables of main function
    /*mainVar1 and mainVar2 variables are having scope
    within this main function only.
    These are not visible to test function.*/
    /* If you try to access testVar1 and testVar2 in this function,
    you will get 'testVar1' undeclared and 'testVar2' undeclared er-
    ror */
    printf("All global variables are accessed from main function");
    printf("\nvalues: globalVar1=%d:globalVar2=%d",globalVar1,globalVar2);
    printf("\n main local variables values : mainVar1 = %d and main-
    Var2 = %d", mainVar1, mainVar2);
    test();
}
void test()
{
    int testVar1 = 50, testVar2 = 80;
    // testVar1, testVar2 are local variables of test function
    /*testVar1 and testVar2 variables are having scope
    within this test function only.
    These are not visible to main function.*/
    /* If you try to access mainVar1 and mainVar2 in this function,
    you will get 'mainVar1' undeclared and 'mainVar2' undeclared
    error */
    printf("\nAll global variables are accessed from test function");
    printf("\nvalues: globalVar1=%d:globalVar2=%d\n",globalVar1,global-
    Var2);
    printf("\ntest local variables values : testVar1 = %d and test-
    Var2 = %d", testVar1, testVar2);
}
```


➤ Function using integer array

Q42. Write a C function to find count of square in given a m x n rectangle.

Input: $m = 2, n = 2$

Output: 5

There are 4 squares of size 1x1 + 1 square of size 2x2.

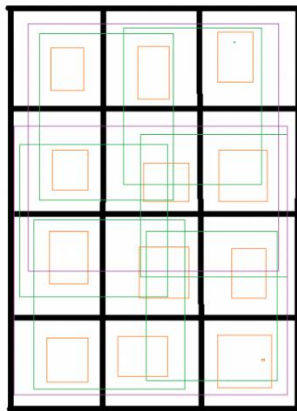
Input: $m = 4, n = 3$

Output: 20

There are 12 squares of size 1x1 +

6 squares of size 2x2 +

2 squares of size 3x3



Total 20 Squares in Matrix of size 4 x 3

12 Squares of Size 1 x 1

6 Squares of size 2 x 2

2 Squares of size 3 x 3

Sample Solution:

C Code:

```
// C program to count squares
// in a rectangle of size m x n

#include <stdio.h>

// Returns count of all squares
// in a rectangle of size m x n
int countSquares(int m, int n)
{
    int temp;
    // If n is smaller, swap m and n
    if (n < m)
    {
        temp=n;
        n=m;
        m=temp;
    }
    // Now n is greater dimension,
    // apply formula
    return m * (m + 1) * (2 * m + 1) /
        6 + (n - m) * m * (m + 1) / 2;
}

// Driver Code
void main()
{
    int m = 4, n = 3;
    printf("Count of squares is %d",countSquares(m, n));
}
```

- Q43. Write a C program to implement iterative Binary Search.
A iterative binary search function. It returns location of x in given array arr[l..r] if present, otherwise -1.

Enter number of elements:

5

Enter 5 integers:

1

9

22

24

46

Input:

Enter the value to find:

24

Output:4

24 is present at index 4.

Input:

Enter the value to find:

4

Output: -1

4 is not present.

Sample Solution:

C Code:

```
// C program to implement iterative Binary Search
#include <stdio.h>
// A iterative binary search function. It returns
// location of x in given array arr[l..r] if present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }
    // if we reach here, then element was
    // not present
    return -1;
}

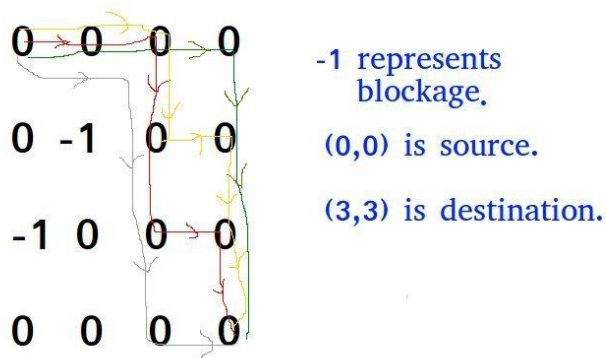
void main()
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? printf("Element is not present"
                           " in array")
                  : printf("Element is present at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

- Q44. Write a C function to find the count of the number of ways to reach from source to destination in a given a maze with obstacles, count number of paths to reach rightmost-bottommost cell from topmost-leftmost cell. A cell in given maze has value -1 if it is a blockage or dead end, else 0.
From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only.

Input: $\text{maze}[R][C] = \{\{0, 0, 0, 0\},$
 $\{0, -1, 0, 0\},$
 $\{-1, 0, 0, 0\},$
 $\{0, 0, 0, 0\}\};$

Output: 4

There are four possible paths as shown in below diagram.



There are total four paths from source to destination

Sample Solution:

C Code:

```
// C program to count number of paths in a maze with obstacles.
#include <stdio.h>
#include <stdlib.h>

#define R 4
#define C 4

// Returns count of possible paths in a maze[R][C]
// from (0,0) to (R-1,C-1)
int countPaths(int maze[][C])
{
    // If the initial cell is blocked, there is no
    // way of moving anywhere
    if (maze[0][0]==-1)
        return 0;

    // Initializing the leftmost column
    for (int i=0; i<R; i++)
    {
        if (maze[i][0] == 0)
            maze[i][0] = 1;
        // If we encounter a blocked cell in leftmost
        // row, there is no way of visiting any cell
        // directly below it.
        else
            break;
    }

    // Similarly initialize the topmost row
    for (int i=1; i<C; i++)
    {
        if (maze[0][i] == 0)
            maze[0][i] = 1;
        // If we encounter a blocked cell in bottommost
        // row, there is no way of visiting any cell
        // directly below it.
        else
            break;
    }
}
```

```

// The only difference is that if a cell is -1,
// simply ignore it else recursively compute
// count value maze[i][j]
for (int i=1; i<R; i++)
{
    for (int j=1; j<C; j++)
    {
        // If blockage is found, ignore this cell
        if (maze[i][j] == -1)
            continue;

        // If we can reach maze[i][j] from maze[i-1][j]
        // then increment count.
        if (maze[i-1][j] > 0)
            maze[i][j] = (maze[i][j] + maze[i-1][j]);

        // If we can reach maze[i][j] from maze[i][j-1]
        // then increment count.
        if (maze[i][j-1] > 0)
            maze[i][j] = (maze[i][j] + maze[i][j-1]);
    }
}
// If the final cell is blocked, output 0, otherwise
// the answer
return (maze[R-1][C-1] > 0)? maze[R-1][C-1] : 0;
}

// Driver code
void main()
{
    int maze[R][C] = {{0, 0, 0, 0},
                      {0, -1, 0, 0},
                      {-1, 0, 0, 0},
                      {0, 0, 0, 0}};
    printf("%d\n", countPaths(maze));
}

```

➤ Functions using char Array

- Q45. Write a C program to implement strstr library function (take two string s1 and s2, if first string s1 is substring of second string s2 then return index number else it will return -1 if string is not matching)

Input: s1 = "for", s2 = "formula"

Output: 5

Explanation:

String "for" is present as a substring of s2.

Input: s1 = "practice", s2 = "learn"

Output: -1.

Explanation:

There is no occurrence of "practice" in "learn"

Sample Solution:

C Code:

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

// Returns true if s1 is substring of s2
int isSubstring(char *s1, char *s2)
{
    int M = strlen(s1);
    int N = strlen(s2);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++)
    {
        int j;
        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (s2[i + j] != s1[j])
                break;

        if (j == M)
            return i;
    }

    return -1;
}

void main()
{
    char *s1 = "possible\0";
    char *s2 = "impossible\0";
    int res = isSubstring(s1, s2);
    if (res == -1)
        printf( "Not present\n");
    else
        printf("Present at index %d", res);
}
```

Q46. Given two strings where first string may contain wild card characters and second string is a normal string. Write a function that returns true if the two strings match. The following are allowed wild card characters in first string.

** --> Matches with 0 or more instances of any character or set of characters.*

? --> Matches with any one character.

Example 1:

Input: s = "aa", p = "a"

Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = ""*

Output: true

Explanation: '' matches any sequence.*

Example 3:

Input: s = "cb", p = "? a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

Example 4:

*Input: s = "adceb", p = "**a*b"*

Output: true

Explanation: The first '' matches the empty sequence, while the second '*' matches the substring "dce".*

Example 5:

*Input: s = "acdc b", p = "a*c? b"*

Output: false

Sample Solution:

C Code:

```
// A C program to match wild card characters
#include <stdio.h>
#include <stdbool.h>

// The main function that checks if two given strings
// match. The first string may contain wildcard characters
bool match(char *first, char * second)
{
    // If we reach at the end of both strings, we are done
    if (*first == '\0' && *second == '\0')
        return true;

    // Make sure that the characters after '*' are present
    // in second string. This function assumes that the first
    // string will not contain two consecutive '*'
    if (*first == '*' && *(first+1) != '\0' && *second == '\0')
        return false;

    // If the first string contains '?', or current characters
    // of both strings match
    if (*first == '?' || *first == *second)
        return match(first+1, second+1);

    // If there is *, then there are two possibilities
    // a) We consider current character of second string
    // b) We ignore current character of second string.
    if (*first == '*')
        return match(first+1, second) || match(first, second+1);
    return false;
}

// A function to run test cases
void test(char *first, char *second)
{ match(first, second)? puts("Yes"): puts("No"); }
```

```

// Driver program to test above functions
void main()
{
    test("*pqrs", "pqrst"); // No because 't' is not in first
    test("abc*bcd", "abcdhghgbcd"); // Yes
    test("abc*c?d", "abcd"); // No because second must have 2
                                // instances of 'c'
    test("*c*d", "abcd"); // Yes
    test("*?c*d", "abcd"); // Yes
}

```

Q47. Write code to convert a given 4-digit number into words.

Input: 1234

Output: one thousand two hundred thirty four

Sample Solution:

C Code:

```
/* C program to print a given number in words. The program
handles numbers from 0 to 9999 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* A function that prints given number in words */
void convert_to_words(char* num)
{
    int len = strlen(num); // Get number of digits in given number

    /* Base cases */
    if (len == 0) {
        printf("empty string\n");
        return;
    }
    if (len > 4) {
        printf("Length more than 4 is not supported\n");
        return;
    }

    /* The first string is not used, it is to make
    array indexing simple */
    char* single_digits[]
        = { "zero", "one", "two", "three", "four",
            "five", "six", "seven", "eight", "nine" };
    /* The first string is not used, it is to make
    array indexing simple */
    char* two_digits[]
        = { "", "ten", "eleven", "twelve",
            "thirteen", "fourteen", "fifteen", "sixteen",
            "seventeen", "eighteen", "nineteen" };
    /* The first two string are not used, they are to make
    array indexing simple*/
    char* tens_multiple[] = { "", "", "twenty",
                               "thirty", "forty", "fifty",
                               "sixty", "seventy", "eighty",
                               "ninety" };
```

```

char* tens_power[] = { "hundred", "thousand" };

/* For single digit number */
if (len == 1) {
    printf("%s\n", single_digits[*num - '0']);
    return;
}

/* Iterate while num is not '\0' */
while (*num != '\0') {

    /* Code path for first 2 digits */
    if (len >= 3) {
        if (*num - '0' != 0) {
            printf("%s ", single_digits[*num - '0']);
            printf("%s ",
                tens_power[len - 3]); // here len can
                                     // be 3 or 4
        }
        --len;
    }

    /* Code path for last 2 digits */
    else {
        /* Need to explicitly handle 10-19. Sum of the
        two digits is used as index of "two_digits"
        array of strings */
        if (*num == '1') {
            int sum = *num - '0' + *(num + 1) - '0';
            printf("%s\n", two_digits[sum]);
            return;
        }

        /* Need to explicitly handle 20 */
        else if (*num == '2' && *(num + 1) == '0') {
            printf("twenty\n");
            return;
        }
    }
}

```

```

        /* Rest of the two digit numbers i.e., 21 to 99
        */
        else {
            int i = *num - '0';
            printf("%s ", i ? tens_multiple[i] : "");
            ++num;
            if (*num != '0')
                printf("%s ",
                    single_digits[*num - '0']);
        }
    }
    ++num;
}

/* Driver program to test above function */
void main()
{
    convert_to_words("9923");
    convert_to_words("523");
    convert_to_words("89");
    convert_to_words("8");
}

```

- Q48. Write a C program given a string containing lowercase characters. The task is to print the maximum occurring character in the input string. If 2 or more characters appear the same number of times, print the lexicographically (alphabetically) lowest (first) character

Input: test sample

Output: e

't', 'e' and 's' appears 2 times, but 'e' is the lexicographically smallest character.

Input: sample program

Output: a

Sample Solution:

C Code:

```
// C implementation to find the maximum occurring character in
// an input string which is lexicographically first
#include<stdio.h>
#include<stdlib.h>
// function to find the maximum occurring character in
// an input string which is lexicographically first
char getMaxOccurringChar(char str[])
{
    // freq[] used as hash table
    int freq[26] = { 0 };

    // to store maximum frequency
    int max = -1;

    // to store the maximum occurring character
    char result;

    // length of 'str'
    int len = strlen(str);

    // get frequency of each character of 'str'
    for (int i = 0; i < len; i++)
        freq[str[i] - 'a']++;

    // for each character, where character is obtained by
    // (i + 'a') check whether it is the maximum character
    // so far and accordingly update 'result'
    for (int i = 0; i < 26; i++)
        if (max < freq[i]) {
            max = freq[i];
            result = (char)(i + 'a');
        }

    // maximum occurring character
    return result;
}
```

```
// Driver Code
void main()
{
    char str[] = "sample program";
    printf("Maximum occurring character = %c ",getMaxOccurringChar(str));
}
```

Q49. Given a string *s* input it from user, write a C function to find the length of the longest substring, it will return count and take input as given string

Example 1:

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"

Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Example 4:

Input: s = ""

Output: 0 without repeating characters

Sample Solution:

C Code:

```
// C program to find the length of the longest substring
// without repeating characters

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

#define NO_OF_CHARS 256

int max(int a, int b){return (a>b?a:b);}

int longestUniqueSubstr(char * str)
{
    int n = strlen(str);
    int res = 0; // result
    int lastIndex[NO_OF_CHARS];
    // last index of all characters is initialized
    // as -1
    for (int k=0; k <255; k++)
    {
        lastIndex[k] = -1;
    }
    // Initialize start of current window
    int i = 0;
    // Move end of current window
    for (int j = 0; j < n; j++)
    {
        // Find the last index of str[j]
        // Update i (starting index of current window)
        // as maximum of current value of i and last
        // index plus 1
        i = max(i, lastIndex[str[j]] + 1);
        // Update result if we get a larger window
        res = max(res, j - i + 1);
        // Update last index of j.
        lastIndex[str[j]] = j;
    }
    return res;
}
```

```
// Driver code
void main()
{
    char *str1 = "abcabcbb\0";
    char *str2 = "bbbb\0";
    char *str3 = "pwwkew\0";

    printf(" The input string is %s\n ",str1);
    printf(" The length of the longest non-repeating character sub-
string is %d\n",longestUniqueSubstr(str1));
    printf(" The input string is %s\n ",str2);
    printf(" The length of the longest non-repeating character sub-
string is %d\n",longestUniqueSubstr(str2));
    printf(" The input string is %s\n ",str3);
    printf(" The length of the longest non-repeating character sub-
string is %d\n",longestUniqueSubstr(str3));
}
```

- Q50. Write a program to Validate an IPv4 Address. IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1

Input: 128.0.0.1

Output: valid

Explanation:

This is a valid IP address.

Input: "125.512.100.abc"

Output: Invalid

Explanation:

invalid IP address with this string.

Sample Solution:

C Code:

```
// C program to check valid ip address
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int validate_number(char *str) {
    while (*str) {
        if(!isdigit(*str)){
            /*if the character is not a number, return
            false*/
            return 0;
        }
        str++; //point to next character
    }
    return 1;
}

int validate_ip(char *ip) { //check whether the IP is valid or not
    int i, num, dots = 0;
    char *ptr;
    if (ip == NULL)
        return 0;
    ptr = strtok(ip, "."); //cut the string using dot delimiter
    if (ptr == NULL)
        return 0;
    while (ptr) {
        if (!validate_number(ptr)) /*check whether the sub string is
        holding only number or not*/
            return 0;
        num = atoi(ptr); //convert substring to number
        if (num >= 0 && num <= 255) {
            ptr = strtok(NULL, "."); //cut the next part of the string
            if (ptr != NULL)
                dots++; //increase the dot count
        } else
            return 0;
    }
    if (dots != 3) //if the number of dots are not 3, return false
```

```
/* Driver program to test above function */
```

```
void main()
{
    char ip1[] = "192.168.4.1";
    char ip2[] = "172.16.253.1";
    char ip3[] = "192.800.100.1";
    char ip4[] = "125.512.100.abc";
    validate_ip(ip1)? printf("Valid\n"): printf("Not valid\n");
    validate_ip(ip2)? printf("Valid\n"): printf("Not valid\n");
    validate_ip(ip3)? printf("Valid\n"): printf("Not valid\n");
    validate_ip(ip4)? printf("Valid\n"): printf("Not valid\n");
}
```

```
*****END*****
```