```
/**
 * C program to print the sum of all odd numbers from 1 to n
 */

#include <stdio.h>

int main()
{
    int i, n, sum=0;

    /* Input range to find sum of odd numbers */
    printf("Enter upper limit: ");
    scanf("%d", &n);

    /* Find the sum of all odd number */
    for(i=1; i<=n; i+=2)
    {
        sum += i;
    }

    printf("Sum of odd numbers = %d", sum);

    return

/**
 * C program to check whether a number is palindrome or not
 */

#include <stdio.h>

int main()
{
    int n, num, rev = 0;

    /* Input a number from user */
    printf("Enter any number to check palindrome: ");
    scanf("%d", &n);

    /* Copy original value to 'num' to 'n'*/
    num = n;

    /* Find reverse of n and store in rev */
    while(n != 0)
    {
        rev = (rev * 10) + (n % 10);
        n  /= 10;
    }
    /* Check if reverse is equal to 'num' or not */
    if(rev == num)
    {
        printf("%d is palindrome.", num);
    }
    else
    {
        printf("%d is not palindrome.", num);
    }

    return 0;
}



/**
 * C program to swap first and last digit of a number
 */

#include <stdio.h>
#include <math.h>

int main()
{
    int num, swappedNum;
    int firstDigit, lastDigit, digits;

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

    /* Find last digit */
    lastDigit  = num % 10;

    /* Find total number of digit - 1 */
    digits     = (int)log10(num);

    /* Find first digit */
    firstDigit = (int)(num / pow(10, digits));

    swappedNum  = lastDigit;
    swappedNum *= (int) pow(10, digits);
    swappedNum += num % ((int) pow(10, digits));
    swappedNum -= lastDigit;
    swappedNum += firstDigit;

    printf("Original number = %d", num);
    printf("Number after swapping first and last digit: %d", swappedNum);

    return 0;
}


/**
 * C program to print Fibonacci series up to n terms
 */

#include <stdio.h>

int main()
{
    int a, b, c, i, terms;

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

    /* Fibonacci magic initialization */
    a = 0;
    b = 1;
    c = 0;

    printf("Fibonacci terms: \n");

    /* Iterate through n terms */
    for(i=1; i<=terms; i++)
    {
        printf("%d, ", c);

        a = b;     // Copy n-1 to n-2
        b = c;     // Copy current to n-1
        c = a + b; // New term
    }

    return 0;
}



/**
 * C program to check whether a number is Perfect number or not
 */

#include <stdio.h>
```

```
int main()
{
   int i, num, sum = 0;

   /* Input a number from user */
   printf("Enter any number to check perfect number: ");
   scanf("%d", &num);

   /* Calculate sum of all proper divisors */
   for(i = 1; i <= num / 2; i++)
   {
      /* If i is a divisor of num */
      if(num%i == 0)
      {
         sum += i;
      }
   }

   /* Check whether the sum of proper divisors is equal to num */
   if(sum == num && num > 0)
   {
      printf("%d is PERFECT NUMBER", num);
   }
   else
   {
      printf("%d is NOT PERFECT NUMBER", num);
   }

   return 0;
}
```

Skip to content
Search for:
C program to print reverse Pyramid star pattern
July 2, 2015PankajC programmingC, Loop, Program, Star Patterns

Write a C program to print reverse Pyramid or reverse equilateral triangle star pattern series using for loop. How to print inverted pyramid or inverted equilateral triangle star pattern series in C program. Logic to print reverse pyramid star pattern series in C programming.

Example

Input

Input rows: 5
Output

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

Required knowledge
Basic C programming, For loop, Nested loop

Logic to print reverse pyramid star pattern

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

The above pattern has N (in this case 5) rows. Each row has exactly N * 2 - (i * 2 - 1) stars. In addition the pattern consists of leading spaces. For each row it contain i - 1 leading space (where i is current row number).

Step by step descriptive logic to print reverse pyramid star pattern.

Input number of rows to print from user. Store it in a variable say rows.
To iterate through rows, run an outer loop from 1 to rows. The loop structure should look like for(i=1; i<=rows; i++).
To print spaces, run an inner loop from 1 to i - 1. The loop structure should look like for(j=1; j<i; j++). Inside this loop print single space.
To print stars, run another inner loop from 1 to rows * 2 - (i * 2 - 1). The loop structure should look like for(j=1; j<= (rows*2 - (i*2-1)); j++). Inside this loop print star.
After printing all stars for each row, move to next line i.e. print new line.
Program to print reverse pyramid star pattern

```c
/**
 * C program to print reverse pyramid star pattern
 */

#include <stdio.h>

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

    /* Input rows to print from user */
    printf("Enter number of rows : ");
    scanf("%d", &rows);
    for(i=1; i<=rows; i++)
    {
        /* Print leading spaces */
        for(j=1; j<i; j++)
        {
            printf(" ");
        }

        /* Print stars */
        for(j=1; j<=(rows*2 -(2*i-1)); j++)
        {
            printf("*");
        }

        /* Move to next line */
        printf("\n");
    }

    return 0;
}

/**
 * C program to find cube of any number using function
 */
#include <stdio.h>

/* Function declaration */
double cube(double num);

int main()
{
    int num;
    double c;

    /* Input number to find cube from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    c = cube(num);

    printf("Cube of %d is %.2f", num, c);

    return 0;
}

/**
 * Function to find cube of any number
 */
```

```
double cube(double num)
{
    return (num * num * num);
}




/**
 * C program to check prime, armstrong and perfect numbers using functions
 */

#include <stdio.h>
#include <math.h>


/* Function declarations */
int isPrime(int num);
int isArmstrong(int num);
int isPerfect(int num);


int main()
{
    int num;

    printf("Enter any number: ");
    scanf("%d", &num);

    // Call isPrime() functions
    if(isPrime(num))
    {
        printf("%d is Prime number.\n", num);
    }
    else
    {
        printf("%d is not Prime number.\n", num);
    }

    // Call isArmstrong() function
    if(isArmstrong(num))
    {
        printf("%d is Armstrong number.\n", num);
    }
    else
    {
        printf("%d is not Armstrong number.\n", num);
    }
    // Call isPerfect() function
    if(isPerfect(num))
    {
        printf("%d is Perfect number.\n", num);
    }
    else
    {
        printf("%d is not Perfect number.\n", num);
    }

    return 0;
}



/**
 * Check whether a number is prime or not.
 * Returns 1 if the number is prime otherwise 0.
 */
int isPrime(int num)
{
    int i;

    for(i=2; i<=num/2; i++)
    {
        /*
         * If the number is divisible by any number
         * other than 1 and self then it is not prime
         */
        if(num%i == 0)
        {
            return 0;
        }
    }

    return 1;
}



/**
 * Check whether a number is Armstrong number or not.
 * Returns 1 if the number is Armstrong number otherwise 0.
 */
int isArmstrong(int num)
{
    int lastDigit, sum, originalNum, digits;
    sum = 0;

    originalNum = num;

    /* Find total digits in num */
    digits = (int) log10(num) + 1;

    /*
     * Calculate sum of power of digits
     */
    while(num > 0)
    {
        // Extract the last digit
        lastDigit = num % 10;

        // Compute sum of power of last digit
        sum = sum + round(pow(lastDigit, digits));

        // Remove the last digit
        num = num / 10;
    }

    return (originalNum == sum);
}



/**
 * Check whether the number is perfect number or not.
 * Returns 1 if the number is perfect otherwise 0.
 */
int isPerfect(int num)
{
    int i, sum, n;
    sum = 0;
    n = num;

    for(i=1; i<n; i++)
    {
        /* If i is a divisor of num */
        if(n%i == 0)
        {
            sum += i;
        }
    }

    return (num == sum);
}
```

```
/**
 * C program to find power of a number using recursion
 */

#include <stdio.h>


/* Power function declaration */
double pow(double base, int expo);


int main()
{
    double base, power;
    int expo;

    /* Input base and exponent from user */
    printf("Enter base: ");
    scanf("%lf", &base);
    printf("Enter exponent: ");
    scanf("%d", &expo);

    // Call pow function
    power = pow(base, expo);

    printf("%.2lf ^ %d = %f", base, expo, power);

    return 0;
}


/**
 * Calculate power of any number.
 * Returns base ^ expo
 */
double pow(double base, int expo)
{
    /* Base condition */
    if(expo == 0)
        return 1;
    else if(expo > 0)
        return base * pow(base, expo - 1);
    else
        return 1 / pow(base, -expo);
}

/**
```

```
 * C program to print all natural numbers from 1 to n using recursion
 */

#include <stdio.h>


/* Function declaration */
void printNaturalNumbers(int lowerLimit, int upperLimit);



int main()
{
    int lowerLimit, upperLimit;

    /* Input lower and upper limit from user */
    printf("Enter lower limit: ");
    scanf("%d", &lowerLimit);
    printf("Enter upper limit: ");
    scanf("%d", &upperLimit);

    printf("All natural numbers from %d to %d are: ", lowerLimit, upperLimit);
    printNaturalNumbers(lowerLimit, upperLimit);

    return 0;
}


/**
 * Recursively prints all natural number between the given range.
 */
void printNaturalNumbers(int lowerLimit, int upperLimit)
{
    if(lowerLimit > upperLimit)
        return;

    printf("%d, ", lowerLimit);

    // Recursively call the function to print next number
    printNaturalNumbers(lowerLimit + 1, upperLimit);
}


/**
 * C program to find sum of array elements using recursion
 */

#include <stdio.h>
```

```
#define MAX_SIZE 100

/* Function declaration to find sum of array */
int sum(int arr[], int start, int len);


int main()
{
   int arr[MAX_SIZE];
   int N, i, sumofarray;


   /* Input size and elements in array  */
   printf("Enter size of the array: ");
   scanf("%d", &N);
   printf("Enter elements in the array: ");
   for(i=0; i<N; i++)
   {
      scanf("%d", &arr[i]);
   }


   sumofarray = sum(arr, 0, N);
   printf("Sum of array elements: %d", sumofarray);

   return 0;
}


/**
 * Recursively find the sum of elements in an array.
 */
int sum(int arr[], int start, int len)
{
   /* Recursion base condition */
   if(start >= len)
      return 0;

   return (arr[start] + sum(arr, start + 1, len));
}




/**
 * C program to merge two sorted array in ascending order
 */

#include <stdio.h>
```

```
#define MAX_SIZE 100      // Maximum size of the array

int main()
{
   int arr1[MAX_SIZE], arr2[MAX_SIZE], mergeArray[MAX_SIZE * 2];
   int size1, size2, mergeSize;
   int index1, index2, mergeIndex;
   int i;

   /* Input size of first array */
   printf("Enter the size of first array : ");
   scanf("%d", &size1);

   /* Input elements in first array */
   printf("Enter elements in first array : ");
   for(i=0; i<size1; i++)
   {
      scanf("%d", &arr1[i]);
   }

   /* Input size of second array */
   printf("\nEnter the size of second array : ");
   scanf("%d", &size2);

   /* Input elements in second array */
   printf("Enter elements in second array : ");
   for(i=0; i<size2; i++)
   {
      scanf("%d", &arr2[i]);
   }


   mergeSize = size1 + size2;


   /*
    * Merge two array in ascending order
    */
   index1 = 0;
   index2 = 0;
   for(mergeIndex=0; mergeIndex < mergeSize; mergeIndex++)
   {
      /*
       * If all elements of one array
       * is merged to final array
       */
      if(index1 >= size1 || index2 >= size2)
      {
          break;
      }


      if(arr1[index1] < arr2[index2])
      {
          mergeArray[mergeIndex] = arr1[index1];
          index1++;
      }
      else
      {
          mergeArray[mergeIndex] = arr2[index2];
          index2++;
      }
   }

   /*
    * Merge remaining array elements
    */
   while(index1 < size1)
   {
      mergeArray[mergeIndex] = arr1[index1];
      mergeIndex++;
      index1++;
   }
   while(index2 < size2)
   {
      mergeArray[mergeIndex] = arr2[index2];
      mergeIndex++;
      index2++;
   }


   /*
    * Print merged array
    */
   printf("\nArray merged in ascending order : ");
   for(i=0; i<mergeSize; i++)
   {
      printf("%d\t", mergeArray[i]);
   }

   return 0;
}
```

C program to find reverse of array

July 17, 2015PankajC programmingArray, C, Program

Write a C program to input elements in array and find reverse of array. How to find reverse of array in C programming. Logic to find reverse of array in C program.

Example

Input

Input array elements: 10, 5, 16, 35, 500
Output

Array elements after reverse: 500, 35, 16, 5, 10

Required knowledge
Basic Input Output, For loop, While loop, Array

There are various ways to reverse an array. Here I will explain the basic three algorithms to reverse a given array. First the simplest and easiest one, so that ever beginner can get what I am up to.

Logic to print array in reverse order
This algorithm in real does not produces a reversed array. Instead it just prints array in reverse order. If you are looking to reverse the elements then skip to next logic. So here goes step by step descriptive logic to print array in reverse order.

Input size and elements in array from user. Store it in some variable say size and arr.
Run a loop from size - 1 to 0 in decremented style. The loop structure should look like for(i=size-1; i>=0; i--).
Inside loop print current array element i.e. arr[i].
Program to print array in reverse

```
/**
 * C program to print array in reverse order
 */

#include <stdio.h>
#define MAX_SIZE 100      // Defines maximum size of array

int main()
{
    int arr[MAX_SIZE];
    int size, i;

    /* Input size of array */
    printf("Enter size of the array: ");
    scanf("%d", &size);
    /* Input array elements */
    printf("Enter elements in array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }

    /*
     * Print array in reversed order
     */
    printf("\nArray in reverse order: ");
    for(i = size-1; i>=0; i--)
    {
        printf("%d\t", arr[i]);
    }

    return 0;
}


/**
 * C program to search element in array
 */

#include <stdio.h>

#define MAX_SIZE 100  // Maximum array size

int main()
{
    int arr[MAX_SIZE];
    int size, i, toSearch, found;

    /* Input size of array */
    printf("Enter size of array: ");
    scanf("%d", &size);

    /* Input elements of array */
    printf("Enter elements in array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }

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

    /* Assume that element does not exists in array */
    found = 0;

    for(i=0; i<size; i++)
    {
        /*
         * If element is found in array then raise found flag
         * and terminate from loop.
         */
        if(arr[i] == toSearch)
        {
            found = 1;
            break;
        }
    }

    /*
     * If element is not found in array
     */
    if(found == 1)
    {
        printf("\n%d is found at position %d", toSearch, i + 1);
    }
    else
    {
        printf("\n%d is not found in the array", toSearch);
    }

    return 0;
}



/**
 * C program to find sum of two matrices of size 3x3
 */

#include <stdio.h>

#define SIZE 3 // Size of the matrix

int main()
{
    int A[SIZE][SIZE]; // Matrix 1
    int B[SIZE][SIZE]; // Matrix 2
    int C[SIZE][SIZE]; // Resultant matrix

    int row, col;
    /* Input elements in first matrix*/
    printf("Enter elements in matrix A of size 3x3: \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &A[row][col]);
        }
    }

    /* Input elements in second matrix */
    printf("\nEnter elements in matrix B of size 3x3: \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &B[row][col]);
        }
    }

    /*
     * Add both matrices A and B entry wise or element wise
     * and stores result in matrix C
     */
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            /* Cij = Aij + Bij */
            C[row][col] = A[row][col] + B[row][col];
        }
    }


    /* Print the value of resultant matrix C */
    printf("\nSum of matrices A+B = \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            printf("%d ", C[row][col]);
        }
        printf("\n");
    }

    return 0;
}
```

```
/**
 * C program to find upper triangular matrix
 */

#include <stdio.h>
#define MAX_ROWS 3
#define MAX_COLS 3

int main()
{
   int array[MAX_ROWS][MAX_COLS];
   int row, col, isUpper;

   /* Input elements in matrix from user */
   printf("Enter elements in matrix of size %dx%d: \n", MAX_ROWS, MAX_COLS);
   for(row=0; row<MAX_ROWS; row++)
   {
      for(col=0; col<MAX_COLS; col++)
      {
         scanf("%d", &array[row][col]);
      }
   }

   /* Check Upper triangular matrix condition */
   isUpper = 1;
   for(row=0; row<MAX_ROWS; row++)
   {
      for(col=0; col<MAX_COLS; col++)
      {
         /*
          * If elements below the main diagonal (col<row)
          * is not equal to zero then it is not upper triangular matrix
          */
         if(col<row && array[row][col]!=0)
         {
            isUpper = 0;
         }
      }
   }

   /* Print elements of upper triangular matrix  */
   if(isUpper == 1)
   {
      printf("\nThe matrix is Upper triangular matrix.\n");

      for(row=0; row<MAX_ROWS; row++)
      {
         for(col=0; col<MAX_COLS; col++)
```

```
            {
                printf("%d ", array[row][col]);
            }

            printf("\n");
        }
    }
    else
    {
        printf("\nThe matrix is not Upper triangular matrix.");
    }

    return 0;
}/**
 * C program to find transpose of a matrix
 */

#include <stdio.h>
#define MAX_ROWS 3
#define MAX_COLS 3

int main()
{
    int A[MAX_ROWS][MAX_COLS];  // Original matrix
    int B[MAX_COLS][MAX_ROWS];  // Transpose matrix

    int row, col;

    /* Input elements in matrix A from user */
    printf("Enter elements in matrix of size %dx%d: \n", MAX_ROWS, MAX_COLS);
    for(row=0; row<MAX_ROWS; row++)
    {
        for(col=0; col<MAX_COLS; col++)
        {
            scanf("%d", &A[row][col]);
        }
    }

    /*
     * Find transpose of matrix A
     */
    for(row=0; row<MAX_ROWS; row++)
    {
        for(col=0; col<MAX_COLS; col++)
        {
            /* Store each row of matrix A to each column of matrix B */
            B[col][row] = A[row][col];
        }
    }

    /* Print the original matrix A */
    printf("\nOriginal matrix: \n");
    for(row=0; row<MAX_ROWS; row++)
    {
        for(col=0; col<MAX_COLS; col++)
        {
            printf("%d ", A[row][col]);
        }

        printf("\n");
    }

    /* Print the transpose of matrix A */
    printf("Transpose of matrix A: \n");
    for(row=0; row<MAX_COLS; row++)
    {
        for(col=0; col<MAX_ROWS; col++)
        {
            printf("%d ", B[row][col]);
        }

        printf("\n");
    }

    return 0;
}

/**
 * C program to check whether a matrix is symmetric matrix or not
 */

#include <stdio.h>
#define SIZE 3

int main()
{
    int A[SIZE][SIZE];  // Original matrix
    int B[SIZE][SIZE];  // Transpose matrix

    int row, col, isSymmetric;

    /* Input elements in matrix A from user */
    printf("Enter elements in matrix of size 3x3: \n");
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            scanf("%d", &A[row][col]);
        }
    }

    /*
     * Find transpose of matrix A
     */
    for(row=0; row<SIZE; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            /* Store each row of matrix A to each column of matrix B */
            B[row][col] = A[col][row];
        }
    }


    /*
     * Check whether matrix A is equal to its transpose or not
     */
    isSymmetric = 1;
    for(row=0; row<SIZE && isSymmetric; row++)
    {
        for(col=0; col<SIZE; col++)
        {
            /* If matrix A is not equal to its transpose */
            if(A[row][col] != B[row][col])
            {
                isSymmetric = 0;
                break;
            }
        }
    }

    /*
     * If the given matrix is symmetric.
     */
    if(isSymmetric == 1)
    {
        printf("\nThe given matrix is Symmetric matrix: \n");

        for(row=0; row<SIZE; row++)
        {
            for(col=0; col<SIZE; col++)
            {
                printf("%d ", A[row][col]);
            }
```

```
            printf("\n");
        }
    }
    else
    {
        printf("\nThe given matrix is not Symmetric matrix.");
    }

    return 0;
}

/**
 * C program to find length of a string using for loop
 */

#include <stdio.h>
#define MAX_SIZE 100 // Maximum size of the string

int main()
{
    char text[MAX_SIZE]; /* Declares a string of size 100 */
    int i;
    int count= 0;

    /* Input a string from user */
    printf("Enter any string: ");
    gets(text);

    /* Iterate till the last character of string */
    for(i=0; text[i]!='\0'; i++)
    {
        count++;
    }

    printf("Length of '%s' = %d", text, count);

    return 0;
}
```