Add

1. Program to add two numbers

```c
#include<stdio.h>

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

   printf("Enter two numbers to add\n");
   scanf("%d%d",&a,&b);

   c = a + b;

   printf("Sum of entered numbers = %d\n",c);

   return 0;
}
```

2. Program to add two numbers repeatedly

```c
#include <stdio.h>

int main()
{
   int a, b, c;
   char ch;

   while (1) {
      printf("Inut two integers\n");
      scanf("%d%d", &a, &b);
      getchar();

      c = a + b;

      printf("(%d) + (%d) = (%d)\n", a, b, c);

      printf("Do you wish to add more numbers (y/n)\n");
      scanf("%c", &ch);

      if (ch == 'y' || ch == 'Y')
         continue;
      else
          break;
   }

   return 0;
}
```

3. Addition without using third variable

```c
#include<stdio.h>

main()
{
   int a = 1, b = 2;

   /* Storing result of addition in variable a */

   a = a + b;

   /** Not recommended because original value of a is lost
    *  and you may be using it somewhere in code considering it
    *  as it was entered by the user.
    */

   printf("Sum of a and b = %d\n", a);

   return 0;
}
```

4. Adding numbers in c using function

```c
#include<stdio.h>

long addition(long, long);

main()
{
   long first, second, sum;

   scanf("%ld%ld", &first, &second);

   sum = addition(first, second);

   printf("%ld\n", sum);

   return 0;
}

long addition(long a, long b)
{
   long result;

   result = a + b;

   return result;
}
```

Even or Odd

1. C program to check odd or even using modulus operator

```
#include <stdio.h>

int main()
{
   int n;

   printf("Enter an integer\n");
   scanf("%d", &n);

   if (n%2 == 0)
      printf("Even\n");
   else
      printf("Odd\n");

   return 0;
}
```

We can use bitwise AND (&) operator to check odd or even, as an example consider binary of 7 (0111) when we perform 7 & 1 the result will be one and you may observe that the least significant bit of every odd number is 1, so (odd number & 1) will be one always and also (even number & 1) is zero.

2. C program to check odd or even using bitwise operator

```
#include <stdio.h>

int main()
{
   int n;

   printf("Enter an integer\n");
   scanf("%d", &n);

   if (n & 1 == 1)
      printf("Odd\n");
   else
      printf("Even\n");

   return 0;
}
```

3. Find odd or even using conditional operator

```
#include <stdio.h>

int main()
{
   int n;

   printf("Input an integer\n");
   scanf("%d", &n);
```

```
    n%2 == 0 ? printf("Even\n") : printf("Odd\n");

    return 0;
}
```

4. C program to check odd or even without using bitwise or modulus operator

```
#include <stdio.h>

int main()
{
    int n;

    printf("Enter an integer\n");
    scanf("%d", &n);

    if ((n/2)*2 == n)
        printf("Even\n");
    else
        printf("Odd\n");

    return 0;
}
```

Add, subtract, multiply and divide

```
#include <stdio.h>

int main()
{
    int first, second, add, subtract, multiply;
    float divide;

    printf("Enter two integers\n");
    scanf("%d%d", &first, &second);

    add = first + second;
    subtract = first - second;
    multiply = first * second;
    divide = first / (float)second;    //typecasting

    printf("Sum = %d\n",add);
    printf("Difference = %d\n",subtract);
    printf("Multiplication = %d\n",multiply);
    printf("Division = %.2f\n",divide);

    return 0;
}
```

Vowel

1. C programming code

```c
#include <stdio.h>

int main()
{
  char ch;

  printf("Enter a character\n");
  scanf("%c", &ch);

  if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E' || ch == 'i' || ch == 'I' || ch =='o' || ch=='O' || ch == 'u' || ch == 'U')
    printf("%c is a vowel.\n", ch);
  else
    printf("%c is not a vowel.\n", ch);

  return 0;
}
```

2. Check vowel using switch statement

```c
#include <stdio.h>

int main()
{
  char ch;

  printf("Input a character\n");
  scanf("%c", &ch);

  switch(ch)
  {
    case 'a':
    case 'A':
    case 'e':
    case 'E':
    case 'i':
    case 'I':
    case 'o':
    case 'O':
    case 'u':
    case 'U':
      printf("%c is a vowel.\n", ch);
      break;
    default:
      printf("%c is not a vowel.\n", ch);
  }

  return 0;
}
```

3. Function to check vowel

```
int check_vowel(char a)
{
    if (a >= 'A' && a <= 'Z')
       a = a + 'a' - 'A';   /* Converting to lower case or use a = a + 32 */

    if (a == 'a' || a == 'e' || a == 'i' || a == 'o' || a == 'u')
       return 1;

    return 0;
}
```

C program to check leap year

```
#include <stdio.h>

int main()
{
  int year;

  printf("Enter a year to check if it is a leap year\n");
  scanf("%d", &year);

  if ( year%400 == 0)
    printf("%d is a leap year.\n", year);
  else if ( year%100 == 0)
    printf("%d is not a leap year.\n", year);
  else if ( year%4 == 0 )
    printf("%d is a leap year.\n", year);
  else
    printf("%d is not a leap year.\n", year);

  return 0;
}
```

Add digits of number in c

C program to add digits of a number: Here we are using modulus operator(%) to extract individual digits of number and adding them.

1. C programming code

```
#include <stdio.h>

int main()
{
   int n, t, sum = 0, remainder;

   printf("Enter an integer\n");
   scanf("%d", &n);

   t = n;
```

```
    while (t != 0)
    {
        remainder = t % 10;
        sum       = sum + remainder;
        t         = t / 10;
    }

    printf("Sum of digits of %d = %d\n", n, sum);

    return 0;
}
```

Output of program:
sum = sum + remainder
so sum = 8 now.
98/10 = 9 because in c whenever we divide integer by another integer we get an integer.
9%10 = 9
sum = 8(previous value) + 9
sum = 17
9/10 = 0.
So finally n = 0, loop ends we get the required sum.

2. Find sum of digits in c without modulus operator

C program to find sum of digit of an integer which does not use modulus operator. Our program uses a character array (string) for storing an integer. We convert every character of string into an integer and add all these integers.

```
#include <stdio.h>

int main()
{
    int c, sum, t;
    char n[1000];

    printf("Input an integer\n");
    scanf("%s", n);

    sum = c = 0;

    while (n[c] != '\0') {
        t   = n[c] - '0'; // Converting character to integer
        sum = sum + t;
        c++;
    }

    printf("Sum of digits of %s = %d\n", n, sum);

    return 0;
}
```

An advantage of this method is that input integer can be very large which may not be stored in int or long long data type see an example below in output.

3. Add digits using recursion

```
#include <stdio.h>

int add_digits(int);

int main()
{
  int n, result;

  scanf("%d", &n);

  result = add_digits(n);

  printf("%d\n", result);

  return 0;
}

int add_digits(int n) {
  static int sum = 0;

  if (n == 0) {
    return 0;
  }

  sum = n%10 + add_digits(n/10);

  return sum;
}
```

Factorial program in c

Factorial program in c: c code to find and print factorial of a number, three methods are given, first one uses for loop, second uses a functionto find factorial and third using recursion. Factorial is represented using '!', so five factorial will be written as (5!), n factorial as (n!). Also n! = n*(n-1)*(n-2)*(n-3)...3.2.1 and zero factorial is defined as one i.e. 0! = 1.

1. Factorial program in c using for loop

Here we find factorial using for loop.

```
#include <stdio.h>

int main()
{
```

```
  int c, n, fact = 1;

  printf("Enter a number to calculate it's factorial\n");
  scanf("%d", &n);

  for (c = 1; c <= n; c++)
    fact = fact * c;

  printf("Factorial of %d = %d\n", n, fact);

  return 0;
}
```

2. Factorial program in c using function

```
#include <stdio.h>

long factorial(int);

int main()
{
  int number;
  long fact = 1;

  printf("Enter a number to calculate it's factorial\n");
  scanf("%d", &number);

  printf("%d! = %ld\n", number, factorial(number));

  return 0;
}

long factorial(int n)
{
  int c;
  long result = 1;

  for (c = 1; c <= n; c++)
    result = result * c;

  return result;
}
```

3. Factorial program in c using recursion

```
#include<stdio.h>

long factorial(int);

int main()
{
  int n;
  long f;

  printf("Enter an integer to find factorial\n");
```

```
  scanf("%d", &n);

  if (n < 0)
    printf("Negative integers are not allowed.\n");
  else
  {
    f = factorial(n);
    printf("%d! = %ld\n", n, f);
  }

  return 0;
}

long factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}
```

Recursion is a technique in which a function calls itself, for example in above code factorial function is calling itself. To solve a problem using recursion you must first express its solution in recursive form.

C program to find hcf and lcm

C program to find hcf and lcm: The code below finds highest common factor and least common multiple of two integers. HCF is also known as greatest common divisor(GCD) or greatest common factor(gcf).

1. C programming code

```
#include <stdio.h>

int main() {
  int a, b, x, y, t, gcd, lcm;

  printf("Enter two integers\n");
  scanf("%d%d", &x, &y);

  a = x;
  b = y;

  while (b != 0) {
    t = b;
    b = a % b;
    a = t;
  }

  gcd = a;
  lcm = (x*y)/gcd;

  printf("Greatest common divisor of %d and %d = %d\n", x, y, gcd);
```

```
  printf("Least common multiple of %d and %d = %d\n", x, y, lcm);

  return 0;
}
```

2. C program to find hcf and lcm using recursion

```
#include <stdio.h>

long gcd(long, long);

int main() {
  long x, y, hcf, lcm;

  printf("Enter two integers\n");
  scanf("%ld%ld", &x, &y);

  hcf = gcd(x, y);
  lcm = (x*y)/hcf;

  printf("Greatest common divisor of %ld and %ld = %ld\n", x, y, hcf);
  printf("Least common multiple of %ld and %ld = %ld\n", x, y, lcm);

  return 0;
}

long gcd(long a, long b) {
  if (b == 0) {
    return a;
  }
  else {
    return gcd(b, a % b);
  }
}
```

3. C program to find hcf and lcm using function

```
#include <stdio.h>

long gcd(long, long);

int main() {
  long x, y, hcf, lcm;

  printf("Enter two integers\n");
  scanf("%ld%ld", &x, &y);

  hcf = gcd(x, y);
  lcm = (x*y)/hcf;

  printf("Greatest common divisor of %ld and %ld = %ld\n", x, y, hcf);
  printf("Least common multiple of %ld and %ld = %ld\n", x, y, lcm);

  return 0;
}
```

```
long gcd(long x, long y) {
  if (x == 0) {
    return y;
  }

  while (y != 0) {
    if (x > y) {
      x = x - y;
    }
    else {
      y = y - x;
    }
  }

  return x;
}
```

Decimal to binary conversion

C program to convert decimal to binary: c language code to convert an integer from decimal number system(base-10) to binary number system(base-2). Size of integer is assumed to be 32 bits. We use bitwise operators to perform the desired task. We right shift the original number by 31, 30, 29, ..., 1, 0 bits using a loop and bitwise AND the number obtained with 1(one), if the result is 1 then that bit is 1 otherwise it is 0(zero).

1. C programming code

```
#include <stdio.h>

int main()
{
  int n, c, k;

  printf("Enter an integer in decimal number system\n");
  scanf("%d", &n);

  printf("%d in binary number system is:\n", n);

  for (c = 31; c >= 0; c--)
  {
    k = n >> c;

    if (k & 1)
      printf("1");
    else
      printf("0");
  }

  printf("\n");

  return 0;
}
```

2. C code to store decimal to binary conversion in a string

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

char *decimal_to_binary(int);

main()
{
   int n, c, k;
   char *pointer;

   printf("Enter an integer in decimal number system\n");
   scanf("%d",&n);

   pointer = decimal_to_binary(n);
   printf("Binary string of %d is: %s\n", n, t);

   free(pointer);

   return 0;
}

char *decimal_to_binary(int n)
{
   int c, d, count;
   char *pointer;

   count = 0;
   pointer = (char*)malloc(32+1);

   if ( pointer == NULL )
      exit(EXIT_FAILURE);

   for ( c = 31 ; c >= 0 ; c-- )
   {
      d = n >> c;

      if ( d & 1 )
         *(pointer+count) = 1 + '0';
      else
         *(pointer+count) = 0 + '0';

      count++;
   }
   *(pointer+count) = '\0';

   return  pointer;
}
```

Memory is allocated dynamically because we can't return a pointer to a local variable (character array in this case). If we return a pointer to local variable then program may crash or we get incorrect result.

C program to find ncr and npr

1. C program to find nCr using function

```c
#include <stdio.h>

long factorial(int);
long find_ncr(int, int);
long find_npr(int, int);

int main()
{
   int n, r;
   long ncr, npr;

   printf("Enter the value of n and r\n");
   scanf("%d%d",&n,&r);

   ncr = find_ncr(n, r);
   npr = find_npr(n, r);

   printf("%dC%d = %ld\n", n, r, ncr);
   printf("%dP%d = %ld\n", n, r, npr);

   return 0;
}

long find_ncr(int n, int r) {
   long result;

   result = factorial(n)/(factorial(r)*factorial(n-r));

   return result;
}

long find_npr(int n, int r) {
   long result;

   result = factorial(n)/factorial(n-r);

   return result;
}

long factorial(int n) {
   int c;
   long result = 1;

   for (c = 1; c <= n; c++)
      result = result*c;

   return result;
}
```

2. Another way to calculate nPr and nCr using functions

We use long long data type in our program to handle large numbers.

```
#include <stdio.h>
#define ll long long

void find_ncr_npr(int, int, ll*, ll*);
ll find_npr(int, int);
ll factorial(int);

int main() {
   int n, r;
   ll ncr, npr;

   printf("Input n and r\n");
   scanf("%d%d", &n, &r);

   find_ncr_npr(n, r, &npr, &ncr);

   printf("%dC%d = %lld\n", n, r, ncr);
   printf("%dP%d = %lld\n", n, r, npr);

   return 0;
}

void find_ncr_npr(int n, int r, ll *npr, ll *ncr) {
   *npr = find_npr(n, r);
   *ncr = *npr/factorial(r);
}

ll find_npr(int n, int r) {
   ll result = 1;
   int c = 1;

   while (c <= r) {
      result = result * (n - r + c);
      c++;
   }

   return result;
}

ll factorial(int n) {
   int c;
   ll result = 1;

   for (c = 1; c <= n; c++)
      result = result*c;

   return result;
}
```

C program to add n numbers

This c program add n numbers which will be entered by the user. Firstly user will enter a number indicating how many numbers user wishes to add and then user will enter n numbers. In the first c program to add numbers we are not using an array, and using array in the second code.

1. C programming code

```
#include <stdio.h>

int main()
{
   int n, sum = 0, c, value;

   printf("Enter the number of integers you want to add\n");
   scanf("%d", &n);

   printf("Enter %d integers\n",n);

   for (c = 1; c <= n; c++)
   {
      scanf("%d", &value);
      sum = sum + value;
   }

   printf("Sum of entered integers = %d\n",sum);

   return 0;
}
```

2. C programming code using array

```
#include <stdio.h>

int main()
{
   int n, sum = 0, c, array[100];

   scanf("%d", &n);

   for (c = 0; c < n; c++)
   {
      scanf("%d", &array[c]);
      sum = sum + array[c];
   }

   printf("Sum = %d\n",sum);

   return 0;
}
```

3. Add n numbers using recursion

```c
#include <stdio.h>

long calculateSum(int [], int);

int main()
{
   int n, c, array[100];
   long result;

   scanf("%d", &n);

   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);

   result = calculateSum(array, n);

   printf("Sum = %ld\n", result);

   return 0;
}

long calculateSum(int a[], int n) {
   static long sum = 0;

   if (n == 0)
      return sum;

   sum = sum + a[n-1];

   return calculateSum(a, --n);
}
```

1. Swapping of two numbers in c

```c
#include <stdio.h>

int main()
{
   int x, y, temp;

   printf("Enter the value of x and y\n");
   scanf("%d%d", &x, &y);

   printf("Before Swapping\nx = %d\ny = %d\n",x,y);

   temp = x;
   x    = y;
   y    = temp;

   printf("After Swapping\nx = %d\ny = %d\n",x,y);

   return 0;
}
```

2. Swapping of two numbers without third variable

You can also swap two numbers *without* using temp or temporary or third variable. In that case c program will be as shown :-

```
#include <stdio.h>

int main()
{
   int a, b;

   printf("Enter two integers to swap\n");
   scanf("%d%d", &a, &b);

   a = a + b;
   b = a - b;
   a = a - b;

   printf("a = %d\nb = %d\n",a,b);
   return 0;
}
```

To understand above logic simply choose a as 7 and b as 9 and then do what is written in program. You can choose any other combination of numbers as well. Sometimes it's a good way to understand a program.

3. Swap two numbers using pointers

```
#include <stdio.h>

int main()
{
   int x, y, *a, *b, temp;

   printf("Enter the value of x and y\n");
   scanf("%d%d", &x, &y);

   printf("Before Swapping\nx = %d\ny = %d\n", x, y);

   a = &x;
   b = &y;

   temp = *b;
   *b   = *a;
   *a   = temp;

   printf("After Swapping\nx = %d\ny = %d\n", x, y);

   return 0;
}
```

4. Swapping numbers using call by reference

In this method we will make a function to swap numbers.

```
#include <stdio.h>

void swap(int*, int*);

int main()
{
   int x, y;

   printf("Enter the value of x and y\n");
   scanf("%d%d",&x,&y);

   printf("Before Swapping\nx = %d\ny = %d\n", x, y);

   swap(&x, &y);

   printf("After Swapping\nx = %d\ny = %d\n", x, y);

   return 0;
}

void swap(int *a, int *b)
{
   int temp;

   temp = *b;
   *b   = *a;
   *a   = temp;
}
```

5. C programming code to swap using bitwise XOR

```
#include <stdio.h>

int main()
{
  int x, y;

  scanf("%d%d", &x, &y);

  printf("x = %d\ny = %d\n", x, y);

  x = x ^ y;
  y = x ^ y;
  x = x ^ y;

  printf("x = %d\ny = %d\n", x, y);

  return 0;
}
```

Swapping is used in sorting algorithms that is when we wish to arrange numbers in a particular order either in ascending order or in descending.

C program to reverse a number

C Program to reverse a number :- This program reverse the number entered by the user and then prints the reversed number on the screen. For example if user enter 123 as input then 321 is printed as output. In our program we use modulus(%) operator to obtain the digits of a number. To invert number look at it and write it from opposite direction or the output of code is a number obtained by writing original number from right to left. To reverse or invert large numbers use long data type or long long data type if your compiler supports it, if you still have large numbers then use strings or other data structure.

1. C programming code

```
#include <stdio.h>

int main()
{
   int n, reverse = 0;

   printf("Enter a number to reverse\n");
   scanf("%d", &n);

   while (n != 0)
   {
      reverse = reverse * 10;
      reverse = reverse + n%10;
      n       = n/10;
   }

   printf("Reverse of entered number is = %d\n", reverse);

   return 0;
}
```

2.C program to reverse number using recursion

```
#include <stdio.h>

long reverse(long);

int main()
{
   long n, r;

   scanf("%ld", &n);

   r = reverse(n);

   printf("%ld\n", r);

   return 0;
}
```

```
long reverse(long n) {
   static long r = 0;

   if (n == 0)
      return 0;

   r = r * 10;
   r = r + n % 10;
   reverse(n/10);
   return r;
}
```

Palindrome Numbers

Palindrome number in c: A palindrome number is a number such that if we reverse it, it will not change. For example some palindrome numbers examples are 121, 212, 12321, -454. To check whether a number is palindrome or not first we reverse it and then compare the number obtained with the original, if both are same then number is palindrome otherwise not. C program for palindrome number is given below.

Palindrome number algorithm

1. Get the number from user.
2. Reverse it.
3. Compare it with the number entered by the user.
4. If both are same then print palindrome number
5. Else print not a palindrome number.

Palindrome number program c

```
#include <stdio.h>

int main()
{
   int n, reverse = 0, temp;

   printf("Enter a number to check if it is a palindrome or not\n");
   scanf("%d",&n);

   temp = n;

   while( temp != 0 )
   {
      reverse = reverse * 10;
      reverse = reverse + temp%10;
      temp = temp/10;
   }

   if ( n == reverse )
      printf("%d is a palindrome number.\n", n);
   else
```

```
      printf("%d is not a palindrome number.\n", n);

   return 0;
}
```

C program to print patterns of numbers and stars

These program prints various different patterns of numbers and stars. These codes illustrate how to create various patterns using c programming. Most of these c programs involve usage of nested loops and space. A pattern of numbers, star or characters is a way of arranging these in some logical manner or they may form a sequence. Some of these patterns are triangles which have special importance in mathematics. Some patterns are symmetrical while other are not. Please see the complete page and look at comments for many different patterns.

```
    *

   ***

  *****

 *******

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

We have shown five rows above, in the program you will be asked to enter the numbers of rows you want to print in the pyramid of stars.

1. C programming code

```
#include <stdio.h>

int main()
{
   int row, c, n, temp;

   printf("Enter the number of rows in pyramid of stars you wish to see ");
   scanf("%d",&n);

   temp = n;

   for ( row = 1 ; row <= n ; row++ )
   {
      for ( c = 1 ; c < temp ; c++ )
         printf(" ");

      temp--;

      for ( c = 1 ; c <= 2*row - 1 ; c++ )
         printf("*");

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

```
    return 0;
}
```

2. Consider the pattern

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

to print above pattern see the code below:

```
#include <stdio.h>

int main()
{
    int n, c, k;

    printf("Enter number of rows\n");
    scanf("%d",&n);

    for ( c = 1 ; c <= n ; c++ )
    {
        for( k = 1 ; k <= c ; k++ )
            printf("*");

        printf("\n");
    }

    return 0;
}
```

Using these examples you are in a better position to create your desired pattern for yourself. Creating a pattern involves how to use nested loops properly, some pattern may involve alphabets or other special characters. Key aspect is knowing how the characters in pattern changes.

3. C pattern programs

Pattern:

```
   *
  *A*
 *A*A*
*A*A*A*.
```

4. C pattern program of stars and alphabets:

```
#include<stdio.h>

main()
{
    int n, c, k, space, count = 1;

    printf("Enter number of rows\n");
    scanf("%d",&n);

    space = n;

    for ( c = 1 ; c <= n ; c++)
    {
        for( k = 1 ; k < space ; k++)
           printf(" ");

        for ( k = 1 ; k <= c ; k++)
        {
            printf("*");

            if ( c > 1 && count < c)
            {
                 printf("A");
                 count++;
            }
        }

        printf("\n");
        space--;
        count = 1;
    }
    return 0;
}
```

5.Pattern:

```
        1

       232

      34543

     4567654

    567898765
```

C program:

```
#include<stdio.h>

main()
{
      int n, c, d, num = 1, space;
```

```
        scanf("%d",&n);

        space = n - 1;

        for ( d = 1 ; d <= n ; d++ )
        {
            num = d;

            for ( c = 1 ; c <= space ; c++ )
                printf(" ");

            space--;

            for ( c = 1 ; c <= d ; c++ )
            {
                printf("%d", num);
                num++;
            }
            num--;
            num--;
            for ( c = 1 ; c < d ; c++)
            {
                printf("%d", num);
                num--;
            }
            printf("\n");

        }

        return 0;
}
```

C program to print diamond pattern

Diamond pattern in c: This code print diamond pattern of stars. Diamond shape is as follows:

```
  *

 ***

*****

 ***

  *
```

1. C programming code

```
#include <stdio.h>

int main()
{
  int n, c, k, space = 1;
```

```
  printf("Enter number of rows\n");
  scanf("%d", &n);

  space = n - 1;

  for (k = 1; k <= n; k++)
  {
    for (c = 1; c <= space; c++)
      printf(" ");

    space--;

    for (c = 1; c <= 2*k-1; c++)
      printf("*");

    printf("\n");
  }

  space = 1;

  for (k = 1; k <= n - 1; k++)
  {
    for (c = 1; c <= space; c++)
      printf(" ");

    space++;

    for (c = 1 ; c <= 2*(n-k)-1; c++)
      printf("*");

    printf("\n");
  }

  return 0;
}
```

2. C program to print diamond using recursion

```
#include <stdio.h>

void print (int);

int main () {
   int rows;

   scanf("%d", &rows);

   print(rows);

   return 0;
}

void print (int r) {
   int c, space;
   static int stars = -1;
```

```
    if (r <= 0)
      return;

    space = r - 1;
    stars += 2;

    for (c = 0; c < space; c++)
       printf(" ");

    for (c = 0; c < stars; c++)
       printf("*");

    printf("\n");

    print(--r);

    space = r + 1;
    stars -= 2;

    for (c = 0; c < space; c++)
       printf(" ");

    for (c = 0; c < stars; c++)
       printf("*");

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

C program for prime number

Prime number program in c: c program for prime number, this code prints prime numbers using c programming language. To check whether a number is prime or not see another code below. Prime number logic: a number is prime if it is divisible only by one and itself. Remember two is the only even and also the smallest prime number. First few prime numbers are 2, 3, 5, 7, 11, 13, 17....etc. Prime numbers have many applications in computer science and mathematics. A number greater than one can be factorized into prime numbers, For example $540 = 2^2*3^3*5^1$

1. Prime number program in c language

```
#include<stdio.h>

int main()
{
   int n, i = 3, count, c;

   printf("Enter the number of prime numbers required\n");
   scanf("%d",&n);

   if ( n >= 1 )
   {
      printf("First %d prime numbers are :\n",n);
      printf("2\n");
   }
```

```
    for ( count = 2 ; count <= n ;  )
    {
       for ( c = 2 ; c <= i - 1 ; c++ )
       {
          if ( i%c == 0 )
             break;
       }
       if ( c == i )
       {
          printf("%d\n",i);
          count++;
       }
       i++;
    }

    return 0;
}
```

2. C program for prime number or not

```
#include<stdio.h>

main()
{
   int n, c = 2;

   printf("Enter a number to check if it is prime\n");
   scanf("%d",&n);

   for ( c = 2 ; c <= n - 1 ; c++ )
   {
      if ( n%c == 0 )
      {
         printf("%d is not prime.\n", n);
         break;
      }
   }
   if ( c == n )
      printf("%d is prime.\n", n);

   return 0;
}
```

3. C program for prime number using function

```
#include<stdio.h>

int check_prime(int);

main()
{
   int n, result;

   printf("Enter an integer to check whether it is prime or not.\n");
```

```
    scanf("%d",&n);

    result = check_prime(n);

    if ( result == 1 )
       printf("%d is prime.\n", n);
    else
       printf("%d is not prime.\n", n);

    return 0;
}

int check_prime(int a)
{
    int c;

    for ( c = 2 ; c <= a - 1 ; c++ )
    {
       if ( a%c == 0 )
          return 0;
    }
    if ( c == a )
       return 1;
}
```

There are many logic to check prime numbers, one given below is more efficient then above method.

```
for ( c = 2 ; c <= (int)sqrt(n) ; c++ )
```

Only checking from 2 to square root of number is sufficient.
There are many more efficient logic available.

Armstrong number c program

Armstrong number c program: c programming code to check whether a number is Armstrong or not. Armstrong number is a number which is equal to sum of digits raise to the power total number of digits in the number. Some Armstrong numbers are: 0, 1, 2, 3, 153, 370, 407, 1634, 8208 etc. Read more about Armstrong numbers at Wikipedia. We will consider base 10 numbers in our program. Algorithm to check Armstrong is: First we calculate number of digits in our program and then compute sum of individual digits raise to the power number of digits. If this sum equals input number then number is Armstrong otherwise not an Armstrong number.

Examples:
7 = 7^1
371 = 3^3 + 7^3 + 1^3 (27 + 343 +1)
8208 = 8^4 + 2^4 +0^4 + 8^4 (4096 + 16 + 0 + 4096).
1741725 = 1^7 + 7^7 + 4^7 + 1^7 + 7^7 + 2^7 +5^7 (1 + 823543 + 16384 + 1 + 823543 +128 + 78125)

A NEW GENERATION OF COMPUTER

1. C programming code

```c
#include <stdio.h>

int power(int, int);

int main()
{
   int n, sum = 0, temp, remainder, digits = 0;

   printf("Input an integer\n");
   scanf("%d", &n);

   temp = n;
   // Count number of digits
   while (temp != 0) {
      digits++;
      temp = temp/10;
   }

   temp = n;

   while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
      temp = temp/10;
   }

   if (n == sum)
      printf("%d is an Armstrong number.\n", n);
   else
      printf("%d is not an Armstrong number.\n", n);

   return 0;
}

int power(int n, int r) {
   int c, p = 1;

   for (c = 1; c <= r; c++)
      p = p*n;

   return p;
}
```

2. C program to check Armstrong number using function

We will use long long data type in our program so that we can check numbers up to 2^64-1.

```c
#include <stdio.h>

int check_armstrong(long long);
long long power(int, int);

int main () {
   long long n;
```

```
   printf("Input a number\n");
   scanf("%lld", &n);

   if (check_armstrong(n) == 1)
      printf("%lld is an armstrong number.\n", n);
   else
      printf("%lld is not an armstrong number.\n", n);

   return 0;
}

int check_armstrong(long long n) {
   long long sum = 0, temp;
   int remainder, digits = 0;

   temp = n;

   while (temp != 0) {
      digits++;
      temp = temp/10;
   }

   temp = n;

   while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
      temp = temp/10;
   }

   if (n == sum)
      return 1;
   else
      return 0;
}

long long power(int n, int r) {
   int c;
   long long p = 1;

   for (c = 1; c <= r; c++)
      p = p*n;

   return p;
}
```

Output of program:

```
Input a number
35641594208964132
35641594208964132 is an Armstrong number.
```

3. C program to generate and print Armstrong numbers

C program to generate Armstrong numbers. In our program user will input two integers and we will print all Armstrong numbers between two integers. Using a for loop we will check numbers in the desired range. In our loop we call our function check Armstrong which returns 1 if number is Armstrong and 0 otherwise. If you are not familiar with Armstrong numbers

C programming code

```
#include <stdio.h>

int check_armstrong(int);
int power(int, int);

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

   printf("Input two integers\n");
   scanf("%d%d", &a, &b);

   for (c = a; c <= b; c++) {
      if (check_armstrong(c) == 1)
         printf("%d\n", c);
   }

   return 0;
}

int check_armstrong(int n) {
   long long sum = 0, temp;
   int remainder, digits = 0;

   temp = n;

   while (temp != 0) {
      digits++;
      temp = temp/10;
   }

   temp = n;

   while (temp != 0) {
      remainder = temp%10;
      sum = sum + power(remainder, digits);
      temp = temp/10;
   }

   if (n == sum)
      return 1;
   else
      return 0;
}

int  power(int n, int r) {
   int c, p = 1;
```

```
    for (c = 1; c <= r; c++)
       p = p*n;

    return p;
}
```

Fibonacci series in c

Fibonacci series in c programming: c program for Fibonacci series without and with recursion. Using the code below you can print as many numbers of terms of series as desired. Numbers of Fibonacci sequence are known as Fibonacci numbers. First few numbers of series are 0, 1, 1, 2, 3, 5, 8 etc, Except first two terms in sequence every other term is the sum of two previous terms, For example 8 = 3 + 5 (addition of 3, 5). This sequence has many applications in mathematics and Computer Science.

1. Fibonacci series in c using for loop

```
/* Fibonacci Series c language */
#include<stdio.h>

int main()
{
   int n, first = 0, second = 1, next, c;

   printf("Enter the number of terms\n");
   scanf("%d",&n);

   printf("First %d terms of Fibonacci series are :-\n",n);

   for ( c = 0 ; c < n ; c++ )
   {
      if ( c <= 1 )
         next = c;
      else
      {
         next = first + second;
         first = second;
         second = next;
      }
      printf("%d\n",next);
   }

   return 0;
}
```

2. Fibonacci series program in c using recursion

```
#include<stdio.h>

int Fibonacci(int);
```

```
main()
{
   int n, i = 0, c;

   scanf("%d",&n);

   printf("Fibonacci series\n");

   for ( c = 1 ; c <= n ; c++ )
   {
      printf("%d\n", Fibonacci(i));
      i++;
   }

   return 0;
}

int Fibonacci(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Recursion method is less efficient as it involves function calls which uses stack, also there are chances of stack overflow if function is called frequently for calculating larger Fibonacci numbers.

C program to print Floyd's triangle

C program to print Floyd's triangle:- This program prints Floyd's triangle. Number of rows of Floyd's triangle to print is entered by the user. First four rows of Floyd's triangle are as follows :-

1
2 3
4 5 6
7 8 9 10

It's clear that in Floyd's triangle nth row contains n numbers.

1. C programming code

```
#include <stdio.h>

int main()
{
  int n, i,  c, a = 1;

  printf("Enter the number of rows of Floyd's triangle to print\n");
  scanf("%d", &n);
```

```
  for (i = 1; i <= n; i++)
  {
    for (c = 1; c <= i; c++)
    {
      printf("%d ",a);
      a++;
    }
    printf("\n");
  }

  return 0;
}
```

2. C program to print Floyd's triangle using recursion

```
#include <stdio.h>

void print_floyd(int);

int main()
{
  int n, i,  c, a = 1;

  printf("Input number of rows of Floyd's triangle to print\n");
  scanf("%d", &n);

  print_floyd(n);

  return 0;
}

void print_floyd(int n) {
   static int row = 1, c = 1;
   int d;

   if (n <= 0)
      return;

   for (d = 1; d <= row; ++d)
      printf("%d ", c++);

   printf("\n");
   row++;

   print_floyd(--n);
}
```

C program to print Pascal triangle

Pascal Triangle in c: C program to print Pascal triangle which you might have studied in Binomial Theorem in Mathematics. Number of rows of Pascal triangle to print is entered by the user. First four rows of Pascal triangle are shown below :-

```
   1

  1 1

 1 2 1

1 3 3 1
```

1. Pascal triangle in c

```c
#include <stdio.h>

long factorial(int);

int main()
{
   int i, n, c;

   printf("Enter the number of rows you wish to see in pascal triangle\n");
   scanf("%d",&n);

   for (i = 0; i < n; i++)
   {
      for (c = 0; c <= (n - i - 2); c++)
         printf(" ");

      for (c = 0 ; c <= i; c++)
         printf("%ld ",factorial(i)/(factorial(c)*factorial(i-c)));

      printf("\n");
   }

   return 0;
}

long factorial(int n)
{
   int c;
   long result = 1;

   for (c = 1; c <= n; c++)
         result = result*c;

   return result;
}
```

C program to add two numbers using pointers

This program performs addition of two numbers using pointers. In our program we have two two integer variables x, y and two pointer variables p and q. Firstly we assign the addresses of x and y to p and q respectively and then assign the sum of x and y to variable sum. Note that & is address of operator and * is value at address operator.

1. C programming code

```c
#include <stdio.h>

int main()
{
   int first, second, *p, *q, sum;

   printf("Enter two integers to add\n");
   scanf("%d%d", &first, &second);

   p = &first;
   q = &second;

   sum = *p + *q;

   printf("Sum of entered numbers = %d\n",sum);

   return 0;
}
```

2. C program to add numbers using call by reference

```c
#include <stdio.h>

long add(long *, long *);

int main()
{
   long first, second, *p, *q, sum;

   printf("Input two integers to add\n");
   scanf("%ld%ld", &first, &second);

   sum = add(&first, &second);

   printf("(%ld) + (%ld) = (%ld)\n", first, second, sum);

   return 0;
}

long add(long *x, long *y) {
   long sum;

   sum = *x + *y;

   return sum;
```

```
}
```

C program for bubble sort

C program for bubble sort: c programming code for bubble sort to sort numbers or arrange them in ascending order. You can easily modify it to print numbers in descending order.

1. Bubble sort algorithm in c

```
/* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);

  return 0;
}
```

:

2. Bubble sort in c language using function

```
#include <stdio.h>

void bubble_sort(long [], long);
```

```
int main()
{
  long array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%ld", &n);

  printf("Enter %ld integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%ld", &array[c]);

  bubble_sort(array, n);

  printf("Sorted list in ascending order:\n");

  for ( c = 0 ; c < n ; c++ )
     printf("%ld\n", array[c]);

  return 0;
}

void bubble_sort(long list[], long n)
{
  long c, d, t;

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (list[d] > list[d+1])
      {
        /* Swapping */

        t         = list[d];
        list[d]   = list[d+1];
        list[d+1] = t;
      }
    }
  }
}
```

You can also sort strings using Bubble sort, it is less efficient as its average and worst case complexity is high, there are many other fast sorting algorithms

C program to find maximum element in array

1. C programming code

```
#include <stdio.h>

int main()
{
  int array[100], maximum, size, c, location = 1;

  printf("Enter the number of elements in array\n");
```

```
  scanf("%d", &size);

  printf("Enter %d integers\n", size);

  for (c = 0; c < size; c++)
    scanf("%d", &array[c]);

  maximum = array[0];

  for (c = 1; c < size; c++)
  {
    if (array[c] > maximum)
    {
       maximum  = array[c];
       location = c+1;
    }
  }

  printf("Maximum element is present at location %d and it's value is %d.\n",
location, maximum);
  return 0;
}
```

2. C programming code to find maximum using function

Our function returns index at which maximum element occur.

```
#include <stdio.h>

int find_maximum(int[], int);

int main() {
  int c, array[100], size, location, maximum;

  printf("Input number of elements in array\n");
  scanf("%d", &size);

  printf("Enter %d integers\n", size);

  for (c = 0; c < size; c++)
    scanf("%d", &array[c]);

  location = find_maximum(array, size);
  maximum  = array[location];

  printf("Maximum element location = %d and value = %d.\n", location + 1,
maximum);
  return 0;
}

int find_maximum(int a[], int n) {
  int c, max, index;

  max = a[0];
```

```
  index = 0;

  for (c = 1; c < n; c++) {
    if (a[c] > max) {
       index = c;
       max = a[c];
    }
  }

  return index;
}
```

3. C programming code using pointers

```
#include <stdio.h>

int main()
{
  long array[100], *maximum, size, c, location = 1;

  printf("Enter the number of elements in array\n");
  scanf("%ld", &size);

  printf("Enter %ld integers\n", size);

  for ( c = 0 ; c < size ; c++ )
    scanf("%ld", &array[c]);

  maximum  = array;
  *maximum = *array;

  for (c = 1; c < size; c++)
  {
    if (*(array+c) > *maximum)
    {
       *maximum = *(array+c);
       location = c+1;
    }
  }

  printf("Maximum element found at location %ld and it's value is %ld.\n",
location, *maximum);
  return 0;
}
```

C program to find minimum element in array

1. C programming code

```
#include <stdio.h>

int main()
{
    int array[100], minimum, size, c, location = 1;

    printf("Enter the number of elements in array\n");
```

```
    scanf("%d",&size);

    printf("Enter %d integers\n", size);

    for ( c = 0 ; c < size ; c++ )
        scanf("%d", &array[c]);

    minimum = array[0];

    for ( c = 1 ; c < size ; c++ )
    {
        if ( array[c] < minimum )
        {
           minimum = array[c];
           location = c+1;
        }
    }

    printf("Minimum element is present at location %d and it's value is
%d.\n", location, minimum);
    return 0;
}
```

If minimum occurs two or more times times in array then index at which it occurs first is printed or minimum value at smallest index. You can modify this code this code to print largest index at which minimum occur. You can also store all indices at which minimum occur in an array.

2. C programming code to find minimum using function

Our function returns index at which minimum element occur.

```
#include <stdio.h>

int find_minimum(int[], int);

int main() {
  int c, array[100], size, location, minimum;

  printf("Input number of elements in array\n");
  scanf("%d", &size);

  printf("Input %d integers\n", size);

  for (c = 0; c < size; c++)
    scanf("%d", &array[c]);

  location = find_minimum(array, size);
  minimum  = array[location];

  printf("Minimum element location = %d and value = %d.\n", location + 1,
minimum);
  return 0;
}

int find_minimum(int a[], int n) {
```

```
  int c, min, index;

  min = a[0];
  index = 0;

  for (c = 1; c < n; c++) {
    if (a[c] < min) {
       index = c;
       min = a[c];
    }
  }

  return index;
}
```

3. C programming code using pointers

```
#include <stdio.h>

int main()
{
    int array[100], *minimum, size, c, location = 1;

    printf("Enter the number of elements in array\n");
    scanf("%d",&size);

    printf("Enter %d integers\n", size);

    for ( c = 0 ; c < size ; c++ )
        scanf("%d", &array[c]);

    minimum = array;
    *minimum = *array;

    for ( c = 1 ; c < size ; c++ )
    {
        if ( *(array+c) < *minimum )
        {
           *minimum = *(array+c);
           location = c+1;
        }
    }

    printf("Minimum element found at location %d and it's value is %d.\n",
location, *minimum);
    return 0;
}
```

Linear search in c

1. Linear search c program

```
#include <stdio.h>

int main()
```

```
{
   int array[100], search, c, n;

   printf("Enter the number of elements in array\n");
   scanf("%d",&n);

   printf("Enter %d integer(s)\n", n);

   for (c = 0; c < n; c++)
      scanf("%d", &array[c]);

   printf("Enter the number to search\n");
   scanf("%d", &search);

   for (c = 0; c < n; c++)
   {
      if (array[c] == search)     /* if required element found */
      {
         printf("%d is present at location %d.\n", search, c+1);
         break;
      }
   }
   if (c == n)
      printf("%d is not present in array.\n", search);

   return 0;
}
```

2. Linear search for multiple occurrences

In the code below we will print all the locations at which required element is found and also the number of times it occur in the list.

```
#include <stdio.h>

int main()
{
   int array[100], search, c, n, count = 0;

   printf("Enter the number of elements in array\n");
   scanf("%d", &n);

   printf("Enter %d numbers\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   printf("Enter the number to search\n");
   scanf("%d", &search);

   for (c = 0; c < n; c++) {
      if (array[c] == search) {
         printf("%d is present at location %d.\n", search, c+1);
         count++;
      }
```

```
    }
    if (count == 0)
       printf("%d is not present in array.\n", search);
    else
       printf("%d is present %d times in array.\n", search, count);

    return 0;
}
```

3. C program for linear search using function

```
#include <stdio.h>

long linear_search(long [], long, long);

int main()
{
    long array[100], search, c, n, position;

    printf("Input number of elements in array\n");
    scanf("%ld", &n);

    printf("Input %d numbers\n", n);

    for (c = 0; c < n; c++)
       scanf("%ld", &array[c]);

    printf("Input number to search\n");
    scanf("%ld",&search);

    position = linear_search(array, n, search);

    if (position == -1)
       printf("%d is not present in array.\n", search);
    else
       printf("%d is present at location %d.\n", search, position+1);

    return 0;
}

long linear_search(long a[], long n, long find) {
    long c;

    for (c = 0 ;c < n ; c++ ) {
       if (a[c] == find)
          return c;
    }

    return -1;
}
```

4. Linear search function using pointers

```
long linear_search(long *pointer, long n, long find)
{
```

```
   long c;

   for (c = 0; c < n; c++) {
      if (*(pointer+c) == find)
         return c;
   }

   return -1;
}
```

C program for binary search

1. C programming code for binary search

```
#include <stdio.h>

int main()
{
   int c, first, last, middle, n, search, array[100];

   printf("Enter number of elements\n");
   scanf("%d",&n);

   printf("Enter %d integers\n", n);

   for (c = 0; c < n; c++)
      scanf("%d",&array[c]);

   printf("Enter value to find\n");
   scanf("%d", &search);

   first = 0;
   last = n - 1;
   middle = (first+last)/2;

   while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
         printf("%d found at location %d.\n", search, middle+1);
         break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
   }
   if (first > last)
      printf("Not found! %d is not present in the list.\n", search);

   return 0;
}
```

C program to reverse an array

C program to reverse an array: This program reverses the array elements. For example if a is an array of integers with three elements such that
a[0] = 1
a[1] = 2
a[2] = 3
Then on reversing the array will be
a[0] = 3
a[1] = 2
a[0] = 1

Given below is the c code to reverse an array.

1. C programming code

```
#include <stdio.h>

int main()
{
   int n, c, d, a[100], b[100];

   printf("Enter the number of elements in array\n");
   scanf("%d", &n);

   printf("Enter the array elements\n");

   for (c = 0; c < n ; c++)
      scanf("%d", &a[c]);

   /*
    * Copying elements into array b starting from end of array a
    */

   for (c = n - 1, d = 0; c >= 0; c--, d++)
      b[d] = a[c];

   /*
    * Copying reversed array into original.
    * Here we are modifying original array, this is optional.
    */

   for (c = 0; c < n; c++)
      a[c] = b[c];

   printf("Reverse array is\n");

   for (c = 0; c < n; c++)
      printf("%d\n", a[c]);

   return 0;
}
```

2. Reverse array by swapping (without using additional memory)

```c
#include <stdio.h>

int main() {
  int array[100], n, c, t, end;

  scanf("%d", &n);
  end = n - 1;

  for (c = 0; c < n; c++) {
    scanf("%d", &array[c]);
  }

  for (c = 0; c < n/2; c++) {
    t           = array[c];
    array[c]    = array[end];
    array[end] = t;

    end--;
  }

  printf("Reversed array elements are:\n");

  for (c = 0; c < n; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

3. C program to reverse an array using pointers

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

void reverse_array(int*, int);

int main()
{
   int n, c, *pointer;

   scanf("%d",&n);

   pointer = (int*)malloc(sizeof(int)*n);

   if( pointer == NULL )
      exit(EXIT_FAILURE);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d",(pointer+c));

   reverse_array(pointer, n);

   printf("Original array on reversal is\n");
```

```
    for ( c = 0 ; c < n ; c++ )
       printf("%d\n",*(pointer+c));

    free(pointer);

    return 0;
}

void reverse_array(int *pointer, int n)
{
    int *s, c, d;

    s = (int*)malloc(sizeof(int)*n);

    if( s == NULL )
       exit(EXIT_FAILURE);

    for ( c = n - 1, d = 0 ; c >= 0 ; c--, d++ )
       *(s+d) = *(pointer+c);

    for ( c = 0 ; c < n ; c++ )
       *(pointer+c) = *(s+c);

    free(s);
}
```

C program to insert an element in an array

This code will insert an element into an array, For example consider an array a[10] having three elements in it initially and a[0] = 1, a[1] = 2 and a[2] = 3 and you want to insert a number 45 at location 1 i.e. a[0] = 45, so we have to move elements one step below so after insertion a[1] = 1 which was a[0] initially, and a[2] = 2 and a[3] = 3. Array insertion does not mean increasing its size i.e array will not be containing 11 elements.

C programming code

```
#include <stdio.h>

int main()
{
    int array[100], position, c, n, value;

    printf("Enter number of elements in array\n");
    scanf("%d", &n);

    printf("Enter %d elements\n", n);

    for (c = 0; c < n; c++)
       scanf("%d", &array[c]);

    printf("Enter the location where you wish to insert an element\n");
    scanf("%d", &position);

    printf("Enter the value to insert\n");
```

```
    scanf("%d", &value);

    for (c = n - 1; c >= position - 1; c--)
       array[c+1] = array[c];

    array[position-1] = value;

    printf("Resultant array is\n");

    for (c = 0; c <= n; c++)
       printf("%d\n", array[c]);

    return 0;
}
```

C program to delete an element from an array

This program delete an element from an array. Deleting an element does not affect the size of array. It is also checked whether deletion is possible or not, For example if array is containing five elements and you want to delete element at position six which is not possible.

C programming code

```
#include <stdio.h>

int main()
{
   int array[100], position, c, n;

   printf("Enter number of elements in array\n");
   scanf("%d", &n);

   printf("Enter %d elements\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   printf("Enter the location where you wish to delete element\n");
   scanf("%d", &position);

   if ( position >= n+1 )
      printf("Deletion not possible.\n");
   else
   {
      for ( c = position - 1 ; c < n - 1 ; c++ )
         array[c] = array[c+1];

      printf("Resultant array is\n");

      for( c = 0 ; c < n - 1 ; c++ )
         printf("%d\n", array[c]);
   }

   return 0;
}
```

C programming code to merge two sorted arrays

```c
#include <stdio.h>

void merge(int [], int, int [], int, int []);

int main() {
  int a[100], b[100], m, n, c, sorted[200];

  printf("Input number of elements in first array\n");
  scanf("%d", &m);

  printf("Input %d integers\n", m);
  for (c = 0; c < m; c++) {
    scanf("%d", &a[c]);
  }

  printf("Input number of elements in second array\n");
  scanf("%d", &n);

  printf("Input %d integers\n", n);
  for (c = 0; c < n; c++) {
    scanf("%d", &b[c]);
  }

  merge(a, m, b, n, sorted);

  printf("Sorted array:\n");

  for (c = 0; c < m + n; c++) {
    printf("%d\n", sorted[c]);
  }

  return 0;
}

void merge(int a[], int m, int b[], int n, int sorted[]) {
  int i, j, k;

  j = k = 0;

  for (i = 0; i < m + n;) {
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;
      }
      i++;
    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
```

```
        i++;
      }
    }
    else {
      for (; i < m + n;) {
        sorted[i] = a[j];
        j++;
        i++;
      }
    }
  }
}
```

Insertion sort algorithm implementation in c

```
/* insertion sort ascending order */

#include <stdio.h>

int main()
{
  int n, array[1000], c, d, t;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++) {
    scanf("%d", &array[c]);
  }

  for (c = 1 ; c <= n - 1; c++) {
    d = c;

    while ( d > 0 && array[d] < array[d-1]) {
      t          = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;

      d--;
    }
  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

Selection sort algorithm implementation in c

```
#include <stdio.h>

int main()
{
   int array[100], n, c, d, position, swap;

   printf("Enter number of elements\n");
   scanf("%d", &n);

   printf("Enter %d integers\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   for ( c = 0 ; c < ( n - 1 ) ; c++ )
   {
      position = c;

      for ( d = c + 1 ; d < n ; d++ )
      {
         if ( array[position] > array[d] )
            position = d;
      }
      if ( position != c )
      {
         swap = array[c];
         array[c] = array[position];
         array[position] = swap;
      }
   }

   printf("Sorted list in ascending order:\n");

   for ( c = 0 ; c < n ; c++ )
      printf("%d\n", array[c]);

   return 0;
}
```

C program to add two matrix

This c program add two matrices i.e. compute the sum of two matrices and then print it. Firstly user will be asked to enter the order of matrix (number of rows and columns) and then two matrices. For example if the user entered order as 2, 2 i.e. two rows and two columns and matrices as

First Matrix :-

1 2

3 4

Second matrix :-

4 5

-1 5
then output of the program (sum of First and Second matrix) will be
5 7
2 9

C programming code

```
#include <stdio.h>

int main()
{
   int m, n, c, d, first[10][10], second[10][10], sum[10][10];

   printf("Enter the number of rows and columns of matrix\n");
   scanf("%d%d", &m, &n);
   printf("Enter the elements of first matrix\n");

   for (c = 0; c < m; c++)
      for (d = 0; d < n; d++)
         scanf("%d", &first[c][d]);

   printf("Enter the elements of second matrix\n");

   for (c = 0; c < m; c++)
      for (d = 0 ; d < n; d++)
            scanf("%d", &second[c][d]);

   printf("Sum of entered matrices:-\n");

   for (c = 0; c < m; c++) {
      for (d = 0 ; d < n; d++) {
         sum[c][d] = first[c][d] + second[c][d];
         printf("%d\t", sum[c][d]);
      }
      printf("\n");
   }

   return 0;
}
```