FACQ TCS Level1 Coding Solutions

New Process for TCS Advanced Coding Round

➢ You will be given 2 coding questions.
➢ 1st coding question at Easy Level - 15 mins
➢ 2nd coding question at Moderate to High Level - 30 mins
➢ Cut off - 1 Question must be solved completely

Note: The new pattern does not have Command Line programming.

Your choice of programming languages is:

★ **C**
★ C++
★ Java
★ Python
★ Perl

Note: The **Python** tool takes a long time to compile compared to the **C** compiler. Your time is at its essence. Hence, you would be better off choosing C over Python in your coding test. However, you may choose Python if your skill level in it is better than C.

FACQ - Frequently Asked Coding Questions: TCS Advanced Coding

- **Level-1: Easy to Medium Complexity**
- Level-2: Medium to Hard Complexity

You will be given a separate set of programs with source codes for each level. **This PDF covers Level-1.** I highly recommend you use your foundational Knowledge and Practice these on a C compiler. **WARNING:** Just looking over the source codes doesn't help you crack this section. You will be better off if you: **Understand, Type, Compile, and Test with data!**

Level-1 Set-1 : Easy to Medium complexity

#	Topic	Frequency
1	HCF/GCD of numbers (Highest Common Factor or Greatest Common Divisor)	66
2	LCM of 2 numbers (Lowest Common Factor)	51

Program to Find GCD of two Numbers

The HCF or GCD of two integers is the largest integer that can exactly divide both numbers (without a remainder).
There are many ways to find the greatest common divisor in C programming.

GCD Using for loop and if Statement

```
#include <stdio.h>
int main()
{
    int n1, n2, i, gcd;

    printf("Enter two integers: ");
    scanf("%d %d", &n1, &n2);

    for(i=1; i <= n1 && i <= n2; ++i)
    {
        // Checks if i is factor of both integers
        if(n1%i==0 && n2%i==0)
            gcd = i;
    }

    printf("G.C.D of %d and %d is %d", n1, n2, gcd);

    return 0;
}
```

In this program, two integers entered by the user are stored in variable n1 and n2.Then, for loop is iterated until i is less than n1 and n2.

In each iteration, if both n1 and n2 are exactly divisible by i, the value of i is assigned to gcd.

When the for loop is completed, the greatest common divisor of two numbers is stored in variable gcd.

GCD Using while loop and if...else Statement

```
#include <stdio.h>
int main()
{
    int n1, n2;

    printf("Enter two positive integers: ");
    scanf("%d %d",&n1,&n2);

    while(n1!=n2)
    {
        if(n1 > n2)
            n1 -= n2;
        else
            n2 -= n1;
    }
    printf("GCD = %d",n1);

    return 0;
}
```

Output

```
Enter two positive integers: 81
153
GCD = 9
```

This is a better way to find the GCD. In this method, the smaller integer is subtracted from the larger integer, and the result is assigned to the variable holding the larger integer. This process is continued until n1 and n2 are equal.

The above two programs work as intended only if the user enters positive integers. Here's a little modification of the second example to find the GCD for both positive and negative integers.

GCD for both positive and negative numbers

```
#include <stdio.h>
int main()
{
   int n1, n2;

   printf("Enter two integers: ");
   scanf("%d %d",&n1,&n2);

   // if user enters negative number, sign of the number is changed to positive
   n1 = ( n1 > 0) ? n1 : -n1;
   n2 = ( n2 > 0) ? n2 : -n2;

   while(n1!=n2)
   {
      if(n1 > n2)
         n1 -= n2;
      else
         n2 -= n1;
   }
   printf("GCD = %d",n1);

   return 0;
}
```

Output

```
Enter two integers: 81
-153
GCD = 9
```

C Program to Find LCM of two Numbers

The LCM of two integers n1 and n2 is the smallest positive integer that is perfectly divisible by both n1 and n2 (without a remainder). For example, the LCM of 72 and 120 is 360.

```
#include <stdio.h>
int main() {
    int n1, n2, max;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);

    // maximum number between n1 and n2 is stored in max
    max = (n1 > n2) ? n1 : n2;

    while (1) {
        if (max % n1 == 0 && max % n2 == 0) {
            printf("The LCM of %d and %d is %d.", n1, n2, max);
            break;
        }
        ++max;
    }
    return 0;
}
```

Output

```
Enter two positive integers: 72
120
The LCM of 72 and 120 is 360.
```

In this program, the integers entered by the user are stored in variable n1 and n2 respectively. The largest number among n1 and n2 is stored in max. The LCM of two numbers cannot be less than max.

The LCM of two numbers can also be found using the formula:
LCM = (num1*num2)/GCD

Program to find the average of numbers using function

In this program, we have created a user defined function average() for the calculation of average. The numbers entered by user are passed to this function during function call.

```
#include <stdio.h>
float average(int a, int b){
   return (float)(a+b)/2;
}
int main()
{
   int num1, num2;
   float avg;

   printf("Enter first number: ");
   scanf("%d",&num1);
   printf("Enter second number: ");
   scanf("%d",&num2);

   avg = average(num1, num2);

   //%.2f is used for displaying output upto two decimal places
   printf("Average of %d and %d is: %.2f",num1,num2,avg);

   return 0;
}
```

Output:

```
Enter first number: 20
Enter second number: 13
Average of 20 and 13 is: 16.50
```

Program to reverse a number

```
#include <stdio.h>

int main() {

  int n, reverse = 0, remainder;
  printf("Enter an integer: ");
  scanf("%d", &n);

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

  printf("Reversed number = %d", reverse);

  return 0;
}
```

Output

```
Enter an integer: 2345
Reversed number = 5432
```

This program takes integer input from the user. Then the while loop is used until n != 0 is false (0).

In each iteration of the loop, the remainder when n is divided by 10 is calculated and the value of n is reduced by 10 times.

Inside the loop, the reversed number is computed using:

reverse = reverse * 10 + remainder;

Let us see how the while loop works when n = 2345.

n	n != 0	remainder	reverse
2345	true	5	0 * 10 + 5 = 5

234	true	4	5 * 10 + 4 = 54
23	true	3	54 * 10 + 3 = 543
2	true	2	543 * 10 + 2 = 5432
0	false	-	Loop terminates.

Finally, the reverse variable (which contains the reversed number) is printed on the screen.

Program to swap two numbers - Swap Numbers Using Temporary Variable

```
#include<stdio.h>
int main() {
  double first, second, temp;
  printf("Enter first number: ");
  scanf("%lf", &first);
  printf("Enter second number: ");
  scanf("%lf", &second);

  // value of first is assigned to temp
  temp = first;

  // value of second is assigned to first
  first = second;

  // value of temp (initial value of first) is assigned to second
  second = temp;

  // %.2lf displays number up to 2 decimal points
  printf("\nAfter swapping, first number = %.2lf\n", first);
  printf("After swapping, second number = %.2lf", second);
  return 0;
}
```

Output

```
Enter first number: 1.20
Enter second number: 2.45

After swapping, first number = 2.45
After swapping, second number = 1.20
```

In the above program, the temp variable is assigned the value of the first variable.
Then, the value of the first variable is assigned to the second variable.
Finally, the temp (which holds the initial value of first) is assigned to second. This completes the swapping process.

Swap Numbers Without Using Temporary Variables

```
#include <stdio.h>
int main() {
  double a, b;
  printf("Enter a: ");
  scanf("%lf", &a);
  printf("Enter b: ");
  scanf("%lf", &b);

  // swapping

  // a = (initial_a - initial_b)
  a = a - b;

  // b = (initial_a - initial_b) + initial_b = initial_a
  b = a + b;

  // a = initial_a - (initial_a - initial_b) = initial_b
  a = b - a;

  // %.2lf displays numbers up to 2 decimal places
  printf("After swapping, a = %.2lf\n", a);
  printf("After swapping, b = %.2lf", b);

  return 0;
}
```

Output

```
Enter a: 10.25
Enter b: -12.5
After swapping, a = -12.50
After swapping, b = 10.25
```

Program to Find Factorial of a Number

In this example, you will learn to calculate the factorial of a number entered by the user. The factorial of a positive number n is given by:
factorial of n (n!) = 1 * 2 * 3 * 4....n

The factorial of a negative number doesn't exist. And, the factorial of 0 is 1.

```
#include <stdio.h>
int main() {
   int n, i;
   unsigned long long fact = 1;
   printf("Enter an integer: ");
   scanf("%d", &n);

   // shows error if the user enters a negative integer
   if (n < 0)
      printf("Error! Factorial of a negative number doesn't exist.");
   else {
      for (i = 1; i <= n; ++i) {
         fact *= i;
      }
      printf("Factorial of %d = %llu", n, fact);
   }

   return 0;
}
```

Output

```
Enter an integer: 10
Factorial of 10 = 3628800
```

This program takes a positive integer from the user and computes the factorial using for loop.
Since the factorial of a number may be very large, the type of factorial variable is declared as unsigned long long.
If the user enters a negative number, the program displays a custom error message.

Factorial of a number using recursion.

```
#include<stdio.h>
long int multiplyNumbers(int n);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, multiplyNumbers(n));
    return 0;
}

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

Output

```
Enter a positive integer: 6
Factorial of 6 = 720
```

Suppose the user entered 6.
Initially, multiplyNumbers() is called from main() with 6 passed as an argument.
Then, 5 is passed to multiplyNumbers() from the same function (recursive call). In each recursive call, the value of argument n is decreased by 1.

When the value of n is less than 1, there is no recursive call and the factorial is returned ultimately to the main() function.

Program to Reverse a Sentence Using Recursion

Take a sentence from the user and reverse it using recursion.

```
#include <stdio.h>
void reverseSentence();
int main() {
   printf("Enter a sentence: ");
   reverseSentence();
   return 0;
}

void reverseSentence() {
   char c;
   scanf("%c", &c);
   if (c != '\n') {
      reverseSentence();
      printf("%c", c);
   }
}
```

Output

```
Enter a sentence: margorp emosewa
awesome program
```

This program first prints Enter a sentence: . Then, the reverseSentence() function is called.

This function stores the first letter entered by the user in c. If the variable is any character other than \n (newline), reverseSentence() is called again.

This process goes on until the user hits enter.

When the user hits enter, the reverseSentence() function starts printing characters from last.

Program to find both HCF and LCM

The code below finds the highest common factor and least common multiple of two integers. HCF is also known as greatest common divisor(GCD) or greatest common factor(gcf).

```
#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;
}
```

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);
        }
    }
```

Program to find Greatest among 10 numbers

```
#include <stdio.h>
 int main() {
   int a[10];
   int i;
   int greatest;
   printf("Enter ten values:");
   //Store 10 numbers in an array
   for (i = 0; i < 10; i++) {
      scanf("%d", &a[i]);
   }
   //Assume that a[0] is greatest
   greatest = a[0];
   for (i = 0; i < 10; i++) {
      if (a[i] > greatest) {
      greatest = a[i];
   }
   }
   printf("
   Greatest of ten numbers is %d", greatest);
   return 0;
 }
```

Program to display Fibonacci series within a range

```
#include<stdio.h>
void fibonacciSeries(int range)
{
   int a=0, b=1, c;
   while (a<=range)
```

```
    {
      printf("%d\t", a);
      c = a+b;
      a = b;
      b = c;
    }
}

int main()
{
    int range;

    printf("Enter range: ");
    scanf("%d", &range);

    printf("The fibonacci series is: \n");

    fibonacciSeries(range);

    return 0;
}
```

Program to convert Decimal to Binary
C Code: L1_tcs11_decimal2binary.c

```
//Convert decimal to binary
#include<stdio.h>
int main()
{
   //for initialize a variables
    long number, dec_num, rem, base = 1, bin = 0, count = 0;
    //To insert a number
    printf("Insert a decimal num \n");
    scanf("%ld", &number);
    dec_num = number;
    while(number > 0)
    {
        rem = number % 2;
        /*  To count no.of 1s */
        if (rem == 1)
        {
            count++;
        }

        bin = bin + rem * base;
       //number/=2;
        number = number / 2;
        base = base * 10;
    }
   //display
    printf("Input num is = %d\n", dec_num);
    printf("Its binary equivalent is = %ld\n", bin);
    printf("Num of 1's in the binary num is = %d\n", count);
   return 0;
}
```

Output:

```
Insert a decimal num
12
Input num is = 12
Its binary equivalent is = 1100
Num of 1's in the binary num is = 2
```

```
/** C program to convert the given Binary number into Decimal**/

#include<stdio.h>

int main()
{
      int  num, binary_val, decimal_val = 0, base = 1, rem;

      printf("Insert a binary num (1s and 0s) \n");
      scanf("%d", &num); /* maximum five digits */

      binary_val = num;
      while (num > 0)
      {
          rem = num % 10;
          decimal_val = decimal_val + rem * base;
         //num/=10;
          num = num / 10 ;
         //base*=2;
          base = base * 2;
      }
     //display binary number
      printf("The Binary num is = %d \n", binary_val);
    //display decimal number
      printf("Its decimal equivalent is = %d \n", decimal_val);
   return 0;
}
```

Output:

```
Insert a binary num (1s and 0s)
11101
The Binary num is = 11101
Its decimal equivalent is = 29
```

```
/** C program to convert the given Decimal number into Octal **/

#include<stdio.h>

int main()
 {
//Variable initialization
     long dec_num, rem, quotient;
     int i, j, octalno[100];
//Taking input from user
     printf("Enter a decimal number : ");
//Storing the value in dec_num variable
     scanf("%ld",&dec_num);
     quotient = dec_num;
     i=1;
//Storing the octal value in octalno[] array
     while (quotient!=0)
       {
        octalno[i++]=quotient%8;
        quotient=quotient/8;
       }
//Printing the octalno [] in reverse order
     printf("\nThe Octal of %ld is:",dec_num);
```

```
      for (j=i-1;j>0;j--)
      //display it
      printf ("%d", octalno[j]);
      return 0;
   }
```

Output:

Enter a decimal number : 15

The Octal of 15 is:17

/*

 * **C Program to Convert Binary to Octal**

Problem Solution

1. Take a binary number as input.

2. Divide the binary number into groups of 3 bits. For each group of 3 bits, multiply each bit with the power of 2 and add them consecutively.

3. Combine the result of all groups to get the output.

Program Explanation

1. Take a binary number as input and store it in the variable binarynum.
2. Obtain the remainder and quotient of the input number by dividing it by 10.
3. Multiply the obtained remainder with variable j and increment the variable octalnum with this value.
4. Increment the variable j by 2 and override the variable binarynum with the quotient obtained.
5. Repeat the steps 2-4 until the variable binarynum becomes zero.
6. Print the variable octalnum as output.

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

int main()
{
    long int binarynum, octalnum = 0, j = 1, remainder;

    printf("Enter the value for  binary number: ");
    scanf("%ld", &binarynum);
    while (binarynum != 0)
    {
        remainder = binarynum % 10;
        octalnum = octalnum + remainder * j;
        j = j * 2;
        binarynum = binarynum / 10;
    }
    printf("Equivalent octal value: %lo", octalnum);
    return 0;
}
```

Output:

```
Enter the value for  binary number: 1101
Equivalent octal value: 15
```

```
/*
```

C program to check a given string is Palindrome.

```
C program to read a string and check if it's a palindrome,
without using library functions. Display the result.

Problem Description
This program accepts a string and checks whether a given string is
palindrome.

Problem Solution
1. Take a string as input and store it in the array.
2. Reverse the string and store it in another array.
3. Compare both the arrays.

 */

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

void main()
{
    char string[25], reverse_string[25] = {'\0'};
    int  i, length = 0, flag = 0;

    fflush(stdin);
    printf("Enter a string \n");
    gets(string);
    /*  keep going through each character of the string till its end */
    for (i = 0; string[i] != '\0'; i++)
    {
        length++;
    }
    for (i = length - 1; i >= 0; i--)
    {
       reverse_string[length - i - 1] = string[i];
    }
    /*
     * Compare the input string and its reverse. If both are equal
     * then the input string is palindrome.
     */
    for (i = 0; i < length; i++)
    {
        if (reverse_string[i] == string[i])
            flag = 1;
        else
            flag = 0;
    }
    if (flag == 1)
        printf("%s is a palindrome \n", string);
    else
        printf("%s is not a palindrome \n", string);
}
```

Output:

```
Enter a string
PalinnilaP
PalinnilaP is a palindrome
Enter a string
hello
hello is not a palindrome
```

```
/*
```

Prime Number in C: A prime number is a natural number that is greater than 1 and is only divisible by 1 and itself. In other words, no number except the number itself, and 1 can divide a prime number.

Example: 2, 3, 5, 7, 11, 13, 17, 19 …., etc.

Problem Description
Write a C program to check if a given number is Prime number. If the number is Prime, then display it is a prime number else display it is not a prime number.

Problem Solution
1. Take a number as input.
2. Check if the number is divisible by any of the natural numbers starting from 2.
3. If it is, then it is not a prime number. Otherwise, it is a prime number.
4. Exit.

Optimized Approach:
Instead of iterating a loop from 2 to N/2, we will iterate through all the numbers starting from 2 to sqrt(N).
The reason for choosing sqrt(N) is that the factor of a number is always present between 2 to sqrt(N) and if we find a factor then we simply need to stop the iteration and print it is not a prime number else it is a prime number.

Examples:

N=36, we need to iterate the loop from 2 to 6(sqrt(36)). If we divide the number 36 by 2, we will get remainder equal to 0 which means 36 has a factor other than 1 and 36. Therefore, our output will be "36 is not a prime number".
N=53, sqrt(53) = 7. So, our loop will iterate from 2 to 7. If we divide the number 53 by any of these numbers (2,3,4,5,6,7) we will not get remainder equal to 0. Therefore our output will be "53 is a prime number".

Time Complexity: O(sqrt(n))
The above program for checking whether a number is prime or not has a time complexity of O(sqrt(n)) as the loop runs from 2 to sqrt(n),

```
where n is the number given as input.

Space Complexity: O(1)
In this program we are not initializing any array or other data types that
takes lot of storage. We are just initializing the variable.
Therefore, our space complexity is constant, i.e. O(1).
 */

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

void main()
{
    int num, j, flag;

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

    if (num <= 1)
    {
        printf("%d is not a prime numbers \n", num);
        exit(1);
    }
    flag = 0;
    // To check prime number
    for (j = 2; j <= sqrt(num); j++)
    {
        if ((num % j) == 0)
        {
            flag = 1;
            break;
        }
    }

    if (flag == 0)
        printf("%d is a prime number \n", num);
    else
        printf("%d is not a prime number \n", num);
}
```

Output:

```
Enter a number
53
53 is a prime number

Enter a number
45
45 is not a prime number
```

```
/*
Leap Year in C: A year is a Leap Year if it satisfies the following
conditions:

The year is exactly divisible by 400 (such as 2000,2400) or,
The year is exactly divisible by 4 (such as 2008, 2012, 2016) and not a
multiple of 100 (such as 1900, 2100, 2200).
Problem Description
Write a C Program to check whether a given year is a leap year or not.

Problem Solution
1. Take a year as input.
2. Check whether a given year is divisible by 400.
3. Check whether a given year is divisible by 100.
4. Check whether a given year is divisible by 4.
5. If the condition at step 2 and 4 becomes true, then the year is a leap
year.
6. If the condition at step 3 becomes true, then the year is not a leap
year.

Program Explanation
1. Take a year as input and store it in the variable year.
2. Using a single if, else statement check, if the year is completely
divisible by 400 or the year is completely divisible by 4 as well as not
divisible by 100.
3. If the above condition is true then print it is a leap year.
4. If the condition becomes false, then the year is not a leap year and
print the same.

Example:
```

```
Consider a year, 2200. Divide it by 400, we see that the remainder
obtained is not 0.
Therefore, we print “It is not a leap year”.

Time Complexity: O(1)
The above program for checking Leap Year has a time complexity of O(1) as
we use only
if else condition and no loops.

Space Complexity: O(1)
In the above program, space complexity is O(1) as no extra variable has
been taken
to store the values in the memory. All the variables initialized takes a
constant O(1)
space.
 */

/*
 * C program to find whether a given year is leap year or not
 */
#include<stdio.h>
void main()
{
    int year;

    printf("Enter a year \n");
    scanf("%d", &year);
    if (((year % 400) == 0)||(((year%4)==0)&&(year%100)!=0))
        printf("%d is a leap year \n", year);
    else
        printf("%d is not a leap year \n", year);
}

/*
Program Output

Enter a year
2012
2012 is a leap year
```

```
Enter a year
2009
2009 is not a leap year


Enter a year
2200
2200 is not a leap year

*/
```

```
/*
C program to find sum of digits of a number using recursion.

Problem Description
This C program finds the sum of digits of a number using recursion.

Problem Solution
The following C program, using recursion, finds the sum of its digits.

 */
#include <stdio.h>

int sum (int a);

int main()
{
    int num, result;

    printf("Enter the number: ");
    scanf("%d", &num);
    result = sum(num);
    printf("Sum of digits in %d is %d\n", num, result);
    return 0;
}

int sum (int num)
```

```
{
    if (num != 0)
    {
        return (num % 10 + sum (num / 10));
    }
    else
    {
       return 0;
    }
}
```

Output:
Enter the number: 123
Sum of digits in 123 is 6

```
/*
```

Find square root without using sqrt math function

```
*/
/*
//Using sqrt function
#include <stdio.h>
#include <math.h>
int main () {
   /// 2.000000
   printf("Square root of %lf is %lf\n", 4.0, sqrt(4.0) );
   /// 2.236068
   printf("Square root of %lf is %lf\n", 5.0, sqrt(5.0) );

   return(0);
}
*/

//Without using sqrt function
#include<stdio.h>

void main()
{
    int number;
    float temp, sqrt;

    printf("Provide the number: \n");

    scanf("%d", &number);

    // store the half of the given number e.g from 256 => 128
    sqrt = number / 2;
    temp = 0;

    // Iterate until sqrt is different of temp, that is updated on the
loop
    while(sqrt != temp){
        // initially 0, is updated with the initial value of 128
        // (on second iteration = 65)
        // and so on
        temp = sqrt;

        // Then, replace values (256 / 128 + 128 ) / 2 = 65
        // (on second iteration 34.46923076923077)
        // and so on
        sqrt = ( number/temp + temp) / 2;
    }

    printf("The square root of '%d' is '%f'", number, sqrt);
}
```

Output:

```
Provide the number:
81
The square root of '81' is '9.000000'
```

```
/*
Check if a 3-digit number is an Armstrong Number
```

```
In this approach, we will only check whether a 3-digit number entered by
the user
is an Armstrong number or not.

Armstrong numbers are 0, 1, 153, 370, 371, 407, etc.

Examples
If the entered number is 371, then 371 = 3*3*3 + 7*7*7 + 1*1*1 = 27 + 343
+ 1 = 371.
Since the sum of the cubes of every digit is equal to 371, the output will
be
“The given number is an Armstrong number.”
If the entered number is 150, then 150 = 1*1*1 + 5*5*5 + 0*0*0 = 1 + 125 +
0 = 126.
Since the sum of the cubes of every digit is not equal to 150, the output
will be
“The given number is not an Armstrong number”
*/
#include <stdio.h>
#include <math.h> // header file for pow() function

void main()
{
    int number, sum = 0, rem = 0, cube = 0, temp;

    printf ("Enter a number:");
    scanf("%d", &number);
    temp = number;
    while (number != 0)  // loop will continue until the number is not 0.
    {
        rem = number % 10;  // This will generate last digit of the
number.
        cube = pow(rem, 3); // The power of last digit will be calculated
here.
        sum = sum + cube;
        number = number / 10; // Now our least significant bit of the
number be removed.
    }
    if (sum == temp)
        printf ("The given number is an Armstrong number");
    else
        printf ("The given number is not an Armstrong number");
}
```

Output:

```
Enter a number:231
The given number is not an Armstrong number

Enter a number:407
The given number is an Armstrong number
```

Program Explanation

In this C program, we are reading the integer value using ‘number’ variable. An Armstrong number is an n-digit base b number, such that the sum of its digits raised to the power n is the number itself. Hence, 371 because 33 + 73 + 13 = 27 + 343 + 1 = 371.

Using while loop checks the value of ‘number’ variable is not equal to 0. If the condition is true, execute the iteration of the loop. The ‘rem’ variable is used to compute the modulus of the value of ‘number’ variable by 10 and ‘cube’ variable is used to compute the cube of the value of ‘rem’ variable using pow().

Then ‘sum’ variable is used to compute the summation of the value of ‘sum’ variable with the value of ‘cube’ variable. The If-else condition statement is used to check both the value of ‘sum’ variable and the value of ‘temp’ variable are equal. If the condition is true, then it will print Armstrong number. Otherwise, it will execute the else condition statement and print not Armstrong number.

Time Complexity: O(log(n))

The above program for checking whether a number is Armstrong or not has a time complexity of O(log(n)) as the while loop runs for log(n) times because at each iteration the number is divided by 10, where n is the number given as input.

Space Complexity: O(1)

In the above program, space complexity is O(1) as no extra variable is taken to store the value. All the variables initialized takes constant O(1) space.

```
/*
Check if N-digit number is an Armstrong Number
Check whether any N-digit number entered by the user is an Armstrong
number or not.

Examples
If the entered number is 1634, then 1634 = 14 + 64 + 34 + 44 = 1 + 1296 +
81 + 256 = 1634.
Since the sum of the cubes of every digit is equal to 1634, the output
will be
"The given number is an Armstrong number."
If the entered number is 1600, then 1600 = 14 + 64 + 04 + 04 = 1 + 1296 +
0 + 0 = 1297.
Since the sum of the cubes of every digit is not equal to 1600, the output
will be
"The given number is not an Armstrong number"
Program/Source Code:
*/
#include <stdio.h>
#include <math.h> // header file for pow() function

void main()
{
    int number, sum = 0, rem = 0, cube = 0, temp;

    printf ("Enter a number:");
    scanf("%d", &number);
    temp = number;
    while (number != 0)  // loop will continue until the number is not 0.
    {
        rem = number % 10;  // This will generate last digit of the
number.
        cube = pow(rem, 3); // The power of last digit will be calculated
here.
        sum = sum + cube;
        number = number / 10; // Now our least significant bit of the
number be removed.
    }
    if (sum == temp)
        printf ("The given number is an Armstrong number");
    else
        printf ("The given number is not an Armstrong number");
}

/*
Program Explanation
In our earlier program, we were only finding if a 3-digit number is
Armstrong or
not whereas in this approach we can check if a number is Armstrong or not
irrespective
of the count of digits in that given number.

The approach of this method is to calculate the power of each digit but
instead of
passing ‘3’ in the power section we need to pass the count of digits ‘n’.

The first while loop is calculating count of digits in the number and the
second
while loop is for finding the sum of the power of each digit and for
checking whether
the number is Armstrong or not.

Time Complexity: O(log(n))
O(log(n)) + O(log(n)) = O(log(n))
The above program for checking whether a number is Armstrong or not has a
time
complexity of O(log(n)) as both the while loop runs for log(n) times
because at
each iteration the number is divided by 10, where n is the number given as
input.

Space Complexity: O(1)
In the above program, space complexity is O(1) as no extra variable is
taken to
store the value. All the variables initialized takes constant O(1) space.

Program Output
```

```
Enter a number: 1634
The given number is an Armstrong number

Enter a number: 1600
The given number is not an Armstrong number
*/
```

```
/*
```

C Program to find area of a triangle given it’s three sides.

```
Problem Solution
The formula or algorithm used is:
Area = sqrt(s(s – a)(s – b)(s – c)),
where s = (a + b + c) / 2 or perimeter / 2. and a, b & c are the sides of
triangle.
 */
#include <stdio.h>
#include <math.h>

void main()
{
    int s, a, b, c, area;

    printf("Enter lengths of 3 sides of s triangle (a, b & c) :\n");
    scanf("%d %d %d", &a, &b, &c);
    /* compute s */
    s = (a + b + c) / 2;
    area = sqrt(s * (s - a) * (s - b) * (s - c));
    printf("Area of a triangle = %d \n", area);
}
```

Output:

```
Enter lengths of 3 sides of s triangle (a, b & c) :
3 5 4
Area of a triangle = 6
```

```
/*
Program Explanation
```

```
In this C program, library function defined in <math.h> header file is
used to
compute mathematical functions. We are reading the three sides of a
triangle using
'a', 'b', 'c' integer variables. To find the area of a triangle,
the following formula is used.
Area = sqrt (s * (s - a) * (s - b) * (s - c))
*/
```

```
/*
```

C program to find the area of a circle using the given radius.

```
Problem Solution
Step 1: Start the program
Step 2: Input the value of radius (float value).
Step 3: Calculate area using formula pi*(r2)
Step 4: Print area of circle
Step 5: End the Program

Area of circle is defined as pi*r*r where pi is a constant whose value
is (22/7 or 3.142) and r is the radius of a circle.

Formula to calculate the area of circle is: Area = pi*r*r

Example:
If the radius of a circle is 20, then its area would be
3.142*15*15 = 706.95 approx.
 */
#include <stdio.h>
#include <math.h>
#define PI 3.142

void main()
{
    float radius, area;

    printf("Enter the radius of a circle \n");
    scanf("%f", &radius);
    area = PI * pow(radius, 2);
    printf("Area of a circle = %5.2f\n", area);
}
```

Output:

```
Enter the radius of a circle
5
Area of a circle = 78.55
```

```
/*
Program Explanation
In this C program, library function defined in <math.h> header file is
used
to compute mathematical functions. We are reading the radius of a circle
using
'radius' variable. To find the area of a circle, the following formula is
used.
Area = PI * pow (radius, 2)

Time Complexity: O(1)
In the above program there are no iterative statements, so time complexity
is O(1).

Space Complexity: O(1)
Since no auxiliary space is required, space complexity is O(1).
*/
```

/*

C program to read two strings and concatenate them, without using library functions. Display the concatenated string.

Problem Description
This program takes two strings as input and concatenate them.

Problem Solution
1. Take two strings as input and store them in two different arrays.
2. Find the position of the last element of the first array and from that position keep on adding the elements of the second array.

```
3. Exit.

*/

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

void main()
{
    char string1[20], string2[20];
    int i, j, pos;

    /*  Initialize the string to NULL values */
    memset(string1, 0, 20);
    memset(string2, 0, 20);

    printf("Enter the first string : ");
    scanf("%s", string1);
    printf("Enter the second string: ");
    scanf("%s", string2);
    printf("First string  = %s\n", string1);
    printf("Second string = %s\n", string2);

    /*  Concate the second string to the end of the first string */
    for (i = 0; string1[i] != '\0'; i++)
    {
        /*  null statement: simply traversing the string1 */
        ;
    }
    pos = i;
    for (j = 0; string2[j] != '\0'; i++)
    {
        string1[i] = string2[j++];
    }
    /*  set the last character of string1 to NULL */
    string1[i] = '\0';
    printf("Concatenated string = %s\n", string1);
}
```

Output:

```
Enter the first string : Procedural
Enter the second string: Language
First string  = Procedural
Second string = Language
Concatenated string = ProceduralLanguage

/*
Program Explanation
1. Take two strings as input and store them in the arrays string1 and
string2 respectively.
2. Using for loop find the position of the last element of the array
string1[]. Store that position in the variable pos.
3. Using another for loop add the elements of the array string2[] into the
array string1[] starting from the obtained position.
4. Print the array string1[] as output.
*/
```

```
/*
```

C program to check whether a given integer is odd or even.

```
Problem Description
The program takes the given integer and checks whether the integer is odd
or even.

Problem Solution
1. Take the integer to be checked as input.
2. Find the remainder of the integer by dividing it by 2.
3. Use if,else statement to check whether the remainder is equal to zero
or not.
4. Print the output and exit.
*/


#include <stdio.h>

void main()
{
    int ival, remainder;

    printf("Enter an integer : ");
    scanf("%d", &ival);
    remainder = ival % 2;
    if (remainder == 0)
        printf("%d is an even integer\n", ival);
    else
        printf("%d is an odd integer\n", ival);
}
```

Output:

```
Enter an integer : 7
7 is an odd integer

Enter an integer : 6
6 is an even integer

/*
Program Explanation
1. User must first enter the integer to be checked which is stored in the
variable ival.
2. Find the remainder of the integer by dividing the variable ival by
integer 2 and the value is stored in the variable remainder.
3. Use if,else statement to check whether the value of the variable
remainder is equal to zero or not.
4. If it is equal to zero, then print the output as "the integer is an
even integer".
5. If it is not equal to zero, then print the output as "the integer is an
odd integer".
*/
```

//**Reverse a number**

```
#include<stdio.h>
int main()
{
    //Initialization of variables where rev='reverse=0'
    int number, rev = 0,store, left;

    //input a numbers for user
      printf("Enter the number\n");
      scanf("%d", &number);
     store= number;

     //use this loop for check true condition
     while (number > 0)
     {
       //left is for remider are left
        left= number%10;

       //for reverse of no.
        rev = rev * 10 + left;

        //number /= 10;
        number=number/10;

      }
     //To show the user value
      printf("Given number = %d\n",store);

      //after reverse show numbers
       printf("Its reverse is = %d\n", rev);

     return 0;
}
```

Output:

```
Enter the number
123
Given number = 123
Its reverse is = 321
```