Date-

Assignment No. :

Problem Statement:

Program in C to convert a number to its English equivalent alphabetical representation.

Theory:

The alphabetical representation of a decimal number is based on repeatation of some primary tokens, such as “one”, “ten”, “hundred” etc. In the alphabetical representation of a decimal number, these tokens can represent either a value, or the positional weight of the value preceding it. Hence, the algorithm of this conversion is based on extracting digits of a decimal number in order, printing the value of the digits, followed by printing their positional value, if any. Positional values include “hundred”, “thousand”, “lakh” and “crore”. Numeric values include “one”, “two”, …, “ten”. The numbers 11-19 deserves special attention, as they are pronounced differently than say, 21 or 31. Hence, the alphabetical representation of the values from 11 to 19 are also considered as primary tokens, and is printed as required. Also, the values 10, 20, 30, …, 90 are also considered as primary tokens, for the same reason.

Example :

12890 → Twelve Thousand Eight Hundred Ninety

Algorithm:

Input Specification: A number in its digit form say a.

Output Specification: Suitable word of the digit number.

Steps:

Algorithm for method TwoDig(value):

Step 1: Set rem = value / 10
Step 2: If(rem > 0) Then
    a. If(rem == 1) Then
        i. Print e[rem – 10]
            // e is an array which contains the alphabetical representation
            // of the numbers 10-19

    b. Else
        i. Print c[rem – 2]
            // c is an array which contains alphabetical representations
            // of the numbers 20, 30, 40, …, 90
    c. [End of if structre]

    d. If(value mod 10 == 0) Then
        i. Return
    e. [End of if structre]

    f. Print " "
    g. [End of if structre]

Step 3: Print f[value mod 10]
            // f is an array which contains alphabetical representation
            // of the numbers 0 – 9

Algorithm for method Num_To_Word(num):

Step 1: If(num > 9999999) Then
- i. Call Num_To_Word(num / 10000000)
- ii. Print " crore "
- iii. Set num = num mod 10000000
- iv. [End of if structre]

Step 2: Set t = num, count = 0
Step 3: Repeat step 3.A while (t > 0)
- i. Set t = t / 10, count = count + 1

[End of while loop]

Step 4: Set t = num, hp = 0
Step 5: If(count < 3) Then
- i. Call TwoDig(t)
- ii. Exit

Step 6: Else if(count == 3 Or count mod 2 == 0) Then
- i. Set hp = PowerOf(10, count – 1)

// PowerOf is a function which calculates the exponent
// of a given base

Step 7: Else
- i. Set hp = PowerOf(10, count – 2)

[End of if structure]

Step 8: Set pos = count, t = num
Step 9: Repeat through step 9.A to 9.H while (t > 0)
- i. Set dig = t / hp
- ii. Call TwoDig(dig)
- iii. If(pos > 2) Then
  1. Print d[pos / 2 – 1]

// d is an array which contains "hundred", "thousand", "lakh"
// and "crore"

[End of if structre]

iv. Set t = t mod hp
v. If(pos == 4 Or pos == 5) Then
    1. Set hp = hp / 10
vi. Else
    1. Set hp = hp / 100
[End of if structre]

vii. If(pos mod 2 == 0) Then
    1. Set pos = pos – 1
viii. Else
    1. Set pos = pos – 2
[End of if structre]

[End of while loop]

Algorithm for method main():

Step 1: Print "Enter the number : "
Step 2: Input a
Step 3: If(a < 0) Then
    a. Print "minus "
    b. a *= -1
    [End of If structure]
Step 4: num_to_word(a)

Source Code:

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

const char *f[] = {"zero", "one", "two", "three", "four", "five", "six", "seven",
"eight", "nine"};
const char *c[] = {"twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty",
"ninety"};
const char *e[] = {"ten", "eleven", "twelve", "thirteen", "forteen", "fifteen",
"sixteen", "seventeen",
   "eighteen", "nineteen"};
```

```
const char *d[] = {"hundred", "thousand", "lakh", "crore"};

static void twodig(long int val){
   long int rem = val / 10;
   if(rem > 0){
      if(rem == 1){
         printf("%s", e[val - 10]);
         return;
      }
      else
         printf("%s", c[rem - 2]);
      if(val % 10 == 0)
         return;
      printf(" ");
   }

   printf("%s", f[val % 10]);
}

static void num_to_word(long int num){
   if(num > 9999999){
      num_to_word(num / 10000000);
      printf(" crore ");
      num = num % 10000000;
   }

   long int t = num;
   int count = 0;

   while(t > 0){
      t /= 10;
      count++;
   }
```

```
    t = num;
    long int hp = 0;
    if(count < 3){
        twodig(t);
        return;
    }
    else if(count == 3 || count % 2 == 0)
        hp = pow(10, count - 1);
    else
        hp = pow(10, count - 2);
    long int pos = count;
    t = num;

    long int dig;
    while(t > 0){
        dig = t / hp;
        twodig(dig);
        if(pos > 2){
            printf(" %s ", d[pos / 2 - 1]);
        }

        t = t % hp;
        hp /= (pos == 4 || pos == 5) ? 10 : 100;
        pos -= (pos % 2 == 0) ? 1 : 2;
    }
}

int main(){
    long int a;
    printf("\nEnter the number : ");
    scanf("%ld", &a);
    if(a < 0){
```

```
        printf("minus ");
        a *= -1;
    }
    num_to_word(a);
    printf("\n");
    return 0;
}
```

Input & Output:

Set 1:

```
Enter the number : 12340
twelve thousand three hundred forty
```

Set 2:

```
Enter the number : 45678
forty five thousand six hundred seventy eight
```

Discussion:

1. Given the inputs of the user, the number can be overflowed, and hence erroneous output can be produced by the program.
2. This program does not support printing values of floating point numbers.
3. The efficiency of this program can be improved.