Analysis, Implementation, Comparison and Evaluation of different Logical Techniques of Solving the Same Computing Problem

Vimal P. Parmar, PhD
Research Scholar,
Dept. of Comp. Sci.
Saurashtra University, Rajkot. Gujarat, INDIA

C. K. Kumbharana, PhD
Head, Guide
Department of Computer Science
Saurashtra University Rajkot.
Gujarat, INDIA

ABSTRACT
Computers are considered as logic driven tools. We can have different solutions for the same problem. How a particular problem can be solved depends on the individual thinking and logic. There may be multiple routes to reach from one city to another, what the route is selected depends on the best route in terms of distance, quality of routes and safety. Same way parallel it can be compared with the approach for solving computing problems. To perform task through a computer we must require writing a program in specific language and then converting it into its equivalent machine language program either by an interpreter or a compiler. Also, there exist more than one such type of solutions for the same problem. Each may have its own advantages and disadvantages in terms of memory requirement, time required for executing by a computer and readability in form of understanding the logic of solution. This research paper is discussing different basic problems to be solved by a computer in different ways and which is better compared to others using above characteristics. Same type of analysis can be applied to any complex problem by determining all possible solutions and comparisons. The examples used are so basic to better understand the power of programming language and the solution itself.

General Terms
Programming, program, solution, logic, programming language

Keywords
Analysis, comparison, evaluation, memory requirement, execution time

1. INTRODUCTION
To solve any problem, it requires some logic applied in step by step logical solution of a problem. A solution to a problem can be obtained by writing a program in any programming language. The popular programming language which is chosen here is C programming language. C is robust and powerful programming language with varieties of constructs available. All the problems discussed here are implemented in C programming language. Four problems are discussed here and solution of each gives the correct answer. Each problem is analyzed and implemented in various ways for comparison. Each solution has its own merits and demerits. Four problems discussed here are odd and even number problem, prime number problem, the problem of determining sign of two given numbers and a series problem. Each problem has multiple alternative logical solutions and each solution is characterized based on memory, time and readability.

2. SOLUTION OF A PROBLEM THROUGH COMPUTERS USING PROGRAM
Computers require a program to solve any problem. We use language for communication, same way computers also require a language. But computer can only understand machine language which is in binary 0 and 1 and also known as low level language. It is difficult for human being to understand this language, so some higher level languages came to exist. C is such a popular language widely used for solving general purpose problems. Compiler is a program that converts a higher level program into machine language program with some diagnostic and error handling facility. Generally each program may have some inputs, some kind of processing to obtain the solution and output as an answer. In this paper we start with the simplest solution and then by improving it by reducing the steps or by taking advantages of C language. Each solution yields the correct output but the steps required to reach to the solution are different. An efficient solution is one which requires minimum resources and provides fast execution. This is necessary to design any program solution for efficient use of computing power. For problems discussed here are implemented in C programming language and each problem has different solutions. These solutions are compared and evaluated and then concluded which is more effective. At last final choice of solution depends on programmer's choice and not strictly necessary to follow the said solution. Further other solutions may also be possible not described in this paper.

3. ODD EVEN NUMBER PROBLEM
The simplest problem is to determine whether the given number is odd or even. To solve this problem a number must be entered through keyboard. Then steps in a program will determine whether the given number is odd or even. A number is said to be odd if dividing a number by 2 and remainder of the division yield one. If the remainder of a division is 0 then the number is said to be as an even number. The simplest solution of this problem is listed as a C program in following table 3.1. A number is accepted using scanf() function and we obtain a remainder of dividing a number by 2 using a formula number – quotient * 2. Odd or even number is determined by comparing remainder with 1 or zero. Integer division always results in an integer number without a fractional part.

```
/* Program to determine given number is odd or even*/
#include<stdio.h>
void main()
{
        int number, quotient, remainder;
        printf("Input number : ");
        scanf("%d",&number);
        quotient = number / 2;
        remainder = number - quotient * 2;
        if (remainder == 0)
                printf("%d is even\n", number);
        else
                printf("%d is odd\n", number);
}
```

Table 3.1 : Odd even number solution – 1 basic rules of division, quotient and remainder

Here program calculates the remainder by writing an expression. The same solution can be obtained by using modulus operator "%" which yields the remainder of an integer division. The listing of the code is depicted in following table 3.2.

```
/* Program to determine given number is odd or even*/
#include<stdio.h>
void main()
{
        int number;
        printf("Input number : ");
        scanf("%d",&number);
        if (number % 2 == 0)
                printf("%d is even\n", number);
        else
                printf("%d is odd\n", number);
}
```

Table 3.2 : Odd even number solution – 2 using modulus operator %

Third solution is more compact and instead of using if else statement, ternary operator is used to determine the value of remainder. The third solution is listed in following table 3.4.

```
/* Program to determine given number is odd or even*/
#include<stdio.h>
void main()
{
    int number;
    printf("Input number : ");
    scanf("%d",&number);
    (number % 2 == 0)? printf("%d is even\n", number)
                     : printf("%d is odd\n", number);
}
```

Table 3.3 : Odd even number solution – 3 using ternary operator ?:

The forth solution uses an array of string to determine the odd or even number. The solution is as given in following table 3.4. The subscript result number%2 always results in zero or one and that zero and one are subscripts for string elements "even" and "odd". This is again more compact by removing of test condition.

```
/* Program to determine given number is odd or even
        */
#include<stdio.h>
void main()
{
        int number;
        char oe[2][5] = {"even","odd"};
        printf("Input number : ");
        scanf("%d",&number);
        printf("%d is %s\n", number, oe[number%2]);
}
```

Table 3.4 : Odd even number solution – 4 using string

Next solution to the same problem is by use of bit wise operator. As in integer number the right most bit is 0 then the number is always even and if it is 1 then the number is odd. The listing of code is given in table 3.5.

```
/* Program to determine given number is odd or even*/
#include<stdio.h>
void main()
{
        int number;
        printf("Input number : ");
        scanf("%d",&number);
        (number & 1) ? printf("%d is odd\n", number)
        : printf("%d is even\n", number);
}
```

Table 3.5 : Odd even number solution – 5 using bitwise AND operator &

Next solution using switch statement is listed below in table 3.6.

```
/* Program to determine given number is odd or even
        */
#include<stdio.h>
void main()
{
        int number;
        printf("Input number : ");
        scanf("%d",&number);
        switch(number%2)
        {
            case 0 :
                    printf("%d is even\n", number);
                    break;
            default :
                    printf("%d is odd\n", number);
                    break;
        }
}
```

Table 3.6 : Odd even number solution – 6 using switch case statement

Another solution using bit wise shift operator is listed below in table 3.7.

```
/* Program to determine given number is odd or even*/
#include<stdio.h>
void main()
{
	int number;
	printf("Input number : ");
	scanf("%d",&number);
	if(number << 15)
		printf("%d is odd\n", number);
	else
		printf("%d is even\n", number);
}
```

Table 3.7 : Odd even number solution – 7 using bitwise shift left operator

Each of the above solution finds odd or even number correctly. It depends on the programmer's choice to select the desired one. The solutions using bitwise operators however might be faster as it works on bit structure of the integer number and the only solution with using string in which no condition is tested at all to determine the result is more interesting. Which solution is to select depends on individual but this are alternative solutions for one of the simplest problem. So it may be possible to implement similar kind of logic in more complex problems and selecting one that suits best. The first solution requires more memory variables and follows exact mathematical calculations. Furthermore other solutions may also exist which are not listed here.

4. PRIME NUMBER PROBLEM

Prime number is a number which is divisible by 1or by number itself. 1 is a special number. The solution of determining a prime number is implemented by dividing a given number from 2 and onwards. If any division yields result zero then the number is not prime number. If we reach at number to itself following the division process then the number is a prime number. The same technique of finding a remainder of a division used in solution table 3.1 can be employed but we can use modulus operator to reduce the steps. The first solution is listed in following table 4.1.

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
void main()
{
	int num,i, flag;
	printf("Input number : ");
	scanf("%d",&num);
	flag = 1;
	for (i = 2; i < num; i++)
	{
		if (num % i == 0)
		{
			flag = 0;
			break;
		}
	}
	if (flag)
		printf("%d is prime\n", num);
	else
		printf("%d is not prime\n", num);
}
```

Table 4.1 : Prime number solution – 1 dividing from 2 to given number-1

In this program of prime number, testing for the remainder of a division of given number begins from 2 to num – 1. If any number in between these two numbers including divide perfectly, then the remainder will become zero and flag is set to 0 and loop is terminated. This process of testing continues till loop counter variable i becomes num. In this case flag remains unchanged as 1 and the number is a prime number. It is wasting of computing time that if number is not divisible up to num/2, it will never be divisible by number from num/2 onwards. So the next solution reduces the steps of loop from 2 up to num/2. The second solution with minor changes is listed in following solution 4.2. The performance of this solution is effective in case of a large given prime number in which half of the loop statements are reduced and program can run faster twice as compared to the first solution.

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
void main()
{
	int num,i, flag;
	printf("Input number : ");
	scanf("%d",&num);
	flag = 1;
	for (i = 2; i <= num/2; i++)
	{
		if (num % i == 0)
		{
			flag = 0;
			break;
		}
	}
	if (flag)
		printf("%d is prime\n", num);
	else
		printf("%d is not prime\n", num);
}
```

Table 4.2 : Prime number solution – 2 dividing from 2 to given number/2

Program behaves same as the previous program but the number of iterations are reduces to half of the given number. The solution is better compared to first one but still it can be improved. For any number it is necessary to only test from number 2 to square root of the number, so again number of iterations is reduced. Any number can not be divisible if it can not be divisible from 2 to its square root. To find out square root we require to include math.h header file. The code is listed in following table 4.3.

Further improvement is possible as if a number is not divisible by 2 then it also can not be divisible by any even number 4, 6, 8 and onwards. So it is necessary to first test with number 2and then start testing for all odd numbers starting from 3up to the square root of the number. The program answers the same output but each solution improves as execution point of view and solutions are moving from more readable easy to understand solution to a complex one.

Which solution is selected is again depends on nature of programmer and the facilities available in programming language. But when solving some other complex problem, the solutions discussed here are helpful for obtaining the efficient solution for the complex problem. Two terms are widely used for algorithm analysis are time complexity and space complexity. Time complexity refers to amount of asymptotic time require when the input number is large enough. Space

complexity is related with amount of memory required during program execution.

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
#include<math.h>
void main()
{
        int num,i, k, flag;
        printf("Input number : ");
        scanf("%d",&num);
        flag = 1;  k = sqrt(num);
        for (i = 2; i <= k;i++)
        {
                if (num % i == 0)
                {
                        flag = 0;
                        break;
                }
        }
        if (flag)
                printf("%d is prime\n", num);
        else
                printf("%d is not prime\n", num);
}
```

Table 4.3 : Prime number solution – 3 dividing from 2 to given square root of number

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
#include<math.h>
void main()
{
        int num,i, k, flag;
        printf("Input number : ");
        scanf("%d",&num);
        flag = 1;
        k = sqrt(num);
        if (num % 2 == 0)
                flag = 0;
        else
        {
                for (i = 3; i <= k;i+=2)
                {       if (num % i == 0)
                        {
                                flag = 0;
                                break;
                        }
                }
        }
        if (flag)    printf("%d is prime\n", num);
        else          printf("%d is not prime\n", num);
}
```

Table 4.4 : Prime number solution – 4 dividing from 3 to given square root of number using odd numbers only

5. PROBLEM OF DETERMINING SIGN OF TWO GIVEN NUMBERS

A problem which accepts two numbers and determine whether either of the number is zero or not, if no number is zero then determining whether both the number have a similar sign or different that means positive or negative. The basic solution is applied by testing each condition described here. The program listing is as below in table 5.1.

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
#include<math.h>
void main()
{
        int num1,num2;
        printf("Input number 1 : ");
        scanf("%d",&num1);
        printf("Input number 2 : ");
        scanf("%d",&num2);
        if (num1 == 0 || num2 == 0)
              printf("Either of the numbers is zero.\n");
        else
        {
           if ( (num1 > 0 && num2 > 0) ||
             (num1 < 0   && num2 < 0))
                printf("Both numbers have same sign\n");
           else
             printf("Both number have different sign\n");
        }
}
```

Table 5.1 : Solution – 1 Determining sign of the two numbers if either of the number is nonzero

First two numbers are tested for either number is zero or not connected with logical OR operator. If none of the number is zero then two conditions connected with OR operator is tested and each condition compares both numbers are positive or negative using logical AND operator. If any one condition from OR is evaluated to true then the numbers have similar sign either negative or positive. If both conditions are false then both the numbers have different signs in this case if first number is negative then the second number is positive or if first number is positive then second number is negative.

The solution is simple one but contains many test expression and many logical operators. It is easy somehow to understand but involve more processing by a computer to solve a simple problem. To overcome this problem an alternate solution should be obtained that must reduce the test conditions and easy to understand with compact code. Here we use the help of mathematics to solve the same problem and program written using this logic is far efficient than to described in table 5.1. A revised complete program is listed in table 5.2 which is better in point of view for user as well as for computer.

```
/* Program to determine given number is prime or not*/
#include<stdio.h>
#include<math.h>
void main()
{
        int num1,num2, sign;
        printf("Input number 1 : ");
        scanf("%d",&num1);
        printf("Input number 2 : ");
        scanf("%d",&num2);
        sign = num1 * num2;
        if (sign == 0)
              printf("Either of the numbers is zero.\n");
        else
        {
            if (sign > 0)
                printf("Both numbers have same sign\n");
           else
             printf("Both number have different sign\n");
        }
}
```

Table 5.1 : Solution – 2 Determining sign of the two numbers if either of the number is nonzero

In this solution a variable sign is used and assigned a value of multiplication of given two numbers. If either or both of numbers are zero then sign will be zero. It then tests by comparing sign with zero. If condition evaluates to false then none of the number is zero. Now to test for the sign of both numbers the variable sign is tested with zero again but now using greater relation operator. Variable sign is positive if both numbers are negative or both numbers are positive. If this condition is false then sign contain negative that means either of the number is negative means both the numbers have different sign. The program listed in table 5.2 is more compact easy to understand and require less computing processing and that's why is more efficient compared to program listed in table 5.2.

6. CALUCLATING THE SUM OF SERIES A + (A+D) + (A + 2D) … … N TERMS

The program to calculate the sum of the arithmetic progression series A + (A+D) + (A+2D) … … N Terms requires input of A, D and total number of terms N. Here we require a loop construct to add each term. The result can be stored in variable SUM. Two solutions are discussed here. First solution computes each term in each solution and then added to variable SUM. This involves multiplication from 0 to N with D and added with A to calculate current term. This process is required more multiplications. Multiplication is the compact process of addition. So by eliminating multiplication and reusing the computation calculated in previous term is more efficient. Also naturally addition is faster compared to multiplication.

First solution is listed in following table 6.1.

```
/* Sum of A + (A+D) + (A+2D) ... ... N Terms */
#include<stdio.h>
void main()
{
    int A, D, N, SUM = 0, i, TERM;
    printf("A = ");
    scanf("%d",&A);
    printf("D = ");
    scanf("%d", &D);
    printf("N = ");
    scanf("%d", &N);
    for (i = 0; i < N; i++)
    {
            TERM = A + i * D;
            SUM = SUM + TERM;
    }
    printf("SUM= %d\n", SUM);
}
```

Table 6.1 : Sum of series Solution – 1

In above solution each time term is calculated by adding A and D term by multiplying 0, 1, 2 …onwards. The process repetitively calculates each term and involves multiplication. An efficient solution by removing multiplication is depicted in table 6.2.

```
/* Sum of A + (A+D) + (A+2D) ... ... N Terms */
#include<stdio.h>
void main()
{
    int A, D, N, SUM = 0, i, TERM;
    printf("A = ");
    scanf("%d",&A);
    printf("D = ");
    scanf("%d", &D);
    printf("N = ");
    scanf("%d", &N);
    TERM = A;
    for (i = 0; i < N; i++)
    {
            SUM = SUM + TERM;
            TERM = TERM + D;
    }
    printf("SUM= %d\n", SUM);
}
```

Table 6.2 : Sum of series Solution – 2

Second solution is more efficient and each time previously calculated term is used to prepare the next term for the next iteration by addition. First time TERM is initialized with A and each time adding D with TERM results in A+D, A+2D and so on. So the second solution is better for computing point of view as well as solving other series problems by applying the similar type of logic.

7. CONCLUSION

Any computing problem can be solved using different logic and different kind of processing. To determine which solution is better, analysis of algorithm is required. Algorithm analysis calculates time complexity in form of big O notation and by comparing this time it is concluded which is better solution. Here for basic different types of problems with multiple solutions are discussed each having its own characteristics. To select the desired one depends on programmer and the programming language in which it is implemented but the logic point of view a solution is said to be efficient which maximum utilize the computing power. Answer can always be obtained correct if program is written correctly but what the processing is involved has greater impact. It may be possible the similar type of logic used in one problem can be extended for other types of similar problems. It is good in programming practice to analyze the solution, determining other possible solutions and implementing the best one by comparing each solution with different characteristics.

8. REFERENCES

[1] Dennis M Ritchie, Brian W. Kernighan "The C Programming Language" Second Edition

[2] Pradip dey, Manas Ghosh " Programming in C" Oxford publication

[3] E. Balagurusamy "Programming in ANSI C"

[4] Yashavant Kanetkar "Let us C"