FUNDAMENTALS OF C PROGRAMMING
Question Bank
SECTION-A (1 mark)

Q1. Identify which shape is commonly used to represent a condition in a flowchart?

1. Rectangle
2. Diamond
3. Oval
4. Parallelogram

Ans: Diamond

Q2. Identify which one of the following is an invalid variable name?

1. number
2. FLOAT
3. totalarea
4. variable name

Ans: variable name

Q3. Predict the output of following code?

```
{
int i=2, j=3, k,p;
float a, b;
k=i/j*j;
p=j/i*i;
a=i/j*j;
b=j/i*i;
printf("%d %d %f %f",k,p,a,b);
}
```

1. 0 2 0.000000 2.000000
2. 0 2 0.00 2.00
3. 1.88 3 1.888888 3.000000
4. 1.00 3.00 0.000000 2.000000

Ans: 0 2 0.000000 2.000000

Q4. Arrange the stages of the problem-solving process in the correct order:

A. Identifying the problem
B. Generating potential solutions
C. Implementing the chosen solution
D. Evaluating the outcomes
E. Analyzing the available information

Choose the correct answer from the options given:

1. A, B, C, E, D
2. E, A ,B, C, D
3. A, E, B, C, D
4. A, E, C ,B, D

Ans: A, E, B, C, D

Q5. Find the Correct precedence of arithmetic operators (from highest to lowest).

1. %, *, /, +, -
2. %, +, /, *, -
3. %, +, -, *, /
4. +, -, %, *, /

Ans: %, *, /, +, -

Q6. Predict the output of following code.

```
void main(){
int a, b = 5, c;
a = 5 * (b++);
c = 5 * (++b);
printf("%d %d",a,c);
}
```

1. 30 35
2. 30 30
3. 25 30
4. 25 35

Ans: 25 35

Q7. Correct answer of given code is:

```
int main(){
float x = 'a';
printf("%f", x);
return 0;
}
```

1. a
2. 0
3. 97
4. Run time error

Ans: 97

Q8. An algorithm should have _____ well-defined inputs.

1. 0
2. 1
3. 0 or more
4. 1 or more

Ans: 0 or more

Q9. In C language the size of char (64 bit architecture) is ?

1. 1
2. 4
3. 8
4. 16

Ans: 1

Q10. Identify the correct way of making a single line comment in C

1. /* */
2. //
3. / /
4. //* *//

Ans: //

Q11. Which keyword is used to define a conditional statement in C?

1. Switch
2. if
3. while
4. for

Ans: if

Q12. Which is the syntax of an if-else statement in C?

1. if { } else { }
2. if condition else
3. if (condition) { }
4. if (condition) { } else { }

Ans: if (condition) { } else { }

Q13. Which one is the correct syntax for an if-else ladder in C?

1. if (condition) { } else if (condition) { }
2. if (condition) { } else { } elseif (condition) { }
3. if (condition) { } elseif (condition) { } else { }
4. if (condition) { } else if (condition) { } else { }

Ans: if (condition) { } else if (condition) { } else { }

Q14. What will be the value of y if x = 8? y = (x > 6 ? 4 : 6);

1. Compilation Error
2. 0
3. 4
4. 6

Ans: 0

Q15. What is a switch case statement used for in C?

1. To define constants
2. To perform different actions based on different conditions
3. To create loops
4. To declare variables

Ans: To perform different actions based on different conditions

Q16. The first expression in a for… loop is

1. Step value of loop
2. Value of the counter variable
3. Condition statement
4. None of the above

Ans: Value of the counter variable

Q17. If switch case is used, then

1. Default case must be present
2. Default case, if used, should not be the last case
3. Default case, if used, should be the last case
4. None of these

Ans: Default case, if used, should be the last case

Q18. Choose a syntax for C Ternary Operator from the list.

1. condition ? expression1 : expression2
2. condition : expression1 ? expression2
3. condition ? expression1 < expression2
4. condition < expression1 ? expression2

Ans: condition ? expression1 : expression2

Q19. Which for loop has range of similar indexes of 'i' used in for (i = 0;i < n; i++)?

1. for (i = n; i>0; i--)
2. for (i = n; i >= 0; i--)

3. for (i = n-1; i>0; i--)
4. for (i = n-1; i>-1; i--)

Ans: for (i = n-1; i>-1; i--)

Q20. do-while loop terminates when conditional expression returns?

1. One
2. Zero
3. Non-Zero
4. None of the above

Ans: Zero

Q21. When a variable is declared as 'static' at the global level in C, where does it get stored?

1. in the stack
2. in the heap
3. in the data segment
4. in the code segment

Ans: in the data segment

Q22. Which storage class in C is used to declare variables that remain in memory for the entire program execution?

1. auto
2. extern
3. static
4. register

Ans: static

Q23. Predict the output:

```
#include<stdio.h>
int main()
{
  typedef static int *i;
  int j;
  i a = &j;
  printf("%d", *a);
  return 0;
}
```

1. Runtime Error
2. 0
3. Garbage Value
4. Compiler Error

Ans: Compiler Error

Q24. In C, how are functions arguments passed by default?

1. by value
2. by reference
3. by pointer
4. by address

Ans: by value

Q25. What is function prototyping in C?

1. Writing the body of a function
2. Creating a reference to a function
3. Declaring a function before its definition
4. Calling a function before it is declared

Ans: Declaring a function before its definition

Q26. What is the return type of the function with declaration:
int fun(char x, float v, double t);
1. char
2. int
3. float
4. double

Ans: int

Q27. Which of the following is an example of function overloading in C?
1. int func(); float func();
2. int func(int a); int func(float a);
3. int func(int a); int func(int a, int b);
4. Function overloading is not possible in C

Ans: Function overloading is not possible in C

Q28. What is the meaning of using extern before function declaration? For example following function sum is made extern:

```
extern int sum(int x, int y, int z)
{
   return (x + y + z);
}
```

1. Function is made globally available
2. extern means nothing, sum() is same without extern keyword.
3. Function need not to be declared before its use
4. Function is made local to the file.

Ans: extern means nothing, sum() is same without extern keyword.

Q29. Predict output of following program: #include <stdio.h>

```
int fun(int n)
{
   if (n == 4)
      return n;
   else return 2*fun(n+1);
}

int main()
{
  printf("%d", fun(2));
  return 0;
}
```

1. 4
2. 8
3. 16
4. Runtime Error

Ans: 16

Q30. Consider the following recursive function fun(x, y). What is the value of fun(4, 3) ?

```
int fun(int x, int y)
{
 if (x == 0)
   return y;
```

```
  return fun(x - 1,  x + y);
}
```
1. 13
2. 12
3. 9
4. 10

Ans: 13

Q31. Identify the right way to initialize an array?

1. int num[6] = {2, 4, 12, 5, 45, 5 };
2. int n{}={2,4,3,5,7,8};
3. int n{6}={1,2,4,5,6,7};
4. int n(6)={2,3,5,6,7,8};

Ans: int num[6] = {2, 4, 12, 5, 45, 5 };

Q32. Analyze the output of the following program?

```
#include<stdio.h>
int main()
{
static int a = 3;
printf("%d", a --);
return 0;
}
```
1. 0
2. 1
3. 2
4. 3

Ans: 3

Q33. Predict the output:

```
#include<stdio.h>
void main()
{
char *s= "MATHS";
char *p = s;
printf("%c\t%c", *(p+3), s[1]);
}
```
1. M S
2. H S
3. A T
4. H A

Ans: H A

Q34. Predict the output:

```
#include<stdio.h>
int main()
{
char ch;
ch = 97;
printf("%c", ch);
return 0;
}
```
1. a

2. A
3. Depends on compiler
4. 97

Ans: a

Q35. Analyze the output of following code:

```
#include<stdio.h>
void main()
{
int a[5] = {5, 1, 15, 20, 25};
int i, j, m;
i = ++a[1];
j = a[1]++;
m = a[i++];
printf("%d, %d, %d", i, j, m);
}
```

1. 3,2,15
2. 1,3,15
3. 2,3,14
4. 2,4,15

Ans: 3,2,15

Q36. Predict the output of following code:

```
#include<stdio.h>
int main()
{
char *ptr = "Pointer in c", arr[15];
arr[15] = *ptr;
printf("%c",arr[0]);
return 0;
}
```

1. Garbage Value
2. Run Time Error
3. Compile Time Error
4. 15

Ans: Garbage Value

Q37. Comment on this const int *ptr;

1. You cannot change the value pointed by ptr
2. You cannot change the pointer ptr itself
3. You can change the pointer as well as the value pointed by it
4. ALL

Ans: You cannot change the value pointed by ptr

Q38. In C language,a pointer variable to an integer can be created by which of the following declarations:

1. int +p;
2. int *p;
3. int p*;
4. int p**;

Ans: int *p;

Q39. Identify the correct statement about C structure elements?

1. Structure elements are stored on random free memory locations

2. structure elements are stored in contiguous memory locations
3. structure elements are stored in register memory locations
4. None of these

Ans: structure elements are stored in contiguous memory locations

Q40. The following cannot be a structure member?

1. Another structure

2. Function
3. Array
4. None of the mentioned

Ans: Function

SECTION-B (2 Mark)

Q1. What does the following code print?

```
int main() {
   int a = 5, b = 5;
   int c = a++ + ++b;
   printf("%d %d %d", a, b, c);
}
```

1. 5 5 10
2. 6 5 11
3. 5 6 10
4. 6 6 11

Ans: 6 6 11

Q2. Given the following code, what is the final value of a and b?

```
   int main() {
   int a = 5, b = 5;
   a = b++ + b++;
   printf("%d %d", a, b);
}
```

1. 11 7
2. 10 6
3. 12 6
4. 10 7

Ans: 10 7

Q3. What does the following C code fragment output?

```
   int main() {
   int x = 9;
   x = x++ + ++x;
   printf("%d", x);
}
```

1. 19
2. 20
3. 21
4. Undefined behavior

Ans: 21

Q4. What will be the output?

```
    int main() {
   int x = 1, y = 2;
   x = y = x + 3;
   printf("%d %d", x, y);
}
```

1. 4 1
2. 1 4
3. 2 4
4. 4 4

Ans: 2 4

Q5. What does this code output?

```
   int main() {
   int num = 40000;
```

```
    printf("%hd", num);
}
```
1. 40000
2. Undefined behavior
3. A compiler error
4. A negative number

Ans: A negative number

Q6. What is the output of the following code?

```
 int main() {
    int i = 10;
    int j = 20;
    int k = (i *= 2, j = i + 10, i + j);
    printf("%d", k);
}
```
1. 30
2. 40
3. 50
4. 60

Ans: 60

Q7. What will be the output of the following code snippet?

```
    int main() {
    int a = 1, b = 2;
    int result = (a++ && (!b));
    printf("%d", result);
}
```
1. 1
2. 2
3. Undefined
4. 0

Ans: 0

Q8. What will the following C code output?

```
  int main() {
    char ch = 'A';
    putchar(65);
    putchar('\n');
    putchar(ch);
}
```
1. A A
2. 65 65
3. Error
4. A\nA

Ans: A\nA

Q9. What is the output of the following code involving bitwise operators?

```
  int main() {
    unsigned int a = 60; // 0011 1100 in binary
    unsigned int b = 13; // 0000 1101 in binary
    int result = a ^ b; // XOR operation
    printf("%d", result);
}
```
1. 60

2. 56
3. 52
4. 49

Ans: 49

Q10. The correct syntax to access the member of the ith structure in the array of structures is?

```
struct temp {
  int b;
}s[50];
```

1. s.b.[i];

2. s.[i].b;
3. s.b[i];
4. s[i].b;

Ans: s[i].b;

Q12. Analyze the behaviour of the following code:

```
#include <stdio.h>
int main() {
    int x = 5;
    if (x > 3)
        printf("A");
    if (x < 3)
        printf("B");
    else
        printf("C");
    return 0;
}
```

a) A
b) AC
c) AB
d) Compilation error

Q13. Determine the behavior of the following code:

```
#include <stdio.h>
int main() {
    int x = 10;
    if (x > 10);
        printf("Hello");
    else
        printf("World");
    return 0;
}
```

a) Prints "Hello" if x > 10, otherwise "World"
b) Always prints "Hello"
c) Compilation error
d) Undefined behavior

Q14. Interpret the behavior of this code snippet:

```
#include <stdio.h>
int main() {
    int x = -105, y;
    y = x > 0 ? x : -x;
    printf("y = %d\n", y);
    return 0;
}
```

a) prints y = -105
b) prints y = 105
c) prints y = 105
d) Causes Error

Q15. Examine the output for the following while loop:

```
#include <stdio.h>
int main() {
   int x = 5;
   do {
      printf("%d ", x);
      x--;
   } while (x > 0);
   return 0;
}
```

a) 5 4 3 2 1
b) 5 4 3 2
c) 4 3 2 1
d) Infinite loop

Q16. Analyze the following code and determine its pattern:

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 3; i++) {
      for (int j = 1; j <= i; j++) {
         printf("*");
      }
      printf("\n");
   }
   return 0;
}
```

a)
```
*
**
***
```
b)
```
***
**
*
```
c)
```
*
*
*
```
d) None of the above

Q17. Determine the total number of stars (*) printed in the output of the following code:

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 3; i++) {
      for (int j = 1; j <= 4; j++) {
         for (int k = 1; k <= 2; k++) {
            printf("*");
         }
      }
   }
   return 0;
}
```

a) 9
b) 24
c) 12

d) 18

Q18. Interpret the behavior of this code snippet:

```
#include <stdio.h>
int main() {
    for (int i = 10; i >= 1; i--) {
        printf("%d ", i);
    }
    return 0;
}
```

a) 10 9 8 7 6 5 4 3 2 1

b) 10 9 8 7 6 5 4 3 2

c) 10 9 8 7 6

d) None of the above

Q 19. Examine the behaviour of the following code:

```
#include <stdio.h>
int main() {
    int i = 0;
    while (i < 5) {
        i--;
    }
    return 0;
}
```

a) 5 times

b) 0 times

c) Infinite loop

d) Undefined behavior

Q 20. Analyze the following code and determine its behavior:

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) continue;
        if (i == 5) break;
        printf("%d ", i);
    }
    return 0;
}
```

a) 1 2 3 4

b) 1 2 4 5

c) 1 2 4

d) 1 2

Q21. Identify the output of the following code:

```
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    printf("%d", a + b);
    return 0;
}
```

a) 5

b) 10

c) 15

d) 50

Q22. Explain the functionality of the following code snippet:

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

```
void printMessage() {
   printf("Hello, World!");
}
int main() {
   printMessage();
   return 0;
}
```

a) Prints "Hello, World!" to the console.
b) Defines a function without calling it.
c) Returns an integer value.
d) Contains a syntax error.

Q23. Implement changes to modify the following code for calculating the square of a number:

```
#include <stdio.h>
int square(int x) {
   return x + x;
}
int main() {
   printf("%d", square(4));
   return 0;
}
```

a) Change return x + x; to return x * x;
b) Change int square(int x) to void square(int x)
c) Change printf("%d", square(4)); to printf("%d", square(2));
d) No changes are needed.

Q24. Analyze the output of this code:

```
#include <stdio.h>
void func(int n) {
   if (n > 0)
      func(n - 1);
   printf("%d ", n);
}
int main() {
   func(3);
   return 0;
}
```

a) Prints numbers from 0 to 3.
b) Prints numbers from 3 to 0.
c) Causes a stack overflow.
d) Prints nothing.

Q25. Evaluate the correctness of this code snippet:

```
#include <stdio.h>
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}

int main() {
   int x = 5, y = 10;
   swap(&x, &y);
   printf("%d %d", x, y);
```

```
    return 0;
}
```
a) Values of x and y remain unchanged.
b) The swap function will not compile due to incorrect syntax.
c) Values of x and y will be swapped successfully.
d) The program will crash at runtime.

Q26. Construct modifications to the following code snippet for calculating the factorial of a number using recursion:
```
#include <stdio.h>
int factorial(int n) {
    // Your code here
}
int main() {
    printf("%d", factorial(5));
    return 0;
}
```
a) Change it to use an iterative approach.
b) Add a base case and recursive call inside the function.
c) Remove the function entirely.
d) Change the return type to void.

Q27. Determine the output of the following code:
```
#include <stdio.h>
int main() {
    int arr[3] = {1, 2, 3};
    printf("%d", arr[1]);
    return 0;
}
```
a) 1
b) 2
c) 3
d) Array index out of bounds

Q28. Interpret the behavior of this code snippet:
```
#include <stdio.h>
void increment(int *num) {
    *num += 1;
}
int main() {
    int value = 10;
    increment(&value);
    printf("%d", value);
    return 0;
}
```
a) Prints 10.
b) Prints 11.
c) Causes a segmentation fault.
d) Prints nothing.

Q 29. Apply changes to the following code to reverse a string:
```
#include <stdio.h>
void reverse(char str[]) {
    // Your code here
}
```

```
int main() {
    char str[] = "Hello";
    reverse(str);
    printf("%s", str);
    return 0;
}
```

a) Replace // Your code here with a loop that swaps characters from the start and end.
b) Change char str[] = "Hello"; to char *str = "Hello";
c) Remove the reverse function entirely.
d) Change the return type of reverse to void.

Q 30. Analyze the following code and determine its behavior:

```
#include <stdio.h>
int main() {
    for (int i = 0; i < 5; i++) {
        if (i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

a) Prints "0 1 2".
b) Prints "0 1 2 3".
c) Prints "0 1 2 4".
d) Prints nothing.

Q31. Comment on the output of hte following C code.

```
#include <stdio.h>
struct temp {
  int a;
  int b;
  int c;
};
main() {
  struct temp p[] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
}
```

1. No Compile time error, generates an array of structure of size 3
2. No Compile time error, generates an array of structure of size 9
3. Compile time error, illegal declaration of a multidimensional array
4. Compile time error, illegal assignment to members of structure

Ans: No Compile time error, generates an array of structure of size 3

Q32. Choose a correct statement about C String.

```
char ary[] = "Hello.....!";
```

1. Character array, ary is a string
2. ary has no Null character at the end
3. String size is not mentioned
4. String can not contain special characters

Ans: Character array, ary is a string

Q33. Predict the output of C Program with Strings?

```
int main() {
char str[] = {'g', 'l', 'o', 'b', 'a', 'l'};
printf("%s", str);
return 0;
}
```

1. g
2. globe
3. globe\0
4. None of these

Ans: None of these

Q34. Analyze the output of the following piece of code?

```
int main() {
  char p[] = "GODZILLA";
  int i = 0;
  while(p[i] != '\0') {
    printf("%c", *(p+i));
    i++;
  }
  return 0;
}
```

1. G
2. GODZILLA
3. Compiler error
4. None of these

Ans: GODZILLA

Q35. Function that returns a pointer to the first character of a token is_____.

1. strstr()
2. strtok()
3. strspn()
4. strcpy()

Ans: strtok()

Q36. Predict the output of following code:

```
#include<stdio.h>
void main()
{
struct Student
{
int no;
char name[20];
}s;
s.no=8;
printf("%d",s.no);
}
```

1. 8
2. Compile time error
3. Nothing
4. Junk

Ans: 8

Q37. Predict the output?

```
void  main()
{
   int a[3] = {10,12,14};
   a[1]=20;
   int i=0;
   while(i<3)
   {
      printf("%d ", a[i]);
      i++;
   }
}
```

1. 20 12 14
2. 10 20 14
3. 10 12 20
4. Compiler error

Ans: 10 20 14

Q38. Predict the output of following code:

```
   #include <stdio.h>
   void main()
   {
      int a[2][3] = {1, 2, 3, 4, 5};
      int i = 0, j = 0;
      for (i = 0; i < 2; i++)
      for (j = 0; j < 3; j++)
      printf("%d", a[i][j]);
   }
```

1. 1 2 3 4 5 0
2. 1 2 3 4 5 5
3. 1 2 3 4 5 junk
4. Run time error

Ans: 1 2 3 4 5 0

Q40. Comment on the following 2 arrays with respect to P and Q.

```
int *a1[8];
int *(a2[8]);
```

P. Array of pointers
Q. Pointer to an array

1. a1 is P, a2 is Q
2. a1 is P, a2 is P
3. a1 is Q, a2 is P
4. a1 is Q, a2 is Q

Ans: a1 is P, a2 is P

SECTION-C (Coding Question) (5 marks)

QUESTION 1:

PROBLEM STATEMENT 1:

Write a C program that evaluates a mathematical expression based on user input. The expression is defined as:
result=a+b∗c/d−e
Where a, b, c, d, and e are integers entered by the user. The program must do the following:

1. Evaluate the expression with the default operator precedence, which follows the standard mathematical rules (multiplication and division are performed before addition and subtraction).
2. Modify the expression to explicitly control the order of operations using parentheses. In this case, addition and subtraction should be performed first, followed by multiplication and division.
3. Display both results clearly to the user.

INPUT FORMAT:

Five integers, a, b, c, d, and e, entered by the user, where d is non-zero to avoid division by zero.

OUTPUT FORMAT:

The program should display:

1. The result of the expression using the default operator precedence.
2. The result of the expression with explicitly controlled precedence using parentheses.

SOLUTION:

```
#include <stdio.h>

int main() {
    int a, b, c, d, e;
    int result_default, result_modified;

    // Input from the user
  //  printf("Enter values for a, b, c, d, and e (d != 0): ");
   printf("Enter five integers (a, b, c, d, e): ");
    scanf("%d %d %d %d %d", &a, &b, &c, &d, &e);

    // Check for division by zero
    if (d == 0) {
        printf("Division by zero is not allowed.\n");
        return 1;
    }

    // Default operator precedence
    result_default = a + b * c / d - e;

    // Modified operator precedence with parentheses
    result_modified = ((a + b) - e) * (c / (float)d);

    // Display results
    printf("Result with default operator precedence: %d\n", result_default);
    printf("Result with modified operator precedence: %d\n", result_modified);

    return 0;
}
```

TEST CASES:

TEST CASES 1:

INPUT:
Enter five integers (a, b, c, d, e): 10 20 30 5 15
OUTPUT:
Default precedence result: 115
Modified precedence result: 90

TEST CASES 2:

INPUT:

Enter five integers (a, b, c, d, e): 5 10 15 3 2
OUTPUT:
Default precedence result: 53
Modified precedence result: 65

TEST CASES 3:
INPUT:
Enter five integers (a, b, c, d, e): 100 50 25 5 10
OUTPUT:
Default precedence result: 340
Modified precedence result: 700

TEST CASES 4:
INPUT:
Enter five integers (a, b, c, d, e): 10 20 0 2 5
OUTPUT:
Default precedence result: 5
Modified precedence result: 0

QUESTION 2:

PROBLEM STATEMENT 2:

Write a C program that evaluates a student's performance based on their scores in three subjects: Math, Science, and English.

- If the student scores **above 80 in all three subjects**, print **"Excellent! You are an all-rounder."**
- If the student scores **above 90 in Math but below 70 in any other subject**, print **"Focus more on other subjects."**
- If the student's **average score is above 75**, but they did not score above 80 in all subjects, print **"Good performance, keep improving."**
- If none of the above conditions are met, print **"You need to work harder."**

INPUT FORMAT:
The program takes three integers as input, representing the scores in Math, Science, and English.
Input format: math_score ,science_score ,english_score
OUTPUT FORMAT:
The program outputs one of the following statements based on the conditions:

- "Excellent! You are an all-rounder."
- "Focus more on other subjects."
- "Good performance, keep improving."
- "You need to work harder."

SOLUTION:

```
#include <stdio.h>

int main() {
   // Declare variables for scores
   int math, science, english;
   float average;

   // Input scores
   printf("Enter the scores for Math, Science, and English: ");
   scanf("%d %d %d", &math, &science, &english);

   // Calculate average
   average = (math + science + english) / 3.0;

   // Evaluate performance
   if (math > 80 && science > 80 && english > 80) {
      printf("Excellent! You are an all-rounder.\n");
   } else if (math > 90 && (science < 70 || english < 70)) {
      printf("Focus more on other subjects.\n");
   } else if (average > 75) {
      printf("Good performance, keep improving.\n");
```

```
    } else {
        printf("You need to work harder.\n");
    }

    return 0;
}
```

TEST CASES:

TEST CASES 1:
INPUT: 85 90 88
OUTPUT: Excellent! You are an all-rounder.
TEST CASES 2:
INPUT: 92 65 75
OUTPUT: Focus more on other subjects.

TEST CASES 3:
INPUT: 78 80 70
OUTPUT: Good performance, keep improving.

TEST CASES 4:
INPUT: 60 65 70
OUTPUT: You need to work harder.

TEST CASES 5:
INPUT: 91 85 79
OUTPUT: Good performance, keep improving.

QUESTION 3:
PROBLEM STATEMENT 3:
Find the Smallest Divisor (Other than 1)
Write a C program that takes a positive integer n as input and finds the smallest divisor of n greater than 1 using a loop.
If n is a prime number (i.e., no divisors other than 1 and itself), print **"Prime number."**
Otherwise, print the smallest divisor.
INPUT:
A single positive integer n ($1 \leq n \leq 10^6$)
OUTPUT:
- If n is prime, output: "Prime number."
- Otherwise, output the smallest divisor of n greater than 1.

SOLUTION:

```
#include <stdio.h>

int smallest_divisor(int n) {
    // Check for divisibility starting from 2
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return i;
        }
    }
    return n; // If no divisors are found, then n is prime
}

int main() {
    int n;
    scanf("%d", &n);

    // Handle the case where n = 1 (no divisors other than itself)
    if (n == 1) {
        printf("Prime number.\n");
    } else {
        int divisor = smallest_divisor(n);
        if (divisor == n) {
            printf("Prime number.\n");
```

```
        } else {
            printf("%d\n", divisor);
        }
    }

    return 0;
}
```

TEST CASES:

TEST CASES 1:
INPUT: 10
OUTPUT: 2

TEST CASES 2:
INPUT: 17
OUTPUT: Prime number.

TEST CASES 3:
INPUT: 1
OUTPUT: Prime number.

TEST CASES 4:
INPUT: 25
OUTPUT: 5

TEST CASES 5:
INPUT: 49
OUTPUT: 7

QUESTION 4:
PROBLEM STATEMENT 4:
Check for Armstrong Numbers in a Range

- Write a C program to find and print all **Armstrong numbers** within a given range [start, end].
- An **Armstrong number** (also known as a narcissistic number) is a number where the sum of its digits raised to the power of the number of digits equals the number itself.
- For example:

153 is an Armstrong number because 13+53+33=1531^3 + 5^3 + 3^3 = 15313+53+33=153.
9474 is an Armstrong number because 94+44+74+44=94749^4 + 4^4 + 7^4 + 4^4 = 947494+44+74+44=9474.

INPUT:
1. Enter the starting number of the range (start).
2. Enter the ending number of the range (e**nd).**

OUTPUT:
Print all Armstrong numbers within the range [start, end].
If no Armstrong numbers exist in the range, print "**No Armstrong numbers found.**"

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

// Function to check if a number is an Armstrong number
int isArmstrong(int num) {
    int originalNum, remainder, n = 0;
    double result = 0.0;

    originalNum = num;

    // Calculate the number of digits
```

```
    while (originalNum != 0) {
        originalNum /= 10;
        n++;
    }

    originalNum = num;

    // Calculate the sum of the digits raised to the power of n
    while (originalNum != 0) {
        remainder = originalNum % 10;
        result += pow(remainder, n);
        originalNum /= 10;
    }

    // Check if the sum is equal to the original number
    return (int)result == num;
}

int main() {
    int start, end;

    // Input range

    scanf("%d", &start);

    scanf("%d", &end);



    int found = 0;

    // Find Armstrong numbers in the range
    for (int i = start; i <= end; i++) {
        if (isArmstrong(i)) {
            printf("%d ", i);
            found = 1;
        }
    }

    if (!found) {
        printf("No Armstrong numbers found.");
    }

    return 0;
}
```

TEST CASES:

TEST CASES 1:
INPUT:
100
500
OUTPUT:
153 370 371 407

TEST CASES 2:
INPUT:
1
1000
OUTPUT:
1 2 3 4 5 6 7 8 9 153 370 371 407

TEST CASES 3:
INPUT:

4000
5000
OUTPUT:
No Armstrong numbers found.

TEST CASES 4:
INPUT:
9000
10000
OUTPUT:
9474

QUESTION 5:

PROBLEM STATEMENT 5:

Write a C program that reads an integer input and determines if the number is positive, negative, or zero. The program should then display the result.

INPUT:

The program expects a single integer input from the user.

OUTPUT:

- If the number is positive, print: "Positive".
- If the number is negative, print: "Negative".
- If the number is zero, print: "Zero".

SOLUTION:

```
#include <stdio.h>

int main() {
   int num;

   // Prompt user for input
   printf("Enter an integer: ");
   scanf("%d", &num);

   // Check if the number is positive, negative, or zero using if-else
   if (num > 0) {
      printf("Positive\n");
   } else if (num < 0) {
      printf("Negative\n");
   } else {
      printf("Zero\n");
   }

   return 0;
}
```

TEST CASES:

TEST CASES 1:
INPUT: Enter an integer: 5
OUTPUT: Positive
TEST CASES 2:
INPUT: Enter an integer: -3
OUTPUT: Negative
TEST CASES 3:
INPUT: Enter an integer: 0
OUTPUT: Zero
TEST CASES 4:
INPUT: Enter an integer: 1000000
OUTPUT: Positive

TEST CASES 5:
INPUT: Enter an integer: -1000000
OUTPUT: Negative

Problem Statement 6:
Write a C program to print the count of even digits and odd digits of a number.

Input Format
The first line contains an integer

Output Format

Print the count of even digits in a number
Print the count of odd digits in a number

Explanation:
Suppose the number entered by the user is 5694113
3, 1, 1, 9 and 5 are the odd digits in the number. Therefore, the count of odd digits is 5
4 and 6 are the even digits in a number. Therefore the count of even digits is 2.

Sample:
Input 1:
2134

Sample Output 1
2
2

Sample Input 2
342
Sample Output 2
2
1
Sample Input 3
1313
Sample Output 3
0
4

……
SOLUTION:

```
#include <stdio.h>
int main() {
   int num,cE=0,cO=0,d;
   scanf("%d",&num);
   while(num!=0)
   {
      d=num%10;
      if(d%2==0)
         cE++;
      else
         cO++;
      num=num/10;

   }
   printf("%d\n%d",cE,cO);
   return 0;
}
```

Test case	Input	Output
Sample Test case-1	1234	2 2

Sample Test case-2	342	2 1
Sample Test case-3	1313	0 4
Test Case1	4680	4 0
Test Case2	1112	1 3
Test Case3	1011	1 3
Test Case4	23456	3 2
Test Case5	-11123	1 4
Test Case6	78659	2 3

Problem Statement 7:
Write a C program to print the day of the week for left most digit of a number. A number can have only valid digits from 1 to 7 if the any one of the digit is other than 1 to 7 i.e. 0, 8 or 9 the display a message "Invalid number".
Display day of the week as per digit extracted (left most digit of the number) from a number

Digit	Display
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7	Sunday

Constraints are:
The number can have only valid digits from 1 to 7 only.
Input Format
The first line contains an integer number
Output Format
Print the day of the week corresponding to left most digit of a number.
Sample Input 1
132456 //integer number
Sample Output 1
Monday //the left most digit of a number is 1 and display Monday for 1 as per the given table
Sample Input 2:
79
Sample Output 2
Invalid number //As the number is having a digit 9
Sample Input 3
63
Sample Output 3
Saturday
Sample Input 3
823
Sample Output 3
Invalid number

……
SOLUTION:

```
#include <stdio.h>
int main() {
   int num,d,flag=0;
   scanf("%d",&num);
```

```
    while(num!=0)
    {
        d=num%10;
        if(d==9 || d==8 || d==0)
            {
            printf("Invalid number");
            break;
            }
        num=num/10;
    }

    if(d==1)
        printf("Monday");
    else if(d==2)
        printf("Tuesday");
    else if(d==3)
        printf("Wednesday");
    else if(d==4)
        printf("Thursday");
    else if(d==5)
        printf("Friday");
    else if(d==6)
        printf("Saturday");
    else if(d==7)
        printf("Sunday");
    return 0;
}
```

Test cases	**Input**	**Output**
Sample Test case-1	132	Monday
Sample Test case-2	79	Invalid number
Sample Test case-3	63	Saturday
Test Case1	812	Invalid number
Test Case2	103	Invalid number
Test Case3	432	Thursday
Test Case4	34	Wednesday
Test Case5	55742	Friday
Test Case6	22	Tuesday

Problem Statement 8.
Objective: Write a C program that defines a function to check if a number is prime or not. The program should:
Take an integer input from the user. checkPrime() function is called from main() function and it will print “Prime” if the number is prime, otherwise print “Not prime”
Constraint:
The input number will be greater than 1.
Sample case 1:
Input:
11
Output:
Prime
Sample case 2:
Input:
4
Output:
Not prime

Required Solution:

```
#include <stdio.h>
void checkPrime(int num);
```

```
int main()
{
   int num;
   scanf("%d", &num);
    checkPrime(num);
   return 0;
}
// Function to check if a number is prime
void checkPrime(int N)
{
   // initially, flag is set to true or 1
   int flag = 1;
     // loop to iterate through 2 to N/2
   for (int i = 2; i <= N / 2; i++) {

      // if N is perfectly divisible by i
      // flag is set to 0 i.e false
      if (N % i == 0) {
         flag = 0;
         break;
      }
   }

   if (flag)
      printf("Prime");

   else
      printf("Not prime");

}
```

Hidden Test Cases:
Input:
2
Output:
Prime
Input:
7919
Output:
Prime
Input:
8000
Output:
Not prime
Input:
29
Output:
Prime
Input:
10
Output:
Not prime
Input:
1411
Output:
Not prime

Problem Statement 9:
Objective: Develop a C program to compute the transpose of a matrix. The program should:

Prompt the user to enter the dimensions of a matrix (rows and columns).
Take the elements of the matrix as input from the user.
Use a function named transposeMatrix() that takes the original matrix, its dimensions (rows and columns), and an output matrix to store the transpose.
Print the original matrix and its transpose.

Input Format:

- The first line contains two space-separated integers, **r** and **c**, representing the number of rows and columns, respectively, of the matrix.
- The next **r** lines contain **c** space-separated integers each, representing the elements of the matrix.

Output Format:

- First, print **Original Matrix:** followed by the matrix as entered by the user.
- Then, print **Transpose Matrix:** followed by the transpose of the matrix.

Constraints:

- 1 <= r, c <= 100
- 0 <= matrix elements <= 1000

Required Solution:

```
#include <stdio.h>
// Function to transpose a matrix
void transposeMatrix(int r, int c, int matrix[][c], int transposed[][r]) {
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            transposed[j][i] = matrix[i][j];
        }
    }
}

// Function to print a matrix
void printMatrix(int r, int c, int matrix[][c]) {
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int r, c;
    printf("\nEnter number of rows and columns: ");
    scanf("%d %d", &r, &c);

    int matrix[r][c];
    int transposed[c][r]; // Transpose matrix will have reversed dimensions

    printf("Enter matrix elements:\n");
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    printf("Original Matrix:\n");
    printMatrix(r, c, matrix);

    transposeMatrix(r, c, matrix, transposed);

    printf("Transpose Matrix:\n");
    printMatrix(c, r, transposed);

    return 0;
}
```

Sample Input 1:

Enter number of rows and columns: 2 3
Enter matrix elements:
1 2 3 4 5 6
Sample Output 1:
Original Matrix:
1 2 3
4 5 6
Transpose Matrix:
1 4
2 5
3 6
Sample Input 2:
Enter number of rows and columns: 3 3
Enter matrix elements:
1 0 0 0 1 0 0 0 1
Sample Output 2:
Original Matrix:
1 0 0
0 1 0
0 0 1
Transpose Matrix:
1 0 0
0 1 0
0 0 1
Hidden Test Case 1:
Input:
Enter number of rows and columns: 1 2
Enter matrix elements:
4 5
Output:
Original Matrix:
4 5
Transpose Matrix:
4
5
Hidden Test Case 2:
Input:
Enter number of rows and columns: 1 4
Enter matrix elements:
1 2 3 4
Output:
Original Matrix:
1 2 3
4
Transpose Matrix:
1
2
3
4
Hidden Test Case 3:
Input:
Enter number of rows and columns: 3 2
Enter matrix elements:
3 4 5 7 9 11
Output:
Original Matrix:
3 4
5 7
9 11
Transpose Matrix:
3 5 9
4 7 11

Hidden Test Case 4:
Input:
Enter number of rows and columns: 3 4
Enter matrix elements:
11 12 13 9 4 6 1 6 8 1 7 8
Output:
Original Matrix:
11 12 13 9
4 6 1 6
8 1 7 8
Transpose Matrix:
11 4 8
12 6 1
13 1 7
9 6 8
Hidden Test Case 5:
Input:
Enter number of rows and columns: 1 3
Enter matrix elements:
3 6 8
Output:
Original Matrix:
3 6 8
Transpose Matrix:
3
6
8
Hidden Test Case 6:
Input
Enter number of rows and columns: 2 2
Enter matrix elements:
1 2 3 4
Output
Original Matrix:
1 2
3 4
Transpose Matrix:
1 3
2 4

Problem Statement 10
Write a function is_leap_year which takes the year as its argument and checks whether the year is a leap year or not and then displays an appropriate message on the screen. If the year is Leap year then display a message <year> is not leap year else <year> is not leap year.
Input Format:
The input consists of one integer, representing the year
Output Format:
The output prints the year is leap year if the year is leap year else year is not leap year.
Code Constraints:
NA
SOLUTION

```
#include <stdio.h>
int is_leap_year(int year) {
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        return 1;
    } else {
        return 0;
    }
}
int main() {
    int year;
```

```
    scanf("%d", &year);
    if (is_leap_year(year)) {
        printf("%d is a leap year", year);
    } else {
        printf("%d is not leap year" , year);}
}
```

Sample Testcase 1
Input: 2024
Output: 2024 is a leap year
Sample Testcase 2
Input: 2022
Output: 2022 is not leap year
Hidden Testcase 1 - (Easy) Weightage 10%
Input: 2019
Output: 2019 is not leap year
Hidden Testcase 2 - (Easy) Weightage 10%
Input: 2013
Output: 2013 is not leap year
Hidden Testcase 3 - (Medium) Weightage 15%
Input: 2010
Output: 2010 is not leap year
Hidden Testcase 4 - (Medium) Weightage 15%
Input: 2000
Output: 2000 is a leap year
Hidden Testcase 5 - (Hard) Weightage 25%
Input: 2012
Output: 2012 is a leap year
Hidden Testcase 6 - (Hard) Weightage 25%
Input: 2016
Output: 2016 is a leap year

Q11. Problem Statement
Write a program in C to find an element in a 1-D array. Initially you have to input 5 elements in an array and the element you want to search. The program should print the position/ (index no +1) of the element in the array (for the first successful match only) otherwise it should print -1.
NOTE: If the search element is found at 2 or more positions in the array then you have to print only the first matching position where the element is found.

Input Format:
First line of input consists of 5 integer numbers to be stored in an array.
Second line should input the element that is to be searched.
Output Format:
The output prints the position of the element in the array (if found) otherwise print -1 (in case the element is not found).
Code Constraints:
NA
Solution:

```
#include<stdio.h>
int main()
{
int arr[5],i,n,element,flag=0,pos=-1;
for(i=0;i<5;i++)
{
scanf("%d",&arr[i]);
}
scanf("%d",&element);//Enter the element you want to search
for(i=0;i<5;i++)
{
if(element==arr[i])
{
pos=i+1;
printf("%d",pos);
flag=1;
```

```
break;
}
}
if(flag==0)
{
printf("%d",pos);
}
}
```

Sample Test case 1
Input:
1 4 3 4 3 // Elements of the array
4 //element to be searched
Output:
2 //position of element in array

Sample Test case 2
Input:
2 7 43 9 74
3
Output:
-1

Hidden Test case 1 - (Easy) Weightage 10%
Input:
4 11 3 13 17
11
Output:
2
Hidden Test case 2 - (Easy) Weightage 10%
Input:
54 77 32 189 32
32
Output:
3
Hidden Test case 3 - (Medium) Weightage 15%
Input:
90 80 70 21 34
90
Output:
1
Hidden Test case 4 - (Medium) Weightage 15%
Input:
-25 -45 87 53 20
-45
Output:
2
Hidden Test case 5 - (Hard) Weightage 25%
Input:
0 0 0 0 0
0
Output:
1
Hidden Test case 6 - (Hard) Weightage 25%
Input:
22 22 22 22 22
33
Output:
-1

Q12. Problem Statement:
Write a C program to find the largest and second-largest numbers from an array of integers using a function. The program should ask the size of array and take integers as inputs according to the size of array and determine the largest and second-largest numbers in the array.
Constraints
The maximum array size should be 1<MAX_SIZE<100
Sample Input 1:
4 //Size of the Array
5 6 7 8 //Enter the elements in the array equal to the size of array
Sample Output 1:
8 //Largest Integer
7 //2nd Largest Integer
Sample Input 2:
3 //Size of the Array
15 6 17 //Enter the elements in the array equal to the size of array
Sample Output 2:
17 //Largest Integer
15 //2nd Largest Integer
Solution:

```
#include <stdio.h>
#define MAX_SIZE 100
void findLargestAndSecondLargest(int arr[], int size)
{
   int i, largest, secondLargest;
   largest = arr[0], secondLargest = arr[1];
   if (secondLargest > largest)
   {
      int temp = largest;
      largest = secondLargest;
      secondLargest = temp;
   }
   for (i = 2; i < size; i++)
   {
      if (arr[i] > largest)
      {
         secondLargest = largest;
         largest = arr[i];
      } else if (arr[i] > secondLargest && arr[i] != largest) {
         secondLargest = arr[i];
      }
   }

   printf("%d\n%d", largest, secondLargest);
}

int main()
{
   int arr[MAX_SIZE];
   int size, i;
   scanf("%d", &size);
   for (i = 0; i < size; i++)
       scanf("%d", &arr[i]);
   findLargestAndSecondLargest(arr, size);
   return 0;
}
```

Hidden Test Case 1:
Input:
6
4 7 11 23 45 6
Output:

23
11
Hidden Test Case 2:
Input:
5
1 66 17 9 2
Output:
66
17
Hidden Test Case 3:
Input:
2
56 90
Output:
90
56
Hidden Test Case 4:
Input:
6
4 88 1 23 56 90
Output:
90
88
Hidden Test Case 5:
Input:
3
23 56 90
Output:
90
56
Hidden Test Case 6:
Input:
7
55 8 43 32 23 56 9
Output:
56
55

Q13. Problem Statement:
Write a C program using concept of arrays that allows you to enter records of students, input their name, roll number, and marks for three subjects, and then calculates the average marks for each student and display it.

Required Solution:

```
#include <stdio.h>
struct Student {
    char name[50];
    int rollNo;
    float marks[3];
    float averageMarks;
};

int main() {
    struct Student students[50];
    int n, i, j;
    //Enter the total number of students
    scanf("%d", &n);
    // Input student records
    for (i = 0; i < n; i++)
{
        scanf(" %s", students[i].name);  // Read name with spaces
// Enter Roll No of the student
scanf("%d", &students[i].rollNo);
```

```
        // Enter marks for 3 subjects
        for (j = 0; j < 3; j++)
        {
            //printf("Subject %d: ", j + 1);
            scanf("%f", &students[i].marks[j]);
        }
    }

    // Calculate average marks for each student
    for (i = 0; i < n; i++) {
        float totalMarks = 0.0;
        for (j = 0; j < 3; j++) {
            totalMarks += students[i].marks[j];
        }
        students[i].averageMarks = totalMarks / 3.0;
    }
//Prints the average marks of the students
    for (i = 0; i < n; i++)
    {
        printf("%.2f\n", students[i].averageMarks);
    }

    return 0;
}
```

Sample Input 1:

1 //Enter total number of Students
Ravi //Enter name of the student
1 // Enter roll no.
3 5 7 //Enter marks for 3 Subjects

Sample Output 1:

5.00 //Average marks of the student

Sample Input 2:

2 //Enter total number of Students
Tina //Enter name
11 //Enter roll no
56 78 77 //Enter marks for 3 subjects
Ram //Enter name
12 //Enter roll no
89 67 78 //Enter marks for 3 subjects

Sample Output 2:

70.33 //Average marks of first student
78.00 //Average marks of 2nd student

Sample Input 3

2
Neelam
32
67 78 98
Harry
33
65 88 56

Sample Output 3

81.00
69.67

Sample Input 4

1
Siya
3
56 88 99

Sample Output 4

81.00

Sample Input 5

2

Neelam
40
45 77 86
aakash
45
78 88 67
Sample Output 5
69.33
77.67
Sample Input 6
3
Neelu
14
95 97 96
Bheem
49
88 67 86
Sample Output 6
96.00
80.33

Q14. Problem Statement
Ten numbers are entered from the keyboard into an array. The number to be searched is entered through the keyboard by the user. Write a program to find if the number to be searched is "found" or "not found" in the array and if it is found then display the number of times it appears in the array.

Input Format:
The input consists of ten integer numbers as array elements and one integer number (to be searched).
Output Format:
The output prints "found n times"/ "Not found", where n is number of times the element found in array.
Code Constraints:
NA
Solution:

```
#include<stdio.h>
int main()
{
int i,a[10],n,count=0;
for(i=0;i<10;i++)
{
scanf("%d",&a[i]);
}
scanf("%d",&n);
for(i=0;i<10;i++)
{
if(n==a[i])
{
count++;
}
}
if(count==0)
{
printf("Not found");
}
else
{
printf("found %d times",count);
}
 return 0;
}
```

Sample Testcase 1

Input:
5 6 3 4 5 1 9 4 3 7
5

Output:
found 2 times

Sample Testcase 2
Input:
1 2 3 4 5 6 7 8 9 3
20

Output:
Not found

Hidden Testcase 1 - (Easy) Weightage 10%
Input:
1 8 5 4 7 8 6 4 9 23
1
Output:
found 1 times

Hidden Testcase 2 - (Easy) Weightage 10%
Input:
90 87 54 -12 9 -16 6 7 -12 12
-12
Output:
found 2 times

Hidden Testcase 3 - (Medium) Weightage 15%
Input:
66 22 11 55 66 33 11 11 11 11
11
Output:
found 5 times

Hidden Testcase 4 - (Medium) Weightage 15%
Input:
14 12 34 32 23 12 45 65 65 21
33
Output:
Not found

Hidden Testcase 5 - (Hard) Weightage 25%
Input:
4 7 8 12 43 65 87 98 3 10
20
Output:
Not found

Hidden Testcase 6 - (Hard) Weightage 25%
Input:
3 4 7 3 2 1 4 3 2 3
3
Output:
found 4 times

Q15. Problem Statement

Write a program in C to print the difference of largest and smallest number in an array of 5 elements. Input will be taken as 5 integer numbers and save these numbers in an array. Find the largest and smallest number from this array. Calculate the difference (largest-smallest). Print this difference as output.

Input Format:
The input consists of 5 integer numbers.
Output Format:
The output prints only one integer number that is difference between largest number and smallest number.
Code Constraints:

NA

Solution:

```
#include <stdio.h>
int main()
{
int arr[5],large,small,i,diff;
for(i=0;i<5;i++)
scanf("%d",&arr[i]);
large=small=arr[0];
for(i=1;i<5;i++)
{
 if(arr[i]>large)
 {
 large=arr[i];
 }
 if(arr[i]<small)
 {
 small=arr[i];
 }
}
diff=large-small;
printf("%d",diff);
}
```

Sample Testcase 1

Input:
3 //Enter 5 integer numbers and save in array
7
9
1
4

Output:
8 //difference between largest and smallest number

Sample Testcase 2

Input:
5
-6
2
0
15

Output:
21

Hidden Testcase 1 - (Easy) Weightage 10%

Input:
4
4
4
4
4

Output:
0

Hidden Testcase 2 - (Easy) Weightage 10%

Input:
200
300
-100
-500
4

Output:
800

Hidden Testcase 3 - (Medium) Weightage 15%

Input:

5
12
13
14
66
Output:
61

Hidden Testcase 4 - (Medium) Weightage 15%

Input:
2
13
19
44
23
Output:
42

Hidden Testcase 5 - (Hard) Weightage 25%

Input:
-45
-90
-12
-67
-200
Output:
188

Hidden Testcase 6 - (Hard) Weightage 25%

Input:
0
0
0
0
0
Output:
0

Problem 16: Concatenating Two Strings

Title: String Concatenation Without Built-in Functions

Write a program to concatenate two strings without using built-in functions.

Sample Test Cases

1. **Input:**
Hello
World
Output:
HelloWorld
2. **Input:**
Open
AI
Output:
OpenAI

Test Cases Table

Test Case	String	Expected Output
1	Hello World	HelloWorld
2	Open AI	OpenAI
3	Learn C	LearnC

4	String Handling	StringHandling
5	Coding Fun	CodingFun

Solution in C

```
#include <stdio.h>

int main() {
   char str1[50], str2[50], result[100];
   int i = 0, j = 0;
   scanf("%s", str1);
   scanf("%s", str2);

   // Copy str1 into result
   while (str1[i] != '\0') {
      result[i] = str1[i];
      i++;
   }

   // Append str2 to result
   while (str2[j] != '\0') {
      result[i] = str2[j];
      i++;
      j++;
   }
   result[i] = '\0';  // Null-terminate the string

   printf("%s", result);
   return 0;
}
```

Bloom's Taxonomy: Applying
Difficulty Level: Moderate

Problem 17: Counting Vowels and Consonants
Title: Vowel and Consonant Counter
Write a program to count the number of vowels and consonants in a given string. Assume the string only contains alphabetic characters.

Sample Test Cases
1. **Input:** Hello
Output:
2 // Number of vowels
3 // Number of consonants
2. **Input:** OpenAI
Output:
4
2

Test Cases Table

Test Case	Input String	Expected Vowels Expected Consonants
1	Hello	2 3
2	OpenAI	4 2
3	CProgramming	3 9

4	Vowels	2 4
5	Fun	1 2

Solution in C

```
#include <stdio.h>

int main() {
   char str[100];
   int vowels = 0, consonants = 0;

   printf("Enter a string: ");
   scanf("%s", str);

   for (int i = 0; str[i] != '\0'; i++) {
      char ch = str[i];
      if (ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' ||
         ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
         vowels++;
      } else {
         consonants++;
      }
   }

   printf("%d\n", vowels);
   printf("%d", consonants);

   return 0;
}
```

Bloom's Taxonomy: Applying
Difficulty Level: Easy

Problem 18: Array of Structures for Employee Records
Title: Managing Employee Records
Define a structure called Employee with fields for name (string), employee ID (integer), and salary (float). Use an array of structures to store details for 3 employees and display their details.

Sample Test Cases
1. **Input: 3**

```
3 // number of employee
John
1001
50000
Alice
1002
60000
Bob
1003
55000
```

Output:
Employee 1: Name = John, ID = 1001, Salary = 50000
Employee 2: Name = Alice, ID = 1002, Salary = 60000
Employee 3: Name = Bob, ID = 1003, Salary = 55000

Test Cases Table

Test Case	Input	Output
1	3 John, 1001, 50000	Employee 1: Name = John, ID = 1001, Salary = 50000 Employee 2: Name = Alice, ID = 1002, Salary = 60000

	Alice, 1002, 60000 Bob, 1003, 55000	Employee 3: Name = Bob, ID = 1003, Salary = 55000
2	1 David, 2001, 40000	Employee 1: Name = David, ID = 2001, Salary = 40000
3	2 Alan, 3001, 48000 Sarah, 3002, 51000	Employee 1: Name = Alan, ID = 3001, Salary = 48000 Employee 2: Name = Sarah, ID = 3002, Salary = 51000
4	4 Mike, 4001, 60000 Tom, 4002, 55000 Eve, 2002, 45000 Mark, 2003, 52000	Employee 1: Name = Mike, ID = 4001, Salary = 60000 Employee 2: Name = Tom, ID = 4002, Salary = 55000 Employee 3: Name = Eve, ID = 2002, Salary = 45000 Employee 4: Name = Mark, ID = 2003, Salary = 52000
5	5 Chris, 3003, 53000 Jane, 5001, 45000 Eric, 5002, 47000 Paul, 5003, 52000 Lily, 4003, 58000	Employee 1: Name = Chris, ID = 3003, Salary = 53000 Employee 2: Name = Jane, ID = 5001, Salary = 45000 Employee 3: Name = Eric, ID = 5002, Salary = 47000 Employee 4: Name = Paul, ID = 5003, Salary = 52000 Employee 5: Name = Lily, ID = 4003, Salary = 58000

Solution in C

```
#include <stdio.h>

struct Employee
{
   char name[50];
   int employeeID;
   float salary;
};

int main()
{
   int n;
   scanf("%d",&n);
   struct Employee employees[n];

   for (int i = 0; i < n; i++)
   {

      scanf("%s", employees[i].name);

      scanf("%d", &employees[i].employeeID);

      scanf("%f", &employees[i].salary);
   }

   printf("\nEmployee Details:\n");
   for (int i = 0; i < n; i++)
   {
      printf("Employee %d: Name = %s, ID = %d, Salary = %.2f\n",
            i + 1, employees[i].name, employees[i].employeeID, employees[i].salary);
   }

   return 0;
}
```

Bloom's Taxonomy: Application
Difficulty Level: Moderate

Problem 19: Structure with Array Members
Title: Storing Marks for Multiple Subjects

Create a structure called StudentMarks that stores the name of the student (string) and an array of marks for 3 subjects (integers). Write a program to take input and display the total marks for all subjects.

Sample Test Cases

1. **Input:**

Alice
85
90
88

Output: 263

2. **Input:**

John
70
75
80

Output:225

Test Cases Table

Test Case	Input	Output
1	Alice 85, 90, 88	263
2	John 70, 75, 80	225
3	Bob 60, 70, 80	210
4	David 95, 92, 90	277
5	Eve 88, 77, 89	254

Solution in C

```
#include <stdio.h>
struct StudentMarks {
   char name[50];
   int marks[3];
};

int main() {
   struct StudentMarks student;
   int total_marks=0;

   scanf("%s", student.name);


   for (int i = 0; i < 3; i++) {
      scanf("%d", &student.marks[i]);
   }
   for (int i = 0; i < 3; i++)
   {
      total_marks= total_marks+student.marks[i];

   }
   printf("%d ", total_marks);

   return 0;
}
```

Bloom's Taxonomy: Application
Difficulty Level: Easy

Problem 20: Searching Employee Records Using Array of Structures
Title: Search for an Employee by ID

Define a structure Employee with name (string), employee ID (integer), and salary (float). Store details for 5 employees in an array of structures. Allow the user to enter an employee ID and search for the corresponding employee's details i.e. Name and salary of that employee. If not found then print "Employee does not exist in the record"
Note: Structure of employee has been initialized in this program.

```
{"John", 1001, 50000},
{"Alice", 1002, 60000},
{"Bob", 1003, 55000},
{"David", 1004, 47000},
{"Eve", 1005, 52000}
```

Sample Test Cases

1. **Input:**

1002 // Enter Employee ID to search

Output:

Alice
60000.00

2. **Input:**

1005

Output:

Employee does not exist in the record

Test Cases Table

Test Case	Searched ID	Expected Output
1	1002	Alice 60000.00
2	2004	Employee does not exist in the record
3	1005	Eve 52000.00
4	1001	John 50000.00
5	5005	Employee does not exist in the record

Solution in C

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

struct Employee {
   char name[50];
   int employeeID;
   float salary;
};

int main() {
   struct Employee employees[5] = {
      {"John", 1001, 50000},
      {"Alice", 1002, 60000},
      {"Bob", 1003, 55000},
      {"David", 1004, 47000},
      {"Eve", 1005, 52000}
   };

   int searchID;

   scanf("%d", &searchID);

   int found = 0;
   for (int i = 0; i < 5; i++) {
      if (employees[i].employeeID == searchID) {
         printf("%s,%.2f\n", employees[i].name, employees[i].salary);
         found = 1;
         break;
```

```
        }
    }

    if (!found) {
        printf("Employee does not exist in the record ");
    }

    return 0;
}
```

Bloom's Taxonomy: Analysis
Difficulty Level: Moderate

10 MARKS

PROBLEM STATEMENT 1:
Once upon a time, a curious student Manu sat in front of their computer, eager to learn about leap years. "Is this year a leap year?" they wondered aloud. The wise computer, always ready with an answer, replied with a simple rule. "A year is a leap year if it's divisible by 4, but not by 100, unless it's also divisible by 400." Intrigued, the student thought deeply about the rule, repeating it to themselves. They realized how clever this logic was and felt proud of the newfound knowledge. From that day on, they could easily identify a leap year with confidence.
INPUT:
The program accepts a single integer input from the user, which represents the year to be checked.
OUTPUT:
The program outputs a message indicating whether the given year is a leap year or not.
SOLUTION:

```
#include <stdio.h>

int main() {
   int year;

   // Accept input from the user
   printf("Enter a year: ");
   scanf("%d", &year);

   // Check if the year is a leap year
   if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
      printf("%d is a Leap Year.\n", year);
   } else {
      printf("%d is NOT a Leap Year.\n", year);
   }

   return 0;
}
```

TEST CASES:
TEST CASE 1:
INPUT: 2000
OUTPUT: 2000 is a Leap Year.
TEST CASE 2:
INPUT:1900
OUTPUT: 1900 is NOT a Leap Year.
TEST CASE 3:
INPUT:2024
OUTPUT: 2024 is a Leap Year.
TEST CASE 4:
INPUT:2023
OUTPUT: 2023 is NOT a Leap Year.
TEST CASE 5:
INPUT:2100
OUTPUT: 2100 is NOT a Leap Year.
TEST CASE 6:
INPUT:2400
OUTPUT: 2400 is a Leap Year.
Explanation for Test Cases:

1. **2000:** Divisible by 400, so it's a leap year.
2. **1900:** Divisible by 100 but not by 400, so it's not a leap year.
3. **2024:** Divisible by 4 and not by 100, so it's a leap year.
4. **2023:** Not divisible by 4, so it's not a leap year.
5. **2100:** Divisible by 100 but not by 400, so it's not a leap year.
6. **2400:** Divisible by 400, so it's a leap year.

PROBLEM STATEMENT 2:
One sunny afternoon, a young student Manu sat at their computer, wondering if they were old enough to vote. They decided to ask their trusty program for the answer. "What is your age?" the program asked. The student entered their age, hoping for an answer. The program thought for a moment and said, "If your age is 18 or older, you are eligible to vote." The student waited nervously. If

the number entered was 18 or more, the program would announce, "Eligible to vote!" But if it was less, it would respond with a simple, "Not eligible to vote."

INPUT:

The program accepts a single integer input from the user, representing their age.

OUTPUT:

The program outputs one of two possible messages:

- "Eligible to vote." if the age is greater than or equal to 18.
- "Not eligible to vote." if the age is less than 18.

SOLUTION:

```
#include <stdio.h>

int main() {
   int age;

   // Ask the user to enter their age

   scanf("%d", &age);

   // Check voting eligibility
   if (age >= 18) {
      printf("Eligible to vote.\n");
   } else {
      printf("Not eligible to vote.\n");
   }

   return 0;
}
```

TEST CASES:

TEST CASE 1:
INPUT:18
OUTPUT: Eligible to vote.

TEST CASE 2:
INPUT:17
OUTPUT: Not eligible to vote.

TEST CASE 3:
INPUT:25
OUTPUT: Eligible to vote.

TEST CASE 4:
INPUT:16
OUTPUT: Not eligible to vote.

TEST CASE 5:
INPUT:40
OUTPUT: Eligible to vote.

TEST CASE 6:
INPUT:0
OUTPUT: Not eligible to vote.

Explanation for Test Cases:

1. **18:** Age is equal to 18, so eligible to vote.
2. **17:** Age is less than 18, so not eligible.
3. **25:** Age is greater than 18, so eligible to vote.
4. **16:** Age is less than 18, so not eligible.
5. **40:** Age is greater than 18, so eligible to vote.
6. **0:** Age is less than 18, so not eligible.

PROBLEM STATEMENT 3:

Write a C program to determine the roots of a quadratic equation of the form ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0. The program should:

1. Accept the coefficients a, b, and c as input.
2. Check if the discriminant (D=b2−4acD = b^2 - 4acD=b2−4ac) is:
    - Positive (real and distinct roots),
    - Zero (real and equal roots), or
    - Negative (complex roots).
3. Print the appropriate message along with the roots.

INPUT:
The program accepts three floating-point numbers as input, representing the coefficients a, b, and c of the quadratic equation ax2+bx+c=0ax^2 + bx + c = 0ax2+bx+c=0.
OUTPUT:
Depending on the discriminant, the output will indicate the type of roots and their values.

- The roots are real and distinct.
- The roots are real and equal.
- The roots are complex.

SOLUTION:

```
#include <stdio.h>
#include <math.h> // For sqrt() function

int main() {
   float a, b, c, discriminant, root1, root2, realPart, imaginaryPart;

   // Accept coefficients a, b, and c
   printf("Enter coefficients a, b, and c: ");
   scanf("%f %f %f", &a, &b, &c);

   // Calculate the discriminant
   discriminant = b * b - 4 * a * c;

   // Check the nature of the discriminant
   if (discriminant > 0) {
      // Real and distinct roots
      root1 = (-b + sqrt(discriminant)) / (2 * a);
      root2 = (-b - sqrt(discriminant)) / (2 * a);
      printf("The roots are real and distinct.\n");
      printf("Root 1: %.2f\n", root1);
      printf("Root 2: %.2f\n", root2);
   } else if (discriminant == 0) {
      // Real and equal roots
      root1 = -b / (2 * a);
      printf("The roots are real and equal.\n");
      printf("Root: %.2f\n", root1);
   } else {
      // Complex roots
      realPart = -b / (2 * a);
      imaginaryPart = sqrt(-discriminant) / (2 * a);
      printf("The roots are complex.\n");
      printf("Root 1: %.2f + %.2fi\n", realPart, imaginaryPart);
      printf("Root 2: %.2f - %.2fi\n", realPart, imaginaryPart);
   }

   return 0;
}
```

TEST CASES:
TEST CASE 1:
INPUT: Enter coefficients a, b, and c: 1 -3 2
OUTPUT: The roots are real and distinct.
Root 1: 2.00
Root 2: 1.00
TEST CASE 2:
INPUT: Enter coefficients a, b, and c: 1 -2 1

OUTPUT: The roots are real and equal.
Root: 1.00
TEST CASE 3:
INPUT: Enter coefficients a, b, and c: 1 2 5
OUTPUT: The roots are complex.
Root 1: -1.00 + 2.00i
Root 2: -1.00 - 2.00i
TEST CASE4:
INPUT: Enter coefficients a, b, and c: 2 -5 3
OUTPUT: The roots are real and distinct.
Root 1: 1.50
Root 2: 1.00
TEST CASE 5:
INPUT: Enter coefficients a, b, and c: 1 4 4
OUTPUT: The roots are real and equal.
Root: -2.00
TEST CASE 6:
INPUT: Enter coefficients a, b, and c: 1 1 1
OUTPUT: The roots are complex.
Root 1: -0.50 + 0.87i
Root 2: -0.50 - 0.87i

Explanation for Test Cases:

1. **Case 1:** D=1D = 1D=1, two real and distinct roots.
2. **Case 2:** D=0D = 0D=0, one real and equal root.
3. **Case 3:** D=−16D = -16D=−16, two complex roots.
4. **Case 4:** D=1D = 1D=1, two real and distinct roots.
5. **Case 5:** D=0D = 0D=0, one real and equal root.
6. **Case 6:** D=−3D = -3D=−3, two complex roots.

PROBLEM STATEMENT 4:
One day, a curious student Manu encountered a number that seemed special, and they wondered if it was an Armstrong number. They decided to ask their magical program. The program explained, "An Armstrong number is one where the sum of its digits raised to the power of the number of digits equals the number itself." Excited, the student typed in the number, and the program started calculating. If the sum of the digits, each raised to the power of how many digits there were, matched the number, it would declare, "This is an Armstrong number!" If not, it would say, "This is not an Armstrong number."
INPUT:
The program accepts a single integer as input, which represents the number to be checked.
OUTPUT:
The program outputs a message indicating whether the number is an Armstrong number or not.
SOLUTION:

```
#include <stdio.h>
#include <math.h> // For pow() function

int main() {
   int number, originalNumber, remainder, n = 0;
   double result = 0.0;

   // Accept input from the user
   printf("Enter a number: ");
   scanf("%d", &number);

   originalNumber = number;

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

   originalNumber = number;
```

```
    // Compute the sum of powers of digits
    while (originalNumber != 0) {
        remainder = originalNumber % 10;
        result += pow(remainder, n);
        originalNumber /= 10;
    }

    // Check if the number is an Armstrong number
    if ((int)result == number) {
        printf("%d is an Armstrong number.\n", number);
    } else {
        printf("%d is NOT an Armstrong number.\n", number);
    }

    return 0;
}
```

TEST CASES:
TEST CASE 1:
INPUT: Enter a number: 153
OUTPUT: 153 is an Armstrong number.
TEST CASE 2:
INPUT: Enter a number: 9474
OUTPUT: 9474 is an Armstrong number.
TEST CASE 3:
INPUT: Enter a number: 123
OUTPUT: 123 is NOT an Armstrong number.
TEST CASE 4:
INPUT: Enter a number: 370
OUTPUT: 370 is an Armstrong number.
TEST CASE5:
INPUT: Enter a number: 9475
OUTPUT: 9475 is NOT an Armstrong number.
TEST CASE 6:
INPUT: Enter a number: 407
OUTPUT: 407 is an Armstrong number.

Explanation for Test Cases:

- Armstrong numbers are numbers where the sum of the digits raised to the power of the number of digits equals the number itself.
- In the test cases:
    - 153, 9474, 370, 407 are Armstrong numbers, as the sum of their digits raised to the respective power equals the number.
    - 123, 9475 are not Armstrong numbers, as the sum of their digits raised to the respective power does not equal the number itself.

Problem 5: Parking Fee Calculator

Title: Calculating Parking Charges

A parking lot charges a fee based on the number of hours a car is parked. The first 3 hours are free, and each additional hour costs ₹50. Write a program using a **do-while loop** to calculate the fee for multiple cars until the user stops.

Sample Test Cases

1. **Input:**

2 // Number of Cars = 2
5 // Car 1 Parking Hours = 5
2 // Car 2 Parking Hours = 2

Output:

```
100
0
```

2. **Input:**
1
6
Output:150

Test Cases Table

Test Case	Cars	Parking Hours	Expected Fee
1	2	5 2 n	100 0
2	1	6	150
3	3	3 4 1 j	0 50 0
4	1	8 m	250
5	2	2 5 o	0 100

Solution in C

```
#include <stdio.h>

int main() {
    int cars, hours, fee;
    char cont;

    do {
        scanf("%d", &cars);

        for (int i = 1; i <= cars; i++) {

            scanf("%d", &hours);

            if (hours <= 3) {
                fee = 0;
            } else {
                fee = (hours - 3) * 50;
            }

            printf("%d",fee);
        }

        printf("Calculate fee for another set of cars? (y/n): ");
        scanf(" %c", &cont);

    } while (cont == 'y' || cont == 'Y');
```

```
    return 0;
}
```

Bloom's Taxonomy: Application

Difficulty Level: Easy

Problem 6: Cinema Seating Arrangement

Title: Display Theatre Seating

A cinema hall wants to display its seating chart. Each row has 5 seats. Write a program to take the number of rows as input and display the seats as row numbers followed by seat numbers.

Sample Test Cases

1. **Input:** 2
Output:
```
Row 1: Seat 1 Seat 2 Seat 3 Seat 4 Seat 5
Row 2: Seat 1 Seat 2 Seat 3 Seat 4 Seat 5
```

2. **Input:** 3
Output:
```
Row 1: Seat 1 Seat 2 Seat 3 Seat 4 Seat 5
Row 2: Seat 1 Seat 2 Seat 3 Seat 4 Seat 5
Row 3: Seat 1 Seat 2 Seat 3 Seat 4 Seat 5
```

Test Cases Table

Test Case	Rows	Expected Output Description
1	1	`Seat 1 Seat 2 Seat 3 Seat 4 Seat 5`
2	2	`Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5`
3	3	`Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5`
4	4	`Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5`
5	5	`Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5` `Seat 1 Seat 2 Seat 3 Seat 4 Seat 5`

		Seat 1 Seat 2 Seat 3 Seat 4 Seat 5 Seat 1 Seat 2 Seat 3 Seat 4 Seat 5

Solution in C

```
#include <stdio.h>

int main() {
    int rows;
  //  printf("Enter the number of rows: ");
    scanf("%d", &rows);

    for (int i = 1; i <= rows; i++) {
        printf("Row %d: ", i);
        for (int j = 1; j <= 5; j++) {
            printf("Seat %d ", j);
        }
        printf("\n");
    }

    return 0;
}
```

Bloom's Taxonomy: Understanding

Difficulty Level: Easy

Problem 7: Countdown Timer for Fitness Routine

Title: Fitness Timer for Workouts

Create a countdown timer for a fitness workout that counts down from a user-defined number of seconds and displays each second in real-time.

Sample Test Cases

Sample Input: 5
Output:

```
5
4
3
2
1
```

Sample Input: 3
Output:

```
3

2

1
```

Test Cases Table

Test Case	Input Seconds	Expected Output Sequence
1	5	5 4 3 2 1
2	3	3 2 1
3	10	10 9 8 7 6 5 4 3 2 1
4	1	1
5	7	7 6 5 4 3 2 1

Solution in C

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

int main() {
    int seconds;
    //printf("Enter countdown time in seconds: ");
    scanf("%d", &seconds);

    while (seconds > 0) {
        printf("%d\n", seconds);
        sleep(1);
        seconds--;
    }


    return 0;
}
```

Problem 8: Finding the Sum of Digits Using Recursion

Title: Recursive Sum of Digits

Given an integer N, write a recursive function to find the sum of its digits. For example, if N=1234 the sum of digits is 1+2+3+4=10.

Sample Test Cases

1. **Input:**
Number = 1234
Output:10

2. **Input:**
Number = 1023
Output:6

Test Cases Table

Test Case	Input Number	Expected Output
1	1234	10
2	1023	6
3	7	7
4	999	27
5	456	15

Solution in C

```
#include <stdio.h>

int sumOfDigits(int n) {
    if (n == 0)
        return 0;
    return (n % 10) + sumOfDigits(n / 10);
}

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("%d\n", sumOfDigits(num));
    return 0;
}
```

Bloom's Taxonomy: Application

Difficulty Level: Easy

Q9. Problem Statement
Mr. Ramesh, an encoder, needs assistance in developing a C program to generate a password based on the addition of the differences between successive elements of a user-input array. As a programmer, you are tasked with helping Mr. Ramesh develop this program.

Question:
Design a C program that takes input from the user for a 5-element array and generates a password based on the addition of the differences between successive elements (result should be positive integer, if negative then convert it into positive integer) **using pointer**. The password is addition of cube of each digit of the result. The program should perform the following steps:

1. The user inputs 5 integers to populate the array.
2. Calculate the differences between successive elements of the array.
3. Calculate the sum of these differences. If calculated sum is negative then convert it into positive number.
4. Generate a password based on the sum obtained in step 3, which is addition of cube of each digit of the sum (in step 3).

Use appropriate data types and functions to implement the required functionality.

Program template is as follows:

```
// Function to calculate the sum of differences between successive elements
int calculatePassword(int arr[], int size) {



}

int main() {


   // Calculate the password
   int password = calculatePassword(arr, 5);

return 0;
   }
```

Input:
5 9 11 9 5 // Single line input consists of 5 elements.

Output Format:
0 // Calculated password

Constraints:
Range of Integer
Solution: (in required language)

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

// Function to calculate the sum of differences between successive elements
int calculatePassword(int arr[], int size) {
   int sum = 0, i, pass=0, sumT, digR;
   for (i = 0; i < size - 1; i++) {
      sum += *(arr+i+1) - *(arr+i);
   }
   if (sum<0)
   sum=sum*-1;
   sumT=sum;
   while(sumT>0){
                                                                    digR=sumT%10;
                                                                    sumT=sumT/10;
                                                                    pass+=pow(digR, 3);
                                                                    }

   return pass;
}

int main() {
   // Declare an array to store 5 integers
   int arr[5], i;

   // Prompt the user to input 5 integers
   //printf("Enter 5 integers to generate the password:\n");
   for (i = 0; i < 5; i++) {
      //printf("Enter integer %d: ", i + 1);
      scanf("%d", arr+i);
   }

   // Calculate the password
   int password = calculatePassword(arr, 5);

   // Display the generated password
   printf("%d", password);
```

```
    return 0;
}
```

Sample Testcase 1
Input: 23 45 43 67 43
Output: 8
Sample Testcase 2
Input: 23 32 34 56 78
Output: 250

Hidden Testcase 1 - (Easy) Weightage 10%
Input: 23 45 4 67 2
Output: 9
Hidden Testcase 2 - (Easy) Weightage 10%
Input: 34 5 67 8 90
Output: 341
Hidden Testcase 3 - (Medium) Weightage 15%
Input: 34 56 4 0 23
Output: 2
Hidden Testcase 4 - (Medium) Weightage 15%
Input: 23 43 45 67 8
Output: 126
Hidden Testcase 5 - (Hard) Weightage 25%
Input: 34 5 6 7 8
Output: 224
Hidden Testcase 6 - (Hard) Weightage 25%
Input: 23 4 34 56 78
Output: 250

Q10. Problem Statement
Write a C program to reverse an array using pointers. Implement a function that takes a pointer to the array and its size as arguments and reverses the array in place.
Solution:

```
#include <stdio.h>

// Function to reverse the array using pointers
void reverseArray(int *arr, int size) {
    int *start = arr;          // Pointer to the beginning of the array
    int *end = arr + size - 1; // Pointer to the end of the array
    int temp;

    while (start < end) {
        // Swap the values pointed by start and end
        temp = *start;
        *start = *end;
        *end = temp;

        // Move the pointers towards the middle
        start++;
        end--;
    }
}

int main() {
    int n;


    scanf("%d", &n);

    int arr[n];
```

```
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    reverseArray(arr, n);

    printf("Reversed array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Test Cases:
Test Case 1
Input:
5
1 2 3 4 5

Output:

Reversed array: 5 4 3 2 1

Test Case 2
Input:
4
10 20 30 40
Output:

Reversed array: 40 30 20 10
Test Case 3
Input:
6
6 7 8 9 10 11
Output:

Reversed array: 11 10 9 8 7 6

Test Case 4
Input:
3
100 200 300
Output:
Reversed array: 300 200 100
Test Case 5
Input:
1
5
Output:
Reversed array: 5
Test Case 6
Input:
7
3 5 7 9 11 13 15
Output:
Reversed array: 15 13 11 9 7 5 3

Q11. Problem Statement

Ten numbers are entered from the keyboard into an array. Write a program to find out how many of them are positive and how many are negative.

Input Format:
The input consists of ten integer numbers as array elements.
Output Format:
The output prints count of positive numbers in first line and count of negative numbers in second line.
Code Constraints:
NA
Solution:

```
#include<stdio.h>
int main() {
int i,a[10],n,pos=0,neg=0;
for(i=0;i<10;i++)
{
scanf("%d",&a[i]);
if(a[i]<0)
{
neg++;
}
else
{
pos++;
}
}
printf("%d\n%d",pos,neg);
   return 0;
}
```

Sample Testcase 1
Input:
2 4 5 -9 -10 23 4 -12 5 34
Output:
7
3

Sample Testcase 2
Input:
0 6 8 -9 -5 -33 23 43 -21 20

Output:
6
4

Hidden Testcase 1 - (Easy) Weightage 10%
Input:
0 0 0 0 0 0 0 0 0 0
Output:
10
0

Hidden Testcase 2 - (Easy) Weightage 10%
Input:
3 -9 -6 -8 -10 -12 -34 -98 -2000 -5000
Output:
0
10

Hidden Testcase 3 - (Medium) Weightage 15%
Input:
1 2 3 4 5 6 7 8 9 4
Output:
10

0

Hidden Testcase 4 - (Medium) Weightage 15%
Input:
12 765 -50 20 -312 30 -3456 2 3 2
Output:
7
3
Hidden Testcase 5 - (Hard) Weightage 25%
Input:
22 33 99 0 0 0 54 76 87 0
Output:
10
0

Hidden Testcase 6 - (Hard) Weightage 25%
Input:
10 0 9 -5 -21 -32 -1000 0 23 0
Output:
5
5