1a. Program for stack-based buffer overflow:

```
#include<stdio.h>
void secretfunction( )
{
    printf(" ** Password ** \n");
    printf(" Password is :querty123\n");
}
void doit( )
{
    char buffer[5];
    printf(" Enter some te~~x~~t: \n");
    scanf(" %s", buffer);
    printf(" You entered : %s\n", buffer);
}
int main ( )
{
    doit();
    return 0;
}
```

Output:

1b. Program for Heap-based Buffer overflow:

```
#include<stdio.h>
#include <string.h>
int main ()
{
    char *buffer = (char *) malloc (10);
    strcpy (buffer , "Hello world! ");
    printf( "%s", buffer);
    free( buffer);
    return 0;
}
```

Output:

1c. Write a program for Authetication system

```
<html>
<head>
<title> User Authentication </title>
<script>
    function authentication(){
        const username = "Tuvvishree";
        const password = "Tuvvishree";
        const input_username = document.getelementby
        id ("username").value;
        const input_password = document.getelementby
        id("password").value;
        if(input_username === username && input_pass
        word === password){
            alert("Authentication successful");
        }
        else{
            alert(" Invalid username or password");
        }
</script>
</head>
(</html>)
<body>
<form>
    < Label for = "username"> Username : </Label>
    <input type = "text" id = "username" name = "usern
    <br><br>
    <Label for= "password"> Password </Label>
    <input type= "password" id = "password" name =
    "password"> <br>
    <input type = "button" value = "submit" onclick
```

```
        "authentication()">
</form>
</body>
</html>
```

Output:

2a. Write a program that has default fail_safe Mechani

```
#include <stdio.h>
#define Default-value 0
int main()
{
        int num = Default_value;
        printf(" Enter an integer: ");
        scanf(" %d ", &num);
        if(num <0 || num >100)
        {
             num = Default_value;
             printf(" Input is outside the valid range using
             default value: %d \n", num);
        }
        printf(" Input value: %d \n", num);
        return 0;
}
```

Output :

2b. Example1 for principle of least Mechanism

```
#include<stdio.h>
int main()
{
    const int MAX_SIZE = 100;
    int age =0;
    Char name[MAX_SIZE];
    printf("Enter your name: ");
    fgets(name, MAX_SIZE, stdin);
    printf(" Enter your age: ");
    scanf("%d", &age);
    printf("Name: %s \n", name);
    printf("Age: %d \n", age);
    return 0;
}
```

Output:

2c. Examplez for principle of Least Mechanism.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    FILE *file;
    file = fopen("data.txt", "r");
    if(file == NULL)
    {
        printf("Error: could not open file\n");
        exit(1);
    }
    char buffer[256];
    fgets(buffer, 256, file);
    fclose(file);
    printf("Data: %s\n", buffer);
    return 0;
}
```

Output:

a (i) Fragile and Robust code for division by zero.

Fragile code

```
#include<stdio.h>
int main()
{
    int x ;
    int y;
    printf(" Enter First number: ");
    scanf("%d ", &x);
    printf(" Enter second number: ");
    scanf("%d", &y);
    int z = x/y ;
    printf(" Result: %d\n", z);
    return ;
}
```

Robust code

```
#include<stdio.h>
int main()
{
    int x;
    int y;
    printf(" Enter First number: ");
    scanf("%d", &x);
    Printf(" Enter second number: ");
    scanf("%d", &y);
    if(y = =0)
    {
        printf(" Error: Cannot divide by zero\n");
        return ;
    z = x/y;
    printf(" Result : %d \n", z);
    return ;
}
```

<u>Output:</u>

3a(ii) Fragile and Robust code for finding average of ar

Fragile code

```
#include<stdio.h>
int main()
{
    int nums[5] = {5, 3, 6, 2, 8};
    int sum = 0;
    int i;
    for(i=0; i<5; i++)
    {
        sum += nums[i];
    }
    int avg = sum/5;
    printf("Average : %d \n", avg);
    return 0;
}
```

Robust code

```
#include <stdio.h>
int main()
{
    int nums[] = {5, 3, 6, 2, 8};
    int count = size of (nums)/ size of (nums[0]);
    int sum = 0;
    for (int i=0; i< count; i++)
    {
        sum + = nums[i];
    }
    double avg = (double) sum/count;
    printf( "Average : %.2f\n", avg);
    return 0;
}
```

rray Output :

3a(iii) Fragile and Robust code for addition of two positi

Fragile code

```
#include<stdio.h>
int main()
{
    int num1, num2, sum;
    printf(" Enter two numbers separated by a space ");
    scanf("%d %d", &num1, &num2);
    sum = num1+num2;
    printf(" The sum of %d and %d is %d\n", num1, num2,
    return 0;
}
```

Robust code

```
#include<stdio.h>
int main()
{
   int num1, num2, sum;
   printf(" Enter two numbers separated by a space: ");
   if(scanf("%d %d", &num1, &num2)≠2)
   {
       printf(" Invalid Input: Please enter two numbers
       separated by a space \n");
       return 1;
   }
   sum = num1 + num2;
   printf(" The sum of %d and %d is %d \n", num1,
   sum);
   return 0;
}
```

Output:

3a (iv) Fragile and Robust code

Fragile code

```
#include<stdio.h>
int main()
{
    int num = 10;
    printf("Num: %d\n", num);
    char *ptr = (char *) &num;
    ptr [0] = 0;
    ptr [1] = 0;
    ptr[2] = 0;
    ptr [3] = 0;
    printf(" Num : %d\n ", num);
    return 0;
}
```

Robust code

```
#include<stdio.h>
int main ()
{
    const int num = 10;
    printf('Num : %d\n ", num);
    num = 20;
    printf(" Num : %d \n", num);
    return 0;
}
```

Output:

3 b (i) Error handling using 'return'

```
#include<stdio.h>
#include <errno.h>
#include<string.h>
int main ()
{
      FILE *file;
      Char *filename = "text.txt";
      file = fopen(filename, "r");
      if(file == NULL)
      {
            printf("Error opening file %s: %s\n", filename,
            strerror(errno));
            return 1;
      }
      fclose (file);
      return 0;
}
```

Output:

3 b (ii) Error handling using global variable

```
#include<stdio.h>
int errno;
void divide (int num, int den)
{   if(den ==0)
    {
        error =1;
        return;
    }
}
int main ()
{
    int num =10,den= 0;
    divide(num, den);
    if(errno== 1)
    {   printf(" Error: Division by zero\n");
        return 1;
    }
    return 0;
}
```

<u>Output</u>:

3b (iii) Error handling using setjmp and longjmp.

```
#include <stdio.h>
#include <setjmp.h>
jmp_buf error_buffer;
void divide (int num, int den)
{
    if(den == 0)
        longjmp(error_buffer,1);
}
int main()
{
    int num=10, den=0;
    int result;
    if(setjmp(error_buffer)==0)
    {
        divide(num, dem);
        result = num/den :
    }
    else
    {
        printf(" Error: Division by zero");
        return 1;
    }
    return 0;
}
```

Output:

3 b(iv) Error handleing using <errno.h> header file.

```
#include <stdio.h>
#include <ermo.h>
int main()
{
    FILE *fp = fopen("Non_Existed_file.txt", "r");
    if (fp == NULL)
    {
        perror("Error opening file");
        printf("Errno = %d \n", errno);
    }
    return 0;
}
```

Output: