WEEK 13

Objective: Explore the basic difference between normal and pointer variables, Arithmetic operations using pointers and passing variables to functions using pointers

Suggested Experiments/Activities:

Tutorial 13: Call by reference, dangling pointers

Lab 13: Simple functions using Call by reference, Dangling pointers.

i) Write a C program to swap two numbers using call by reference.
ii) Demonstrate Dangling pointer problem using a C program.
iii) Write a C program to copy one string into another using pointer.
iv) Write a C program to find no of lowercase, uppercase, digits and other characters using pointers

i) Write a C program to swap two numbers using call by reference.

Aim: To write a C program that swaps two numbers using a function with call by reference, where the actual variables are modified using their memory addresses.

Algorithm

```
1. Start the program.
2. Declare two integer variables a and b.
3. Read the values of a and b from the user.
4. Call a function swap() and pass the addresses of a and b as arguments.
5. Inside the swap() function:
      o Receive the addresses using pointer parameters.
      o Exchange the values by using a temporary variable.
6. After returning from the function, print the swapped values in a and b.
7. Stop the program.
```

Program:

```c
#include <stdio.h>
void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
Void main( )
{
   int a, b;
   printf("Enter two numbers: ");
   scanf("%d %d", &a, &b);
   printf("\n Before swapping: a = %d, b = %d\n", a, b);
   swap(&a, &b);
   printf("After swapping:  a = %d, b = %d\n", a, b);
}
```

Expected Input and Output:

```
Enter two numbers: 2   3
Before swapping: a = 2, b = 3
After swapping:  a = 3, b = 2
```

ii) Demonstrate Dangling pointer problem using a C program.

Aim: To write a C program that demonstrates the **dangling pointer problem**, this occurs when a pointer continues to reference memory that has already been freed or is no longer valid.

Algorithm

1. Start the program.
2. Declare a pointer variable.
3. Dynamically allocate memory using malloc() and assign its address to the pointer.
4. Store some value in the allocated memory and display it.
5. Free the memory using free().
6. After freeing, try to access the pointer again.
7. Show that the pointer becomes a **dangling pointer** because it is pointing to deallocated memory.
8. Stop the program.

Program:

```
#include <stdio.h>
#include <stdlib.h>
void main() {
   int *ptr = NULL;
   ptr = (int *)malloc(sizeof(int));
   if (ptr == NULL)
     {
      printf("Memory allocation failed!\n");
      }
   *ptr = 50;
   printf("Value stored in allocated memory: %d\n", *ptr);
   free(ptr);
   printf("Memory freed.\n");
   printf("Accessing ptr after free (Dangling Pointer): %d\n", *ptr);
   ptr = NULL;
}
```

Expected Input and Output:

```
Value stored in allocated memory: 50
Memory freed.
Accessing ptr after free (Dangling Pointer): 250998
```

iii) Write a C program to copy one string into another using pointer.

Aim : To write a C program that copies one string into another using pointer variables, without using the built-in string library functions.

Algorithm

1. Start the program.
2. Declare two character arrays:
   - str1 for the original string

    - str2 for the copied string
3. Declare two pointer variables that point to str1 and str2.
4. Read the input string into str1.
5. Using a loop, copy each character from the source pointer to the destination pointer until the null character '\0' is reached.
6. Add the null character to the end of the destination string.
7. Print the copied string.
8. Stop the program.

Program:

```
#include <stdio.h>
void main()
{
   char str1[100], str2[100];
   char *p1, *p2;
   printf("Enter a string: ");
   gets(str1);
   p1 = str1;
   p2 = str2;
   while (*p1 != '\0')
   {
      *p2 = *p1;
      p1++;
      p2++;
   }
   *p2 = '\0';
   printf("Copied string: %s\n", str2);
}
```

iv) Write a C program to find no of lowercase, uppercase, digits and other characters using pointers

Aim: To write a C program that counts the number of lowercase letters, uppercase letters, digits, and other characters in a given string using pointer variables.

Algorithm:

- ✓ Start the program.
- ✓ Declare a character array to store the input string.
- ✓ Declare a pointer variable and integer counters for lowercase, uppercase, digits, and other characters.
- ✓ Read the string from the user.
- ✓ Initialize the pointer to point to the beginning of the string.
- ✓ Repeat the following steps until the pointer reaches the null character '\0':
    - If the character is between 'a' and 'z', increment the lowercase counter.
    - Else if the character is between 'A' and 'Z', increment the uppercase counter.
    - Else if the character is between '0' and '9', increment the digit counter.
    - Else if the character is not a newline, increment the "other characters" counter.
    - Move the pointer to the next character.

- ✓ Display the values of all counters.
- ✓ Stop the program.

Program:

```
#include <stdio.h>

int main() {
    char str[200];
    char *p;
    int lower = 0, upper = 0, digit = 0, other = 0;

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    p = str;   // pointer pointing to the start of string

    while (*p != '\0') {
        if (*p >= 'a' && *p <= 'z')
            lower++;
        else if (*p >= 'A' && *p <= 'Z')
            upper++;
        else if (*p >= '0' && *p <= '9')
            digit++;
        else if (*p != '\n')  // avoid counting newline
            other++;

        p++;   // move to next character
    }

    printf("\nLowercase letters : %d\n", lower);
    printf("Uppercase letters : %d\n", upper);
    printf("Digits            : %d\n", digit);
    printf("Other characters  : %d\n", other);

    return 0;
}
```

Expected Input and Output-1:

```
Enter a string: rajani
Lowercase letters : 6
Uppercase letters : 0
Digits            : 0
Other characters  : 0
```

Expected Input and Output-2:

```
Enter a string: Gist123@edu.in
Lowercase letters : 8
Uppercase letters : 1
Digits            : 3
Other characters  : 2
```