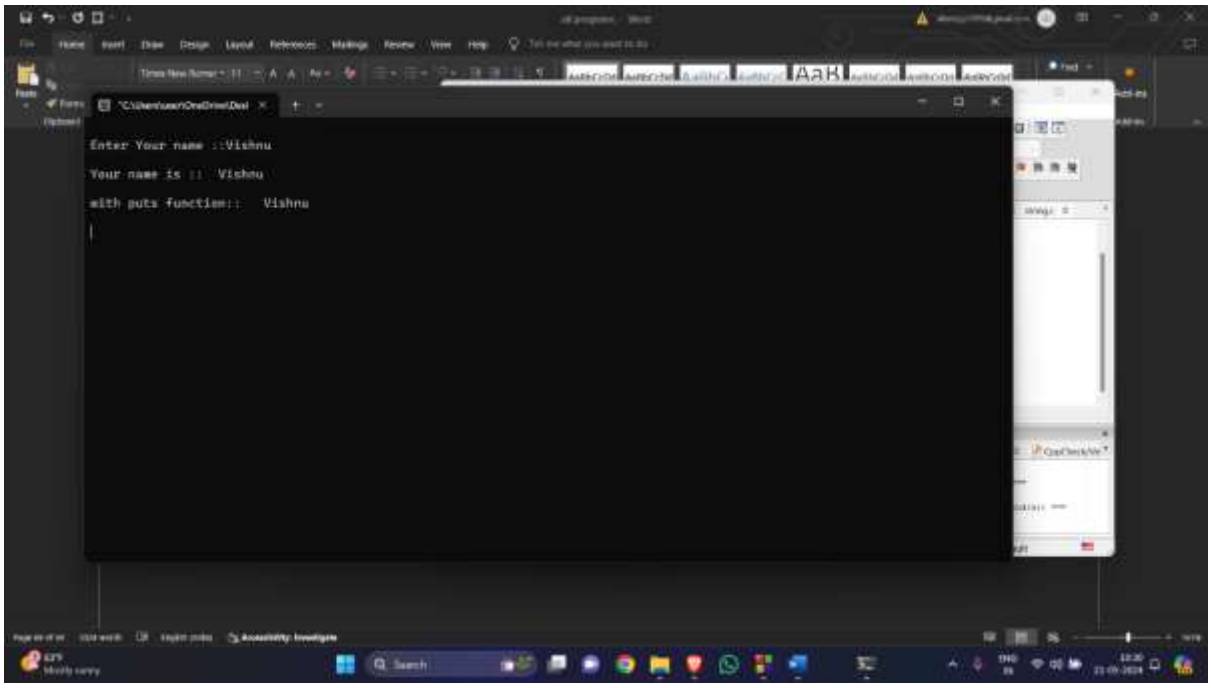
DAY 7

1. String

```
#include <stdio.h>
#include <conio.h>
main()
{
  char name[30];
  printf("\nEnter Your name ::");
  //scanf ("%s", name);
  fgets(name, sizeof(name), stdin);
  printf("\nYour name is ::  %s", name);
  printf("\nwith puts function::   ");
  puts(name);
  getch();
}
```

OUTPUT:



2. Doubly Linked list

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

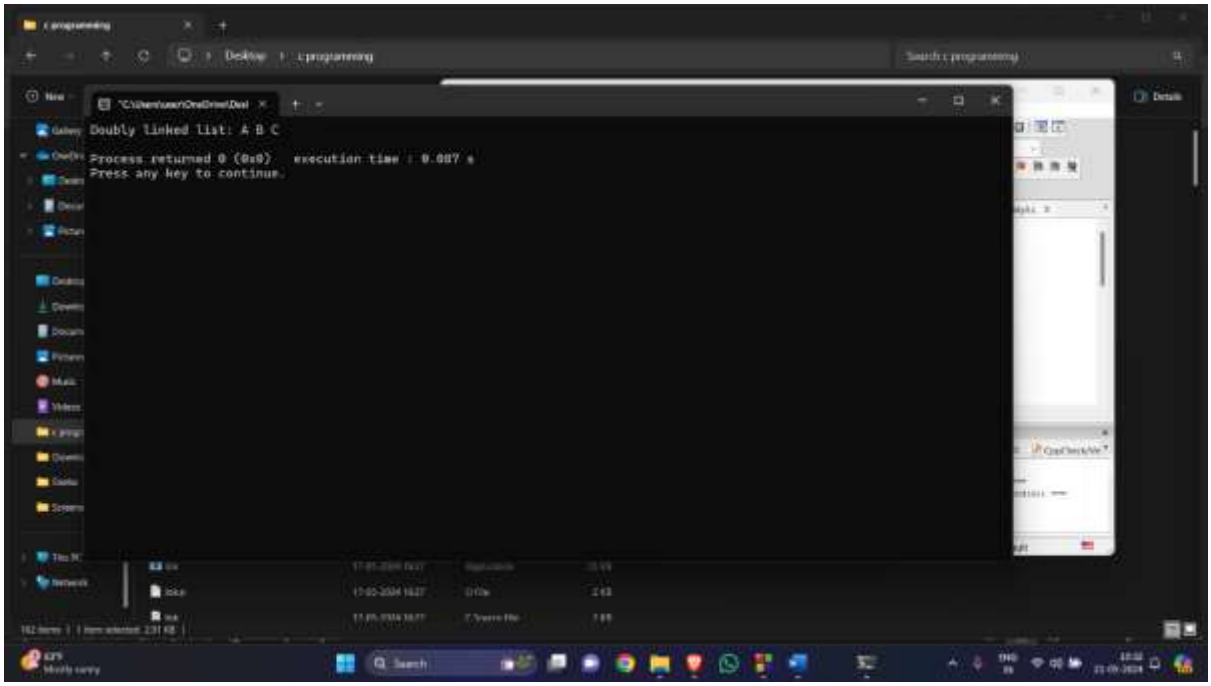
```
struct node {
    char info;
    struct node *lst;
    struct node *rst;
};

int main() {

    struct node *head = NULL;
    struct node *one = NULL;
    struct node *two = NULL;
    struct node *three = NULL;
    one = (struct node *)malloc(sizeof(struct node));
    two = (struct node *)malloc(sizeof(struct node));
    three = (struct node *)malloc(sizeof(struct node));
    one->info = 'A';
    two->info = 'B';
    three->info = 'C';
    one->lst = NULL;
    one->rst = two;
    two->lst = one;
    two->rst = three;
    three->lst = two;
    three->rst = NULL;
    head = one;
    printf("Doubly linked list: ");
    struct node *current = head;
    while (current != NULL) {
        printf("%c ", current->info);
        current = current->rst;
    }
```

```
    printf("\n");
    free(one);
    free(two);
    free(three);
    return 0;
}
```

OUTPUT:



3. Infix ,Prefix,Postfix conversion

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

struct Node {
    char value;
    struct Node *left;
    struct Node *right;
};

struct Node *root = NULL;

struct Node* insert() {
```

```
    char data[10];
    struct Node *nn = (struct Node *)malloc(sizeof(struct Node));

    printf("Enter Data (operator or operand, or -1 for no node): ");
    scanf("%s", data);

    if (data[0] == '-' && data[1] == '1') {
        free(nn);
        return NULL;
    }

    nn->value = data[0];
    printf("Enter Left Node of %c\n", nn->value);
    nn->left = insert();
    printf("Enter Right Node of %c\n", nn->value);
    nn->right = insert();

    return nn;
}

void preorder(struct Node *root) {
    if (root == NULL) {
        return;
    }
    printf("%c ", root->value);
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct Node *root) {
    if (root == NULL) {
        return;
```

```
    }
    inorder(root->left);
    printf("%c ", root->value);
    inorder(root->right);
}
void postorder(struct Node *root) {
    if (root == NULL) {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%c ", root->value);
}
int evaluate(struct Node *root) {
    if (root == NULL) {
        return 0;
    }
    if (root->left == NULL && root->right == NULL) {
        return root->value - '0';
    }
    int left_val = evaluate(root->left);
    int right_val = evaluate(root->right);
    switch (root->value) {
        case '+': return left_val + right_val;
        case '-': return left_val - right_val;
        case '*': return left_val * right_val;
        case '/': return left_val / right_val;
    }
    return 0;
}
int main() {
    printf("Building the expression tree...\n");
```

```
    root = insert();
    printf("PreOrder Traversal: ");
    preorder(root);
    printf("\n");
    printf("InOrder Traversal: ");
    inorder(root);
    printf("\n");
    printf("PostOrder Traversal: ");
    postorder(root);
    printf("\n");
    printf("Evaluating the expression tree...\n");
    int result = evaluate(root);
    printf("Result of the expression: %d\n", result);
    return 0;
}
```

OUTPUT: