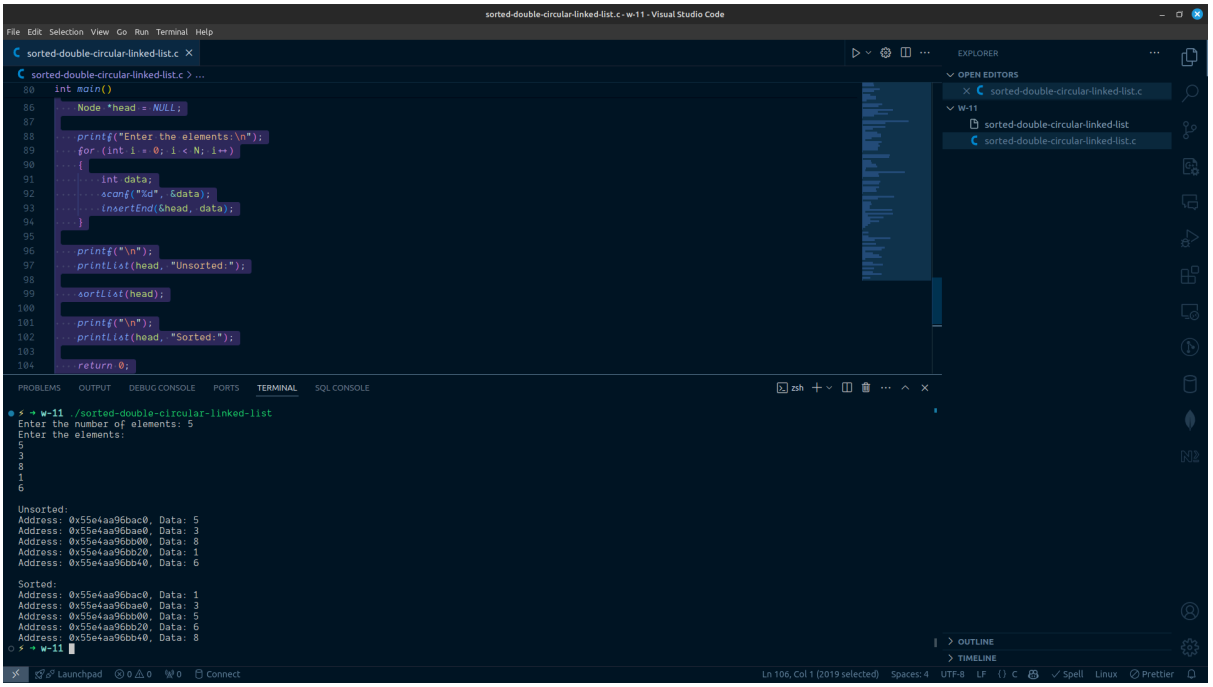
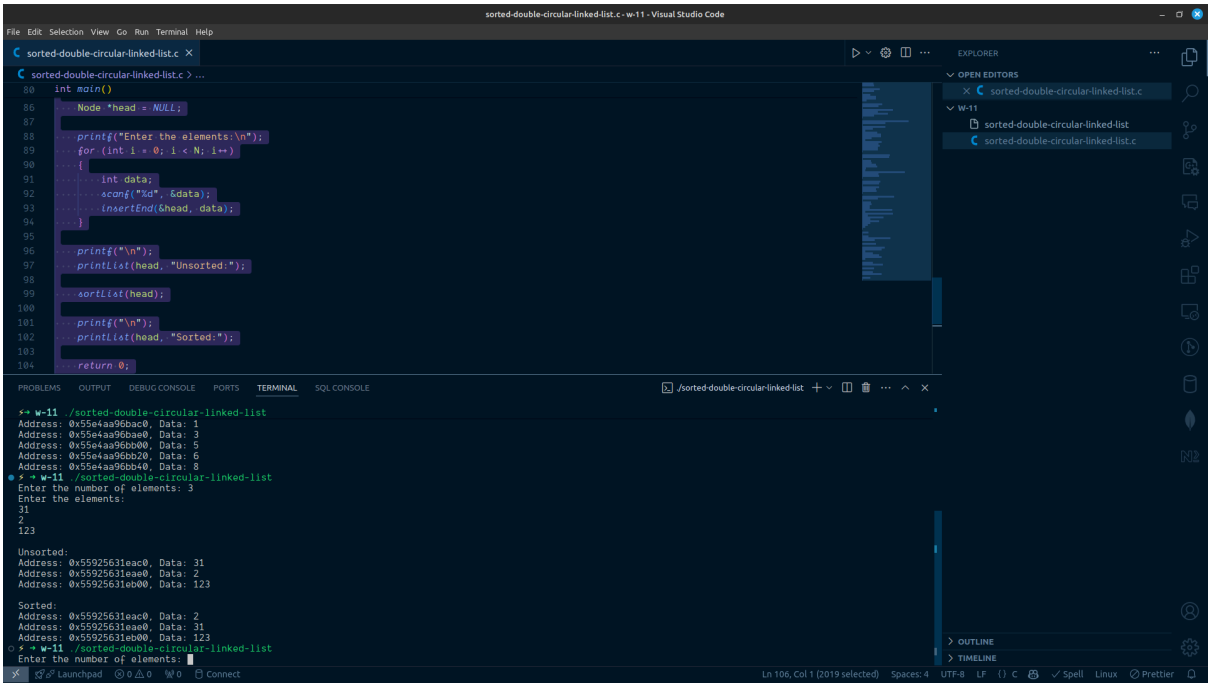
Nama : Arif Widianto
NIM : 1203230051
Kelas : IF 03
Tugas : Praktikum Sort Data Circular Double Linked List

# TUGAS PRAKTIKUM

1. Sort Data Circular Double Linked List
   a. Output



b. Source Code

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

// Define the node structure
typedef struct Node
{
    int data;
    struct Node *next;
    struct Node *prev;
} Node;

// Function to create a new node
Node *createNode(int data)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}

// Function to insert a node at the end of the circular doubly linked
list
void insertEnd(Node **head, int data)
{
    Node *newNode = createNode(data);
    if (!(*head))
    {
        *head = newNode;
        (*head)->next = *head;
        (*head)->prev = *head;
        return;
    }
    Node *last = (*head)->prev;
    newNode->next = *head;
    (*head)->prev = newNode;
    newNode->prev = last;
    last->next = newNode;
}

// Function to print the list with addresses
void printList(Node *head, const char *msg)
{
    if (!head)
        return;
    Node *temp = head;
```

```
    printf("%s\n", msg);
    do
    {
        printf("Address: %p, Data: %d\n", (void *)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
}

// Function to swap two nodes
void swap(Node *a, Node *b)
{
    int tempData = a->data;
    a->data = b->data;
    b->data = tempData;
}

// Function to sort the circular doubly linked list
void sortList(Node *head)
{
    if (!head)
        return;
    Node *i;
    Node *j;
    for (i = head; i->next != head; i = i->next)
    {
        for (j = i->next; j != head; j = j->next)
        {
            if (i->data > j->data)
            {
                swap(i, j);
            }
        }
    }
}

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

    Node *head = NULL;
    printf("Enter the elements:\n");
```

```
    for (int i = 0; i < N; i++)
    {
        int data;
        scanf("%d", &data);
        insertEnd(&head, data);
    }

    printf("\n");
    printList(head, "Unsorted:");

    sortList(head);

    printf("\n");
    printList(head, "Sorted:");

    return 0;
}
```

c. Penjelasan Kode ( pada tiap struct, function, procedure )

1. **Struct Node**

```
typedef struct Node {
    int data;
    struct Node *next;
    struct Node *prev;
} Node;
```

**Struct Node** mendefinisikan sebuah node dalam *linked list* ganda sirkular, yang memiliki tiga anggota: data (menyimpan nilai data), *next* (pointer ke node berikutnya), dan *prev* (pointer ke *node* sebelumnya).

2. **Fungsi createNode**

```
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = newNode->prev = NULL;
    return newNode;
}
```

**Fungsi createNode** bertanggung jawab untuk membuat *node* baru. Fungsi ini menggunakan *malloc* untuk mengalokasikan memori untuk *node* baru dan menginisialisasi nilai data dari *node* tersebut. Pointer *next* dan *prev* diatur menjadi *NULL* karena *node* belum dimasukkan ke dalam *linked list*.

3. **Fungsi insertEnd**

```
void insertEnd(Node** head, int data) {
    Node* newNode = createNode(data);
    if (!(*head)) {
        *head = newNode;
        (*head)->next = *head;
        (*head)->prev = *head;
        return;
    }
    Node* last = (*head)->prev;
    newNode->next = *head;
    (*head)->prev = newNode;
    newNode->prev = last;
    last->next = newNode;
}
```

**Fungsi insertEnd** digunakan untuk menambahkan *node* baru di akhir *linked list*. Jika *linked list* kosong, *node* baru menjadi *node* pertama dan dihubungkan ke dirinya sendiri (sirkular). Jika tidak kosong, *node* baru akan ditempatkan di akhir dan *pointer* diatur agar sesuai dengan struktur *linked list* ganda sirkular.

4. **Fungsi printList**

```
void printList(Node* head, const char* msg) {
    if (!head) return;
    Node* temp = head;
    printf("%s\n", msg);
    do {
        printf("Address: %p, Data: %d\n", (void*)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
}
```

**Fungsi printList** digunakan untuk mencetak seluruh isi *linked list* bersama dengan alamat memori dan data dari setiap *node*. Fungsi ini menerima sebuah pesan sebagai parameter untuk menunjukkan apakah daftar yang dicetak dalam keadaan tersortir atau tidak tersortir.

5. **Fungsi swap**

```
void swap(Node* a, Node* b) {
    int tempData = a->data;
    a->data = b->data;
    b->data = tempData;
}
```

**Fungsi swap** bertugas untuk menukar nilai/data antara dua *node*. Fungsi ini mengambil dua pointer *node* sebagai argumen dan menukar nilai data yang disimpan dalam *node* tersebut.

6. **Fungsi sortList**

```
void sortList(Node* head) {
    if (!head) return;
    Node* i;
    Node* j;
    for (i = head; i->next != head; i = i->next) {
        for (j = i->next; j != head; j = j->next) {
            if (i->data > j->data) {
                swap(i, j);
            }
        }
    }
}
```

**Fungsi sortList** digunakan untuk mengurutkan *linked list* dengan metode *bubble sort*. Fungsi ini melakukan iterasi melalui *node-node* dalam *linked list* dan menukar data di antara *node-node* yang diperlukan hingga semua *node* terurut secara *ascending* berdasarkan nilai data mereka.

7. **Fungsi main**

```
int main() {
    int N;
    printf("Enter the number of elements: ");
    scanf("%d", &N);

    Node* head = NULL;

    printf("Enter the elements:\n");
    for (int i = 0; i < N; i++) {
        int data;
        scanf("%d", &data);
        insertEnd(&head, data);
    }

    printList(head, "Unsorted");
    sortList(head);
    printList(head, "Sorted");

    return 0;
}
```

**Fungsi main** adalah titik masuk (*entry point)* utama program. Program meminta pengguna untuk memasukkan jumlah elemen dan kemudian membaca data untuk setiap elemen tersebut. Data dimasukkan ke dalam linked list menggunakan **insertEnd**. Setelah itu, daftar yang tidak tersortir dicetak menggunakan **printList**. Fungsi **sortList** kemudian digunakan untuk mengurutkan linked list, dan daftar yang telah tersortir dicetak lagi.

Dengan menggunakan *malloc* untuk mengalokasikan memori, program memastikan bahwa alamat *node* tetap konstan. Sorting dilakukan dengan menukar data antar *node*, bukan dengan memindahkan *node*, sehingga alamat memori tidak berubah.