

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

## TUGAS PRAKTIKUM

1. Sort Data Circular Double Linked List
  - a. Output

The screenshot shows the Visual Studio Code interface with a C program in the editor and its execution output in the terminal. The program, `sorted-double-circular-linked-list.c`, defines a circular double-linked list structure. The `main` function prompts the user to enter the number of elements (5) and then the elements themselves (5, 3, 8, 1, 6). It then prints the list in its initial 'Unsorted' state and after a `sortList` operation, showing the elements in ascending order (1, 3, 5, 6, 8). The terminal output includes memory addresses for each node.

```
sorted-double-circular-linked-list.c: ...
80 int main()
81 {
82     Node *head = NULL;
83     printf("Enter the elements:\n");
84     for (int i = 0; i < N; i++)
85     {
86         int data;
87         scanf("%d", &data);
88         insertEnd(&head, data);
89     }
90     printf("\n");
91     printList(head, "Unsorted:");
92     sortList(head);
93     printf("\n");
94     printList(head, "Sorted:");
95     return 0;
96 }
```

Terminal Output:

```
* * W-11 ./sorted-double-circular-linked-list
Enter the number of elements: 5
Enter the elements:
5
3
8
1
6

Unsorted:
Address: 0x5e4aa96bac0, Data: 5
Address: 0x5e4aa96bae0, Data: 3
Address: 0x5e4aa96bb00, Data: 8
Address: 0x5e4aa96bb20, Data: 1
Address: 0x5e4aa96bb40, Data: 6

Sorted:
Address: 0x5e4aa96bac0, Data: 1
Address: 0x5e4aa96bae0, Data: 3
Address: 0x5e4aa96bb00, Data: 5
Address: 0x5e4aa96bb20, Data: 6
Address: 0x5e4aa96bb40, Data: 8
* * W-11
```

This screenshot shows the same C program being executed with a different input: 3 elements (31, 2, 123). The terminal output displays the 'Unsorted' list (31, 2, 123) and the 'Sorted' list (2, 31, 123) after the `sortList` function is called. Memory addresses for the nodes are also printed.

```
sorted-double-circular-linked-list.c: ...
80 int main()
81 {
82     Node *head = NULL;
83     printf("Enter the elements:\n");
84     for (int i = 0; i < N; i++)
85     {
86         int data;
87         scanf("%d", &data);
88         insertEnd(&head, data);
89     }
90     printf("\n");
91     printList(head, "Unsorted:");
92     sortList(head);
93     printf("\n");
94     printList(head, "Sorted:");
95     return 0;
96 }
```

Terminal Output:

```
* * W-11 ./sorted-double-circular-linked-list
Enter the number of elements: 3
Enter the elements:
31
2
123

Unsorted:
Address: 0x55925631eac0, Data: 31
Address: 0x55925631eae0, Data: 2
Address: 0x55925631eb00, Data: 123

Sorted:
Address: 0x55925631eac0, Data: 2
Address: 0x55925631eae0, Data: 31
Address: 0x55925631eb00, Data: 123
* * W-11 ./sorted-double-circular-linked-list
Enter the number of elements: 1
```

## b. Source Code

```
#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.