**NAMA** : **BENEDICTA EKA WIJAYANTI**

**NIM** : **1203230059**

**KELAS** : **IF 03 02**

1. source code

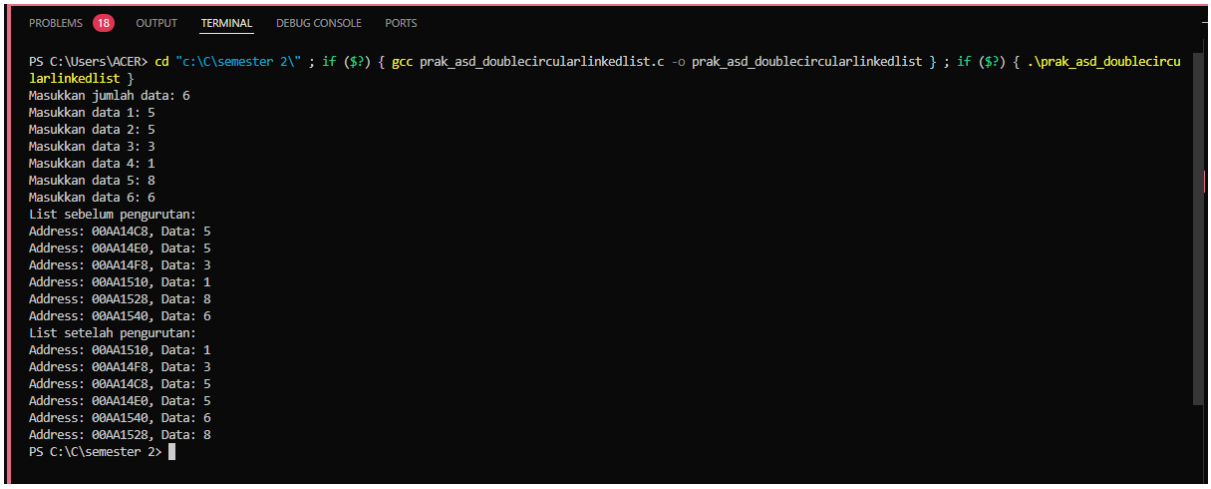
```c
#include <stdio.h>
#include <stdlib.h>

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

Node *head = NULL;
Node *tail = NULL;

Node *createNode(int data)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode)
    {
        printf("Tidak bisa membuat Node!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void addNode(int data)
{
    Node *newNode = createNode(data);
    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    }
    else
    {
        tail->next = newNode;
        newNode->prev = tail;
```

```
        newNode->next = head;
        head->prev = newNode;
        tail = newNode;
    }
}

void printList()
{
    if (head == NULL)
    {
        printf("List kosong.\n");
        return;
    }

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

void swap(Node *node1, Node *node2)
{
    if (node1->next == node2)
    {
        node1->next = node2->next;
        node2->prev = node1->prev;
        node1->prev->next = node2;
        node2->next->prev = node1;
        node2->next = node1;
        node1->prev = node2;
    }
    else if (node2->next == node1)
    {
        node2->next = node1->next;
        node1->prev = node2->prev;
        node2->prev->next = node1;
        node1->next->prev = node2;
        node1->next = node2;
        node2->prev = node1;
    }
    else
    {
        Node *tempNext = node1->next;
        Node *tempPrev = node1->prev;
        node1->next = node2->next;
        node1->prev = node2->prev;
```

```
        node2->next = tempNext;
        node2->prev = tempPrev;
        node1->next->prev = node1;
        node1->prev->next = node1;
        node2->next->prev = node2;
        node2->prev->next = node2;
    }

    if (head == node1)
        head = node2;
    else if (head == node2)
        head = node1;

    if (tail == node1)
        tail = node2;
    else if (tail == node2)
        tail = node1;
}

void sortList()
{
    if (head == NULL || head->next == head)
        return;

    int swapped;
    Node *ptr;

    do
    {
        swapped = 0;
        ptr = head;

        do
        {
            Node *nextNode = ptr->next;
            if (ptr->data > nextNode->data)
            {
                swap(ptr, nextNode);
                swapped = 1;
            }
            ptr = nextNode;
        } while (ptr->next != head);
    } while (swapped);
}

int main()
{
    int b;
```

```c
    printf("Masukkan jumlah data: ");
    scanf("%d", &b);

    for (int i = 0; i < b; i++)
    {
        int data;
        printf("Masukkan data %d: ", i + 1);
        scanf("%d", &data);
        addNode(data);
    }

    printf("List sebelum pengurutan:\n");
    printList();

    sortList();

    printf("List setelah pengurutan:\n");
    printList();

    return 0;
}
```

2. output




3. penjelasan

Kode untuk implementasi double linked list circular. Pertama membuat header dgn include stdio.h dan stdlib.h.

**Mendefinisikan struktur node**

```c
typedef struct Node
{
    int data;
```

```
    struct Node *next;
    struct Node *prev;
} Node;
```

Terdapat 3 elemen pada satu linked list :

- **Data = nilai dari node**
- **Next = pointer ke node selanjurnya**
- **Prev = pointer ke node sebelumnya**

**Mendeklarasi head dan tail**

```
Node *head = NULL;
Node *tail = NULL;
```

Pointer head dan tail digunakan untuk menunjukkan ke node pertama dan terakhir dalam masing-masing list. Kedua pointer dibuat NULL untuk menandai bahwa linked list masih kosong.

**Membuat node baru**

```
Node *createNode(int data)
{
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode)
    {
        printf("Tidak bisa membuat Node!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
```

Fungsi createNode digunakan untuk membuat node baru. Fungsi ini menerima **int data** yang berarti data akan disimpan kedalam node baru, lalu mengembalikan pointer ke node yang baru dibuat.

```
Node *newNode = (Node *)malloc(sizeof(Node));
```

Malloc digunakan untuk mengalokasikan memori sebesar ukuran Node

```
if (!newNode)
    {
        printf("Tidak bisa membuat Node!\n");
        exit(1);
    }
```

Memeriksa apakah newNode adalah NULL. Jika malloc gagal mengalokasikan memori, maka akan mengembalikan NULL. Jika newNode adalah NULL akan mencetak printf dan menghentikan program dgn memanggil exit 1

```
newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
```

Menginisialisasi kode.

- Mengatur data dari newNode dengan nilai yang diberi argument data
- Mengatur next dari newNode dengan NULL
- Mengatur prev dari newNode dengan NULL

NULL menunjukkan bahwa node yang baru dibuat tidak memiliki node sebelumnya/belum terhubung ke list. Lalu mengembalikan node dengan return newNode.

**Menambahkan node ke list**

```
void addNode(int data)
{
    Node *newNode = createNode(data);
    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    }
    else
    {
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
        tail = newNode;
    }
}
```

Fungsi ini tidak mengembalikan nilai karena menggunakan **void**.

```
Node *newNode = createNode(data);
```

Membuat node baru dgn memanggil fungsi createNode. Node baru akan disimpan di pointer newNode

```
if (head == NULL)
    {
        head = newNode;
        tail = newNode;
        newNode->next = newNode;
```

```
        newNode->prev = newNode;
    }
```

Mengecek apakah list kosong, dengan memeriksa apakah head adalah null. Jika list kosong maka

- Head dan tail menunjuk ke newNode
- newNode -> next dan prev akan menunjuk ke newNode (nunjuk diri sendiri) supaya list jadi melingkar

```
else
    {
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
        tail = newNode;
    }
```

Jika list tidak kosong, maka

- tail->next akan menunjuk ke newNode
- prev pada newNode diarahkan ke tail (spy tau node sblm nya)
- next pada newNode diarahkan ke head (menghubungkan node baru Kembali ke awal list // supaya melingkar)
- head prev diatur menunjuk newNode (memperbarui node pertama spy tau node terakhir)
- tail = newNode, artinya jadiin node terbaru sbg node terakhir di list.

**Mencetak list**

```
void printList()
{
    if (head == NULL)
    {
        printf("List kosong.\n");
        return;
    }

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

Diawal dilakukan pengecekan apakah list kosong atau tidak.

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

```
    } while (temp != head);
}
```

Loop do while memastikan loop dijalankan sekali. Lalu mencetak node pada setiap iterasi, disini mencetak alamat dan data dari node yang ditunjuk sm temp. setelah itu temp akan pindah ke node selanjutnya (**temp = temp ->next).** Kondisi ini akan berhenti saat temp menunjuk ke head.

**Menukar node**

```
void swap(Node *node1, Node *node2)
{
    if (node1->next == node2)
    {
        node1->next = node2->next;
        node2->prev = node1->prev;
        node1->prev->next = node2;
        node2->next->prev = node1;
        node2->next = node1;
        node1->prev = node2;
    }
    else if (node2->next == node1)
    {
        node2->next = node1->next;
        node1->prev = node2->prev;
        node2->prev->next = node1;
        node1->next->prev = node2;
        node1->next = node2;
        node2->prev = node1;
    }
    else
    {
        Node *tempNext = node1->next;
        Node *tempPrev = node1->prev;
        node1->next = node2->next;
        node1->prev = node2->prev;
        node2->next = tempNext;
        node2->prev = tempPrev;
        node1->next->prev = node1;
        node1->prev->next = node1;
        node2->next->prev = node2;
        node2->prev->next = node2;
    }

    if (head == node1)
        head = node2;
    else if (head == node2)
        head = node1;

    if (tail == node1)
```

```
        tail = node2;
    else if (tail == node2)
        tail = node1;
}
```

Fungsi ini menerima 2 argumen bertipe node * yaitu node1 dan node2 yang merupakan pointer ke node yang akan ditukar posisinya.

Node2 setelah node1

```
    if (node1->next == node2)
    {
        node1->next = node2->next;
        node2->prev = node1->prev;
        node1->prev->next = node2;
        node2->next->prev = node1;
        node2->next = node1;
        node1->prev = node2;
    }
```

Mengecek apakan node2 berada setelah node1. Lalu melakukan penukaran :

- ngatur next dari node1 untuk nunjuk ke node setelah node2
- prev dari node2 akan nunjuk kenode sebelum node1
- lalu menghubungkan node sebelum node1 ke node2
- menghubungkan node setelah node2 ke node1
- menghubungkan Kembali next dari node2 ke node1
- menghubungkan prev dari node1 ke node2

node1 setelah node2

```
    else if (node2->next == node1)
    {
        node2->next = node1->next;
        node1->prev = node2->prev;
        node2->prev->next = node1;
        node1->next->prev = node2;
        node1->next = node2;
        node2->prev = node1;
    }
```

- next dari node2 nunjuk ke node setelah node1
- prev dari node1 nunjuk ke node sebelum node2
- menghubungkan node sebelum node2 ke node1
- menghubungkan node setelah node1 ke node2
- next dari node1 nunjuk ke node2
- prev dari node2 nunjuk ke node1

jika node tidak bersebelahan

```
    else
    {
```

```
        // next dan prev dari node1 diletakkan di temp
        Node *tempNext = node1->next;
        Node *tempPrev = node1->prev;
        // menukar next dan prev dari node1 dengan node2
        node1->next = node2->next;
        node1->prev = node2->prev;
        node2->next = tempNext;
        node2->prev = tempPrev;
        // memperbaiki pointer yang terhubung ke node1 dan node2
        node1->next->prev = node1;
        node1->prev->next = node1;
        node2->next->prev = node2;
        node2->prev->next = node2;
    }
```

Memperbaiki head dan tail

```
if (head == node1)
        head = node2;
    else if (head == node2)
        head = node1;

    if (tail == node1)
        tail = node2;
    else if (tail == node2)
        tail = node1;
}
```

- jike head menunjuk ke node1, akan diatur utk menunjuk ke node2
- jika tail nunjuk ke node1, diatur utk nunjuk node2
- begitu jg sebaliknya

**Mengurutkan list**

```
void sortList()
{
    if (head == NULL || head->next == head)
        return;

    int swapped;
    Node *ptr;

    do
    {
        swapped = 0;
        ptr = head;

        do
        {
            Node *nextNode = ptr->next;
```

```
            if (ptr->data > nextNode->data)
            {
                swap(ptr, nextNode);
                swapped = 1;
            }
            ptr = nextNode;
        } while (ptr->next != head);
    } while (swapped);
}
```

Fungsi ini untuk mengurutkan linkedlist berdasarkan data yang tersimpan dalam setiap node. Pertama dilakukan pengecekan apakah list kosong (**head == NULL)** atau cuma berisi 1 node (**head -> next == head).** Kalau salah satu kondisinya terpenuhi artinya list sudah terurut.

```
int swapped;
    Node *ptr;
```

Inisialisasi variabel swapped utk nandain terjadi pertukaran selama iterasi. Pointer ptr digunakan utk iterasi node dalam list.

```
do
    {
    } while (swapped);
```

Loop ini akan berjalan selama terjadi pertukaran pada iterasi sebelumnya.

```
swapped = 0;
        ptr = head;
```

swapper akan bernilai 0 pada awal iterasi buat nandain apakah terjadi pertukaran pada iterasi tersebut. Lalu ptr diatur utk menunjuk ke head utk memulai iterasi dari awal list

```
do
        {
        } while (ptr->next != head);
```

Melakukan iterasi setiap node dari dalam list sampai Kembali ke node awal/head

```
Node *nextNode = ptr->next;
```

Mengambil pointer ke node berikutnya setelah ptr

```
            if (ptr->data > nextNode->data)
            {
                swap(ptr, nextNode);
                swapped = 1;
            }
```

Jika data ptr lebih besar dari data nextNode maka akan memanggil fungsi swap utk menukan posisi ptr dan nextNode. Lalu swapped di set 1 utk nandain terjadi pertukaran

```
            ptr = nextNode;
```

mengupdate ptr utk nunjuk ke node selanjutnya

**Fungsi utama**

```
int main()
{
```

```
    int b;
    printf("Masukkan jumlah data: ");
    scanf("%d", &b);

    for (int i = 0; i < b; i++)
    {
        int data;
        printf("Masukkan data %d: ", i + 1);
        scanf("%d", &data);
        addNode(data);
    }

    printf("List sebelum pengurutan:\n");
    printList();

    sortList();

    printf("List setelah pengurutan:\n");
    printList();

    return 0;
}
```

Mendeklarasi variabel b utk menyimpan jumlah data yang dimasukkan oleh user.

```
printf("Masukkan jumlah data: ");
    scanf("%d", &b);
```

meminta user utk memasukkan jumlah data

```
for (int i = 0; i < b; i++)
    {
        int data;
        printf("Masukkan data %d: ", i + 1);
        scanf("%d", &data);
        addNode(data);
    }
```

Loop utk memasukkan data. Pertama mendeklarasi variabel data lalu meminta user memasukkan data sebanyak jumlah yg diinputkan sebelumnya. Lalu memanggil fungsi **addNode** untuk nambahin node baru berisi data yang diinput user

```
printf("List sebelum pengurutan:\n");
    printList();
```

nampilin list dengan memanggil fungsi **printList** spy semua isi linkedlist yang belum diurutkan tercetak.

```
sortList();
```

memanggil fungsi **sortList** untuk mengurutkan linkedlist

```
printf("List setelah pengurutan:\n");
```

```
    printList();

    return 0;
```
menampilkan Kembali list yg sudah diurutkan ke layar lalu melakukan return