Nama: Jonathan Ferdianand Mayon

Kelas: IF 03 03/1203230087

PRAK ASD Double Linked List

Source Code:

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

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

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

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

void printList(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    do {
        printf("%p %d\n", (void*)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
```

```
}

void swapNodes(Node** head, Node* a, Node* b) {
    if (a == b) return;

    Node* aPrev = a->prev;
    Node* aNext = a->next;
    Node* bPrev = b->prev;
    Node* bNext = b->next;

    if (aNext == b) {
        a->next = bNext;
        b->prev = aPrev;
        a->prev = b;
        b->next = a;
        aPrev->next = b;
        bNext->prev = a;
    } else if (bNext == a) {
        b->next = aNext;
        a->prev = bPrev;
        b->prev = a;
        a->next = b;
        bPrev->next = a;
        aNext->prev = b;
    } else {
        a->next = bNext;
        a->prev = bPrev;
        b->next = aNext;
        b->prev = aPrev;
        aPrev->next = b;
        aNext->prev = b;
        bPrev->next = a;
        bNext->prev = a;
    }

    if (*head == a) {
        *head = b;
    } else if (*head == b) {
        *head = a;
    }
}

void sortList(Node** head) {
    if (*head == NULL) return;
```

```
    Node* current = *head;
    Node* index = NULL;
    int swapped;

    do {
        swapped = 0;
        current = *head;

        while (current->next != *head) {
            index = current->next;
            if (current->data > index->data) {
                swapNodes(head, current, index);
                swapped = 1;
            } else {
                current = current->next;
            }
        }
    } while (swapped);
}

int main() {
    Node* head = NULL;
    int N, A;
    printf("Berapa data yang ingin diinput: ");
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &A);
        insertEnd(&head, A);
    }
    printf("Data sebelum diurutkan:\n");
    printList(head);
    sortList(&head);
    printf("Data setelah diurutkan:\n");
    printList(head);

    return 0;
}
```

Output 1:

```
Berapa data yang ingin diinput: 6
5
5
3
8
1
6
Data sebelum diurutkan:
007416D0 5
007416E8 5
00741700 3
00741718 8
00741730 1
00741748 6
Data setelah diurutkan:
00741730 1
00741700 3
007416D0 5
007416E8 5
00741748 6
00741718 8
```

Output 2:

```
Berapa data yang ingin diinput: 4
3
31
2
123
Data sebelum diurutkan:
00B416D0 3
00B416E8 31
00B41700 2
00B41718 123
Data setelah diurutkan:
00B41700 2
00B416D0 3
00B416E8 31
00B41718 123
```

Penjelasan:

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

- **typedef struct Node**: Mendefinisikan sebuah struktur yang disebut **Node**.
- **int data;**: Menyimpan data integer pada node.
- **struct Node* next;**: Pointer ke node berikutnya.
- **struct Node* prev;**: Pointer ke node sebelumnya.
- **} Node;**: Menamakan tipe data ini sebagai **Node**.

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

- **Node* createNode(int data)**: Fungsi untuk membuat node baru dengan data yang diberikan.
- **Node* newNode = (Node*)malloc(sizeof(Node));**: Mengalokasikan memori untuk node baru.
- **newNode->data = data;**: Mengisi node baru dengan data.
- **newNode->next = newNode;** dan **newNode->prev = newNode;**: Menginisialisasi **next** dan **prev** menunjuk pada diri sendiri (untuk circular linked list).
- **return newNode;**: Mengembalikan pointer ke node baru.

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

- **void insertEnd(Node** head, int data)**: Fungsi untuk menambahkan node baru di akhir list.
- **Node* newNode = createNode(data);**: Membuat node baru dengan data yang diberikan.
- **if (*head == NULL)**: Jika list kosong.
    - ***head = newNode;**: Node baru menjadi head.
- **else**: Jika list tidak kosong.
    - **Node* tail = (*head)->prev;**: Menemukan node terakhir.
    - **tail->next = newNode;**: Menghubungkan node terakhir ke node baru.
    - **newNode->prev = tail;**: Mengatur **prev** node baru ke node terakhir.
    - **newNode->next = *head;**: Mengatur **next** node baru ke head.
    - **(*head)->prev = newNode;**: Mengatur **prev** head ke node baru.

```
void printList(Node* head) {
    if (head == NULL) return;
    Node* temp = head;
    do {
        printf("%p %d\n", (void*)temp, temp->data);
        temp = temp->next;
    } while (temp != head);
}
```

- **void printList(Node* head)**: Fungsi untuk mencetak semua node dalam list.
- **if (head == NULL) return;**: Jika list kosong, keluar dari fungsi.
- **Node* temp = head;**: Menginisialisasi pointer sementara ke head.
- **do { ... } while (temp != head);**: Loop untuk mencetak data setiap node hingga kembali ke head.

```
void swapNodes(Node** head, Node* a, Node* b) {
    if (a == b) return;

    Node* aPrev = a->prev;
    Node* aNext = a->next;
    Node* bPrev = b->prev;
    Node* bNext = b->next;

    if (aNext == b) {
        a->next = bNext;
        b->prev = aPrev;
        a->prev = b;
        b->next = a;
        aPrev->next = b;
        bNext->prev = a;
    } else if (bNext == a) {
        b->next = aNext;
        a->prev = bPrev;
        b->prev = a;
        a->next = b;
        bPrev->next = a;
        aNext->prev = b;
    } else {
        a->next = bNext;
        a->prev = bPrev;
        b->next = aNext;
```

```
        b->prev = aPrev;
        aPrev->next = b;
        aNext->prev = b;
        bPrev->next = a;
        bNext->prev = a;
    }

    if (*head == a) {
        *head = b;
    } else if (*head == b) {
        *head = a;
    }
}
```

- **void swapNodes(Node** head, Node* a, Node* b)**: Fungsi untuk menukar dua node dalam list.
- **if (a == b) return;**: Jika node yang sama, tidak perlu menukar.
- **Node* aPrev = a->prev; ...**: Menyimpan pointer **prev** dan **next** dari **a** dan **b**.
- **if (aNext == b) { ... }**: Jika **b** langsung mengikuti **a**.
    - Mengatur ulang pointer **next** dan **prev** untuk menukar **a** dan **b**.
- **else if (bNext == a) { ... }**: Jika **a** langsung mengikuti **b**.
    - Mengatur ulang pointer **next** dan **prev** untuk menukar **b** dan **a**.
- **else { ... }**: Jika **a** dan **b** tidak bersebelahan.
    - Menukar pointer **next** dan **prev** dari **a** dan **b**.
- **if (*head == a) { ... }**: Jika **head** adalah **a**, atur ulang **head** ke **b**.
- **else if (*head == b) { ... }**: Jika **head** adalah **b**, atur ulang **head** ke **a**.

```
void sortList(Node** head) {
    if (*head == NULL) return;

    Node* current = *head;
    Node* index = NULL;
    int swapped;

    do {
        swapped = 0;
        current = *head;
```

```
        while (current->next != *head) {
            index = current->next;
            if (current->data > index->data) {
                swapNodes(head, current, index);
                swapped = 1;
            } else {
                current = current->next;
            }
        }
    } while (swapped);
}
```

- **void sortList(Node** head)**: Fungsi untuk mengurutkan node dalam list menggunakan bubble sort.
- **if (*head == NULL) return;**: Jika list kosong, keluar dari fungsi.
- **Node* current = *head;**: Inisialisasi pointer **current** ke head.
- **Node* index = NULL;**: Pointer untuk node yang akan dibandingkan.
- **int swapped;**: Variabel untuk melacak apakah ada penukaran.
- **do { ... } while (swapped);**: Loop hingga tidak ada penukaran lagi.
- **swapped = 0;**: Reset variabel **swapped**.
- **while (current->next != *head) { ... }**: Loop untuk membandingkan dan menukar node.
    - **index = current->next;**: Inisialisasi pointer **index** ke node berikutnya.
    - **if (current->data > index->data) { ... }**: Jika data **current** lebih besar dari **index**, tukar node.
        - **swapNodes(head, current, index);**: Memanggil fungsi **swapNodes**.
        - **swapped = 1;**: Menandakan bahwa ada penukaran.
    - **else { current = current->next; }**: Jika tidak, lanjutkan ke node berikutnya.

```c
int main() {
    Node* head = NULL;
    int N, A;
    printf("Berapa data yang ingin diinput: ");
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &A);
        insertEnd(&head, A);
    }
    printf("Data sebelum diurutkan:\n");
    printList(head);
    sortList(&head);
    printf("Data setelah diurutkan:\n");
    printList(head);

    return 0;
}
```

- **int main()**: Fungsi utama program.
- **Node* head = NULL;**: Menginisialisasi head list sebagai NULL.
- **int N, A;**: Deklarasi variabel untuk jumlah data dan data yang akan diinput.
- **printf("Berapa data yang ingin diinput: ");**: Mencetak pesan untuk meminta jumlah data.
- **scanf("%d", &N);**: Membaca jumlah data yang akan diinput.
- **for (int i = 0; i < N; i++) { ... }**: Loop untuk membaca dan menambahkan data ke list.
    - **scanf("%d", &A);**: Membaca data.
    - **insertEnd(&head, A);**: Menambahkan data ke akhir list.
- **printf("Data sebelum diurutkan:\n");**: Mencetak pesan sebelum mencetak list yang belum diurutkan.
- **printList(head);**: Mencetak