NAMA : MOCH. JOHAN BINTANG PRATAMA

NIM : 1203230063

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 append(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    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);
    printf("\n");
}

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;
        a->prev = b;
        b->next = a;
        b->prev = aPrev;

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

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

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

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

void sortList(Node** head) {
    if (*head == NULL) return;
    int swapped;
    Node* ptr1;
    Node* lptr = NULL;

    if ((*head)->next == *head) return;

    do {
        swapped = 0;
```

```
        ptr1 = *head;

        do {
            if (ptr1->data > ptr1->next->data) {
                // Tukar posisi node
                swapNodes(head, ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        } while (ptr1->next != *head);

        lptr = ptr1;
    } while (swapped);
}

int main() {
    int N, i, data;
    Node* head = NULL;

    scanf("%d", &N);

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

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

    sortList(&head);

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

    return 0;
}
```

PENJELASAN

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

Include stdio.h ini untuk menambahkan input dan output

Stdlib.h untuk menambahkan operasi pembanding dan konversi

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

struct Node: Definisi struktur bernama Node.

int data;: Anggota struktur untuk menyimpan data integer.

struct Node* next;: Pointer ke node berikutnya dalam list.

struct Node* prev;: Pointer ke node sebelumnya dalam list.

typedef ... Node;: Membuat Node(sementara) untuk struct Node, sehingga kita bisa menggunakan Node sebagai tipe data tanpa menulis struct.

```c
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) {: Mendefinisikan fungsi bernama createNode yang mengembalikan pointer ke Node.

Node* newNode = (Node*)malloc(sizeof(Node));: Mengalokasikan memori untuk node baru menggunakan malloc dengan ukuran sebesar sizeof(Node). Hasilnya adalah pointer ke node baru yang disimpan dalam variabel newNode.

newNode->data = data;: Menetapkan nilai data node baru dengan nilai yang diterima sebagai argumen.

newNode->next = newNode;: Menetapkan pointer next dari node baru ke dirinya sendiri, karena saat ini node baru adalah satu-satunya node dalam list.

newNode->prev = newNode;: Menetapkan pointer prev dari node baru ke dirinya sendiri, karena saat ini node baru adalah satu-satunya node dalam list.

return newNode;: Mengembalikan pointer ke node baru yang telah dibuat.

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

void append(Node** head, int data) {: Mendefinisikan fungsi bernama append yang mengambil pointer ke pointer kepala head dan data yang akan ditambahkan ke node baru.

Node* newNode = createNode(data);: Membuat node baru dengan data yang diberikan menggunakan fungsi createNode.

if (*head == NULL) { ... }: Memeriksa apakah linked list kosong. Jika ya, node baru dijadikan sebagai node pertama dan head diarahkan ke node baru.

Node* tail = (*head)->prev;: Menemukan node terakhir dalam linked list, yang memiliki pointer prev mengarah ke node terakhir.

tail->next = newNode;: Menghubungkan node terakhir dengan node baru dengan menetapkan pointer next dari node terakhir ke node baru.

newNode->prev = tail;: Menetapkan pointer prev dari node baru ke node terakhir, menunjukkan bahwa node baru sekarang adalah node sebelumnya dalam linked list.

newNode->next = *head;: Menetapkan pointer next dari node baru ke node pertama dalam linked list.

(*head)->prev = newNode;: Menghubungkan node pertama dengan node baru dengan menetapkan pointer prev dari node pertama 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);
    printf("\n");
}
```

void printList(Node* head) {: Mendefinisikan fungsi printList yang mengambil pointer ke node kepala head sebagai argumen.

if (head == NULL) return;: Memeriksa apakah linked list kosong (jika head adalah NULL). Jika ya, fungsi langsung mengembalikan tanpa melakukan apa-apa.

Node* temp = head;: Mendefinisikan pointer temp dan menginisialisasinya dengan head untuk mulai iterasi dari node pertama.

do { ... } while (temp != head);: Menggunakan loop do-while untuk memastikan bahwa setidaknya satu iterasi dilakukan (berguna karena list melingkar). Loop akan terus berlanjut hingga kembali ke node kepala.

printf("(%p %d)\n", (void*)temp, temp->data);: Mencetak alamat memori (temp) dan data yang disimpan dalam node (temp->data). (void*)temp digunakan untuk meng-cast pointer temp ke tipe void* agar bisa dicetak sebagai alamat.

temp = temp->next;: Menggerakkan pointer temp ke node berikutnya dalam list.

printf("\n");: Mencetak baris baru setelah seluruh list telah dicetak.

```
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;
        a->prev = b;
        b->next = a;
        b->prev = aPrev;

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

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

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

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

Initinya fungsi ini memastikan secara keseluruhan bahwa node a dan b bertukar posisi dalam list.

```
void sortList(Node** head) {
    if (*head == NULL) return;
    int swapped;
    Node* ptr1;
    Node* lptr = NULL;

    if ((*head)->next == *head) return;

    do {
        swapped = 0;
        ptr1 = *head;

        do {
            if (ptr1->data > ptr1->next->data) {
                // Tukar posisi node
                swapNodes(head, ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        } while (ptr1->next != *head);

        lptr = ptr1;
    } while (swapped);
}
```

void sortList(Node** head) {: Mendefinisikan fungsi sortList yang menerima pointer ke pointer kepala head.

if (*head == NULL) return;: Memeriksa apakah list kosong. Jika ya, fungsi langsung kembali.

int swapped;: Mendefinisikan variabel swapped untuk melacak apakah ada pertukaran yang terjadi selama iterasi.

Node* ptr1;: Mendefinisikan pointer ptr1 yang digunakan untuk iterasi melalui list.

Node* lptr = NULL;: Mendefinisikan pointer lptr yang digunakan untuk menandai bagian akhir dari list yang sudah diurutkan.

if ((*head)->next == *head) return;: Memeriksa apakah list hanya memiliki satu node. Jika ya, fungsi langsung kembali karena tidak perlu diurutkan.

do {: Memulai loop utama yang akan terus berjalan hingga tidak ada lagi pertukaran.

swapped = 0;: Mengatur swapped ke 0 di awal setiap iterasi.

ptr1 = *head;: Mengatur ptr1 ke kepala list untuk memulai iterasi baru.

do {: Memulai loop internal untuk membandingkan dan menukar node jika perlu.

if (ptr1->data > ptr1->next->data) {: Memeriksa apakah data di node ptr1 lebih besar dari data di node berikutnya.

swapNodes(head, ptr1, ptr1->next);: Jika ya, tukar posisi kedua node.

swapped = 1;: Mengatur swapped ke 1 untuk menandakan bahwa terjadi pertukaran.

ptr1 = ptr1->next;: Maju ke node berikutnya dalam list.

} while (ptr1->next != *head);**: Loop internal terus berjalan hingga kembali ke kepala list.

lptr = ptr1;: Setelah satu iterasi penuh, mengatur lptr ke node terakhir yang diperiksa.

} while (swapped);**: Loop utama terus berjalan hingga tidak ada lagi pertukaran yang terjadi selama iterasi penuh.

```
int main() {
    int N, i, data;
    Node* head = NULL;

    scanf("%d", &N);

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

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

    sortList(&head);

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

    return 0;
}
```

int main() {: Mendefinisikan fungsi utama program.

int N, i, data;: Mendeklarasikan variabel integer N (untuk jumlah elemen), i (sebagai indeks loop), dan data (untuk menyimpan nilai data yang diinput).

Node* head = NULL;: Mendeklarasikan pointer head yang akan menunjuk ke kepala list, diinisialisasi dengan NULL untuk menunjukkan bahwa list awalnya kosong.

scanf("%d", &N);: Membaca jumlah elemen N dari input.

for (i = 0; i < N; i++) {: Memulai loop untuk iterasi sebanyak N kali.

scanf("%d", &data);: Membaca nilai data dari input.

append(&head, data);: Menambahkan node dengan nilai data ke dalam list menggunakan fungsi append.

}: Menutup loop.

printf("List sebelum pengurutan:\n");: Mencetak pesan ke layar untuk menunjukkan bahwa list belum diurutkan.

printList(head);: Memanggil fungsi printList untuk mencetak elemen-elemen dalam list sebelum pengurutan.

sortList(&head);: Memanggil fungsi sortList untuk mengurutkan list.

printf("List setelah pengurutan:\n");: Mencetak pesan ke layar untuk menunjukkan bahwa list sudah diurutkan.

printList(head);: Memanggil fungsi printList untuk mencetak elemen-elemen dalam list setelah pengurutan.

return 0;: Mengembalikan nilai 0 untuk menunjukkan bahwa program selesai dijalankan dengan sukses.

}: Menutup fungsi main.

OUTPUT

```
PS D:\Telkom University\Tugas\Semester 2\Algoritma Pemrograman\oth15mei> cd "d:\Telkom University\Tugas\Semester 2\Algoritma Pemrograman\oth15mei\" ; if ($?) { gcc o
th.c -o oth } ; if ($?) { .\oth }
5
5
3
8
1
6
List sebelum pengurutan:
(00C42FC0 5)
(00C429F8 3)
(00C42A10 8)
(00C42A28 1)
(00C42A40 6)

List setelah pengurutan:
(00C42A28 1)
(00C429F8 3)
(00C42FC0 5)
(00C42A40 6)
(00C42A10 8)
```

```
PS D:\Telkom University\Tugas\Semester 2\Algoritma Pemrograman\oth15mei> cd "d:\Telkom University\Tugas\Semester 2\Algoritma Pemrograman\oth15mei\"
; if ($?) { gcc oth.c -o oth } ; if ($?) { .\oth }
3
31
2
123
List sebelum pengurutan:
(00852FC0 31)
(008529F8 2)
(00852A10 123)

List setelah pengurutan:
(008529F8 2)
(00852FC0 31)
(00852A10 123)
```