NAMA:WICIPTO SETIADI

NIM: 1203230042

KELAS: IF 03-01

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

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

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

    int swapped;
    Node* ptr1;
    Node* lptr = NULL;

    do {
        swapped = 0;
        ptr1 = *head;

        while (ptr1->next != *head) {
            if (ptr1->data > ptr1->next->data) {
                // Swap nodes, not just data
                Node* temp = ptr1->next;
                ptr1->next = temp->next;
                temp->next->prev = ptr1;
                temp->prev = ptr1->prev;
                ptr1->prev->next = temp;
                ptr1->prev = temp;
                temp->next = ptr1;

                if (*head == ptr1) {
                    *head = temp;
                }

                swapped = 1;
            } else {
                ptr1 = ptr1->next;
            }
        }
        lptr = ptr1;
    } while (swapped);
}

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

    // Input jumlah data
    scanf("%d", &N);

    // Input data dan masukkan ke dalam list
    for (int i = 0; i < N; i++) {
        scanf("%d", &data);
        insert(&head, data);
    }

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

    // Urutkan list
    sortList(&head);

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

    return 0;
}
```

PENJELASAN CODINGAN OTH

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

- Ini mendefinisikan struktur **Node** untuk daftar berantai ganda sirkular.
- Setiap node berisi:
    - **data**: nilai integer yang disimpan dalam node.
    - **next**: pointer ke node berikutnya dalam daftar.
    - **prev**: pointer ke node sebelumnya dalam daftar.

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

- Mengalokasikan memori untuk node baru.
- Menginisialisasi **data** node dengan nilai yang diberikan.
- Menetapkan pointer **next** dan **prev** node untuk menunjuk ke dirinya sendiri, membentuk referensi sirkular.

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

- Node** head, int data: Fungsi ini menerima dua parameter, yaitu pointer ke kepala daftar berantai (head) dan data yang akan dimasukkan ke dalam daftar berantai.
- Node* newNode = createNode(data);: Membuat node baru dengan data yang diberikan.
- if (*head == NULL) { *head = newNode; }: Jika daftar berantai kosong (kepala daftar berantai adalah NULL), maka node baru menjadi kepala daftar berantai.
- else { ... }: Jika daftar berantai tidak kosong, maka kode di dalam blok ini akan dijalankan.

- Node* last = (*head)->prev;: Mengambil node terakhir dalam daftar berantai. Dalam daftar berantai ganda, kita dapat langsung mengakses node terakhir dari kepala daftar berantai.
- newNode->next = *head;: Menyambungkan node baru dengan kepala daftar berantai. Node baru ini akan menjadi node sebelum kepala daftar berantai.
- (*head)->prev = newNode;: Menyambungkan kepala daftar berantai dengan node baru. Kepala daftar berantai ini akan menjadi node setelah node baru.
- newNode->prev = last;: Menyambungkan node baru dengan node terakhir dalam daftar berantai. Node baru ini akan menjadi node setelah node terakhir.
- last->next = newNode;: Menyambungkan node terakhir dalam daftar berantai dengan node baru. Node terakhir ini akan menjadi node sebelum node baru.

```
93   void printList(Node* head) {
94       if (head == NULL) return;
95       Node* temp = head;
96       do {
97           printf("%d ", temp->data);
98           temp = temp->next;
99       } while (temp != head);
100      printf("\n");
101  }
```

- Mencetak alamat dan data dari semua node dalam daftar berantai ganda sirkular.
- Melakukan iterasi melalui daftar mulai dari **head**.
- Menggunakan loop **do-while** untuk memastikan mencetak setidaknya sekali (untuk daftar sirkular).

```
41  void sortList(Node** head) {
42      if (*head == NULL) return;
43
44      int swapped;
45      Node* ptr1;
46      Node* lptr = NULL;
47
48      do {
49          swapped = 0;
50          ptr1 = *head;
51
52          while (ptr1->next != *head) {
53              if (ptr1->data > ptr1->next->data) {
54                  // Swap nodes, not just data
55                  Node* temp = ptr1->next;
56                  ptr1->next = temp->next;
57                  temp->next->prev = ptr1;
58                  temp->prev = ptr1->prev;
59                  ptr1->prev->next = temp;
60                  ptr1->prev = temp;
61                  temp->next = ptr1;
62
63                  if (*head == ptr1) {
64                      *head = temp;
65                  }
66
67                  swapped = 1;
68              } else {
69                  ptr1 = ptr1->next;
70              }
71          }
72          lptr = ptr1;
73      } while (swapped);
74  }
```

1. if (*head == NULL) return;: Jika daftar berantai kosong (kepala daftar berantai adalah NULL), maka fungsi ini akan langsung selesai.
2. do { ... } while (swapped);: Loop ini akan terus berjalan selama ada elemen yang ditukar posisinya.
3. swapped = 0; ptr1 = *head;: Menginisialisasi variabel swapped dengan 0 dan ptr1 dengan kepala daftar berantai.
4. while (ptr1->next != *head) { ... }: Loop ini akan berjalan selama node berikutnya dari ptr1 bukanlah kepala daftar berantai.
5. if (ptr1->data > ptr1->next->data) { ... }: Jika data pada node ptr1 lebih besar dari data pada node berikutnya, maka kode di dalam blok ini akan dijalankan.

6. Node* temp = ptr1->next; ... temp->next = ptr1;: Blok kode ini bertugas untuk menukar posisi antara node ptr1 dan node berikutnya.
7. if (*head == ptr1) { *head = temp; }: Jika ptr1 adalah kepala daftar berantai, maka kepala daftar berantai akan diubah menjadi temp.
8. swapped = 1;: Menandai bahwa ada elemen yang ditukar posisinya.
9. else { ptr1 = ptr1->next; }: Jika data pada node ptr1 tidak lebih besar dari data pada node berikutnya, maka ptr1 akan bergerak ke node berikutnya.
10. lptr = ptr1;: Menyimpan node terakhir yang diperiksa.

```
104  int main() {
105      Node* head = NULL;
106      int N, data;
107      scanf("%d", &N);
108      for (int i = 0; i < N; i++) {
109          scanf("%d", &data);
110          head = insertEnd(head, data);
111      }
112
113      printf("Sebelum Pengurutan:\n");
114      printList(head);
115      sortList(&head);
116      printf("Setelah Pengurutan:\n");
117      printList(head);
118
119      printf("Terima kasih telah menggunakan program ini!\n");
120
121      return 0;
122  }
```

1. Node* head = NULL;: Membuat variabel head yang merupakan kepala daftar berantai dan menginisialisasinya dengan NULL. Ini berarti daftar berantai awalnya kosong.
2. int N, data;: Mendeklarasikan variabel N untuk menyimpan jumlah elemen yang akan dimasukkan ke dalam daftar berantai dan variabel data untuk menyimpan data yang akan dimasukkan ke dalam daftar berantai.
3. scanf("%d", &N);: Membaca jumlah elemen yang akan dimasukkan ke dalam daftar berantai dari input pengguna.
4. for (int i = 0; i < N; i++) { ... }: Loop ini akan berjalan sebanyak N kali.
5. scanf("%d", &data);: Membaca data yang akan dimasukkan ke dalam daftar berantai dari input pengguna.
6. head = insertEnd(head, data);: Memasukkan data ke dalam daftar berantai. Fungsi insertEnd ini memasukkan data ke akhir daftar berantai.
7. printf("Sebelum Pengurutan:\n");: Menampilkan teks “Sebelum Pengurutan:” ke layar.
8. printList(head);: Menampilkan semua elemen dalam daftar berantai ke layar.
9. sortList(&head);: Mengurutkan elemen dalam daftar berantai.
10. printf("Setelah Pengurutan:\n");: Menampilkan teks “Setelah Pengurutan:” ke layar.
11. printList(head);: Menampilkan semua elemen dalam daftar berantai yang sudah diurutkan ke layar.
12. return 0;: Mengakhiri fungsi main dan mengembalikan nilai 0. Ini menandakan bahwa program berjalan dengan sukses.

OUTPUT

```
PS C:\Users\Wicipto> cd "c:\Users\Wicipto\Documents\ALGORITMA PEMROGRAMAN\S
6
5
5
3
8
1
6
List sebelum pengurutan:
Address: 00B11598, Data: 5
Address: 00B115B0, Data: 5
Address: 00B115C8, Data: 3
Address: 00B115F0, Data: 8
Address: 00B11608, Data: 1
Address: 00B11620, Data: 6

List setelah pengurutan:
Address: 00B11608, Data: 1
Address: 00B115C8, Data: 3
Address: 00B11598, Data: 5
Address: 00B115B0, Data: 5
Address: 00B11620, Data: 6
Address: 00B115F0, Data: 8

PS C:\Users\Wicipto\Documents\ALGORITMA PEMROGRAMAN\SEMESTER 2>
```