

Nama : Gede Satyamahinsa Prastita Utama

NIM : 1203220054

Kelas : IF-02-03

## TUGAS DOUBLE LINKED LIST CIRCULAR

### 1. Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
5  // Di dalamnya, terdapat variabel data bertipe data integer,
6  // pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
7  struct DoubleLinkedListCircular {
8      struct DoubleLinkedListCircular *prev;
9      int data;
10     struct DoubleLinkedListCircular *next;
11 };
12
13 // Membuat sebuah alias dalam membentuk node dengan menyimpan pointer-nya.
14 typedef struct DoubleLinkedListCircular *node;
15
16 // createNode() ⇒ digunakan untuk membuat node baru.
17 node createNode() {
18     // Inisialisasi variabel node_baru untuk menyimpan node baru.
19     node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
20     // Meminta masukkan dari user sebagai data di dalam node.
21     printf("Masukkan data: ");
22     scanf("%d", &node_baru->data);
23     // Mengembalikan node_baru untuk digunakan di dalam list.
24     return node_baru;
25 }
26
27 // insertLast() ⇒ digunakan untuk menambahkan node baru ke dalam list.
28 node insertLast(node head, node node_baru) {
29     // Jika list masih kosong, maka node_baru akan menjadi head.
30     if(head == NULL) {
31         // Pointer prev pada node_baru merujuk ke node itu sendiri.
32         node_baru->prev = node_baru;
33         // Pointer next pada node_baru merujuk ke node itu sendiri.
34         node_baru->next = node_baru;
35         // node_baru menjadi head.
36         head = node_baru;
37     } // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
38     else {
39         // Inisialisasi node last yang merujuk ke node terakhir di list.
40         node last = head->prev;
41         // Pointer next pada node_baru merujuk ke node awal di list.
42         node_baru->next = head;
43         // Pointer prev pada node_baru merujuk ke node terakhir di list.
44         node_baru->prev = last;
45         // Pointer prev pada head merujuk ke node_baru.
46         head->prev = node_baru;
47         // Pointer next pada last merujuk ke node_baru.
48         last->next = node_baru;
49     }
50
51     // Mengembalikan head yang sudah diupdate.
52     return head;
53 }
```

```

1 // acakData() ⇒ digunakan untuk mengacak data yang ada di dalam suatu list.
2 node acakData(node head, int banyak_data) {
3     // Inisialisasi node curr yang merujuk ke head.
4     node curr = head;
5     int i = 0;
6
7     // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
8     do {
9         // Inisialisasi variabel random_data bertipe data integer
10        // yang berisi bilangan acak antara 1 dan banyak_data.
11        int random_data = (rand() % banyak_data) + 1;
12        // Inisialisasi node tukar yang merujuk ke node curr.
13        node tukar = curr;
14        // Lakukan perulangan sebanyak bilangan yang dihasilkan oleh random_data.
15        for(int i = 0; i < random_data; i++) {
16            // Update node tukar menjadi node selanjutnya.
17            tukar = tukar→next;
18        }
19
20        // Lakukan pertukaran data antara node curr dan node tukar
21        // dengan melibatkan variabel temp sebagai tempat penampungan data sementara.
22        int temp = curr→data;
23        curr→data = tukar→data;
24        tukar→data = temp;
25        curr = curr→next;
26        banyak_data--;
27    } while(curr ≠ head);
28
29    // Mengembalikan head yang sudah diupdate.
30    return head;
31 }
32
33 // viewData() ⇒ digunakan untuk melihat seluruh data di dalam list.
34 void viewData(node head) {
35     // Inisialisasi node curr yang merujuk ke head.
36     node curr = head;
37
38     printf("List: ");
39     // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
40     while(curr→next ≠ head) {
41         // Menampilkan data pada node saat ini.
42         printf("%d ", curr→data);
43         // Mengubah node curr menjadi node selanjutnya.
44         curr = curr→next;
45     }
46     // Menampilkan data pada node saat ini.
47     printf("%d\n", curr→data);
48 }

```

```

1  int main() {
2      // Inisialisasi node head dengan nilai awal, yaitu NULL.
3      node head = NULL;
4      // Inisialisasi variabel banyak_data bertipe data integer dengan nilai awal, yaitu 0.
5      int banyak_data = 0;
6      // Deklarasi variabel input bertipe data char.
7      char input;
8
9      // Lakukan perulangan berikut hingga user melakukan input 'n'.
10     do {
11         // Update node head dengan memanggil fungsi insertLast().
12         head = insertLast(head, createNode());
13         // Tambah banyak_data dengan 1 setiap node yang berhasil masuk ke dalam list.
14         banyak_data++;
15         // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
16         printf("Tambah data lagi? (y/n): ");
17         fflush(stdin);
18         scanf("%c", &input);
19     } while(input != 'n');
20
21     // Menampilkan seluruh data yang ada di dalam list sebelum direverse.
22     printf("Sebelum diacak: \n");
23     viewData(head);
24
25     // Update head dengan memanggil fungsi acakData().
26     head = acakData(head, banyak_data);
27
28     // Menampilkan seluruh data yang ada di dalam list setelah direverse.
29     printf("Setelah diacak: \n");
30     viewData(head);
31
32     return 0;
33 }

```

## Output

```

Sebelum diacak:
List: 1 2 3 4 5
Setelah diacak:
List: 4 3 5 1 2

```

```

Sebelum diacak:
List: 34 21 4 72 101
Setelah diacak:
List: 72 4 101 34 21

```

## 2. Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
5  // Di dalamnya, terdapat variabel data bertipe data integer,
6  // pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
7  struct DoubleLinkedListCircular {
8      struct DoubleLinkedListCircular *prev;
9      int data;
10     struct DoubleLinkedListCircular *next;
11 };
12
13 // Membuat sebuah alias dalam membentuk node dengan menyimpan pointer-nya.
14 typedef struct DoubleLinkedListCircular *node;
15
16 // createNode() ⇒ digunakan untuk membuat node baru.
17 node createNode() {
18     // Inisialisasi variabel node_baru untuk menyimpan node baru.
19     node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
20     // Meminta masukan dari user sebagai data di dalam node.
21     printf("Masukkan data: ");
22     scanf("%d", &node_baru->data);
23     // Mengembalikan node_baru untuk digunakan di dalam list.
24     return node_baru;
25 }
26
27 // insertLast() ⇒ digunakan untuk menambahkan node baru ke dalam list.
28 node insertLast(node head, node node_baru) {
29     // Jika list masih kosong, maka node_baru akan menjadi head.
30     if(head == NULL) {
31         // Pointer prev pada node_baru merujuk ke node itu sendiri.
32         node_baru->prev = node_baru;
33         // Pointer next pada node_baru merujuk ke node itu sendiri.
34         node_baru->next = node_baru;
35         // node_baru menjadi head.
36         head = node_baru;
37     } // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
38     else {
39         // Inisialisasi node last yang merujuk ke node terakhir di list.
40         node last = head->prev;
41         // Pointer next pada node_baru merujuk ke node awal di list.
42         node_baru->next = head;
43         // Pointer prev pada node_baru merujuk ke node terakhir di list.
44         node_baru->prev = last;
45         // Pointer prev pada head merujuk ke node_baru.
46         head->prev = node_baru;
47         // Pointer next pada last merujuk ke node_baru.
48         last->next = node_baru;
49     }
50
51     // Mengembalikan head yang sudah diupdate.
52     return head;
53 }
```

```

1 // reverseData() ⇒ digunakan untuk membalikkan posisi node di dalam list.
2 // node awal menjadi node terakhir dan begitu seterusnya.
3 node reverseData(node head) {
4     // Inisialisasi node head_baru sebagai head yang sudah direverse
5     node head_baru = NULL;
6     // Inisialisasi node last yang merujuk ke node terakhir di list.
7     node last = head→prev;
8     // Inisialisasi node curr yang merujuk ke node last
9     // Deklarasi node prev untuk merujuk ke node sebelumnya.
10    node curr = last, prev;
11
12    // Perulangan berikut dilakukan hingga bertemu dengan node awal di dalam list.
13    while(curr→prev ≠ last) {
14        // Node prev merujuk ke node sebelumnya.
15        prev = curr→prev;
16        // Update head_baru dengan memasukkan node curr ke dalam list.
17        head_baru = insertLast(head_baru, curr);
18        // Update curr menjadi node sebelumnya.
19        curr = prev;
20    }
21
22    // Update head_baru dengan memasukkan node curr ke dalam list.
23    head_baru = insertLast(head_baru, curr);
24    // Mengembalikan head_baru yang sudah direverse.
25    return head_baru;
26 }
27
28 // viewData() ⇒ digunakan untuk melihat seluruh data di dalam list.
29 void viewData(node head) {
30     // Inisialisasi node curr yang merujuk ke head.
31     node curr = head;
32
33     printf("List: ");
34     // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
35     while(curr→next ≠ head) {
36         // Menampilkan data pada node saat ini.
37         printf("%d ", curr→data);
38         // Mengubah node curr menjadi node selanjutnya.
39         curr = curr→next;
40     }
41     // Menampilkan data pada node saat ini.
42     printf("%d\n", curr→data);
43 }
44
45 int main() {
46     // Inisialisasi node head dengan nilai awal, yaitu NULL.
47     node head = NULL;
48     // Deklarasi variabel input bertipe data char.
49     char input;
50
51     // Lakukan perulangan berikut hingga user melakukan input 'n'.
52     do {
53         // Update node head dengan memanggil fungsi insertLast().
54         head = insertLast(head, createNode());
55         // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
56         printf("Tambah data lagi? (y/n): ");
57         fflush(stdin);
58         scanf("%c", &input);
59     } while(input ≠ 'n');
60
61     // Menampilkan seluruh data yang ada di dalam list sebelum direverse.
62     printf("Sebelum direverse: \n");
63     viewData(head);
64
65     // Update head dengan memanggil fungsi reverse().
66     head = reverseData(head);
67
68     // Menampilkan seluruh data yang ada di dalam list setelah direverse.
69     printf("Setelah direverse: \n");
70     viewData(head);
71     return 0;
72 }

```

## Output

Sebelum direverse: List: 10 20 30 40 50	Sebelum direverse: List: 23 41 67 78 92
Setelah direverse: List: 50 40 30 20 10	Setelah direverse: List: 92 78 67 41 23

### 3. Source Code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
5  // Di dalamnya, terdapat variabel data bertipe data integer,
6  // pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
7  struct DoubleLinkedListCircular {
8      struct DoubleLinkedListCircular *prev;
9      int data;
10     struct DoubleLinkedListCircular *next;
11 };
12
13 // Membuat sebuah alias dalam membentuk node dengan menyimpan pointer-nya.
14 typedef struct DoubleLinkedListCircular *node;
15
16 // createNode() ⇒ digunakan untuk membuat node baru.
17 node createNode() {
18     // Inisialisasi variabel node_baru untuk menyimpan node baru.
19     node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
20     // Meminta masukkan dari user sebagai data di dalam node.
21     printf("Masukkan data: ");
22     scanf("%d", &node_baru->data);
23     // Mengembalikan node_baru untuk digunakan di dalam list.
24     return node_baru;
25 }
26
27 // insertLast() ⇒ digunakan untuk menambahkan node baru ke dalam list.
28 node insertLast(node head, node node_baru) {
29     // Jika list masih kosong, maka node_baru akan menjadi head.
30     if(head == NULL) {
31         // Pointer prev pada node_baru merujuk ke node itu sendiri.
32         node_baru->prev = node_baru;
33         // Pointer next pada node_baru merujuk ke node itu sendiri.
34         node_baru->next = node_baru;
35         // node_baru menjadi head.
36         head = node_baru;
37     } // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
38     else {
39         // Inisialisasi node last yang merujuk ke node terakhir di list.
40         node last = head->prev;
41         // Pointer next pada node_baru merujuk ke node awal di list.
42         node_baru->next = head;
43         // Pointer prev pada node_baru merujuk ke node terakhir di list.
44         node_baru->prev = last;
45         // Pointer prev pada head merujuk ke node_baru.
46         head->prev = node_baru;
47         // Pointer next pada last merujuk ke node_baru.
48         last->next = node_baru;
49     }
50
51     // Mengembalikan head yang sudah diupdate.
52     return head;
53 }
```



```

1 // sortingAscending() ⇒ digunakan untuk menempatkan node pada posisi yang tepat secara ascending.
2 node sortingAscending(node head, node curr1) {
3     // Jika list masih kosong, maka node curr1 menjadi head.
4     if(head == NULL) {
5         head = insertLast(head, curr1);
6         // Jika sudah ada node di dalam list, maka lakukan pengecekan terhadap data di dalamnya
7         // Jika data pada head lebih besar atau sama dengan data pada curr1, maka curr1 menjadi head.
8     } else if(head->data ≥ curr1->data) {
9         // Pointer prev pada head merujuk ke curr1.
10        head->prev = curr1;
11        // Pointer next pada head merujuk ke curr1.
12        head->next = curr1;
13        // Pointer prev pada curr1 merujuk ke head.
14        curr1->prev = head;
15        // Pointer next pada curr1 merujuk ke head.
16        curr1->next = head;
17        // curr1 menjadi head.
18        head = curr1;
19        // Jika sudah ada node di dalam list, maka lakukan pengecekan terhadap data di dalamnya
20    } else {
21        // Inisialisasi node curr2 yang merujuk ke head.
22        node curr2 = head;
23
24        // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list atau
25        // data pada node setelah node curr2 lebih besar daripada data pada curr1.
26        while(curr2->next ≠ head && curr2->next->data < curr1->data) {
27            // Update curr2 menjadi node selanjutnya.
28            curr2 = curr2->next;
29        }
30
31        // Jika node curr2 adalah node terakhir, maka ubah node curr1 menjadi node terakhir.
32        if(curr2->next == head) {
33            // Pointer next pada curr1 merujuk ke head.
34            curr1->next = curr2->next;
35            // Pointer next pada curr2 merujuk ke curr1.
36            curr2->next = curr1;
37            // Pointer prev pada curr1 merujuk ke curr2.
38            curr1->prev = curr2;
39            // Pointer prev pada head merujuk ke curr1.
40            head->prev = curr1;
41            // Jika node curr2 bukan node terakhir, maka sisipkan curr1 setelah curr2.
42        } else {
43            // Pointer next pada curr1 merujuk ke node setelah curr2.
44            curr1->next = curr2->next;
45            // Pointer prev pada curr1 merujuk ke curr2.
46            curr1->prev = curr2;
47            // Pointer next pada curr2 merujuk ke curr1.
48            curr2->next = curr1;
49            // Pointer prev pada node setelah curr2 merujuk ke curr1.
50            curr2->next->prev = curr1;
51        }
52    }
53
54    // Mengembalikan head yang sudah diupdate.
55    return head;
56 }
57
58 // ascendingData() ⇒ digunakan untuk mengubah data di dalam suatu list menjadi terurut secara ascending.
59 node ascendingData(node head) {
60     // Inisialisasi node head_baru sebagai list data yang sudah diurutkan.
61     node head_baru = NULL;
62     // Inisialisasi node curr1 yang merujuk ke head.
63     // Deklarasi node next.
64     node curr1 = head, next;
65
66     // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
67     while(curr1->next ≠ head) {
68         // Simpan node selanjutnya ke dalam node next.
69         next = curr1->next;
70         // Update head_baru dengan memanggil fungsi sortingData().
71         head_baru = sortingAscending(head_baru, curr1);
72         // Update curr1 menjadi next.
73         curr1 = next;
74     }
75
76     // Update head_baru dengan memanggil fungsi sortingData().
77     head_baru = sortingAscending(head_baru, curr1);
78     // Mengembalikan head_baru yang sudah diupdate.
79     return head_baru;
80 }

```

```

1 // viewData() ⇒ digunakan untuk melihat seluruh data di dalam list.
2 void viewData(node head) {
3     // Inisialisasi node curr yang merujuk ke head.
4     node curr = head;
5
6     // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
7     while(curr->next ≠ head) {
8         // Menampilkan data pada node saat ini.
9         printf("Elemen %d ", curr->data);
10        // Mengubah node curr menjadi node selanjutnya.
11        curr = curr->next;
12    }
13    // Menampilkan data pada node saat ini.
14    printf("Elemen %d\n", curr->data);
15 }
16
17 int main() {
18     // Inisialisasi node head dengan nilai awal, yaitu NULL.
19     node head = NULL;
20     // Deklarasi variabel input bertipe data char.
21     char input;
22
23     // Lakukan perulangan berikut hingga user melakukan input 'n'.
24     do {
25         // Update node head dengan memanggil fungsi insertLast().
26         head = insertLast(head, createNode());
27         // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
28         printf("Tambah data lagi? (y/n): ");
29         fflush(stdin);
30         scanf("%c", &input);
31     } while(input ≠ 'n');
32
33     // Menampilkan seluruh data yang ada di dalam list sebelum diurutkan.
34     printf("List sebelum diurutkan: ");
35     viewData(head);
36
37     head = ascendingData(head);
38
39     // Menampilkan seluruh data yang ada di dalam list sebelum diurutkan.
40     printf("List setelah diurutkan: ");
41     viewData(head);
42
43     return 0;
44 }

```

## Output

```

List sebelum diurutkan: Elemen 3 Elemen 1 Elemen 4 Elemen 2 Elemen 5
List setelah diurutkan: Elemen 1 Elemen 2 Elemen 3 Elemen 4 Elemen 5

```

```

List sebelum diurutkan: Elemen 32 Elemen 11 Elemen 53 Elemen 98 Elemen 74
List setelah diurutkan: Elemen 11 Elemen 32 Elemen 53 Elemen 74 Elemen 98

```