Nama : Gede Satyamahinsa Prastita Uttama

NIM : 1203220054

Kelas : IF-02-03

TUGAS DOUBLE LINKED LIST CIRCULAR

1. Source Code

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

// Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
// Di dalamnya, terdapat variabel data bertipe data integer,
// pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
struct DoubleLinkedListCircular {
  struct DoubleLinkedListCircular *prev;
  int data;
  struct DoubleLinkedListCircular *next;
};

// Membuat sebuah alias dalam membentuk node dengan menyimpan pointernya.
typedef struct DoubleLinkedListCircular *node;

// createNode() => digunakan untuk membuat node baru.
node createNode() {
  // Inisialisasi cariabel node_baru untuk menyimpan node baru.
  node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
  // Meminta masukkan dari user sebagai data di dalam node.
  printf("Masukkan data: ");
  scanf("%d", &node_baru->data);
  // Mengembalikan node_baru untuk digunakan di dalam list.
  return node_baru;
}

// insertLast() => digunakan untuk menambahkan node baru ke dalam list.
node insertLast(node head, node node_baru) {
  // Jika list masih kosong, maka node_baru akan menjadi head.
  if(head == NULL) {
    // Pointer prev pada node_baru merujuk ke node itu sendiri.
    node_baru->prev = node_baru;
    // Pointer next pada node_baru merujuk ke node itu sendiri.
    node_baru->next = node_baru;
    // node_baru menjadi head.
    head = node_baru;
  // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
  } else {
    // Inisialisasi node last yang merujuk ke node terakhir di list.
    node last = head->prev;
    // Pointer next pada node_baru merujuk ke node awal di list.
    node_baru->next = head;
    // Pointer prev pada node_baru merujuk ke node terakhir di list.
    node_baru->prev = last;
    // Pointer prev pada head merujuk ke node_baru.
    head->prev = node_baru;
    // Pointer next pada last merujuk ke node_baru.
    last->next = node_baru;
  }

  // Mengembalikan head yang sudah diupdate.
  return head;
}
```

```c
// acakData() => digunakan untuk mengacak data yang ada di dalam suatu list.
node acakData(node head, int banyak_data) {
  // Inisialisasi node curr yang merujuk ke head.
  node curr = head;
  int i = 0;

  // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
  do {
    // Inisialisasi variabel random_data bertipe data integer
    // yang berisi bilangan acak antara 1 dan banyak_data.
    int random_data = (rand() % banyak_data) + 1;
    // Inisialisasi node tukar yang merujuk ke node curr.
    node tukar = curr;
    // Lakukan perulangan sebanyak bilangan yang dihasilkan oleh random_data.
    for(int i = 0; i < random_data; i++) {
      // Update node tukar menjadi node selanjutnya.
      tukar = tukar->next;
    }

    // Lakukan pertukaran data antara node curr dan node tukar
    // dengan melibatkan variabel temp sebagai tempat penampungan data sementara.
    int temp = curr->data;
    curr->data = tukar->data;
    tukar->data = temp;
    curr = curr->next;
    banyak_data--;
  } while(curr != head);

  // Mengembalikan head yang sudah diupdate.
  return head;
}

// viewData() => digunakan untuk melihat seluruh data di dalam list.
void viewData(node head) {
  // Inisialisasi node curr yang merujuk ke head.
  node curr = head;

  printf("List: ");
  // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
  while(curr->next != head) {
    // Menampilkan data pada node saat ini.
    printf("%d ", curr->data);
    // Mengubah node curr menjadi node selanjutnya.
    curr = curr->next;
  }
  // Menampilkan data pada node saat ini.
  printf("%d\n", curr->data);
}
```

```c
int main() {
  // Inisialisasi node head dengan nilai awal, yaitu NULL.
  node head = NULL;
  // Inisialisasi variabel banyak_data bertipe data integer dengan nilai awal, yaitu 0.
  int banyak_data = 0;
  // Deklarasi variabel input bertipe data char.
  char input;

  // Lakukan perulangan berikut hingga user melakukan input 'n'.
  do {
    // Update node head dengan memanggil fungsi insertLast().
    head = insertLast(head, createNode());
    // Tambah banyak_data dengan 1 setiap node yang berhasil masuk ke dalam list.
    banyak_data++;
    // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
    printf("Tambah data lagi? (y/n): ");
    fflush(stdin);
    scanf("%c", &input);
  } while(input != 'n');

  // Menampilkan seluruh data yang ada di dalam list sebelum direverse.
  printf("Sebelum diacak: \n");
  viewData(head);

  // Update head dengan memanggil fungsi acakData().
  head = acakData(head, banyak_data);

  // Menampilkan seluruh data yang ada di dalam list setelah direverse.
  printf("Setelah diacak: \n");
  viewData(head);

  return 0;
}
```

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

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

// Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
// Di dalamnya, terdapat variabel data bertipe data integer,
// pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
struct DoubleLinkedListCircular {
  struct DoubleLinkedListCircular *prev;
  int data;
  struct DoubleLinkedListCircular *next;
};

// Membuat sebuah alias dalam membentuk node dengan menyimpan pointernya.
typedef struct DoubleLinkedListCircular *node;

// createNode() ⇒ digunakan untuk membuat node baru.
node createNode() {
  // Inisialisasi cariabel node_baru untuk menyimpan node baru.
  node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
  // Meminta masukkan dari user sebagai data di dalam node.
  printf("Masukkan data: ");
  scanf("%d", &node_baru->data);
  // Mengembalikan node_baru untuk digunakan di dalam list.
  return node_baru;
}

// insertLast() ⇒ digunakan untuk menambahkan node baru ke dalam list.
node insertLast(node head, node node_baru) {
  // Jika list masih kosong, maka node_baru akan menjadi head.
  if(head == NULL) {
    // Pointer prev pada node_baru merujuk ke node itu sendiri.
    node_baru->prev = node_baru;
    // Pointer next pada node_baru merujuk ke node itu sendiri.
    node_baru->next = node_baru;
    // node_baru menjadi head.
    head = node_baru;
  // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
  } else {
    // Inisialisasi node last yang merujuk ke node terakhir di list.
    node last = head->prev;
    // Pointer next pada node_baru merujuk ke node awal di list.
    node_baru->next = head;
    // Pointer prev pada node_baru merujuk ke node terakhir di list.
    node_baru->prev = last;
    // Pointer prev pada head merujuk ke node_baru.
    head->prev = node_baru;
    // Pointer next pada last merujuk ke node_baru.
    last->next = node_baru;
  }

  // Mengembalikan head yang sudah diupdate.
  return head;
}
```

```c
// reverseData() => digunakan untuk membalikkan posisi node di dalam list.
// node awal menjadi node terakhir dan begitu seterusnya.
node reverseData(node head) {
  // Inisialisasi node head_baru sebagai head yang sudah direverse
  node head_baru = NULL;
  // Inisialisasi node last yang merujuk ke node terakhir di list.
  node last = head->prev;
  // Inisialisasi node curr yang merujuk ke node last
  // Deklarasi node prev untuk merujuk ke node sebelumnya.
  node curr = last, prev;

  // Perulangan berikut dilakukan hingga bertemu dengan node awal di dalam list.
  while(curr->prev != last) {
    // Node prev merujuk ke node sebelumnya.
    prev = curr->prev;
    // Update head_baru dengan memasukkan node curr ke dalam list.
    head_baru = insertLast(head_baru, curr);
    // Update curr menjadi node sebelumnya.
    curr = prev;
  }

  // Update head_baru dengan memasukkan node curr ke dalam list.
  head_baru = insertLast(head_baru, curr);
  // Mengembalikan head_baru yang sudah direverse.
  return head_baru;
}

// viewData() => digunakan untuk melihat seluruh data di dalam list.
void viewData(node head) {
  // Inisialisasi node curr yang merujuk ke head.
  node curr = head;

  printf("List: ");
  // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
  while(curr->next != head) {
    // Menampilkan data pada node saat ini.
    printf("%d ", curr->data);
    // Mengubah node curr menjadi node selanjutnya.
    curr = curr->next;
  }
  // Menampilkan data pada node saat ini.
  printf("%d\n", curr->data);
}

int main() {
  // Inisialisasi node head dengan nilai awal, yaitu NULL.
  node head = NULL;
  // Deklarasi variabel input bertipe data char.
  char input;

  // Lakukan perulangan berikut hingga user melakukan input 'n'.
  do {
    // Update node head dengan memanggil fungsi insertLast().
    head = insertLast(head, createNode());
    // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
    printf("Tambah data lagi? (y/n): ");
    fflush(stdin);
    scanf("%c", &input);
  } while(input != 'n');

  // Menampilkan seluruh data yang ada di dalam list sebelum direverse.
  printf("Sebelum direverse: \n");
  viewData(head);

  // Update head dengan memanggil fungsi reverse().
  head = reverseData(head);

  // Menampilkan seluruh data yang ada di dalam list setelah direverse.
  printf("Setelah direverse: \n");
  viewData(head);
  return 0;
}
```

Output

```
Sebelum direverse:
List: 10 20 30 40 50
Setelah direverse:
List: 50 40 30 20 10
```

```
Sebelum direverse:
List: 23 41 67 78 92
Setelah direverse:
List: 92 78 67 41 23
```

3. Source Code

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

// Deklarasi node dengan menggunakan struct DoubleLinkedListCircular.
// Di dalamnya, terdapat variabel data bertipe data integer,
// pointer prev yang merujuk ke node sebelumnya dan pointer next yang merujuk ke node selanjutnya
struct DoubleLinkedListCircular {
  struct DoubleLinkedListCircular *prev;
  int data;
  struct DoubleLinkedListCircular *next;
};

// Membuat sebuah alias dalam membentuk node dengan menyimpan pointernya.
typedef struct DoubleLinkedListCircular *node;

// createNode() => digunakan untuk membuat node baru.
node createNode() {
  // Inisialisasi cariabel node_baru untuk menyimpan node baru.
  node node_baru = (node)malloc(sizeof(struct DoubleLinkedListCircular));
  // Meminta masukkan dari user sebagai data di dalam node.
  printf("Masukkan data: ");
  scanf("%d", &node_baru->data);
  // Mengembalikan node_baru untuk digunakan di dalam list.
  return node_baru;
}

// insertLast() => digunakan untuk menambahkan node baru ke dalam list.
node insertLast(node head, node node_baru) {
  // Jika list masih kosong, maka node_baru akan menjadi head.
  if(head == NULL) {
    // Pointer prev pada node_baru merujuk ke node itu sendiri.
    node_baru->prev = node_baru;
    // Pointer next pada node_baru merujuk ke node itu sendiri.
    node_baru->next = node_baru;
    // node_baru menjadi head.
    head = node_baru;
  // Jika sudah ada node di dalam list, maka tambahkan node_baru ke akhir list.
  } else {
    // Inisialisasi node last yang merujuk ke node terakhir di list.
    node last = head->prev;
    // Pointer next pada node_baru merujuk ke node awal di list.
    node_baru->next = head;
    // Pointer prev pada node_baru merujuk ke node terakhir di list.
    node_baru->prev = last;
    // Pointer prev pada head merujuk ke node_baru.
    head->prev = node_baru;
    // Pointer next pada last merujuk ke node_baru.
    last->next = node_baru;
  }

  // Mengembalikan head yang sudah diupdate.
  return head;
}
```

```
// sortingAscending() => digunakan untuk menempatkan node pada posisi yang tepat secara ascending.
node sortingAscending(node head, node curr1) {
  // Jika list masih kosong, maka node curr1 menjadi head.
  if(head == NULL) {
    head = insertLast(head, curr1);
  // Jika sudah ada node di dalam list, maka lakukan pengecekan terhadap data di dalamnya
  // Jika data pada head lebih besar atau sama dengan data pada curr1, maka curr1 menjadi head.
  } else if(head->data ≥ curr1->data) {
    // Pointer prev pada head merujuk ke curr1.
    head->prev = curr1;
    // Pointer next pada head merujuk ke curr1.
    head->next = curr1;
    // Pointer prev pada curr1 merujuk ke head.
    curr1->prev = head;
    // Pointer next pada curr1 merujuk ke head.
    curr1->next = head;
    // curr1 menjadi head.
    head = curr1;
  // Jika sudah ada node di dalam list, maka lakukan pengecekan terhadap data di dalamnya
  } else {
    // Inisialisasi node curr2 yang merujuk ke head.
    node curr2 = head;

    // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list atau
    // data pada node setelah node curr2 lebih besar daripada data pada curr1.
    while(curr2->next ≠ head && curr2->next->data < curr1->data) {
      // Update curr2 menjadi node selanjutnya.
      curr2 = curr2->next;
    }

    // Jika node curr2 adalah node terakhir, maka ubah node curr1 menjadi node terakhir.
    if(curr2->next == head) {
      // Pointer next pada curr1 merujuk ke head.
      curr1->next = curr2->next;
      // Pointer next pada curr2 merujuk ke curr1.
      curr2->next = curr1;
      // Pointer prev pada curr1 merujuk ke curr2.
      curr1->prev = curr2;
      // Pointer prev pada head merujuk ke curr1.
      head->prev = curr1;
    // Jika node curr2 bukan node terakhir, maka sisipkan curr1 setelah curr2.
    } else {
      // Pointer next pada curr1 merujuk ke node setelah curr2.
      curr1->next = curr2->next;
      // Pointer prev pada curr1 merujuk ke curr2.
      curr1->prev = curr2;
      // Pointer next pada curr2 merujuk ke curr1.
      curr2->next = curr1;
      // Pointer prev pada node setelah curr2 merujuk ke curr1.
      curr2->next->prev = curr1;
    }
  }

  // Mengembalikan head yang sudah diupdate.
  return head;
}

// ascendingData() => digunakan untuk mengubah data di dalam suatu list menjadi terurut secara ascending.
node ascendingData(node head) {
  // Inisialisasi node head_baru sebagai list data yang sudah diurutkan.
  node head_baru = NULL;
  // Inisialisasi node curr1 yang merujuk ke head.
  // Deklarasi node next.
  node curr1 = head, next;

  // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
  while(curr1->next ≠ head) {
    // Simpan node selanjutnya ke dalam node next.
    next = curr1->next;
    // Update head_baru dengan memanggil fungsi sortingData().
    head_baru = sortingAscending(head_baru, curr1);
    // Update curr1 menjadi next.
    curr1 = next;
  }

  // Update head_baru dengan memanggil fungsi sortingData().
  head_baru = sortingAscending(head_baru, curr1);
  // Mengembalikan head_baru yang sudah diupdate.
  return head_baru;
}
```

```c
// viewData() => digunakan untuk melihat seluruh data di dalam list.
void viewData(node head) {
  // Inisialisasi node curr yang merujuk ke head.
  node curr = head;

  // Perulangan berikut dilakukan hingga bertemu dengan node terakhir di dalam list.
  while(curr->next != head) {
    // Menampilkan data pada node saat ini.
    printf("Elemen %d ", curr->data);
    // Mengubah node curr menjadi node selanjutnya.
    curr = curr->next;
  }
  // Menampilkan data pada node saat ini.
  printf("Elemen %d\n", curr->data);
}

int main() {
  // Inisialisasi node head dengan nilai awal, yaitu NULL.
  node head = NULL;
  // Deklarasi variabel input bertipe data char.
  char input;

  // Lakukan perulangan berikut hingga user melakukan input 'n'.
  do {
    // Update node head dengan memanggil fungsi insertLast().
    head = insertLast(head, createNode());
    // Meminta user untuk memasukkan input dalam melanjutkan perulangan.
    printf("Tambah data lagi? (y/n): ");
    fflush(stdin);
    scanf("%c", &input);
  } while(input != 'n');

  // Menampilkan seluruh data yang ada di dalam list sebelum diurutkan.
  printf("List sebelum diurutkan: ");
  viewData(head);

  head = ascendingData(head);

  // Menampilkan seluruh data yang ada di dalam list sebelum diurutkan.
  printf("List setelah diurutkan: ");
  viewData(head);

  return 0;
}
```

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
```