Nama : Gede Satyamahinsa Prastita Uttama

NIM : 1203220054

Kelas : IF-02-03

SOAL DOUBLE LINKED LIST

1. Source Code

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

// Deklarasi node dengan menggunakan struct DoubleLinkedList.
// Di dalamnya, terdapat pointer prev yang mengarah ke node sebelumnya,
// data digunakan untuk menyimpan nilai, dan pointer next yang mengarah ke node selanjutnya.
struct DoubleLinkedList {
  struct DoubleLinkedList *prev;
  int data;
  struct DoubleLinkedList *next;
};

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

// insertLast() ⇒ digunakan untuk memasukkan sebuah node ke akhir list.
node insertLast(node head) {
  // Inisialisasi sebuah node baru untuk menyimpan data baru.
  node node_baru = (node)malloc(sizeof(struct DoubleLinkedList));
  // Inisialisasi node bernama tail untuk mengetahui node terakhir.
  node tail = head;

  // Meminta masukan dari user untuk dimasukkan ke dalam data di node baru.
  printf("Masukkan bilangan: ");
  scanf("%d", &node_baru->data);

  // Jika belum ada node di dalam list, maka node baru akan menjadi head.
  if(head == NULL) {
    // Pointer prev pada node baru akan merujuk ke head.
    node_baru->prev = head;
    // Pointer next pada node baru akan merujuk ke head.
    node_baru->next = head;
    // Node baru akan menjadi head.
    head = node_baru;
  } else {
    // Jika sudah ada node di dalam list,
    // maka lakukan pelacakan untuk mengetahui node terakhir di dalam list.
    while(tail->next != NULL) {
      tail = tail->next;
    }

    // Pointer next pada node terakhir akan merujuk ke node baru.
    tail->next = node_baru;
    // Pointer prev pada node baru akan merujuk ke node terakhir.
    node_baru->prev = tail;
    // Pointer next pada node baru akan menjadi NULL.
    node_baru->next = NULL;
  }

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

```c
// view() => digunakan untuk melihat seluruh isi list.
void view(node head) {
  // Inisialisasi node bernama curr untuk melacak node saat ini di dalam list.
  node curr = head;
  // Deklarasi node bernama lastNode untuk mendapatkan node terakhir.
  node lastNode;

  // Mencetak seluruh bilangan secara First In First Out.
  printf("Data Bilangan yang Telah Diinputkan secara FIFO: ");
  // Jika node curr bernilai NULL,
  if(curr == NULL) {
    // Maka tampilkan pesan bahwa data kosong.
    printf("Data Kosong!");
  } else {
    // Jika node curr tidak kosong,
    printf("\n");
    // Maka lacak setiap node yang ada di dalam list.
    // Lakukan perulangan hingga node curr bernilai NULL.
    while(curr != NULL) {
      // Cetak data yang ada di dalam node curr.
      printf("%d\t", curr->data);
      // Update lastNode menjadi node curr.
      lastNode = curr;
      // Update node curr menjadi node selanjutnya.
      curr = curr->next;
    }

    // Update curr menjadi lastNode sebagai perulangan dari akhir node.
    curr = lastNode;
    printf("\n");


    // Mencetak seluruh bilangan secara Last In First Out.
    printf("Data Bilangan yang Telah Diinputkan secara LIFO: ");
    printf("\n");
    // Lakukan perulangan hingga node curr bernilai NULL.
    while(curr != NULL) {
      // Cetak data yang ada di dalam node curr.
      printf("%d\t", curr->data);
      // Update node curr menjadi node sebelumnya.
      curr = curr->prev;
    }
  }
}

int main() {
  printf("------ PROGRAM DLL LIFO & FIFO ------\n");
  // Inisialisasi node bernama head dengan nilai awal, yaitu NULL.
  node head = NULL;
  // Deklarasi variabel bernama input bertipe data char.
  char input;

  // Lakukan perulangan berikut terlebih dahulu hingga input dari user bernilai 't'.
  do {
    // Update node head dengan memanggil fungsi insertLast().
    head = insertLast(head);
    // Meminta masukan dari user untuk menambah node ke dalam list.
    printf("Ada data lagi (y/t): ");
    // fflush(stdin) digunakan untuk menghapus stream/buffer yang ada.
    fflush(stdin);
    scanf("%c", &input);
  } while(input != 't');

  // Memanggil fungsi view() untuk menampilkan seluruh bilangan secara FIFO maupun LIFO.
  view(head);
  return 0;
}
```

Output

```
------ PROGRAM DLL LIFO & FIFO ------
Masukkan bilangan: 10
Ada data lagi (y/t): y
Masukkan bilangan: 20
Ada data lagi (y/t): y
Masukkan bilangan: 30
Ada data lagi (y/t): y
Masukkan bilangan: 40
Ada data lagi (y/t): y
Masukkan bilangan: 50
Ada data lagi (y/t): t
Data Bilangan yang Telah Diinputkan secara FIFO:
10      20      30      40      50
Data Bilangan yang Telah Diinputkan secara LIFO:
50      40      30      20      10
```

2. Source Code

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

// Deklarasi node dengan menggunakan struct DataPasien.
// Di dalamnya, terdapat pointer prev yang mengarah ke node sebelumnya,
// nama digunakan untuk menyimpan nama pasien,
// dan pointer next yang mengarah ke node selanjutnya.
struct DataPasien {
  struct DataPasien *prev;
  char nama[100];
  struct DataPasien *next;
};

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

// tambahAntrian() ⇒ digunakan untuk menambahkan data pasien ke dalam antrian.
pasien tambahAntrian(pasien head) {
  // Inisialisasi pasien baru untuk menyimpan nama pasien.
  pasien pasien_baru = (pasien)malloc(sizeof(struct DataPasien));

  // Meminta masukan dari user untuk dimasukkan ke dalam nama di node.
  printf("Masukkan nama pasien: ");
  scanf("%s", &pasien_baru->nama);

  // Inisialisasi node pasien bernama tail untuk mengetahui node terakhir.
  pasien tail = head;

  // Jika belum ada node di dalam antrian,
  if(head == NULL) {
    // Pointer prev pada pasien baru akan merujuk ke head.
    pasien_baru->prev = head;
    // Pointer next pada pasien baru akan merujuk ke head.
    pasien_baru->next = head;
    // Pasien baru akan menjadi head.
    head = pasien_baru;
  } else {
    // Jika sudah ada node di dalam antrian,
    // maka lakukan pelacakan untuk mengetahui node terakhir di dalam antrian.
    while(tail->next != NULL) {
      tail = tail->next;
    }

    // Pointer next pada node terakhir akan merujuk ke pasien baru.
    tail->next = pasien_baru;
    // Pointer prev pada pasien baru akan merujuk ke tail.
    pasien_baru->prev = tail;
    // Pointer next pada pasien baru akan menjadi NULL.
    pasien_baru->next = NULL;
  }

  // Menampilkan pesan bahwa data berhasil dimasukkan.
  printf("Data berhasil dimasukkan!\n");
  // Mengembalikan head yang sudah diupdate.
  return head;
}
```

```c
// hapusAntrian() => digunakan untuk menghapus data pasien dari dalam antrian.
pasien hapusAntrian(pasien head) {
  // Jika belum ada node di dalam antrian,
  // maka tampilkan pesan bahwa data kosong.
  if(head == NULL) {
    printf("Data kosong!\n");
  } else {
    // Jika sudah ada node di dalam antrian,
    // maka lanjutkan kode berikut.

    // Deklarasi variabel bernama nama_pasien bertipe data char.
    char nama_pasien[100];

    // Meminta masukan dari user untuk dimasukkan ke dalam nama_pasien.
    printf("Masukkan nama pasien: ");
    scanf("%s", &nama_pasien);
    // Inisialisasi node bernama cari untuk melacak node sesuai dengan pasien yang dicari.
    pasien cari = head;

    // Lakukan perulangan hingga cari bernilai NULL dan
    // nama pasien yang dicari sama dengan nama yang ada di dalam antrian.
    while(cari != NULL && strcmp(cari->nama, nama_pasien)) {
      // Jika kondisi terpenuhi, maka lanjut ke node berikutnya.
      cari = cari->next;
    }

    // Jika cari bernilai NULL, maka data tidak ditemukan.
    if(cari == NULL) {
      printf("Data tidak ditemukan!\n");
    } else {
      // Jika cari bernilai head, maka
      if(cari == head) {
        // cari menjadi head dan head menjadi node berikutnya.
        cari = head;
        head = head->next;
      } else if(cari->next == NULL) {
        // Jika node berikutnya pada cari bernilai NULL, maka
        // node berikutnya pada node sebelumnya di node cari bernilai NULL.
        cari->prev->next = NULL;
      } else {
        // Jika semua kondisi di atas tidak terpenuhi, maka
        // node berikutnya pada node sebelumnya di node cari bernilai node berikutnya di node cari.
        cari->prev->next = cari->next;
        // node sebelumnya pada node berikutnya di node cari bernilai node sebelumnya di node cari.
        cari->next->prev = cari->prev;
      }
      // Menampilkan pesan bahwa data berhasil dihapus.
      printf("Data berhasil dihapus!\n");
    }
    // Lakukan penghapusan memori pada node cari.
    free(cari);
  }

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

```c
// cekAntrian() ⇒ digunakan untuk mengetahui banyak antrian di depan pasien.
void cekAntrian(pasien head) {
  // Jika belum ada node di dalam antrian,
  // maka tampilkan pesan bahwa data kosong.
  if(head == NULL) {
    printf("Data kosong!\n");
  } else {
    // Jika sudah ada node di dalam antrian,
    // maka lanjutkan kode berikut.

    // Deklarasi variabel bernama nama_pasien bertipe data char.
    char nama_pasien[100];
    // Inisialisasi variabel count bertipe data integer dengan nilai awal, yaitu 0.
    int count = 0;

    // Meminta masukan dari user untuk dimasukkan ke dalam nama_pasien.
    printf("Masukkan nama pasien: ");
    scanf("%s", &nama_pasien);
    // Inisialisasi node bernama cari untuk melacak node sesuai dengan pasien yang dicari.
    pasien cari = head;

    // Lakukan perulangan hingga cari bernilai NULL dan
    // nama pasien yang dicari sama dengan nama yang ada di dalam antrian.
    while(cari ≠ NULL && strcmp(cari→nama, nama_pasien)) {
      // Tambah count senilai 1.
      count++;
      // Jika kondisi terpenuhi, maka lanjut ke node berikutnya.
      cari = cari→next;
    }

    // Jika cari bernilai NULL, maka data tidak ditemukan.
    if(cari == NULL) {
      printf("Data tidak ditemukan!\n");
    } else {
      // Jika nama pasien ditemukan, maka tampilkan pesan berikut.
      printf("Antrian atas nama %s, kurang %d antrian lagi.\n", cari→nama, count);
    }
  }
}

// listAntrian() ⇒ digunakan untuk mencetak semua data nama pasien di dalam antrian.
void listAntrian(pasien head) {
  // Inisialisasi node bernama curr untuk melacak node saat ini di dalam list.
  pasien curr = head;
  // Inisialisasi variabel i bertipe data integer dengan nilai awal, yaitu 0.
  int i = 0;

  printf("Daftar Pasien dalam Antrian: \n");
  // Jika node curr bernilai NULL, maka data kosong.
  if(curr == NULL) {
    printf("Data Kosong!");
  } else {
    // Jika node curr tidak kosong, maka lakukan perulangan berikut hingga curr bernilai NULL.
    while(curr ≠ NULL) {
      // Tambah i senilai 1.
      i++;
      // Menampilkan nomor dan nama pasien saat ini.
      printf("[%d] Nama: %s\n", i, curr→nama);
      // Jika kondisi terpenuhi, maka lanjut ke node berikutnya.
      curr = curr→next;
    }
  }
}
```

```c
int main() {
  printf("------ ANTRIAN RUMAH SAKIT ------\n");
  // Inisialisasi node bernama head dengan nilai awal, yaitu NULL.
  pasien head = NULL;
  // Inisialisasi variabel pilihan bertipe data integer dengan nilai awal yaitu, 0.
  int pilihan = 0;

  // Lakukan perulangan berikut.
  while(1) {
    system("cls");
    // Menampilkan semua data nama pasien di dalam antrian.
    listAntrian(head);
    printf("\n\n");

    // Memberikan beberapa pilihan kepada user.
    printf("1. Tambah Antrian\n");
    printf("2. Hapus Antrian\n");
    printf("3. Cek Antrian\n");
    printf("4. Exit\n");
    printf("Pilihan: ");
    // Meminta masukan kepada user untuk dimasukkan ke dalam pilihan.
    scanf("%d", &pilihan);
    scanf("%*[^\n]");

    printf("\n");
    // Lakukan pengecekan terhadap nilai pilihan untuk mengeksekusi kode yang sesuai.
    switch(pilihan) {
      // Jika pilihan adalah 1, maka
      case 1:
        // Update head dengan memanggil fungsi tambahAntrian().
        head = tambahAntrian(head);
        system("pause");
        break;
      // Jika pilihan adalah 2, maka
      case 2:
        // Update head dengan memanggil fungsi hapusAntrian().
        head = hapusAntrian(head);
        system("pause");
        break;
      // Jika pilihan adalah 3, maka
      case 3:
        // Memanggil fungsi cekAntrian();
        cekAntrian(head);
        system("pause");
        break;
      // Jika pilihan adalah 4, maka
      case 4:
        // Keluar dari perulangan.
        exit(1);
      // Jika user memasukkan input yang tidak sesuai,
      // maka tampilkan pesan berikut.
      default:
        printf("Pilihan tidak valid!\n");
    }
  }

  return 0;
}
```

Output

```
Daftar Pasien dalam Antrian:
Data Kosong!

1. Tambah Antrian
2. Hapus Antrian
3. Cek Antrian
4. Exit
Pilihan:
```

```
Daftar Pasien dalam Antrian:
[1] Nama: budi
[2] Nama: angel
[3] Nama: jono
[4] Nama: kiko


1. Tambah Antrian
2. Hapus Antrian
3. Cek Antrian
4. Exit
Pilihan: 1

Masukkan nama pasien: lusi
Data berhasil dimasukkan!
Press any key to continue . . .
```

```
Daftar Pasien dalam Antrian:
[1] Nama: budi
[2] Nama: angel
[3] Nama: jono
[4] Nama: kiko
[5] Nama: lusi


1. Tambah Antrian
2. Hapus Antrian
3. Cek Antrian
4. Exit
Pilihan: 2

Masukkan nama pasien: jono
Data berhasil dihapus!
Press any key to continue . . .
```

```
Daftar Pasien dalam Antrian:
[1] Nama: budi
[2] Nama: angel
[3] Nama: kiko
[4] Nama: lusi


1. Tambah Antrian
2. Hapus Antrian
3. Cek Antrian
4. Exit
Pilihan: 3

Masukkan nama pasien: kiko
Antrian atas nama kiko, kurang 2 antrian lagi.
Press any key to continue . . .
```

3. Source Code

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

// Deklarasi node dengan menggunakan struct Node.
// Di dalamnya, terdapat pointer prev yang mengarah ke node sebelumnya,
// num digunakan untuk menyimpan bilangan,
// dan pointer next yang mengarah ke node selanjutnya.
struct Node {
  struct Node *prev;
  int num;
  struct Node *next;
};

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

// insertNilai() ⇒ digunakan untuk memasukkan sebuah bilangan ke akhir list.
node insertNilai(node head) {
  // Inisialisasi sebuah node baru dengan untuk menyimpan data baru.
  node nilai_baru = (node)malloc(sizeof(struct Node));

  // Meminta masukan dari user untuk dimasukkan ke dalam num di node,
  printf("Masukkan nilai: ");
  scanf("%d", &nilai_baru->num);

  // Pointer prev pada nilai baru akan menjadi NULL.
  nilai_baru->prev = NULL;
  // Pointer next pada nilai baru akan merujuk ke head.
  nilai_baru->next = head;

  // Jika head tidak bernilai NULL,
  // maka pointer prev pada head akan merujuk ke nilai baru.
  if(head != NULL) {
    head->prev = nilai_baru;
  }

  // Update head menjadi nilai baru.
  head = nilai_baru;
  // Mengembalikan head yang sudah diupdate.
  return head;
}
```

```c
// reverse() ⇒ digunakan untuk membalikkan posisi node secara keseluruhan.
node reverse(node head) {
  // Inisialisasi node curr dengan nilai head.
  node curr = head;
  // Inisialisasi node temp dengan nilai NULL.
  node temp = NULL;

  // Lakukan perulangan hingga node curr bernilai NULL.
  while(curr ≠ NULL) {
    // Update node temp menjadi pointer prev pada curr.
    temp = curr→prev;
    // Pointer prev pada curr akan merujuk ke pointer next pada curr.
    curr→prev = curr→next;
    // Pointer next pada curr akan merujuk ke temp.
    curr→next = temp;
    // Update node curr menjadi pointer prev pada curr.
    curr = curr→prev;
  }

  // Jika temp tidak bernilai NULL,
  // maka head akan merujuk ke pointer prev pada temp.
  if(temp ≠ NULL) {
    head = temp→prev;
  }

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

// view() ⇒ digunakan untuk melihat seluruh isi list.
void view(node head) {
  // Inisialisasi node bernama curr untuk melacak node saat ini di dalam list.
  node curr = head;

  // Jika node curr bernilai NULL,
  if(curr == NULL) {
    // Maka cetak bahwa data kosong.
    printf("Data Kosong!");
  } else {
    // Jika node curr tidak kosong,
    // maka lacak setiap node yang ada di dalam list.
    // Lakukan perulangan hingga node curr bernilai NULL.
    while(curr ≠ NULL) {
      // Cetak bilangan yang ada di dalam node curr
      printf("%d ", curr→num);
      // Update node curr menjadi node selanjutnya.
      curr = curr→next;
    }
  }
}
```

```c
int main() {
  printf("------ PROGRAM REVERSE DLL STACK ------\n");
  // Inisialisasi node bernama head dengan nilai awal, yaitu NULL.
  node head = NULL;
  // Deklarasi variabel bernama lanjut bertipe data integer.
  int lanjut;

  // Lakukan perulangan berikut terlebih dahulu hingga input dari user bernilai 999.
  do {
    // Update node head dengan memanggil fungsi insertNilai();
    head = insertNilai(head);
    // Meminta masukan dari user untuk menambah node ke dalam list.
    printf("Tambah nilai lagi? 999 = Exit : ");
    scanf("%d", &lanjut);
  } while(lanjut != 999);

  // Memanggil fungsi view() untuk menampilkan seluruh bilangan sebelum direverse.
  printf("\nStack sebelum di reverse: ");
  view(head);

  // Update node head dengan memanggil fungsi reverse().
  // Digunakan untuk membalikkan posisi node secara keseluruhan.
  head = reverse(head);

  // Memanggil fungsi view() untuk menampilkan seluruh bilangan setelah direverse.
  printf("\nStack setelah di reverse: ");
  view(head);

  return 0;
}
```

Output

```
------ PROGRAM REVERSE DLL STACK ------
Masukkan nilai: 1
Tambah nilai lagi? 999 = Exit : 1
Masukkan nilai: 2
Tambah nilai lagi? 999 = Exit : 1
Masukkan nilai: 3
Tambah nilai lagi? 999 = Exit : 1
Masukkan nilai: 4
Tambah nilai lagi? 999 = Exit : 1
Masukkan nilai: 5
Tambah nilai lagi? 999 = Exit : 1
Masukkan nilai: 6
Tambah nilai lagi? 999 = Exit : 999

Stack sebelum di reverse: 6 5 4 3 2 1
Stack setelah di reverse: 1 2 3 4 5 6
```