LAPORAN PRAKTIKUM
PERTEMUAN 6
DOUBLE LINKED LIST

Nama :

Martryatus Sofia 2311104003

Dosen :

Yudha Islami Sulistya, S.Kom., M.Cs

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO 2024

SOAL TP

Soal 1: Menambahkan Elemen di Awal dan Akhir DLL

Deskripsi Soal:

Buatlah program yang mengizinkan pengguna menambahkan elemen ke dalam DoublyLinked List di awal dan di akhir list.

Instruksi:

1. Implementasikan fungsi `insertFirst` untuk menambahkan elemen di awal list.
2. Implementasikan fungsi `insertLast` untuk menambahkan elemen di akhir list.
3. Tampilkan seluruh elemen dalam list dari depan ke belakang setelah penambahandilakukan.

Contoh Input:

- Input: Masukkan elemen pertama = 10
- Input: Masukkan elemen kedua di awal = 5
- Input: Masukkan elemen ketiga di akhir = 20

Output:

- DAFTAR ANGGOTA LIST: 5 <-> 10 <-> 20

struct Buku {
int id;
string judul;
string penulis;
};
	Struktur ini menyimpan informasi tentang buku.
	Memiliki tiga atribut:
	id: untuk menyimpan ID buku.
	judul: untuk menyimpan judul buku.
	penulis: untuk menyimpan nama penulis buku.

struct Node {
Buku info;
Node* next;
Node* prev;
};
	Struktur ini merepresentasikan node dalam Doubly Linked List.
	Memiliki dua pointer: next dan prev, yang menunjuk ke node berikutnya dan sebelumnya, serta atribut info yang menyimpan data buku.

struct List { Node* first;	Struktur ini menyimpan pointer ke node pertama (first) dan node terakhir (last) dari

Node* last; };	linked list.

void createList(List &L) {
L.first = nullptr;
L.last = nullptr;
}
	Menginisialisasi list dengan mengatur pointer first dan last ke nullptr, menandakan bahwa list kosong.

Node* alokasi(Buku buku) {
Node* newNode = new Node;
newNode->info = buku;
newNode->next = nullptr;
newNode->prev = nullptr;
return newNode;
}
	Menciptakan dan mengalokasikan node baru dengan data buku yang diberikan.
	Mengembalikan pointer ke node yang baru dibuat.

void insertLast(List &L, Node* newNode) {
if (L.first == nullptr) { // Jika list kosong
L.first = newNode;
L.last = newNode;
} else { // Jika list tidak kosong
L.last->next = newNode;
newNode->prev = L.last;
L.last = newNode;
}
}
	Menambahkan node baru di akhir linked list.
	Jika list kosong, node baru menjadi first dan last.
	Jika tidak kosong, node baru ditambahkan setelah node terakhir yang ada.

void printForward(const List &L) {
Node* current = L.first;
int i = 1;
cout << "Daftar Buku (dari awal ke akhir):\n";
while (current != nullptr) {
cout << "Buku " << i++ << ":\n";
cout << "ID Buku : " << current->info.id << endl;
cout << "Judul : " << current->info.judul << endl;
	Menampilkan daftar buku dari node pertama hingga node terakhir.
	Menggunakan loop untuk mengakses setiap node dan mencetak informasi buku.

cout << "Penulis : " << current->info.penulis << endl;
current = current->next;
}
}	

void printBackward(const List &L) {
Node* current = L.last;
int i = 1;
cout << "Daftar Buku (dari akhir ke awal):\n";
while (current != nullptr) {
cout << "Buku " << i++ << ":\n";
cout << "ID Buku : " << current->info.id << endl;
cout << "Judul : " << current->info.judul << endl;
cout << "Penulis : " << current->info.penulis << endl;
current = current->prev;
}
}
	Menampilkan daftar buku dari node terakhir hingga node pertama.
	Juga menggunakan loop untuk mengakses setiap node secara terbalik dan mencetak informasi buku.

int main() {
List L;
createList(L);
 	
int jumlahBuku;
cout << "Masukkan jumlah buku yang ingin ditambahkan: ";
cin >> jumlahBuku;
cin.ignore();
 	
for (int i = 0; i < jumlahBuku; ++i) {
Buku buku;
cout << "\nMasukkan ID Buku : ";
	Program utama dimulai dengan membuat list kosong.
	Meminta pengguna untuk memasukkan jumlah buku yang ingin ditambahkan.
	Menggunakan loop untuk mendapatkan informasi buku (ID, judul, penulis) dari pengguna, membuat node baru dengan data tersebut, dan menambahkannya ke list.
	Setelah semua buku ditambahkan, program menampilkan daftar buku dari depan ke belakang dan dari belakang ke depan.

```
        cin >> buku.id;
        cin.ignore();
        cout << "Masukkan Judul Buku: ";
        getline(cin, buku.judul);
        cout << "Masukkan Penulis   : ";
        getline(cin, buku.penulis);

        Node* newNode = alokasi(buku);
        insertLast(L, newNode);
    }

    cout << "\n";
    printForward(L);
    cout << "\n";
    printBackward(L);

    return 0;
}
```

OUTPUT :

```
Double_Linked_List\UnGuided\" ; if ($?) { g++ TP_01.cpp -o TP_01 } ; if ($?) { .\TP_01 }
Masukkan elemen pertama = 10
Masukkan elemen kedua di awal = 5
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST: 5 <-> 10 <-> 20
PS F:\Kuliah\Semester3\Strukdat - praktikum\06_Double_Linked_list\06_Double_Linked_List\UnGuided>
```

CODINGAN VS CODE

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur Node untuk menyimpan data buku
struct Buku {
    int id;
    string judul;
    string penulis;
};

struct Node {
    Buku info;
    Node* next;
    Node* prev;
};

// Struktur List untuk menyimpan pointer awal dan akhir
struct List {
    Node* first;
    Node* last;
};

// Membuat list kosong
void createList(List &L) {
    L.first = nullptr;
    L.last = nullptr;
}

// Alokasi node baru dengan data buku
Node* alokasi(Buku buku) {
    Node* newNode = new Node;
    newNode->info = buku;
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

// Menambah buku di akhir linked list
void insertLast(List &L, Node* newNode) {
    if (L.first == nullptr) { // Jika list kosong
        L.first = newNode;
        L.last = newNode;
    } else { // Jika list tidak kosong
        L.last->next = newNode;
        newNode->prev = L.last;
        L.last = newNode;
    }
}

// Menampilkan daftar buku dari awal ke akhir
void printForward(const List &L) {
    Node* current = L.first;
    int i = 1;
    cout << "Daftar Buku (dari awal ke akhir):\n";
    while (current != nullptr) {
        cout << "Buku " << i++ << ":\n";
        cout << "ID Buku   : " << current->info.id << endl;
        cout << "Judul     : " << current->info.judul <<
endl;
        cout << "Penulis   : " << current->info.penulis <<
endl;
        current = current->next;
    }
}

// Menampilkan daftar buku dari akhir ke awal
void printBackward(const List &L) {
    Node* current = L.last;
    int i = 1;
    cout << "Daftar Buku (dari akhir ke awal):\n";
    while (current != nullptr) {
        cout << "Buku " << i++ << ":\n";
        cout << "ID Buku   : " << current->info.id << endl;
        cout << "Judul     : " << current->info.judul <<
endl;
        cout << "Penulis   : " << current->info.penulis <<
endl;
        current = current->prev;
    }
}

int main() {
    List L;
    createList(L);

    int jumlahBuku;
    cout << "Masukkan jumlah buku yang ingin ditambahkan: "
;
    cin >> jumlahBuku;
    cin.ignore();

    for (int i = 0; i < jumlahBuku; ++i) {
        Buku buku;
        cout << "\nMasukkan ID Buku   : ";
        cin >> buku.id;
        cin.ignore();
        cout << "Masukkan Judul Buku: ";
        getline(cin, buku.judul);
        cout << "Masukkan Penulis   : ";
        getline(cin, buku.penulis);

        Node* newNode = alokasi(buku);
        insertLast(L, newNode);
    }

    cout << "\n";
    printForward(L);
    cout << "\n";
    printBackward(L);

    return 0;
}
```

Soal 2: Menghapus Elemen di Awal dan Akhir DLL

Deskripsi Soal:

Buatlah program yang memungkinkan pengguna untuk menghapus elemen pertama danelemen terakhir dalam Doubly Linked List.

Instruksi:

1. Implementasikan fungsi `deleteFirst` untuk menghapus elemen pertama.
2. Implementasikan fungsi `deleteLast` untuk menghapus elemen terakhir.
3. Tampilkan seluruh elemen dalam list setelah penghapusan dilakukan.

Contoh Input:

- Input: Masukkan elemen pertama = 10
- Input: Masukkan elemen kedua di akhir = 15
- Input: Masukkan elemen ketiga di akhir = 20
- Hapus elemen pertama dan terakhir.

Output:

- DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: 15

struct Node {
int data;
Node* next;
Node* prev;
};
	Struktur ini merepresentasikan elemen dalam linked list.
	Memiliki:
	data: untuk menyimpan nilai integer.
	next: pointer ke node berikutnya dalam linked list.
	prev: pointer ke node sebelumnya.

struct List {
Node* first;
Node* last;
};
	Struktur ini merepresentasikan linked list itu sendiri.
	Memiliki dua pointer:
	first: menunjuk ke node pertama dari list.

	last: menunjuk ke node terakhir dari list.

void createList(List &L) {
  L.first = nullptr;
  L.last = nullptr;
}
	Menginisialisasi list dengan menetapkan first dan last ke nullptr, menandakan bahwa list kosong.

Node* alokasi(int data) {
  Node* newNode = new Node;
  newNode->data = data;
  newNode->next = nullptr;
  newNode->prev = nullptr;
  return newNode;
}
	Menciptakan dan mengalokasikan node baru dengan nilai data yang diberikan.
	Mengatur pointer next dan prev ke nullptr.
	Mengembalikan pointer ke node yang baru dibuat.

void insertLast(List &L, int data) {
  Node* newNode = alokasi(data);
  if (L.first == nullptr) { // Jika list kosong
  L.first = newNode;
  L.last = newNode;
  } else { // Jika list tidak kosong
  newNode->prev = L.last;
  L.last->next = newNode;
  L.last = newNode;
  }
}
	Menambahkan elemen baru di akhir linked list.
	Jika list kosong, node baru menjadi node pertama dan terakhir.
	Jika tidak kosong, node baru ditambahkan setelah node terakhir yang ada.

void deleteFirst(List &L) {
  if (L.first != nullptr) { // Jika list tidak kosong
  Node* temp = L.first;
  if (L.first == L.last) { // Jika hanya ada satu elemen
  L.first = nullptr;
  L.last = nullptr;
  } else {
  L.first = L.first->next;
	Menghapus elemen pertama dari linked list.
	Jika list tidak kosong:
	Jika hanya ada satu elemen, set kedua pointer first dan last ke nullptr.
	Jika lebih dari satu elemen, memindahkan pointer first ke node kedua dan mengatur pointer prev pada node baru first ke nullptr.
	Menghapus node yang lama untuk mencegah kebocoran memori.

L.first->prev = nullptr;
}
delete temp;
} else {
cout << "List kosong, tidak ada elemen yang bisa dihapus." << endl;
}
}	

void deleteLast(List &L) {
if (L.last != nullptr) { // Jika list tidak kosong
Node* temp = L.last;
if (L.first == L.last) { // Jika hanya ada satu elemen
L.first = nullptr;
L.last = nullptr;
} else {
L.last = L.last->prev;
L.last->next = nullptr;
}
delete temp;
} else {
cout << "List kosong, tidak ada elemen yang bisa dihapus." << endl;
}
}
	Menghapus elemen terakhir dari linked list.
	Jika list tidak kosong:
	Jika hanya ada satu elemen, set kedua pointer first dan last ke nullptr.
	Jika lebih dari satu elemen, memindahkan pointer last ke node sebelumnya dan mengatur pointer next pada node baru last ke nullptr.

void printForward(const List &L) {
Node* current = L.first;
cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: ";
if (current == nullptr) {
cout << "List kosong." << endl;
return;
}
	Menampilkan semua elemen dari linked list dari awal ke akhir.
	Jika list kosong, mencetak pesan yang sesuai.
	Menggunakan loop untuk mengakses setiap node dan mencetak nilai data.

```
    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}
```

OUTPUT CODINGAN

```
Masukkan elemen pertama = 10
Masukkan elemen kedua di akhir = 15
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: 15
PS F:\Kuliah\Semester3\Strukdat - praktikum\06_Double_Linked_list\06_Double_Linked_List\UnGuided>
```

CODINGAN VS CODE

```cpp
#include <iostream>

using namespace std;

// Struktur Node untuk menyimpan data elemen
struct Node {
    int data;
    Node* next;
    Node* prev;
};

// Struktur List untuk menyimpan pointer awal dan akhir
struct List {
    Node* first;
    Node* last;
};

// Membuat list kosong
void createList(List &L) {
    L.first = nullptr;
    L.last = nullptr;
}

// Alokasi node baru dengan data elemen
Node* alokasi(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = nullptr;
    newNode->prev = nullptr;
    return newNode;
}

// Menambahkan elemen di akhir linked list
void insertLast(List &L, int data) {
    Node* newNode = alokasi(data);
    if (L.first == nullptr) { // Jika list kosong
        L.first = newNode;
        L.last = newNode;
    } else { // Jika list tidak kosong
        newNode->prev = L.last;
        L.last->next = newNode;
        L.last = newNode;
    }
}

// Menghapus elemen pertama dari linked list
void deleteFirst(List &L) {
    if (L.first != nullptr) { // Jika list tidak kosong
        Node* temp = L.first;
        if (L.first == L.last) {
 // Jika hanya ada satu elemen
            L.first = nullptr;
            L.last = nullptr;
        } else {
            L.first = L.first->next;
            L.first->prev = nullptr;
        }
        delete temp;
    } else {
        cout << "
List kosong, tidak ada elemen yang bisa dihapus." << endl;
    }
}

// Menghapus elemen terakhir dari linked list
void deleteLast(List &L) {
    if (L.last != nullptr) { // Jika list tidak kosong
        Node* temp = L.last;
        if (L.first == L.last) {
 // Jika hanya ada satu elemen
            L.first = nullptr;
            L.last = nullptr;
        } else {
            L.last = L.last->prev;
            L.last->next = nullptr;
        }
        delete temp;
    } else {
        cout << "
List kosong, tidak ada elemen yang bisa dihapus." << endl;
    }
}

// Menampilkan semua elemen dari depan ke belakang
void printForward(const List &L) {
    Node* current = L.first;
    cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: ";
    if (current == nullptr) {
        cout << "List kosong." << endl;
        return;
    }
    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}

int main() {
    List L;
    createList(L);

    // Input dan penambahan elemen sesuai instruksi soal
    int data;

    cout << "Masukkan elemen pertama = ";
    cin >> data;
    insertLast(L, data);
   // Menambahkan elemen pertama di akhir (atau awal karena
list kosong)

    cout << "Masukkan elemen kedua di akhir = ";
    cin >> data;
    insertLast(L, data);
   // Menambahkan elemen kedua di akhir

    cout << "Masukkan elemen ketiga di akhir = ";
    cin >> data;
    insertLast(L, data);
  // Menambahkan elemen ketiga di akhir

    // Hapus elemen pertama dan terakhir
    deleteFirst(L);
    deleteLast(L);

    // Tampilkan list setelah penghapusan
    printForward(L);

    return 0;
}
```

Soal 3: Menampilkan Elemen dari Depan ke Belakang dan Sebaliknya

Deskripsi Soal: Buatlah program yang memungkinkan pengguna memasukkan beberapaelemen ke dalam Doubly Linked List. Setelah elemen dimasukkan, tampilkan seluruh
elemen dalam list dari depan ke belakang, kemudian dari belakang ke depan.

Instruksi:

1. Implementasikan fungsi untuk menampilkan elemen dari depan ke belakang.
2. Implementasikan fungsi untuk menampilkan elemen dari belakang ke depan.
3. Tambahkan 4 elemen ke dalam list dan tampilkan elemen tersebut dalam dua arah.

Contoh Input:

- Input: Masukkan 4 elemen secara berurutan: 1, 2, 3, 4

Output:

- Daftar elemen dari depan ke belakang: 1 <-> 2 <-> 3 <-> 4
- Daftar elemen dari belakang ke depan: 4 <-> 3 <-> 2 <-> 1

HASIL CODINGAN VS CODE

```
Masukkan 4 elemen secara berurutan:
Masukkan elemen ke-1: 1
Masukkan elemen ke-2: 2
Masukkan elemen ke-3: 3
Masukkan elemen ke-4: 4
Daftar elemen dari depan ke belakang: 1 <-> 2 <-> 3 <-> 4
Daftar elemen dari belakang ke depan: 4 <-> 3 <-> 2 <-> 1
```

```cpp
#include <iostream>
using namespace std;

// Struktur untuk node dalam Doubly Linked List
struct Node {
    int data;
    Node* next;
    Node* prev;

    Node(int val) {
        data = val;
        next = nullptr;
        prev = nullptr;
    }
};

// Kelas untuk Doubly Linked List
class DoublyLinkedList {
private:
    Node* head;

public:
    DoublyLinkedList() {
        head = nullptr;
    }

    // Fungsi untuk menambahkan elemen di akhir list
    void append(int data) {
        Node* newNode = new Node(data);
        if (!head) {  // Jika list kosong
            head = newNode;
            return;
        }

        Node* last = head;
        while (last->next) {  // Menemukan node terakhir
            last = last->next;
        }

        last->next = newNode;  // Menghubungkan node baru
        newNode->prev = last;
   // Mengatur prev untuk node baru
    }

    // Fungsi untuk menampilkan elemen dari depan ke belakan
g
    void displayForward() {
        Node* current = head;
        cout << "Daftar elemen dari depan ke belakang: ";
        while (current) {
            cout << current->data;
            if (current->next) {
                cout << " <-> ";
            }
            current = current->next;
        }
        cout << endl;
    }

    // Fungsi untuk menampilkan elemen dari belakang ke depa
n
    void displayBackward() {
        Node* current = head;
        if (!current) return;

        // Menemukan node terakhir
        while (current->next) {
            current = current->next;
        }

        cout << "Daftar elemen dari belakang ke depan: ";
        while (current) {
            cout << current->data;
            if (current->prev) {
                cout << " <-> ";
            }
            current = current->prev;
        }
        cout << endl;
    }
};

// Fungsi utama
int main() {
    DoublyLinkedList dll;

    // Input dari pengguna
    cout << "Masukkan 4 elemen secara berurutan:" << endl;
    for (int i = 0; i < 4; ++i) {
        int element;
        cout << "Masukkan elemen ke-" << (i + 1) << ": ";
        cin >> element;
        dll.append(element);
   // Menambahkan elemen ke dalam list
    }

    // Menampilkan elemen
    dll.displayForward();
   // Tampilkan dari depan ke belakang
    dll.displayBackward();
  // Tampilkan dari belakang ke depan

    return 0;
}
```