

LAPORAN PRAKTIKUM
MODUL 6
DOUBLE LINKED LIST BAGIAN 1



Nama :

Fajar Budiawan (2311104039)

Dosen :

Yudha Islami Sulistya S.Kom,
M.Cs

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2024

TUGAS PENDAHULUAN

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

Output

```
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6> cd 'd:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output'
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> &
Masukkan elemen pertama = 10
Masukkan elemen kedua di awal = 15
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST: 15 <-> 10 <-> 20
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> |
```

```

1  #include <iostream>
2  using namespace std;
3
4  // Node class untuk Doubly Linked List
5  class Node {
6  public:
7      int data;
8      Node* prev;
9      Node* next;
10
11      Node(int value) {
12          data = value;
13          prev = nullptr;
14          next = nullptr;
15      }
16 };
17
18 // Doubly Linked List class
19 class DoublyLinkedList {
20 private:
21     Node* head;
22
23 public:
24     DoublyLinkedList() {
25         head = nullptr;
26     }
27
28     // Fungsi untuk menambahkan elemen di awal list
29     void insertFirst(int data) {
30         Node* newNode = new Node(data);
31         if (head == nullptr) { // Jika list kosong
32             head = newNode;
33         } else {
34             newNode->next = head;
35             head->prev = newNode;
36             head = newNode;
37         }
38     }
39
40     // Fungsi untuk menambahkan elemen di akhir list
41     void insertLast(int data) {
42         Node* newNode = new Node(data);
43         if (head == nullptr) { // Jika list kosong
44             head = newNode;
45         } else {
46             Node* current = head;
47             while (current->next != nullptr) { // Mencari node terakhir
48                 current = current->next;
49             }
50             current->next = newNode;
51             newNode->prev = current;
52         }
53     }
54
55     // Fungsi untuk menampilkan seluruh elemen dalam list
56     void display() {
57         Node* current = head;
58         cout << "DAFTAR ANGGOTA LIST: ";
59         while (current != nullptr) { // Menampilkan elemen dari depan ke belakang
60             cout << current->data;
61             if (current->next != nullptr) {
62                 cout << " <-> ";
63             }
64             current = current->next;
65         }
66         cout << endl;
67     }
68 };
69
70 // Fungsi utama
71 int main() {
72     DoublyLinkedList dll;
73
74     // Input elemen pertama
75     int firstElement;
76     cout << "Masukkan elemen pertama = ";
77     cin >> firstElement;
78     dll.insertFirst(firstElement);
79
80     // Input elemen kedua di awal
81     int secondElement;
82     cout << "Masukkan elemen kedua di awal = ";
83     cin >> secondElement;
84     dll.insertFirst(secondElement);
85
86     // Input elemen ketiga di akhir
87     int thirdElement;
88     cout << "Masukkan elemen ketiga di akhir = ";
89     cin >> thirdElement;
90     dll.insertLast(thirdElement);
91
92     // Tampilkan seluruh elemen dalam list
93     dll.display();
94
95     return 0;
96 }

```

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

```
1 #include <iostream>
2 using namespace std;
3
4 // Node class untuk Doubly Linked List
5 class Node {
6 public:
7     int data;
8     Node* prev;
9     Node* next;
10
11     Node(int value) {
12         data = value;
13         prev = nullptr;
14         next = nullptr;
15     }
16 };
17
18 // Doubly Linked List class
19 class DoublyLinkedList {
20 private:
21     Node* head;
22
23 public:
24     DoublyLinkedList() {
25         head = nullptr;
26     }
27
28     // Fungsi untuk menambahkan elemen di akhir list
29     void insertLast(int data) {
30         Node* newNode = new Node(data);
31         if (head == nullptr) { // Jika list kosong
32             head = newNode;
33         } else {
34             Node* current = head;
35             while (current->next != nullptr) { // Mencari node terakhir
36                 current = current->next;
37             }
38             current->next = newNode;
39             newNode->prev = current;
40         }
41     }
42
43     // Fungsi untuk menghapus elemen pertama
44     void deleteFirst() {
45         if (head == nullptr) { // Jika list kosong
46             cout << "List kosong, tidak ada elemen yang dihapus." << endl;
47             return;
48         }
49         Node* temp = head;
50         head = head->next; // Pindahkan head ke node berikutnya
51         if (head != nullptr) {
52             head->prev = nullptr; // Set prev dari head baru menjadi nullptr
53         }
54         delete temp; // Hapus node yang lama
55     }
56
57     // Fungsi untuk menghapus elemen terakhir
58     void deleteLast() {
59         if (head == nullptr) { // Jika list kosong
60             cout << "List kosong, tidak ada elemen yang dihapus." << endl;
61             return;
62         }
63         if (head->next == nullptr) { // Jika hanya ada satu elemen
64             delete head;
65             head = nullptr;
66             return;
67         }
68         Node* current = head;
69         while (current->next != nullptr) { // Mencari node terakhir
70             current = current->next;
71         }
72         current->prev->next = nullptr; // Set next dari node sebelum terakhir menjadi nullptr
73         delete current; // Hapus node terakhir
74     }
75
76     // Fungsi untuk menampilkan seluruh elemen dalam list
77     void display() {
78         Node* current = head;
79         cout << "DAFTAR ANGGOTA LIST: ";
80         while (current != nullptr) { // Menampilkan elemen dari depan ke belakang
81             cout << current->data;
82             if (current->next != nullptr) {
83                 cout << " <-> ";
84             }
85             current = current->next;
86         }
87         cout << endl;
88     }
89 };
90
91 // Fungsi utama
92 int main() {
93     DoublyLinkedList dll;
94
95     // Input elemen pertama
96     int firstElement;
97     cout << "Masukkan elemen pertama = ";
98     cin >> firstElement;
99     dll.insertLast(firstElement);
100
101     // Input elemen kedua di akhir
102     int secondElement;
103     cout << "Masukkan elemen kedua di akhir = ";
104     cin >> secondElement;
105     dll.insertLast(secondElement);
106
107     // Input elemen ketiga di akhir
108     int thirdElement;
109     cout << "Masukkan elemen ketiga di akhir = ";
110     cin >> thirdElement;
111     dll.insertLast(thirdElement);
112
113     // Tampilkan seluruh elemen dalam list sebelum penghapusan
114     cout << "DAFTAR ANGGOTA LIST SEBELUM PENGHAPUSAN: ";
115     dll.display();
116
117     // Hapus elemen pertama dan terakhir
118     dll.deleteFirst();
119     dll.deleteLast();
120
121     // Tampilkan seluruh elemen dalam list setelah penghapusan
122     cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: ";
123     dll.display();
124
125     return 0;
126 }
```

Output

```
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> cd 'd:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output'
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> & .\'Soal_2.1\main.cpp'
Masukkan elemen pertama = 10
Masukkan elemen kedua di akhir = 15
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST SEBELUM PENGHAPUSAN: DAFTAR ANGGOTA LIST: 10 <-> 15 <-> 20
DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: DAFTAR ANGGOTA LIST: 15
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> █
```

3. Buatlah program yang memungkinkan pengguna memasukkan beberapa elemen ke dalam Doubly Linked List. Setelah elemen dimasukkan, tampilkan seluruh elemen dalam list dari depan ke belakang, kemudian dari belakang ke depan.

```
1 #include <iostream>
2 using namespace std;
3
4 // Node class untuk Doubly Linked List
5 class Node {
6 public:
7     int data; // Data yang disimpan dalam node
8     Node* prev; // Pointer ke node sebelumnya
9     Node* next; // Pointer ke node berikutnya
10
11     // Constructor untuk menginisialisasi node
12     Node(int value) {
13         data = value;
14         prev = nullptr;
15         next = nullptr;
16     }
17 };
18
19 // Doubly Linked List class
20 class DoublyLinkedList {
21 private:
22     Node* head; // Pointer ke node pertama
23
24 public:
25     // Constructor untuk menginisialisasi list
26     DoublyLinkedList() {
27         head = nullptr; // List dimulai dengan head yang kosong
28     }
29
30     // Fungsi untuk menambahkan elemen di akhir list
31     void insertLast(int data) {
32         Node* newNode = new Node(data); // Buat node baru
33         if (head == nullptr) { // Jika list kosong
34             head = newNode; // Node baru menjadi head
35         } else {
36             Node* current = head; // Mulai dari head
37             while (current->next != nullptr) { // Cari node terakhir
38                 current = current->next;
39             }
40             current->next = newNode; // Hubungkan node terakhir dengan node baru
41             newNode->prev = current; // Set prev dari node baru
42         }
43     }
44
45     // Fungsi untuk menampilkan elemen dari depan ke belakang
46     void displayForward() {
47         Node* current = head; // Mulai dari head
48         cout << "Daftar elemen dari depan ke belakang: ";
49         while (current != nullptr) { // Selama current tidak null
50             cout << current->data; // Tampilkan data
51             if (current->next != nullptr) {
52                 cout << " <-> "; // Tampilkan panah jika ada node berikutnya
53             }
54             current = current->next; // Pindah ke node berikutnya
55         }
56         cout << endl;
57     }
58
59     // Fungsi untuk menampilkan elemen dari belakang ke depan
60     void displayBackward() {
61         if (head == nullptr) { // Jika list kosong
62             cout << "List kosong." << endl;
63             return;
64         }
65
66         Node* current = head; // Mulai dari head
67         // Mencari node terakhir
68         while (current->next != nullptr) {
69             current = current->next;
70         }
71
72         cout << "Daftar elemen dari belakang ke depan: ";
73         while (current != nullptr) { // Selama current tidak null
74             cout << current->data; // Tampilkan data
75             if (current->prev != nullptr) {
76                 cout << " <-> "; // Tampilkan panah jika ada node sebelumnya
77             }
78             current = current->prev; // Pindah ke node sebelumnya
79         }
80         cout << endl;
81     }
82 };
83
84 // Fungsi utama
85 int main() {
86     DoublyLinkedList dll; // Buat objek dari DoublyLinkedList
87
88     // Input 4 elemen
89     for (int i = 0; i < 4; i++) {
90         int element;
91         cout << "Masukkan elemen ke-" << (i + 1) << ": ";
92         cin >> element; // Ambil input dari pengguna
93         dll.insertLast(element); // Tambahkan elemen ke list
94     }
95
96     // Tampilkan elemen dari depan ke belakang
97     dll.displayForward();
98
99     // Tampilkan elemen dari belakang ke depan
100    dll.displayBackward();
101
102    return 0;
103 }
```

Output

```
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6> cd 'd:\Prak
enalan_CPP_Bagian_6\TP\output'
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output> 8
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
PS D:\Praktikum Struktur Data\06_Pengenalan_CPP_Bagian_6\TP\output>
```