

**LAPORAN PRAKTIKUM
STRUKTUR DATA
MODUL 6
“DOUBLE LINKED LIST (BAGIAN PERTAMA) ”**



Disusun Oleh:
Dhiya Ulhaq Ramadhan 2211104053
Kelas :
S1SE-07-02
Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng.

**PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024**

1. Tujuan

- Memahami konsep modul linked list
- Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C

Landasan Teori

Double Linked List merupakan struktur data yang terdiri dari sekumpulan elemen yang saling terhubung melalui dua buah pointer, yaitu pointer yang menunjuk ke elemen sebelumnya (prev) dan pointer yang menunjuk ke elemen berikutnya (next). Struktur ini memungkinkan traversal data baik ke depan maupun ke belakang, memberikan fleksibilitas lebih dibandingkan Single Linked List.

Operasi-operasi dasar yang dapat dilakukan pada Single Linked List meliputi pembuatan list kosong (CreateList), penambahan node baru (Insert), penghapusan node (Delete), dan pencarian node (Search). Operasi CreateList digunakan untuk menginisialisasi list kosong dengan pointer first yang menunjuk ke NULL. Operasi Insert dapat dilakukan di awal list (InsertFirst), di akhir list (InsertLast), atau setelah node tertentu (InsertAfter). Operasi Delete juga dapat dilakukan di berbagai posisi seperti awal, akhir, atau setelah node tertentu.

Implementasi Double Linked List menggunakan struktur data yang didefinisikan dalam bahasa C, dengan tipe data bentukan yang mencakup infotype untuk menyimpan informasi, address sebagai pointer ke elemen list, dan struktur elmlist yang memuat komponen info, next, dan prev. Keseluruhan list direpresentasikan dalam struktur list yang memuat pointer first dan last.

2. Guided

Source code :

```
1  #include <iostream>
2  using namespace std;
3
4  class Node {
5  public:
6      int data;
7      Node* prev;
8      Node* next;
9  };
10
11 class DoublyLinkedList {
12 public:
13     Node* head;
14     Node* tail;
15
16     // Constructor untuk inisialisas
17     DoublyLinkedList() {
18         head = nullptr;
19         tail = nullptr;
20     }
21
22     // Fungsi untuk menambahkan elem
23     void insert(int data) {
24         Node* newNode = new Node;
25         newNode->data = data;
26         newNode->prev = nullptr;
27         newNode->next = head;
28
29         if (head != nullptr) {
30             head->prev = newNode;
31         } else {
32             tail = newNode; // Jika
33         }
34         head = newNode;
35     }
36 }
```

```
37 // Fungsi untuk menghapus elemen dari de
38 void deleteNode() {
39     if (head == nullptr) {
40         return; // Jika list kosong
41     }
42     Node* temp = head;
43     head = head->next;
44     if (head != nullptr) {
45         head->prev = nullptr;
46     } else {
47         tail = nullptr; // Jika hanya sa
48     }
49     delete temp; // Hapus elemen
50 }
51
52 // Fungsi untuk mengupdate data di list
53 bool update(int oldData, int newData) {
54     Node* current = head;
55     while (current != nullptr) {
56         if (current->data == oldData) {
57             current->data = newData;
58             return true; // Jika data di
59         }
60         current = current->next;
61     }
62     return false; // Jika data tidak dit
63 }
64
65 // Fungsi untuk menghapus semua elemen d
66 void deleteAll() {
67     Node* current = head;
68     while (current != nullptr) {
69         Node* temp = current;
70         current = current->next;
71         delete temp;
72     }
73     head = nullptr;
74     tail = nullptr;
75 }
```

```

75     }
76     // Fungsi untuk menampilkan semua elemen d
77     void display() {
78         Node* current = head;
79         while (current != nullptr) {
80             cout << current->data << " ";
81             current = current->next;
82         }
83         cout << endl;
84     }
85 };
86
87 int main() {
88     DoublyLinkedList list;
89     while (true) {
90         cout << "1. Add data" << endl;
91         cout << "2. Delete data" << endl;
92         cout << "3. Update data" << endl;
93         cout << "4. Clear data" << endl;
94         cout << "5. Display data" << endl;
95         cout << "6. Exit" << endl;
96
97         int choice;
98         cout << "Enter your choice: ";
99         cin >> choice;
100
101         switch (choice) {
102             case 1: {
103                 int data;
104                 cout << "Enter data to add: ";
105                 cin >> data;
106                 list.insert(data);
107                 break;
108             }
109             case 2: {
110                 list.deleteNode();
111                 break;
112             }
113             case 3: {
114                 int oldData, newData;
115                 cout << "Enter old data: ";
116                 cin >> oldData;
117                 cout << "Enter new data: ";
118                 cin >> newData;
119                 bool updated = list.update(oldData, newData);
120                 if (!updated) {
121                     cout << "Data not found" << endl;
122                 }
123                 break;
124             }
125             case 4: {
126                 list.deleteAll();
127                 break;
128             }
129             case 5: {
130                 list.display();
131                 break;
132             }
133             case 6: {
134                 return 0;
135             }
136             default: {
137                 cout << "Invalid choice" << endl;
138                 break;
139             }
140         }
141     }
142     return 0;
143 }

```

Output :

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 24
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 24
Enter new data: 45
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
45
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
```

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 47.312 s
Press any key to continue.
```

Deskripsi program

implementasi struktur data Double Linked List menggunakan C++. Program ini menyediakan operasi dasar untuk mengelola data dalam bentuk list yang memiliki pointer ke node sebelum dan sesudahnya (doubly linked). Program dibuat dengan pendekatan Object-Oriented Programming (OOP) menggunakan dua class utama: Node dan DoublyLinkedList.

Program dimulai dengan menampilkan menu dengan enam pilihan: Add data (1), Delete data (2), Update data (3), Clear data (4), Display data (5), dan Exit (6). User diminta memilih operasi yang diinginkan dengan memasukkan angka 1-6.

Untuk pilihan Add data, user diminta memasukkan nilai yang akan ditambahkan ke dalam list. Untuk Delete data, program akan menghapus node pertama dari list. Untuk Update data, user diminta memasukkan nilai lama yang akan diganti dan nilai baru penggantinya. Untuk Clear data, program akan menghapus seluruh data dalam list. Untuk Display data, program akan menampilkan seluruh data dalam list. Untuk Exit, program akan berakhir.

UNGUIDED

1.

1. Buatlah ADT *Double Linked list* sebagai berikut di dalam file **"doublelist.h"**:

```
Type infotype : kendaraan <
  nopol : string
  warna : string
  thnBuat : integer
>
Type address : pointer to El mList
Type El mList <
  info : infotype
  next : address
  prev : address
>
Type List <
  First : address
  Last : address
>
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertLast( in/out L : List, in P : address )
```

Buatlah implementasi ADT *Double Linked list* pada file **"doublelist.cpp"** dan coba hasil implementasi ADT pada file **"main.cpp"**.

Contoh *Output* :

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1

no polisi : D004
warna      : kuning
tahun      : 90
no polisi : D003
warna      : putih
tahun      : 70
no polisi : D001
warna      : hitam
tahun      : 90
```

Jawaban :

Source code :

main.cpp

```
1  #include "doublelist.h"
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6      List L;
7      CreateList(L);
8
9      string nopol, warna;
10     int tahun;
11     char lanjut;
12
13     do {
14         infotype kendaraan;
15         cout << "masukkan nomor polisi: ";
16         cin >> kendaraan.nopol;
17
18         // Cek apakah nomor polisi sudah ada
19         bool nopolExists = false;
20         address current = L.First;
21         while (current != nullptr) {
22             if (current->info.nopol == kendaraan.nopol) {
23                 nopolExists = true;
24                 break;
25             }
26             current = current->next;
27         }
28
29         if (nopolExists) {
30             cout << "nomor polisi sudah terdaftar" << endl;
31         } else {
32             cout << "masukkan warna kendaraan: ";
33             cin >> kendaraan.warna;
34             cout << "masukkan tahun kendaraan: ";
35             cin >> kendaraan.thnBuat;
36
37             address P = alokasi(kendaraan);
38             insertLast(L, P);
39         }
40
41         printInfo(L);
42
43         cout << "\nLanjut (y/n)? ";
44         cin >> lanjut;
45     } while (lanjut == 'y' || lanjut == 'Y');
46
47     return 0;
48 }
```

doublelist.cpp

```

1   #include "doublelist.h"
2   #include <iostream>
3   using namespace std;
4
5   void CreateList(List &L) {
6       L.First = nullptr;
7       L.Last = nullptr;
8   }
9
10  address alokasi(infotype x) {
11      address P = new ElmList;
12      P->info = x;
13      P->next = nullptr;
14      P->prev = nullptr;
15      return P;
16  }
17
18  void dealokasi(address &P) {
19      delete P;
20      P = nullptr;
21  }
22
23  void printInfo(List L) {
24      address P = L.First;
25      cout << "\nDATA LIST" << endl;
26      while (P != nullptr) {
27          cout << "no polisi : " << P->info.nopol << endl;
28          cout << "warna      : " << P->info.warna << endl;
29          cout << "tahun      : " << P->info.thnBuat << endl;
30          P = P->next;
31      }
32  }
33
34  void insertLast(List &L, address P) {
35      if (L.First == nullptr) {
36          L.First = P;
37          L.Last = P;
38      } else {
39          P->prev = L.Last;
40          L.Last->next = P;
41          L.Last = P;
42      }
43  }

```

doublelist.h

```
1  #ifndef DOUBLELIST_H
2  #define DOUBLELIST_H
3  #include <string>
4  using namespace std;
5
6  struct Kendaraan {
7      string nopol;
8      string warna;
9      int thnBuat;
10 };
11
12 typedef Kendaraan infotype;
13 typedef struct ElmList *address;
14
15 struct ElmList {
16     infotype info;
17     address next;
18     address prev;
19 };
20
21 struct List {
22     address First;
23     address Last;
24 };
25
26 void CreateList(List &L);
27 address alokasi(infotype x);
28 void dealokasi(address &P);
29 void printInfo(List L);
30 void insertLast(List &L, address P);
31
32 #endif
```

Output :

```
"D:\bersama berkarya\SEMES" x + v
masukkan nomor polisi: R44RR
masukkan warna kendaraan: Pelangi
masukkan tahun kendaraan: 2019

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019

Lanjut (y/n)? y
masukkan nomor polisi: R14RQ
masukkan warna kendaraan: RGB
masukkan tahun kendaraan: 2020

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019
no polisi : R14RQ
warna     : RGB
tahun     : 2020

Lanjut (y/n)?
y
masukkan nomor polisi: A12BC
masukkan warna kendaraan: Coquette
masukkan tahun kendaraan: 2024

DATA LIST
no polisi : R44RR
warna     : Pelangi
tahun     : 2019
no polisi : R14RQ
warna     : RGB
tahun     : 2020
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Lanjut (y/n)?
```

Deskripsi program

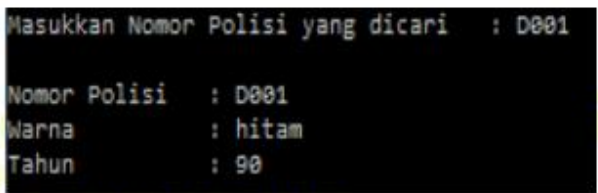
File pertama yaitu doublelist.h yang berfungsi sebagai header file. File ini mendefinisikan seluruh struktur data dan deklarasi fungsi yang akan digunakan dalam program. Di dalamnya terdapat struktur Kendaraan yang menyimpan informasi berupa nomor polisi dalam bentuk string, warna kendaraan dalam bentuk string, dan tahun pembuatan dalam bentuk integer. Selanjutnya terdapat struktur ElmList yang merupakan node dalam double linked list, berisi informasi kendaraan serta pointer next dan prev untuk menghubungkan dengan node lainnya. Terakhir ada struktur List yang memiliki pointer First dan Last untuk menandai awal dan akhir dari list.

File kedua ada `doublelist.cpp` yang berisi implementasi dari fungsi-fungsi yang telah dideklarasikan dalam header file. Fungsi `CreateList` digunakan untuk membuat list kosong dengan mengatur pointer `First` dan `Last` menjadi `nullptr`. Fungsi alokasi bertugas membuat node baru, mengisi data kendaraan ke dalamnya, dan menginisialisasi pointer `next` dan `prev`. Fungsi dealokasi digunakan untuk menghapus node dari memori dan mengatur pointer menjadi `nullptr`. Fungsi `printInfo` bertugas menampilkan seluruh data kendaraan yang ada dalam list. Fungsi `insertLast` digunakan untuk menambahkan node baru di akhir list dengan mengatur pointer-pointer yang terkait

File ketiga adalah `main.cpp` yang berisi program utama. File ini mengatur alur program secara keseluruhan, mulai dari pembuatan list kosong, proses input data kendaraan dari user, pengecekan duplikasi nomor polisi, hingga menampilkan data dan mengatur pengulangan program.

2.

Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru.
fungsi `findElm(L : List, x : infotype) : address`



```
Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi : D001
Warna : hitam
Tahun : 90
```

Jawaban : Lanjutan dari code pada no 1

Source code :

Menambahkan fungsi baru pada `doublelist.h`

```
32 | address findElm(List &L, infotype x); // Menambahkan deklarasi fungsi baru
```

Menambahkan implementasi pada `doublelist.cpp`

```

46 address findElm(List &L, infotype x) {
47     address P;
48     bool found = false;
49
50     P = L.First;
51     while ((P != nullptr) && (!found)) {
52         if (P->info.nopol == x.nopol) {
53             found = true;
54         } else {
55             P = P->next;
56         }
57     }
58     return P;
59 }

```

Menambahkan fungsi findElm pada main.cpp

```

48 // Contoh penggunaan findElm
49 infotype searchKey;
50 cout << "\nMasukkan nomor polisi yang dicari: ";
51 cin >> searchKey.nopol;
52
53 address hasil = findElm(L, searchKey);
54 if (hasil != nullptr) {
55     cout << "\nData Kendaraan ditemukan:" << endl;
56     cout << "no polisi : " << hasil->info.nopol << endl;
57     cout << "warna      : " << hasil->info.warna << endl;
58     cout << "tahun       : " << hasil->info.thnBuat << endl;
59 } else {
60     cout << "\nData kendaraan dengan nomor polisi " << searchKey.nopol << " tidak ditemukan." << endl;
61 }
62
63 return 0;
64 }

```

Output :

```
Lanjut (y/n)? y
masukkan nomor polisi: A12BC
masukkan warna kendaraan: Coquette
masukkan tahun kendaraan: 2024

DATA LIST
no polisi : R44RR
warna      : Pelangi
tahun     : 2019
no polisi : R14RQ
warna      : RGB
tahun     : 2020
no polisi : A12BC
warna      : Coquette
tahun     : 2024

Lanjut (y/n)? n

Masukkan nomor polisi yang dicari: A12BC

Data Kendaraan ditemukan:
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Process returned 0 (0x0)   execution time
Press any key to continue.
```

R

Deskripsi program

Pada file doublelist.h menambahkan deklarasi fungsi findElm yang menerima parameter List sebagai referensi dan infotype x sebagai data yang akan dicari. Fungsi ini mengembalikan nilai bertipe address yang merupakan pointer ke elemen list yang dicari.

Di dalam file doublelist.cpp, kita mengimplementasikan fungsi findElm dengan logika pencarian sekuensial pada doubly linked list. Fungsi ini menggunakan variabel lokal P bertipe address untuk traversal list dan variabel boolean found untuk menandai status pencarian. Pencarian dimulai dari node pertama (L.First) dan akan terus berlanjut selama belum mencapai akhir list ($P \neq \text{nullptr}$) dan data belum ditemukan (!found). Pada setiap iterasi, fungsi membandingkan nomor polisi pada node saat ini dengan nomor polisi yang dicari. Jika cocok, found diset true dan loop berhenti.

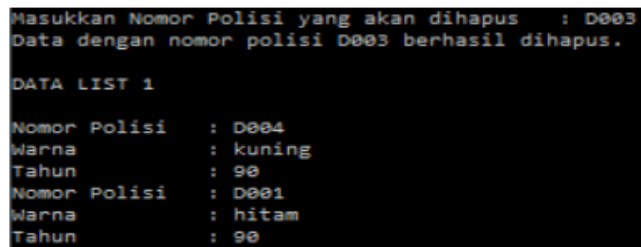
Dan pada file main.cpp, saya menambahkan kode untuk menguji fungsi findElm setelah loop input data selesai. Program akan meminta user memasukkan nomor polisi yang ingin dicari. Nomor polisi tersebut disimpan dalam variabel searchKey bertipe infotype. Kemudian program memanggil fungsi findElm dengan parameter list L dan searchKey. Hasil pencarian disimpan dalam variabel hasil bertipe address. Program mengecek nilai hasil - jika tidak nullptr berarti data ditemukan

dan program menampilkan informasi lengkap kendaraan tersebut (nomor polisi, warna, dan tahun). Jika hasil adalah nullptr, program menampilkan pesan bahwa data tidak ditemukan.

3.

3. Hapus elemen dengan nomor polisi D003 dengan prosedur *delete*.

- prosedur `deleteFirst(in/out L : List, in/out P : address)`
- prosedur `deleteLast(in/out L : List, in/out P : address)`
- prosedur `deleteAfter(in Prec : address, in/out: P : address)`



```
Masukkan Nomor Polisi yang akan dihapus : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1
Nomor Polisi : D004
Warna : kuning
Tahun : 90
Nomor Polisi : D001
Warna : hitam
Tahun : 90
```

Gambar 6-25 Output menghapus data nomor polisi

Jawaban : lanjutan dari code no 1 dan 2

Source code :

Menambahkan fungsi tambahan pada doublelist.h

```
32 void deleteFirst(List &L, address &P);
33 void deleteLast(List &L, address &P);
34 void deleteAfter(address Prec, address &P);
35 void deleteElm(List &L, string nopol);
36
37 #endif
```

Menambahkan implementasi pada doublelist.cpp


```

60 void deleteFirst(List &L, address &P) {
61     if (L.First != nullptr) {
62         P = L.First;
63         if (L.First == L.Last) {
64             L.First = nullptr;
65             L.Last = nullptr;
66         } else {
67             L.First = P->next;
68             L.First->prev = nullptr;
69             P->next = nullptr;
70         }
71     }
72 }
73
74 void deleteLast(List &L, address &P) {
75     if (L.First != nullptr) {
76         P = L.Last;
77         if (L.First == L.Last) {
78             L.First = nullptr;
79             L.Last = nullptr;
80         } else {
81             L.Last = P->prev;
82             L.Last->next = nullptr;
83             P->prev = nullptr;
84         }
85     }
86 }
87
88 void deleteAfter(address Prec, address &P) {
89     if (Prec != nullptr) {
90         P = Prec->next;
91         if (P != nullptr) {
92             if (P->next == nullptr) { // P adalah elemen terakhir
93                 Prec->next = nullptr;
94                 P->prev = nullptr;
95             } else {
96                 Prec->next = P->next;
97                 P->next->prev = Prec;
98                 P->next = nullptr;
99                 P->prev = nullptr;
100             }
101         }
102     }
103 }
104
105 void deleteElm(List &L, string nopol) {
106     address P;
107     infotype x;
108     x.nopol = nopol;
109
110     P = findElm(L, x);
111     if (P != nullptr) {
112         if (P == L.First) {
113             deleteFirst(L, P);
114         } else if (P == L.Last) {
115             deleteLast(L, P);
116         } else {
117             address Prec = P->prev;
118             deleteAfter(Prec, P);
119         }
120         dealokasi(P);
121         cout << "Data berhasil dihapus" << endl;
122     } else {
123         cout << "Data tidak ditemukan" << endl;
124     }
125 }

```

Menambahkan fungsi Menu dan hapus pada main.cpp

```
46         } while (lanjut == 'y' || lanjut == 'Y');
47
48     do {
49         cout << "\nMenu:" << endl;
50         cout << "1. Cari data kendaraan" << endl;
51         cout << "2. Hapus data kendaraan" << endl;
52         cout << "3. Tampilkan semua data" << endl;
53         cout << "4. Keluar" << endl;
54         cout << "Pilihan: ";
55         cin >> pilihan;
56
57         switch(pilihan) {
74             case '2': {
75                 string nopolHapus;
76                 cout << "\nData kendaraan yang tersedia:" << endl;
77                 printInfo(L);
78                 cout << "\nMasukkan nomor polisi yang akan dihapus: ";
79                 cin >> nopolHapus;
80                 deleteElm(L, nopolHapus);
81                 cout << "\nData setelah penghapusan:" << endl;
82                 printInfo(L);
83                 break;
84             }
```

Output :

```
Masukkan nomor polisi yang akan dihapus: R44RR
Data berhasil dihapus

Data setelah penghapusan:

DATA LIST
no polisi : R14RQ
warna     : RGB
tahun     : 2020
no polisi : A12BC
warna     : Coquette
tahun     : 2024

Menu:
1. Cari data kendaraan
2. Hapus data kendaraan
3. Tampilkan semua data
4. Keluar
Pilihan: 4

Program selesai...
```

Deskripsi program :

Fungsi hapus elemen (deleteElm) dalam program ini memungkinkan pengguna untuk menghapus data kendaraan berdasarkan nomor polisi. Saat pengguna memilih opsi hapus data kendaraan dari menu, program akan menampilkan terlebih dahulu seluruh data kendaraan yang tersedia menggunakan fungsi printInfo agar pengguna dapat melihat data mana yang ingin dihapus.

Setelah itu, program akan meminta pengguna memasukkan nomor polisi kendaraan yang ingin dihapus. Fungsi `deleteElm` akan mencari node dengan nomor polisi yang sesuai dalam double linked list. Jika data ditemukan, fungsi akan mengatur ulang pointer `prev` dan `next` dari node-node yang bersebelahan untuk memastikan list tetap terhubung dengan benar setelah penghapusan. Node yang dihapus kemudian di-dealokasi untuk membebaskan memori.

Kesimpulan

Menurut saya, Double Linked List merupakan struktur data yang menawarkan fleksibilitas dalam pengaksesan data, baik maju maupun mundur, melalui penggunaan dua pointer pada setiap elemennya. Struktur ini memungkinkan operasi-operasi manipulasi data seperti penyisipan dan penghapusan dapat dilakukan dengan lebih efisien dibandingkan Single Linked List, karena setiap elemen dapat diakses langsung dari kedua arah.