ALGORITMA DAN STUKTUR DATA

Linked List

Oleh :

Fina Salsabila Pramudita (5223600006)

Program Studi Sarjana Terapan Teknologi Game

Departemen Teknologi Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2024

Membuat Program utuh linked list.

Langkah membangun Linked List

1. Deklarasi
2. Alokasi memori
3. Mengisi data
4. Menyiapkan untuk dihubungkan dengan data baru berikutnya

Program Utuh Liked List

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

// Struktur Node untuk merepresentasikan elemen dalam linked list
struct Node {
    int data;
    Node* next;
};

Node* tail = NULL; // Inisialisasi pointer ekor

// Fungsi untuk menambahkan node baru ke akhir linked list
void create(Node*& head, int a);
void createAtBeginning(Node*& head, int a);
void display(Node* head);
void updateTail(Node* head, Node*& tail);
void insertAfter(Node*& head, int after, int value);
void insertBefore(Node*& head, int before, int value);
void search(Node*& head, int i);
void remove(Node*& head);
void deleteNodeAtIndex(Node *&head, int index);
void deleteNodeWithData(Node *&head, int data);

int main() {
    Node* head = NULL; // Inisialisasi pointer kepala linked list

    // Contoh penggunaan fungsi-fungsi linked list
    create(head, 20);
    display(head);
    create(head, 55);
    display(head);
```

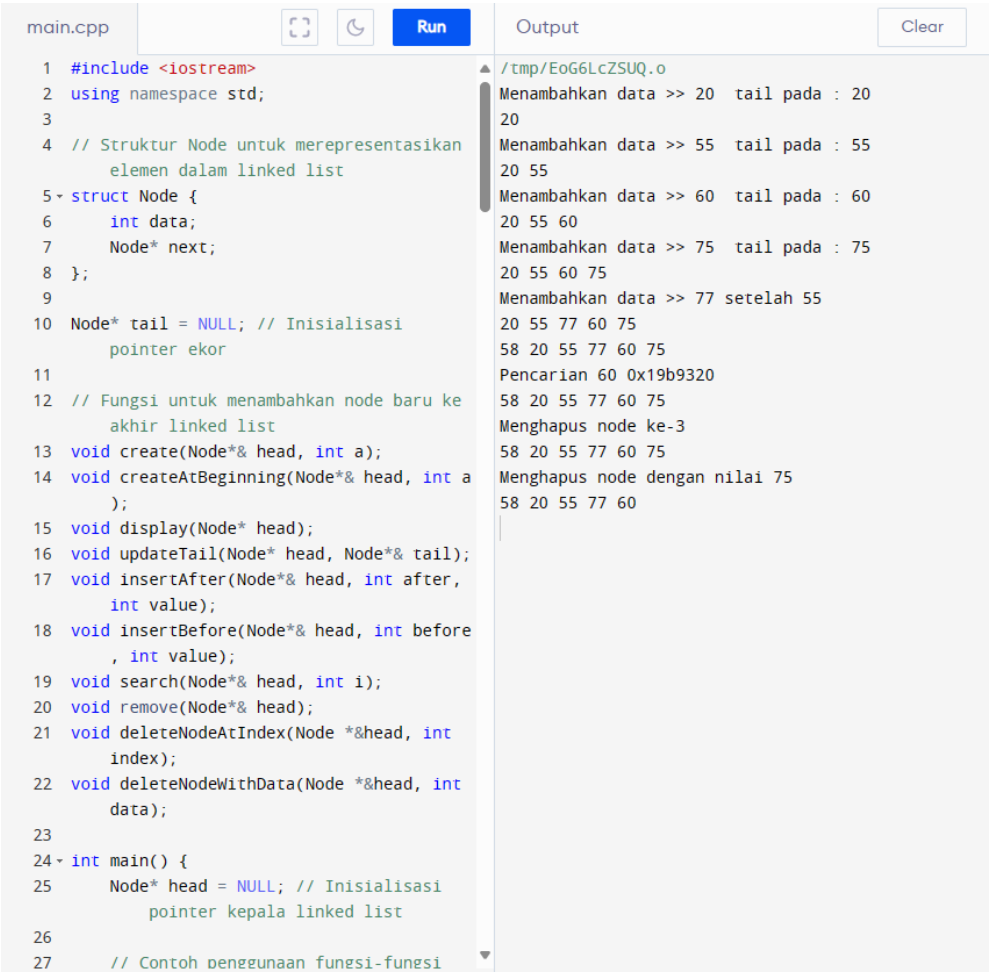
```
    create(head, 60);
    display(head);
    create(head, 75);
    display(head);
    insertAfter(head, 55, 77);
    display(head);
    insertBefore(head, 20, 58);
    display(head);
    search(head, 60);
    display(head);
    deleteNodeAtIndex(head, 3);
    display(head);
    deleteNodeWithData(head, 75);
    display(head);
    remove(head); // Menghapus seluruh linked list
}

// Fungsi untuk menambahkan node baru ke akhir linked list
void create(Node*& head, int a) {
    cout << "Menambahkan data >> " << a;
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        tail = head; // Jika linked list kosong, node baru
menjadi kepala dan ekor
    } else {
        tail->next = newNode;
        tail = newNode; // Menunjukkan ekor ke node baru
    }
    updateTail(head, tail); // Memperbarui pointer ekor
}

// Fungsi untuk menambahkan node baru di awal linked list
void createAtBeginning(Node*& head, int a) {
    Node* newNode = new Node();
    newNode->data = a;
```

```
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        tail = head; // Jika linked list kosong, node baru
menjadi kepala dan ekor
    } else {
        newNode->next = head;
        head = newNode; // Node baru menjadi kepala linked list
    }
}

// Fungsi untuk menampilkan linked list
void display(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Fungsi untuk memperbarui pointer ekor
void updateTail(Node* head, Node*& tail) {
    tail = head;
    while (tail->next != NULL) {
        tail = tail->next;
    }
    cout << "\ttail pada : " << tail->data << endl;
}

// Fungsi untuk mencari sebuah nilai dalam linked list
void search(Node*& head, int i) {
    Node* searchNode = head;
    while (searchNode->data != i) {
        searchNode = searchNode->next;
    }
    cout << "Pencarian " << searchNode->data << " " <<
searchNode->next << endl;
```

```
}

// Fungsi untuk menyisipkan node baru setelah node tertentu
void insertAfter(Node*& head, int after, int value) {
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;
    while (temp != nullptr && temp->data != after) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "Data " << after << " tidak ada" << endl;
    }
    cout << "Menambahkan data >> " << value << " setelah " <<
after << endl;
}

// Fungsi untuk menghapus seluruh linked list
void remove(Node*& head) {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

// Fungsi untuk menyisipkan node baru sebelum node tertentu
void insertBefore(Node*& head, int before, int value) {
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;

    if (head != nullptr && head->data == before) {
```

```
            newNode->next = head;
            head = newNode;
            return;
        }

        while (temp != nullptr && temp->next->data != before) {
            temp = temp->next;
        }

        if (temp != nullptr) {
            newNode->next = temp->next;
            temp->next = newNode;
        } else {
            cout << "Data " << before << " tidak ada" << endl;
        }
        cout << "Menambahkan data >> " << value << " sebelum " <<
    before << endl;
    }

    // Fungsi untuk menghapus node pada indeks tertentu
    void deleteNodeAtIndex(Node *&head, int index) {
        cout << "Menghapus node ke-" << index << endl;
        Node* newNode = head;
        Node* bridge = head;
        for (int k = 0; k < index; k++) {
            newNode = newNode->next;
        }
        for (int i = 0; i < index - 1; i++) {
            bridge = bridge->next;
        }
    }

    // Fungsi untuk menghapus node berdasarkan data
    void deleteNodeWithData(Node *&head, int data) {
        Node* newNode = head;
        Node* bridge = new Node();
        while (newNode->next->data != data) {
            newNode = newNode->next;
        }
```

```
        bridge = newNode;
        newNode = newNode->next;
        bridge->next = newNode->next;

        delete newNode;
        cout << "Menghapus node dengan nilai " << data << endl;
    }
```



Gambar 1. Hasil implementasi linked list

PEMBAHASAN

1. Pendahuluan

   Struktur Node untuk merepresentasikan elemen ldalam linked list. Setiap node memiliki data dan pointer yang menunjuk ke node berikutnya. Node tail digunakan untuk menunjuk ke node terakhir dalam linked list.

```
. . .
      struct Node {
          int data;
          Node* next;
      };

Node* tail = NULL; // Inisialisasi pointer ekor
```

```
. . .
```

2. Fungsi – fungsi linked list
   - ‘create’ : menambahkan node bar uke akhir linked list. Jika linked list kosong, node baru menjadi kepala dan ekor. Jika tidak, node baru dutambahkan setelah ekor, dan ekor akan diperbarui.
   - ‘creatAtBeginning’ : menambahkan node baru pada awal linked list
   - ‘display’ menampilkan isi linked list dari kepala hingga akhir
   - ‘updateTail’ : memperbarui pointer ekor untuk menunjuk ke node terakhir dalam linked list.
   - ‘search’ : mencari sebuah nilai dalam linked list dan menampilkan hasil pencarian.
   - ‘insertAfter’ : menyisipkan node baru setelah node tertentu dalam linked list.
   - ‘deleteNodeAtIndex” : menghapus node pada indeks tertentu dalam linked list.
   - ‘deleteNodeWithData’ : menghapus node dari linked list berdasarkan nilai data yang diberikan.
   - ‘remove’ : menghapus semya node dalam linked list dan membebaskan memori yang dialokasikan untuk setiap node.

```
. . .
      int main() {
          Node* head = NULL; // Inisialisasi pointer kepala
      linked list

          // Contoh penggunaan fungsi-fungsi linked list
          create(head, 20);
          display(head);
          create(head, 55);
          display(head);
          create(head, 60);
          display(head);
          create(head, 75);
          display(head);
          insertAfter(head, 55, 77);
          display(head);
          insertBefore(head, 20, 58);
          display(head);
          search(head, 60);
          display(head);
          deleteNodeAtIndex(head, 3);
          display(head);
          deleteNodeWithData(head, 75);
          display(head);
    remove(head); // Menghapus seluruh linked list
. . .
```

3. Fungsi dalam bentuk program

Program	Fungsi
. . . void create(Node*& head, int a) { cout << "Menambahkan data >> " << a;	Menambahkan node bar uke akhir linked list

Node* newNode = new Node();
newNode->data = a;
newNode->next = NULL;
 	
if (head == NULL) {
head = newNode;
tail = head;
} else {
tail->next = newNode;
tail = newNode;
. . .	

. . .
void createAtBeginning(Node*& head, int a) {
Node* newNode = new Node();
newNode->data = a;
newNode->next = NULL;
 	
if (head == NULL) {
head = newNode;
tail = head; // Jika linked list kosong, node baru menjadi kepala dan ekor
} else {
newNode->next = head;
head = newNode; // Node baru menjadi kepala linked list
. . .
	Menambahkan node baru di awal linked list

. . .
void display(Node* head) {
Node* temp = head;
while (temp != NULL) {
cout << temp->data << " ";
temp = temp->next;
}
cout << endl;
}
. . .
	Menampilkan linked list

. . .
void updateTail(Node* head, Node*& tail) {
tail = head;
while (tail->next != NULL) {
tail = tail->next;
}
cout << "\ttail pada : " << tail->data << endl;
}
	Memperbarui pointer ekor

. . .	

. . .
`void search(Node*& head, int i) {`
`    Node* searchNode = head;`
`    while (searchNode->data != i) {`
`        searchNode = searchNode->next;`
`    }`
`    cout << "Pencarian " << searchNode->data << " " << searchNode->next << endl;`
`}`
. . .
	Mencari sebuah nilai dalam linked list

. . .
`void insertAfter(Node*& head, int after, int value) {`
`    Node* newNode = new Node();`
`    newNode->data = value;`
 	
`    Node* temp = head;`
`    while (temp != nullptr && temp->data != after) {`
`        temp = temp->next;`
`    }`
 	
`    if (temp != nullptr) {`
`        newNode->next = temp->next;`
`        temp->next = newNode;`
`    } else {`
`        cout << "Data " << after << " tidak ada" << endl;`
`    }`
`    cout << "Menambahkan data >> " << value << " setelah " << after << endl;`
`}`
. . .
	Menyisipkan node baru setelah node tertentu

. . .
`void remove(Node*& head) {`
`    while (head != nullptr) {`
`        Node* temp = head;`
`        head = head->next;`
`        delete temp;`
`    }`
`}`
. . .
	Menghapus seluruh linked list

. . .
void insertBefore(Node*& head, int before, int value)
{
Node* newNode = new Node();
newNode->data = value;
 	
Node* temp = head;
 	
if (head != nullptr && head->data == before) {
newNode->next = head;
head = newNode;
return;
}
 	
while (temp != nullptr && temp->next->data != before) {
temp = temp->next;
}
 	
if (temp != nullptr) {
newNode->next = temp->next;
temp->next = newNode;
} else {
cout << "Data " << before << " tidak ada" << endl;
}
cout << "Menambahkan data >> " << value << " sebelum " << before << endl;
}
. . .
	Menyisipkan node baru sebelum node tertentu

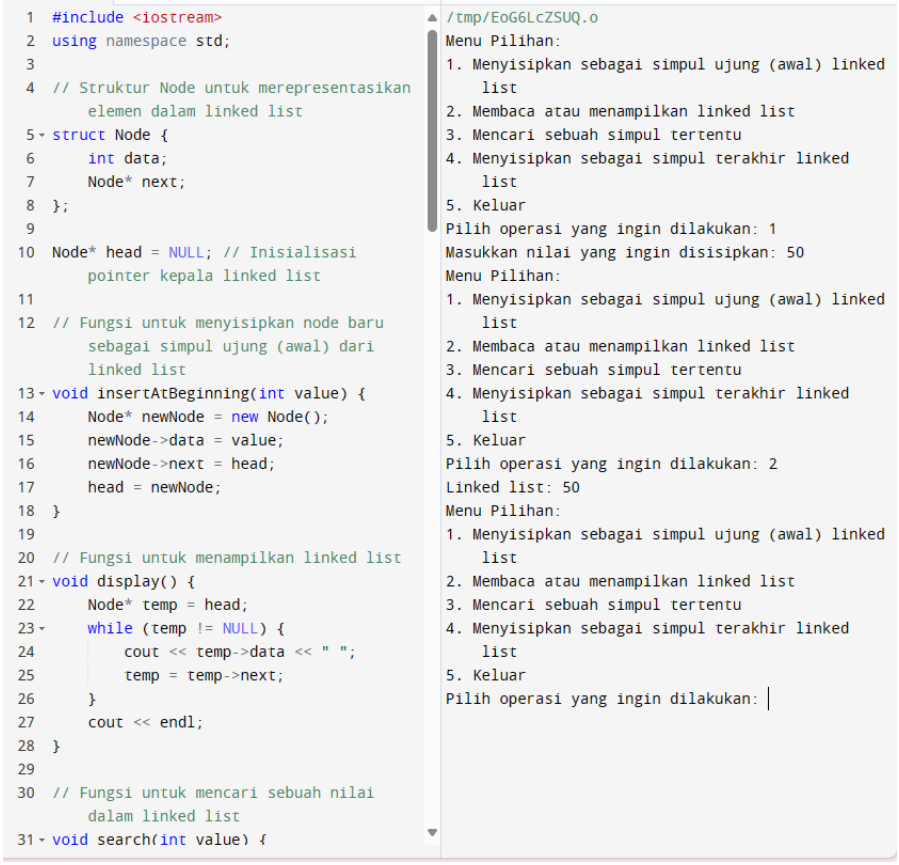
. . .
void deleteNodeAtIndex(Node *&head, int index) {
cout << "Menghapus node ke-" << index << endl;
Node* newNode = head;
Node* bridge = head;
for (int k = 0; k < index; k++) {
newNode = newNode->next;
}
for (int i = 0; i < index - 1; i++) {
bridge = bridge->next;
}
}
	Menghapus node pada indeks tertentu

. . .	

. . .
void deleteNodeWithData(Node *&head, int data) {
    Node* newNode = head;
    Node* bridge = new Node();
    while (newNode->next->data != data) {
        newNode = newNode->next;
    }
    bridge = newNode;
    newNode = newNode->next;
    bridge->next = newNode->next;
 	
    delete newNode;
    cout << "Menghapus node dengan nilai " << data << endl;
}
. . .
	Menghapus node berdasarkan data

PERCOBAAN

1. Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul ujung(awal) dari linked list.
2. Implementasikan operasi dasar Single linked list : Membaca atau menampilkan
3. Implementasikan operasi dasar Single linked list : Mencari sebuah simpul tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.
4. Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul terakhir
5. Gabungkan semua operasi di atas dalam sebuah Menu Pilihan.



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

// Struktur Node untuk merepresentasikan elemen dalam linked list
struct Node {
    int data;
    Node* next;
};

Node* head = NULL; // Inisialisasi pointer kepala linked list

// Fungsi untuk menyisipkan node baru sebagai simpul ujung (awal) dari linked list
void insertAtBeginning(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menampilkan linked list
void display() {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Fungsi untuk mencari sebuah nilai dalam linked list
void search(int value) {
```

```
/tmp/EoG6LcZSUQ.o
Menu Pilihan:
1. Menyisipkan sebagai simpul ujung (awal) linked list
2. Membaca atau menampilkan linked list
3. Mencari sebuah simpul tertentu
4. Menyisipkan sebagai simpul terakhir linked list
5. Keluar
Pilih operasi yang ingin dilakukan: 1
Masukkan nilai yang ingin disisipkan: 50
Menu Pilihan:
1. Menyisipkan sebagai simpul ujung (awal) linked list
2. Membaca atau menampilkan linked list
3. Mencari sebuah simpul tertentu
4. Menyisipkan sebagai simpul terakhir linked list
5. Keluar
Pilih operasi yang ingin dilakukan: 2
Linked list: 50
Menu Pilihan:
1. Menyisipkan sebagai simpul ujung (awal) linked list
2. Membaca atau menampilkan linked list
3. Mencari sebuah simpul tertentu
4. Menyisipkan sebagai simpul terakhir linked list
5. Keluar
Pilih operasi yang ingin dilakukan:
```

Gambar 2. Hasil Percobaan Linked List

Program Utuh

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

// Struktur Node untuk merepresentasikan elemen dalam linked list
struct Node {
    int data;
    Node* next;
};

Node* head = NULL; // Inisialisasi pointer kepala linked list

// Fungsi untuk menyisipkan node baru sebagai simpul ujung (awal) dari linked list
void insertAtBeginning(int value) {
    Node* newNode = new Node();
```

```
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }

    // Fungsi untuk menampilkan linked list
    void display() {
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // Fungsi untuk mencari sebuah nilai dalam linked list
    void search(int value) {
        Node* searchNode = head;
        while (searchNode != NULL && searchNode->data != value) {
            searchNode = searchNode->next;
        }
        if (searchNode != NULL) {
            cout << "Nilai " << value << " ditemukan dalam linked
    list" << endl;
        } else {
            cout << "Nilai " << value << " tidak ditemukan dalam
    linked list" << endl;
        }
    }

    // Fungsi untuk menyisipkan node baru sebagai simpul terakhir
    void insertAtEnd(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }

    // Fungsi untuk menampilkan menu pilihan
    void displayMenu() {
        cout << "Menu Pilihan:" << endl;
        cout << "1. Menyisipkan sebagai simpul ujung (awal) linked
    list" << endl;
```

```
        cout << "2. Membaca atau menampilkan linked list" << endl;
        cout << "3. Mencari sebuah simpul tertentu" << endl;
        cout << "4. Menyisipkan sebagai simpul terakhir linked
    list" << endl;
        cout << "5. Keluar" << endl;
    }

    int main() {
        int choice;
        do {
            displayMenu();
            cout << "Pilih operasi yang ingin dilakukan: ";
            cin >> choice;

            switch (choice) {
                case 1: {
                    int value;
                    cout << "Masukkan nilai yang ingin disisipkan:
    ";
                    cin >> value;
                    insertAtBeginning(value);
                    break;
                }
                case 2: {
                    cout << "Linked list: ";
                    display();
                    break;
                }
                case 3: {
                    int value;
                    cout << "Masukkan nilai yang ingin dicari: ";
                    cin >> value;
                    search(value);
                    break;
                }
                case 4: {
                    int value;
                    cout << "Masukkan nilai yang ingin disisipkan:
    ";
                    cin >> value;
                    insertAtEnd(value);
                    break;
                }
                case 5:
                    cout << "Program selesai." << endl;
                    break;
                default:
                    cout << "Pilihan tidak valid. Silakan pilih
    lagi." << endl;
            }
        } while (choice != 5);
```

```
            return 0;
}
```