SOURCE CODE CONTOH LINKED LIST SEDERHANA

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
   int data;      // Data di dalam node
   Node* next;    // Pointer ke node berikutnya

   // Konstruktor untuk mempermudah pembuatan node baru
   Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
   Node* head;    // Pointer ke node pertama dalam linked list

public:
   // Konstruktor untuk membuat linked list kosong
   LinkedList() : head(nullptr) {}

   // Destructor untuk membersihkan memori
   ~LinkedList() {
      Node* current = head;
      Node* nextNode;
      while (current != nullptr) {
         nextNode = current->next;
         delete current;
         current = nextNode;
      }
   }

   // Fungsi untuk menyisipkan node di depan linked list
   void insertFront(int value) {
      Node* newNode = new Node(value);
      newNode->next = head;
      head = newNode;
   }

   // Fungsi untuk menampilkan linked list
   void display() const {
      Node* current = head;
      while (current != nullptr) {
         std::cout << current->data << " ";
         current = current->next;
      }
      std::cout << std::endl;
```

```
    }
};

int main() {
    LinkedList list;

    // Menyisipkan beberapa elemen di linked list
    list.insertFront(30);
    list.insertFront(20);
    list.insertFront(10);

    // Menampilkan linked list
    list.display(); // Output: 10 20 30

    return 0;
}
```

SOURCE CODE CONTOH LINKED LIST SEDERHANA (INSERT DIDEPAN)

```
#include <iostream>

class Node {
public:
    int data;
    Node* next;

    Node(int data) : data(data), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}

    // Menambahkan node di depan linked list
    void insertAtFront(int data) {
        Node* new_node = new Node(data);
        new_node->next = head;
        head = new_node;
    }

    // Menampilkan seluruh isi linked list
    void display() const {
        Node* current = head;
        while (current != nullptr) {
```

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

    ~LinkedList() {
        Node* current = head;
        while (current != nullptr) {
            Node* next_node = current->next;
            delete current;
            current = next_node;
        }
    }
};

// Contoh penggunaan
int main() {
    LinkedList ll;
    ll.insertAtFront(3);
    ll.insertAtFront(2);
    ll.insertAtFront(1);

    std::cout << "Linked List:" << std::endl;
    ll.display();  // Output: 1 -> 2 -> 3

    return 0;
}
```

- data menyimpan data yang dibawa oleh node.
- next adalah pointer ke node berikutnya.
- insertAtFront(int data): Membuat node baru dengan data yang diberikan, dan menyisipkannya di depan linked list dengan mengatur next dari node baru ke head saat ini dan memperbarui head menjadi node baru.

SOURCE CODE CONTOH LINKED LIST SEDERHANA (INSERT DIBELAKANG)

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
   int data;      // Data di dalam node
   Node* next;    // Pointer ke node berikutnya

   // Konstruktor untuk mempermudah pembuatan node baru
   Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
   Node* head;    // Pointer ke node pertama dalam linked list

public:
   // Konstruktor untuk membuat linked list kosong
   LinkedList() : head(nullptr) {}

   // Destructor untuk membersihkan memori
   ~LinkedList() {
      Node* current = head;
      Node* nextNode;
      while (current != nullptr) {
         nextNode = current->next;
         delete current;
         current = nextNode;
      }
   }

   // Fungsi untuk menyisipkan node di akhir linked list
   void insertBack(int value) {
      Node* newNode = new Node(value);

      // Jika linked list kosong, node baru menjadi head
      if (head == nullptr) {
         head = newNode;
      } else {
         // Jika tidak kosong, iterasi sampai node terakhir
         Node* current = head;
         while (current->next != nullptr) {
            current = current->next;
         }
         // Menyambungkan node baru di akhir list
         current->next = newNode;
```

```
        }
    }

    // Fungsi untuk menampilkan linked list
    void display() const {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    LinkedList list;

    // Menyisipkan beberapa elemen di akhir linked list
    list.insertBack(10);
    list.insertBack(20);
    list.insertBack(30);

    // Menampilkan linked list
    list.display(); // Output: 10 20 30

    return 0;
}
```

- **Konstruktor:** Menginisialisasi linked list kosong.
- **Destruktor:** Membersihkan memori yang digunakan oleh linked list untuk menghindari memory leak.
- **Fungsi insertBack:** Menyisipkan node baru di akhir list. Jika list kosong, node baru menjadi head. Jika tidak, kita iterasi sampai node terakhir dan menyambungkan node baru di akhir.
- **Fungsi display:** Menampilkan elemen-elemen di linked list

SOURCE CODE CONTOH LINKED LIST SEDERHANA (INSERT TENGAH)

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
    int data;      // Data di dalam node
    Node* next;    // Pointer ke node berikutnya

    // Konstruktor untuk mempermudah pembuatan node baru
```

```
    Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
    Node* head;     // Pointer ke node pertama dalam linked list

public:
    // Konstruktor untuk membuat linked list kosong
    LinkedList() : head(nullptr) {}

    // Destructor untuk membersihkan memori
    ~LinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }

    // Fungsi untuk menyisipkan node pada posisi tertentu
    void insertAt(int position, int value) {
        Node* newNode = new Node(value);

        // Jika posisi 0, menyisipkan di depan (head)
        if (position == 0) {
            newNode->next = head;
            head = newNode;
            return;
        }

        // Iterasi untuk menemukan posisi sebelum posisi yang diinginkan
        Node* current = head;
        for (int i = 0; current != nullptr && i < position - 1; ++i) {
            current = current->next;
        }

        // Jika posisi di luar batas, tidak menyisipkan
        if (current == nullptr) {
            std::cout << "Posisi di luar batas!" << std::endl;
            delete newNode;
            return;
        }

        // Menyisipkan node baru di posisi yang diinginkan
```

```
            newNode->next = current->next;
            current->next = newNode;
        }

        // Fungsi untuk menampilkan linked list
        void display() const {
            Node* current = head;
            while (current != nullptr) {
                std::cout << current->data << " ";
                current = current->next;
            }
            std::cout << std::endl;
        }
    };

    int main() {
        LinkedList list;

        // Menyisipkan beberapa elemen di linked list
        list.insertAt(0, 10);  // Menyisipkan 10 di posisi 0
        list.insertAt(1, 20);  // Menyisipkan 20 di posisi 1
        list.insertAt(1, 15);  // Menyisipkan 15 di posisi 1

        // Menampilkan linked list
        list.display(); // Output: 10 15 20

        return 0;
}
```

- **Konstruktor:** Menginisialisasi linked list kosong.
- **Destruktor:** Membersihkan memori yang digunakan oleh linked list untuk menghindari memory leak.
- **Fungsi insertAt:** Menyisipkan node baru di posisi tertentu.
  - Jika posisi adalah 0, menyisipkan node di depan (head).
  - Jika tidak, iterasi melalui linked list untuk menemukan posisi sebelum posisi yang diinginkan, kemudian menyisipkan node baru di posisi yang diinginkan.
- **Fungsi display:** Menampilkan elemen-elemen di linked list.

SOURCE CODE CONTOH LINKED LIST SEDERHANA (HAPUS DIDEPAN)

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
   int data;      // Data di dalam node
   Node* next;    // Pointer ke node berikutnya

   // Konstruktor untuk mempermudah pembuatan node baru
   Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
   Node* head;    // Pointer ke node pertama dalam linked list

public:
   // Konstruktor untuk membuat linked list kosong
   LinkedList() : head(nullptr) {}

   // Destructor untuk membersihkan memori
   ~LinkedList() {
      Node* current = head;
      Node* nextNode;
      while (current != nullptr) {
         nextNode = current->next;
         delete current;
         current = nextNode;
      }
   }

   // Fungsi untuk menyisipkan node di depan linked list
   void insertFront(int value) {
      Node* newNode = new Node(value);
      newNode->next = head;
      head = newNode;
   }

   // Fungsi untuk menghapus node dari depan linked list
   void deleteFront() {
      if (head == nullptr) {
         std::cout << "Linked list kosong!" << std::endl;
         return;
      }

      Node* temp = head;
```

```
        head = head->next;  // Memindahkan head ke node berikutnya
        delete temp;        // Menghapus node lama
    }

    // Fungsi untuk menampilkan linked list
    void display() const {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    LinkedList list;

    // Menyisipkan beberapa elemen di linked list
    list.insertFront(30); // Menyisipkan 30 di depan
    list.insertFront(20); // Menyisipkan 20 di depan
    list.insertFront(10); // Menyisipkan 10 di depan

    // Menampilkan linked list sebelum penghapusan
    std::cout << "Linked list sebelum penghapusan: ";
    list.display(); // Output: 10 20 30

    // Menghapus elemen dari depan
    list.deleteFront();

    // Menampilkan linked list setelah penghapusan
    std::cout << "Linked list setelah penghapusan: ";
    list.display(); // Output: 20 30

    return 0;
}
```

- **Konstruktor:** Menginisialisasi linked list kosong.
- **Destruktor:** Membersihkan memori yang digunakan oleh linked list untuk menghindari memory leak.
- **Fungsi insertFront:** Menyisipkan node baru di depan linked list.
- **Fungsi deleteFront:** Menghapus node dari depan linked list. Jika list kosong, fungsi ini hanya mencetak pesan. Jika tidak kosong, fungsi ini memindahkan head ke node berikutnya dan menghapus node lama.
- **Fungsi display:** Menampilkan elemen-elemen di linked list.

SOURCE CODE CONTOH LINKED LIST SEDERHANA (HAPUS DIBELAKANG)

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
   int data;      // Data di dalam node
   Node* next;    // Pointer ke node berikutnya

   // Konstruktor untuk mempermudah pembuatan node baru
   Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
   Node* head;    // Pointer ke node pertama dalam linked list

public:
   // Konstruktor untuk membuat linked list kosong
   LinkedList() : head(nullptr) {}

   // Destructor untuk membersihkan memori
   ~LinkedList() {
      Node* current = head;
      Node* nextNode;
      while (current != nullptr) {
         nextNode = current->next;
         delete current;
         current = nextNode;
      }
   }

   // Fungsi untuk menyisipkan node di depan linked list
   void insertFront(int value) {
      Node* newNode = new Node(value);
      newNode->next = head;
      head = newNode;
   }

   // Fungsi untuk menghapus node dari belakang linked list
   void deleteBack() {
      if (head == nullptr) {
         std::cout << "Linked list kosong!" << std::endl;
         return;
      }

      // Jika hanya ada satu elemen
```

```
        if (head->next == nullptr) {
            delete head;
            head = nullptr;
            return;
        }

        // Iterasi untuk menemukan node sebelum node terakhir
        Node* current = head;
        while (current->next->next != nullptr) {
            current = current->next;
        }

        // Menghapus node terakhir
        Node* temp = current->next;
        current->next = nullptr; // Menghapus sambungan ke node terakhir
        delete temp;
    }

    // Fungsi untuk menampilkan linked list
    void display() const {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main() {
    LinkedList list;

    // Menyisipkan beberapa elemen di linked list
    list.insertFront(30); // Menyisipkan 30 di depan
    list.insertFront(20); // Menyisipkan 20 di depan
    list.insertFront(10); // Menyisipkan 10 di depan

    // Menampilkan linked list sebelum penghapusan
    std::cout << "Linked list sebelum penghapusan: ";
    list.display(); // Output: 10 20 30

    // Menghapus elemen dari belakang
    list.deleteBack();

    // Menampilkan linked list setelah penghapusan
    std::cout << "Linked list setelah penghapusan: ";
    list.display(); // Output: 10 20
```

```
    return 0;
}
```

- Konstruktor: Menginisialisasi linked list kosong.
- Destruktor: Membersihkan memori yang digunakan oleh linked list untuk menghindari memory leak.
- Fungsi insertFront: Menyisipkan node baru di depan linked list.
- Fungsi deleteBack: Menghapus node dari belakang linked list.
    - Jika list kosong, fungsi ini mencetak pesan dan keluar.
    - Jika hanya ada satu elemen, fungsi ini menghapus elemen tersebut dan mengatur head ke nullptr.
    - Jika ada lebih dari satu elemen, fungsi ini iterasi untuk menemukan node sebelum node terakhir, lalu menghapus node terakhir.
- Fungsi display: Menampilkan elemen-elemen di linked list.

SOURCE CODE CONTOH LINKED LIST SEDERHANA (HAPUS TENGAH)

```
#include <iostream>

// Struktur untuk node linked list
struct Node {
    int data;       // Data di dalam node
    Node* next;     // Pointer ke node berikutnya

    // Konstruktor untuk mempermudah pembuatan node baru
    Node(int value) : data(value), next(nullptr) {}
};

// Kelas LinkedList
class LinkedList {
private:
    Node* head;     // Pointer ke node pertama dalam linked list

public:
    // Konstruktor untuk membuat linked list kosong
    LinkedList() : head(nullptr) {}

    // Destructor untuk membersihkan memori
    ~LinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
```

```
        current = nextNode;
    }
}

// Fungsi untuk menyisipkan node di depan linked list
void insertFront(int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menghapus node dari posisi tertentu
void deleteAt(int position) {
    if (head == nullptr) {
        std::cout << "Linked list kosong!" << std::endl;
        return;
    }

    // Jika posisi 0, hapus node dari depan
    if (position == 0) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    // Iterasi untuk menemukan node sebelum posisi yang diinginkan
    Node* current = head;
    for (int i = 0; current != nullptr && i < position - 1; ++i) {
        current = current->next;
    }

    // Jika posisi di luar batas, tidak hapus
    if (current == nullptr || current->next == nullptr) {
        std::cout << "Posisi di luar batas!" << std::endl;
        return;
    }

    // Menghapus node dari posisi yang diinginkan
    Node* temp = current->next;
    current->next = current->next->next;
    delete temp;
}

// Fungsi untuk menampilkan linked list
void display() const {
    Node* current = head;
    while (current != nullptr) {
```

```
        std::cout << current->data << " ";
        current = current->next;
      }
      std::cout << std::endl;
   }
};

int main() {
   LinkedList list;

   // Menyisipkan beberapa elemen di linked list
   list.insertFront(40); // Menyisipkan 40 di depan
   list.insertFront(30); // Menyisipkan 30 di depan
   list.insertFront(20); // Menyisipkan 20 di depan
   list.insertFront(10); // Menyisipkan 10 di depan

   // Menampilkan linked list sebelum penghapusan
   std::cout << "Linked list sebelum penghapusan: ";
   list.display(); // Output: 10 20 30 40

   // Menghapus elemen dari posisi tertentu
   list.deleteAt(2); // Menghapus elemen di posisi 2 (30)

   // Menampilkan linked list setelah penghapusan
   std::cout << "Linked list setelah penghapusan: ";
   list.display(); // Output: 10 20 40

   return 0;
}
```

- Konstruktor: Menginisialisasi linked list kosong.
- Destruktor: Membersihkan memori yang digunakan oleh linked list untuk menghindari memory leak.
- Fungsi insertFront: Menyisipkan node baru di depan linked list.
- Fungsi deleteAt: Menghapus node dari posisi tertentu.
    - Jika posisi adalah 0, fungsi ini menghapus node dari depan.
    - Jika tidak, fungsi ini iterasi untuk menemukan node sebelum posisi yang diinginkan, kemudian menghapus node pada posisi yang diinginkan.
    - Jika posisi di luar batas, fungsi ini mencetak pesan dan tidak melakukan penghapusan.
- Fungsi display: Menampilkan elemen-elemen di linked list.