SOURCE CODE DOUBLE LINKED LIST SEDERHANA

```
#include <iostream>

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

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

// Kelas DoublyLinkedList
class DoublyLinkedList {
private:
   Node* head;    // Pointer ke node pertama dalam doubly linked list
   Node* tail;    // Pointer ke node terakhir dalam doubly linked list

public:
   // Konstruktor untuk membuat doubly linked list kosong
   DoublyLinkedList() : head(nullptr), tail(nullptr) {}

   // Destructor untuk membersihkan memori
   ~DoublyLinkedList() {
      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);
      if (head == nullptr) {
         head = tail = newNode; // Jika list kosong, head dan tail adalah node baru
      } else {
         newNode->next = head;
         head->prev = newNode;
         head = newNode;
      }
   }

   // Fungsi untuk menyisipkan node di belakang linked list
```

```
    void insertBack(int value) {
        Node* newNode = new Node(value);
        if (tail == nullptr) {
            head = tail = newNode; // Jika list kosong, head dan tail adalah node baru
        } else {
            newNode->prev = tail;
            tail->next = newNode;
            tail = newNode;
        }
    }

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

    // Fungsi untuk menampilkan linked list dari belakang
    void displayBackward() const {
        Node* current = tail;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->prev;
        }
        std::cout << std::endl;
    }
};

int main() {
    DoublyLinkedList list;

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

    // Menampilkan linked list dari depan
    std::cout << "Linked list (forward): ";
    list.displayForward(); // Output: 10 20 30

    // Menampilkan linked list dari belakang
    std::cout << "Linked list (backward): ";
    list.displayBackward(); // Output: 30 20 10
```

```
    return 0;
}
```

Struktur Node:

- Menyimpan data dan memiliki dua pointer: next untuk node berikutnya dan prev untuk node sebelumnya.

Kelas Double Linked List:

- Konstruktor: Inisialisasi list kosong dengan head dan tail diatur ke nullptr.
- Destruktor: Membersihkan memori dengan menghapus semua node dalam list.
- Fungsi insertFront(int value): Menyisipkan node baru di depan list.
- Fungsi insertBack(int value): Menyisipkan node baru di belakang list.
- Fungsi displayForward(): Menampilkan elemen-elemen list dari depan ke belakang.
- Fungsi displayBackward(): Menampilkan elemen-elemen list dari belakang ke depan.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (INSERT DEPAN)

```
#include <iostream>

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

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

// Kelas DoublyLinkedList
class DoublyLinkedList {
private:
    Node* head;     // Pointer ke node pertama dalam doubly linked list
    Node* tail;     // Pointer ke node terakhir dalam doubly linked list

public:
    // Konstruktor untuk membuat doubly linked list kosong
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Destructor untuk membersihkan memori
    ~DoublyLinkedList() {
```

```
        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); // Membuat node baru
        if (head == nullptr) {
            // Jika list kosong, node baru adalah head dan tail
            head = tail = newNode;
        } else {
            // Jika list tidak kosong, tambahkan node baru di depan
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

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

    // Fungsi untuk menampilkan linked list dari belakang
    void displayBackward() const {
        Node* current = tail;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->prev;
        }
        std::cout << std::endl;
    }
};

int main() {
    DoublyLinkedList list;

    // Menyisipkan beberapa elemen di depan 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 dari depan
    std::cout << "Linked list (forward): ";
    list.displayForward(); // Output: 10 20 30

    // Menampilkan linked list dari belakang
    std::cout << "Linked list (backward): ";
    list.displayBackward(); // Output: 30 20 10

    return 0;
}
```

- Konstruktor: Menginisialisasi list kosong dengan head dan tail diatur ke nullptr.
- Destruktor: Membersihkan memori dengan menghapus semua node dalam list untuk menghindari memory leak.
- Fungsi insertFront(int value):
    - Membuat node baru dengan nilai yang diberikan.
    - Jika list kosong (head adalah nullptr), node baru menjadi head dan tail.
    - Jika list tidak kosong, node baru ditambahkan di depan dengan memperbarui pointer next dari node baru dan pointer prev dari node yang sebelumnya menjadi head.
- Fungsi displayForward(): Menampilkan elemen-elemen dari head hingga tail.
- Fungsi displayBackward(): Menampilkan elemen-elemen dari tail hingga head.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (INSERT TENGAH)

```
#include <iostream>

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

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

class DoublyLinkedList {
public:
    Node* head;
```

```
    DoublyLinkedList() : head(nullptr) {}

    // Menambahkan node di akhir list
    void append(int data) {
        Node* newNode = new Node(data);
        if (head == nullptr) {
            head = newNode;
            return;
        }

        Node* last = head;
        while (last->next != nullptr) {
            last = last->next;
        }
        last->next = newNode;
        newNode->prev = last;
    }

    // Menyisipkan node di tengah list
    void insertMiddle(int data) {
        Node* newNode = new Node(data);
        if (head == nullptr) {
            head = newNode;
            return;
        }

        Node* slow = head;
        Node* fast = head;

        // Temukan node tengah menggunakan pointer lambat dan cepat
        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
        }

        // Sisipkan node baru di tengah
        if (slow->next != nullptr) {
            newNode->next = slow->next;
            slow->next->prev = newNode;
        }
        newNode->prev = slow;
        slow->next = newNode;
    }

    // Mencetak elemen dalam list
    void printList() {
        Node* current = head;
        while (current != nullptr) {
```

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

  ~DoublyLinkedList() {
    // Hapus semua node untuk menghindari kebocoran memori
    Node* current = head;
    Node* nextNode;
    while (current != nullptr) {
      nextNode = current->next;
      delete current;
      current = nextNode;
    }
  }
};

// Contoh penggunaan
int main() {
  DoublyLinkedList dll;
  dll.append(1);
  dll.append(3);
  dll.append(5);
  dll.append(7);

  std::cout << "Original list:" << std::endl;
  dll.printList();

  dll.insertMiddle(4);  // Sisipkan di tengah list

  std::cout << "List after inserting 4 in the middle:" << std::endl;
  dll.printList();

  return 0;
}
```

- append(int data): Menambahkan node baru di akhir daftar.
- insertMiddle(int data): Menyisipkan node baru di tengah daftar. Metode ini menggunakan dua pointer (slow dan fast) untuk menemukan posisi tengah dan menyisipkan node baru di sana.
- printList(): Mencetak elemen-elemen dalam daftar dari awal hingga akhir.
- Destructor ~DoublyLinkedList(): Menghapus semua node untuk mencegah kebocoran memori.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (INSERT BELAKANG)

```
#include <iostream>

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

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

class DoublyLinkedList {
public:
   Node* head;

   DoublyLinkedList() : head(nullptr) {}

   // Menambahkan node di akhir list
   void append(int data) {
      Node* newNode = new Node(data);
      if (head == nullptr) {
         head = newNode;
         return;
      }

      Node* last = head;
      while (last->next != nullptr) {
         last = last->next;
      }
      last->next = newNode;
      newNode->prev = last;
   }

   // Menyisipkan node di akhir list
   void insertBack(int data) {
      append(data);  // Karena kita hanya menambahkan di akhir, kita bisa memanggil append
   }

   // Mencetak elemen dalam list
   void printList() const {
      Node* current = head;
      while (current != nullptr) {
         std::cout << current->data << " ";
         current = current->next;
      }
      std::cout << std::endl;
```

```
    }

    ~DoublyLinkedList() {
        // Hapus semua node untuk menghindari kebocoran memori
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

// Contoh penggunaan
int main() {
    DoublyLinkedList dll;
    dll.append(1);
    dll.append(3);
    dll.append(5);
    dll.append(7);

    std::cout << "Original list:" << std::endl;
    dll.printList();

    dll.insertBack(9);  // Sisipkan di akhir list

    std::cout << "List after inserting 9 at the end:" << std::endl;
    dll.printList();

    return 0;
}
```

- append(int data): Menambahkan node baru di akhir daftar.
- insertBack(int data): Menyisipkan node baru di akhir daftar. Dalam hal ini, insertBack hanya memanggil append karena fungsionalitasnya sama.
- printList(): Mencetak elemen-elemen dalam daftar dari awal hingga akhir.
- Destructor ~DoublyLinkedList(): Menghapus semua node untuk mencegah kebocoran memori.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (HAPUS BELAKANG)

```
#include <iostream>

class Node {
public:
    int data;
```

```
    Node* next;
    Node* prev;

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

class DoublyLinkedList {
public:
    Node* head;

    DoublyLinkedList() : head(nullptr) {}

    // Menambahkan node di akhir list
    void append(int data) {
        Node* newNode = new Node(data);
        if (head == nullptr) {
            head = newNode;
            return;
        }

        Node* last = head;
        while (last->next != nullptr) {
            last = last->next;
        }
        last->next = newNode;
        newNode->prev = last;
    }

    // Menghapus node di akhir list
    void deleteBack() {
        if (head == nullptr) {
            std::cout << "List is empty. Nothing to delete." << std::endl;
            return;
        }

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

        Node* last = head;
        while (last->next != nullptr) {
            last = last->next;
        }

        Node* secondLast = last->prev;
        secondLast->next = nullptr;
```

```
        delete last;
    }

    // Mencetak elemen dalam list
    void printList() const {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }

    ~DoublyLinkedList() {
        // Hapus semua node untuk menghindari kebocoran memori
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

// Contoh penggunaan
int main() {
    DoublyLinkedList dll;
    dll.append(1);
    dll.append(3);
    dll.append(5);
    dll.append(7);

    std::cout << "Original list:" << std::endl;
    dll.printList();

    dll.deleteBack();  // Hapus node di akhir list

    std::cout << "List after deleting the last node:" << std::endl;
    dll.printList();

    return 0;
}
```

- append(int data): Menambahkan node baru di akhir daftar.
- deleteBack(): Menghapus node terakhir dari daftar.
- Jika daftar kosong (head == nullptr), maka tidak ada yang dihapus.
- Jika hanya ada satu node di daftar (head->next == nullptr), hapus node tersebut dan atur head menjadi nullptr.
- Jika ada lebih dari satu node, temukan node terakhir, atur prev dari node terakhir menjadi nullptr untuk node kedua terakhir, dan hapus node terakhir.

printList(): Mencetak elemen-elemen dalam daftar dari awal hingga akhir.

Destructor ~DoublyLinkedList(): Menghapus semua node untuk mencegah kebocoran memori.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (HAPUS TENGAH)

```
#include <iostream>

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

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

class DoublyLinkedList {
public:
   Node* head;

   DoublyLinkedList() : head(nullptr) {}

   // Menambahkan node di akhir list
   void append(int data) {
      Node* newNode = new Node(data);
      if (head == nullptr) {
         head = newNode;
         return;
      }

      Node* last = head;
      while (last->next != nullptr) {
         last = last->next;
      }
      last->next = newNode;
      newNode->prev = last;
   }
```

```
// Menghapus node di tengah list
void deleteMiddle() {
    if (head == nullptr) {
        std::cout << "List is empty. Nothing to delete." << std::endl;
        return;
    }

    Node* slow = head;
    Node* fast = head;

    // Temukan node tengah menggunakan pointer lambat dan cepat
    while (fast != nullptr && fast->next != nullptr && fast->next->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // Hapus node tengah
    Node* middle = slow;
    if (middle == nullptr) {
        std::cout << "List does not have enough elements to delete." << std::endl;
        return;
    }

    if (middle->prev != nullptr) {
        middle->prev->next = middle->next;
    } else {
        // Jika node tengah adalah node pertama
        head = middle->next;
    }

    if (middle->next != nullptr) {
        middle->next->prev = middle->prev;
    }

    delete middle;
}

// Mencetak elemen dalam list
void printList() const {
    Node* current = head;
    while (current != nullptr) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}
```

```
    ~DoublyLinkedList() {
        // Hapus semua node untuk menghindari kebocoran memori
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

// Contoh penggunaan
int main() {
    DoublyLinkedList dll;
    dll.append(1);
    dll.append(3);
    dll.append(5);
    dll.append(7);
    dll.append(9);

    std::cout << "Original list:" << std::endl;
    dll.printList();

    dll.deleteMiddle();  // Hapus node di tengah list

    std::cout << "List after deleting the middle node:" << std::endl;
    dll.printList();

    return 0;
}
```

- append(int data): Menambahkan node baru di akhir daftar.
- deleteMiddle(): Menghapus node yang berada di tengah daftar.
- Menggunakan teknik dua pointer (slow dan fast) untuk menemukan node tengah.
- Jika node tengah ditemukan, hapus node tersebut dengan mengubah referensi dari node sebelum dan setelahnya. Jika node tengah adalah node pertama, perbarui head ke node berikutnya.
- printList(): Mencetak elemen-elemen dalam daftar dari awal hingga akhir.
- Destructor ~DoublyLinkedList(): Menghapus semua node untuk mencegah kebocoran memori.

SOURCE CODE DOUBLE LINKED LIST SEDERHANA (HAPUS DEPAN)

```
#include <iostream>

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

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

class DoublyLinkedList {
public:
   Node* head;

   DoublyLinkedList() : head(nullptr) {}

   // Menambahkan node di akhir list
   void append(int data) {
      Node* newNode = new Node(data);
      if (head == nullptr) {
         head = newNode;
         return;
      }

      Node* last = head;
      while (last->next != nullptr) {
         last = last->next;
      }
      last->next = newNode;
      newNode->prev = last;
   }

   // Menghapus node di awal list
   void deleteFront() {
      if (head == nullptr) {
         std::cout << "List is empty. Nothing to delete." << std::endl;
         return;
      }

      Node* temp = head;
      head = head->next;

      if (head != nullptr) {
         head->prev = nullptr;
      }
```

```
        delete temp;
    }

    // Mencetak elemen dalam list
    void printList() const {
        Node* current = head;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }

    ~DoublyLinkedList() {
        // Hapus semua node untuk menghindari kebocoran memori
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

// Contoh penggunaan
int main() {
    DoublyLinkedList dll;
    dll.append(1);
    dll.append(3);
    dll.append(5);
    dll.append(7);

    std::cout << "Original list:" << std::endl;
    dll.printList();

    dll.deleteFront();  // Hapus node di awal list

    std::cout << "List after deleting the front node:" << std::endl;
    dll.printList();

    return 0;
}
```

- append(int data): Menambahkan node baru di akhir daftar.
- deleteFront(): Menghapus node pertama dari daftar.

- Jika daftar kosong (head == nullptr), maka tidak ada yang dihapus.
- Jika ada node, hapus node pertama dan atur head ke node berikutnya.
- Jika setelah menghapus node pertama, daftar tidak kosong, atur prev dari node berikutnya menjadi nullptr.
- printList(): Mencetak elemen-elemen dalam daftar dari awal hingga akhir.
- Destructor ~DoublyLinkedList(): Menghapus semua node untuk mencegah kebocoran memori.