SOURCE CODE CSLL INSERT DEPAN

```
#include <iostream>

using namespace std;

// Struktur node untuk daftar terhubung
struct Node {
    int data;
    Node* next;

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

// Kelas untuk daftar terhubung melingkar
class CircularLinkedList {
private:
    Node* head;

public:
    // Konstruktor
    CircularLinkedList() : head(nullptr) {}

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

    // Fungsi untuk menyisipkan elemen di depan
    void insertFront(int data) {
        Node* newNode = new Node(data);

        if (!head) {
            // Jika list kosong, node ini akan menunjuk ke dirinya sendiri
            head = newNode;
            newNode->next = head;
        } else {
            // Menyisipkan node baru di depan dan mengubah pointer
            newNode->next = head;
```

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

  // Fungsi untuk menampilkan elemen list
  void display() const {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
  }
};

int main() {
  CircularLinkedList cll;

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

  // Menampilkan elemen list
  cout << "Circular Linked List elements: ";
  cll.display();

  return 0;
}
```

SOURCE CODE CSLL INSERT TENGAH

```
#include <iostream>

using namespace std;

// Struktur node untuk daftar terhubung
```

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

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

// Kelas untuk daftar terhubung melingkar
class CircularLinkedList {
private:
   Node* head;

public:
   // Konstruktor
   CircularLinkedList() : head(nullptr) {}

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

   // Fungsi untuk menyisipkan elemen di tengah
   void insertMiddle(int data) {
      Node* newNode = new Node(data);

      if (!head) {
         // Jika list kosong, node ini akan menunjuk ke dirinya sendiri
         head = newNode;
         newNode->next = head;
      } else if (head->next == head) {
         // Jika hanya ada satu node di list, sisipkan setelah head
         newNode->next = head->next;
         head->next = newNode;
      } else {
         // Menyisipkan node baru di tengah
         Node* slow = head;
         Node* fast = head;
         // Menemukan posisi di tengah menggunakan dua pointer
         do {
```

```
            fast = fast->next;
            if (fast != head) {
               fast = fast->next;
            }
            if (fast == head || fast->next == head) {
               break;
            }
            slow = slow->next;
         } while (fast != head);

         // Sisipkan node baru di posisi di depan slow
         newNode->next = slow->next;
         slow->next = newNode;
      }
   }

   // Fungsi untuk menampilkan elemen list
   void display() const {
      if (!head) {
         cout << "List is empty" << endl;
         return;
      }

      Node* temp = head;
      do {
         cout << temp->data << " ";
         temp = temp->next;
      } while (temp != head);
      cout << endl;
   }
};

int main() {
   CircularLinkedList cll;

   // Menyisipkan beberapa elemen
   cll.insertMiddle(10);
   cll.insertMiddle(20);
   cll.insertMiddle(30);
   cll.insertMiddle(40);

   // Menampilkan elemen list
   cout << "Circular Linked List elements: ";
   cll.display();

   return 0;
}
```

SOURCE CODE CSLL INSERT BELAKANG

```
#include <iostream>

using namespace std;

// Struktur node untuk daftar terhubung
struct Node {
   int data;
   Node* next;

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

// Kelas untuk daftar terhubung melingkar
class CircularLinkedList {
private:
   Node* head;

public:
   // Konstruktor
   CircularLinkedList() : head(nullptr) {}

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

   // Fungsi untuk menyisipkan elemen di belakang (akhir)
   void insertEnd(int data) {
      Node* newNode = new Node(data);

      if (!head) {
         // Jika list kosong, node ini akan menunjuk ke dirinya sendiri
         head = newNode;
         newNode->next = head;
      } else {
         // Menyisipkan node baru di akhir
         Node* temp = head;
```

```
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
  }

  // Fungsi untuk menampilkan elemen list
  void display() const {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
  }
};

int main() {
  CircularLinkedList cll;

  // Menyisipkan beberapa elemen di belakang
  cll.insertEnd(10);
  cll.insertEnd(20);
  cll.insertEnd(30);
  cll.insertEnd(40);

  // Menampilkan elemen list
  cout << "Circular Linked List elements: ";
  cll.display();

  return 0;
}
```

SOURCE CODE CSLL HAPUS DEPAN

```
#include <iostream>

using namespace std;

// Struktur node untuk daftar terhubung
```

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

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

// Kelas untuk daftar terhubung melingkar
class CircularLinkedList {
private:
   Node* head;

public:
   // Konstruktor
   CircularLinkedList() : head(nullptr) {}

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

   // Fungsi untuk menyisipkan elemen di depan
   void insertFront(int data) {
      Node* newNode = new Node(data);

      if (!head) {
         // Jika list kosong, node ini akan menunjuk ke dirinya sendiri
         head = newNode;
         newNode->next = head;
      } else {
         // Menyisipkan node baru di depan dan mengubah pointer
         newNode->next = head;
         Node* temp = head;
         while (temp->next != head) {
            temp = temp->next;
         }
         temp->next = newNode;
         head = newNode;
      }
```

```
    }

    // Fungsi untuk menghapus elemen dari depan
    void deleteFront() {
        if (!head) {
            cout << "List is empty. Nothing to delete." << endl;
            return;
        }

        if (head->next == head) {
            // Jika hanya ada satu node di list
            delete head;
            head = nullptr;
        } else {
            Node* temp = head;
            // Temukan node terakhir
            while (temp->next != head) {
                temp = temp->next;
            }
            // Node terakhir menunjuk ke node berikutnya dari head
            Node* nodeToDelete = head;
            head = head->next;
            temp->next = head;
            delete nodeToDelete;
        }
    }

    // Fungsi untuk menampilkan elemen list
    void display() const {
        if (!head) {
            cout << "List is empty" << endl;
            return;
        }

        Node* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << endl;
    }
};

int main() {
    CircularLinkedList cll;

    // Menyisipkan beberapa elemen di depan
    cll.insertFront(10);
```

```
    cll.insertFront(20);
    cll.insertFront(30);

    // Menampilkan elemen list
    cout << "Circular Linked List elements before deletion: ";
    cll.display();

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

    // Menampilkan elemen list setelah penghapusan
    cout << "Circular Linked List elements after deletion: ";
    cll.display();

    return 0;
}
```

SOURCE CODE CSLL HAPUS TENGAH

```
#include <iostream>

// Struktur node untuk circular singly linked list
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk membuat node baru
Node* createNode(int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Fungsi untuk menambahkan node ke akhir daftar
void appendNode(Node*& head, int data) {
    Node* newNode = createNode(data);
    if (!head) {
        head = newNode;
        newNode->next = head; // Menghubungkan node dengan dirinya sendiri
    } else {
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
```

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

// Fungsi untuk menampilkan daftar
void printList(Node* head) {
    if (!head) return;
    Node* temp = head;
    do {
        std::cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    std::cout << std::endl;
}

// Fungsi untuk menghapus elemen tengah dari daftar
void deleteMiddle(Node*& head) {
    if (!head) return; // Daftar kosong
    if (head->next == head) { // Daftar hanya memiliki satu elemen
        delete head;
        head = nullptr;
        return;
    }

    Node* slow = head;
    Node* fast = head;
    Node* prev = nullptr;

    // Mencari elemen tengah
    while (fast->next != head && fast->next->next != head) {
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    }

    // Menghapus elemen tengah
    if (prev) {
        prev->next = slow->next;
    }

    if (slow == head) { // Jika elemen tengah adalah head
        head = slow->next;
    }

    delete slow;
}

// Fungsi utama
```

```
int main() {
    Node* head = nullptr;

    // Menambahkan beberapa elemen
    appendNode(head, 1);
    appendNode(head, 2);
    appendNode(head, 3);
    appendNode(head, 4);
    appendNode(head, 5);

    std::cout << "Daftar sebelum menghapus elemen tengah:" << std::endl;
    printList(head);

    deleteMiddle(head);

    std::cout << "Daftar setelah menghapus elemen tengah:" << std::endl;
    printList(head);

    return 0;
}
```

SOURCE CODE CSLL HAPUS BELAKANG

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

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

class CircularLinkedList {
private:
    Node* last;

public:
    CircularLinkedList() {
        last = nullptr;
    }

    // Fungsi untuk menambah node di akhir list
    void insertLast(int value) {
        Node* newNode = new Node();
        newNode->data = value;

        if (last == nullptr) {
            last = newNode;
```

```
            last->next = last;
        } else {
            newNode->next = last->next;
            last->next = newNode;
            last = newNode;
        }
    }

    // Fungsi untuk menghapus node terakhir (hapus belakang)
    void deleteLast() {
        if (last == nullptr) {
            cout << "List is empty, cannot delete." << endl;
            return;
        }

        if (last->next == last) {
            // Jika hanya ada satu node di list
            delete last;
            last = nullptr;
        } else {
            Node* temp = last->next;  // Pointer untuk node pertama
            while (temp->next != last) {
                temp = temp->next;
            }

            temp->next = last->next;  // Set next dari node kedua terakhir ke node pertama
            delete last;              // Hapus node terakhir
            last = temp;              // Ubah 'last' menjadi node kedua terakhir
        }
    }

    // Fungsi untuk menampilkan isi circular linked list
    void display() {
        if (last == nullptr) {
            cout << "List is empty." << endl;
            return;
        }

        Node* temp = last->next;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != last->next);
        cout << endl;
    }
};

int main() {
```

```
    CircularLinkedList list;

    list.insertLast(10);
    list.insertLast(20);
    list.insertLast(30);
    list.insertLast(40);

    cout << "List before deletion: ";
    list.display();

    list.deleteLast();

    cout << "List after deletion: ";
    list.display();

    return 0;
}
```