Day 3
Doubly Linked Lists

Intro

```
struct DLLNode {
     int data;
     struct DLLNode *next;
     struct DLLNode *prev;
}
```

Insertion at beginning

```
void insertAtBeginning(Node*& head, int value) {
    Node* newNode = new Node(value);
    if(!head) {
        head = newNode;
        return;
    }
    head->prev = newNode;
    newNode->next = head;
    head = newNode;
}
```

Insertion at end

```
void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if(!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while(temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
```

Insertion at a given position

```
void insertAtPosition(Node*& head, int position, int value) {
    Node* newNode = new Node(value);
    if(position == 1) {
        newNode->next = head;
        if(head) {
            head->prev = newNode;
        }
        head = newNode;
        return;
    }

    Node* temp = head;
    int count = 1;

    while(temp != nullptr && count < position - 1) {
        temp = temp->next;
        count++;
    }
    if(temp == nullptr) {
        cout << "Position out of bound\n";
        delete newNode; // to free the memory
        return;
    }
    newNode->next = temp->next;
    newNode->prev = temp;
    if(temp->next != nullptr) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
}
```

Deleting first node

```
void deleteFirstNode(Node*& head) {
    if(!head) {
        cout << "List is already empty.\n";
        return;
    }
    head = head->next;
    if(head) {
        head->prev = nullptr;
    }
    delete temp;
}
```

Deleting last node

```
void deleteLastNode(Node*& head) {
    if(!head) {
        cout << "List is already empty.\n";
        return;
    }
    if(head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }
    Node* last = head;
    while(last->next != nullptr) {
        last = last->next;
    }
    last->prev->next = nullptr;
    delete last;
}
```

Deleting Intermediate Node :

```
void deleteNodeAtPosition(Node*& head, int position) {
    if(!head) {
        cout << "List is empty.\n";
        return;
    }
    Node* current = head;
    int count = 1;
    while(current && count < position) {
        current = current->next;
        count++;
    }
    if(!current) {
        cout << "Position out of bounds/\n";
        return;
    }
    if(current == head) {
        head = head->next;
        if(head) {
            head->prev = nullptr;
        }
        delete current;
        return;
    }
    if(current->next) {
        current->next->prev = current->prev;
    }
    if(current->prev) {
        current->prev->next = current->next;
    }
    delete current;
}
```

Circular Linked Lists

Counting Nodes

```
int length(struct CLLNode *head) {
    struct CLLNode *current = head;
    int count = 0;
    if(head == NULL) {
        return 0;
    }
    do {
        current = current->next;
        count++;
    } while(current != head);
    return count;
}
```

Printing the contents of a CLL

```
void print(struct CLLNode *head) {
    struct CLLNode *current = head;
    if(head == NULL) {
        return;
    }
    do {
        cout << current->data;
        current = current->next;
    } while(current != head);
}
```

Inserting a node at the end

```
void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if(!head) {
        head = newNode;
        head->next = head;
        return;
    }
    Node* temp = head;
    while(temp->next != head) {
        temp = temp->next;
    }
    temp->next = newMode;
    newNode->next = head;
}
```

Inserting a node at the front
```
void insertAtFront(Node*& head, int value) {
    Node* newNode = new Node(value);
    if(!head) {
        head = newNode;
        head->next = head;
        return;
    }
    Node* temp = head;
    while(temp->next != head) {
        temp = temp->next;
    }
    newNode->next = head;
    temp->next = newMode;
    head = newNode;
}
```

Deleting the last node
```
void deleteLastNode(Node*& head) {
    if(!head) {
        cout << "List is empty.\n";
    }
    if(head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    Node* prev = nullptr;
    while(temp->next != head) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = head;
    delete temp;
}
```

Deleting the first node
```
void deleteFirstNode(Node*& head) {
    if(!head) {
        cout << "List is empty.\n";
    }
    if(head->next == head) {
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    Node* last = head;
    while(last->next != head) {
        last = last->next;
    }
    head = head->next;
    last->next = head;
    delete temp;
}
```