YEDİTEPE
UNIVERSITY

# Lab Exam 1
# Linked List

CSE 211 Data Structures

Batuhan Edgüer
bedguer@cse.yeditepe.edu.tr

November 11, 2025

# Contents

# 1 Introduction

This document contains 4 carefully designed coding questions for mastering linked list data structures in C++.

## 1.1 Prerequisites and Setup

Before starting this lab, you must download and extract the lab files. Follow these steps carefully:

1. **Download from YULearn:** Log in to YULearn and download the compressed lab archive (e.g., `linkedlist-lab.tar.gz`) to your Windows Downloads folder
2. **Open Ubuntu WSL Terminal:**
   - Click the Windows Start menu
   - Type "Ubuntu" and select the Ubuntu WSL application
   - Wait for the terminal to open
3. **Navigate to Downloads Folder:**
   ```
   cd /mnt/c/Users/user/Downloads/
   ```
4. **Extract the Lab Files:**
   ```
   make uncompress ARCHIVE=labM_N.tar.gz
   ```
   Replace `labM_N.tar.gz` with the actual filename you downloaded
5. **Navigate to Lab Directory:**
   ```
   cd labM_N/
   ```
   Replace `labM_N` with the actual filename you downloaded
6. **Verify Setup:** Check that the extraction was successful:
   ```
   ls -la
   ```
   You should see directories: `src/`, `include/`, and files: `Makefile`, `questions.pdf`

### 1.1.1 Building and Running the Lab

Once setup is complete, you can build and run the demonstration programs:

- **Build the program:**
  ```
  make
  ```
  This compiles all source files and creates the executable in `bin/program`
- **Run all question demonstrations:**

```
make run
```

This runs demonstrations for all four questions

- **Run a specific question:**

```
make run question=1    # Run Question 1 only
make run question=2    # Run Question 2 only
make run question=3    # Run Question 3 only
make run question=4    # Run Question 4 only
```

Use these commands to test individual questions as you implement them

- **Clean build files:**

```
make clean
```

Removes compiled object files and executables

**Tip:** Use `make run question=N` frequently while working on each question to verify your implementation matches the expected output.

### 1.1.2 If Make is Not Installed

If you don't have `make` installed on your system, you can compile and run the program directly using `g++`:

1. **Compile all source files and create the executable:**

```
g++ -Wall -Wextra -std=c++17 -g -Iinclude src/*.cpp -o program
```

2. **Run the program:**

```
./program           # Run all questions
./program 1         # Run question 1
./program 2         # Run question 2
./program 3         # Run question 3
./program 4         # Run question 4
./program all       # Run all questions
```

**Note:** You'll need to recompile manually after every code change using the first command.

## 1.2 Provided Resources

The `include/` directory provides:

- `LinkedList.h` - Complete template-based linked list implementation
- `LinkedListQuestions.h` - Function declarations for all 20 questions

- `Color.h` - ANSI color codes for terminal output visualization

The `src/` directory contains:

- `LinkedList.cpp` - Implementation of basic linked list operations
- `LinkedListQuestions.cpp` - Complete implementations of all 20 questions
- `main.cpp` - Comprehensive demonstrations of all questions with visual output

Study these implementations to understand proper linked list design, memory management, and algorithmic patterns. Each question builds upon fundamental concepts while introducing new challenges.

## 1.3 How to Use This Document

1. Read each question carefully and understand the requirements
2. Analyze the provided example to grasp expected behavior
3. Consider edge cases (empty list, single node, etc.)
4. Design your solution before coding
5. Implement and test thoroughly
6. Compare with the reference implementation in `LinkedListQuestions.cpp`

# 2 Questions

## 2.1 Question 1: Display list in reverse

**Description:**

Create a function that prints the linked list in reverse order without actually reversing the list structure. This encourages thinking about different approaches (recursion or stack-based).

**Function Signature:**

```
template <typename T>
void displayReverse(const LinkedList<T>& list);
```

**Example:**

```
LinkedList<int> list;
list.push_back(1);
list.push_back(2);
list.push_back(3);
list.push_back(4);

displayReverse(list);
// Output: 4 <- 3 <- 2 <- 1
```

**Requirements:**

- Do not modify the list structure
- Print elements in reverse order
- Use either recursive or stack-based approach
- Handle empty lists appropriately

**Learning Objectives:**

- Recursive thinking for linked lists
- Alternative: Stack data structure usage
- Understanding call stack behavior
- Read-only operations

## 2.2 Question 2: Swap Adjacent Nodes Pairwise

**Description:**

Write a function that swaps every two adjacent nodes in the list. For example, {1, 2, 3, 4} becomes {2, 1, 4, 3}. Students must think through pointer rearrangement carefully.

**Function Signature:**

```
template <typename T>
void swapAdjacent(LinkedList<T>& list);
```

**Example:**

```
LinkedList<int> list;
list.push_back(1);
list.push_back(2);
list.push_back(3);
list.push_back(4);
list.push_back(5);

swapAdjacent(list);
// List is now: [2 -> 1 -> 4 -> 3 -> 5]
```

**Requirements:**

- Swap nodes by changing pointers, not data
- Handle odd-length lists (last node stays in place)
- Update head pointer if first pair swapped
- Maintain proper linking between pairs
- Process pairs sequentially

**Learning Objectives:**

- Complex pointer manipulation
- Multi-pointer tracking (prev, current, next)
- Pair-wise processing patterns
- Head pointer updates

## 2.3 Question 3: Rotate List by K Positions

**Description:**

Implement a function that rotates a linked list to the right by k positions. For example, rotating {1, 2, 3, 4, 5} by 2 positions gives {4, 5, 1, 2, 3}.

**Function Signature:**

```
template <typename T>
void rotateRight(LinkedList<T>& list, size_t k);
```

**Example:**

```
LinkedList<int> list;
list.push_back(1);
list.push_back(2);
list.push_back(3);
list.push_back(4);
list.push_back(5);

rotateRight(list, 2);
// List is now: [4 -> 5 -> 1 -> 2 -> 3]
```

**Requirements:**

- Handle k greater than list size (use modulo)
- Handle k = 0 (no rotation needed)
- Find new tail and new head positions
- Connect old tail to old head
- Update head pointer
- Do not allocate new nodes

**Learning Objectives:**

- Circular list concept
- Modular arithmetic for rotation
- Pointer reconnection patterns
- Finding rotation point efficiently

## 2.4 Question 4: Merge Two Lists Alternately

**Description:**

Given two linked lists, create a function that merges them by alternating nodes from each list. For example, {1, 2, 3} and {4, 5, 6} becomes {1, 4, 2, 5, 3, 6}.

**Function Signature:**

```
template <typename T>
LinkedList<T> mergeAlternately(const LinkedList<T>& list1,
                               const LinkedList<T>& list2);
```

**Example:**

```
LinkedList<int> list1;
list1.push_back(1);
list1.push_back(3);
list1.push_back(5);

LinkedList<int> list2;
list2.push_back(2);
list2.push_back(4);
list2.push_back(6);

LinkedList<int> merged = mergeAlternately(list1, list2);
// merged is: [1 -> 2 -> 3 -> 4 -> 5 -> 6]
```

**Requirements:**

- Alternate elements from both lists
- Handle lists of different lengths
- If one list is longer, append remaining elements
- Create new list (do not modify originals)
- Use two pointers to traverse both lists

**Learning Objectives:**

- Multi-list traversal
- Alternating merge pattern
- Handling unequal-length lists
- Building new list from multiple sources

**Good luck!**