King Abdulaziz University Faculty of Engineering

# Lab (4)
# The C Programming Environment in Linux
# Gcc, Make, Makefiles and Gdb

**Operating Systems– EE463**

Lecturer(s): Dr. Abdulghani M. Al-Qasimi
& Eng. Turki Abdul Hafeez
3rd Semester Spring 2023
Section: C3 (Thursday 13 to 14:40)

| # | Name | ID |
|---|---|---|
| 1 | Moad Abdulaali Alshikh | 1937343 |

**Exercises:**

**1- Memory leak**

The memory leak in the program occurs because not all Node instances are deleted when they are removed from the LinkedList. To find the memory leak, let's first examine the LinkedList::remove function in main.c

```
bool LinkedList::remove (int item_to_remove) {
    Node *marker = head_;
    Node *temp = 0; // temp points to one behind as we iterate

    // Iterate through the list
    while (marker != 0) {
        if (marker->value() == item_to_remove) {
            // Found the value in the list; let's remove it
            if (temp == 0) {
                head_ = marker->next();
            } else {
                temp->set_next(marker->next());
            }
            return true;
        }

        // Move on to the next item in the list
        temp = marker;
        marker = marker->next();
    }

    // Value was not found in the list
    return false;
}
```

When a node is removed from the list, its memory is not deallocated. We can fix this by deleting the node after updating the pointers.

```cpp
bool LinkedList::remove (int item_to_remove) {
    Node *marker = head_;
    Node *temp = 0; // temp points to one behind as we iterate

    // Iterate through the list
    while (marker != 0) {
        if (marker->value() == item_to_remove) {
            // Found the value in the list; let's remove it
            if (temp == 0) {
                head_ = marker->next();
            } else {
                temp->set_next(marker->next());
            }
            delete marker; // Free the memory of the removed node
            return true;
        }

        // Move on to the next item in the list
        temp = marker;
        marker = marker->next();
    }

    // Value was not found in the list
    return false;
}
```

## 2- Bug

The bug occurs when trying to remove an element from the middle of the list. Let's modify the driver code to insert 1, 2, 3, and 4, and then try to remove 2.

```cpp
int main() {
    LinkedList *list = new LinkedList;
    list->insert(1);
    list->insert(2);
    list->insert(3);
    list->insert(4);
    list->print();
    list->remove(2);
    list->print();
    delete list;
    return 0;
}
```

The problem lies in the LinkedList::remove function. When removing a node from the middle of the list, the temp pointer should point to the previous node, but it currently points to the node being removed. We can fix this by updating the temp pointer after removing the node.

```
bool LinkedList::remove (int item_to_remove) {
    Node *marker = head_;
    Node *temp = 0; // temp points to one behind as we iterate

    // Iterate through the list
    while (marker != 0) {
        if (marker->value() == item_to_remove) {
            // Found the value in the list; let's remove it
            if (temp == 0) {
                head_ = marker->next();
            } else {
                temp->set_next(marker->next());
            }
            delete marker; // Free the memory of the removed node
            return true;
        }

        // Move on to the next item in the list
        temp = marker;
        marker = marker->next();
    }

    // Value was not found in the list
    return false;
}
```

Now, the driver code should run correctly, and the **output** will show the list with the element 2 removed:

```
4
3
2
1
4
3
1
```