Mark Shinozaki
HW03 – Observations Report
Professor: Ananth Jillepalli
Date: 10/18/23

- Includes:
    - Screenshots providing you did preform the tutorial tasks i.e. activities, challenges, and answer all questions
    - A filled out OHD_Checklist, found in the 2nd link above.
    - Report any bugs, typos, broken links etc.
    - A brief discussion on the skills you've learned from the tutorial (7 lines max)

Code File.PDF

Linked_list.h

```
#if !defined(LINKED_LIST_H)
#define LINKED_LIST_H
#include <stdlib.h> // For definition of NULL
Extern short g_node_counter;
Template <class ITEM_TYPE, int MAX_NODES = 100000>
Class LinkedListNode
{
public:
        LinkedLinkNode()
        {
                If (g_node_counter < MAX_NODES) {
                        m_next = NULL:
                        ++g_node_counter;
                }
        }
        ~LinkedListNode()
        {
                delete m_next;
                m_next = NULL.
                --g_node_counter;
        }
        Void insert (LinkedListNode & * pNode)
        {
                m_next = pNode;
                pNode = m_next;
        }
        Void append (LinkedListNode & * pNode)
        {
                M_next = pNode->m_next;
                pNode->m_next = this;
        }
        Void remove (LinkedListNode & * pNode)
        {
                pNode = pNode->m_next;
        }
        Unsigned int get_count (void)
        {
```

```
        Return g_node_counter;
}
ITEM_TYPE & data(void) { return m_data; }
ITEM_TYPE m_data;
LinkedListNode * m_next;
}
#endif // LINKED_LIST_H
////////////////////////////////////////////////////////
```

Linked_list.cpp

```
#include “linked_list.h”
Short g_node_counter:
////////////////////////////////////////////////////////
```

main.cpp

```
#include <iostream>
#include <map>
#include <linked_list.h>
#define MAX_NODES 75000
Typedef LinkedListNode<int, MAX_NODES> int_list_t;
Typedef LinkedListNode<double, MAX_NODES> double_list_t;
////////////////////////////////////////////////////////
Class UberNode : public int_list
{
        UberNode () {   }
        ~UberNode () {}
        Void add_value (std::string key) { return m_map[key] ; }
Private:
        std::map<std::string, int> m_map:
};
////////////////////////////////////////////////////////
Int main(int, char **)
{
        UberNode *pRoot = new UberNode;
        UberNode *pLast = pRoot;
        Int count = pRoot->get_count();
        While (count < MAX_NODES)
        {
                UberNode * pNew = new UberNode();
                pNew->add_value (“COUNT”, count):
                pDoubleLast->append(pNew);
                pDoubleLast = pNew;
        }
        delete pRoot;
        delete pDoubleRoot;
        return 0;
}
```

Mark Shinozaki
HW03 – Observations Report
Professor: Ananth Jillepalli
Date: 10/18/23

Notes:

1. Since linkedListNode does not have a virtual destructor, when an instance of Uber-Node is destroyed via a pointer to it's parent, UberNode's destructor will not be called. That means the destructors of UberNode's members (e.g., m_map) will not be called so they will not have a chance to free any memory they allocate.
2. Static Analysis is a subset of Static Testing. Static Testing involves using mostly mechanical methods to analyze software without executing it. Static Testing encompasses both static Analysis and various forms of code review.
3. Trying to write an algorithm to prove that non-trivial program is correct is reducible to the "halting problem" which has been proven (by alan turing) to be unsolvable (caveat, the proof was actually for a Turing Machine, which has infinite resources, not for a resource contrained physical computer)

Questions:

1. In the Code File of the challenge, there was a potential for a major memory leak in the "UberNode". What was it?
    - **Potential Memory Leak in "UberNode":**
        - The potential memory leak in the "UberNode" class is related to the destructor. "UberNode" contains a map member variable, "m_map", which is a dynamic data structure that allocates memory for its entries. However, the destructor for "UberNode" doesn't clean up or release the memory allocated for "m_map." When an instance of "UberNode" is destroyed, the memory allocated for the map entries is not released, which can lead to a memory leak. To avoid this issue, the destructor of "UberNode" should explicitly release the memory allocated for "m_map."

2. What is the difference between Static Analysis and Static Testing?
    - Static Analysis: Static Analysis is a process of examining the source code or program without executing it. It uses automated tools and techniques to analyze the codes structure, syntax, and semantics. The primary goal of static analysis is to identify potential issues and defects in the code, such as coding standards violations, potential memory leaks, and data flow anomalies. Static analysis is preformed without running the program.
    - Static Testing: Static testing is a broader concept that encompasses both static analysis and various forms of code review. It involves examining the code, design, and documentation to identify issues and defects.
    - **Difference:**
        - A few different qualities make the main differences and those include, Nature, Human Involvement, Defect Discovery, Automation, scope, integration, complexity. In practice, organizations often use a

combination of both static analysis and static testing to achieve a comprehensive approach to software quality assurance. Each approach complements the other and helps ensure that software is both defect-free and meets quality standards.

3. How do we know we cant write an algorithm to prove a non-trivial program is correct?
    - The question mentions or alludes to this concept of the halting problem. The halting problem from Alan Turing states that no algorithm that can determine, for all possible programs and inputs, whether a given program will halt (terminate) or run forever. This undecidability result has profound implications for the program and its correctness verification. If we could write an algorithm to prove the correctness of all non-trivial programs, it would solve the halting problem but this cannot happen since the problem is unsolvable.

A brief discussion on the skills you've learned from the tutorial:

- Memory Leaks
- Static Testing VS Static Analysis
- Non-trivial programs
- Halting problem
- Techincal Review and Code Inspection
- Using Code review Checklists

    - Memory leaks occur when a program fails to release allocated memory, potentially causing performance issues. Static testing and static analysis are software quality assurance techniques-static testing involves human-driven inspections, while static analysis relies on automated tools to detect code issues without execution. Non-trivial programs are complex, they prove correct due to their intricate behaviors. The halting problem highlights the impossibility of creating an algorithm that determines program termination for all inputs. Technical reviews and code inspections are peer-based assessments that are meant to identify defects, improve quality, and ensure compliance with coding standards.