Swinburne University of Technology

School of Science, Computing and Engineering Technologies

FINAL EXAM COVER SHEET

Subject Code: COS30008

Subject Title: Data Structures & Patterns

Due date: Nov 29, 2023

Lecturer: Dr. James Jackson

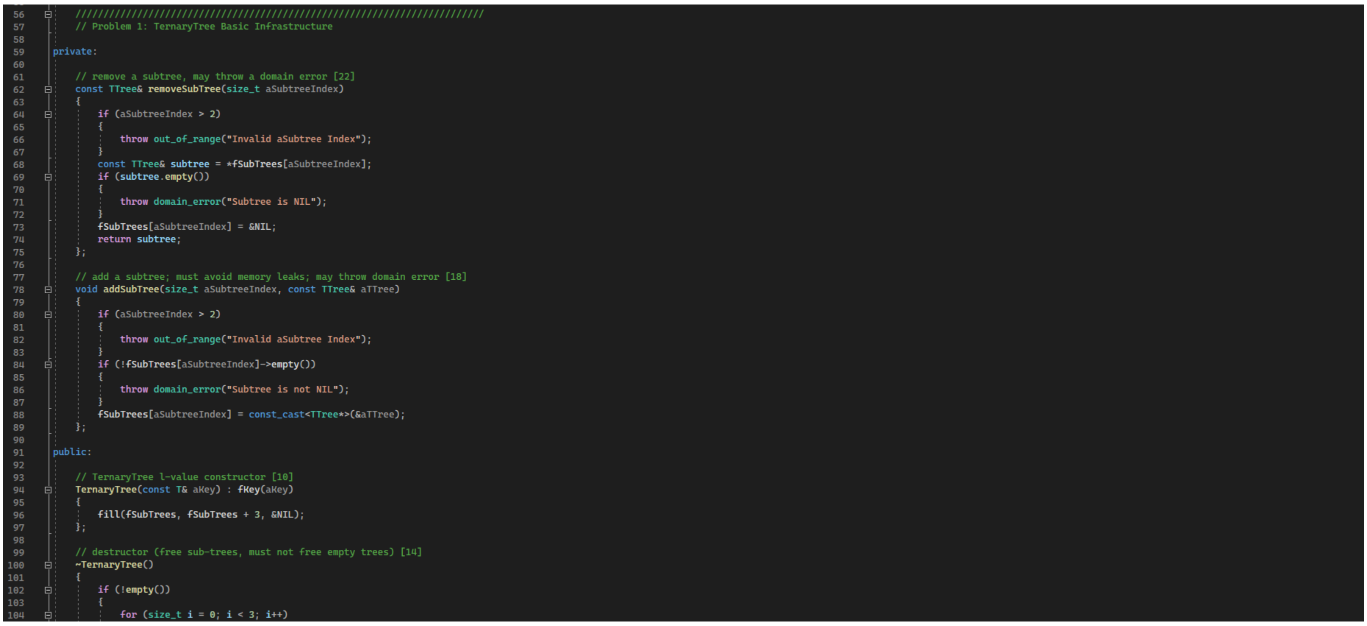
Your name:Tran Hoang Hai Anh Your student id:_ 104177513

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30

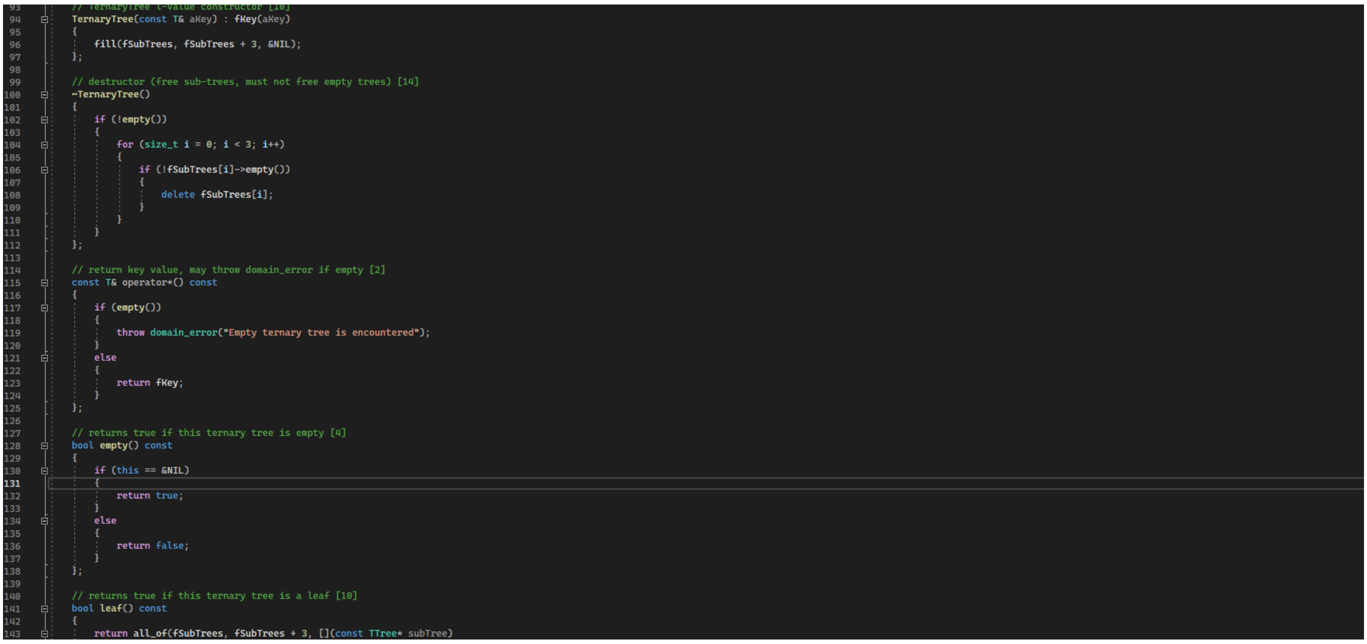
Marker's comments:

Problem	Marks	Time Estimate in minutes	Obtained
1	132	30	
2	56	10	
3	60	15	
4	10+88=98	45	
5	50	20	
Total	396	120	

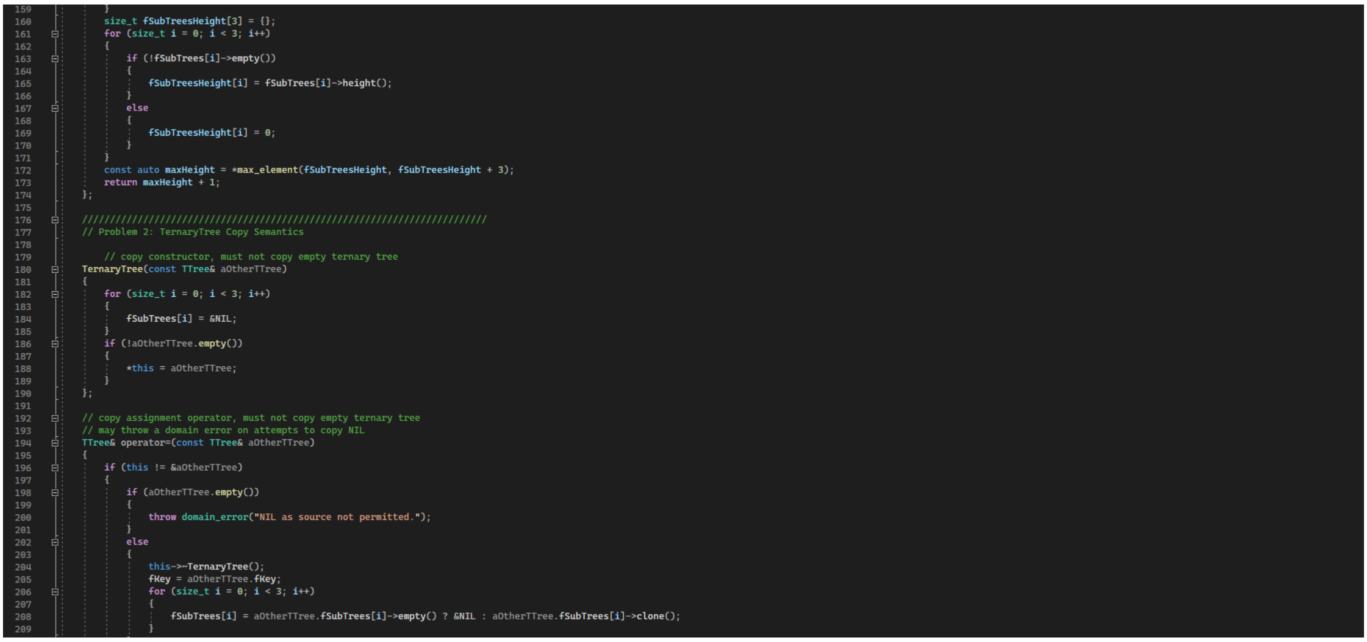
This test requires approx. 2 hours and accounts for 50% of your overall mark.



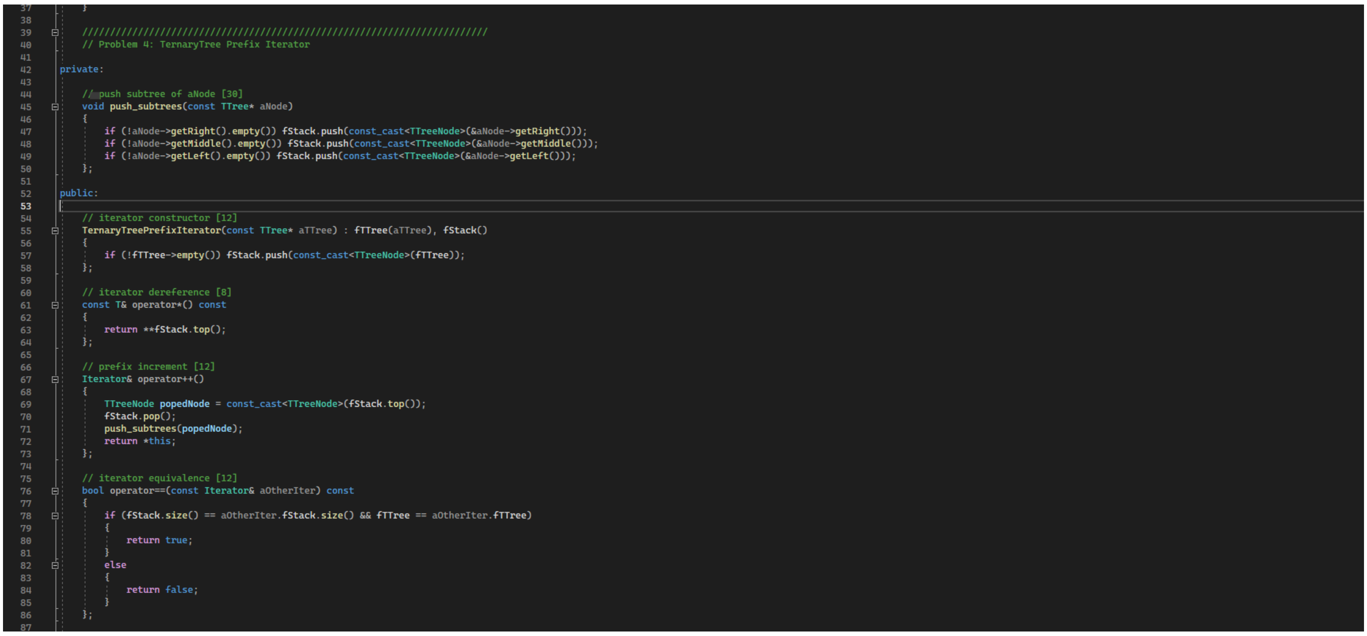
```cpp
    ///////////////////////////////////////////////////////////////////////
    // Problem 1: TernaryTree Basic Infrastructure

private:

    // remove a subtree, may throw a domain error [22]
    const TTree& removeSubTree(size_t aSubtreeIndex)
    {
        if (aSubtreeIndex > 2)
        {
            throw out_of_range("Invalid aSubtree Index");
        }
        const TTree& subtree = *fSubTrees[aSubtreeIndex];
        if (subtree.empty())
        {
            throw domain_error("Subtree is NIL");
        }
        fSubTrees[aSubtreeIndex] = &NIL;
        return subtree;
    };

    // add a subtree; must avoid memory leaks; may throw domain error [18]
    void addSubTree(size_t aSubtreeIndex, const TTree& aTTree)
    {
        if (aSubtreeIndex > 2)
        {
            throw out_of_range("Invalid aSubtree Index");
        }
        if (!fSubTrees[aSubtreeIndex]->empty())
        {
            throw domain_error("Subtree is not NIL");
        }
        fSubTrees[aSubtreeIndex] = const_cast<TTree*>(&aTTree);
    };

public:

    // TernaryTree l-value constructor [10]
    TernaryTree(const T& aKey) : fKey(aKey)
    {
        fill(fSubTrees, fSubTrees + 3, &NIL);
    };

    // destructor (free sub-trees, must not free empty trees) [14]
    ~TernaryTree()
    {
        if (!empty())
        {
            for (size_t i = 0; i < 3; i++)
```



```cpp
    // TernaryTree l-value constructor [10]
    TernaryTree(const T& aKey) : fKey(aKey)
    {
        fill(fSubTrees, fSubTrees + 3, &NIL);
    };

    // destructor (free sub-trees, must not free empty trees) [14]
    ~TernaryTree()
    {
        if (!empty())
        {
            for (size_t i = 0; i < 3; i++)
            {
                if (!fSubTrees[i]->empty())
                {
                    delete fSubTrees[i];
                }
            }
        }
    };

    // return key value, may throw domain_error if empty [2]
    const T& operator*() const
    {
        if (empty())
        {
            throw domain_error("Empty ternary tree is encountered");
        }
        else
        {
            return fKey;
        }
    };

    // returns true if this ternary tree is empty [4]
    bool empty() const
    {
        if (this == &NIL)
        {
            return true;
        }
        else
        {
            return false;
        }
    };

    // returns true if this ternary tree is a leaf [10]
    bool leaf() const
    {
        return all_of(fSubTrees, fSubTrees + 3, [](const TTree* subTree)
```



```
        }
        size_t fSubTreesHeight[3] = {};
        for (size_t i = 0; i < 3; i++)
        {
            if (!fSubTrees[i]->empty())
            {
                fSubTreesHeight[i] = fSubTrees[i]->height();
            }
            else
            {
                fSubTreesHeight[i] = 0;
            }
        }
        const auto maxHeight = *max_element(fSubTreesHeight, fSubTreesHeight + 3);
        return maxHeight + 1;
    };

    ////////////////////////////////////////////////////////////////////////////
    // Problem 2: TernaryTree Copy Semantics

        // copy constructor, must not copy empty ternary tree
    TernaryTree(const TTree& aOtherTTree)
    {
        for (size_t i = 0; i < 3; i++)
        {
            fSubTrees[i] = &NIL;
        }
        if (!aOtherTTree.empty())
        {
            *this = aOtherTTree;
        }
    };

    // copy assignment operator, must not copy empty ternary tree
    // may throw a domain error on attempts to copy NIL
    TTree& operator=(const TTree& aOtherTTree)
    {
        if (this != &aOtherTTree)
        {
            if (aOtherTTree.empty())
            {
                throw domain_error("NIL as source not permitted.");
            }
            else
            {
                this->~TernaryTree();
                fKey = aOtherTTree.fKey;
                for (size_t i = 0; i < 3; i++)
                {
                    fSubTrees[i] = aOtherTTree.fSubTrees[i]->empty() ? &NIL : aOtherTTree.fSubTrees[i]->clone();
                }
```



```
    }

    ////////////////////////////////////////////////////////////////////////////
    // Problem 4: TernaryTree Prefix Iterator

private:

    // push subtree of aNode [30]
    void push_subtrees(const TTree* aNode)
    {
        if (!aNode->getRight().empty()) fStack.push(const_cast<TTreeNode>(&aNode->getRight()));
        if (!aNode->getMiddle().empty()) fStack.push(const_cast<TTreeNode>(&aNode->getMiddle()));
        if (!aNode->getLeft().empty()) fStack.push(const_cast<TTreeNode>(&aNode->getLeft()));
    };

public:

    // iterator constructor [12]
    TernaryTreePrefixIterator(const TTree* aTTree) : fTTree(aTTree), fStack()
    {
        if (!fTTree->empty()) fStack.push(const_cast<TTreeNode>(fTTree));
    };

    // iterator dereference [8]
    const T& operator*() const
    {
        return **fStack.top();
    };

    // prefix increment [12]
    Iterator& operator++()
    {
        TTreeNode popedNode = const_cast<TTreeNode>(fStack.top());
        fStack.pop();
        push_subtrees(popedNode);
        return *this;
    };

    // iterator equivalence [12]
    bool operator==(const Iterator& aOtherIter) const
    {
        if (fStack.size() == aOtherIter.fStack.size() && fTTree == aOtherIter.fTTree)
        {
            return true;
        }
        else
        {
            return false;
        }
    };

```