Swinburne University of Technology

Faculty of Science, Engineering and Technology

FINAL EXAM COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures & Patterns
Due date: June 3, 2021, 13:00
Lecturer: Dr. Markus Lumpe

Your name: Nguyen Duc Manh **Your student id:** 103434699

Check Tutorial	Wed 08:30	Wed 10:30	Wed 16:30	Thurs 08:30	Thurs 10:30	Thurs 14:30	Thurs 16:30	Fri 08:30	Fri 10:30	Fri 14:30
			X							

Marker's comments:

Problem	Marks	Time Estimate in minutes	Obtained
1	50	20	
2	54	15	
3	42	10	
4	60	15	
5	8+128=136	60	
Total	342	120	

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

Problem 1 (50 marks)

Answer the following questions in one or two sentences:

a. How can we construct a tree where all subtrees have the same degree? (4 marks)

1a)

We need a special notation, called empty tree, to realize trees in which all nodes have the same degree

b. What are reference data members and how do we initialize them? (2 marks)

1b)

Reference data members store references to data outside an object. These references are established upon object creation via a member initializer.

c. What is the difference between l-value and r-value references? (6 marks)

1c)

L-value referers to a thing that occur on the left side of an assignment named objects that have a defined storage location. L-value can only be bound to l-values/

R-value referes to thing that occur on the right side of an assignment literals and temporaries that do not have a defined storage location. R-value references can only be bind to r-value

d. What is an object adapter? (6 marks)

1d)

When clients call operations on an Adapter instance of objects. In turn, the adapter calls Adapter operations that carry out the request.

e. What is a key concept of an abstract data types? (4 marks)

1e)

The technique used to define new data types independently of their actual representations is called data abstraction.
A client (program) can use values of an abstract data type by means of the interface without knowing their actual representations.

f. How do we define mutual dependent classes in C++? (4 marks)

1f)

We can use foward declaration on at least 1 class.

g. What must a value-based data type define in C++? (2 marks)

1g)

They must have a copy constructor to create a copy of another variable
A value based data type has an assignment operator that assigns itself a copy of other variables.
These kind of data type mostly located on the stack.
If there are changes to one variable then it will not affect other variable values.

h. What is the difference between copy constructor and assignment operator and how do we guarantee safe operation? (8 marks)

1h)

A copy constructor is used to initialize a previously uninitialized object from some other object's data

An assignment operator is used to replace the data of a previously initialized object with some other object's data

i. What is the best-case, average-case, and worse-case for a lookup in a binary tree? (6 marks)

1i)

Best case: O(1)
Average case: O(logn)
Worst case: O(n)

j. You are given a set of n-1 numbers out of n numbers. How do we find the missing number n_k, $1 \leq k \leq n$, in linear time? (8 marks)

1j)

First i will create an array of n-1 numbers, and get the size of that array.
A linear function will be created which have 2 parameters which is the array and the size of the array.
I will calculated the total sum by ((n+1)*(n+2))/2) of the array and using a for loop to decrement the total by each array element.
At the end i will find the missing number.
i provide the coding answer below due to the length of the text.

QUESTION 1-J

```cpp
#include <iostream>
using namespace std;

int getMissingNumber(int arr[], int n) {
    //get total if the missing number is presented.
    int arrtotal = (n + 1) * (n + 2) / 2;
    for (int i = 0; i < n; i++) {
        // decerement til the end of the array
        arrtotal -= arr[i];
    }
    return arrtotal;
}

int main()
{
    int arr[] = { 1, 2, 3, 5, 6, 7 };
    int arrsize = sizeof(arr) / sizeof(arr[0]);
    int missednum = getMissingNumber(arr, arrsize);
    cout << "given array :";
    for (int i = 0; i < arrsize; i++) {
        cout << arr[i] << ",";
    }
    cout << endl;
    cout << "the missing number is: " << missednum << endl;
}
```

```cpp
// COS30008, Final Exam, 2021

#pragma once

#include <stdexcept>

template<typename T>
class TTreePostfixIterator;

template<typename T>
class TTree
{
private:

    T fKey;
    TTree<T>* fLeft;
    TTree<T>* fMiddle;
    TTree<T>* fRight;

    TTree() : fKey(T())     // use default constructor to initialize fKey
    {
        fLeft = &NIL;       // loop-back: The sub-trees of a TTree object with
        fMiddle = &NIL;     //            no children point to NIL.
        fRight = &NIL;
    }

    void addSubTree(TTree<T>** aBranch, const TTree<T>& aTTree)
    {
        if (!(*aBranch)->empty())
        {
            delete* aBranch;
        }

        *aBranch = const_cast<TTree<T>*>(&aTTree);
    }

public:

    using Iterator = TTreePostfixIterator<T>;

    static TTree<T> NIL;    // sentinel

    // getters for subtrees
    const TTree<T>& getLeft() const { return *fLeft; }
```

```
    const TTree<T>& getMiddle() const { return *fMiddle; }
    const TTree<T>& getRight() const { return *fRight; }

    // add a subtree
    void addLeft(const TTree<T>& aTTree) { addSubTree(&fLeft, aTTree); }
    void addMiddle(const TTree<T>& aTTree) { addSubTree(&fMiddle, aTTree); }
    void addRight(const TTree<T>& aTTree) { addSubTree(&fRight, aTTree); }

    // remove a subtree, may through a domain error
    const TTree<T>& removeLeft() { return removeSubTree(&fLeft); }
    const TTree<T>& removeMiddle() { return removeSubTree(&fMiddle); }
    const TTree<T>& removeRight() { return removeSubTree(&fRight); }

    // Problem 1: TTree Basic Infrastructure

private:

    // remove a subtree, may through a domain error
    const TTree<T>& removeSubTree(TTree<T>** aBranch) {
        if (!(*aBranch)->empty())
        {
            *aBranch = &NIL;
        }
        return *this;
    }

public:

    // TTree l-value constructor
    TTree(const T& aKey) :
        fKey(aKey)
    {
        fLeft = &NIL;       // loop-back: The sub-trees of a TTree object with
        fMiddle = &NIL;     //            no children point to NIL.
        fRight = &NIL;
    }

    // destructor (free sub-trees, must not free empty trees)
    ~TTree() {
        if (!empty()) {
            removeLeft();
            removeMiddle();
            removeRight();
        }
    }
```

```
// return key value, may throw domain_error if empty
const T& operator*() const {
    if (empty())
        throw std::domain_error("Empty TTree encountered.");
    return fKey;
}

// returns true if this TTree is empty
bool empty() const {
    return this == &NIL;
}

// returns true if this TTree is a leaf
bool leaf() const {
    return fLeft->empty() && fMiddle->empty() && fRight->empty();
}

// Problem 2: TTree Copy Semantics

    // copy constructor, must not copy empty TTree
TTree(const TTree<T>& aOtherTTree)
{
    if (aOtherTTree.empty()) {
        throw std::domain_error("Copying NIL");
    }
    fLeft = new TTree<T>();
    fRight = new TTree<T>();
    fMiddle = new TTree<T>();
    if (!aOtherTTree.empty()) {
        fKey = aOtherTTree.fKey;
        fLeft = aOtherTTree.fLeft->clone();
        fRight = aOtherTTree.fRight->clone();
        fMiddle = aOtherTTree.fMiddle->clone();
    }
}

// assignment operator, must not copy empty TTree
TTree<T>& operator=(const TTree<T>& aOtherTTree) {
    if (aOtherTTree.empty()) {
        throw std::domain_error("Copying NIL");
    }
    if (**this != *aOtherTTree) {
        //delete
```

```
            delete this;
            //copy
            fKey = std::move(aOtherTTree.fKey);
            fLeft = std::move(aOtherTTree.fLeft);
            fMiddle = std::move(aOtherTTree.fMiddle);
            fRight = std::move(aOtherTTree.fRight);
        }
        return *this;
    }

    // clone TTree, must not copy empty trees
    TTree<T>* clone() const {
        if (!empty()) {
            return new TTree(*this);
        }
    }

    // Problem 3: TTree Move Semantics

        // TTree r-value constructor
    TTree(T&& aKey) : fKey(std::move(aKey)), fLeft(&NIL), fMiddle(&NIL),
fRight(&NIL) {}

    // move constructor, must not copy empty TTree
    TTree(TTree<T>&& aOtherTTree) : fKey(aOtherTTree.fKey),
fLeft(aOtherTTree.fLeft), fRight(aOtherTTree.fRight),
fMiddle(aOtherTTree.fMiddle) {
        if (aOtherTTree.empty()) {
            throw std::domain_error("Moving NIL");
        }
        aOtherTTree.removeLeft();
        aOtherTTree.removeRight();
        aOtherTTree.removeMiddle();
    }

    //move assignment operator, must not copy empty TTree
    TTree<T>& operator=(TTree<T>&& aOtherTTree) {
        if (!aOtherTTree.empty()) {
            if (this != &aOtherTTree) {
                //delete
                if (fLeft != &NIL) {
                    delete fLeft;
                }
                if (fMiddle != &NIL) {
                    delete fMiddle;
```

```
                }
                if (fRight != &NIL) {
                    delete fRight;
                }

                fLeft = std::move(aOtherTTree.fLeft);
                fMiddle = std::move(aOtherTTree.fMiddle);
                fRight = std::move(aOtherTTree.fRight);
                //copy
                &aOtherTTree.removeLeft();
                &aOtherTTree.removeMiddle();
                &aOtherTTree.removeRight();
            }
            return *this;
        }
        else {
            throw std::domain_error("Moving NIL");
        }
    }

    // Problem 4: TTree Postfix Iterator

        // return TTree iterator positioned at start
    Iterator begin() const;

    // return TTree iterator positioned at end
    Iterator end() const;
};

template<typename T>
TTree<T> TTree<T>::NIL;
```