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: June 7, 2022, 18:00
Lecturer: Dr. Markus Lumpe

Your name: Trac Duc Anh Luong **Your student id:** 103488117

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
				X							

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.

```
 1
 2  // COS30008, Final Exam, 2022
 3
 4  #pragma once
 5
 6  #include <stdexcept>
 7  #include <algorithm>
 8
 9  using namespace std;
10
11  template<typename T>
12  class TernaryTreePrefixIterator;
13
14  template<typename T>
15  class TernaryTree
16  {
17  public:
18
19      using TTree = TernaryTree<T>;
20      using TSubTree = TTree*;
21
22  private:
23
24      T fKey;
25      TSubTree fSubTrees[3];
26
27      // private default constructor used for declaration of NIL
28      TernaryTree() :
29          fKey(T())
30      {
31          for ( size_t i = 0; i < 3; i++ )
32          {
33              fSubTrees[i] = &NIL;
34          }
35      }
36
37  public:
38
39      using Iterator = TernaryTreePrefixIterator<T>;
40
41      static TTree NIL;           // sentinel
42
43      // getters for subtrees
44      const TTree& getLeft() const { return *fSubTrees[0]; }
45      const TTree& getMiddle() const { return *fSubTrees[1]; }
46      const TTree& getRight() const { return *fSubTrees[2]; }
47
48      // add a subtree
49      void addLeft( const TTree& aTTree ) { addSubTree( 0, aTTree ); }
```

```
50      void addMiddle( const TTree& aTTree ) { addSubTree( 1, aTTree ); }
51      void addRight( const TTree& aTTree ) { addSubTree( 2, aTTree ); }
52
53      // remove a subtree, may through a domain error
54      const TTree& removeLeft() { return removeSubTree( 0 ); }
55      const TTree& removeMiddle() { return removeSubTree( 1 ); }
56      const TTree& removeRight() { return removeSubTree( 2 ); }
57
58  ////////////////////////////////////////////////////////////////////////
59  // Problem 1: TernaryTree Basic Infrastructure
60
61  private:
62
63      // remove a subtree, may throw a domain error [22]
64      const TTree& removeSubTree( size_t aSubtreeIndex )
65      {
66          if (fSubTrees[aSubtreeIndex]->empty())
67          {
68              throw domain_error("Subtree is NIL");
69          }
70          if (aSubtreeIndex > 2)
71          {
72              throw out_of_range("Illegal subtree index");
73          }
74          const TTree& index = const_cast<TTree&>(*fSubTrees
              [aSubtreeIndex]);
75          fSubTrees[aSubtreeIndex] = &NIL;
76          return index;
77      }
78
79      // add a subtree; must avoid memory leaks; may throw domain error [18]
80      void addSubTree( size_t aSubtreeIndex, const TTree& aTTree )
81      {
82          if (empty())
83          {
84              throw domain_error("Operation not supported");
85          }
86          if (aSubtreeIndex > 2)
87          {
88              throw out_of_range("Illegal subtree index");
89          }
90          if (!fSubTrees[aSubtreeIndex]->empty())
91          {
92              throw domain_error("Subtree is not NIL");
93          }
94          fSubTrees[aSubtreeIndex] = const_cast<TTree*>(& aTTree);
95      }
96
97  public:
```

```
 98
 99      TernaryTree(const T& aKey) :fKey(aKey)
100      {
101          for (int i = 0; i < 3; i++)
102          {
103              fSubTrees[i] = &NIL;
104          }
105      }
106
107      // destructor (free sub-trees, must not free empty trees) [14]
108      ~TernaryTree()
109      {
110          if (!empty())
111          {
112              for (int i = 0; i < 3; i++)
113              {
114                  if (!fSubTrees[i]->empty())
115                  {
116                      delete fSubTrees[i];
117                  }
118              }
119          }
120      }
121
122      // return key value, may throw domain_error if empty [2]
123      const T& operator*() const
124      {
125          if (empty())
126          {
127              throw domain_error("Tree is empty");
128          }
129          return fKey;
130      }
131
132      // returns true if this ternary tree is empty [4]
133      bool empty() const { return this == &NIL; }
134
135      // returns true if this ternary tree is a leaf [10]
136      bool leaf() const
137      {
138          for (int i = 0; i < 3; i++)
139          {
140              if (!fSubTrees[i]->empty()) return false;
141          }
142          return true;
143      }
144
145      // return height of ternary tree, may throw domain_error if empty [48]
146      size_t height() const
```

```
147     {
148         if (empty())
149         {
150             throw domain_error("Operation not supported");
151         }
152         if (leaf()) return 0;
153         size_t height[3] = {};
154 
155         for (int i = 0; i < 3; i++)
156         {
157             height[i] = fSubTrees[i]->empty() ? 0 : fSubTrees[i]->height
                ();
158         }
159         return *max_element(height, height + 3) + 1;
160     }
161 
162 ////////////////////////////////////////////////////////////////////////
163 // Problem 2: TernaryTree Copy Semantics
164 
165     // copy constructor, must not copy empty ternary tree
166     TernaryTree( const TTree& aOtherTTree )
167     {
168         for (int i = 0; i < 3; i++)
169         {
170             fSubTrees[i] = &NIL;
171         }
172         *this = aOtherTTree;
173     }
174 
175     // copy assignment operator, must not copy empty ternary tree
176     // may throw a domain error on attempts to copy NIL
177     TTree& operator=(const TTree& aOtherTTree)
178     {
179         if (this != &aOtherTTree)
180             if (!aOtherTTree.empty())
181             {
182                 this->~TernaryTree();
183                 fKey = aOtherTTree.fKey;
184                 for (size_t i = 0; i < 3; i++)
185                 {
186                     if (!aOtherTTree.fSubTrees[i]->empty())
187                     {
188                         fSubTrees[i] = aOtherTTree.fSubTrees[i]->clone();
189                     }
190                     else
191                     {
192                         fSubTrees[i] = &NIL;
193                     }
194                 }
```

```
195             }
196             else
197             {
198                 throw domain_error("NIL as source not permitted.");
199             }
200         return *this;
201     }
202
203     // clone ternary tree, must not copy empty trees
204     TSubTree clone() const
205     {
206         if (empty())
207         {
208             throw domain_error("NIL as source not permitted.");
209         }
210         return new TTree(*this);
211     }
212
213 ////////////////////////////////////////////////////////////////////////
214 // Problem 3: TernaryTree Move Semantics
215
216     // TTree r-value constructor
217     TernaryTree( T&& aKey ): fKey(std::move(aKey))
218     {
219         for (int i = 0; i < 3; i++)
220         {
221             fSubTrees[i] = &NIL;
222         }
223     }
224
225     // move constructor, must not copy empty ternary tree
226     TernaryTree( TTree&& aOtherTTree )
227     {
228         for (int i = 0; i < 3; i++)
229         {
230             fSubTrees[i] = &NIL;
231         }
232         *this = move(aOtherTTree);
233     }
234
235     // move assignment operator, must not copy empty ternary tree
236     TTree& operator=( TTree&& aOtherTTree )
237     {
238         if (this != &aOtherTTree)
239         {
240             if (!aOtherTTree.empty())
241             {
242                 this->~TernaryTree();
243                 fKey = std::move(aOtherTTree.fKey);
```

```
244                 for (int i = 0; i < 3; i++)
245                 {
246                     if (!aOtherTTree.fSubTrees[i]->empty()) fSubTrees[i] = const_cast<TSubTree>(&aOtherTTree.removeSubTree(i));
247                     else fSubTrees[i] = &NIL;
248                 }
249             }
250             else
251             {
252                 throw std::domain_error("NIL as source not permitted.");
253             }
254         }
255     }
256 
257 ////////////////////////////////////////////////////////////////////////
258 // Problem 4: TernaryTree Prefix Iterator
259 
260     // return ternary tree prefix iterator positioned at start
261     Iterator begin() const
262     {
263         return Iterator(this).begin();
264     }
265 
266     // return ternary prefix iterator positioned at end
267     Iterator end() const
268     {
269         return Iterator(this).end();
270     }
271 };
272 
273 template<typename T>
274 TernaryTree<T> TernaryTree<T>::NIL;
275 
```

```
 1
 2  // COS30008, Final Exam, 2022
 3
 4  #pragma once
 5
 6  #include "TernaryTree.h"
 7
 8  #include <stack>
 9
10  template<typename T>
11  class TernaryTreePrefixIterator
12  {
13  private:
14      using TTree = TernaryTree<T>;
15      using TTreeNode = TTree*;
16      using TTreeStack = std::stack<const TTree*>;
17
18      const TTree* fTTree;                // ternary tree
19      TTreeStack fStack;                  // traversal stack
20
21  public:
22
23      using Iterator = TernaryTreePrefixIterator<T>;
24
25      Iterator operator++(int)
26      {
27          Iterator old = *this;
28
29          ++(*this);
30
31          return old;
32      }
33
34      bool operator!=( const Iterator& aOtherIter ) const
35      {
36          return !(*this == aOtherIter);
37      }
38
39  ////////////////////////////////////////////////////////////////////////
40  // Problem 4: TernaryTree Prefix Iterator
41
42  private:
43
44      // push subtree of aNode [30]
45      void push_subtrees( const TTree* aNode )
46      {
47          if (!(*aNode).getRight().empty())
48          {
49              fStack.push(const_cast<TTreeNode>(&(*aNode).getRight()));
```

```
50          }
51          if (!(*aNode).getMiddle().empty())
52          {
53              fStack.push(const_cast<TTreeNode>(&(*aNode).getMiddle()));
54          }
55          if (!(*aNode).getLeft().empty())
56          {
57              fStack.push(const_cast<TTreeNode>(&(*aNode).getLeft())); 5;
58          }
59      }
60
61  public:
62
63      // iterator constructor [12]
64      TernaryTreePrefixIterator( const TTree* aTTree ): fTTree(aTTree),
          fStack()
65      {
66          if (!(*fTTree).empty())
67          {
68              fStack.push(const_cast<TTreeNode>(fTTree));
69          }
70      }
71
72      // iterator dereference [8]
73      const T& operator*() const
74      {
75          return **fStack.top();
76      }
77
78      // prefix increment [12]
79      Iterator& operator++()
80      {
81          TTreeNode lPopped = const_cast<TTreeNode>(fStack.top());
82          fStack.pop();
83          push_subtrees(lPopped);
84          return *this;
85      }
86
87      // iterator equivalence [12]
88      bool operator==( const Iterator& aOtherIter ) const
89      {
90          return fTTree == aOtherIter.fTTree && fStack.size() ==
              aOtherIter.fStack.size();
91      }
92
93      // auxiliaries [4,10]
94      Iterator begin() const
95      {
96          Iterator temp = *this;
```

```
 97          temp.fStack = TTreeStack();
 98          temp.fStack.push(const_cast<TTreeNode>(temp.fTTree));
 99          return temp;
100      }
101      Iterator end() const
102      {
103          Iterator temp = *this;
104          temp.fStack = TTreeStack();
105          return temp;
106      }
107  };
108
```

Problem 5 (50 marks)

Answer the following questions in one or two sentences:

a. How can we construct a tree where all nodes have the same degree? [4]

5a)

Set the max number of nodes:
- max number of nodes at x level = 2^x when x >= 0
- max number of nodes in a tree of y height = 2^2*y - 1

b. What is the difference between l-value and r-value references? [6]

5b)

- l-value denoted a memory location identifies an object
- r-value denotes the data stored in a memory address

c. What is a key concept of an abstract data types? [4]

5c)

An object's behavior can be described by a set of values and a set of actions, and this behavior is known as an abstract data type (ADT). The definition of ADT merely specifies the actions that must be taken, not how they must be carried out. It is unclear what algorithms will be utilized to carry out the operations and how the data will be structured in memory. Because it provides an implementation-independent view, it is dubbed "abstract."

d. How do we define mutual dependent classes in C++? [4]

5d)

```
class Employee; // Employee.h
#include "Employee.h" // Company.h
class Company
{
private:
...
public:
...
}
```

e. What must a value-based data type define in C++? [2]

5e)

Data types specify the kind of value that can be found in the data. Values-based data types maintain their own memory space for storing data. This include bool, char, etc.

f. What is an object adapter? [6]

5f)

The main method for an object to access ORB services like object reference generation is through an object adapter. Standard interfaces are exported to the object through a portable object adaptor. An object adapter's primary duties are the generation and interpretation of object references.

g. What is the difference between copy constructor and assignment operator and how do we guarantee safe operation? [8]

5g)

Copy constructor:
- When an existing item is copied to generate a new one, this process is known as duplication.
- The new item is given its own memory block.
- It is a constructor that is overloaded.
- In the absence of a copy constructor defined in the class, the C++ compiler implicitly provides one.

Assignment operator:
- This operator is called when an already initialized object is assigned a new value from another existing object.
- It does not create a separate memory block or new memory space.
- It is a bitwise operator.
- A bitwise copy gets created, if the Assignment operator is not overloaded.

h. What is the best-case, average-case, and worse-case for a lookup in a binary tree? [6]

5h)

The worst-case scenario is when the search list's first item is the target object O(1).
The worst-case scenario has the longest completion time and requires the most input.
The average-case involves taking an average number of steps with n-element input data O(log n) (log n).

i. What are reference data members and how do we initialize them? [2]

5i)

An object of type T, a function of type T, or an object that is implicitly convertible to type T can be used to initialize a reference to that type. A reference cannot be modified to refer to a different object after it has been initialized.

j. You are given 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]

5j)

Sort the supplied array.
- We locate the first positive number in the array after sorting.
- As we now iterate through the array, we keep printing elements in the spaces between any two adjacent array elements.
- The largest array element is printed if gaps don't fill in the k missing numbers.