Swinburne University of Technology

School of Science, Computing and Engineering Technologies

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures and Patterns
Assignment number and title: 4, A Tree-like Priority Queue
Due date: Friday, May 26, 2023, 23:59
Lecturer: Dr. Markus Lumpe

Your name: _____   **Your student id:** _____

Check Tutorial	Tues 08:30	Tues 10:30	Tues 12:30 BA603	Tues 12:30 ATC627	Tues 14:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30	Thurs 08:30	Thurs 10:30

Marker's comments:

Problem	Marks	Obtained
1	66	
Total	66	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener: _____

```cpp
1
2   // COS30008, Problem Set 4, 2023
3
4   #pragma once
5
6   #include <vector>
7   #include <optional>
8   #include <algorithm>
9
10  template<typename T, typename P>
11  class PriorityQueue
12  {
13  private:
14
15      struct Pair
16      {
17          P priority;
18          T payload;
19
20          Pair( const P& aPriority, const T& aPayload ) :
21              priority(aPriority),
22              payload(aPayload)
23          {}
24      };
25
26      std::vector<Pair> fHeap;
27
28      /*
29
30       In the array representation, if we are starting to count indices from 0,
31       the children of the i-th node are stored in the positions (2 * i) + 1 and
32       2 * (i + 1), while the parent of node i is at index (i - 1) / 2 (except
33       for the root, which has no parent).
34
35       */
36
37      void bubbleUp( size_t aIndex ) noexcept
38      {
39          if ( aIndex > 0 )
40          {
41              Pair lCurrent = fHeap[aIndex];
42
43              do
44              {
45                  size_t lParentIndex = (aIndex - 1) / 2;
46
47                  if ( fHeap[lParentIndex].priority < lCurrent.priority )
48                  {
49                      fHeap[aIndex] = fHeap[lParentIndex];
```

```cpp
50                            aIndex = lParentIndex;
51                        }
52                        else
53                        {
54                            break;
55                        }
56                } while (aIndex > 0);
57
58                fHeap[aIndex] = lCurrent;
59            }
60        }
61
62        void pushDown( size_t aIndex = 0 ) noexcept
63        {
64            if ( fHeap.size() > 1 )
65            {
66                size_t lFirstLeafIndex = ((fHeap.size() - 2) / 2) + 1;
67
68                if ( aIndex < lFirstLeafIndex )
69                {
70                    Pair lCurrent = fHeap[aIndex];
71
72                    do
73                    {
74                        size_t lChildIndex = (2 * aIndex) + 1;
75                        size_t lRight = 2 * (aIndex + 1);
76
77                        if ( lRight < fHeap.size() && fHeap[lChildIndex].priority < ⏎
                            fHeap[lRight].priority )
78                        {
79                            lChildIndex = lRight;
80                        }
81
82                        if ( fHeap[lChildIndex].priority > lCurrent.priority )
83                        {
84                            fHeap[aIndex] = fHeap[lChildIndex];
85                            aIndex = lChildIndex;
86                        }
87                        else
88                        {
89                            break;
90                        }
91
92                    } while ( aIndex < lFirstLeafIndex );
93
94                    fHeap[aIndex] = lCurrent;
95                }
96            }
97        }
```

```cpp
 98
 99  public:
100
101      size_t size() const noexcept {
102          return fHeap.size();
103      }
104
105      std::optional<T> front() noexcept {
106          if (size() > 0) {
107              Pair front = fHeap[0];
108              std::swap(fHeap[0], fHeap[size() - 1]);
109              fHeap.pop_back();
110              if (size() > 0)  pushDown();
111              return front.payload;
112          }
113          return std::optional<T>();
114      }
115      void insert(const T& aPayload, const P& aPriority) noexcept {
116          fHeap.push_back(Pair(aPriority, aPayload));
117          bubbleUp(size() - 1);
118      }
119      void update(const T& aPayload, const P& aNewPriority) noexcept {
120          size_t index{0};
121          bool found = false;
122          for (; index < size(); index++) {
123              if (fHeap[index].payload == aPayload) {
124                  found = true;
125                  break;
126              }
127          }
128          if (index) {
129              P oldPriority = fHeap[index].priority;
130              fHeap[index].priority = aNewPriority;
131              fHeap[index].priority > oldPriority ? bubbleUp(index) : pushDown     ↵
                     (index);
132          }
133      }
134  };
135
```