Swinburne University of Technology

School of Science, Computing and Engineering Technologies

MIDTERM COVER SHEET

Check 08:30 10:30 12:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 12:30 14:30 12:30 12:30 14:30 12:30 12:30 14:30 12:3	Subject Code:					COS30008							
Thursday, April 27, 2023, 23:59 Dr. Markus Lumpe Total	Subject Title:					Data Structures and Patterns							
Your name:	Assig	nment n	e: Mid	term									
Your name: Your student ID: Check Tues 08:30 Tues 10:30 Tues 12:30 Tues 12:30 Tues 12:30 Med 10:30 Wed 10:30 Wed 12:30 Wed 12:30 It will will will will will will will wil	Due o	late:			Thu	ırsday, <i>F</i>	April 27, 2	2023, 23	:59				
Tues	Lectu	irer:			Dr.	Markus	Lumpe						
Check 08:30 10:30 12:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 10:30 12:30 14:30 08:30 12:30 14:30 12:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 14:30 12:30 12:30 14:30 12:30 12:30 14:30 12:3	Your name:					Your student ID:							
Marker's comments: Problem	Check			12:30	12:30							TI	
2 74 3 108 Total 234 Extension certification: This assignment has been given an extension and is now due on	Marke											_	
Total 234 Extension certification: This assignment has been given an extension and is now due on	Marke		oblem							Obtaine	ed		
Total 234 Extension certification: This assignment has been given an extension and is now due on	Marke		oblem 1				52			Obtaine	ed		
Extension certification: This assignment has been given an extension and is now due on	Marke		oblem 1 2				52 74			Obtaine	ed		
This assignment has been given an extension and is now due on	Marke	Pro	oblem 1 2 3			1	52 74 08			Obtaine	ed		
This assignment has been given an extension and is now due on	Marke	Pro	oblem 1 2 3			1	52 74 08			Obtaine	ed		
		Pro	oblem 1 2 3 Fotal			1	52 74 08			Obtaine	ed		
		Pro	oblem 1 2 3 Fotal	on:		1	52 74 08			Obtaine	ed		
Signature of Convener:	Exter	Pro	oblem 1 2 3 Total		an exter	1 2	52 74 .08 .34	due on					

```
2 //Sartaj Khan Midterm
 3 //Date: 26/04/23
 4
 6 #include "PrefixString.h"
 7 #include <cassert>
9
10 PrefixString::PrefixString(char aExtension) noexcept :
       fCode(static_cast<uint16_t>(-1)),
       fPrefix(static_cast<uint16_t>(-1)),
12
       fExtension(aExtension)
13
14 {}
15
16 PrefixString::PrefixString(uint16_t aPrefix, char aExtension) noexcept :
17
       fCode(static_cast<uint16_t>(-1)),
       fPrefix(aPrefix),
18
       fExtension(aExtension)
19
20 {}
21
22 uint16_t PrefixString::getCode() const noexcept {
23
       return fCode;
24 }
25
26 void PrefixString::setCode(uint16_t aCode) noexcept {
27
       fCode = aCode;
28 }
29
30 uint16_t PrefixString::w() const noexcept {
       return fPrefix;
31
32 }
33
34 char PrefixString::K() const noexcept {
35
       return fExtension;
36 }
37
38 PrefixString PrefixString::operator+(char aExtension) const noexcept {
39
       assert(fCode != -1);
40
       return PrefixString(fCode, aExtension);
41 }
42
43 bool PrefixString::operator==(const PrefixString& aOther) const noexcept {
       return fPrefix == aOther.w() && fExtension == aOther.K();
45 }
46
47 std::ostream& operator<<(std::ostream& aOStream, const PrefixString& aObject) {
       return aOStream << "(" << aObject.getCode() << "," << aObject.w() << "," << >
48
         aObject.K() << ")";
```

```
2 //Sartaj Khan
 3 //Date: 26/04/23
 5 #include "LZWTable.h"
 6 #include <cassert>
 7
 8 LZWTable::LZWTable(uint16_t aInitialCharacters) :
 9
        fIndex(0),
        fInitialCharacters(aInitialCharacters)
10
11 {}
12
13 void LZWTable::initialize() {
14
        for (size_t i = 0; i < 128; i++) {
15
            fEntries[fIndex] = PrefixString((char)i);
            fEntries[fIndex++].setCode(i);
16
17
        }
18 }
19
20 const PrefixString& LZWTable::lookupStart(char aK) const noexcept {
21
        assert(0 <= aK < fIndex);</pre>
        PrefixString found = PrefixString();
22
23
        for (PrefixString aPrefixString : fEntries) {
            if (aPrefixString.K() == aK) {
24
25
                found = aPrefixString;
26
                break;
27
            }
28
29
        return found;
30 }
31
   bool LZWTable::contains(PrefixString& aWK) const noexcept {
33
        assert(aWK.w() != static_cast<uint16_t>(-1));
34
        for (size_t index = fIndex - 1; index >= aWK.w(); index--) {
35
            if (fEntries[index] == aWK) {
                aWK = fEntries[index];
36
37
                return true;
38
            }
39
        }
40
        return false;
41 }
42
43 void LZWTable::add(PrefixString& aWK) noexcept {
        assert(aWK.w() != static_cast<uint16_t>(-1));
        aWK.setCode(fIndex);
45
        fEntries[fIndex++] = aWK;
46
47 }
```

```
2 //Sartaj Khan
 3 //Date: 26/04/23
 5 #include "LZWCompressor.h"
 7
 8
   bool LZWCompressor::readK() noexcept {
 9
        if (fInput[fIndex + 1]) {
            fK = fInput[++fIndex];
10
11
            return true;
12
        }
        fK = -1;
13
        return false;
14
15 }
16
17
18 void LZWCompressor::start() {
19
        fTable = LZWTable();
        fTable.initialize();
20
21
        fK = fInput[0];
        fW = fTable.lookupStart(fK);
22
23
        fCurrentCode = nextCode();
24 }
25
26
   uint16_t LZWCompressor::nextCode() {
28
        if (fK == -1) {
29
            return -1;
30
31
        while (readK()) {
32
            PrefixString next = fW + fK;
33
            if (fTable.contains(next)) {
34
                fW = next;
35
            }
            else {
36
37
                fTable.add(next);
38
                uint16_t code = fW.getCode();
                fW = fTable.lookupStart(fK);
39
40
                return code;
41
            }
42
43
        return fW.getCode();
44 }
45
46
   LZWCompressor::LZWCompressor(const std::string& aInput) :
48
        fInput(aInput),
        fIndex(0)
49
```

```
...winburne University\DSP\Midterm\Midterm\LZWCompressor.cpp
```

```
2
```

```
50 {
51
       start();
52 }
53
54 const uint16_t& LZWCompressor::operator*() const noexcept {
        return fCurrentCode;
56 }
57
58 LZWCompressor& LZWCompressor::operator++() noexcept {
59
       fCurrentCode = nextCode();
60
       return *this;
61 }
62
63 LZWCompressor LZWCompressor::operator++(int) noexcept {
64
       LZWCompressor old = *this;
       ++(*this);
65
66
       return old;
67 }
68
69 bool LZWCompressor::operator==(const LZWCompressor& a0ther) const noexcept {
       return (fInput == a0ther.fInput) && (fIndex == a0ther.fIndex && fK ==
70
         aOther.fK && fCurrentCode == aOther.fCurrentCode);
71 }
72
73 bool LZWCompressor::operator!=(const LZWCompressor& a0ther) const noexcept {
74
       return !(*this == a0ther);
75 }
76
77 LZWCompressor LZWCompressor::begin() const noexcept {
78
       LZWCompressor begin = *this;
79
       begin.fIndex = 0;
80
       begin.start();
81
       return begin;
82 }
83
84 LZWCompressor LZWCompressor::end() const noexcept {
85
       LZWCompressor end = *this;
86
       end.fIndex = end.fInput.size() - 1;
       end.fCurrentCode = -1;
87
88
       end.fK = -1;
89
       return end;
90 }
91
```