Swinburne University of Technology

School of Science, Computing and Engineering Technologies

MIDTERM COVER SHEET

Subject Code:	COS30008
Subject Title:	Data Structures and Patterns
Assignment number and title:	Midterm
Due date:	Thursday, April 27, 2023, 23:59
Lecturer:	Dr. Markus Lumpe

Your name: Sartaj Khan **Your student ID:** 103509694

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
								X			

Marker's comments:

Problem	Marks	Obtained
1	52	
2	74	
3	108	
Total	234	

Extension certification:

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

Signature of Convener:________________

```cpp

//Sartaj Khan Midterm
//Date: 26/04/23


#include "PrefixString.h"
#include <cassert>


PrefixString::PrefixString(char aExtension) noexcept :
    fCode(static_cast<uint16_t>(-1)),
    fPrefix(static_cast<uint16_t>(-1)),
    fExtension(aExtension)
{}

PrefixString::PrefixString(uint16_t aPrefix, char aExtension) noexcept :
    fCode(static_cast<uint16_t>(-1)),
    fPrefix(aPrefix),
    fExtension(aExtension)
{}

uint16_t PrefixString::getCode() const noexcept {
    return fCode;
}

void PrefixString::setCode(uint16_t aCode) noexcept {
    fCode = aCode;
}

uint16_t PrefixString::w() const noexcept {
    return fPrefix;
}

char PrefixString::K() const noexcept {
    return fExtension;
}

PrefixString PrefixString::operator+(char aExtension) const noexcept {
    assert(fCode != -1);
    return PrefixString(fCode, aExtension);
}

bool PrefixString::operator==(const PrefixString& aOther) const noexcept {
    return fPrefix == aOther.w() && fExtension == aOther.K();
}

std::ostream& operator<<(std::ostream& aOStream, const PrefixString& aObject) {
    return aOStream << "(" << aObject.getCode() << "," << aObject.w() << "," << 
      aObject.K() << ")";
}
```

```

//Sartaj Khan
//Date: 26/04/23

#include "LZWTable.h"
#include <cassert>

LZWTable::LZWTable(uint16_t aInitialCharacters) :
    fIndex(0),
    fInitialCharacters(aInitialCharacters)
{}

void LZWTable::initialize() {
    for (size_t i = 0; i < 128; i++) {
        fEntries[fIndex] = PrefixString((char)i);
        fEntries[fIndex++].setCode(i);
    }
}

const PrefixString& LZWTable::lookupStart(char aK) const noexcept {
    assert(0 <= aK < fIndex);
    PrefixString found = PrefixString();
    for (PrefixString aPrefixString : fEntries) {
        if (aPrefixString.K() == aK) {
            found = aPrefixString;
            break;
        }
    }
    return found;
}

bool LZWTable::contains(PrefixString& aWK) const noexcept {
    assert(aWK.w() != static_cast<uint16_t>(-1));
    for (size_t index = fIndex - 1; index >= aWK.w(); index--) {
        if (fEntries[index] == aWK) {
            aWK = fEntries[index];
            return true;
        }
    }
    return false;
}

void LZWTable::add(PrefixString& aWK) noexcept {
    assert(aWK.w() != static_cast<uint16_t>(-1));
    aWK.setCode(fIndex);
    fEntries[fIndex++] = aWK;
}
```

```cpp

//Sartaj Khan
//Date: 26/04/23

#include "LZWCompressor.h"


bool LZWCompressor::readK() noexcept {
    if (fInput[fIndex + 1]) {
        fK = fInput[++fIndex];
        return true;
    }
    fK = -1;
    return false;
}


void LZWCompressor::start() {
    fTable = LZWTable();
    fTable.initialize();
    fK = fInput[0];
    fW = fTable.lookupStart(fK);
    fCurrentCode = nextCode();
}


uint16_t LZWCompressor::nextCode() {
    if (fK == -1) {
        return -1;
    }
    while (readK()) {
        PrefixString next = fW + fK;
        if (fTable.contains(next)) {
            fW = next;
        }
        else {
            fTable.add(next);
            uint16_t code = fW.getCode();
            fW = fTable.lookupStart(fK);
            return code;
        }
    }
    return fW.getCode();
}


LZWCompressor::LZWCompressor(const std::string& aInput) :
    fInput(aInput),
    fIndex(0)
```

```
{
    start();
}

const uint16_t& LZWCompressor::operator*() const noexcept {
    return fCurrentCode;
}

LZWCompressor& LZWCompressor::operator++() noexcept {
    fCurrentCode = nextCode();
    return *this;
}

LZWCompressor LZWCompressor::operator++(int) noexcept {
    LZWCompressor old = *this;
    ++(*this);
    return old;
}

bool LZWCompressor::operator==(const LZWCompressor& aOther) const noexcept {
    return (fInput == aOther.fInput) && (fIndex == aOther.fIndex && fK == 
      aOther.fK && fCurrentCode == aOther.fCurrentCode);
}

bool LZWCompressor::operator!=(const LZWCompressor& aOther) const noexcept {
    return !(*this == aOther);
}

LZWCompressor LZWCompressor::begin() const noexcept {
    LZWCompressor begin = *this;
    begin.fIndex = 0;
    begin.start();
    return begin;
}

LZWCompressor LZWCompressor::end() const noexcept {
    LZWCompressor end = *this;
    end.fIndex = end.fInput.size() - 1;
    end.fCurrentCode = -1;
    end.fK = -1;
    return end;
}

```