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: Ekrar Efaz **Your student ID:** 103494172

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
//
// Created by Ekrar Efaz on 25/4/23.
//
#include <cassert>

#include "PrefixString.h"


PrefixString::PrefixString( char aExtension ) noexcept:
        // static cast necessary because uintt16_t representation of -1 required
        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 != static_cast<uint16_t>(-1)); // ensuring operation allowed on present strings in table
    return PrefixString(fCode,aExtension);
}

bool PrefixString::operator==( const PrefixString& aOther ) const noexcept{
    return (this->fPrefix == aOther.fPrefix && this->fExtension == aOther.fExtension);
}

std::ostream& operator<<( std::ostream& aOStream, const PrefixString& aObject ){
    return aOStream << "(" << aObject.fCode << "," << aObject.fPrefix << "," << aObject.fExtension << ")";
}
```

```cpp
//
// Created by Ekrar Efaz on 26/4/23.
//
#include <cassert>
#include <iostream>

#include "LZWTable.h"

LZWTable::LZWTable( uint16_t aInitialCharacters){
    // task constraint 128 single character prefix strings
    if(aInitialCharacters <= 128){
        fInitialCharacters = aInitialCharacters;
    }
    else{
        fInitialCharacters = 128;
    }
    initialize();
}

void LZWTable::initialize(){
    // loop for the number of initial characters
    for(size_t i = 0; i < fInitialCharacters; i++){
        PrefixString aPrefixString(static_cast<char>(i));
        // set the code to its corresponding entry in ASCII table
        aPrefixString.setCode(i);
        fEntries[i] = aPrefixString;
    }
    // set the index to the number of initial characters added
    fIndex = fInitialCharacters;
}

const PrefixString& LZWTable::lookupStart( char aK ) const noexcept{
    // assertion required to check if out of bounds
    assert(static_cast<uint16_t>(aK) < fInitialCharacters);
    // implicit conversion happens from char to uint_16
    return fEntries[aK];
}

bool LZWTable::contains(PrefixString &aWK) const noexcept {
    assert(aWK.w() < fIndex);
    for (int i = aWK.w(); i < fIndex; i++) {
        if (fEntries[i] == aWK) {
            aWK = fEntries[i];
            return true;
        }
    }
    return false;
}
void LZWTable::add( PrefixString& aWK ) noexcept{
    assert(aWK.w() != static_cast<uint16_t>(-1));
    aWK.setCode(fIndex);
    fEntries[fIndex++] = aWK;
}
```

```
//
// Created by Ekrar Efaz on 26/4/23.
//

#include <iostream>
#include "LZWCompressor.h"

bool LZWCompressor::readK() noexcept{
    //Code to read input character from fInput
    if (fIndex >= fInput.length()) {
        fK = -1;
        return false;
    }
    fK = fInput[fIndex];
    return true;
}
void LZWCompressor::start(){
    //Code to read the first character from fInput
    if(readK()){
        fTable.initialize();
        //Code to initialize fW
        fW = fTable.lookupStart(fK);
        fCurrentCode = nextCode();
    }
}
uint16_t LZWCompressor::nextCode() {
    if (fK == -1) {
        return -1;
    }
    while (readK() && fK != -1) {
        // Code to check if fW + fK is in the table
        PrefixString aWK = fW + fK;
        if (fTable.contains(aWK)) {
            // Code to set fW to fW + fK
            fW = aWK;
        }
        else {
            // Code to add fW + fK to the table
            fTable.add(aWK);
            // Code to output the code for fW
            fW = fTable.lookupStart(fK);
            return fW.getCode();
        }
    }
    return fW.getCode();
}

LZWCompressor::LZWCompressor(const std::string& aInput) : fInput(aInput), fIndex(0), fK(-1), fCurrentCode(0){
    start();
}
const uint16_t& LZWCompressor::operator*() const noexcept{
    return fCurrentCode;
}
LZWCompressor& LZWCompressor::operator++() noexcept {
    if (fIndex >= fInput.length()) {
        return *this; // already at the end
    }
    fCurrentCode = nextCode();
    fIndex++;
    return *this;
}
LZWCompressor LZWCompressor::operator++(int) noexcept {
    LZWCompressor temp = *this;
    ++(*this);
    return temp;
}
bool LZWCompressor::operator==( const LZWCompressor& aOther ) const noexcept{
    return (this->fInput == aOther.fInput &&
            this->fIndex == aOther.fIndex &&
            this->fCurrentCode == aOther.fCurrentCode &&
            this->fK == aOther.fK);
}
bool LZWCompressor::operator!=( const LZWCompressor& aOther ) const noexcept{
    return !((this->fInput == aOther.fInput &&
              this->fIndex == aOther.fIndex &&
```

```
            this->fCurrentCode == aOther.fCurrentCode &&
            this->fK == aOther.fK));
}
LZWCompressor LZWCompressor::begin() const noexcept {
    LZWCompressor iter(*this); // make a copy of the current object
    iter.start();
    iter.fIndex = 0;
    return iter;
}
LZWCompressor LZWCompressor::end() const noexcept {
    LZWCompressor iter(fInput);
    iter.fIndex = fInput.length();
    return iter;
}
```