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: Md Redwan Ahmed Zawad **Your student ID: 103501849**

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	52	
2	74	
3	108	
Total	234	

Extension certification:

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

Signature of Convener:_____

```
#include "PrefixString.h"
PrefixString::PrefixString(char aExtension) noexcept
{
	fCode = static_cast<uint16_t>(-1);
	fExtension = aExtension;
	fPrefix = static_cast<uint16_t>(-1);
}

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

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

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

PrefixString PrefixString:: operator+(char aExtension) const noexcept
{
	PrefixString lPrefix;
	if(fCode!=-1)
	{
		lPrefix.fPrefix = fCode;
		lPrefix.fCode = -1;
		lPrefix.fExtension = aExtension;
	}
	return lPrefix;
}

bool PrefixString:: operator==(const PrefixString& aOther) const noexcept
{
	if (fPrefix == aOther.fPrefix && fExtension == aOther.fExtension) {
		return true;
	}
	else {
		return false;
	}
}

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


#include "LZWTable.h"

LZWTable::LZWTable(uint16_t aInitialCharacter)
{
	fInitialCharacters = aInitialCharacter;
	fIndex = 0;
	initialize();
}

void LZWTable::initialize()
{
	while (fIndex < 128)
	{
		fEntries[fIndex] = PrefixString(fIndex);
		fEntries[fIndex].setCode(fIndex);

		fIndex+=1;
```

```
	}

}

const PrefixString& LZWTable::lookupStart(char aK) const noexcept
{
	return fEntries[aK];
}

bool LZWTable::contains(PrefixString& aWK) const noexcept
{

	if (aWK.w() != -1)
	{

		for (uint16_t i = fIndex; i >= aWK.w(); i--)
		{
			if (fEntries[i] == aWK)
			{
				aWK = fEntries[i];
				return true;
			}

		}
	}
	return false;

}

void LZWTable::add(PrefixString& aWK)noexcept
{

	if (aWK.w() != -1)
	{

		aWK.setCode(fIndex);
		fEntries[fIndex++] = aWK;




	}

}

#include "LZWCompressor.h"
#include<iostream>
LZWCompressor::LZWCompressor(const std::string& aInput):
	fTable(),
	fW()
{

	fInput = aInput;
	fIndex = 0;
	fK = -1;
	fCurrentCode = 0;
	start();
}

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

void LZWCompressor::start()
{
```

```
	fTable.initialize();
	readK();
	fW = fTable.lookupStart(fK);

	//fW.setCode(fK);
	fCurrentCode = nextCode();
}

uint16_t LZWCompressor::nextCode()
{



	if(fK!=-1)
	{
		while (readK())
		{



			PrefixString lwK = fW + fK;
			if (fTable.contains(lwK))
			{
				fW = lwK;
			}
			else {
				uint16_t lResult = lwK.w();
				fTable.add(lwK);

				fW = fTable.lookupStart(lwK.K());
				//fW.setCode(fK);
				return lResult;
			}
			//std::cout << fCurrentCode;


		}
	}
	else {
		return -1;
	}

	 return fW.getCode();


}

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

LZWCompressor& LZWCompressor:: operator++()noexcept
{
	if (fK != -1)
	{
		fCurrentCode = nextCode();
	}
	else {
		fCurrentCode = -1;
	}

	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 Result = LZWCompressor(fInput);

	return Result;

}

LZWCompressor LZWCompressor::end()const noexcept
{
	LZWCompressor Result = *this;

	Result.fIndex = fInput.size();
	Result.fK = -1;
	Result.fCurrentCode = -1;
	return Result;


}
```