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: Brandy Dan **Your student ID:** 103864526

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"
#include <assert.h>

PrefixString::PrefixString(char aExtension)noexcept :
	fCode(-1),
	fPrefix(-1),
	fExtension(aExtension)
{}

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


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


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


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


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

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

	return PrefixString(this->getCode(), aExtension);


}

bool PrefixString::operator==(const PrefixString& aOther) const noexcept
{
	if (this->w() == aOther.w() && this->K() == aOther.K())
	{
		return true;
```

```
    }

    return false;
}

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

```
#include "LZWTable.h"
#include <assert.h>



void LZWTable::initialize()
{

	for (; fIndex < fInitialCharacters; fIndex++)
	{
		fEntries[fIndex] = PrefixString(fIndex);
		fEntries[fIndex].setCode(fIndex);
	}

}

LZWTable::LZWTable(uint16_t aInitialCharacters) :
	fEntries(),
	fIndex(0),
	fInitialCharacters(128)
{
	initialize();
}

const PrefixString& LZWTable::lookupStart(char aK) const noexcept
{
	assert(static_cast<int>(aK) < 128);

	for (int i = 0; i < 128; i++)
	{
		if (fEntries[i].K() == aK)
		{
			return fEntries[i];
		}
	}
}

bool LZWTable::contains(PrefixString& aWK) const noexcept
{
	assert(aWK.w() != -1);

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

}
```

```
void LZWTable::add(PrefixString& aWK) noexcept
{
	assert(aWK.w() != -1);

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

```
#include "LZWCompressor.h"

bool LZWCompressor::readK() noexcept
{
	fIndex++;

	if (fIndex <= fInput.size())
	{

		fK = fInput[fIndex];
		return true;

	}

	fK = -1;
	return false;


}

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

uint16_t LZWCompressor::nextCode()
{

	PrefixString wk;
	uint16_t out;

	if (fK == -1)
	{
		return	-1;
	}

	else
	{
		while (readK())
		{

			PrefixString wk = fW + fK;

			if (fTable.contains(wk))
			{
				fW = wk;

			}
			else
			{
				out = fW.getCode();
				fTable.add(wk);
				fW = fTable.lookupStart(fK);
				return out;
			}
		}
		return fW.getCode();
	}


}


LZWCompressor::LZWCompressor(const std::string& aInput) :
	fInput(aInput),
	fIndex(0),
	fCurrentCode(),
	fK(),
	fW(),
	fTable()
{
	start();
}

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 fIndex == aOther.fIndex && fCurrentCode == aOther.fCurrentCode && fK == aOther.fK;
}

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

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

LZWCompressor LZWCompressor::begin() const noexcept
```

```
{
	LZWCompressor copy = *this;
	copy.start();
	copy.fCurrentCode = this->fCurrentCode;
	return copy;

}

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

```
// COS30008, Midterm 2023

#include <iostream>

//#define P1
//#define P2
//#define P3

#ifdef P1

#include "PrefixString.h"

void runP1()
{
	std::cout << "Test PrefixString:\n" << std::endl;

	PrefixString lString0;
	PrefixString lStringA( 'a' );
	PrefixString lStringB( 'b' );
	PrefixString lStringAB( 97, 'b' );
	PrefixString lStringBA( 98, 'a' );

	lStringA.setCode( 97 );
	lStringB.setCode( 98 );
	lStringAB.setCode( 127 );
	lStringBA.setCode( 128 );

	std::cout
		<< "0 string ::= "
		<< "code= " << lString0.getCode()
		<< ", w = " << lString0.w()
		<< ", K = " << lString0.K() << std::endl;

	std::cout
		<< "A string ::= "
		<< "code= " << lStringA.getCode()
		<< ", w = " << lStringA.w()
		<< ", K = " << lStringA.K() << std::endl;

	std::cout
		<< "BA string ::= "
		<< "code= " << lStringBA.getCode()
		<< ", w = " << lStringBA.w()
		<< ", K = " << lStringBA.K() << std::endl;

	PrefixString lW = lStringB + 'a';

	std::cout << "lW == lStringBA? ";
	std::cout << (lW == lStringBA ? "true" : "false") << std::endl;

	if ( lW == lStringBA )
	{
        lW.setCode( lStringBA.getCode() );
    }
    else
    {
        lW.setCode( 129 );
    }

    std::cout << "All strings:" << std::endl;
    std::cout << "lString0 = " << lString0 << std::endl;
    std::cout << "lStringA = " << lStringA << std::endl;
    std::cout << "lStringB = " << lStringB << std::endl;
    std::cout << "lStringAB = " << lStringAB << std::endl;
    std::cout << "lStringBA = " << lStringBA << std::endl;
    std::cout << "lW = " << lW << std::endl;

    std::cout << "\nPrefixString test complete." << std::endl;
}

#endif

#ifdef P2

#include "LZWTable.h"

void runP2()
{
    std::cout << "Test LZW Table:\n" << std::endl;

    LZWTable lTable ( 456 );

    lTable.initialize();

    std::cout << "LZW Table contains 128 entries." << std::endl;
    std::cout << "Next available index is 128." << std::endl;

    PrefixString lA = lTable.lookupStart( 'a' );

    std::cout << "lA = " << lA << std::endl;

    PrefixString lW_1 = lA + 'b';

    std::cout << "Is lW_1 = " << lW_1 << " in LZW table? ";

    if ( lTable.contains( lW_1 ) )
    {
        std::cout << "Yes." << std::endl;
        std::cout << "lW_1 = " << lW_1 << std::endl;
    }
    else
    {
        std::cout << "No." << std::endl;
        lTable.add( lW_1 );
        std::cout << "lW_1 = " << lW_1 << std::endl;
    }
    PrefixString lW_2 = lA + 'b';

    std::cout << "Is lW_2 = " << lW_2 << " in LZW table? ";

    if ( lTable.contains( lW_2 ) )
    {
        std::cout << "Yes." << std::endl;
        std::cout << "lW_2 = " << lW_2 << std::endl;
    }
    else
    {
        std::cout << "No." << std::endl;
        lTable.add( lW_1 );
        std::cout << "lW_2 = " << lW_2 << std::endl;
    }

    std::cout << "\nLZWTable test complete." << std::endl;
    std::string lInput = "ababcbababaaaaaaa";
    std::cout << lInput[0];
}

#endif

#ifdef P3

#include "LZWCompressor.h"

void runP3()
{
    std::string lInput = "ababcbababaaaaaaa";
    size_t lCount = 0;

    // Output: 97 98 128 99 129 132 97 134 135 97

    std::cout << "Test LZW Compression:\n" << std::endl;
    std::cout << "Input String:" << lInput << std::endl;
    std::cout << "LZW Codes:" << std::endl;

    for ( const auto& item : LZWCompressor( lInput ) )
    {
        std::cout << item << std::endl;

        lCount++;
    }

    float lUncompressedSize = 8.0f * lInput.size();
    float lCompressedSize = 10.0f * lCount;
    float lOverhead = static_cast<float>((10 * lCount) % 8);
    float lCompressionRatio = lUncompressedSize / lCompressedSize;
    float lSaving = (1.0f - ((lCompressedSize + lOverhead) / lUncompressedSize)) * 100.0f;

    std::cout << "\nCompression Ratio: " << lCompressionRatio << "/1" << std::endl;
    std::cout << "Overhead in Bits: " << lOverhead << std::endl;
```

```
	std::cout << "Space Saving: " << lSaving << "%" << std::endl;
	std::cout << "\nLZW Compression test complete." << std::endl;
}

#endif

int main( int argc, const char* argv[] )
{

#ifdef P1

	runP1();

#endif

#ifdef P2

	runP2();

#endif

#ifdef P3

	runP3();

#endif

#ifndef P1
    #ifndef P2
	#ifndef P3

    std::cout << "No Test enabled."  << std::endl;

	#endif
    #endif
#endif

	return 0;
}
```