CSC 216 Portfolio 1

Nicolas Nytko

December 7, 2016

Contents

1 Homework

1.1 Linked Lists

1.1.1 Problem R-3.7

> *Give an algorithm for finding the penultimate (second to last) node in a singly linked list where the last element is indicated by a null next link.*

2 Projects

2.1 Expression Solver

Name:	Nicolas Nytko	Course: CSC216
Activity:	Expression Solver	
Level:	5	
Description:	Write a program that takes, as input, a fully parenthesized, arithmetic expression and converts it to a binary expression tree. Your program should display the tree in some way and also print the value associated with the root. For an additional challenge, allow for the leaves to store variables of the form x1, x2, x3, and so on, which are initially 0 and which can be updated interactively by your program, with the corresponding update in the printed value of the root of the expression tree. Begin with the "For an additional challenge..." version including at least +, -, *, /, % (call fmod if using double values in C++), parentheses to alter precedence, =, +=, -=, *=, /=, %=. (This is not a complete set of C/C++/Java operations, but to be complete, you'd have to allow data types or break some actual C++ rules. Data types are beyond the scope of this project!) now parse not just one but a sequence of ;-terminated expressions ending ultimately at an EOF marker; the sequence should be able to come from either a file or the keyboard; at the end of the sequence, display the values of all variables assigned to during the expressions.	

2.1.1 Compiler Environment

Listing 1: environment

```
    tex git:(master)    pwd
/Users/nicolas/Git/portfolio2/tex
    tex git:(master)    uname -a
Darwin Nicolass-MacBook-Pro.local 16.1.0 Darwin Kernel Version 16.1.0: Thu Oct 13 21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64 x86_64
    tex git:(master)    clang --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
    tex git:(master)    harper_cpp --version
This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:

Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

2.1.2 Source

Listing 2: ../project/arithmetic-expression/Makefile

```
CC=g++
OUTPUT=arithmetic.out
INPUT=main.cpp exprtree.cpp varstore.cpp exprset.cpp
CCFLAGS=

all:
    $(CC) $(CCFLAGS) -std=c++14 $(INPUT) -o $(OUTPUT)
```

Listing 3: ../project/arithmetic-expression/main.cpp

```
#include <iostream>
#include "exprset.hpp"

int testSet( )
{
    ExpressionSet pExprSet;

    std::cout << "Enter equation(s) separated by semicolons:" << std::endl;
    std::cin >> pExprSet;

    std::cout << std::endl << "Values: " << std::endl;

    for ( size_t i=0; i < pExprSet.size( ); i++ )
    {
        std::cout << pExprSet.getValue( i ) << std::endl;
    }

    VariableStore* pVariables = pExprSet.getVariables( );

    if ( pVariables->size( ) > 0 )
    {
        std::cout << std::endl << "Variables: " << std::endl;

        for ( size_t i=0; i < pVariables->size( ); i++ )
        {
            ExpressionVariable* pVar = pVariables->getVarIndex( i );

            std::cout << pVar->cVariable;

            if ( pVar->nSubscript != 0 )
            {
                std::cout << pVar->nSubscript;
            }

            std::cout << " = " << pVar->dValue << std::endl;
        }
    }

    return 0;
}

int testTree( )
{
    ExpressionTree pTree;
    std::cout << pTree.parenthesize( "5+(5*2)-(2*6/2)" ) << std::endl;

    return 0;
}

int main( )
{
    return testSet( );
}
```

Listing 4: ../project/arithmetic-expression/btree.hpp

```
#ifndef BTREE_HPP
#define BTREE_HPP

template<typename Data>
class BinaryTreeNode
{
private:
    BinaryTreeNode<Data>* pLeft,* pRight;
    Data pData;

public:
    /* big three */

    BinaryTreeNode( ): pLeft( nullptr ), pRight( nullptr ), pData( ) { }

    BinaryTreeNode( const Data& pDataNew ): pLeft( nullptr ), pRight( nullptr ),
        pData( pDataNew ) { }

    BinaryTreeNode( const BinaryTreeNode& pNode ): pLeft( nullptr ), pRight( nullptr
         ), pData( )
    {
        operator=( pNode );
    }

    BinaryTreeNode& operator=( const BinaryTreeNode& pNode )
    {
        if ( pNode.pLeft != nullptr )
        {
            pLeft = new BinaryTreeNode( *pNode.pLeft );
        }

        if ( pNode.pRight != nullptr )
        {
            pRight = new BinaryTreeNode( *pNode.pRight );
        }

        return *this;
    }

    ~BinaryTreeNode( )
    {
        if ( pLeft != nullptr )
        {
            delete pLeft;
        }

        if ( pRight != nullptr )
        {
            delete pRight;
        }
    }

    /* reference pointers so that we can modify the children */

    BinaryTreeNode*& getLeftNode( )
    {
        return pLeft;
    }

    BinaryTreeNode*& getRightNode( )
    {
        return pRight;
    }

    Data& getData( )
    {
        return pData;
    }

    /* const non-reference versions */

    BinaryTreeNode* getLeftNode( ) const
    {
        return pLeft;
    }

    BinaryTreeNode* getRightNode( ) const
    {
        return pRight;
    }

    Data getData( ) const
    {
        return pData;
    }

    /* checks to see if children exist */

    bool hasLeft( ) const
    {
        return ( pLeft ? true : false );
    }

    bool hasRight( ) const
    {
        return ( pRight ? true : false );
    }

    bool hasChildren( ) const
    {
        return ( ( pRight && pLeft ) ? true : false );
    }
};

template<typename Data>
class BinaryTree
{
protected:
    BinaryTreeNode<Data>* pRoot;

public:
    /* big three */

    BinaryTree( ): pRoot( nullptr ) { }

    BinaryTree( const BinaryTree& pTree ): pRoot( nullptr )
    {
        if ( pTree.pRoot != nullptr )
        {
            pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
        }
    }

    BinaryTree& operator=( const BinaryTree& pTree )
    {
        if ( pRoot != nullptr )
        {
            delete pRoot;
            pRoot = nullptr;
        }

        if ( pTree.pRoot != nullptr )
        {
            pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
        }

        return *this;
    }

    ~BinaryTree( )
    {
        if ( pRoot != nullptr )
        {
            delete pRoot;
        }
    }

    BinaryTreeNode<Data>*& getRoot( )
    {
        return pRoot;
    }

    BinaryTreeNode<Data>* getRoot( ) const
    {
        return pRoot;
    }
};

#endif
```

Listing 5: ../project/arithmetic-expression/exprset.hpp

```
#ifndef EXPRESSION_SET_HPP
#define EXPRESSION_SET_HPP

#include <vector>
#include "exprtree.hpp"

class ExpressionSet
{
private:
    std::vector<ExpressionTree> vExpressions;
    std::vector<std::string> vExpressionsInput;
    std::vector<double> vExpressionsValues;
    VariableStore pVariables;

public:
    ExpressionSet( ): vExpressions( ), vExpressionsInput( ), vExpressionsValues( ),
        pVariables( ) { }

    void addExpression( std::string sInput );
    void addMultiExpressions( std::string sInput );

    size_t length( ) const
    {
        return vExpressions.size( );
    }

    size_t size( ) const
    {
        return vExpressions.size( );
    }

    VariableStore* getVariables( ){ return &pVariables; }

    std::string getValue( size_t nIndex ) const;
    double getNumericalValue( size_t nIndex ) const;

    friend std::istream& operator>>( std::istream& pInput, ExpressionSet& pExprSet )
        ;
};

#endif
```

Listing 6: ../project/arithmetic-expression/exprtree.hpp

```
#ifndef EXPR_TREE_HPP
#define EXPR_TREE_HPP

#include <string>
#include <vector>
#include "btree.hpp"

enum ExpressionOperator
{
    OP_POWER,
    OP_MULTIPLICATION,
    OP_DIVISION,
    OP_ADDITION,
    OP_SUBTRACTION,
    OP_MODULUS,
    OP_EQUALS,
};

enum ExpressionNodeType
{
    TYPE_VARIABLE,
    TYPE_NUMBER,
    TYPE_OPERATOR
};

struct ExpressionVariable
{
    char cVariable;
    unsigned short nSubscript;
    double dValue;
};

struct ExpressionNode
{
    ExpressionNodeType nType;

    union
    {
        double dValue;
        ExpressionOperator nOperator;
        ExpressionVariable* pVariable;
    };
};

class VariableStore
{
private:
    std::vector<ExpressionVariable> vVariables;

public:
    VariableStore( ): vVariables( ) { }

    ExpressionVariable* getVariable( char cVariable,
                                     unsigned short nSubscript );
    ExpressionVariable* createVariable( char cVariable,
                                        unsigned short nSubscript );
    ExpressionVariable* findVariable( char cVariable,
                                      unsigned short nSubscript );

    size_t length( ) const
    {
        return vVariables.size( );
    }

    size_t size( ) const
    {
        return vVariables.size( );
    }

    ExpressionVariable* getVarIndex( size_t nIndex )
    {
        return &( vVariables[nIndex] );
    }
};

class ExpressionTree
{
private:
    BinaryTree<ExpressionNode> pTree;
    VariableStore* pVariables;
    bool bCreatedVarStore;

    bool isOperator( char cParse ) const;
    size_t getTopLevelOpLocation( std::string sParse ) const;
    std::string getTopLevelOp( std::string sParse ) const;
    std::string removeParen( std::string sInput ) const;
    ExpressionOperator getOpFromChar( char cOperator ) const;
    int getOpPriority( ExpressionOperator nOperator ) const;

    void parseInputTree( std::string sInput,
                         BinaryTreeNode<ExpressionNode>& bTreeNode );
    double calculateTree( BinaryTreeNode<ExpressionNode>& bTreeNode );

    std::string parenthesizeOp( std::string sInputm, int nOperators ) const;

public:
    std::string parenthesize( std::string sInput ) const;

    ExpressionTree( );
    ExpressionTree( const ExpressionTree& pExprTree );
    ExpressionTree( std::string sInput );
    ExpressionTree& operator=( const ExpressionTree& pExprTree );
    ~ExpressionTree( )
    {
        if ( pVariables && bCreatedVarStore )
        {
            delete pVariables;
        }
    }

    void readline( std::string sInput, bool bParenthesized=false );
    double calculate( );
    void setVariableStore( VariableStore* pVar )
    {
        if ( pVariables && bCreatedVarStore )
        {
            delete pVariables;
            bCreatedVarStore = false;
        }

        pVariables = pVar;
    }
};

#endif
```

Listing 7: ../project/arithmetic-expression/varstore.cpp

```
#include <cstddef>
#include "exprtree.hpp"

ExpressionVariable* VariableStore::getVariable( char cVariable, unsigned short
    nSubscript )
{
    if ( vVariables.size( ) == 0 )
    {
        return nullptr;
    }
    else
    {
        for ( size_t i=0; i < vVariables.size( ); i++ )
        {
            if ( cVariable == vVariables[i].cVariable &&
                 nSubscript == vVariables[i].nSubscript )
            {
                return &( vVariables[i] );
            }
        }

        return nullptr;
    }
}

ExpressionVariable* VariableStore::createVariable( char cVariable, unsigned short
    nSubscript )
{
    ExpressionVariable pVariable;
    pVariable.cVariable = cVariable;
    pVariable.nSubscript = nSubscript;
    pVariable.dValue = 0;

    vVariables.push_back( pVariable );
    return &( vVariables.back( ) );
}

ExpressionVariable* VariableStore::findVariable( char cVariable, unsigned short
    nSubscript )
{
    ExpressionVariable* pVariable = getVariable( cVariable, nSubscript );

    if ( pVariable == nullptr )
    {
        pVariable = createVariable( cVariable, nSubscript );
    }

    return pVariable;
}
```

Listing 8: ../project/arithmetic-expression/exprset.cpp

```
#include <iostream>
#include <sstream>
#include <cstring>
#include "exprset.hpp"

void ExpressionSet::addExpression( std::string sInput )
{
    ExpressionTree pTemp;

    std::string sParen = pTemp.parenthesize( sInput );

    pTemp.setVariableStore( &pVariables );
    pTemp.readline( sParen, true );

    vExpressions.push_back( pTemp );
    vExpressionsInput.push_back( sParen );
    vExpressionsValues.push_back( pTemp.calculate( ) );
}

void ExpressionSet::addMultiExpressions( std::string sInput )
{
    char* szInput = new char[sInput.length( ) + 1];
    strcpy( szInput, sInput.c_str( ) );

    char* szToken = strtok( szInput, ";" );

    while ( szToken != nullptr )
    {
        addExpression( szToken );

        szToken = strtok( nullptr , ";" );
    }

    delete[] szInput;
}

std::string ExpressionSet::getValue( size_t nIndex ) const
{
    if ( nIndex >= length( ) )
    {
        throw std::out_of_range( "Trying to access expression out of bounds." );
    }

    std::stringstream sstream;
    sstream << vExpressionsInput[nIndex] << " = " << vExpressionsValues[nIndex];

    return sstream.str( );
}

double ExpressionSet::getNumericalValue( size_t nIndex ) const
{
    if ( nIndex >= length( ) )
    {
        throw std::out_of_range( "Trying to access expression out of bounds." );
    }

    return vExpressionsValues[nIndex];
}

std::istream& operator>>( std::istream& pInput , ExpressionSet& pExprSet )
{
    std::string sInput;
    getline( pInput , sInput );

    pExprSet.addMultiExpressions( sInput );

    return pInput;
}
```

Listing 9: ../project/arithmetic-expression/exprtree.cpp

```
#include <iostream>
#include <cmath>
#include <cstddef>

#include "exprtree.hpp"

#define CERR_DEBUG_PRINT 0

/**
 * The big three.
 */

ExpressionTree::ExpressionTree( ):
    pTree( ),
    pVariables( nullptr ),
    bCreatedVarStore( false ) { }

ExpressionTree::ExpressionTree( const ExpressionTree& pExprTree ):
    pTree( pExprTree.pTree ),
    pVariables( pExprTree.pVariables ),
    bCreatedVarStore( pExprTree.bCreatedVarStore ) { }

ExpressionTree::ExpressionTree( std::string sInput ):
    pTree( ),
    pVariables( nullptr ),
    bCreatedVarStore( false )
{
    readline( sInput );
}

ExpressionTree& ExpressionTree::operator=( const ExpressionTree& pExprTree )
{
    pTree = pExprTree.pTree;
    pVariables = pExprTree.pVariables;
    bCreatedVarStore = pExprTree.bCreatedVarStore;

    return *this;
}

/**
 * Gets the relative priority of an operator.
 */

int ExpressionTree::getOpPriority( ExpressionOperator nOperator ) const
{
    int nPriority;

    switch ( nOperator )
    {
    case OP_POWER:
        nPriority = 5;
        break;
    case OP_MULTIPLICATION:
        nPriority = 2;
        break;
    case OP_DIVISION:
        nPriority = 2;
        break;
    case OP_ADDITION:
        nPriority = 1;
        break;
    case OP_SUBTRACTION:
        nPriority = 1;
        break;
    case OP_MODULUS:
        nPriority = 2;
        break;
    case OP_EQUALS:
        nPriority = 0;
        break;
    default:
        nPriority = -1;
        break;
    }

    return nPriority;
}

/**
 * Returns the operator enum from a character.
 */

ExpressionOperator ExpressionTree::getOpFromChar( char cOperator ) const
{
    ExpressionOperator nReturn;

    switch ( cOperator )
    {
    default:
    case '+':
        nReturn = OP_ADDITION;
        break;
    case '-':
        nReturn = OP_SUBTRACTION;
        break;
    case '^':
        nReturn = OP_POWER;
        break;
    case '*':
        nReturn = OP_MULTIPLICATION;
        break;
    case '/':
        nReturn = OP_DIVISION;
        break;
    case '%':
        nReturn = OP_MODULUS;
        break;
    case '=':
        nReturn = OP_EQUALS;
        break;
    }

    return nReturn;
}

/**
 * Returns true if the given character is an operator.
 */

bool ExpressionTree::isOperator( char cParse ) const
{
    bool bReturn;

    switch ( cParse )
    {
    case '+':
    case '-':
    case '^':
    case '/':
    case '%':
    case '=':
```

```
    case '*':
        bReturn = true;
        break;
    default:
        bReturn = false;
        break;
    }

    return bReturn;
}

/**
 * Get the top level operator in an expression.  If the expression is fully
     parenthesized,
 * then there will be only one top level operator.
 */

std::string ExpressionTree::getTopLevelOp( std::string sParse ) const
{
    int nParen = 0;
    bool bParsing = true;
    std::string sReturn;

    for ( size_t i=0; i < sParse.length( ) && bParsing; i++ )
    {
        if ( sParse[i] == '(' )
        {
            nParen++;
        }
        if ( sParse[i] == ')' )
        {
            nParen--;
        }
        if ( nParen == 0 && isOperator( sParse[i] ) )
        {
            sReturn += sParse[i];

            /* In case we have multi-character operators such as +=, -=, etc. */

            if ( isOperator( sParse[i+1] ) )
            {
                sReturn += sParse[i+1];
            }

            bParsing = false;
        }
    }

    return sReturn;
}

/**
 * Get the location of the top-level operator.
 */

size_t ExpressionTree::getTopLevelOpLocation( std::string sParse ) const
{
    int nParen = 0;
    bool bParsing = true;
    size_t nLocation = 0;

    for ( size_t i=0; i < sParse.length( ) && bParsing; i++ )
    {
        if ( sParse[i] == '(' )
        {
            nParen++;
        }
        if ( sParse[i] == ')' )
        {
            nParen--;
        }
        if ( nParen == 0 && isOperator( sParse[i] ) )
        {
            nLocation = i;
            bParsing = false;
        }
    }

    return nLocation;
}

/**
 * Removes outer parenthesis in an expression if it contains them.
 */

std::string ExpressionTree::removeParen( std::string sInput ) const
{
    if ( sInput[0] == '(' && sInput[sInput.length( ) - 1 ] == ')' )
    {
        return sInput.substr( 1, sInput.length( ) - 2 );
    }
    else
    {
        return sInput;
    }
}

/**
 * Recursively subdivide a string into an expression tree.
 */

void ExpressionTree::parseInputTree( std::string sInput,
                                     BinaryTreeNode<ExpressionNode>& bTreeNode )
{
    std::string sOperator = getTopLevelOp( sInput );

    /* If there is no operator then are at the end of the branch. */

    if ( sOperator.length( ) != 0 )
    {
        std::string sExprLeft, sExprRight;
        sExprLeft = removeParen( sInput.substr( 0, getTopLevelOpLocation( sInput ) )
            );
        sExprRight = removeParen( sInput.substr( getTopLevelOpLocation( sInput ) +
            sOperator.length( ),
                                                 sInput.length( ) - 1 ) );

        /* Handle special case for +=, -=, etc. */

        if ( sOperator.length( ) == 2 && sOperator[1] == '=' )
        {
            /* If we have x+=5, convert it to x=x+5 */

            sExprRight = sExprLeft + sOperator[0] + "(" + sExprRight + ")";
            sOperator = "=";
        }

#if CERR_DEBUG_PRINT
        std::cerr << "Input expression: " << sInput << std::endl;
        std::cerr << "Operator: " << sOperator << std::endl;
        std::cerr << "Left expression: " << sExprLeft << std::endl;
        std::cerr << "Right expression: " << sExprRight << std::endl;
#endif

        bTreeNode.getData( ).nType = TYPE_OPERATOR;
        bTreeNode.getData( ).nOperator = getOpFromChar( sOperator[0] );

        bTreeNode.getLeftNode( ) = new BinaryTreeNode<ExpressionNode>;
        bTreeNode.getRightNode( ) = new BinaryTreeNode<ExpressionNode>;

        parseInputTree( sExprLeft, *bTreeNode.getLeftNode( ) );
        parseInputTree( sExprRight, *bTreeNode.getRightNode( ) );
    }
    else
    {
        /* If there is no operator then this is either a variable or number. */

        if ( std::isalpha( sInput[0] ) )
        {
            /* If the first character is a letter then it's a variable. */

            char cVar = sInput[0];
            unsigned short nSubscript = 0;

            if ( sInput.length( ) > 1 )
            {
                /* Subscripts are optional. */

                nSubscript = static_cast<unsigned short>
                    ( atoi( sInput.c_str( ) + 1 ) );
            }

            bTreeNode.getData( ).nType = TYPE_VARIABLE;
            bTreeNode.getData( ).pVariable =
                pVariables->findVariable( cVar, nSubscript );
        }
        else
        {
            /* Else, it's a number. */

            bTreeNode.getData( ).nType = TYPE_NUMBER;
            bTreeNode.getData( ).dValue = strtod( sInput.c_str( ), nullptr );
        }

    }
}

```

```
/**
 * Recursively calculate the numerical value of a node and it's children.
 */

double ExpressionTree::calculateTree( BinaryTreeNode<ExpressionNode>& bTreeNode )
{
    if ( bTreeNode.getData( ).nType == TYPE_OPERATOR )
    {
        double dReturn = 0;
        double dLeft = calculateTree( *bTreeNode.getLeftNode( ) );
        double dRight = calculateTree( *bTreeNode.getRightNode( ) );

        switch ( bTreeNode.getData( ).nOperator )
        {
        case OP_POWER:
            dReturn = pow( dLeft, dRight );
            break;
        case OP_MULTIPLICATION:
            dReturn = dLeft * dRight;
            break;
        case OP_DIVISION:
            dReturn = dLeft / dRight;
            break;
        case OP_ADDITION:
            dReturn = dLeft + dRight;
            break;
        case OP_SUBTRACTION:
            dReturn = dLeft - dRight;
            break;
        case OP_MODULUS:
            dReturn = fmod( dLeft, dRight );
            break;
        case OP_EQUALS:
            if ( bTreeNode.getLeftNode( )->getData( ).nType == TYPE_VARIABLE )
            {
                /* Set the variable and return its new value. */

                bTreeNode.getLeftNode( )->getData( ).pVariable->dValue = dRight;
                dReturn = bTreeNode.getLeftNode( )->getData( ).pVariable->dValue;
            }
            else
            {
                /* Return whether or not the two sides are equal. */

                dReturn = ( fabs( dLeft - dRight ) < 0.001 );
            }
            break;
        }

        return dReturn;
    }
    else if ( bTreeNode.getData( ).nType == TYPE_VARIABLE )
    {
        return bTreeNode.getData( ).pVariable->dValue;
    }
    else if ( bTreeNode.getData( ).nType == TYPE_NUMBER )
    {
        return bTreeNode.getData( ).dValue;
    }

    return 0;
}

/**
 * Second stage recursive function in the parenthesizer.
 */

std::string ExpressionTree::parenthesizeOp( std::string sInput, int nOperators )
    const
{
    if ( nOperators == 1 )
    {
        /* If there is only one operator then we don't have to
           parenthesize anything. */

        return sInput;
    }
    else
    {
        /* Find the highest priority operator. */

        int nParen = 0, nPriority = -1;
        size_t nHighestPos = 0;

        for ( size_t i=0; i < sInput.length( ); i++ )
        {
            if ( sInput[i] == '(' )
            {
                nParen++;
            }
            else if ( sInput[i] == ')' )
            {
                nParen--;
            }
            else if ( nParen == 0 && isOperator( sInput[i] ) )
            {
                int nCurPriority = getOpPriority( getOpFromChar( sInput[i] ) );

                if ( nCurPriority > nPriority )
                {
                    nPriority = nCurPriority;
                    nHighestPos = i;
                }
            }
        }

        /* Find where to put the parentheses */

        size_t nLeftParen = nHighestPos-1;
        size_t nRightParen = nHighestPos+1;
        bool bLooping = true;

        nParen = 0;

        /* Go left from the operator for the left parenthesis */

        while ( nLeftParen > 0 && bLooping )
        {
            if ( sInput[nLeftParen] == '(' )
            {
                nParen++;
            }
            else if ( sInput[nLeftParen] == ')' )
            {
                nParen--;
            }
            else if ( nParen == 0 && isOperator( sInput[nLeftParen] ) )
            {
                bLooping = false;
                nLeftParen++;
            }

            if ( bLooping )
            {
                nLeftParen--;
            }
        }

        nParen = 0;
        bLooping = true;

        /* Go right from the operator for the right parenthesis */

        while ( nRightParen < sInput.length( ) && bLooping )
        {
            if ( sInput[nRightParen] == '(' )
            {
                nParen++;
            }
            else if ( sInput[nRightParen] == ')' )
            {
                nParen--;
            }
            else if ( nParen == 0 && isOperator( sInput[nRightParen] ) )
            {
                bLooping = false;
            }

            if ( bLooping )
            {
                nRightParen++;
            }
        }

        sInput.insert( nLeftParen, 1, '(' );
        sInput.insert( nRightParen + 1, 1, ')' );

        return parenthesizeOp( sInput, nOperators - 1 );
    }

    return sInput;
}

/**
 * Converts a given expression to be fully parenthesized.
 */

```

```
std::string ExpressionTree::parenthesize( std::string sInput ) const
{
    size_t nStartParen = 0;
    int nParen = 0, nOper = 0;
    std::vector<std::string> vSubParen;

    /* Check if there are any parentheses and if so split those off
       and recursively parenthesize them. */

    for ( size_t i=0; i < sInput.length( ); i++ )
    {
        if ( sInput[i] == '(' )
        {
            if ( nParen == 0 )
            {
                nStartParen = i;
            }

            nParen++;
        }
        if ( sInput[i] == ')' )
        {
            if ( nParen == 1 )
            {
                std::string sAdd = sInput.substr( nStartParen + 1, i - ( nStartParen + 1) );

                vSubParen.push_back( parenthesize( sAdd ) );
                sInput.erase( nStartParen + 1, sAdd.length( ) + 1 );
                sInput[nStartParen] = '!';

                i -= sAdd.length( ) + 1;
            }

            nParen--;
        }
        if ( nParen == 0 && isOperator( sInput[i] ) )
        {
            nOper++;
        }
    }

    /* Parenthesize order of operations */

    if ( nOper > 1 )
    {
        sInput = parenthesizeOp( sInput, nOper );
    }

    /* Replace parentheses expressions back into our expression */

    if ( vSubParen.size( ) != 0 )
    {
        for ( size_t i=0; i < vSubParen.size( ); i++ )
        {
            size_t nSpot = sInput.find_first_of( "!" );

            sInput.erase( nSpot, 1 );
            sInput.insert( nSpot, vSubParen[i] );
        }
    }

    return "(" + sInput + ")";
}

/**
 * Reads and parses one expression.
 */

void ExpressionTree::readline( std::string sInput, bool bParenthesized )
{
    /* Kill all whitespace */

    std::string sInputNoWs;

    for ( size_t i=0; i < sInput.length( ); i++ )
    {
        if ( !std::isspace( sInput[i] ) )
        {
            sInputNoWs += sInput[i];
        }
    }

    /* Check if the parenthesis are correct */

    int nParen = 0;

    for ( size_t i=0; i < sInputNoWs.length( ); i++ )
    {
        if ( sInputNoWs[i] == '(' )
        {
            nParen++;
        }
        else if ( sInputNoWs[i] == ')' )
        {
            if ( nParen > 0 )
            {
                nParen--;
            }
            else
            {
                throw std::runtime_error( "Ending parenthesis before closing
                    parenthesis." );
            }
        }
    }

    if ( nParen != 0 )
    {
        throw std::runtime_error( "Parenthesis in expression are unbalanced." );
    }

    /* Check if we have a variables store */

    if ( pVariables == nullptr )
    {
        pVariables = new VariableStore;
        bCreatedVarStore = true;
    }

    /* Parenthesize our function if it isn't already. */

    if ( !bParenthesized )
    {
        sInputNoWs = parenthesize( removeParen( sInputNoWs ) );
    }

    pTree.getRoot( ) = new BinaryTreeNode<ExpressionNode>;
    parseInputTree( removeParen( sInputNoWs ), *pTree.getRoot( ) );
}

/**
 * Calculates the numerical value of the expression.
 */

double ExpressionTree::calculate( )
{
    if ( pTree.getRoot( ) == nullptr )
    {
        return 0;
    }
    else
    {
        return calculateTree( *pTree.getRoot( ) );
    }
}
```

2.1.3 Compiler Output

Listing 10: ../project/arithmetic-expression/compilerout

```
     arithmetic-expression git:(master)     make CC=harper_cpp
harper_cpp  -std=c++14 main.cpp exprtree.cpp varstore.cpp exprset.cpp -o arithmetic.
    out
exprset.cpp...
exprtree.cpp...
main.cpp***
varstore.cpp...
```

2.1.4 Program Output

Listing 11: ../project/arithmetic-expression/progout

```
     arithmetic-expression git:(master)     ./arithmetic.out
Enter equation(s) separated by semicolons:
5+5-2

Values:
((5+5)-2) = 8
     arithmetic-expression git:(master)     ./arithmetic.out
Enter equation(s) separated by semicolons:
x=7;7*x

Values:
(x=7) = 7
```

```
(7*x) = 49

Variables:
x = 7
      arithmetic-expression git:(master)      ./arithmetic.out
Enter equation(s) separated by semicolons:
5*5*5*5*5^2

Values:
((((5*5)*5)*5)*(5^2)) = 15625
```

3 Labs

3.1 Text Editor

Name:	Nicolas Nytko	Course: CSC216
Activity:	Text Editor	
Level:	5	
Description:	P-6.3. Write a simple text editor using a list to store all the lines.	

3.1.1 Compiler Environment

Listing 12: environment

```
    tex git:(master)    pwd
/Users/nicolas/Git/portfolio2/tex
    tex git:(master)    uname -a
Darwin Nicolass-MacBook-Pro.local 16.1.0 Darwin Kernel Version 16.1.0: Thu Oct 13 21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64 x86_64
    tex git:(master)    clang --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
    tex git:(master)    harper_cpp --version
This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:

Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.1.2 Source

Listing 13: ../lab/text-editor/main.cpp

```cpp
#include <iostream>
#include <fstream>
#include "fakecurses.hpp"

class StringList
{
private:
    struct StringListNode
    {
        StringListNode( ): cChar( 0 ), pNext( nullptr ), pPrev( nullptr ) { }
        StringListNode( const StringListNode& pOther ):
            cChar( pOther.cChar ), pNext( pOther.pNext ), pPrev( nullptr ) { }
        StringListNode& operator=( const StringListNode& pOther )
        {
            cChar = pOther.cChar;
            pNext = pOther.pNext;
            pPrev = pOther.pPrev;

            return *this;
        }
        ~StringListNode( )
        {
            if ( pNext != nullptr )
            {
                delete pNext;
            }
        }

        char cChar;
        StringListNode* pNext, *pPrev;
    };

    StringListNode* pHead;
    StringListNode* pCursor;
    size_t nCursorPos;

public:
    StringList( ): pHead( nullptr ), pCursor( nullptr ), nCursorPos( 0 )
    {
        pHead = new StringListNode( );
        pCursor = pHead;
    }
    StringList( const StringList& pOther ): pHead( pOther.pHead ), pCursor( nullptr
        ), nCursorPos( pOther.nCursorPos ) { }
    StringList& operator=( const StringList& pOther )
    {
        pHead = pOther.pHead;
        pCursor = pOther.pCursor;
        nCursorPos = pOther.nCursorPos;

        return *this;
    }
    ~StringList( )
    {
        if ( pHead != nullptr )
        {
            delete pHead;
        }
    }

    void clear( )
    {
        if ( pHead != nullptr )
        {
            delete pHead;
        }

        pHead = new StringListNode( );
        pCursor = pHead;
        nCursorPos = 0;
    }

    void left( )
    {
        if ( pCursor->pPrev != nullptr )
        {
            pCursor = pCursor->pPrev;
            nCursorPos--;
        }
    }

    void right( )
    {
        if ( pCursor->pNext != nullptr )
        {
            pCursor = pCursor->pNext;
            nCursorPos++;
        }
    }

    void erase( )
    {
        if ( nCursorPos != 0 )
        {
            StringListNode* pPrev, *pNext;
            pCursor = pCursor->pPrev;
            pPrev = pCursor->pPrev;
            pNext = pCursor->pNext;

            pCursor->pPrev = nullptr;
            pCursor->pNext = nullptr;

            if ( pCursor == pHead )
            {
                pHead = pNext;
            }

            delete pCursor;
            pCursor = pNext;

            if ( pPrev != nullptr )
            {
                pPrev->pNext = pNext;
            }

            if ( pNext != nullptr )
            {
                pNext->pPrev = pPrev;
            }

            nCursorPos--;
        }
    }

    void insert( char c )
    {
        if ( nCursorPos == 0 )
        {
            StringListNode* pPrevHead = pHead;
            pHead = new StringListNode;
            pHead->pNext = pPrevHead;
            pHead->cChar = c;
            pPrevHead->pPrev = pHead;
        }
        else
        {
            StringListNode* pPrev, *pNext, *pInsert;
```

```
                pPrev = pCursor->pPrev;
                pNext = pCursor;

                pInsert = new StringListNode;
                pInsert->cChar = c;
                pInsert->pPrev = pPrev;
                pInsert->pNext = pNext;

                pCursor = pInsert->pNext;

                if ( pPrev != nullptr )
                {
                    pPrev->pNext = pInsert;
                }

                if ( pNext != nullptr )
                {
                    pNext->pPrev = pInsert;
                }
            }

            nCursorPos++;
        }

        size_t getCursorPos( ) const
        {
            return nCursorPos;
        }

        std::string toString( )
        {
            std::string sReturn;
            StringListNode* pNode = pHead;

            while ( pNode->cChar != 0 )
            {
                sReturn += pNode->cChar;
                pNode = pNode->pNext;
            }

            return sReturn;
        }

        friend std::ostream& operator<<( std::ostream& oInput, const StringList& pList )
        {
            StringListNode* pNode = pList.pHead;

            while ( pNode != nullptr )
            {
                if ( pNode->cChar != 0 )
                {
                    oInput << pNode->cChar;
                }

                pNode = pNode->pNext;
            }

            return oInput;
        }
```

```cpp

    friend std::istream& operator>>( std::istream& oInput, StringList& pList )
    {
        pList.clear( );

        while ( !oInput.eof( ) )
        {
            char cTemp = static_cast<char>( oInput.get( ) );

            if ( cTemp >= ' ' && cTemp <= '~' )
            {
                pList.insert( cTemp );
            }
        }

        return oInput;
    }
};

enum ProgramMode
{
    MODE_NORMAL,
    MODE_SAVING,
    MODE_LOADING
};

int main( )
{
    StringList pList, pFileBuffer;
    char cInput = -1;
    ProgramMode nMode = MODE_NORMAL;

    fakecurses::init( );
    fakecurses::clearScreen( );

    while ( cInput != 3 )
    {
        cInput = fakecurses::getKey( );

        if ( cInput != -1 )
        {
            StringList& pCurrentList = ( nMode == MODE_NORMAL ? pList : pFileBuffer );

            if ( cInput >= ' ' && cInput <= '~' )
            {
                /* See if we did a control sequence using the arrow keys */

                if ( cInput == '[' )
                {
                    char cNext = fakecurses::getKey( );

                    if ( cNext == -1 )
                    {
                        pCurrentList.insert( cInput );
                    }
                    else
                    {
                        if ( cNext == 'D' )
                        {
                            pCurrentList.left( );
                        }
                        else if ( cNext == 'C' )
                        {
                            pCurrentList.right( );
                        }
                    }
                }
                else
                {
                    pCurrentList.insert( cInput );
                }
            }
            else if ( cInput == 127 )
            {
                pCurrentList.erase( );
            }
            else if ( cInput == 19 && nMode == MODE_NORMAL) /* Control+S */
            {
                nMode = MODE_SAVING;
            }
            else if ( cInput == 6 && nMode == MODE_NORMAL ) /* Control+F */
            {
                nMode = MODE_LOADING;
            }
            else if ( cInput == 13 && nMode != MODE_NORMAL ) /* Enter button */
            {
                std::string sFilename = pFileBuffer.toString( );
                std::cerr << sFilename << std::endl;

                if ( nMode == MODE_SAVING )
                {
                    std::ofstream sStream( sFilename );
                    sStream << pList;
                }
                else if ( nMode == MODE_LOADING )
                {
                    std::ifstream sStream( sFilename );
                    sStream >> pList;
                }

                nMode = MODE_NORMAL;
                pFileBuffer.clear( );
            }

            fakecurses::clearScreen( );
            fakecurses::setCursor( 1, 1 );
            std::cout << pList;
            fakecurses::setCursor( 1 + static_cast<short>( pList.getCursorPos( ) ),
                1 );

            if ( nMode != MODE_NORMAL )
            {
                fakecurses::setCursor( 1, fakecurses::getScrHeight( ) );

                if ( nMode == MODE_SAVING )
                {
                    std::cout << "Save to: ";
                }
                else
                {
                    std::cout << "Load from: ";
                }

                std::cout << pFileBuffer;
            }
        }
    }

    fakecurses::cleanup( );

    return 0;
}
```

Listing 14: ../lab/text-editor/fakecurses.cpp

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>

#include "fakecurses.hpp"

const static char* ANSI_PREFIX          = "\x1B[";
const static char* ANSI_CLEARSCR        = "2J";
const static char* ANSI_CLEARLINE       = "K";
const static char* ANSI_POSCURSOR       = "H";
const static char* ANSI_MOVEUP          = "A";
const static char* ANSI_MOVEDOWN        = "B";
const static char* ANSI_MOVEFOR         = "C";
const static char* ANSI_MOVEBACK        = "D";
const static char* ANSI_SETMODE         = "m";
const static char* ANSI_SHOWCURSOR      = "?25h";
const static char* ANSI_HIDECURSOR      = "?25l";

const static int ANSI_FG_BASE           = 30;
const static int ANSI_BG_BASE           = 40;

static struct termios originalTermios;

void fakecurses::setCursor( short x, short y )
{
    printf( "%s%i;%i%s", ANSI_PREFIX, y, x, ANSI_POSCURSOR );
}

void fakecurses::moveCursor( short x, short y )
{
    if ( x < 0 )
        printf( "%s%i%s", ANSI_PREFIX, x*-1, ANSI_MOVEBACK );

    if ( x > 0 )
        printf( "%s%i%s", ANSI_PREFIX, x, ANSI_MOVEFOR );

    if ( y < 0 )
        printf( "%s%i%s", ANSI_PREFIX, y*-1, ANSI_MOVEUP );

    if ( y > 0 )
        printf( "%s%i%s", ANSI_PREFIX, y, ANSI_MOVEDOWN );
}

void fakecurses::showCursor( )
{
    printf( "%s%s", ANSI_PREFIX, ANSI_SHOWCURSOR );
}

void fakecurses::hideCursor( )
{
    printf( "%s%s", ANSI_PREFIX, ANSI_HIDECURSOR );
}

void fakecurses::resetColor( )
{
    printf( "%s0%s", ANSI_PREFIX, ANSI_SETMODE );
}

void fakecurses::setColor( ANSI_COLOR fg, ANSI_COLOR bg )
{
    printf( "%s%i;%i%s", ANSI_PREFIX, ANSI_FG_BASE + fg, ANSI_BG_BASE + bg,
        ANSI_SETMODE );
}

void fakecurses::setTextMode( ANSI_TEXT mode )
{
    printf( "%s%i%s", ANSI_PREFIX, mode, ANSI_SETMODE );
}
void fakecurses::clearScreen( )
{
    printf( "%s%s", ANSI_PREFIX, ANSI_CLEARSCR );
    printf( "%s0;0%s", ANSI_PREFIX, ANSI_POSCURSOR );
}

void fakecurses::clearLine( )
{
    printf( "%s%s", ANSI_PREFIX, ANSI_CLEARLINE );
}

void fakecurses::printChar( char c )
{
    printf( "%c", c );
}

void fakecurses::printString( const char* szStr )
{
    printf( "%s", szStr );
}

short fakecurses::getScrWidth( )
{
    struct winsize w;
    ioctl( 1, TIOCGWINSZ, &w );

    return static_cast<short>( w.ws_col );
```

```
}

short fakecurses::getScrHeight( )
{
    struct winsize w;
    ioctl( 1, TIOCGWINSZ, &w );

    return static_cast<short>( w.ws_row );
}

void fakecurses::getScrSize( short& w, short& h )
{
    struct winsize ws;
    ioctl( 1, TIOCGWINSZ, &ws );

    w = static_cast<short>( ws.ws_col );
    h = static_cast<short>( ws.ws_row );
}

void fakecurses::init( )
{
    struct termios newTermios;

    /* Save old terminal information */

    tcgetattr( 0, &originalTermios );
    memcpy( &newTermios, &originalTermios, sizeof( termios ) );

    /* Set the terminal to raw input mode immediately */

    cfmakeraw( &newTermios );
    tcsetattr( 0, TCSANOW, &newTermios );

    setbuf( stdout, NULL );

    clearScreen( );
}

void fakecurses::cleanup( )
{
    tcsetattr( 0, TCSANOW, &originalTermios );

    setColor( COLOR_WHITE, COLOR_BLACK );
    showCursor( );
    resetColor( );
}

bool fakecurses::isKeypress( )
{
    struct timeval sTimeout;
    fd_set fds;

    sTimeout.tv_sec = 0;
    sTimeout.tv_usec = 0;

    FD_ZERO( &fds );
    FD_SET( 0, &fds );

    return ( select( 1, &fds, NULL, NULL, &sTimeout ) != 0 );
```

```cpp
}

char fakecurses::getKey( )
{
    char cKey;
    long nResult;

    nResult = read( 0, &cKey, sizeof( char ) );

    if ( nResult == -1 )
        return -1;

    return cKey;
}
```

Listing 15: ../lab/text-editor/fakecurses.hpp

```cpp
#ifndef LIB_FAKECURSES_HPP
#define LIB_FAKECURSES_HPP

namespace fakecurses
{
    enum ANSI_TEXT
    {
        TEXT_NORMAL = 0,
        TEXT_BOLD    = 1,
        TEXT_UNDERSCORE = 4,
        TEXT_BLINK   = 5,
        TEXT_REVERSE     = 7,
        TEXT_CONCEALED   = 8
    };

    enum ANSI_COLOR
    {
        COLOR_BLACK = 0,
        COLOR_RED,
        COLOR_GREEN,
        COLOR_YELLOW,
        COLOR_BLUE,
        COLOR_MAGENTA,
        COLOR_CYAN,
        COLOR_LGRAY,
        COLOR_DGRAY = 60,
        COLOR_LRED,
        COLOR_LGREEN,
        COLOR_LYELLOW,
        COLOR_LBLUE,
        COLOR_LMAGENTA,
        COLOR_LCYAN,
        COLOR_WHITE
    };

    void setCursor( short x, short y );
    void moveCursor( short x, short y );
    void showCursor( );
    void hideCursor( );

    void resetColor( );
    void setColor( ANSI_COLOR fg, ANSI_COLOR bg );
    void setTextMode( ANSI_TEXT mode );

    void clearScreen( );
    void clearLine( );

    void printChar( char c );
    void printString( const char* szStr );

    short getScrWidth( );
    short getScrHeight( );
    void getScrSize( short& w, short& h );

    void init( );
    void cleanup( );

    bool isKeypress( );
    char getKey( );
}

#endif
```

3.1.3 Compiler Output

Listing 16: ../lab/text-editor/compilerout

```
    text-editor git:(master)     make CC=harper_cpp
harper_cpp  -std=c++14 main.cpp fakecurses.cpp -o editor.out
fakecurses.cpp...
main.cpp***
```

3.2 RPN Calculator

Name:	Nicolas Nytko	Course: CSC216
Activity:	RPN Calculator	
Level:	2	
Description:	P-5.12. Implement a program that can input an expression in postfix notation (see Exercise C-5.8) and output its value.	

3.2.1 Compiler Environment

Listing 17: environment

```
    tex git:(master)     pwd
/Users/nicolas/Git/portfolio2/tex
    tex git:(master)     uname -a
Darwin Nicolass-MacBook-Pro.local 16.1.0 Darwin Kernel Version 16.1.0: Thu Oct 13
    21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64 x86_64
    tex git:(master)     clang --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
    tex git:(master)     harper_cpp --version
This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:

```

```
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
    -dir=/usr/include/c++/4.2.1
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.2.2 Source

Listing 18: ../lab/reverse-polish/main.cpp

```
#include <iostream>
#include <stack>
#include <cmath>
#include <cstring>

double calculateRPN( std::string sInput )
{
    std::stack<double> pOperands;

    /* Check if there are spaces around every operator */

    for ( size_t i=0; i < sInput.length( ); i++ )
    {
        if ( sInput[i] == '+' ||
             sInput[i] == '-' ||
             sInput[i] == '*' ||
             sInput[i] == '/' ||
             sInput[i] == '%' ||
             sInput[i] == '^' )
        {
            if ( i != 0 && sInput[i-1] != ' ' )
            {
                sInput.insert( i, 1, ' ' );
                i++;
            }
            if ( i != sInput.length( ) -1 && sInput[i+1] != ' ' )
            {
                sInput.insert( i+1, 1, ' ' );
                i++;
            }
        }
    }

    /* Make a copy of the string so we can blow it up with strtok( ) */

    char* szInput = new char[sInput.length( ) + 1];
    strcpy( szInput, sInput.c_str( ) );

    char* szToken = strtok( szInput, " " );

    while ( szToken != nullptr )
    {
        if ( ( szToken[0] >= '0' && szToken[0] <= '9' ) || szToken[0] == '.' )
        {
            /* We have a number if the first character is a digit or begins with . (
               i.e. .5) */

            pOperands.push( strtod( szToken, nullptr ) );
        }
        else
        {
            /* Else, we have an operator */

            double dRight, dLeft, dResult;

            dRight = pOperands.top( );
            pOperands.pop( );

            dLeft = pOperands.top( );
            pOperands.pop( );

            switch ( szToken[0] )
            {
            case '+':
                dResult = dLeft + dRight;
                break;
            case '-':
                dResult = dLeft - dRight;
                break;
            case '/':
                dResult = dLeft / dRight;
                break;
            case '*':
                dResult = dLeft * dRight;
                break;
            case '^':
                dResult = pow( dLeft, dRight );
                break;
            case '%':
                dResult = fmod( dLeft, dRight );
                break;
            }

            pOperands.push( dResult );
        }

        szToken = strtok( nullptr, " " );
    }

    delete[] szInput;

    /* The last operand on the stack is the value of the expression */

    return pOperands.top( );
}

int main( )
{
    std::string sExpression;
    std::cout << "Please enter an expression in postfix (reverse polish) notation:"
        << std::endl;
    std::getline( std::cin, sExpression );

    std::cout << std::endl << "The value is: " << calculateRPN( sExpression ) << std
        ::endl;

    return 0;
}
```

3.2.3 Compiler Output

Listing 19: ../lab/reverse-polish/compilerout

```
    reverse-polish git:(master)    make CC=harper_Cpp
harper_Cpp  -std=c++14 main.cpp -o rpn.out
main.cpp***
```

3.2.4 Program Output

Listing 20: ../lab/reverse-polish/progout

```
    reverse-polish git:(master)    ./rpn.out
Please enter an expression in postfix (reverse polish) notation:
5 5 +

The value is: 10
    reverse-polish git:(master)    ./rpn.out
Please enter an expression in postfix (reverse polish) notation:
5 5 + 10 *

The value is: 100
    reverse-polish git:(master)    ./rpn.out
Please enter an expression in postfix (reverse polish) notation:
5 5 - 2 +

The value is: 2
```

3.3 Binary Tree Implementation

Name:	Nicolas Nytko	Course: CSC216
Activity:	Binary Tree	
Level:	2	
Description:	P-7.4. Implement the binary tree ADT using a linked structure.	

Listing 21: environment

```
    tex git:(master)    pwd
/Users/nicolas/Git/portfolio2/tex
    tex git:(master)    uname -a
Darwin Nicolass-MacBook-Pro.local 16.1.0 Darwin Kernel Version 16.1.0: Thu Oct 13
    21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64 x86_64
    tex git:(master)    clang --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
    tex git:(master)    harper_cpp --version
This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:

Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
    -dir=/usr/include/c++/4.2.1
```

```
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.3.1 Source

Listing 22: ../lab/linked-btree/main.cpp

```
#include <iostream>
#include "btree.hpp"

int main( )
{
    BinaryTree<int> pTree;
    pTree.createRoot( 5 );
    pTree.getRoot( )->createLeftNode( 7 );
    pTree.getRoot( )->createRightNode( 3 );

    int nSum = 0;
    auto pSumFunction = [&nSum]( BinaryTreeNode<int>* pNode )
    {
        nSum += pNode->getData( );
    };

    pTree.inorderTraversal( pSumFunction );

    std::cout << "Sum of the tree is: " << nSum << std::endl;

    return 0;
}
```

Listing 23: ../lab/linked-btree/btree.hpp

```
#ifndef BTREE_HPP
#define BTREE_HPP

template<typename Data>
class BinaryTreeNode
{
private:
    BinaryTreeNode<Data>* pLeft,* pRight,* pParent;
    Data pData;

public:
    /* big three */

    BinaryTreeNode( ): pLeft( nullptr ), pRight( nullptr ), pParent( nullptr ),
        pData( ) { }

    BinaryTreeNode( const Data& pDataNew ): pLeft( nullptr ), pRight( nullptr ),
        pParent( nullptr ), pData( pDataNew ) { }

    BinaryTreeNode( const BinaryTreeNode& pNode ): pLeft( nullptr ), pRight( nullptr
        ), pParent( nullptr ), pData( )
    {
        operator=( pNode );
    }

    BinaryTreeNode& operator=( const BinaryTreeNode& pNode )
    {
        if ( pNode.pLeft != nullptr )
        {
            pLeft = new BinaryTreeNode( *pNode.pLeft );
        }

        if ( pNode.pRight != nullptr )
        {
            pRight = new BinaryTreeNode( *pNode.pRight );
        }

        pParent = pNode.pParent;

        return *this;
    }

    ~BinaryTreeNode( )
    {
        if ( pLeft != nullptr )
        {
            delete pLeft;
        }

        if ( pRight != nullptr )
        {
            delete pRight;
        }
    }

    Data& getData( )
    {
        return pData;
    }

    /* const non-reference versions */

    BinaryTreeNode* getLeftNode( ) const
    {
        return pLeft;
    }

    BinaryTreeNode* getRightNode( ) const
    {
        return pRight;
    }

    BinaryTreeNode* getParent( ) const
    {
        return pParent;
    }

    Data getData( ) const
    {
        return pData;
    }

    Data setData( const Data& pDataNew )
    {
        pData = pDataNew;
    }

    /* create children */

    BinaryTreeNode* createLeftNode( )
    {
        if ( pLeft )
        {
            delete pLeft;
            pLeft = nullptr;
        }

        pLeft = new BinaryTreeNode;
        pLeft->pParent = this;

        return pLeft;
    }

    BinaryTreeNode* createLeftNode( const Data& pDataNew )
    {
        if ( pLeft )
        {
            delete pLeft;
            pLeft = nullptr;
        }

        pLeft = new BinaryTreeNode( pDataNew );
        pLeft->pParent = this;

        return pLeft;
    }

    BinaryTreeNode* createRightNode( )
    {
        if ( pRight )
        {
            delete pRight;
            pRight = nullptr;
        }

        pRight = new BinaryTreeNode;
        pRight->pParent = this;

        return pRight;
    }

    BinaryTreeNode* createRightNode( const Data& pDataNew )
    {
        if ( pRight )
        {
            delete pRight;
            pRight = nullptr;
        }

        pRight = new BinaryTreeNode( pDataNew );
        pRight->pParent = this;

```

```cpp
        return pRight;
    }

    /* checks to see if children exist */

    bool hasLeft( ) const
    {
        return ( pLeft ? true : false );
    }

    bool hasRight( ) const
    {
        return ( pRight ? true : false );
    }

    bool hasChildren( ) const
    {
        return ( ( pRight && pLeft ) ? true : false );
    }

    /* traversals */

    void preorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
       pTraverseFunction )
    {
        pTraverseFunction( this );

        if ( pLeft != nullptr )
        {
            pLeft->preorderTraversal( pTraverseFunction );
        }

        if ( pRight != nullptr )
        {
            pRight->preorderTraversal( pTraverseFunction );
        }
    }

    void postorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
       pTraverseFunction )
    {
        if ( pLeft != nullptr )
        {
            pLeft->postorderTraversal( pTraverseFunction );
        }

        if ( pRight != nullptr )
        {
            pRight->postorderTraversal( pTraverseFunction );
        }

        pTraverseFunction( this );
    }

    void inorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
       pTraverseFunction )
    {
        if ( pLeft != nullptr )
        {
            pLeft->inorderTraversal( pTraverseFunction );
        }

        pTraverseFunction( this );

        if ( pRight != nullptr )
        {
            pRight->inorderTraversal( pTraverseFunction );
        }
    }
};

template<typename Data>
class BinaryTree
{
protected:
    BinaryTreeNode<Data>* pRoot;

public:
    /* big three */

    BinaryTree( ): pRoot( nullptr ) { }

    BinaryTree( const BinaryTree& pTree ): pRoot( nullptr )
    {
        if ( pTree.pRoot != nullptr )
        {
            pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
        }
    }

    BinaryTree& operator=( const BinaryTree& pTree )
    {
        if ( pRoot != nullptr )
        {
            delete pRoot;
            pRoot = nullptr;
        }

        if ( pTree.pRoot != nullptr )
        {
            pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
        }

        return *this;
    }

    ~BinaryTree( )
    {
        if ( pRoot != nullptr )
        {
            delete pRoot;
        }
    }

    BinaryTreeNode<Data>*& getRoot( )
    {
        return pRoot;
    }

    BinaryTreeNode<Data>* getRoot( ) const
    {
        return pRoot;
    }

    BinaryTreeNode<Data>* createRoot( )
    {
        if ( pRoot )
        {
            delete pRoot;
            pRoot = nullptr;
        }

        pRoot = new BinaryTreeNode<Data>;

        return pRoot;
    }

    BinaryTreeNode<Data>* createRoot( const Data& pData )
    {
        if ( pRoot )
        {
            delete pRoot;
            pRoot = nullptr;
        }

        pRoot = new BinaryTreeNode<Data>( pData );

        return pRoot;
    }

    void preorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
    {
        if ( pRoot != nullptr )
        {
            pRoot->preorderTraversal( pTraverseFunction );
        }
    }

    void postorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
    {
        if ( pRoot != nullptr )
        {
            pRoot->postorderTraversal( pTraverseFunction );
        }
    }

    void inorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
    {
        if ( pRoot != nullptr )
        {
            pRoot->inorderTraversal( pTraverseFunction );
        }
    }
};
```

```

#endif
```

3.3.2 Compiler Output

Listing 24: ../lab/linked-btree/compilerout

```
    linked-btree git:(master)    make CC=harper_cpp
harper_cpp  -std=c++14 main.cpp -o btree.out
main.cpp***
```

3.3.3 Program Output

Listing 25: ../lab/linked-btree/progout

```
    linked-btree git:(master)    ./btree.out
Sum of the tree is: 15
```

3.4 In-Place Heap Sort

Name:	Nicolas Nytko	Course: CSC216
Activity:	Heap Sort	
Level:	2	
Description:	P-8.4. Implement the in-place heap-sort algorithm. Compare its running time with that of the standard heap-sort that uses an external heap.	

Listing 26: environment

```
    tex git:(master)    pwd
/Users/nicolas/Git/portfolio2/tex
    tex git:(master)    uname -a
Darwin Nicolass-MacBook-Pro.local 16.1.0 Darwin Kernel Version 16.1.0: Thu Oct 13 21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64 x86_64
    tex git:(master)    clang --version
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
    tex git:(master)    harper_cpp --version
This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:

Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 8.0.0 (clang-800.0.42.1)
Target: x86_64-apple-darwin16.1.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.4.1 Source

Listing 27: ../lab/heapsort/main.cpp

```
#include <iostream>
#include <vector>
```

```
#include <cstdlib>
#include <ctime>
#include <typeinfo>

template <typename T>
class VectorHeap
{
private:
    std::vector<T>& pHeap;

public:
    VectorHeap( std::vector<T>& pSetHeap ): pHeap( pSetHeap ) { }

    constexpr size_t getRoot( ) const
    {
        return 0;
    }

    constexpr size_t getLeft( size_t n ) const
    {
        return ( n * 2 ) + 1;
    }

    constexpr size_t getRight( size_t n ) const
    {
        return ( n * 2 ) + 2;
    }

    constexpr size_t getParent( size_t n ) const
    {
        if ( n > 0 )
        {
            size_t nParent = n - 1;

            if ( nParent % 2 == 1 )
            {
                nParent--;
            }

            return nParent / 2;
        }
        else
        {
            return 0;
        }
    }

    T& getRootValue( )
    {
        return pHeap[0];
    }

    T& getLeftValue( size_t n )
    {
        return pHeap[getLeft(n)];
    }

    T& getRightValue( size_t n )
    {
        return pHeap[getRight(n)];
    }

    T& getParentValue( size_t n )
    {
        return pHeap[getParent(n)];
    }

    /* Filter the heap so that the largest value is at the top
       of the heap. */

    void filterDown( size_t nStart, size_t nEnd )
    {
        size_t nRoot = nStart;
        bool bLooping = true;

        while ( getLeft( nRoot ) <= nEnd && bLooping )
        {
            /* While the root has at least one child left */

            size_t nChild, nSwap;

            nChild = getLeft( nRoot );
            nSwap = nRoot;

            if ( pHeap[nSwap] < pHeap[nChild] )
            {
                /* If the left child is greater, then swap with it */

                nSwap = nChild;
            }

            if ( nChild + 1 <= nEnd && pHeap[nSwap] < pHeap[nChild+1] )
            {
                /* If the right child exists, and is greater than what we have
                   currently, then swap with it. */

                nSwap = nChild + 1;
            }

            if ( nSwap == nRoot )
            {
                /* If the root is the largest value, then we're done */

                bLooping = false;
            }
            else
            {
                /* Else, swap the value with the root */

                std::swap( pHeap[nRoot], pHeap[nSwap] );
                nRoot = nSwap;
            }
        }
    }

    void filterDown( )
    {
        filterDown( 0, pHeap.size( ) );
    }

    void filterUp( size_t nStart, size_t nEnd )
    {
        size_t nChild = nEnd;
        bool bLooping = true;

        while ( nChild > nStart && bLooping )
        {
            size_t nParent = getParent( nChild );

            if ( pHeap[nParent] < pHeap[nChild] )
            {
                /* If the heap property is not satisfied */

                std::swap( pHeap[nParent], pHeap[nChild] );
                nChild = nParent;
            }
            else
            {
                /* We're done and the heap is fully sorted */

                bLooping = false;
            }
        }
    }

    void filterUp( )
    {
        filterUp( 0, pHeap.size( ) );
    }

    void heapify( )
    {
        for ( size_t i=1; i < pHeap.size( ); i++ )
        {
            filterUp( 0, i );
        }
    }
};

template<typename T>
void printVector( const std::vector<T>& pVector )
{
    std::cout << "Contents of: " << typeid( T ).name( ) << " vector: " << std::endl;

    for ( size_t i=0; i < pVector.size( ); i++ )
    {
        std::cout << pVector[i] << " ";
    }

    std::cout << std::endl;
}

template<typename T>
void heapSort( std::vector<T>& pSort )
{
    VectorHeap<T> pHeap( pSort );
    pHeap.heapify( );

    for ( size_t i=pSort.size( ) - 1; i > 0; i-- )
    {
        std::swap( pSort[i], pSort[0] );
        pHeap.filterDown( 0, i - 1 );
    }
}

int main( )
{
    std::vector<int> pSortVec;

    srand( static_cast<unsigned int>( time( nullptr ) ) );

    for ( int i=0; i < 20; i++ )
    {
        pSortVec.push_back( rand( ) % 100 );
    }

    printVector( pSortVec );

    std::cout << std::endl << "Sorting vector..." << std::endl << std::endl;

    heapSort( pSortVec );
    printVector( pSortVec );

    return 0;
}
```

3.4.2 Compiler Output

Listing 28: ../lab/heapsort/compilerout

```
      heapsort git:(master)      make CC=harper_cpp
harper_cpp  -std=c++14 main.cpp -o heapsort.out
main.cpp***
```

3.4.3 Program Output

Listing 29: ../lab/heapsort/progout

```
      heapsort git:(master)      ./heapsort.out
Contents of: i vector:
38 77 8 37 43 36 84 89 18 51 42 73 80 74 16 19 8 3 77 12

Sorting vector...

Contents of: i vector:
3 8 8 12 16 18 19 36 37 38 42 43 51 73 74 77 77 80 84 89
      heapsort git:(master)      ./heapsort.out
Contents of: i vector:
45 26 34 48 73 8 28 20 94 13 35 91 25 16 3 75 88 32 70 77

Sorting vector...

Contents of: i vector:
3 8 13 16 20 25 26 28 32 34 35 45 48 70 73 75 77 88 91 94
```

```
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
52 75 60 6 56 80 72 51 70 75 75 56 70 11 90 78 15 61 10 89

Sorting vector...

Contents of: i vector:
6 10 11 15 51 52 56 56 60 61 70 70 72 75 75 75 78 80 89 90
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
59 24 33 64 86 5 16 29 93 37 68 74 15 53 77 34 95 90 50 1

Sorting vector...

Contents of: i vector:
1 5 15 16 24 29 33 34 37 50 53 59 64 68 74 77 86 90 93 95
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
59 24 33 64 86 5 16 29 93 37 68 74 15 53 77 34 95 90 50 1

Sorting vector...

Contents of: i vector:
1 5 15 16 24 29 33 34 37 50 53 59 64 68 74 77 86 90 93 95
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
66 26 59 75 69 77 60 60 69 99 8 39 60 95 64 37 75 19 43 13

Sorting vector...

Contents of: i vector:
8 13 19 26 37 39 43 59 60 60 60 64 66 69 69 75 75 77 95 99
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
66 26 59 75 69 77 60 60 69 99 8 39 60 95 64 37 75 19 43 13

Sorting vector...

Contents of: i vector:
8 13 19 26 37 39 43 59 60 60 60 64 66 69 69 75 75 77 95 99
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
73 75 85 33 99 49 4 91 45 61 48 57 5 37 51 93 2 48 83 25

Sorting vector...

Contents of: i vector:
2 4 5 25 33 37 45 48 48 49 51 57 61 73 75 83 85 91 93 99
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
73 75 85 33 99 49 4 91 45 61 48 57 5 37 51 93 2 48 83 25

Sorting vector...

Contents of: i vector:
2 4 5 25 33 37 45 48 48 49 51 57 61 73 75 83 85 91 93 99
     heapsort git:(master)     ./heapsort.out
Contents of: i vector:
80 24 11 44 82 21 48 22 21 23 41 75 97 79 38 96 82 77 23 37
```

```

Sorting vector...

Contents of: i vector:
11 21 21 22 23 23 24 37 38 41 44 48 75 77 79 80 82 82 96 97
```