CSC 216 Portfolio 1

Nicolas Nytko

December 14, 2016

Contents

1 Homework

1.1 Stacks Queues and Deques

1.1.1 Problem R-5.6 (0.5)

> *Give a recursive function for removing all the elements in a stack.*

Listing 1: ../hw/r-5.6.cpp

```
1  void recursiveRemove( int size, Stack& S )
2  {
3      if ( size > 0 )
4      {
5          recursiveRemove( size-1, S );
6      }
7
8      S.pop( );
9  }
10
11 void popall( Stack& S )
12 {
13     recursiveRemove( S.size( ), S );
14 }
```

1.1.2 Problem C-5.1 (1)

> *Explain how you can implement all the functions of the deque ADT using two stacks. Describe the running time of each operation.*

A deque can be implemented by splitting it into two stacks: one for the front and one for the back. The running time of this deque now depends on the internal implementation of the stack.

Operation	**Array-Based Average**	**Array-Based Worst**	**List-Based Average**	**List-Based Worst**
`insertFront()`	$O(n)$	$O(n)$	$O(1)$	$O(1)$
`insertBack()`	$O(n)$	$O(n)$	$O(1)$	$O(1)$
`eraseFront()`	$O(n)$	$O(n)$	$O(1)$	$O(1)$
`eraseBack()`	$O(n)$	$O(n)$	$O(1)$	$O(1)$
`front()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`back()`	$O(1)$	$O(1)$	$O(n)$	$O(n)$
`size()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$
`empty()`	$O(1)$	$O(1)$	$O(1)$	$O(1)$

1.1.3 Problem C-5.3 (1)

> *Give a pseudo-code description for an array-based implementation of the deque ADT. What is the running time for each operation?*

Listing 2: ../hw/c-5.3.cpp

```
1  template<typename T>
2  class ArrayDeque
3  {
4  private:
5      Array<T> array;
6
```

```
7  public:
8      void insertFront( T )
9      {
10         /* O(n) because it has to shift everything down */
11
12         array.shiftRight( );
13         array[0] = T;
14     }
15
16     void insertBack( T )
17     {
18         /* O(1) because it just places it at the end */
19
20         array[array.size( ) - 1] = T;
21     }
22
23     void eraseFront( )
24     {
25         /* O(n) because it must shift everything to the left */
26
27         array.shiftLeft( );
28     }
29
30     void eraseBack( )
31     {
32         /* O(1) because it just has to decrease the size variable */
33
34         array.remove( array.size( ) - 1 );
35     }
36
37     T& getFront( )
38     {
39         return array[0];
40     }
41
42     T& getBack( )
43     {
44         return array[array.size( ) - 1];
45     }
46 };
```

1.1.4 Problem C-5.7 (1)

> *Describe a nonrecursive algorithm for enumerating all permutations of the numbers* $\{1, 2, \ldots, n\}$.

First, find the number of permutations by calculating $n!$, this will be the number of possible permutations. Fix the first item in the set. For $i = 0$ to $(n-1)!$ loops, repeatedly swap the second item until the end. Each swap is a different permutation. Once all $(n-1)!$ have been found, repeat the process by replacing the first item with the second, third, fourth, etc and keeping it fixed. Repeat until a total of $n!$ permutations are found.

1.1.5 Problem C-5.8 (0.5)

> *Describe a nonrecursive way of evaluating an expression in postfix notation.*

Create an operand stack to hold all of the numbers in the expression. Go through the expression one token at a time. If it is a number, push it to the end of the operand stack. If it is an operator, pop the last two

operands and perform the operation on them, then push it back into the stack. At the end of the expression, there should be one item left in the operand stack, pop that and it will be the value of the expression.

1.2 Trees

1.2.1 Problem R-7.1 (1)

> *Describe an algorithm for counting the number of left external nodes in a binary tree, using the Binary tree ADT.*

Loop through the binary tree and find each external node. For each external node, find its parent node and check if that parent's left node is that external node. If it is the left node, increment a counter by one.

1.2.2 Problem R-7.2 (0.5)

> *The following questions refer to the tree of Figure 7.3:*
>
> 1. *Which node is the root?*
> 2. *What are the internal nodes?*
> 3. *How many descendents does node `cs016/` have?*
> 4. *How many ancestors does node `cs016/` have?*
> 5. *What are the siblings of node homeworks/?*
> 6. *Which nodes are in the subtree rooted at node `projects/`?*
> 7. *What is the depth of node `papers/`?*
> 8. *What is the height of the tree?*

1. `/user/rt/courses/`
2. The internal nodes are: `cs016/`, `homeworks/`, `programs/`, `cs252/`, `projects/`, `papers`, `demos`.
3. 9
4. 1
5. `grades`, `homeworks/`
6. `papers/`, `demos/`
7. 3
8. 5

1.2.3 Problem R-7.3 (0.5)

> *Find the value of the arithmetic expression associated with each subtree of the binary tree of Figure 7.11.*

$$((((3+1)\times 3)/((95)+2))((3\times(74))+6)) = -13$$

1.2.4 Problem R-7.14 (0.5)

> *Let T be a tree with n nodes. What is the running time of the function `parenPrint(T,T.root())`? (See Code Fragment 7.11.)*

The running time is $O(n)$ because it loops through each node exactly once.

1.2.5 Problem C-7.6 (1)

> *Give an O(n)-time algorithm for computing the depth of all the nodes of a tree T, where n is the number of nodes of T.*

Listing 3: ../hw/c-7.6.cpp

```
1  void calculateDepthRec( TreeNode& T, int depth )
2  {
3      /* The current node is depth, do whatever you want with that */
4
5      for ( size_t i=0; i < T.numChildren( ); i++ )
6      {
7          calculateDepthRec( T.getChild(i), depth+1 );
8      }
9  }
10
11 void calculateDepth( Tree& T )
12 {
13     calculateDepthRec( T.root( ), 1 );
14 }
```

1.2.6 Problem C-7.33 (1)

> *Describe, in pseudo-code, a nonrecursive method for performing an in- order traversal of a binary tree in linear time. (Hint: Use a stack.)*

Listing 4: ../hw/c-7.33.cpp

```
1  inorderDFS( Tree T )
2  {
3      Stack S;
4      TreeNode* current = T.getRoot( );
5      bool finished = false;
6
7      while ( !finished )
8      {
9          if ( current != NULL )
10         {
11             S.push( current );
12             current = current->getLeft( );
13         }
14         else
15         {
16             if ( !S.empty( ) )
17             {
18                 current = S.pop( );
19
20                 /* Do something with current's data */
21
22                 current = current.getRight( );
23             }
24             else
25             {
26                 finished = true;
27             }
28         }
29     }
```

```
}
```

1.2.7 Problem C-7.35 (0.5)

> *The path length of a tree T is the sum of the depths of all the nodes in T. Describe a linear-time method for computing the path length of a tree T (which is not necessarily binary).*

Listing 5: ../hw/c-7.35.cpp

```
void pathLengthRec( TreeNode& T, int depth, int& length )
{
    length += depth;

    for ( size_t i=0; i < T.numChildren( ); i++ )
    {
        pathLengthRec( T.getChild(i), depth+1, length );
    }
}

int pathLength( Tree& T )
{
    int length = 0;
    calculateDepthRec( T.root( ), 1, length );

    return length;
}
```

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 6: 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 7: ../project/arithmetic-expression/Makefile

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

all:
```

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

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

```
1  #include <iostream>
2  #include "exprset.hpp"
3
4  int testSet( )
5  {
6      ExpressionSet pExprSet;
7
8      std::cout << "Enter equation(s) separated by semicolons:" << std::endl;
9      std::cin >> pExprSet;
10
11     std::cout << std::endl << "Values: " << std::endl;
12
13     for ( size_t i=0; i < pExprSet.size( ); i++ )
14     {
15         std::cout << pExprSet.getValue( i ) << std::endl;
16     }
17
18     VariableStore* pVariables = pExprSet.getVariables( );
19
20     if ( pVariables->size( ) > 0 )
21     {
22         std::cout << std::endl << "Variables: " << std::endl;
23
24         for ( size_t i=0; i < pVariables->size( ); i++ )
25         {
26             ExpressionVariable* pVar = pVariables->getVarIndex( i );
27
28             std::cout << pVar->cVariable;
29
30             if ( pVar->nSubscript != 0 )
31             {
32                 std::cout << pVar->nSubscript;
33             }
34
35             std::cout << " = " << pVar->dValue << std::endl;
36         }
37     }
38
39     return 0;
40 }
41
42 int testTree( )
43 {
44     ExpressionTree pTree;
45     std::cout << pTree.parenthesize( "5+(5*2)-(2*6/2)" ) << std::endl;
46
47     return 0;
48 }
49
50 int main( )
51 {
52     return testSet( );
53 }
```

Listing 9: ../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;
```

```
56      }
57
58      BinaryTreeNode*& getRightNode( )
59      {
60          return pRight;
61      }
62
63      Data& getData( )
64      {
65          return pData;
66      }
67
68      /* const non-reference versions */
69
70      BinaryTreeNode* getLeftNode( ) const
71      {
72          return pLeft;
73      }
74
75      BinaryTreeNode* getRightNode( ) const
76      {
77          return pRight;
78      }
79
80      Data getData( ) const
81      {
82          return pData;
83      }
84
85      /* checks to see if children exist */
86
87      bool hasLeft( ) const
88      {
89          return ( pLeft ? true : false );
90      }
91
92      bool hasRight( ) const
93      {
94          return ( pRight ? true : false );
95      }
96
97      bool hasChildren( ) const
98      {
99          return ( ( pRight && pLeft ) ? true : false );
100     }
101 };
102
103 template<typename Data>
104 class BinaryTree
105 {
106 protected:
107     BinaryTreeNode<Data>* pRoot;
108
109 public:
110     /* big three */
111
112     BinaryTree( ): pRoot( nullptr ) { }
113
114     BinaryTree( const BinaryTree& pTree ): pRoot( nullptr )
```

```
115     {
116         if ( pTree.pRoot != nullptr )
117         {
118             pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
119         }
120     }
121
122     BinaryTree& operator=( const BinaryTree& pTree )
123     {
124         if ( pRoot != nullptr )
125         {
126             delete pRoot;
127             pRoot = nullptr;
128         }
129
130         if ( pTree.pRoot != nullptr )
131         {
132             pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
133         }
134
135         return *this;
136     }
137
138     ~BinaryTree( )
139     {
140         if ( pRoot != nullptr )
141         {
142             delete pRoot;
143         }
144     }
145
146     BinaryTreeNode<Data>*& getRoot( )
147     {
148         return pRoot;
149     }
150
151     BinaryTreeNode<Data>* getRoot( ) const
152     {
153         return pRoot;
154     }
155 };
156
157 #endif
```

Listing 10: ../project/arithmetic-expression/exprset.hpp

```
1  #ifndef EXPRESSION_SET_HPP
2  #define EXPRESSION_SET_HPP
3
4  #include <vector>
5  #include "exprtree.hpp"
6
7  class ExpressionSet
8  {
9  private:
10     std::vector<ExpressionTree> vExpressions;
11     std::vector<std::string> vExpressionsInput;
12     std::vector<double> vExpressionsValues;
13     VariableStore pVariables;
```

```
14
15  public:
16      ExpressionSet( ): vExpressions( ), vExpressionsInput( ), vExpressionsValues( ),
            pVariables( ) { }
17
18      void addExpression( std::string sInput );
19      void addMultiExpressions( std::string sInput );
20
21      size_t length( ) const
22      {
23          return vExpressions.size( );
24      }
25
26      size_t size( ) const
27      {
28          return vExpressions.size( );
29      }
30
31      VariableStore* getVariables( ){ return &pVariables; }
32
33      std::string getValue( size_t nIndex ) const;
34      double getNumericalValue( size_t nIndex ) const;
35
36      friend std::istream& operator>>( std::istream& pInput, ExpressionSet& pExprSet )
            ;
37  };
38
39  #endif
```

Listing 11: ../project/arithmetic-expression/exprtree.hpp

```
1   #ifndef EXPR_TREE_HPP
2   #define EXPR_TREE_HPP
3
4   #include <string>
5   #include <vector>
6   #include "btree.hpp"
7
8   enum ExpressionOperator
9   {
10      OP_POWER,
11      OP_MULTIPLICATION,
12      OP_DIVISION,
13      OP_ADDITION,
14      OP_SUBTRACTION,
15      OP_MODULUS,
16      OP_EQUALS,
17  };
18
19  enum ExpressionNodeType
20  {
21      TYPE_VARIABLE,
22      TYPE_NUMBER,
23      TYPE_OPERATOR
24  };
25
26  struct ExpressionVariable
27  {
28      char cVariable;
```

```
29      unsigned short nSubscript;
30      double dValue;
31  };
32
33  struct ExpressionNode
34  {
35      ExpressionNodeType nType;
36
37      union
38      {
39          double dValue;
40          ExpressionOperator nOperator;
41          ExpressionVariable* pVariable;
42      };
43  };
44
45  class VariableStore
46  {
47  private:
48      std::vector<ExpressionVariable> vVariables;
49
50  public:
51      VariableStore( ): vVariables( ) { }
52
53      ExpressionVariable* getVariable( char cVariable,
54                                       unsigned short nSubscript );
55      ExpressionVariable* createVariable( char cVariable,
56                                          unsigned short nSubscript );
57      ExpressionVariable* findVariable( char cVariable,
58                                        unsigned short nSubscript );
59
60      size_t length( ) const
61      {
62          return vVariables.size( );
63      }
64
65      size_t size( ) const
66      {
67          return vVariables.size( );
68      }
69
70      ExpressionVariable* getVarIndex( size_t nIndex )
71      {
72          return &( vVariables[nIndex] );
73      }
74  };
75
76  class ExpressionTree
77  {
78  private:
79      BinaryTree<ExpressionNode> pTree;
80      VariableStore* pVariables;
81      bool bCreatedVarStore;
82
83      bool isOperator( char cParse ) const;
84      size_t getTopLevelOpLocation( std::string sParse ) const;
85      std::string getTopLevelOp( std::string sParse ) const;
86      std::string removeParen( std::string sInput ) const;
87      ExpressionOperator getOpFromChar( char cOperator ) const;
```

```
88      int getOpPriority( ExpressionOperator nOperator ) const;
89
90      void parseInputTree( std::string sInput,
91                           BinaryTreeNode<ExpressionNode>& bTreeNode );
92      double calculateTree( BinaryTreeNode<ExpressionNode>& bTreeNode );
93
94      std::string parenthesizeOp( std::string sInputm, int nOperators ) const;
95
96  public:
97      std::string parenthesize( std::string sInput ) const;
98
99      ExpressionTree( );
100     ExpressionTree( const ExpressionTree& pExprTree );
101     ExpressionTree( std::string sInput );
102     ExpressionTree& operator=( const ExpressionTree& pExprTree );
103     ~ExpressionTree( )
104     {
105         if ( pVariables && bCreatedVarStore )
106         {
107             delete pVariables;
108         }
109     }
110
111     void readline( std::string sInput, bool bParenthesized=false );
112     double calculate( );
113     void setVariableStore( VariableStore* pVar )
114     {
115         if ( pVariables && bCreatedVarStore )
116         {
117             delete pVariables;
118             bCreatedVarStore = false;
119         }
120
121         pVariables = pVar;
122     }
123 };
124
125 #endif
```

Listing 12: ../project/arithmetic-expression/varstore.cpp

```
1   #include <cstddef>
2   #include "exprtree.hpp"
3
4   ExpressionVariable* VariableStore::getVariable( char cVariable, unsigned short
        nSubscript )
5   {
6       if ( vVariables.size( ) == 0 )
7       {
8           return nullptr;
9       }
10      else
11      {
12          for ( size_t i=0; i < vVariables.size( ); i++ )
13          {
14              if ( cVariable == vVariables[i].cVariable &&
15                   nSubscript == vVariables[i].nSubscript )
16              {
17                  return &( vVariables[i] );
```

```
18              }
19          }
20
21          return nullptr;
22      }
23  }
24
25  ExpressionVariable* VariableStore::createVariable( char cVariable, unsigned short
        nSubscript )
26  {
27      ExpressionVariable pVariable;
28      pVariable.cVariable = cVariable;
29      pVariable.nSubscript = nSubscript;
30      pVariable.dValue = 0;
31
32      vVariables.push_back( pVariable );
33      return &( vVariables.back( ) );
34  }
35
36  ExpressionVariable* VariableStore::findVariable( char cVariable, unsigned short
        nSubscript )
37  {
38      ExpressionVariable* pVariable = getVariable( cVariable, nSubscript );
39
40      if ( pVariable == nullptr )
41      {
42          pVariable = createVariable( cVariable, nSubscript );
43      }
44
45      return pVariable;
46  }
```

Listing 13: ../project/arithmetic-expression/exprset.cpp

```
1   #include <iostream>
2   #include <sstream>
3   #include <cstring>
4   #include "exprset.hpp"
5
6   void ExpressionSet::addExpression( std::string sInput )
7   {
8       ExpressionTree pTemp;
9
10      std::string sParen = pTemp.parenthesize( sInput );
11
12      pTemp.setVariableStore( &pVariables );
13      pTemp.readline( sParen, true );
14
15      vExpressions.push_back( pTemp );
16      vExpressionsInput.push_back( sParen );
17      vExpressionsValues.push_back( pTemp.calculate( ) );
18  }
19
20  void ExpressionSet::addMultiExpressions( std::string sInput )
21  {
22      char* szInput = new char[sInput.length( ) + 1];
23      strcpy( szInput, sInput.c_str( ) );
24
25      char* szToken = strtok( szInput, ";" );
```

```
26
27      while ( szToken != nullptr )
28      {
29          addExpression( szToken );
30
31          szToken = strtok( nullptr, ";" );
32      }
33
34      delete[] szInput;
35  }
36
37  std::string ExpressionSet::getValue( size_t nIndex ) const
38  {
39      if ( nIndex >= length( ) )
40      {
41          throw std::out_of_range( "Trying to access expression out of bounds." );
42      }
43
44      std::stringstream sstream;
45      sstream << vExpressionsInput[nIndex] << " = " << vExpressionsValues[nIndex];
46
47      return sstream.str( );
48  }
49
50  double ExpressionSet::getNumericalValue( size_t nIndex ) const
51  {
52      if ( nIndex >= length( ) )
53      {
54          throw std::out_of_range( "Trying to access expression out of bounds." );
55      }
56
57      return vExpressionsValues[nIndex];
58  }
59
60  std::istream& operator>>( std::istream& pInput, ExpressionSet& pExprSet )
61  {
62      std::string sInput;
63      getline( pInput, sInput );
64
65      pExprSet.addMultiExpressions( sInput );
66
67      return pInput;
68  }
```

Listing 14: ../project/arithmetic-expression/exprtree.cpp

```
1   #include <iostream>
2   #include <cmath>
3   #include <cstddef>
4
5   #include "exprtree.hpp"
6
7   #define CERR_DEBUG_PRINT 0
8
9   /**
10   * The big three.
11   */
12
13  ExpressionTree::ExpressionTree( ):
```

```
14      pTree( ),
15      pVariables( nullptr ),
16      bCreatedVarStore( false ) { }
17
18  ExpressionTree::ExpressionTree( const ExpressionTree& pExprTree ):
19      pTree( pExprTree.pTree ),
20      pVariables( pExprTree.pVariables ),
21      bCreatedVarStore( pExprTree.bCreatedVarStore ) { }
22
23  ExpressionTree::ExpressionTree( std::string sInput ):
24      pTree( ),
25      pVariables( nullptr ),
26      bCreatedVarStore( false )
27  {
28      readline( sInput );
29  }
30
31  ExpressionTree& ExpressionTree::operator=( const ExpressionTree& pExprTree )
32  {
33      pTree = pExprTree.pTree;
34      pVariables = pExprTree.pVariables;
35      bCreatedVarStore = pExprTree.bCreatedVarStore;
36
37      return *this;
38  }
39
40  /**
41   * Gets the relative priority of an operator.
42   */
43
44  int ExpressionTree::getOpPriority( ExpressionOperator nOperator ) const
45  {
46      int nPriority;
47
48      switch ( nOperator )
49      {
50      case OP_POWER:
51          nPriority = 5;
52          break;
53      case OP_MULTIPLICATION:
54          nPriority = 2;
55          break;
56      case OP_DIVISION:
57          nPriority = 2;
58          break;
59      case OP_ADDITION:
60          nPriority = 1;
61          break;
62      case OP_SUBTRACTION:
63          nPriority = 1;
64          break;
65      case OP_MODULUS:
66          nPriority = 2;
67          break;
68      case OP_EQUALS:
69          nPriority = 0;
70          break;
71      default:
72          nPriority = -1;
```

```
73          break;
74      }
75
76      return nPriority;
77  }
78
79  /**
80   * Returns the operator enum from a character.
81   */
82
83  ExpressionOperator ExpressionTree::getOpFromChar( char cOperator ) const
84  {
85      ExpressionOperator nReturn;
86
87      switch ( cOperator )
88      {
89      default:
90      case '+':
91          nReturn = OP_ADDITION;
92          break;
93      case '-':
94          nReturn = OP_SUBTRACTION;
95          break;
96      case '^':
97          nReturn = OP_POWER;
98          break;
99      case '*':
100         nReturn = OP_MULTIPLICATION;
101         break;
102     case '/':
103         nReturn = OP_DIVISION;
104         break;
105     case '%':
106         nReturn = OP_MODULUS;
107         break;
108     case '=':
109         nReturn = OP_EQUALS;
110         break;
111     }
112
113     return nReturn;
114 }
115
116 /**
117  * Returns true if the given character is an operator.
118  */
119
120 bool ExpressionTree::isOperator( char cParse ) const
121 {
122     bool bReturn;
123
124     switch ( cParse )
125     {
126     case '+':
127     case '-':
128     case '^':
129     case '/':
130     case '%':
131     case '=':
```

```
132     case '*':
133         bReturn = true;
134         break;
135     default:
136         bReturn = false;
137         break;
138     }
139 
140     return bReturn;
141 }
142 
143 /**
144  * Get the top level operator in an expression.  If the expression is fully
        parenthesized,
145  * then there will be only one top level operator.
146  */
147 
148 std::string ExpressionTree::getTopLevelOp( std::string sParse ) const
149 {
150     int nParen = 0;
151     bool bParsing = true;
152     std::string sReturn;
153 
154     for ( size_t i=0; i < sParse.length( ) && bParsing; i++ )
155     {
156         if ( sParse[i] == '(' )
157         {
158             nParen++;
159         }
160         if ( sParse[i] == ')' )
161         {
162             nParen--;
163         }
164         if ( nParen == 0 && isOperator( sParse[i] ) )
165         {
166             sReturn += sParse[i];
167 
168             /* In case we have multi-character operators such as +=, -=, etc. */
169 
170             if ( isOperator( sParse[i+1] ) )
171             {
172                 sReturn += sParse[i+1];
173             }
174 
175             bParsing = false;
176         }
177     }
178 
179     return sReturn;
180 }
181 
182 /**
183  * Get the location of the top-level operator.
184  */
185 
186 size_t ExpressionTree::getTopLevelOpLocation( std::string sParse ) const
187 {
188     int nParen = 0;
189     bool bParsing = true;
```

```
190     size_t nLocation = 0;
191
192     for ( size_t i=0; i < sParse.length( ) && bParsing; i++ )
193     {
194         if ( sParse[i] == '(' )
195         {
196             nParen++;
197         }
198         if ( sParse[i] == ')' )
199         {
200             nParen--;
201         }
202         if ( nParen == 0 && isOperator( sParse[i] ) )
203         {
204             nLocation = i;
205             bParsing = false;
206         }
207     }
208
209     return nLocation;
210 }
211
212 /**
213  * Removes outer parenthesis in an expression if it contains them.
214  */
215
216 std::string ExpressionTree::removeParen( std::string sInput ) const
217 {
218     if ( sInput[0] == '(' && sInput[sInput.length( ) - 1 ] == ')' )
219     {
220         return sInput.substr( 1, sInput.length( ) - 2 );
221     }
222     else
223     {
224         return sInput;
225     }
226 }
227
228 /**
229  * Recursively subdivide a string into an expression tree.
230  */
231
232 void ExpressionTree::parseInputTree( std::string sInput,
233                                      BinaryTreeNode<ExpressionNode>& bTreeNode )
234 {
235     std::string sOperator = getTopLevelOp( sInput );
236
237     /* If there is no operator then are at the end of the branch. */
238
239     if ( sOperator.length( ) != 0 )
240     {
241         std::string sExprLeft, sExprRight;
242         sExprLeft = removeParen( sInput.substr( 0, getTopLevelOpLocation( sInput ) )
                ;
243         sExprRight = removeParen( sInput.substr( getTopLevelOpLocation( sInput ) +
                sOperator.length( ),
244                                                  sInput.length( ) - 1 ) );
245
246         /* Handle special case for +=, -=, etc. */
```

```
247
248         if ( sOperator.length( ) == 2 && sOperator[1] == '=' )
249         {
250             /* If we have x+=5, convert it to x=x+5 */
251
252             sExprRight = sExprLeft + sOperator[0] + "(" + sExprRight + ")";
253             sOperator = "=";
254         }
255
256 #if CERR_DEBUG_PRINT
257         std::cerr << "Input expression: " << sInput << std::endl;
258         std::cerr << "Operator: " << sOperator << std::endl;
259         std::cerr << "Left expression: " << sExprLeft << std::endl;
260         std::cerr << "Right expression: " << sExprRight << std::endl;
261 #endif
262
263         bTreeNode.getData( ).nType = TYPE_OPERATOR;
264         bTreeNode.getData( ).nOperator = getOpFromChar( sOperator[0] );
265
266         bTreeNode.getLeftNode( ) = new BinaryTreeNode<ExpressionNode>;
267         bTreeNode.getRightNode( ) = new BinaryTreeNode<ExpressionNode>;
268
269         parseInputTree( sExprLeft, *bTreeNode.getLeftNode( ) );
270         parseInputTree( sExprRight, *bTreeNode.getRightNode( ) );
271     }
272     else
273     {
274         /* If there is no operator then this is either a variable or number. */
275
276         if ( std::isalpha( sInput[0] ) )
277         {
278             /* If the first character is a letter then it's a variable. */
279
280             char cVar = sInput[0];
281             unsigned short nSubscript = 0;
282
283             if ( sInput.length( ) > 1 )
284             {
285                 /* Subscripts are optional. */
286
287                 nSubscript = static_cast<unsigned short>
288                     ( atoi( sInput.c_str( ) + 1 ) );
289             }
290
291             bTreeNode.getData( ).nType = TYPE_VARIABLE;
292             bTreeNode.getData( ).pVariable =
293                 pVariables->findVariable( cVar, nSubscript );
294         }
295         else
296         {
297             /* Else, it's a number. */
298
299             bTreeNode.getData( ).nType = TYPE_NUMBER;
300             bTreeNode.getData( ).dValue = strtod( sInput.c_str( ), nullptr );
301         }
302
303     }
304 }
305
```

```
306 /**
307  * Recursively calculate the numerical value of a node and it's children.
308  */
309
310 double ExpressionTree::calculateTree( BinaryTreeNode<ExpressionNode>& bTreeNode )
311 {
312     if ( bTreeNode.getData( ).nType == TYPE_OPERATOR )
313     {
314         double dReturn = 0;
315         double dLeft = calculateTree( *bTreeNode.getLeftNode( ) );
316         double dRight = calculateTree( *bTreeNode.getRightNode( ) );
317
318         switch ( bTreeNode.getData( ).nOperator )
319         {
320         case OP_POWER:
321             dReturn = pow( dLeft, dRight );
322             break;
323         case OP_MULTIPLICATION:
324             dReturn = dLeft * dRight;
325             break;
326         case OP_DIVISION:
327             dReturn = dLeft / dRight;
328             break;
329         case OP_ADDITION:
330             dReturn = dLeft + dRight;
331             break;
332         case OP_SUBTRACTION:
333             dReturn = dLeft - dRight;
334             break;
335         case OP_MODULUS:
336             dReturn = fmod( dLeft, dRight );
337             break;
338         case OP_EQUALS:
339             if ( bTreeNode.getLeftNode( )->getData( ).nType == TYPE_VARIABLE )
340             {
341                 /* Set the variable and return its new value. */
342
343                 bTreeNode.getLeftNode( )->getData( ).pVariable->dValue = dRight;
344                 dReturn = bTreeNode.getLeftNode( )->getData( ).pVariable->dValue;
345             }
346             else
347             {
348                 /* Return whether or not the two sides are equal. */
349
350                 dReturn = ( fabs( dLeft - dRight ) < 0.001 );
351             }
352             break;
353         }
354
355         return dReturn;
356     }
357     else if ( bTreeNode.getData( ).nType == TYPE_VARIABLE )
358     {
359         return bTreeNode.getData( ).pVariable->dValue;
360     }
361     else if ( bTreeNode.getData( ).nType == TYPE_NUMBER )
362     {
363         return bTreeNode.getData( ).dValue;
364     }
```

```
365
366     return 0;
367 }
368
369 /**
370  * Second stage recursive function in the parenthesizer.
371  */
372
373 std::string ExpressionTree::parenthesizeOp( std::string sInput, int nOperators )
        const
374 {
375     if ( nOperators == 1 )
376     {
377         /* If there is only one operator then we don't have to
378            parenthesize anything. */
379
380         return sInput;
381     }
382     else
383     {
384         /* Find the highest priority operator. */
385
386         int nParen = 0, nPriority = -1;
387         size_t nHighestPos = 0;
388
389         for ( size_t i=0; i < sInput.length( ); i++ )
390         {
391             if ( sInput[i] == '(' )
392             {
393                 nParen++;
394             }
395             else if ( sInput[i] == ')' )
396             {
397                 nParen--;
398             }
399             else if ( nParen == 0 && isOperator( sInput[i] ) )
400             {
401                 int nCurPriority = getOpPriority( getOpFromChar( sInput[i] ) );
402
403                 if ( nCurPriority > nPriority )
404                 {
405                     nPriority = nCurPriority;
406                     nHighestPos = i;
407                 }
408             }
409         }
410
411         /* Find where to put the parentheses */
412
413         size_t nLeftParen = nHighestPos-1;
414         size_t nRightParen = nHighestPos+1;
415         bool bLooping = true;
416
417         nParen = 0;
418
419         /* Go left from the operator for the left parenthesis */
420
421         while ( nLeftParen > 0 && bLooping )
422         {
```

```
423             if ( sInput[nLeftParen] == '(' )
424             {
425                 nParen++;
426             }
427             else if ( sInput[nLeftParen] == ')' )
428             {
429                 nParen--;
430             }
431             else if ( nParen == 0 && isOperator( sInput[nLeftParen] ) )
432             {
433                 bLooping = false;
434                 nLeftParen++;
435             }
436 
437             if ( bLooping )
438             {
439                 nLeftParen--;
440             }
441         }
442 
443         nParen = 0;
444         bLooping = true;
445 
446         /* Go right from the operator for the right parenthesis */
447 
448         while ( nRightParen < sInput.length( ) && bLooping )
449         {
450             if ( sInput[nRightParen] == '(' )
451             {
452                 nParen++;
453             }
454             else if ( sInput[nRightParen] == ')' )
455             {
456                 nParen--;
457             }
458             else if ( nParen == 0 && isOperator( sInput[nRightParen] ) )
459             {
460                 bLooping = false;
461             }
462 
463             if ( bLooping )
464             {
465                 nRightParen++;
466             }
467         }
468 
469         sInput.insert( nLeftParen, 1, '(' );
470         sInput.insert( nRightParen + 1, 1, ')' );
471 
472         return parenthesizeOp( sInput, nOperators - 1 );
473     }
474 
475     return sInput;
476 }
477 
478 /**
479  * Converts a given expression to be fully parenthesized.
480  */
481 
```

```
482 std::string ExpressionTree::parenthesize( std::string sInput ) const
483 {
484     size_t nStartParen = 0;
485     int nParen = 0, nOper = 0;
486     std::vector<std::string> vSubParen;
487 
488     /* Check if there are any parentheses and if so split those off
489        and recursively parenthesize them. */
490 
491     for ( size_t i=0; i < sInput.length( ); i++ )
492     {
493         if ( sInput[i] == '(' )
494         {
495             if ( nParen == 0 )
496             {
497                 nStartParen = i;
498             }
499 
500             nParen++;
501         }
502         if ( sInput[i] == ')' )
503         {
504             if ( nParen == 1 )
505             {
506                 std::string sAdd = sInput.substr( nStartParen + 1, i - ( nStartParen
                        + 1) );
507 
508                 vSubParen.push_back( parenthesize( sAdd ) );
509                 sInput.erase( nStartParen + 1, sAdd.length( ) + 1 );
510                 sInput[nStartParen] = '!';
511 
512                 i -= sAdd.length( ) + 1;
513             }
514 
515             nParen--;
516         }
517         if ( nParen == 0 && isOperator( sInput[i] ) )
518         {
519             nOper++;
520         }
521     }
522 
523     /* Parenthesize order of operations */
524 
525     if ( nOper > 1 )
526     {
527         sInput = parenthesizeOp( sInput, nOper );
528     }
529 
530     /* Replace parentheses expressions back into our expression */
531 
532     if ( vSubParen.size( ) != 0 )
533     {
534         for ( size_t i=0; i < vSubParen.size( ); i++ )
535         {
536             size_t nSpot = sInput.find_first_of( "!" );
537 
538             sInput.erase( nSpot, 1 );
539             sInput.insert( nSpot, vSubParen[i] );
```

```
540         }
541     }
542
543     return "(" + sInput + ")";
544 }
545
546 /**
547  * Reads and parses one expression.
548  */
549
550 void ExpressionTree::readline( std::string sInput, bool bParenthesized )
551 {
552     /* Kill all whitespace */
553
554     std::string sInputNoWs;
555
556     for ( size_t i=0; i < sInput.length( ); i++ )
557     {
558         if ( !std::isspace( sInput[i] ) )
559         {
560             sInputNoWs += sInput[i];
561         }
562     }
563
564     /* Check if the parenthesis are correct */
565
566     int nParen = 0;
567
568     for ( size_t i=0; i < sInputNoWs.length( ); i++ )
569     {
570         if ( sInputNoWs[i] == '(' )
571         {
572             nParen++;
573         }
574         else if ( sInputNoWs[i] == ')' )
575         {
576             if ( nParen > 0 )
577             {
578                 nParen--;
579             }
580             else
581             {
582                 throw std::runtime_error( "Ending parenthesis before closing
                        parenthesis." );
583             }
584         }
585     }
586
587     if ( nParen != 0 )
588     {
589         throw std::runtime_error( "Parenthesis in expression are unbalanced." );
590     }
591
592     /* Check if we have a variables store */
593
594     if ( pVariables == nullptr )
595     {
596         pVariables = new VariableStore;
597         bCreatedVarStore = true;
```

```
598     }
599
600     /* Parenthesize our function if it isn't already. */
601
602     if ( !bParenthesized )
603     {
604         sInputNoWs = parenthesize( removeParen( sInputNoWs ) );
605     }
606
607     pTree.getRoot( ) = new BinaryTreeNode<ExpressionNode>;
608     parseInputTree( removeParen( sInputNoWs ), *pTree.getRoot( ) );
609 }
610
611 /**
612  * Calculates the numerical value of the expression.
613  */
614
615 double ExpressionTree::calculate( )
616 {
617     if ( pTree.getRoot( ) == nullptr )
618     {
619         return 0;
620     }
621     else
622     {
623         return calculateTree( *pTree.getRoot( ) );
624     }
625 }
```

2.1.3 Compiler Output

Listing 15: ../project/arithmetic-expression/compilerout

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

2.1.4 Program Output

Listing 16: ../project/arithmetic-expression/progout

```
1  arithmetic-expression git:(master)  ./arithmetic.out
2  Enter equation(s) separated by semicolons:
3  5+5-2
4
5  Values:
6  ((5+5)-2) = 8
7  arithmetic-expression git:(master)  ./arithmetic.out
8  Enter equation(s) separated by semicolons:
9  x=7;7*x
10
11 Values:
12 (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 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
```

3.1.2 Source

Listing 18: ../lab/text-editor/main.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;
```

```
20          }
21          ~StringListNode( )
22          {
23              if ( pNext != nullptr )
24              {
25                  delete pNext;
26              }
27          }
28
29          char cChar;
30          StringListNode* pNext, *pPrev;
31      };
32
33      StringListNode* pHead;
34      StringListNode* pCursor;
35      size_t nCursorPos;
36
37  public:
38      StringList( ): pHead( nullptr ), pCursor( nullptr ), nCursorPos( 0 )
39      {
40          pHead = new StringListNode( );
41          pCursor = pHead;
42      }
43      StringList( const StringList& pOther ): pHead( pOther.pHead ), pCursor( nullptr
            ), nCursorPos( pOther.nCursorPos ) { }
44      StringList& operator=( const StringList& pOther )
45      {
46          pHead = pOther.pHead;
47          pCursor = pOther.pCursor;
48          nCursorPos = pOther.nCursorPos;
49
50          return *this;
51      }
52      ~StringList( )
53      {
54          if ( pHead != nullptr )
55          {
56              delete pHead;
57          }
58      }
59
60      void clear( )
61      {
62          if ( pHead != nullptr )
63          {
64              delete pHead;
65          }
66
67          pHead = new StringListNode( );
68          pCursor = pHead;
69          nCursorPos = 0;
70      }
71
72      void left( )
73      {
74          if ( pCursor->pPrev != nullptr )
75          {
76              pCursor = pCursor->pPrev;
77              nCursorPos--;
```

```
78          }
79      }
80
81      void right( )
82      {
83          if ( pCursor->pNext != nullptr )
84          {
85              pCursor = pCursor->pNext;
86              nCursorPos++;
87          }
88      }
89
90      void erase( )
91      {
92          if ( nCursorPos != 0 )
93          {
94              StringListNode* pPrev, *pNext;
95              pCursor = pCursor->pPrev;
96              pPrev = pCursor->pPrev;
97              pNext = pCursor->pNext;
98
99              pCursor->pPrev = nullptr;
100             pCursor->pNext = nullptr;
101
102             if ( pCursor == pHead )
103             {
104                 pHead = pNext;
105             }
106
107             delete pCursor;
108             pCursor = pNext;
109
110             if ( pPrev != nullptr )
111             {
112                 pPrev->pNext = pNext;
113             }
114
115             if ( pNext != nullptr )
116             {
117                 pNext->pPrev = pPrev;
118             }
119
120             nCursorPos--;
121         }
122     }
123
124     void insert( char c )
125     {
126         if ( nCursorPos == 0 )
127         {
128             StringListNode* pPrevHead = pHead;
129             pHead = new StringListNode;
130             pHead->pNext = pPrevHead;
131             pHead->cChar = c;
132             pPrevHead->pPrev = pHead;
133         }
134         else
135         {
136             StringListNode* pPrev, *pNext, *pInsert;
```

```
137             pPrev = pCursor->pPrev;
138             pNext = pCursor;
139 
140             pInsert = new StringListNode;
141             pInsert->cChar = c;
142             pInsert->pPrev = pPrev;
143             pInsert->pNext = pNext;
144 
145             pCursor = pInsert->pNext;
146 
147             if ( pPrev != nullptr )
148             {
149                 pPrev->pNext = pInsert;
150             }
151 
152             if ( pNext != nullptr )
153             {
154                 pNext->pPrev = pInsert;
155             }
156         }
157 
158         nCursorPos++;
159     }
160 
161     size_t getCursorPos( ) const
162     {
163         return nCursorPos;
164     }
165 
166     std::string toString( )
167     {
168         std::string sReturn;
169         StringListNode* pNode = pHead;
170 
171         while ( pNode->cChar != 0 )
172         {
173             sReturn += pNode->cChar;
174             pNode = pNode->pNext;
175         }
176 
177         return sReturn;
178     }
179 
180     friend std::ostream& operator<<( std::ostream& oInput, const StringList& pList )
181     {
182         StringListNode* pNode = pList.pHead;
183 
184         while ( pNode != nullptr )
185         {
186             if ( pNode->cChar != 0 )
187             {
188                 oInput << pNode->cChar;
189             }
190 
191             pNode = pNode->pNext;
192         }
193 
194         return oInput;
195     }
```

```
196
197     friend std::istream& operator>>( std::istream& oInput, StringList& pList )
198     {
199         pList.clear( );
200
201         while ( !oInput.eof( ) )
202         {
203             char cTemp = static_cast<char>( oInput.get( ) );
204
205             if ( cTemp >= ' ' && cTemp <= '~' )
206             {
207                 pList.insert( cTemp );
208             }
209         }
210
211         return oInput;
212     }
213 };
214
215 enum ProgramMode
216 {
217     MODE_NORMAL,
218     MODE_SAVING,
219     MODE_LOADING
220 };
221
222 int main( )
223 {
224     StringList pList, pFileBuffer;
225     char cInput = -1;
226     ProgramMode nMode = MODE_NORMAL;
227
228     fakecurses::init( );
229     fakecurses::clearScreen( );
230
231     while ( cInput != 3 )
232     {
233         cInput = fakecurses::getKey( );
234
235         if ( cInput != -1 )
236         {
237             StringList& pCurrentList = ( nMode == MODE_NORMAL ? pList : pFileBuffer
                    );
238
239             if ( cInput >= ' ' && cInput <= '~' )
240             {
241                 /* See if we did a control sequence using the arrow keys */
242
243                 if ( cInput == '[' )
244                 {
245                     char cNext = fakecurses::getKey( );
246
247                     if ( cNext == -1 )
248                     {
249                         pCurrentList.insert( cInput );
250                     }
251                     else
252                     {
253                         if ( cNext == 'D' )
```

```
254                         {
255                             pCurrentList.left( );
256                         }
257                         else if ( cNext == 'C' )
258                         {
259                             pCurrentList.right( );
260                         }
261                     }
262                 }
263                 else
264                 {
265                     pCurrentList.insert( cInput );
266                 }
267             }
268             else if ( cInput == 127 )
269             {
270                 pCurrentList.erase( );
271             }
272             else if ( cInput == 19 && nMode == MODE_NORMAL) /* Control+S */
273             {
274                 nMode = MODE_SAVING;
275             }
276             else if ( cInput == 6 && nMode == MODE_NORMAL ) /* Control+F */
277             {
278                 nMode = MODE_LOADING;
279             }
280             else if ( cInput == 13 && nMode != MODE_NORMAL ) /* Enter button */
281             {
282                 std::string sFilename = pFileBuffer.toString( );
283                 std::cerr << sFilename << std::endl;
284 
285                 if ( nMode == MODE_SAVING )
286                 {
287                     std::ofstream sStream( sFilename );
288                     sStream << pList;
289                 }
290                 else if ( nMode == MODE_LOADING )
291                 {
292                     std::ifstream sStream( sFilename );
293                     sStream >> pList;
294                 }
295 
296                 nMode = MODE_NORMAL;
297                 pFileBuffer.clear( );
298             }
299 
300             fakecurses::clearScreen( );
301             fakecurses::setCursor( 1, 1 );
302             std::cout << pList;
303             fakecurses::setCursor( 1 + static_cast<short>( pList.getCursorPos( ) ),
                    1 );
304 
305             if ( nMode != MODE_NORMAL )
306             {
307                 fakecurses::setCursor( 1, fakecurses::getScrHeight( ) );
308 
309                 if ( nMode == MODE_SAVING )
310                 {
311                     std::cout << "Save to: ";
```

```
                }
                else
                {
                    std::cout << "Load from: ";
                }

                std::cout << pFileBuffer;
            }
        }
    }

    fakecurses::cleanup( );

    return 0;
}
```

Listing 19: ../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 );

```

```
42      if ( y < 0 )
43          printf( "%s%i%s", ANSI_PREFIX, y*-1, ANSI_MOVEUP );
44
45      if ( y > 0 )
46          printf( "%s%i%s", ANSI_PREFIX, y, ANSI_MOVEDOWN );
47  }
48
49  void fakecurses::showCursor( )
50  {
51      printf( "%s%s", ANSI_PREFIX, ANSI_SHOWCURSOR );
52  }
53
54  void fakecurses::hideCursor( )
55  {
56      printf( "%s%s", ANSI_PREFIX, ANSI_HIDECURSOR );
57  }
58
59  void fakecurses::resetColor( )
60  {
61      printf( "%s0%s", ANSI_PREFIX, ANSI_SETMODE );
62  }
63
64  void fakecurses::setColor( ANSI_COLOR fg, ANSI_COLOR bg )
65  {
66      printf( "%s%i;%i%s", ANSI_PREFIX, ANSI_FG_BASE + fg, ANSI_BG_BASE + bg,
            ANSI_SETMODE );
67  }
68
69  void fakecurses::setTextMode( ANSI_TEXT mode )
70  {
71      printf( "%s%i%s", ANSI_PREFIX, mode, ANSI_SETMODE );
72  }
73  void fakecurses::clearScreen( )
74  {
75      printf( "%s%s", ANSI_PREFIX, ANSI_CLEARSCR );
76      printf( "%s0;0%s", ANSI_PREFIX, ANSI_POSCURSOR );
77  }
78
79  void fakecurses::clearLine( )
80  {
81      printf( "%s%s", ANSI_PREFIX, ANSI_CLEARLINE );
82  }
83
84  void fakecurses::printChar( char c )
85  {
86      printf( "%c", c );
87  }
88
89  void fakecurses::printString( const char* szStr )
90  {
91      printf( "%s", szStr );
92  }
93
94  short fakecurses::getScrWidth( )
95  {
96      struct winsize w;
97      ioctl( 1, TIOCGWINSZ, &w );
98
99      return static_cast<short>( w.ws_col );
```

```
100 }
101
102 short fakecurses::getScrHeight( )
103 {
104     struct winsize w;
105     ioctl( 1, TIOCGWINSZ, &w );
106
107     return static_cast<short>( w.ws_row );
108 }
109
110 void fakecurses::getScrSize( short& w, short& h )
111 {
112     struct winsize ws;
113     ioctl( 1, TIOCGWINSZ, &ws );
114
115     w = static_cast<short>( ws.ws_col );
116     h = static_cast<short>( ws.ws_row );
117 }
118
119 void fakecurses::init( )
120 {
121     struct termios newTermios;
122
123     /* Save old terminal information */
124
125     tcgetattr( 0, &originalTermios );
126     memcpy( &newTermios, &originalTermios, sizeof( termios ) );
127
128     /* Set the terminal to raw input mode immediately */
129
130     cfmakeraw( &newTermios );
131     tcsetattr( 0, TCSANOW, &newTermios );
132
133     setbuf( stdout, NULL );
134
135     clearScreen( );
136 }
137
138 void fakecurses::cleanup( )
139 {
140     tcsetattr( 0, TCSANOW, &originalTermios );
141
142     setColor( COLOR_WHITE, COLOR_BLACK );
143     showCursor( );
144     resetColor( );
145 }
146
147 bool fakecurses::isKeypress( )
148 {
149     struct timeval sTimeout;
150     fd_set fds;
151
152     sTimeout.tv_sec = 0;
153     sTimeout.tv_usec = 0;
154
155     FD_ZERO( &fds );
156     FD_SET( 0, &fds );
157
158     return ( select( 1, &fds, NULL, NULL, &sTimeout ) != 0 );
```

```
159 }
160
161 char fakecurses::getKey( )
162 {
163     char cKey;
164     long nResult;
165
166     nResult = read( 0, &cKey, sizeof( char ) );
167
168     if ( nResult == -1 )
169         return -1;
170
171     return cKey;
172 }
```

Listing 20: ../lab/text-editor/fakecurses.hpp

```
1  #ifndef LIB_FAKECURSES_HPP
2  #define LIB_FAKECURSES_HPP
3
4  namespace fakecurses
5  {
6      enum ANSI_TEXT
7      {
8          TEXT_NORMAL = 0,
9          TEXT_BOLD   = 1,
10         TEXT_UNDERSCORE = 4,
11         TEXT_BLINK  = 5,
12         TEXT_REVERSE    = 7,
13         TEXT_CONCEALED  = 8
14     };
15
16     enum ANSI_COLOR
17     {
18         COLOR_BLACK = 0,
19         COLOR_RED,
20         COLOR_GREEN,
21         COLOR_YELLOW,
22         COLOR_BLUE,
23         COLOR_MAGENTA,
24         COLOR_CYAN,
25         COLOR_LGRAY,
26         COLOR_DGRAY = 60,
27         COLOR_LRED,
28         COLOR_LGREEN,
29         COLOR_LYELLOW,
30         COLOR_LBLUE,
31         COLOR_LMAGENTA,
32         COLOR_LCYAN,
33         COLOR_WHITE
34     };
35
36     void setCursor( short x, short y );
37     void moveCursor( short x, short y );
38     void showCursor( );
39     void hideCursor( );
40
41     void resetColor( );
42     void setColor( ANSI_COLOR fg, ANSI_COLOR bg );
```

```
43      void setTextMode( ANSI_TEXT mode );
44
45      void clearScreen( );
46      void clearLine( );
47
48      void printChar( char c );
49      void printString( const char* szStr );
50
51      short getScrWidth( );
52      short getScrHeight( );
53      void getScrSize( short& w, short& h );
54
55      void init( );
56      void cleanup( );
57
58      bool isKeypress( );
59      char getKey( );
60  }
61
62  #endif
```

3.1.3 Compiler Output

Listing 21: ../lab/text-editor/compilerout

```
1       text-editor git:(master)       make CC=harper_cpp
2   harper_cpp  -std=c++14 main.cpp fakecurses.cpp -o editor.out
3   fakecurses.cpp...
4   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 22: environment

```
1       tex git:(master)       pwd
2   /Users/nicolas/Git/portfolio2/tex
3       tex git:(master)       uname -a
4   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
5       tex git:(master)       clang --version
6   Apple LLVM version 8.0.0 (clang-800.0.42.1)
7   Target: x86_64-apple-darwin16.1.0
8   Thread model: posix
9   InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10      tex git:(master)       harper_cpp --version
11  This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
```

```
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 23: ../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) */

```

```
47              pOperands.push( strtod( szToken, nullptr ) );
48          }
49          else
50          {
51              /* Else, we have an operator */
52
53              double dRight, dLeft, dResult;
54
55              dRight = pOperands.top( );
56              pOperands.pop( );
57
58              dLeft = pOperands.top( );
59              pOperands.pop( );
60
61              switch ( szToken[0] )
62              {
63              case '+':
64                  dResult = dLeft + dRight;
65                  break;
66              case '-':
67                  dResult = dLeft - dRight;
68                  break;
69              case '/':
70                  dResult = dLeft / dRight;
71                  break;
72              case '*':
73                  dResult = dLeft * dRight;
74                  break;
75              case '^':
76                  dResult = pow( dLeft, dRight );
77                  break;
78              case '%':
79                  dResult = fmod( dLeft, dRight );
80                  break;
81              }
82
83              pOperands.push( dResult );
84          }
85
86          szToken = strtok( nullptr, " " );
87      }
88
89      delete[] szInput;
90
91      /* The last operand on the stack is the value of the expression */
92
93      return pOperands.top( );
94  }
95
96  int main( )
97  {
98      std::string sExpression;
99      std::cout << "Please enter an expression in postfix (reverse polish) notation:"
            << std::endl;
100     std::getline( std::cin, sExpression );
101
102     std::cout << std::endl << "The value is: " << calculateRPN( sExpression ) << std
            ::endl;
103
```

```
104     return 0;
105 }
```

3.2.3 Compiler Output

Listing 24: ../lab/reverse-polish/compilerout

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

3.2.4 Program Output

Listing 25: ../lab/reverse-polish/progout

```
1     reverse-polish git:(master)      ./rpn.out
2 Please enter an expression in postfix (reverse polish) notation:
3 5 5 +
4
5 The value is: 10
6     reverse-polish git:(master)      ./rpn.out
7 Please enter an expression in postfix (reverse polish) notation:
8 5 5 + 10 *
9
10 The value is: 100
11     reverse-polish git:(master)      ./rpn.out
12 Please enter an expression in postfix (reverse polish) notation:
13 5 5 - 2 +
14
15 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 26: environment

```
1     tex git:(master)      pwd
2 /Users/nicolas/Git/portfolio2/tex
3     tex git:(master)      uname -a
4 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
5     tex git:(master)      clang --version
6 Apple LLVM version 8.0.0 (clang-800.0.42.1)
7 Target: x86_64-apple-darwin16.1.0
8 Thread model: posix
9 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10     tex git:(master)      harper_cpp --version
11 This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13 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 27: ../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 28: ../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 );
    }
```

```
22
23      BinaryTreeNode& operator=( const BinaryTreeNode& pNode )
24      {
25          if ( pNode.pLeft != nullptr )
26          {
27              pLeft = new BinaryTreeNode( *pNode.pLeft );
28          }
29
30          if ( pNode.pRight != nullptr )
31          {
32              pRight = new BinaryTreeNode( *pNode.pRight );
33          }
34
35          pParent = pNode.pParent;
36
37          return *this;
38      }
39
40      ~BinaryTreeNode( )
41      {
42          if ( pLeft != nullptr )
43          {
44              delete pLeft;
45          }
46
47          if ( pRight != nullptr )
48          {
49              delete pRight;
50          }
51      }
52
53      Data& getData( )
54      {
55          return pData;
56      }
57
58      /* const non-reference versions */
59
60      BinaryTreeNode* getLeftNode( ) const
61      {
62          return pLeft;
63      }
64
65      BinaryTreeNode* getRightNode( ) const
66      {
67          return pRight;
68      }
69
70      BinaryTreeNode* getParent( ) const
71      {
72          return pParent;
73      }
74
75      Data getData( ) const
76      {
77          return pData;
78      }
79
80      Data setData( const Data& pDataNew )
```

```
81      {
82          pData = pDataNew;
83      }
84
85      /* create children */
86
87      BinaryTreeNode* createLeftNode( )
88      {
89          if ( pLeft )
90          {
91              delete pLeft;
92              pLeft = nullptr;
93          }
94
95          pLeft = new BinaryTreeNode;
96          pLeft->pParent = this;
97
98          return pLeft;
99      }
100
101     BinaryTreeNode* createLeftNode( const Data& pDataNew )
102     {
103         if ( pLeft )
104         {
105             delete pLeft;
106             pLeft = nullptr;
107         }
108
109         pLeft = new BinaryTreeNode( pDataNew );
110         pLeft->pParent = this;
111
112         return pLeft;
113     }
114
115     BinaryTreeNode* createRightNode( )
116     {
117         if ( pRight )
118         {
119             delete pRight;
120             pRight = nullptr;
121         }
122
123         pRight = new BinaryTreeNode;
124         pRight->pParent = this;
125
126         return pRight;
127     }
128
129     BinaryTreeNode* createRightNode( const Data& pDataNew )
130     {
131         if ( pRight )
132         {
133             delete pRight;
134             pRight = nullptr;
135         }
136
137         pRight = new BinaryTreeNode( pDataNew );
138         pRight->pParent = this;
139
```

```
140         return pRight;
141     }
142
143     /* checks to see if children exist */
144
145     bool hasLeft( ) const
146     {
147         return ( pLeft ? true : false );
148     }
149
150     bool hasRight( ) const
151     {
152         return ( pRight ? true : false );
153     }
154
155     bool hasChildren( ) const
156     {
157         return ( ( pRight && pLeft ) ? true : false );
158     }
159
160     /* traversals */
161
162     void preorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
            pTraverseFunction )
163     {
164         pTraverseFunction( this );
165
166         if ( pLeft != nullptr )
167         {
168             pLeft->preorderTraversal( pTraverseFunction );
169         }
170
171         if ( pRight != nullptr )
172         {
173             pRight->preorderTraversal( pTraverseFunction );
174         }
175     }
176
177     void postorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
            pTraverseFunction )
178     {
179         if ( pLeft != nullptr )
180         {
181             pLeft->postorderTraversal( pTraverseFunction );
182         }
183
184         if ( pRight != nullptr )
185         {
186             pRight->postorderTraversal( pTraverseFunction );
187         }
188
189         pTraverseFunction( this );
190     }
191
192     void inorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
            pTraverseFunction )
193     {
194         if ( pLeft != nullptr )
195         {
```

```
196             pLeft->inorderTraversal( pTraverseFunction );
197         }
198 
199         pTraverseFunction( this );
200 
201         if ( pRight != nullptr )
202         {
203             pRight->inorderTraversal( pTraverseFunction );
204         }
205     }
206 };
207 
208 template<typename Data>
209 class BinaryTree
210 {
211 protected:
212     BinaryTreeNode<Data>* pRoot;
213 
214 public:
215     /* big three */
216 
217     BinaryTree( ): pRoot( nullptr ) { }
218 
219     BinaryTree( const BinaryTree& pTree ): pRoot( nullptr )
220     {
221         if ( pTree.pRoot != nullptr )
222         {
223             pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
224         }
225     }
226 
227     BinaryTree& operator=( const BinaryTree& pTree )
228     {
229         if ( pRoot != nullptr )
230         {
231             delete pRoot;
232             pRoot = nullptr;
233         }
234 
235         if ( pTree.pRoot != nullptr )
236         {
237             pRoot = new BinaryTreeNode<Data>( *pTree.pRoot );
238         }
239 
240         return *this;
241     }
242 
243     ~BinaryTree( )
244     {
245         if ( pRoot != nullptr )
246         {
247             delete pRoot;
248         }
249     }
250 
251     BinaryTreeNode<Data>*& getRoot( )
252     {
253         return pRoot;
254     }
```

```
255
256     BinaryTreeNode<Data>* getRoot( ) const
257     {
258         return pRoot;
259     }
260
261     BinaryTreeNode<Data>* createRoot( )
262     {
263         if ( pRoot )
264         {
265             delete pRoot;
266             pRoot = nullptr;
267         }
268
269         pRoot = new BinaryTreeNode<Data>;
270
271         return pRoot;
272     }
273
274     BinaryTreeNode<Data>* createRoot( const Data& pData )
275     {
276         if ( pRoot )
277         {
278             delete pRoot;
279             pRoot = nullptr;
280         }
281
282         pRoot = new BinaryTreeNode<Data>( pData );
283
284         return pRoot;
285     }
286
287     void preorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
288     {
289         if ( pRoot != nullptr )
290         {
291             pRoot->preorderTraversal( pTraverseFunction );
292         }
293     }
294
295     void postorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
296     {
297         if ( pRoot != nullptr )
298         {
299             pRoot->postorderTraversal( pTraverseFunction );
300         }
301     }
302
303     void inorderTraversal( const std::function<void(BinaryTreeNode<Data>*)>&
        pTraverseFunction )
304     {
305         if ( pRoot != nullptr )
306         {
307             pRoot->inorderTraversal( pTraverseFunction );
308         }
309     }
310 };
```

```
311
312 #endif
```

3.3.2 Compiler Output

Listing 29: ../lab/linked-btree/compilerout

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

3.3.3 Program Output

Listing 30: ../lab/linked-btree/progout

```
1      linked-btree git:(master)      ./btree.out
2 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 31: environment

```
1      tex git:(master)      pwd
2 /Users/nicolas/Git/portfolio2/tex
3      tex git:(master)      uname -a
4 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
5      tex git:(master)      clang --version
6 Apple LLVM version 8.0.0 (clang-800.0.42.1)
7 Target: x86_64-apple-darwin16.1.0
8 Thread model: posix
9 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
10      tex git:(master)      harper_cpp --version
11 This is harper_cpp version 1.221 executing under perl v5.18.2 and compiling with:
12
13 Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include
      -dir=/usr/include/c++/4.2.1
14 Apple LLVM version 8.0.0 (clang-800.0.42.1)
15 Target: x86_64-apple-darwin16.1.0
16 Thread model: posix
17 InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

3.4.1 Source

Listing 32: ../lab/heapsort/main.cpp

```
1 #include <iostream>
2 #include <vector>
```

```
3  #include <cstdlib>
4  #include <ctime>
5  #include <typeinfo>
6
7  template <typename T>
8  class VectorHeap
9  {
10 private:
11     std::vector<T>& pHeap;
12
13 public:
14     VectorHeap( std::vector<T>& pSetHeap ): pHeap( pSetHeap ) { }
15
16     constexpr size_t getRoot( ) const
17     {
18         return 0;
19     }
20
21     constexpr size_t getLeft( size_t n ) const
22     {
23         return ( n * 2 ) + 1;
24     }
25
26     constexpr size_t getRight( size_t n ) const
27     {
28         return ( n * 2 ) + 2;
29     }
30
31     constexpr size_t getParent( size_t n ) const
32     {
33         if ( n > 0 )
34         {
35             size_t nParent = n - 1;
36
37             if ( nParent % 2 == 1 )
38             {
39                 nParent--;
40             }
41
42             return nParent / 2;
43         }
44         else
45         {
46             return 0;
47         }
48     }
49
50     T& getRootValue( )
51     {
52         return pHeap[0];
53     }
54
55     T& getLeftValue( size_t n )
56     {
57         return pHeap[getLeft(n)];
58     }
59
60     T& getRightValue( size_t n )
61     {
```

```
62          return pHeap[getRight(n)];
63      }
64
65      T& getParentValue( size_t n )
66      {
67          return pHeap[getParent(n)];
68      }
69
70      /* Filter the heap so that the largest value is at the top
71         of the heap. */
72
73      void filterDown( size_t nStart, size_t nEnd )
74      {
75          size_t nRoot = nStart;
76          bool bLooping = true;
77
78          while ( getLeft( nRoot ) <= nEnd && bLooping )
79          {
80              /* While the root has at least one child left */
81
82              size_t nChild, nSwap;
83
84              nChild = getLeft( nRoot );
85              nSwap = nRoot;
86
87              if ( pHeap[nSwap] < pHeap[nChild] )
88              {
89                  /* If the left child is greater, then swap with it */
90
91                  nSwap = nChild;
92              }
93
94              if ( nChild + 1 <= nEnd && pHeap[nSwap] < pHeap[nChild+1] )
95              {
96                  /* If the right child exists, and is greater than what we have
97                     currently, then swap with it. */
98
99                  nSwap = nChild + 1;
100             }
101
102             if ( nSwap == nRoot )
103             {
104                 /* If the root is the largest value, then we're done */
105
106                 bLooping = false;
107             }
108             else
109             {
110                 /* Else, swap the value with the root */
111
112                 std::swap( pHeap[nRoot], pHeap[nSwap] );
113                 nRoot = nSwap;
114             }
115         }
116     }
117
118     void filterDown( )
119     {
120         filterDown( 0, pHeap.size( ) );
```

```
121     }
122
123     void filterUp( size_t nStart, size_t nEnd )
124     {
125         size_t nChild = nEnd;
126         bool bLooping = true;
127
128         while ( nChild > nStart && bLooping )
129         {
130             size_t nParent = getParent( nChild );
131
132             if ( pHeap[nParent] < pHeap[nChild] )
133             {
134                 /* If the heap property is not satisfied */
135
136                 std::swap( pHeap[nParent], pHeap[nChild] );
137                 nChild = nParent;
138             }
139             else
140             {
141                 /* We're done and the heap is fully sorted */
142
143                 bLooping = false;
144             }
145         }
146     }
147
148     void filterUp( )
149     {
150         filterUp( 0, pHeap.size( ) );
151     }
152
153     void heapify( )
154     {
155         for ( size_t i=1; i < pHeap.size( ); i++ )
156         {
157             filterUp( 0, i );
158         }
159     }
160 };
161
162 template<typename T>
163 void printVector( const std::vector<T>& pVector )
164 {
165     std::cout << "Contents of: " << typeid( T ).name( ) << " vector: " << std::endl;
166
167     for ( size_t i=0; i < pVector.size( ); i++ )
168     {
169         std::cout << pVector[i] << " ";
170     }
171
172     std::cout << std::endl;
173 }
174
175 template<typename T>
176 void heapSort( std::vector<T>& pSort )
177 {
178     VectorHeap<T> pHeap( pSort );
179     pHeap.heapify( );
```

```
180
181     for ( size_t i=pSort.size( ) - 1; i > 0; i-- )
182     {
183         std::swap( pSort[i], pSort[0] );
184         pHeap.filterDown( 0, i - 1 );
185     }
186 }
187
188 int main( )
189 {
190     std::vector<int> pSortVec;
191
192     srand( static_cast<unsigned int>( time( nullptr ) ) );
193
194     for ( int i=0; i < 20; i++ )
195     {
196         pSortVec.push_back( rand( ) % 100 );
197     }
198
199     printVector( pSortVec );
200
201     std::cout << std::endl << "Sorting vector..." << std::endl << std::endl;
202
203     heapSort( pSortVec );
204     printVector( pSortVec );
205
206     return 0;
207 }
```

3.4.2 Compiler Output

Listing 33: ../lab/heapsort/compilerout

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

3.4.3 Program Output

Listing 34: ../lab/heapsort/progout

```
1      heapsort git:(master)      ./heapsort.out
2 Contents of: i vector:
3 38 77 8 37 43 36 84 89 18 51 42 73 80 74 16 19 8 3 77 12
4
5 Sorting vector...
6
7 Contents of: i vector:
8 3 8 8 12 16 18 19 36 37 38 42 43 51 73 74 77 77 80 84 89
9      heapsort git:(master)      ./heapsort.out
10 Contents of: i vector:
11 45 26 34 48 73 8 28 20 94 13 35 91 25 16 3 75 88 32 70 77
12
13 Sorting vector...
14
15 Contents of: i vector:
16 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
```

```
76
77 Sorting vector...
78
79 Contents of: i vector:
80 11 21 21 22 23 23 24 37 38 41 44 48 75 77 79 80 82 82 96 97
```