makefile

```
################################################################
# Program:
#     Lesson 02, STACK
#     Brother Helfrich, CS235
# Author:
#     Derek Calkins and David Lambertson
# Summary:
#     Using a stack impelemented by Derek, take an infix
#     equation and turn it into postfix, implemented by David.
# Time:
#     Total hours spent working on this assignment was an average of
#     8 hours. split 50% David and 50% Derek.
################################################################

################################################################
# The main rule
################################################################
a.out: stack.h lesson02.o infix.o
       g++ -g -o a.out lesson02.o infix.o
       tar -cf lesson02.tar *.h *.cpp makefile

################################################################
# The individual components
#      lesson02.o     : the driver program
#      infix.o        : the logic for the infix --> postfix program
################################################################
lesson02.o: stack.h infix.h lesson02.cpp
       g++ -g -c lesson02.cpp

infix.o: stack.h infix.h infix.cpp
       g++ -g -c infix.cpp
```

Commented [HJ1]: perfect

stack.h

```
/***********************************************************************
* Header:
*    Stack
* Summary:
*    This class contains the notion of a stack: a bucket to hold
*    data for the user. This is just a starting-point for more advanced
*    constainers such as the vector, set, stack, queue, deque, and map
*    which we will build later this semester.
*
*    This will contain the class definition of:
*        Stack         : A class that holds stuff
* Author
*    Br. Helfrich
************************************************************************/

#ifndef STACK_H
#define STACK_H

#include <cassert>
#include <iostream>

/*********************************************
 * STACK
 * A class that holds things in an array
 * based on the fact of "First in, last out"
 *********************************************/
template <class T>
class Stack
{
public:
   // default constructor : empty and kinda useless
   Stack() : numItems(0), myCapacity(0), data(0x00000000) {}
```

```
   // copy constructor : copy it
   Stack(const Stack & rhs) throw (const char *);

   // non-default constructor : pre-allocate
   Stack(int myCapacity) throw (const char *);

   // destructor : free everything
   ~Stack()        { if (myCapacity) delete [] data; }

   // is the stack currently empty
   bool empty() const  { return numItems == 0;         }

   // how many items are currently in the stack?
   int size() const    { return numItems;              }

   // add an item to the top of the stack
   void push(const T & t) throw (const char *);

   // checks to see if we need to make the stack size bigger
   void resize(const T & t);

   // removes an item from the top of the stack
   void pop() throw (const char *);

   // looks at what is on the top of the stack
   const T top() const throw (const char *);

private:
   T * data;          // dynamically allocated array of T
   int numItems;      // how many items are currently in the Stack?
   int myCapacity;    // how many items can I put on the Stack before full?
};

/*****************************************
 * STACK :: COPY CONSTRUCTOR
 *****************************************/
template <class T>
Stack <T> :: Stack(const Stack <T> & rhs) throw (const char *)
{
   assert(rhs.myCapacity >= 0);

   // do nothing if there is nothing to do
   if (rhs.myCapacity == 0)
   {
      myCapacity = numItems = 0;
      data = 0x00000000;
      return;
   }

   // attempt to allocate
   try
   {
      data = new T[rhs.myCapacity];
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate buffer";
   }

   // copy over the stuff
   assert(rhs.numItems >= 0 && rhs.numItems <= rhs.myCapacity);
   myCapacity = rhs.myCapacity;
   numItems = rhs.numItems;
   for (int i = 0; i < numItems; i++)
      data[i] = rhs.data[i];
}

/*****************************************
 * STACK :: NON-DEFAULT CONSTRUCTOR
 * Preallocate the stack to "myCapacity"
 *****************************************/
template <class T>
Stack <T> :: Stack(int myCapacity) throw (const char *)
{
   assert(myCapacity >= 0);

   // do nothing if there is nothing to do
   if (myCapacity == 0)
   {
```

Commented [HJ2]: this could definitely throw. Why are you taking a T as a parameter? I am quite confused. This should definitely be a private member function.

Commented [HJ3]: Should be by-reference so we can change the top element.

Commented [HJ4]: Why not call grow()?

```
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return;
   }

   // attempt to allocate
   try
   {
      data = new T[myCapacity];
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate buffer";
   }

   // copy over the stuff
   this->myCapacity = myCapacity;
   this->numItems = 0;
}

/*****************************************
 * STACK :: PUSH
 * Adds to the top of the stack
 *****************************************/
template <class T>
void Stack <T> :: push(const T & t) throw (const char *)
{
   resize(t);
   data[numItems++] = t;
}

/*****************************************
 * STACK :: POP
 * Removes the top of the stack
 *****************************************/
template <class T>
void Stack <T> :: pop() throw (const char *)
{
   if (numItems == 0)
      throw "ERROR: Unable to pop from an empty Stack";
   else
      data[numItems] = data[numItems--];
}

/*****************************************
 * STACK :: TOP
 * Looks at the top of the stack
 *****************************************/
template <class T>
const T Stack <T> :: top() const throw (const char *)
{
   if (numItems == 0)
      throw "ERROR: Unable to reference the element from an empty Stack";
   else
      return data[numItems - 1];
}

/*****************************************
 * STACK :: RESIZE
 * Reallocates the size of the stack if you need
 * to push more items than current capacity.
 *****************************************/
template <class T>
void Stack <T> :: resize(const T & t)
{
   if (myCapacity == 0)
   {
      myCapacity += 1;
      data = new T[myCapacity];
   }
   else if (myCapacity == numItems)
   {
      T * newData;
      myCapacity *= 2;
      try
      {
         newData = new T[myCapacity];
      }
      catch(std::bad_alloc)
```

Commented [HJ5]: The assignment operator in this case does nothing.

```
      {
         throw "Unable to allocate a new buffer for Stack";
         myCapacity /= 2;
      }

      int i;
      for (i = 0; i < numItems; i++)
         newData[i] = data[i];
      newData[i] = '\0';

      delete [] data;
      data = newData;
   }
}

#endif // STACK_H
```

Commented [HJ6]: When you can throw, you should have a throw-list.

vector.h

Commented [HJ7]: Are you using this here? If not, please don't include.

```
/******************
 * Vector.h
 * This file contains the classes for a Vector
 * and its iterator and the different functions
 * needed for its implementation.
 ******************/

#ifndef VECTOR_H
#define VECTOR_H

#include <cassert>
#include <iostream>
using namespace std;

//because we use this in the Vector class
template<class T>
class VectorIterator;

/********
 * the implementation of the Vector Class
 ********/
template <class T>
class Vector
{
  public:
   // default constructor
  Vector() : myCapacity(0), numItems(0), data(0x00000000) {}

   // non-default contsructor
   Vector(int capacity) throw (const char *);

   // copy constructor
   Vector(const Vector & rhs) throw (const char *) { *this = rhs; }

   // destructor
   ~Vector() { if (myCapacity) delete [] data; }

   // is the container empty
   bool empty() const { return numItems == 0; }

   //return how many items are in the Vector
   int size() const { return numItems; }

   //how big is the Vector
   int capacity() const { return myCapacity; }

   //clears the Vector
   void clear() { numItems = 0; }

   // the push back function, allowing the user to add values to the Vector.
   // Also doubles the size and reallocates when the Vector gets full.
   void push_back(T newValue);

   // the square bracket operator overload
   const T operator [](int item) const;

   //overloaded assignment operator
```

```
   Vector<T> & operator = (const Vector<T>& rhs);

   // sets an iterator to the first item of data
   VectorIterator<T> begin() { return VectorIterator<T>(data); }

   // sets an iterator to the last item of data
   VectorIterator<T> end() {return VectorIterator<T>(data +numItems); }

  private:
   int myCapacity; //how big the Vector is
   int numItems;   //the number of Items in the Vector
   T * data;       // the storage unit of the Vector

};

//implementation of the VectorIterator class
template <class T>
class VectorIterator
{
  public:
   //default constructor
  VectorIterator() : p(0x00000000) {}

   //non-default constructor
  VectorIterator(T * p) : p(p) {}

   //copy contructor
   VectorIterator(const VectorIterator & rhs) { *this = rhs; }

   //assignent operator overloaded
   VectorIterator & operator = (const VectorIterator & rhs)
   {
      this->p = rhs.p;
      return *this;
   }

   //not equal operator
   bool operator != (const VectorIterator & rhs) const
   {
      return rhs.p != this->p;
   }
   //de reference operator
   T & operator * ()
   {
      return *p;
   }

   //increment.
   VectorIterator <T> & operator ++ ()
   {
      p++;
      return *this;
   }

   VectorIterator <T> operator++ (int postfix)
   {
      VectorIterator tmp(*this);
      p++;
      return tmp;
   }

  private:
   T * p;

};



//implementation of the non default constructor
template <class T>
Vector <T> :: Vector (int capacity) throw (const char *)
{
   assert(capacity >= 0);

   if (capacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return;
```

```
   }

   try
   {
      data = new T[capacity];
   }
   catch (std::bad_alloc)
   {
      throw "ERROR: Unable to allocate a new buffer for Vector";
   }


   this->myCapacity = capacity;
   this->numItems = 0;
}

//implementation of the push back function
template<class T>
void Vector <T> :: push_back(T newValue)
{
   T * newData;
   if (myCapacity == 0)
   {
      myCapacity += 1;
      data = new T[myCapacity];
   }

   if (myCapacity == numItems)
   {
      myCapacity *= 2;

      try
      {
         newData = new T[myCapacity];
      }

      catch(...)
      {
         cout << "Unable to allocate a buffer for Vector";
         myCapacity /= 2;
      }

      int i;
      for (i = 0; i < numItems; i++)
      {
         newData[i] = data[i];
      }
      newData[i] = '\0';

      delete [] data;
      data = newData;

   }

   data[numItems] = newValue;
   numItems++;
}

//implementation of the square bracket operator
template <class T>
const T Vector<T> :: operator [] (int item) const
{
   return this->data[item];
}

//implementation of the assignment overator
template<class T>
Vector<T> & Vector<T> :: operator = (const Vector<T>& rhs)
{
   assert(rhs.myCapacity >= 0);

   if (rhs.myCapacity == 0)
   {
      this->myCapacity = this->numItems = 0;
      this->data = 0x00000000;
      return *this;
   }

   try
```

```
   {
      this->data = new T[rhs.myCapacity];
   }
   catch(std:: bad_alloc)
   {
      throw "ERROR: Unable to allocate buffer";
   }

   //  assert(rhs.numItems >= 0 && rhs.numItems <= rhs.myCapacity);
   this->myCapacity = rhs.myCapacity;
   this->numItems = rhs.numItems;
   for (int i = 0; i < numItems; i++)
   {
      this->data[i] = rhs.data[i];
   }
}

#endif
```

infix.h

Commented [HJ8]: Unchanged.

```
/***********************************************************************
* Header:
*    INFIX
* Summary:
*    This will contain just the prototype for the convertInfixToPostfix()
*    function
* Author
*    <your names here>
***********************************************************************/

#ifndef INFIX_H
#define INFIX_H

/*****************************************************
 * TEST INFIX TO POSTFIX
 * Prompt the user for infix text and display the
 * equivalent postfix expression
 *****************************************************/
void testInfixToPostfix();

/*****************************************************
 * TEST INFIX TO ASSEMBLY
 * Prompt the user for infix text and display the
 * resulting assembly instructions
 *****************************************************/
void testInfixToAssembly();

#endif // INFIX_H
```

infix.cpp

```
/***********************************************************************
 * Module:
 *    Lesson 02, Stack
 *    Brother Helfrich, CS 235
 * Author:
 *    David Lambertson and Derek Calkins
 * Summary:
 *    This program will implement the testInfixToPostfix()
 *    and testInfixToAssembly() functions
 ************************************************************************/

#include <iostream>    // for ISTREAM and COUT
#include <string>      // for STRING
#include <cassert>     // for ASSERT
#include "stack.h"     // for STACK
using namespace std;

string getWord(const string & infix, int & i);
int check(const string & infix, const int & spot);
void orderOfOps(const string & infix, int & spot, Stack<int> & operators1,
                Stack<char> & operators, int & operation, string & postfix);

/*****************************************************
 * CONVERT INFIX TO POSTFIX
 * Convert infix equation "5 + 2" into postfix "5 2 +"
```

```
 ************************************************/
string convertInfixToPostfix(const string & infix)
{
   string postfix;

   Stack<char> operators; //stack to store the
   Stack<int> operators1; //stack to store the

   int spot = 0;
   int operation = -1;
   bool para = false;

   postfix += ' ';
   //loop until we are at the end of the string
   do
   {
      //if we have a paraenthesis, do the order of operations first.
      if (infix[spot] == '(')
      {
         para = true;
      }
      else
         para = false;

      if (infix[spot] == ' ' || infix[spot+1] == ' ')
      {
         while (infix[spot] ==  ' ')
            spot++;
      }


      if (para)
      {
         //what operation are we trying to do
         operation = check(infix, spot);
         // use order of operations to place them properly
         orderOfOps(infix, spot, operators1, operators,
                    operation, postfix);
      }

      //gets the words from the string
      postfix += getWord(infix, spot);

      //add space to string
      if (postfix[spot] != ' ')
         postfix += ' ';

      if (!para)
      {
         operation = check(infix, spot);
         orderOfOps(infix, spot, operators1, operators,
                    operation, postfix);
      }

   }
   while(spot < infix.size());

   //gets rid of any extra space at the end of the string
   while (postfix[postfix.size()-1] == ' ')
   {
      if ( postfix[postfix.size() - 1] == ' ')
         postfix.erase(postfix.size()-1);
   }

   string tmp;

   //gets rid of any double spaces in the string
   for ( int i = 0; i < postfix.size(); i++)
   {

      tmp += postfix[i];
      if (postfix[i] == ' ' && postfix[i + 1] == ' ')
      {
         i++;
      }
   }
   postfix = tmp;
   return postfix;
}
```

Commented [HJ9]: What is this? A paragraph mark?

Commented [HJ10]: Should be done in the toxenizing

Commented [HJ11]: Could be any number of spaces.

```
/*************************************************
 * TEST INFIX TO POSTFIX
 * Prompt the user for infix text and display the
 * equivalent postfix expression
 *************************************************/
void testInfixToPostfix()
{
   string input;
   cout << "Enter an infix equation.  Type \"quit\" when done.\n";

   do
   {
      // handle errors
      if (cin.fail())
      {
         cin.clear();
         cin.ignore(256, '\n');
      }

      // prompt for infix
      cout << "infix > ";
      getline(cin, input);

      // generate postfix
      if (input != "quit")
      {
         string postfix = convertInfixToPostfix(input);
         cout << "\tpostfix: " << postfix << endl << endl;
      }
   }
   while (input != "quit");
}

/*******************************************
 * CONVERT POSTFIX TO ASSEMBLY
 * Convert postfix "5 2 +" to assembly:
 *     LOAD 5
 *     ADD 2
 *     STORE VALUE1
 *******************************************/
string convertPostfixToAssembly(const string & postfix)
{
   string assembly;

   return assembly;
}

/*************************************************
 * TEST INFIX TO ASSEMBLY
 * Prompt the user for infix text and display the
 * resulting assembly instructions
 *************************************************/
void testInfixToAssembly()
{
   string input;
   cout << "Enter an infix equation.  Type \"quit\" when done.\n";

   do
   {
      // handle errors
      if (cin.fail())
      {
         cin.clear();
         cin.ignore(256, '\n');
      }

      // prompt for infix
      cout << "infix > ";
      getline(cin, input);

      // generate postfix
      if (input != "quit")
      {
         string postfix = convertInfixToPostfix(input);
         cout << convertPostfixToAssembly(postfix);
      }
   }
   while (input != "quit");
```

Commented [HJ12]: Not attempted.

```
}

/*************
 * pulls the word out of the
 *string given to us by the user.
 *****************/
string getWord(const string & infix, int & spot)
{
   string tmp2;
   for (spot; spot < infix.size(); spot++)
   {
      //break out of the loop if we encounter any of these characters
      if (infix[spot] == '(' || infix[spot] == ')' || infix[spot] == '*'||
          infix[spot] == '+' || infix[spot] == '-' ||infix[spot] == '/' ||
          infix[spot] == '^' || spot == infix.size() || infix[spot] == ' ')
      {
         break;
      }
      else
         tmp2 += infix[spot];
   }

   // returns the word which we have gotten out of the string. just one word
   return tmp2;
}

/********************
 * uses the order of operations to
 * determine where to place the
 * operators given to us in the string.
 ******************/
void orderOfOps(const string & infix, int & spot, Stack<int> & operators1,
                Stack<char> & operators, int & operation,
                  string & postfix)
{
   if (operators.empty())                  //if the string is empty
   {
      operators.push(infix[spot]);
      operators1.push(operation);
      spot += 1;                           //move to next item in input
      if (infix[spot] == ' ')              //skip reading spaces
         spot += 1;
   }
   else if (spot < infix.size())  //while we still have data in the string
   {
      if (operators1.top() < operation)    //if the order is higher
      {
         operators.push(infix[spot]);
         operators1.push(operation);
         spot++;
         if (infix[spot] == ' ')
            spot++;
         if (operators1.top() == 9)        //if we run into a ')'
         {
            operators.pop();
            operators1.pop();
            while (operators.top() != '(') // pull all out until '('
            {
               postfix += operators.top();
               operators.pop();
               operators1.pop();
            }
            operators.pop();              //pop off '('
            operators1.pop();

         }
      }
      else if (operators1.top() >= operation)  // if the order is lower
      {
         if (operation == 0)              // if the operation is '('
         {
            operators.push(infix[spot]);
            operators1.push(operation);
            spot++;
            if (infix[spot] == ' ')
               spot++;
            return;
         }
```

Commented [HJ13]: Need better variable names.

Where is tmp1?

Commented [HJ14]: Wow that is a ton of parameters. Are all really needed?

```
         postfix += operators.top();      //top, pop, then push.
         if (infix[spot+1] != ' ')        //we add our own spaces
            postfix += ' ';
         operators.pop();
         operators1.pop();
         operators.push(infix[spot]);
         operators1.push(operation);


         spot++;
         if (infix[spot] == ' ')         //skip reading in spaces in input
            spot++;
      }
   }

   if (spot == infix.size()) // if we have reach the end of the given string
   {
      int i = 0;
      while (!operators.empty())    //get the rest of the operators left over
      {

         postfix += ' ';                 //add our own spaces
         postfix += operators.top();
         operators.pop();
         operators1.pop();
         i++;
      }
   }
}
/***************
 * Checks to see what symbol we have ran into
 *****************/
int check(const string & infix, const int & spot)
{
   char tmp = infix[spot];

   //checks what operator we are at and returns a number likewise
   switch (tmp)
   {
      case '(':
         return 0;
         break;
      case ')':
         return 9;
         break;
      case '+':
         return 1;
         break;
      case '-':
         return 1;
         break;
      case '*':
         return 2;
         break;
      case '/':
         return 2;
         break;
      case '%':
         return 2;
         break;
      case '^':
         return 3;
         break;
      default:
         return -1;
         break;
   }
   return -1;
}
```

Commented [HJ15]: You only need to pass a single char rather than an entire string and an index.

Commented [HJ16]: Need a better variable name. How about operator?

Commented [HJ17]: I like this.Very nicely done.

lesson02.cpp

```
/**********************************************************************
* Program:
*    Lesson 02, Stack
*    Brother Helfrich, CS 235
* Author:
*    Br. Helfrich
```

```
* Summary:
*    This is a driver program to exercise the Stack class.  When you
*    submit your program, this should not be changed in any way.  That being
*    said, you may need to modify this once or twice to get it to work.
************************************************************************/

#include <iostream>    // for CIN and COUT
#include <string>      //
#include "stack.h"     // your Stack class should be in stack.h
#include "infix.h"     // for testInfixToPostfix() and testInfixToAssembly()
using namespace std;


// prototypes for our four test functions
void testSimple();
void testPush();
void testPop();
void testErrors();

// To get your program to compile, you might need to comment out a few
// of these. The idea is to help you avoid too many compile errors at once.
// I suggest first commenting out all of these tests, then try to use only
// TEST1.  Then, when TEST1 works, try TEST2 and so on.
#define TEST1   // for testSimple()
#define TEST2   // for testPush()
#define TEST3   // for testPop()
#define TEST4   // for testErrors()

/************************************************************************
 * MAIN
 * This is just a simple menu to launch a collection of tests
 ************************************************************************/
int main()
{
   // menu
   cout << "Select the test you want to run:\n";
   cout << "\t1. Just create and destroy a Stack.\n";
   cout << "\t2. The above plus push items onto the Stack.\n";
   cout << "\t3. The above plus pop items off the stack.\n";
   cout << "\t4. The above plus exercise the error handling.\n";
   cout << "\ta. Infix to Postfix.\n";
   cout << "\tb. Extra credit: Infix to Assembly.\n";

   // select
   char choice;
   cout << "> ";
   cin  >> choice;
   switch (choice)
   {
      case 'a':
         cin.ignore();
         testInfixToPostfix();
         break;
      case 'b':
         cin.ignore();
         testInfixToAssembly();
         break;
      case '1':
         testSimple();
         cout << "Test 1 complete\n";
         break;
      case '2':
         testPush();
         cout << "Test 2 complete\n";
         break;
      case '3':
         testPop();
         cout << "Test 3 complete\n";
         break;
      case '4':
         testErrors();
         cout << "Test 4 complete\n";
         break;
      default:
         cout << "Unrecognized command, exiting...\n";
   }

   return 0;
}
```

Commented [HJ18]: Good.

```
/*****************************************
 * TEST SIMPLE
 * Very simple test for a Stack: create and destroy
 *****************************************/
void testSimple()
{
#ifdef TEST1
   // Test1: a bool Stack with default constructor
   cout << "Create a bool Stack using the default constructor\n";
   Stack <bool> s1;
   cout << "\tSize:  " << s1.size() << endl;
   cout << "\tEmpty? " << (s1.empty() ? "Yes" : "No") << endl;

   // Test2: double Stack with non-default constructor
   cout << "Create a double Stack using the non-default constructor\n";
   Stack <double> s2(10 /*capacity*/);
   cout << "\tSize:  " << s2.size() << endl;
   cout << "\tEmpty? " << (s2.empty() ? "Yes" : "No") << endl;

   {
      // Test3: copy the bool Stack
      cout << "Copy the double Stack using the copy-constructor\n";
      Stack <double> s3(s2);
      cout << "\tSize:  " << s1.size() << endl;
      cout << "\tEmpty? " << (s1.empty() ? "Yes" : "No") << endl;
   }
   cout << "\tDestroying the third Stack\n";
#endif //TEST1
}

/*****************************************
 * TEST PUSH
 * Add a whole bunch of items to the stack.  This will
 * test the stack growing algorithm
 *****************************************/
void testPush()
{
#ifdef TEST2
   // create
   cout << "Create an integer Stack with the default constructor\n";
   Stack <int> s;

   cout << "\tEnter numbers, type 0 when done\n";
   int number;
   do
   {
      cout << "\t> ";
      cin  >> number;
      if (number)
         s.push(number);
   }
   while (number);

   // display how big it is
   cout << "\tSize:  " << s.size() << endl;
   cout << "\tEmpty? " << (s.empty() ? "Yes" : "No") << endl;
#endif // TEST2
}

/*****************************************
 * TEST POP
 * We will test both Stack::pop() and Stack::top()
 * to make sure the stack looks the way we expect
 * it to look.
 *****************************************/
void testPop()
{
#ifdef TEST3
   // create
   cout << "Create a string Stack with the default constructor\n";
   Stack <string> s;

   // instructions
   cout << "\tTo add the word \"dog\", type +dog\n";
   cout << "\tTo pop the word off the stack, type -\n";
   cout << "\tTo see the top word, type *\n";
   cout << "\tTo quit, type !\n";
```

```
   // interact
   char instruction;
   string word;
   try
   {
      do
      {
         cout << "\t> ";
         cin  >> instruction;
         switch (instruction)
         {
            case '+':
               cin >> word;
               s.push(word);
               break;
            case '-':
               s.pop();
               break;
            case '*':
               cout << s.top() << endl;
               break;
            case '!':
               cout << "\tSize:  " << s.size() << endl;
               cout << "\tEmpty? " << (s.empty() ? "Yes" : "No") << endl;
               break;
            default:
               cout << "\tInvalid command\n";
         }
      }
      while (instruction != '!');
   }
   catch (const char * error)
   {
      cout << error << endl;
   }
#endif // TEST3
}

/*****************************************
 * TEST ERRORS
 * Numerous error conditions will be tested
 * here, including bogus popping and the such
 *****************************************/
void testErrors()
{
#ifdef TEST4
   // create
   cout << "Create a char Stack with the default constructor\n";
   Stack <char> s;

   // test using Top with an empty stack
   try
   {
      s.top();
      cout << "BUG! We should not be able to top() with an empty stack!\n";
   }
   catch (const char * error)
   {
      cout << "\tStack::top() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }

   // test using Pop with an empty stack
   try
   {
      s.pop();
      cout << "BUG! We should not be able to pop() with an empty stack!\n";
   }
   catch (const char * error)
   {
      cout << "\tStack::pop() error message correctly caught.\n"
           << "\t\"" << error << "\"\n";
   }
#endif // TEST4
}
```

Test Bed Results

```
Test bed did not pass

cs235d.out:

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > 1
   > Create a bool Stack using the default constructor
   >    Size:  0
   >    Empty? Yes
   > Create a double Stack using the non-default constructor
   >    Size:  0
   >    Empty? Yes
   > Copy the double Stack using the copy-constructor
   >    Size:  0
   >    Empty? Yes
   >    Destroying the third Stack
   > Test 1 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > 2
   > Create an integer Stack with the default constructor
   >    Enter numbers, type 0 when done
   >    > 9
   >    > 8
   >    > 7
   >    > 6
   >    > 5
   >    > 4
   >    > 3
   >    > 2
   >    > 1
   >    > 0
   >    Size:  9
   >    Empty? No
   > Test 2 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > 3
   > Create a string Stack with the default constructor
   >    To add the word "dog", type +dog
   >    To pop the word off the stack, type -
   >    To see the top word, type *
   >    To quit, type !
   >    > +Genesis
   >    > +Exodus
   >    > +Levidicus
   >    > +Numbers
   >    > +Deuteronomy
   >    > *
   > Deuteronomy
   >    > -
   >    > -
   >    > -
```

```
   >    > -
   >    > *
   > Genesis
   >    > +Matthew
   >    > +Mark
   >    > *
   > Mark
   >    > +Luke
   >    > +John
   >    > +Acts
   >    > -
   >    > !
   >    Size:  5
   >    Empty? No
   > Test 3 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > 4
   > Create a char Stack with the default constructor
   >    Stack::top() error message correctly caught.
   >    "ERROR: Unable to reference the element from an empty Stack"
   >    Stack::pop() error message correctly caught.
   >    "ERROR: Unable to pop from an empty Stack"
   > Test 4 complete
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > a
   > Enter an infix equation.  Type "quit" when done.
   > infix > 4 + 6
   >    postfix:  4 6 +
   >
   > infix > 3.14159 * diameter
   >    postfix:  3.14159 diameter *
   >
   > infix > 4.5+a5+.1215  +   1
   >    postfix:  4.5 a5 + .1215 + 1 +
   >
   > infix > pi*r^2
   >    postfix:  pi r 2 ^ *
   >
   > infix > (5.0  /  .9)*(fahrenheit - 32)
   >    postfix:  5.0 .9 / fahrenheit 32 - *
   >
   > infix > quit
Program terminated successfully

Started program
   > Select the test you want to run:
   >    1. Just create and destroy a Stack.
   >    2. The above plus push items onto the Stack.
   >    3. The above plus pop items off the stack.
   >    4. The above plus exercise the error handling.
   >    a. Infix to Postfix.
   >    b. Extra credit: Infix to Assembly.
   > > b
   > Enter an infix equation.  Type "quit" when done.
   > infix > 4 + 6
   > infix >
Exp: \tLOAD 4\n
   >
Exp: \tADD 6\n
   >
Exp: \tSTORE VALUE1\n
```

Commented [HJ19]: Good!

```
   > 
Exp: infix >
3.14159 * diameter
   > infix > 
Exp: \tLOAD 3.14159\n
   > 
Exp: \tMULTIPY diameter\n
   > 
Exp: \tSTORE VALUE1\n
   > 
Exp: infix >
4.5+a5+.1215  +   1
   > infix > 
Exp: \tLOAD 4.5\n
   > 
Exp: \tADD a5\n
   > 
Exp: \tSTORE VALUE1\n
   > 
Exp: \tLOAD VALUE1\n
   > 
Exp: \tADD .1215\n
   > 
Exp: \tSTORE VALUE2\n
   > 
Exp: \tLOAD VALUE2\n
   > 
Exp: \tADD 1\n
   > 
Exp: \tSTORE VALUE3\n
   > 
Exp: infix >
pi*r^2
   > infix > 
Exp: \tLOAD r\n
   > 
Exp: \tEXPONENT 2\n
   > 
Exp: \tSTORE VALUE1\n
   > 
Exp: \tLOAD pi\n
   > 
Exp: \tMULTIPY VALUE1\n
   > 
Exp: \tSTORE VALUE2\n
   > 
Exp: infix >
(5.0  /  .9)*(fahrenheit - 32)
   > infix > 
Exp: \tLOAD 5.0\n
   > 
Exp: \tDIVIDE .9\n
   > 
Exp: \tSTORE VALUE1\n
   > 
Exp: \tLOAD fahrenheit\n
   > 
Exp: \tSUBTRACT 32\n
   > 
Exp: \tSTORE VALUE2\n
   > 
Exp: \tLOAD VALUE1\n
   > 
Exp: \tMULTIPY VALUE2\n
   > 
Exp: \tSTORE VALUE3\n
   > 
Exp: infix >
quit
Program terminated successfully

Failed 1/6 tests
```

Grading Criteria

Criteria	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%	Weight	Score
Stack interface	The interfaces are perfectly specified with respect to const, pass-by-reference, etc.	lesson02.cpp compiles without modification	All of the methods in Stack match the problem definition	Stack has many of the same interfaces as the problem definition	The public methods in the Stack class do not resemble the problem definition	20	18
Stack Implementation	Passes all four Stack testBed tests	Passes three testBed tests	Passes two testBed tests	Passes one testBed teste	Program fails to compile or does not pass any testBed tests	20	20
Infix to Postfix	The code is elegant and efficient	Passes the Infix to Postfix testBed test	The code essentially works but with minor defects	Elements of the solution are present	The infix to postfix problem was not attempted	30	27
Code Quality	There is no obvious room for improvement	All the principles of encapsulation and modularization are honored	One function is written in a "backwards" way or could be improved	Two or more functions appears "thrown together."	The code appears to be written without any obvious forethought	20	14
Style	Great variable names, no errors, great comments	No obvious style errors	A few minor style errors: non-standard spacing, poor variable names, missing comments, etc.	Overly generic variable names, misleading comments, or other gross style errors	No knowledge of the BYU-I code style guidelines were demonstrated	10	7
Extra Credit	Postfix to assembly function is elegant and efficient	Passes extra credit test bed	Able to generate correct assembly for a simple equation	Elements of the solution are present	Extra credit was not attempted	20	0
Total							86

Commented [HJ20]: You accomplished the assignment, but I do not feel like the code is well engineered. More time on design, less on coding please.