SFT221 – Workshop 6

Learning Outcomes

- Learn to use a logger to debug code,
- Learn to use assertions to aid in debugging,
- Learn debugging techniques.

Instructions

It just gets worse and worse. We let a junior developer work on the code you just fixed and it is broken again! This time, we want you to use a log file and assertions to find the bugs and fix them. DO NOT use the debugger as we want you to compare using log files and assertions to using the debugger. Take notes as to which debugging tools you use as there will be questions about which ones you used to find each bug later.

You should use the log4c library that was discussed in class and add log statements to the code to help you debug the code. You should also insert assertions to detect impossible situations and situations that indicate that something is wrong with the code. These assertions should be spread across all the stringhelp functions to detect possible errors, even after debugging is complete. You should consider that the assertions are being added for the long-term life of the program, not just for this debugging session. With luck, the assertions you add today will detect bugs introduced by other junior developers.

stringhelp.h

```
#pragma once
#ifndef STRINGHELP_H
#define STRINGHELP_H

#define MAX_STRING_SIZE 511
#define MAX_INDEX_SIZE 100
#define MAX_WORD_SIZE 23

struct StringIndex
{
	char str[MAX_STRING_SIZE + 1];
	int wordStarts[MAX_INDEX_SIZE];
	int lineStarts[MAX_INDEX_SIZE];
	int numberStarts[MAX_INDEX_SIZE];
	int numWords, numLines, numNumbers;
};

/**
* Return the index of the next whitespace.
* @param str - the string to search
```

```
* @returns the index of the next white space or the position of the
string terminator.
*/
int nextWhite(const char* str);

/**
* Return true if the string contains only digits.
* @param str - the string to check
* @param len - the number of characters to check
* @returns true if the string is only digits.
*/
int isNumber(const char* str, const int len);


/**
* Build an index for a string showing the start of the lines,
* words, and numbers in the string.
* @param str - the string to search
* @returns the index of the next white space or -1 if there is none.
*/
struct StringIndex indexString(const char* str);

/**
* Return the number of lines in the index.
* @param idx - the index to get the number of lines in it
* @returns the number of lines in the index
*/
int getNumberLines(const struct StringIndex* idx);

/**
* Return the number of words in the index.
* @param idx - the index to get the number of words in it
* @returns the number of words in the index
*/
int getNumberWords(const struct StringIndex* idx);

/**
* Return the number of numbers in the index.
* @param idx - the index to get the number of numbers in it
* @returns the number of numbers in the index
*/
int getNumberNumbers(const struct StringIndex* idx);

/**
* Return the nth word from the index
* @param word - where the result will be placed
* @param idx - the index to use
* @param wordNum - the index of the word to retrieve
* @returns the word or an empty string if index is invalid
*/
```

```
void getWord(char word[], const struct StringIndex* idx, int wordNum);

/**
* Return the nth number from the index
* @param word - where the result will be placed
* @param idx - the index to use
* @param wordNum - the index of the number to retrieve
* @returns the number or an empty string if index is invalid
*/
void getNumber(char word[], const struct StringIndex* idx, int
numberNum);

/**
* Return a pointer to the start of a line
* @param idx - the index to use
* @param lineNum - the index of the line to retrieve
* @returns a pointer to the start of the line
*/
char* getLine(struct StringIndex* idx, int lineNum);

/**
* Prints characters until the terminator is found.
* @param s - the string to print
* @param start - the index to start printing
* @param terminator - the character to stop printing at when
encountered.
*/
void printUntil(const char* s, const int start, const char terminator);

/**
* Prints characters until a space is found.
* @param s - the string to print
* @param start - the index to start printing
*/
void printUntilSpace(const char* s, const int start);

#endif
```

stringhelp.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "stringhelp.h"
#include <ctype.h>
#include <string.h>
#include <stdio.h>

int nextWhite(const char* str)
{
```

```
	int i, result = -1;

	for (i = 0; result < 0 && str[i] != '\0'; i++)
	{
		if (str[i] == ' ' || str[i] == '\t')
		{
			result = i;
		}
	}
	return ((result * -1) != result) ? i : result;
}

int isNumber(const char* str, const int len)
{
	int result;

	for (result = 1; result <= len && result; result++)
	{
		if (!isdigit(str[result - 1])) result = -1;
	}
	return result;
}


struct StringIndex indexString(const char* str)
{
	struct StringIndex result = { {0}, {0}, {0}, 0, 0, 0 };
	char buf[MAX_WORD_SIZE + 1] = { 0 };
	int i, sp;

	strcpy(result.str, str);
	if (str[0] != '\0')
	{
		result.lineStarts[0] = 0;
		result.numLines = 1;
	}

	for (i = 0; str[i] != '\0'; i++)
	{
		while (str[i] != '\0' && isspace(str[i]))
		{
			if (str[i] == '\0')
			{
				result.lineStarts[result.numLines++] = i + 1;
			}
			i++;
		}

		for (sp = 0; str[sp + i] != '\0' && !isspace(str[sp + i]); sp++);
```

```
            if (isNumber(str + i, sp))
            {
                  result.numberStarts[result.numNumbers++] = i;
            }
            else
            {
                  result.wordStarts[result.numWords++] = i;
            }

            i += sp - 1;
      }
      return result;
}

int getNumberLines(const struct StringIndex* idx)
{
      return idx->numLines;
}

int getNumberWords(const struct StringIndex* idx)
{
      return idx->numWords;
}

int getNumberNumbers(const struct StringIndex* idx)
{
      return idx->numNumbers;
}

void getWord(char word[], const struct StringIndex* idx, int wordNum)
{
      int i, sp, start;

      word[0] = '\0';
      if (wordNum < idx->numWords && wordNum >= 0)
      {
            start = idx->wordStarts[wordNum];
            sp = nextWhite(idx->str + start);

            for (i = 0; i <= sp; i++)
            {
                  word[i] = idx->str[start + i];
            }
            word[sp] = '\0';
      }
}

void getNumber(char word[], const struct StringIndex* idx, int numberNum)
{
      int i, sp, start;
```

```
	word[0] = '\0';
	if (numberNum < idx->numNumbers && numberNum >= 0)
	{
		start = idx->numberStarts[numberNum];
		sp = nextWhite(idx->str + start);
		for (i = 0; i < sp; i++)
		{
			word[i] = idx->str[sp + i];
		}
		word[sp] = '\0';
	}
}

char* getLine(struct StringIndex* idx, int lineNum)
{
	char* result = NULL;

	if (lineNum < idx->numLines && lineNum >= 0)
	{
		result = idx->str + idx->lineStarts[lineNum];
	}
	return result;
}

void printUntil(const char* s, const int start, const char terminator)
{
	int i;

	for (i = start; s[i] != '\0' && s[i] != terminator; i++)
		printf("%c", s[i]);
}

void printUntilSpace(const char* s, const int start)
{
	int i;

	for (i = start; s[i] != '\0' && !isspace(s[i]); i++)
		printf("%c", s[i]);
}
```

main.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "stringhelp.h"
#include <string.h>
#include <ctype.h>
```

```
int main(void)
{
	char testStr[] = { "This is a\n string with embedded newline
characters and\n12345 numbers inside it as well 7890" };
	struct StringIndex index = indexString(testStr);
	int i;

	printf("LINES\n");
	for (i = 0; i < index.numLines; i++)
	{
		printUntil(index.str, index.lineStarts[i], '\n');
		printf("\n");
	}

	printf("\nWORDS\n");
	for (i = 0; i < index.numWords; i++)
	{
		printUntilSpace(index.str, index.wordStarts[i]);
		printf("\n");
	}

	printf("\nNUMBERS\n");
	for (i = 0; i < index.numNumbers; i++)
	{
		printUntilSpace(index.str, index.numberStarts[i]);
		printf("\n");
	}

	return 0;
}
```

Deliverables

DueDate:

This workshop is due **at 23:59 pm 2 days after your lab day**. Late submits will not be accepted.

You should submit:

- A zipped Visual Studio project that contains the debugged, working version of the code that also includes your use of the log file and assertions,
- The log file you produced to help in the debugging process,
- A document which lists:
  - The line(s) containing each bug,
  - The corrected version of the line(s),
  - What was wrong with the line(s) and how you fixed it,
  - The debugger tool or technique you used to recognize and find the bug.
- A document called reflect.txt which answers the reflections below.

A Reflection, Research and Assessment

Reflections should consider the questions in depth and not be trivial. Some reflections might require research to properly answer the question. As a rough guideline, the answer to each reflection questions should be about 100 words in length.

1. Did you find more bugs using the log file or assertions? Why did one work better than the other?
2. Do you like to debug using log files or the debugger? Why do you prefer one over the other? Can you envision a situation where you would only be able to use log files? Would you be able to debug as well? Would you be able to debug at the same speed as using a debugger, slower or faster?
3. How did you gain confidence that you had found all the bugs? Would you bet your life savings that you found all the bugs? If not, what would you do so that you would feel comfortable betting that you had found all the bugs?

Marking Rubric

Number of bugs found	15%
Number of bugs correctly fixed	15%
Quality of explanation of each bug	15%
Usage of Log file and assertions	15%
Reflection 1	10%
Reflection 2	15%
Reflection 3	15%