SFT221 – Workshop 5

Learning Outcomes

- Learn to use the Visual Studio debugger,
- Learn general debugging concepts,
- Learn debugging techniques.

Instructions

The following program takes a string and builds an index of it. The index indicates the:

- Start of each line in the string,
- Start of each word in the string,
- Start of each number in the string.

char of array

Unfortunately, a junior developer worked on the code and there are bugs in it. Your job is to use the Visual Studio debugger to find and fix the bugs in the code. You should only use the debugger to find the bugs – DO NOT USE print statements or other techniques. The purpose of this workshop is to get experience with 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 must fix the bugs not re-write the code/functions.

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;
};

/**
```

```c
* 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>
```

```c
int nextWhite(const char* str)
{
	int i, result = -1;

	for (i = 0; result < 0 && str[i] != '\0'; i++)
	{
		if (isspace(str[i]))
		{
			result = i;
		}
	}
	return (result < 0) ? i : result;
}

int isNumber(const char* str, const int len)
{
	int i, result = 1;

	for (i = 0; i < len && result; i++)
	{
		result = result && isdigit(str[i]);
	}
	return result;
}


struct StringIndex indexString(const char* str)
{
	struct StringIndex result = { {0}, {0}, {0}, 0, 0, 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] == '\n')
			{
				result.lineStarts[result.numLines] = i + 1;
			}
			i++;
		}

		sp = nextWhite(str + i);
		if (isNumber(str + i, sp - i + 1))
		{
			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[i];
		}
		word[sp] = '\0';
		((struct StringIndex*)idx)->numWords--;
	}
}

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[start + 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;
}
                                          index
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]);
}
                                               index
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
character and\n12345 numbers inside it as well 67890" };
	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

Due Date:

This workshop is due 2 days after your lab day. Late workshops will not be accepted.

You should submit:

- A zipped Visual Studio project that contains the debugged, working version of the code,
- 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. What was the most useful debugger tool you used to find the bugs? Why was it more useful than other techniques you tried?
2. Which debugger features did you NOT use? Why did you not use these features? For each feature you did not use, give an example of a problem you would use it for.
3. Which do you think is a faster way to find bugs:
    a. Using the debugger alone,
    b. Using print statements alone,
    c. Using the debugger and print statements?
    Justify your answer.
4. How did you test the program to find the bugs and to make sure you had fixed the bugs? Did you use any additional test data other than that supplied? If so, describe the techniques you used to create the test data. How confident are you that you found all the bugs?

Marking Rubric

Number of bugs found	10%
Number of bugs correctly fixed	15%
Quality of explanation of each bug	15%
Updated code compiles and works	20%
Reflection 1	10%
Reflection 2	10%
Reflection 3	10%
Reflection 4	10%