```
// Spring 2019 Midterm 2 Answers.cpp : Defines the entry point for the console
application.
//

#include "stdafx.h"
#include <iostream>

using namespace std;
// PART 1
//H1) HowMany - returns the number of times a character is found in a string
//HowMany(“ABC”, ’A’)➔ 1
//HowMany(“ABC”, ’a’)➔ 0
int HowMany(const char *str, const char c)
{
	int count = 0;
	int i = 0;
	while (str[i])
	{
		if (str[i] == c)
			count++;
		i++;
	}
	return(count);
}

//H2) AlphaOrder – If Letter is A..Z or a..z returns its order in alphabet as number
0..25 otherwise returns - 1
//AlphaOrder(‘A’) ➔ 0 AlphaOrder(‘B’) ➔ 1 AlphaOrder(‘c’) ➔ 2
//AlphaOrder(‘$) ➔ - 1
int AlphaOrder(const char c)
{
	if (c >= 'A' && c <= 'Z')
		return(c - 'A');
	if (c >= 'a' && c <= 'z')
		return(c - 'a');
	return(-1);
}

//H3) StringCopy - overwrites dst with src
void StringCopy(char *dst, const char *src)
{
	int i = 0;
	while (src[i])
	{
		dst[i] = src[i];
		i++;
	}
	dst[i] = 0;
}

//H4) StringLength - returns the length of the string
//StringLength("ABC") == > 3
//StringLength("") == > 0
int StringLength(const char *str)
{
	int i = 0;
	while (str[i])
```

```
            i++;
      return(i);
}

//H5 Swap – Different forms of the Swap function.
//No Partial Credit - To get any points, a) b) and c) must be answered correctly.
//a) Swap - swaps the values between integers i1 and i2
//Swap(1, 2) == > 2, 1
//Swap(3, 4) == > 4, 3
void Swap(int &i1, int &i2)
{
      int temp = i1;
      i1 = i2;
      i2 = temp;
}

//b) Swap - swaps the values between characters c1 and c2
//Swap('A', 'B') == > 'B', 'A'
//Swap('C', 'D') == > 'D', 'C'
void Swap(char &c1, char &c2)
{
      char temp = c1;
      c1 = c2;
      c2 = temp;
}

//c) Swap – swaps two strings with each other
// assumes that no string is larger than 1000
// characters
void Swap(char *str1, char *str2)
{
      char temp[1001];
      StringCopy(temp, str1);
      StringCopy(str1, str2);
      StringCopy(str2, temp);
}

//H6) StringNCopy - Same as StringCopy except will only copy a maximum at most maxDigits
characters
//StringNCopy(“”, ”ABCDE”, 10) ➔ dst = “ABCDE”
//StringNCopy(“”, ”ABCDE”, 2) ➔ dst = “AB”
void StringNCopy(char *dst, const char *src, int maxDigits)
{
      int i = 0;
      while (src[i] && i < maxDigits)
      {
            dst[i] = src[i];
            i++;
      }
      dst[i] = 0;
}

//H7) IsWordDelimiter – determines if c is a specific character that delimits words they
are " ,\r\n\t"
//IsWordDelimiter returns true if this character is a special character false otherwise
//IsWordDelimiter(‘a’)➔false
//IsWordDelimiter(‘, ’)➔true
```

```
bool IsWordDelimiter(const char c)
{
	return(HowMany(" ,\r\n\t", c) > 0);
}

//H8) Sort - takes an array of numbers and sorts them ascending or descending
//Hint - you should use one of the above helper functions
// Sort({1,3,2,4},4) ➔ {1,2,3,4}
// Sort({1,3,2,4},4, false) ➔ {4,3,2,1}
void Sort(int *numbers, int size, bool ascending = true)
{
	bool swapped = true;
	while (swapped)
	{
		swapped = false;
		for (int i = 0; i < size - 1; i++)
		{
			if (ascending)
			{
				if (numbers[i] > numbers[i + 1])
				{
					Swap(numbers[i], numbers[i + 1]);
					swapped = true;
				}
			}
			else // desc
			{
				if (numbers[i] < numbers[i + 1])
				{
					Swap(numbers[i], numbers[i + 1]);
					swapped = true;
				}
			}
		}
	}
}
bool StringCompare(const char *str1, const char*str2)
{
	int i = 0;
	while (str1[i] || str2[i])
	{
		if (str1[i] != str2[i])
			return(false);
		i++;
	}
	return(true);
}
//PART 2
//The function below looks like it works but it doesn’t always.
//Find and Fix the bug.
//Q1) StringCompareWithWildCards - Compares src and pattern to see if they are
equivalent.
//If pattern contains a ‘ ? ’ then any character matches it
//StringCompareWithWildCards("ABC", "AB?") == > true
//StringCompareWithWildCards("ABC", "ABD") == > false
bool StringCompareWithWildCards(const char *src, const char *pattern)
{
```

```
	int i = 0;
//	while (src[i] == pattern[i] || pattern[i] == '?')
// CORRECTION
	while (src[i] && pattern[i] &&
		(src[i] == pattern[i] || pattern[i] == '?'))
	i++;
	return(src[i] == pattern[i]);
}
//Q2) SortByLength - Sorts an array of strings by length.The number of strings in the
array is the “Size” parameter.
//Hint - you should probably use two of the above helper functions
//strings = { “abc”,”ab”,”a”,”” } ➔{ “”,”a”,”ab”,”abc” }
char strings[][1000] = {"abc", "ab", "a", ""};
void SortByLength(char strings[][1000], int Size)
{
	bool swapped = true;
	while (swapped)
	{
		swapped = false;
		for (int i = 0; i < Size - 1; i++)
		{
			if (StringLength(strings[i]) >
				StringLength(strings[i + 1]))
			{
				Swap(strings[i], strings[i + 1]);
				swapped = true;
			}
		}
	}
}
//Q3) StringICompare – Compares two strings using case insensitive compare
//Hint - you should probably use one of the above helper functions
//StringICompare(“ABC”, ”abc”) ➔ true
//StringICompare(“A123”, ”123”) ➔ false
//StringICompare(“A123”, ”a123”) ➔ true
bool StringICompare(const char *str1, const char*str2)
{
	int i = 0;
	while (str1[i] || str2[i])
	{
		if (AlphaOrder(str1[i]) > -1 && AlphaOrder(str2[i]) > -1)
		{
			if (AlphaOrder(str1[i]) != AlphaOrder(str2[i]))
				return(false);
		}
		else if (str1[i] != str2[i])
			return(false);
		i++;
	}
	return(true);
}
//Q4) StringAppend – Appends one string to another.Returns the appended string.
//StringAppend(“”, ”ABC”)➔”ABC”
//StringAppend(“ABC”, ”DE”)➔”ABCDE”
const char * StringAppend(char *dst, const char *src)
{
	int i = 0;
```

```
	while (dst[i])
		i++;
	int j = 0;
	while (src[j])
	{
		dst[i] = src[j];
		i++;
		j++;
	}
	dst[i] = 0;

	return(dst);
}

//Q5) NumberOfWords - determine how many words are in the text.
//Words are text separated by delimiters that are defined in the
// IsWordDelimiter function.
//More than one delimiter together counts as one delimiter.
//For example “ “ and “, , , , ” count as one space and comma respectively.
//NumberOfWords(“ I went to the store”) ➔ 5
//NumberOfWords(“ I, , @%#, , went, to Paris”) ➔ 5
int NumberOfWords(const char *text)
{
	int i = 0;
	int count = 0;
	bool inWord = false;
	while (text[i])
	{
		while (text[i] && IsWordDelimiter(text[i]))
		{
			i++;
			inWord = false;
		}
		while (text[i] && !IsWordDelimiter(text[i]))
		{
			i++;
			if (!inWord)
			{
				inWord = true;
				count++;
			}
		}
	}
	return(count);
}
//Q6) Median - given an array of unsorted numbers return:
//a) The middle number there are an odd amount of numbers
//b) The average of the two middle numbers if the amount of numbers are even
//Hint - numbers are unsorted so you will probably want a helper function
//It can b assumed that there is at least one number in the array
//// {4,3,1,2} ➔ 2.5
//// {4,3,1,2,5} ➔ 3
double Median(int *numbers, int size)
{
	Sort(numbers, size);
	if (size % 2 == 1)
		return(numbers[size / 2]);
```

```
	else
	{
		int sum = numbers[size / 2];
		sum += numbers[(size / 2)-1] ;
		return(sum / 2.0);
	}
}

//PART 3
int ii = 1;
int f2(int i = 20)
{
	return(ii + i);
}

char f1(int i)
{
	ii = 1;
	ii += i + 10;
	return(i);
}

double f1(char c)
{
	return c + 2;
}
int f1(double d)
{
	return(d + 4);
}
int main()
{
	//In the questions 1) and 2) the code and variables continue from one question to
the next.
	//L1)
	double d = 'A';
	char c = f1(f1(d));
	cout << "L1 Output " << c << endl;
	// ANSWER E

	//L2)
	int ii = 1;
	f1(ii);
	ii++;
	cout << "L2 Output " << ii << " " << f2(0) << " " << f2() << endl;
	// ANSWER 2 12 32

	//L3)
	int x = 100, y = 2, z = 3;
	cout << "L3 Output " << x * (y / z) << endl;
	// ANSWER 0

	//L4)
	cout << "L4 Output " << StringLength("") * StringLength("ABC") << endl;
	// ANSWER 0

	//L5)
	char buffer[100];
```

```
	StringCopy(buffer, "abcdefg");
	StringNCopy(buffer, "abcdefg", 5);
	cout << "L5 Output " <<buffer << endl;
	// ANSWER abcde

	//L6
	buffer[5] = 'z';
	cout << "L6 Output " << buffer << endl;
	// ANSWER abcdezg

	//L7
	buffer[0] = 0;
	if (StringCompare("AAA", "aaa"))
		StringNCopy(buffer, "zzz", StringLength("1234567890"));
	else
		StringNCopy(buffer, "z", StringLength(""));
	cout << "L7 Output " << buffer << endl;
	// ANSWER [BLANK LINE]

	//L8
	int i = 10;
	int j = 1;
	while (i)
	{
		i -= j;
		j += 1;
	}
	cout << "L8 Output " << j << endl;
	// ANSWER 5
}
```