CS111 – Spring 2020 Final Answers

TEST A

Part 1 - Basic Functions:10 Points each

B1) StringCopyMax – Never copies more than maxChars to the dest

result: "AB" cout << StringCopyMax(dest, "ABCDEFG", 2) << endl;

result: "ABCDEFG" cout << StringCopyMax(dest, "ABCDEFG", 20) << endl;

```
char* StringCopyMax(char dest[],const char source[], const int maxChars)
{
    int i = 0;
    while (source[i] && i < maxChars)
    {
        dest[i] = source[i];
        i++;
    }
    dest[i] = 0;
    return(dest);
}
```

B2) SumOfEvenNumbers

```
int nums[5] = { 1,2,3,4,5 };
```

result: 6 cout << SumOfEvenNumbers(nums, 5) << endl;

```
int SumOfEvenNumbers (const int nums[],const int size)
{
    int sum = 0;
    for (int i = 0; i < size; i++)
        if (nums[i] % 2 == 0)
            sum += nums[i];
    return(sum)
}
```

B3) StringCompare

result: 0 or false cout << StringCompare("ABC", "abc") << endl;

result: 1 or true cout << StringCompare("ABC", "ABC") << endl;

```
bool StringCompare (const char str1[],const char str2[])
{
    int i = 0;
    while (str1[i] && str1[i] == str2[i])
        i++;
    return(str1[i] == str2[i]);
}
```

Part 2 – Functions – Many of the functions below should use a helper function. Either one from

this test or one you write yourself. 15 points each.

P1a) IsNumber – returns true if a character is a numeric character.
result: 0 or false IsNumber('B')
result: 1 or true IsNumber('0')

```
bool IsNumber(const char c)
{
	return(c >= '0' && c <= '9' ? true : false);
}
```

P1b) ValidatePhoneNumber – returns true if a string has exactly 10 numbers.
A '0' when it is the first number does not count as a phone number.
Non-numeric characters are ignored.
result: 1 or true ValidatePhoneNumber("&&00666--888 1234")
result: 0 or false ValidatePhoneNumber("&&666--0888 1234")

```
bool ValidatePhoneNumber(const char phoneNumber[])
{
	int count = 0;
	for (int i = 0; phoneNumber[i]; i++)
	{
		if (!count && phoneNumber[i] == '0')
			continue;
		if (IsNumber(phoneNumber[i]))
			count++;
	}
	return(count == 10);
}
```

P2a) Upper – Returns the upper case of a character
result: 'A' Upper('a')
result: 'B' Upper('B')

```
char Upper(const char c)
{
	if (c >= 'a' && c <= 'z')
		return('A' + c - 'a');
	else
		return(c);
}
```

P2b) Swap – Swaps the values of the characters passed to it
result: c1 will have the value of c2 and c2 will have the value of c1

```
void Swap(char& c1, char& c2)
{
	char temp = c1;
	c1 = c2;
```

```
	c2 = temp;
}

//"ZqPnMzWa" ==> ZzWqPnMa
```
P2c) BubbleSortDesc – Sorts the characters in a string in Descending order. Note: This function is case insensitive.
result: "ZzWqPnMa" BubbleSortDesc(""ZqPnMzWa"")
```
char *BubbleSortDesc(char str[])
{
	bool swapped=false;
	do
	{
		swapped = false;
		for (int i = 0; i < StringLength(buffer) - 1; i++)
		{
			if (Upper(buffer[i]) < Upper(buffer[i + 1]))
			{
				Swap(buffer[i], buffer[i + 1]);
				swapped = true;
			}
		}

	} while (swapped);

	return(buffer);
}
```
P3a) ContainsChar – returns true or false on whether a string contains a particular character Note: This function is case insensitive
result: true ContainsChar("ABC",'a')
result: false ContainsChar("ABC",'D')
```
bool ContainsChar(const char str[], const char c)
{
	for (int i = 0; str[i]; i++)
		if (Upper(str[i]) == Upper(c))
			return(true);
	return(false);
}
```
P3b) UniqueCharacters – Returns all unique characters found in a string. Note: This function is case insensitive.
result: "aBC" UniqueCharacters(buffer,"aAaaBbbbCccc")
```
char *UniqueCharacters(char buffer[100], const char str[])
{
	int iPos = 0;
	for (int i = 0; str[i]; i++)
		if (ContainsChar(buffer, str[i]) == false)
```

```
        {
            buffer[iPos] = str[i];
            iPos++;
        }

    buffer[iPos] = 0;
    return(buffer);
}
```

TEST B

Part 1 - Basic Functions:10 Points each

B1) StringCopy – Copies the source string into the destination string

result: " ABCDEFG " cout << StringCopy(dest, "ABCDEFG") << endl;

result: "ABC" cout << StringCopy(dest, "ABC") << endl;

```
char* StringCopy(char dest[],const char source[])
{
    int i = 0;
    while (source[i])
    {
        dest[i] = source[i];
        i++;
    }
    dest[i] = 0;
}
```

B2) Swap – Swaps two strings of less than 100 characters with each other

Swap("ABC","DEF")

result: the destination array will hold the contents of the source array and vice versa

```
void Swap(char str1[], char str2[])
{
    char temp[100];
    StringCopy(temp, str1);
    StringCopy(str1, str2);
    StringCopy(str2, temp);
}
```

B3) Min – Returns the second smallest number in a 3 dimensional array

Mistake: Should say "Returns the ~~second~~ smallest number in a 3 dimensional array"

result: If this array was filled with positive numbers, and only one number was negative, this function would return the negative number.

```
int Min(const int nums[5][10][15])
{
    int min = nums[0][0][0];
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 10; j++)
            for (int k = 0; k < 15; k++)
```

```
                                        if (min > nums[i][j][k])
                                                min = nums[i][j][k];
        return(min);
  }
```

Part 2 – Functions – Many of the functions below should use a helper function. Either one from
this test or one you write yourself. 15 points each.

P1a) IndexInArray – return position of c in array, -1 if not found.
result: **0** IndexInArray("ABCDEF", 'A')
result: **-1** IndexInArray("ABCDEF", 'G')

```
int IndexInArray(const char str[], const char c)
{
        for (int i = 0; str[i]; i++)
                if (str[i] == c)
                        return(i);
        return(-1);
}
```

P1b) ReplaceCharacter – returns a replacement character from the second array for the character passed in, if the character exists in the first array. If the character does not exist in the first array, then the character itself is returned. Note: that is function is not case sensitive.

```
replacements = {"AEIOU","@#$%^"}
result:  '@'    ReplaceCharacter('a',replacements)
result:  '^'    ReplaceCharacter('U',replacements) ==>
result:  '^'    ReplaceCharacter('u',replacements) ==>
char ReplaceCharacter(const char c, const char replacements[2][6])
{
        int pos = IndexInArray(replacements[0], ToUpper(c));
        if (pos >= 0)
                return(replacements[1][pos]);
        else
                return(c);
}
```

ReplaceIntoNewString – Takes a string and copies it into another string replacing where possible using the 2D replacements array. Note this function is not case sensitive.

```
replacements = {"AEIOU","@#$%^"}
```

result: "$ w@lk#d t% th# st%r#" ReplaceCharacter(buffer,"I walked to the store",replacements)
suggestion: use ReplaceCharacter as a helper function

```
char* ReplaceIntoNewString(char buffer[], const char str[],
        const char replacements[2][6])
{
        int i = 0;
        for (; i < StringLength(str); i++)
```

```
		buffer[i] = ReplaceCharacter(str[i], replacements);
	buffer[i] = 0;
	return(buffer);
}
```

2) SumProduct - returns the sum of the products of corresponding arrays. I.e. multiply the first row by the second row and return the sum

```
#define AMOUNT_OF_NUMBERS 5
result SumProduct({{1,2,3,4,5},{6,7,8,9,10}}) = (1*6)+(2*7)+(3*8)+(4*9)+(5*10) = 130
int SumProduct(int nums[2][AMOUNT_OF_NUMBERS])
{
	int sum = 0;
	for (int i = 0; i < AMOUNT_OF_NUMBERS; i++)
		sum += nums[0][i] * nums[1][i];
	return(sum);
}
```

3) RandomlyDrawAndDisplay – Using a 2D character array, whose size is SIZE * SIZE, does the following:
1) Initialize the array to underscore.
2) Randomly place the symbol specified as a parameter to the function the number of times specified in the parameter. Note: You cannot place the symbol in the same position twice.
3) Display your results by writing the contents of the 2D to the screen using cout.
See example output below for RandomlyDrawAndDisplay(screen, 13, '#')

```
/*
___#_
#_###
#___#
_#__#
#_###
#define SIZE 5
void RandomlyDrawAndDisplay(char screen[SIZE][SIZE],
	int times,
	const char symbol)
{
	for (int i = 0; i < SIZE; i++)
		for (int j = 0; j < SIZE; j++)
			screen[i][j] = '_';
	int i = 0;
	srand(time(0));
	while (i < times)
	{
		int x = rand() % SIZE;
		int y = rand() % SIZE;
		if (screen[x][y] != symbol)
		{
```

```
                    screen[x][y] = symbol;
                    i++;
                }
        }
        for (int i = 0; i < SIZE; i++)
        {
                for (int j = 0; j < SIZE; j++)
                        cout << screen[i][j];
                cout << endl;
        }
}
```

TEST C

Part 1 - Basic Functions:10 Points each

B1) AppendChar – Adds a character to the end of an existing string and returns a pointer to the string

result: "abcdef" AppendChar("abcde",'f')

result: "a" AppendChar("",'a')

```
char* AppendChar(char str[], const char c)
{
    int i = StringLength(str);
    str[i] = c;
    str[i+1] = 0;
    return(str);
}
```

B2) StringLength – Returns the number of characters in a string

result: 5 StringLength ("abcde")

result: 0 StringLength ("")

```
int StringLength(const char str[])
{
    int i = 0;
    while (str[i])
        i++;
    return(i);
}
```

B3) Min – Returns the second smallest number in a 3 dimensional array

Mistake: Should say "Returns the ~~second~~ smallest number in a 3 dimensional array"

result: If this array was filled with positive numbers, and only one number was negative, this function would return the negative number.

```
int Min(const int nums[5][10][15])
{
    int min = nums[0][0][0];
    for (int i = 0; i < 5; i++)
```

```
                for (int j = 0; j < 10; j++)
                        for (int k = 0; k < 15; k++)
                                if (min > nums[i][j][k])
                                        min = nums[i][j][k];
        return(min);
}
```

Part 2 – Functions – Many of the functions below should use a helper function. Either one from this test or one you write yourself. 15 points each.

P1a) ToUpper – this function returns the upper case value for a character
result: ‘A’ ToUpper(“a”)
result: ‘B’ ToUpper(“B”)

```
char ToUpper(const char c)
{
        if (c >= 'a' && c <= 'z')
                return('A' + c - 'a');
        else
                return(c);
}
```

P1b) CapitalizeName – This function capitalizes the first letter of the string and every character after a space. This result is copied into the buffer parameter and a pointer to the buffer is returned.
result: “Sammy Berda” CapitalizeName(buffer, “sammy berda”)
result: “Jimmy Frank” CapitalizeName(buffer, “jimmy frank”)

```
char* CapitalizeName(char buffer[], const char source[])
{
        int i = 0;
        bool space = false;
        while (source[i])
        {
                if (i == 0 || space == true)
                        buffer[i] = ToUpper(source[i]);
                else
                        buffer[i] = source[i];

                space = false;

                if (source[i] == ' ')
                        space = true;
                i++;
        }
        buffer[i] = 0;
        return(buffer);
}
```

P2) GraphData – Display each amount in an array as a line of symbols. The sysmbol displayed is defined by the displayChar parameter. The number of symbols, correspond to the value of the member of the array divided by the parameter NumberPerSymbols. See example below.

```
int nums[10] = { 10,13,20,50,100,78,56,65,67,45 };
GraphData(nums, 10, 5, '$');
result:
$$
$$
$$$$
$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$
$$$$$$$$$$
$$$$$$$$$$$$
$$$$$$$$$$$$
$$$$$$$$
void GraphData(const int nums[],
        const int arraySize,
        const int numberPerUnit,
        const char displayChar)
{
        for (int i = 0; i < size; i++)
        {
                for (int n = 0; n < nums[i] / numberPerUnit; n++)
                        cout << displayChar;
                cout << endl;
        }
}
```

3) SubString – Copies into buffer and returns the characters beginning at location specified by the start parameter and copies only the number of characters specified by the numOfChars parameter.

Note: start is 0 based. You can never copy past the end of source.

```
result “BC”       SubString(buffer,"ABCD",1, 2);
result “”         SubString(buffer,"ABCD",4, 2);
result: “ABCD” SubString(buffer,"ABCD",0, 10); ==> "ABCD"
char* SubString(char buffer[], const char source[],
        int start, int numOfChars)
{
        int i = 0;
        while (start < StringLength(source) && numOfChars>0)
        {
                buffer[i] = source[start];
                start++;
                numOfChars--;
```

```
                i++;
        }
        buffer[i] = 0;

        return(buffer);
}
```