## Instructions:

Any C++ functions like **cout,** and **srand** that are commonly used in the lecture, can also be used on the test. Other than those functions, you may only use use functions that are included in this test.

Therefore:

1) You cannot use the string class in this test.
2) You may also **not** use recursive functions.
3) You can write your own helper functions as you need them.

# CS111 Final Spring 2019

**NAME** _____

**CUNYFIRSTID** _____

## Helper Functions – These functions may be used elsewhere in the test.

```
// swaps values i1 and i2
void Swap(int &i1, int &i2)
{
	int temp = i1;
	i1 = i2;
	i2 = temp;
}

// swaps values c1 and c2
void Swap(char &c1, char &c2)
{
	char temp = c1;
	c1 = c2;
	c2 = temp;
}


// for letters A..Z and a..z returns a..z
const char  MakeLowerCase(char c);
{
	if (c >= 'A' && c <= 'Z')
	{
		int offset = c - 'A';
		c = 'a' + offset;
	}
	return(c);
}
```

```
// for letters A..Z and a..z returns A..Z
const char  MakeUpperCase(char c)
{
	if (c >= 'a' && c <= 'z')
	{
		int offset = c - 'a';
		c = 'A' + c - 'a';
	}
	return(c);
}


// returns the number of characters in a string
int StringLength(const char *str)
{
	int i = 0;
	while (str[i])
		i++;
	return(i);
}



//copies a string from source array (src) onto the first array (dst)
void StringCopy(char dst[], const char*src)
{
	int i = 0;
	while (src[i])
	{
		dst[i] = src[i];
		i++;
	}
	dst[i] = 0;
}
```

## PART A – Functions – 5 points each

A1) Min - Returns the smallest number in a array

```
//example Min({-7,5,6,-25,87},5) ➔ -25
//example Min({2,2,2,2},4) ➔ 2
// assumes that size is at least one
int Min(const int nums[], int size)
{
	int min = nums[0];
	for (int i = 0; i < size; i++)
	{
		if (nums[i] < min)
			min = nums[i];
	}

	return(min);
}
```

A2) CommonDigits - Outputs digits that two numbers have in common to the screen

```
// Assumes both numbers are greater than 0
// CommonDigits (54321,45679)➔ 5 4
// CommonDigits (2,1)➔
// CommonDigits (313,3)➔ 3 3
void CommonDigits (int num1, int num2)
{
	int original = num2;
	do
	{
		int d1 = num1 % 10;
		num1 /= 10;
		num2 = original;
		do
		{
			int d2 = num2 % 10;
			num2 /= 10;
			if (d1 == d2)
				cout << d1 << " ";
		} while (num2);

	} while (num1);
}
```

A3) NumberOfDigits – Returns the number of digits in a number

```
// NumberOfDigits(0) ➔ 1
// NumberOfDigits(123) ➔ 3
int NumberOfDigits(int num)
{
	int numberOfDigits = 0;
	do
	{
		num /= 10;
		numberOfDigits++;
	} while (num);

	return(numberOfDigits);
}
```

A4) Randomize - Randomly moves every number in an array at least once to a random location in the array

```
//example Randomize ({1,2,3,4,5},5)➔ {5,1,2,4,3}
//example Randomize ({1,2},2)➔ {1,2}
// you can assume that size is at least 2
void Randomize(int numbers[],int size)
{
	srand(time(0));
	for (int i = 0; i < size; i++)
	{
		int r = rand() % size;
		while (r == i)
			r = rand() % size;
		Swap(numbers[i], numbers[r]);
	}
}
```

A5) Fix the array indices below so that they will compile.
If code is correct, write “Correct”.

```
char arr[3] = "ABC";
```

a) arr[4]

```
int arr1[3] = {1,2,3};
```

b) Correct

```
char arr2[4][100] = {“abc”,”def”,”ghi”,”jkl”};
```

c) Correct

```
int arr3[4][3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12},{13,14,15}};
```

d) arr3[5][3]

A6) Regarding the following array: char arr1[5][5][100]

a) What is the maximum number of strings that arr1 can store?
   a) 25
b) What is the maximum length of a string in arr1? b) 99

Regarding the following array: int arr2[5][5][100]

c) What is the maximum number of numbers that arr2 can store?
   c) 2500

## PART B – Functions 6 points each

B1) NumToString - converts an integer to a string
Hint - use the NumberOfDigits function above
// NumToString(buffer,12340)➔ “12340”
// NumToString(`,0)➔ “0”

```
const char *NumToString(char *buffer, int num)
{
	buffer[NumberOfDigits(num)] = 0;
	int i = NumberOfDigits(num) - 1;
	do
	{
		buffer[i] = '0' + num % 10;
		num /= 10;
		i--;
	} while (num);
	return(buffer);
}
```

B2) NetSale - Returns the total dollar amount of all sales. Each sale is a combination of units and sale price.
// NetSale ({{2.0,3.50},{2.0,5.50}},2) ➔ 7.00 + 11.00 ➔ 18.00

```
double NetSale(double sales[][2], int size)
{
	double sum = 0;
	for (int i = 0; i < size; i++)
		sum += (sales[i][0] * sales[i][1]);
	return(sum);
}
```

B3) EvenlyDivide - Output any two numbers in an array that evenly divide with each other

//EvenlyDivide ({3,8,7,5, 27,14,4, 9},8)➔ 3,27 3,9 8,4 7,14 27,3 27,9 14,7 4,8 9,3 9,27

//EvenlyDivide ({3,5,7,11, 13,17,19, 29},8)➔

// note: pairs may be repeated although better if you can non -repeat

```
void EvenlyDivide (int numbers[], int size)
{
	for (int i = 0; i < size; i++)
	{
		for (int j = i+1; j < size; j++)
		{
			if (i != j && (numbers[i] % numbers[j] == 0 ||
				numbers[j] % numbers[i] == 0))
			{
				cout << numbers[i] << "," << numbers[j] << " ";
			}
		}
	}
}
```

B4) SimpleEncrypt - Encrypts words by right shifting the specified number of characters

```
// SimpleEncrypt(“ABC”,2)➔”CDE”
// SimpleEncrypt(“XYZ”,3)➔”ABC”
// you may assume that only uppercase letters are used
const char *SimpleEncrypt(char str[],int count)
{
	for (int i = 0; i < StringLength(str); i++)
	{
		str[i]+=count;
		if (str[i] > 'Z')
			str[i] = 'A' + str[i] -'Z'-1;
	}
	return(str);
}
```

B5) GetNextLicensePlate – Returns next license plate from “AAA-0000” ➔ “ZZZ-9999”

```
// GetNextLicensePlate (“AAA-0009”) ➔ “AAA-0010”
// GetNextLicensePlate (“AAA-9999”) ➔ “AAB-0000”
// GetNextLicensePlate (“AAZ-9999”) ➔ “ABA-0000”
// GetNextLicensePlate (“AZZ-9999”) ➔ “BAA-0000”
// Hint use StringCopy to get initialize newPlate
const char*GetNextLicensePlate(char *newPlate, const char*lastplate)
{
	StringCopy(newPlate, lastplate);
	int i = StringLength(newPlate) - 1;
	newPlate[i]++;
	while (i>0)
	{
		if (i >= 5 && newPlate[i] > '9')
		{
			newPlate[i] = '0';
			newPlate[i - 1]++;
		}
		else  if(i == 4 && newPlate[i] > '9')
		{
			newPlate[i] = '0';
			newPlate[i - 2]++;
		}
		else if(i==3) // do nothing the dash in the plate number
		{
		}
		else if(newPlate[i] > 'Z')
		{
			newPlate[i] = 'A';
			newPlate[i - 1]++;
		}
		i--;
	}
	return(newPlate);
}
```

B6a) Reverse – returns the reverse of the integers of a number

```
// Reverse(123) ➔ 321
// Reverse(450) ➔ 54
Int Reverse(int n)
{
     int ret = 0;
     while (n)
     {
          ret = ret * 10 + n % 10;
          n /= 10;
     }
     return(ret);
}
```

B6b) ReverseNumbers() –Find all numbers from 1 .. 1,000,000 where the Reverse(n) = n * 1.5. Uses Reverse function above

```
// sample output 6534 4356;65934 43956;659934 439956;
void ReverseNumbers()
{
      for (int i = 1; i <= 1000000; i++)
      {
            if (i == Reverse(i)*1.5)
                  cout << i << " "<< Reverse(i) << ";";
      }
      cout << endl;
}
```

B7) ProperCase – Capitalizes the first letter of each word and makes all others lowercase. Each word is separated one or more spaces. Names do not begin with any spaces.Use helper case functions above.

// ProperCase(“alfred b neuman”) ➔ “Alfred B Neuman”
// ProperCase(“JERRY MANFRED”) ➔ “Jerry Manfred””

```
const char*ProperCase(char *name)
{
	int i = 0;
	while (name[i])
	{
		if (i == 0 && name[i] != ' ')
			name[i] = MakeUpperCase(name[i]);
		else if(i > 0 && name[i-1] == ' ')
			name[i] = MakeUpperCase(name[i]);
		else
			name[i] = MakeLowerCase(name[i]);
		i++;
	}
	return(name);
}
```

## PART C – 10 points each

| X= 0, y=0 | Y | Y | Y | Y | Y | |
|---|---|---|---|---|---|---|
| X | Z | Z | Z | Z | Z | X |
| X | Z | Z | Z | Z | z | X |
| | Y | Y | Y | Y | Y | |
| | Y | Y | Y | Y | Y | |

C1) DoSquaresOverlap - given two squares on a grid
// 1) return true if they overlap false otherwise
// 2) If squares overlap, return many pixels they have I common.
// 3) use the 2D variable **screen** declared below, to draw both squares onto the screen
// and answer the questions
// Hint – Remember to initialize screen to blank
//DoSquaresOverlap(1,1,3,4,10,1,3,4,0) ➔ false, NumSquaresThatOverlapp = 0
// for the figure above represents the parameters below
//DoSquaresOverlap(0, 1, 7,2, 1, 0, 5, 5,) ➔ true, NumSquaresThatOverlapp = 10

**const int screenSize = 20;**
**char screen[screenSize][screenSize];**

```
 bool DoSquaresOverlap(
int x1, // x value of upper left hand corner of square 1
 int y1, // y value of upper left hand corner of square 1
int Width1, // length of square 1 on the X axis
int Height1, // length of square 1 on the Y axis
int x2, // x value of upper left hand corner of square 2
 int y2, // y value of upper left hand corner of square 2
int Width2, // length of square 2 on the X axis
int Height2, // length of square 2 on the Y axis
int &NumSquaresThatOverlapp)
{
	bool overlap = false;
	NumSquaresThatOverlapp = 0;

	for (int i = 0; i < screenSize; i++)
		for (int j = 0; j < screenSize; j++)
			screen[i][j] = ' ';

	for (int y = y1; y < y1 + Height1; y++)
		for (int x = x1; x < x1 + Width1; x++)
		{
			//cout << x << " " << y << endl;
			screen[y][x] = 'X';
		}

	for (int y = y2; y < y2 + Height2; y++)
		for (int x = x2; x < x2 + Width2; x++)
		{
			if (screen[y][x] == 'X')
			{
				overlap = true;
				NumSquaresThatOverlapp++;
			}
			screen[y][x] = 'Y';
		}

	// show both squares
	//for (int i = 0; i < screenSize; i++)
	//{
	//	for (int j = 0; j < screenSize; j++)
	//	{
	//		cout << screen[i][j];
	//	}
	//	cout << endl;
	//}
	return(overlap);
}
```

C2) SubString  returns the index of where substr starts in str -1 otherwise
// SubString(“ABCD” “CD”) ➔ 2
// SubString(“ABCD” “E”) ➔ -1

```
int SubString(const char *str ,const char*substr)
{
	int i = 0;
	while (str[i])
	{
		int j = 0;
		while (substr[j] && substr[j] == str[i + j])
			j++;

		if (substr[j] == 0)
			return(i);
		i++;
	}
	return(-1);
}
```

C3) ChartArray –output the values of the array as character bars. Each character represents a specific amount of a unit.

ChartArray({10},1,2) ➔ 10 (value)/ 2(units per symbol) ➔ 5 ➔ *****
ChartArray({10},1,4) ➔ 10 (value)/ 4(units per symbol) ➔ 2 ➔ **
For example, for the array {5,12,40,56,70},5,5 output the following.
void ChartArray(int data[], int Size, int unitsPerSymbol)

```
  5  *
 12  **
 40  ********
 56  ***********
 70  **************
```

```
void ChartArray(int data[],int size, int unitsPerSymbol)
{
        for (int i = 0; i < Size; i++)
        {
                for (int j = 0; j < data[i] / unitsPerSymbol; j++)
                        cout << "*";
                cout << endl;
        }
}
```

# PART D – Ascii Chart from http://www.asciitable.com/

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |