Part 1 – Questions – 9 Questions 2 points each

Q1) Write code that will populate every element of the array below with a random number from .01 through 99.99.
double sales [12][30][100];

```
srand(time(0));
double sales[12][30][100];
for(int i = 0; i < 12; i++)
      for (int j = 0; j < 30; j++)
            for (int k = 0; k < 100; k++)
            {
                  // number between 1 and 9999
                  double d = rand() % 9999 + 1;
                  d /= 100;
                  sales[i][j][k] = d;
                  //cout << d << endl;
            }
```

Q2) Write code that places the string "Hello" into cArray as many times as possible (using the StringCopy function)
char cArray[6][8][6];

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

Q3) What is the purpose of the **const** keyword? (two reasons)

Q3

1) It allows hard-coded values (constants) to be passed to the function
2) It helps others who read the function signature to understand what the function does.
3) It prevents the programmer from changing values that are const.

Q4) By looking at their function signatures, What difference(s) are there between the two f1() functions below?
char * f1(const char arr[]);
and
void f1(char arr[]);

Q4)

1) The first returns a character array.
2) the second modifies the array that is passed in.

Q5) The array below can store how many numbers?
double sales[2][2][2][2];

Q5) 2*2*2*2 = 16

Q6) int numbers[6] = { 3, 6, 3, 8, 5, 1 }
Can a binary search be performed on this array? Explain.
Q6) No. A binary search can only be performed on an array that is sorted.

Q7) A binary search is very efficient because:
Q7) Each time it searches it divides the elements to be searched by ½.

Q8) Where does the name "bubble sort" for the bubble sort algorithm come from?
Q8) Because the highest (or lowest) number bubbles up at the end every time we go through the array.

Q9) What is the output of the call to f1 below? f1() is defined below.
char str[100] = “Hello”;
cout << f1(str) << endl;
Q9) HELLOHELLO

```
char * f1(char str[])
{
        int i = 0;
        while (str[i])
                i++;
        int k = i;
        int j = i;
        i = 0;
        while(k)
        {
                str[j] = str[i];
                k--;
                j++;
                i++;
        }
        str[j] = 0;
        return(str);
}
```

PART 2 – Basic – 3 Questions 5 points each

Abs returns the absolute value for a number

Abs(2) ==> 2

Abs(-2) ==> 2

```
int Abs(int number)
{
	return( number < 0 ? -number : number);
}
```

SumIfGreaterThan - returns the sum of all numbers greater than a specific value

SumIfGreaterThan({1.1,1.2,2.1,3.1},4,2.0) ==> 5.2

```
double SumIfGreaterThan(const double numbers[],
	const int size,
	const double minValue)
{
	double sum = 0;

	for (int i = 0; i < size; i++)
	{
		if (numbers[i] > minValue)
			sum += numbers[i];
	}
	return(sum);
}
```

StringCompare

```
// Compares two strings and return true if they are identical
// StringCompare("ABC", "AB") ==> false
// StringCompare("ABC", "ABC") ==> true
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 3 – Programming - Helper Functions for problems below

You may use the functions below for the problems below.

```
int Max(const int numbers[], const int size)
{
	int max = numbers[0];
	for (int i = 1; i < size; i++)
		if (numbers[i] > max)
			max = numbers[i];
	return(max);
}

bool IsSpaceCharacter(char ch)
```

```
{
	if (ch == ' '/*space*/ ||
		ch == '\t'/*tab*/ ||
		ch == '\n'/*same as endl */ ||
		ch == '\r'/*carriage return*/)
		return(true);
	else
		return(false);
}
char * InsertElement(char str[], int index, char element)
{
	char carry = element;
	while (carry)
	{
		Swap(carry, str[index]);
		index++;
	}
	str[index] = 0;
	return(str);
}
char * RemoveElement(char str[], int index)
{
	while (str[index])
	{
		index++;
		str[index-1] = str[index];
	}
	return(str);
}

void BubbleSort(int numbers[], int size)
{
	bool swapped;
	do
	{
		swapped = false;
		for (int i = 0; i < size - 1; i++)
			if (numbers[i] > numbers[i + 1])
			{
				Swap(numbers[i], numbers[i + 1]);
				swapped = true;
			}
	} while (swapped);
}
StringLength returns the number of characters in a string
StringLength("") ==> 0
StringLength("A") ==> 1
int StringLength(const char str[])
{
	int i = 0;
	while (str[i])
		i++;
	return(i);
}
```

PART 3 – Problems 10 points each

A - PrintHighestToLowest - using an unsorted array print the numbers in the array from highest to lowest.

Note: you **cannot sort the array,** you may want to use the Max function

PrintHighestToLowest({-1,3,2,5,4,6,7,9,8},9) ==> 9 8 7 6 5 4 3 2 -1

PrintHighestToLowest({-1,3,3,5,4,6,7,9,8},9) ==> 9 8 7 6 5 4 3 3 -1

void PrintHighestToLowest(const int numbers[], const int size)

```
{
	int max = Max(numbers, size);
	int count = 0;
	while (count < size)
	{
		for (int i = 0; i < size; i++)
		{
			if (numbers[i] == max)
			{
				count++;
				cout << numbers[i] << " ";
			}
		}
		max--;
	}
}
```

B - NumWordsInSentence - returns number of words in a sentence. Note sentences are separated by space characters you can assume that there is only one space character between words and none at the beginning or end of the sentence

NumWordsInSentence("I just love to dance.") ==> 5

NumWordsInSentence ("The cow jumped over the moon") ==> 6

NumWordsInSentence ("The") ==> 1

NumWordsInSentence ("") ==> 0

int NumWordsInSentence (const char sentence[])

```
{
	// empty string
	if (sentence[0] == 0)
		return(0);

	int i = 0;
	int spaceCount = 0;
	while (sentence[i])
	{
		if (sentence[i] == ' ')
			spaceCount++;
		i++;
	}
	return(spaceCount + 1);}
```

C- Trim - removes all space characters before first word and after last word in a string
Trim(" \t abc def ") => "abc def"
suggestion: use IsSpaceCharacter RemoveElement and StringLength

```
char *Trim(char str[])
{
	// first trim beginning
	while (IsSpaceCharacter(str[0]))
		RemoveElement(str, 0);

	while (IsSpaceCharacter(str[StringLength(str) - 1]))
		RemoveElement(str, StringLength(str) - 1);

	return(str);
}
```

D - Append - appends the second string to the first
Append("AB", "CDE") ==> "ABCDE"
Append("", "CDE") ==> "CDE"
Append("ABC", "") ==> "ABC"
Note: You should probably use the StringLength function

```
char *Append(char str1[], const char str2[])
{
	int i = 0;
	int j = StringLength(str1);
	while (str2[i])
	{
		str1[j] = str2[i];
		i++;
		j++;
	}
	str1[j] = 0;
	return(str1);
}
```

Part 4 – **Three Helper Functions and one Sort function 5 points for A, B, C and D is 10 points**

The following functions all go together
A - ToUpper returns the uppercase of character
ToUpper('a') ==> 'A'
ToUpper('A') ==> 'A'

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

B - OrderByCase compares two characters returns true if they are exactly the same or the first one is lower case and the second is upper case of the same letter i.e. ‘a’ goes before ‘A’
Examples:
OrderByCase('B','a') ==> false
OrderByCase('A','c') ==> true
OrderByCase('a','A') ==> true
OrderByCase('A','a') ==> false
Note: you will probably need a bunch of if statements and the ToUpper function above and IsLowerCase and IsUpperCase functions below

```
bool IsLowerCase(const char c)
{
	return(c >= 'a'&& c <= 'z');
}

bool IsUpperCase(const char c)
{
	return(c >= 'A'&& c <= 'Z');
}

bool OrderByCase(const char c1, const char c2)
{
	//'a' and 'a' (true)
	//'A' and 'A' (true)
	if (c1 == c2)
		return(true);

	// A before B
	if (ToUpper(c1) < ToUpper(c2))
		return(true);

	// B after A
	if (ToUpper(c1) > ToUpper(c2))
		return(false);

	// or 'A' and 'a'(false)
	// either 'a' and 'A' (true),
	return(IsLowerCase(c1) && IsUpperCase(c2));
}
```

C - Swap - Swaps the value of two characters
Example:
c1 = 'A' c2 = 'B' Swap(c1,c2) ==> c1 == 'B' c2 == 'A'

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

D - SortCharactersByOrderAndCase Sorts an array by both character and case using the OrderByCase function above.

Example:
"QqbAAaaBccFFg" ==> "aaAAbBccFFgqQ"

```
char *SortCharactersByOrderAndCase(char str[])
{
	int size = StringLength(str);
	bool swapped;
	do
	{
		swapped = false;
		for (int i = 0; i < size - 1; i++)
			if (!OrderByCase(str[i],str[i + 1]))
			{
				Swap(str[i], str[i + 1]);
				swapped = true;
			}
	} while (swapped);
	return(str);
}
```

ASCII CHART

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com