LAB: 5
Title : Arrays
Lab Objectives: After performing this lab, the students should be able to

- Explain the syntax of array declaration, array assignments and array initialization in C++.
- Write programs to model repetitive data using arrays
- Manipulate the array data structure.

Background:

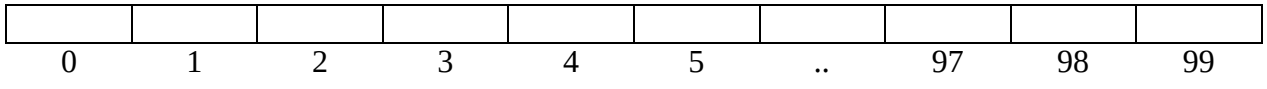
So far we have talked about a variable as a single location in the computer's memory. It is possible to have a collection of memory locations, all of which have the same data type, grouped together under one name. Such a collection is called an array. Like every variable, an array must be defined so that the computer can "reserve" the appropriate amount of memory. This amount is based upon the type of data to be stored and the number of locations, i.e., size of the array, each of which is given in the definition.

Example: Given a list of ages (from a file or input from the keyboard), find and display the number of people for each age. The programmer does not know the ages to be read but needs a space for the total number of occurrences of each "legitimate age." Assuming that ages 1, 2 . . . 100 are possible, the following array definition can be used. Syntax of array in ANSI C++:

```
const int TOTALYEARS = 100;
int main()
{
   int ageFrequency[TOTALYEARS]; //reserves memory for 100 ints
       :
       :
   return 0;
}
```

Following the rules of variable definition, the data type (integer in this case) is given first, followed by the name of the array (*ageFrequency*), and then the total number of memory locations enclosed in brackets. The number of memory locations must be an integer expression greater than zero and can be given either as a named constant (as shown in the above example) or as a literal constant (an actual number such as 100).

Each element of an array, consisting of a particular memory location within the group, is accessed by giving the name of the array and a position with the array (subscript). In C++ the subscript, sometimes referred to as index, is enclosed in square brackets. The numbering of the subscripts always begins at 0 and ends with one less than the total number of locations. Thus the elements in the *ageFrequency* array defined above are referenced as *ageFrequency*[0] through *ageFrequency*[99].



If in our example we want ages from 1 to 100, the number of occurrences of age 4 will be placed in subscript 3 since it is the "fourth" location in the array. This odd way of numbering is often confusing to new programmers; however, it quickly becomes routine.

Array Initialization:

In our example, *ageFrequency*[0] keeps a count of how many 1s we read in, *ageFrequency*[1] keeps count of how many 2s we read in, etc. Thus, keeping track of how many people of a particular age exist in the data read in requires reading each age and then adding one to the location holding the count for that age. Of course it is important that all the counters start at 0. The following shows the initialization of all the elements of our sample array to 0.

```
// pos acts as the array subscript
for (int pos = 0; pos < TOTALYEARS; pos++)
{
     ageFrequency[pos] = 0;
}
```

A simple for loop will process the entire array, adding one to the subscript each time through the loop. Notice that the subscript (*pos*) starts with 0. Why is the condition *pos < TOTALYEARS* used instead of *pos <= TOTALYEARS*? Remember that the last subscript is one less than the total number of elements in the array.Hence the subscripts of this array go from 0 to 99.

Array Processing:

Arrays are generally processed inside loops so that the input/output processing of each element of the array can be performed with minimal statements. Our age frequency program first needs to read in the ages from a file or from the keyboard. For each age read in, the "appropriate" element of the array (the one corresponding to that age) needs to be incremented by one. The following examples show how this can be accomplished:

```
cout << "Please input an age from one" << "to 100. input -99 to
          stop" << endl;
cin >> currentAge;

while (currentAge != -99)
{
     ageFrequency[currentAge-1] = ageFrequency[currentAge-1] + 1;
     cout << "Please input an age from one" << "to 100. input -99
               to stop" << endl;
     cin >> currentAge;
}
```

When we read an age, we increment the location in the array that keeps track of the amount of people in the corresponding age group. Since C++ array indices always start with 0, that location will be at the subscript one value less than the age we read in.

4	0	14	5	0	6		4	1	0
0	1	2	3	4	5	..	97	98	99
1 year	2 years	3 years	2 years	3 years			97 years	99 years	100 years

Each element of the array contains the number of people of a given age. The data shown here is from a random sample run. In writing the information stored in the array, we want to make sure that only those array elements that have values greater than 0 are output. The following code will do this.

Each element of the array contains the number of people of a given age. The data shown here is from a random sample run. In writing the information stored in the array, we want to make sure that only those array elements that have values greater than 0 are output. The following code will do this.

```
for (int ageCounter = 0; ageCounter < TOTALYEARS; ageCounter++)
     if (ageFrequency[ageCounter] > 0)
          cout << "The number of people " << ageCounter + 1
               <<" years old is " << ageFrequency[ageCounter]
               << endl;
```

The `for` loop goes from 0 to one less than `TOTALYEARS` (0 to 99). This will test every element of the array. If a given element has a value greater than 0, it will be output. What does outputting `ageCounter + 1` do? It gives the age we are dealing with at any given time, while the value of `ageFrequency[ageCounter]` gives the number of people in that age group.

Arrays as Arguments:

Arrays can be passed as arguments (parameters) to functions. Although variables can be passed by value or reference, arrays are always passed by pointer, which is similar to pass by reference, since it is not efficient to make a "copy" of all elements of the array. Pass by pointer is discussed further in a later lab. This means that arrays, like pass by reference parameters, can be altered by the calling function. However, they NEVER have the & symbol between the data type and name, as pass by reference parameters do. The program in *lab5sample1.cpp* illustrates how arrays are passed as arguments to functions.

Notice that a set of empty brackets [ ] follows the parameter of an array which indicates that the data type of this parameter is in fact an array. Notice also that no brackets appear in the call to the functions that receive the array. Since arrays in C++ are passed by pointer, which is similar to pass by reference, it allows the original array to be altered, even though no `&` is used to designate this. The `getData` function is thus able to store new values into the array. There may be times when we do not want the function to alter the values of the array. Inserting the word `const` before the

data type on the formal parameter list prevents the function from altering the array even though it is passed by pointer. This is why in the preceding sample program the `findAverage` function and header had the word `const` in front of the data type of the array.

```
// prototype
float findAverage (const int array[], int sizeOfArray);

// function header
float findAverage (const int array[], int sizeOfArray)
```

The variable `numberOfGrades` contains the number of elements in the array to be processed. In most cases not every element of the array is used, which means the size of the array given in its definition and the number of actual elements used are rarely the same. For that reason we often pass the actual number of elements used in the array as a parameter to a procedure that uses the array. The variable `numberOfGrades` is explicitly passed by reference (by using &) to the `getData` function where its corresponding formal parameter is called `sizeOfArray`. Prototypes can be written without named parameters. Function headers must include named parameters.

```
// prototype without named parameters
float findAverage (const int [], int);
```

The use of brackets in function prototypes and headings can be avoided by declaring a programmer defined data type. This is done in the global section with a **`typedef`** statement.

Example: `typedef int GradeType[50];`

This declares a data type, called `GradeType`, that is an array containing 50 integer memory locations. Since `GradeType` is a data type, it can be used in defining variables. The following defines `grades` as an integer array with 50 elements.

```
GradeType grades;
```

It has become a standard practice (although not a requirement) to use an uppercase letter to begin the name of a data type. It is also helpful to include the word "type" in the name to indicate that it is a data type and not a variable. The Program in *lab5sample2.cpp* shows the revised code of Sample Program in *lab5sample1.cpp* using **typedef**.

This method of using **typedef** to eliminate brackets in function prototypes and headings is especially useful for multi-dimensional arrays such as those introduced in the next section.

Two-Dimensional Arrays:

Data is often contained in a table of rows and columns that can be implemented with a two-dimensional array. Suppose we want to read data representing profits (in thousands) for a particular year and quarter. This can be done using a **two-dimensional array**. (see *lab5sample3.cpp*)

Quarter 1	Quarter 2	Quarter 3	Quarter 4
72	80	10	100
82	90	43	42
10	87	48	53

A two dimensional array normally uses two loops (one nested inside the other) to read, process, or output data. How many times will the code above ask for a profit? It processes the inner loop `NO_OF_ROWS * NO_OF_COLS` times, which is 12 times in this case.

Multi-Dimensional Arrays:

C++ arrays can have any number of dimensions (although more than three is rarely used). To input, process or output every item in an n-dimensional array, you need n nested loops.

Arrays of Strings:

Any variable defined as char holds only one character. To hold more than one character in a single variable, that variable needs to be an array of characters. A string (a group of characters that usually form meaningful names or words) is really just an array of characters. A complete lesson on characters and strings is given in a later lab.

Pre-lab

Fill-in-the-Blank Questions

1. The first subscript of every array in C++ is _____ and the last is _____less than the total number of locations in the array.

2. The amount of memory allocated to an array is based on the _____ and the _____ of locations or size of the array.

3. Array initialization and processing is usually done inside a _____.

4. The _____ statement can be used to declare an array type and is often used for multidimensional array declarations so that when passing arrays as parameters, brackets do not have to be used.

5. Multi-dimensional arrays are usually processed within _____ loops.

6. Arrays used as arguments are always passed by _____.

7. In passing an array as a parameter to a function that processes it, it is often necessary to pass a parameter that holds the _____ of _____used in the array.

8. A string is an array of _____.

9. Upon exiting a loop that reads values into an array, the variable used as a(n) _____ to the array will contain the size of that array.

10. An n-dimensional array will be processed within _____nested loops when accessing all members of the array.

Short answer

- Which of the following array declaration and / or initialization is valid (no compile or run time error)? State the reasons. (use *lab5a.cpp* to test your answers)

declaration / initialization	Compile?	Run?	Reason
`int a[80];`			
`int a[10] = {0};`			
`int a [10] = {50};`			
`int a[3] = {0, 1, 2, 3};`			
`int a[ ] = {0, 1, 2};`			

- Write code to initialize an integer array of 1000 elements to {1, 2, 3, 4, 5, …, 1000}.(use *lab5b.cpp* to test your answers)
- Write code to initialize an integer array of 1000 elements to {1000, 999, 998, …. 1} (use *lab5b.cpp* to test your answers)

Lab 1: Working with One-Dimensional Arrays

Retrieve program *lab5d.cpp* from the Lab 5 folder. And perform tasks:

- ***Exercise 1****:* Complete this program as directed.
- ***Exercise 2****:* Run the program with the following data: 90 45 73 62 -99 and record the output here:

  _____

  _____

  _____

Lab 2: Strings as Arrays of Characters

Retrieve program *lab5e.cpp* from the Lab 5 folder. And perform tasks:

- **Exercise 1:** Complete the program by filling in the code. Run the program with 3 grades per student using the sample data below.
  - Mary Brown 100 90 90
  - George Smith 90 30 50
  - Dale Barnes 80 78 82
  - Sally Dolittle 70 65 80
  - Conrad Bailer 60 58 71

  You should get the following results:

```
Mary Brown has an average of 93.33 which gives the letter grade of A
George Smith has an average of 56.67 which gives the letter grade of F
Dale Barnes has an average of 80.00 which gives the letter grade of B
Sally Dolittle has an average of 71.67 which gives the letter grade of C
Conrad Bailer has an average of 63.00 which gives the letter grade of D
```

Lab 3: Working with Two-Dimensional Arrays

Look at the following table containing prices of certain items:
12.78 23.78 45.67 12.67
7.83 4.89 5.99 56.84
13.67 34.84 16.71 50.89
These numbers can be read into a two-dimensional array.

Retrieve program *lab5f.cpp* from the Lab 5 folder. And perform tasks:

- Exercise 1: Fill in the code to complete both functions `getPrices` and `printPrices`, then run the program with the following data:

```
Please input the number of rows from 1 to 10
2
Please input the number of columns from 1 to 10
3
Please input the price of an item with 2 decimal places
1.45
```

```
Please input the price of an item with 2 decimal places
2.56
Please input the price of an item with 2 decimal places
12.98
Please input the price of an item with 2 decimal places
37.86
Please input the price of an item with 2 decimal places
102.34
Please input the price of an item with 2 decimal places
67.89

      1.45     2.56     12.98
      37.86    102.34   67.89
```

- Exercise 2: Why does `getPrices` have the parameters `numOfRows` and `numOfCols` passed by reference whereas `printPrices` has those parameters passed by value?

- Exercise 3: The following code is a function that returns the highest price in the array. After studying it very carefully, place the function in the above program and have the program print out the highest value. (**NOTE**: This is a value returning function. Be sure to include its prototype in the global section.)

```
// This function returns the highest price in the array
float findHighestPrice(PriceType table, int numOfRows, int numOfCols)
{
      float highestPrice;
      highestPrice = table[0][0]; // make first element the highest price

      for (int row = 0; row < numOfRows; row++)
      {
            for (int col = 0; col < numOfCols; col++)
            {
                  if ( highestPrice < table[row][col] )
                  {
                        highestPrice = table[row][col];
                  }
            }
      }
      return highestPrice;
}
```

- Exercise 4: Create another value returning function that finds the lowest price in the array and have the program print that value.

- Exercise 5: After completing all the exercises above, run the program again with the values from Exercise 1 and record your results.

- Exercise 6: Look at the following table that contains quarterly sales transactions for three years of a small company. Each of the quarterly transactions are integers (number of sales) and the year is also an integer.

YEAR	Quarter 1	Quarter 2	Quarter 3	Quarter 4
2000	72	80	60	100
2001	82	90	43	98
2002	64	78	58	84

We could use a two-dimensional array consisting of 3 rows and 5 columns. Even though there are only four quarters we need 5 columns (the first column holds the year). Retrieve program *lab5g.cpp* from the Lab 5 folder. Fill in the code for both getSales and printSales. This is similar to the price.cpp program in Exercise 1; however, the code will be different. This is a table that contains something other than sales in column one.

- Exercise 7: Run the program so that the chart from Exercise 6 is printed.

Lab Assignments:

1. Age Population Program

Write the complete age population program given in the Pre-lab reading assignment.
Statement of the problem: Given a list of ages (1 to 100) from the keyboard, the program will tally how many people are in each age group and prints out the total count for ages with count of greater than zero. Use file *lab5h.cpp* to test and submit your solution.

Sample Run:

```
Please input an age from one to 100, put -99 to stop
5
Please input an age from one to 100, put -99 to stop
10
Please input an age from one to 100, put -99 to stop
100
Please input an age from one to 100, put -99 to stop
20
Please input an age from one to 100, put -99 to stop
5
Please input an age from one to 100, put -99 to stop
8
Please input an age from one to 100, put -99 to stop
20
Please input an age from one to 100, put -99 to stop
5
Please input an age from one to 100, put -99 to stop
9
Please input an age from one to 100, put -99 to stop
17
Please input an age from one to 100, put -99 to stop
-99

The number of people 5 years old is 3
The number of people 8 years old is 1
The number of people 9 years old is 1
The number of people 10 years old is 1
The number of people 17 years old is 1
The number of people 20 years old is 2
The number of people 100 years old is 1
```

2. Average Temperature

Write a program that will input temperatures for consecutive days. The program will store these values into an array and call a function that will return the average of the temperatures. It will also call a function that will return the highest temperature and a function that will return the lowest temperature. The user will input the number of temperatures to be read. There will be no more than 50 temperatures. Use `typedef` to declare the array type. The average should be displayed to two decimal places. Use file *lab5i.cpp* to test and submit your solution.

Sample Run:

```
Please input the number of temperatures to be read
5
Input temperature 1:
68
Input temperature 2:
75
Input temperature 3:
36
Input temperature 4:
91
Input temperature 5:
84


The average temperature is 70.80
The highest temperature is 91.00
The lowest temperature is 36.00
```

3. Number of Grades

Write a program that will input letter grades (A, B, C, D, F), the number of which is input by the user (a maximum of 50 grades). The grades will be read into an array. A function will be called five times (once for each letter grade) and will return the total number of grades in that category. The input to the function will include the array, number of elements in the array and the letter category (A, B, C, D or F). The program will print the number of grades that are A, B, etc. Use file *lab5j.cpp* to test and submit your solution.

Sample Run:

```
Please input the number of grades to be read in. (1-50)
6
All grades must be upper case A B C D or F
Input a grade
A
Input a grade
C
Input a grade
A
Input a grade
B
Input a grade
B
Input a grade
D

Number of A = 2
Number of B = 2
Number of C = 1
Number of D = 1
Number of F = 0
```

4. Hollow Array

An array is said to be hollow if it contains 3 or more zeroes in the middle that are preceded and followed by the same number of non-zero elements. Furthermore, all the zeroes in the array must be in the middle of the array. Write a function named `isHollow` that accepts an integer array and returns 1 if it is a hollow array, otherwise it returns 0. The function signature is

`int isHollow(int a[ ], int len)` where `len` is the number of elements in the array

Use file *lab5k.cpp* to test and submit your solution.

Scenario

Input	hollow?	Reason
{1,2,0,0,0,3,4}	Yes	2 non-zeroes precede and follow 3 zeroes in the middle
{1,1,1,1,0,0,0,0,0,2,1,2,18}	Yes	4 non-zeroes precede and follow the 5 zeroes in the middle
{1, 2, 0, 0, 3, 4}	No	There are only 2 zeroes in the middle; at least 3 are required
{1,2,0,0,0,3,4,5}	No	The number of preceding non-zeroes(2) is not equal to the number of following non-zeroes(3)
{3,8,3,0,0,0,3,3}	No	The number of preceding non-zeroes(3) is not equal to the number of following non-zeroes(2)
{1,2,0,0,0,3,4,0}	No	Not all zeroes are in the middle
{0,1,2,0,0,0,3,4}	No	Not all zeroes are in the middle
{0,0,0}	Yes	The number of preceding non-zeroes is 0 which equals the number of following non-zeroes. And there are three zeroes "in the middle".

Hint: Write three loops. The first counts the number of preceding non-zeroes. The second counts the number of zeroes in the middle. The third counts the number of following non-zeroes. Then analyze the results

5. Flip-flop Array

Define a flip-flop array to be one whose elements alternate between even and odd values or vice versa. A flip-flop array must have at least two elements. For example {0, 3, 4, -7} and {1, 2, 3, 4} are flip-flop arrays but {2, 2, 3, 4}, {1, 2, 3, 3, 4} and {2} are not. Write a function called `isFlipFlop` that returns 1 if its argument is a flip-flop array, otherwise it returns 0. The function signature is

```
int isFlipFlop(int a[ ], int len)
```

Where `len` is the number of elements in the array. Use file *lab5l.cpp* to test and submit your solution.

6. Balanced array

An array is called *balanced* if it's even numbered elements (a[0], a[2], etc.) are even and its odd numbered elements (a[1], a[3], etc.) are odd. Write a function named *isBalanced* that accepts an array of integers and returns 1 if the array is balanced, otherwise it returns 0. The function signature is

```
int isBalanced(int a[ ], int len)
```

Where `len` is the number of elements in the array. Use file *lab5m.cpp* to test and submit your solution.

Scenario

If the input array is	Return	Reason
{2, 3, 6, 7}	1	a[0] and a[2] are even, a[1] and a[3] are odd
{6, 3, 2, 7}	1	a[0] and a[2] are even, a[1] and a[3] are odd.
{6, 7, 2, 3, 12}	1	a[0], a[2] and a[4] are even, a[1] and a[3] are odd
{6, 7, 2, 3, 14, 95}	1	a[0], a[2], and a[4] are even, a[1], a[3] and a[5] are odd.
{7, 15, 2, 3}	0	a[0] is odd
{16, 6, 2, 3}	0	a[1] is even
{2}	1	a[0] is even
{3}	0	a[0] is odd
{}	1	true vacuously

7. Odd-Heavy array

An array is defined to be *odd-heavy* if it contains at least one odd element and every element whose value is odd is greater than every even-valued element. So {11, 4, 9, 2, 8} is odd-heavy because the two odd elements (11 and 9) are greater than all the even elements. And {11, 4, 9, 2, 3, 10} is not odd-heavy because the even element 10 is greater than the odd element 9. Write a function called *isOddHeavy* that accepts an integer array and returns 1 if the array is odd-heavy; otherwise it returns 0. The function signature is

```
int isOddHeavy(int[ ] a, int len)
```

Where `len` is the number of elements in the array. Use file *lab5n.cpp* to test and submit your solution.

Scenario

If the input array is	isOddHeavy should return
{1}	1 (true vacuously)
{2}	0 (contains no odd elements)
{1, 1, 1, 1, 1, 1}	1
{2, 4, 6, 8, 11}	1
{-2, -4, -6, -8, -11}	0

8. Cumulative array

Define an array to be cumulative if the n^{th} (n > 0) element of the array is the sum of the first n elements of the array. So {1, 1, 2, 4, 8} is cumulative because

a[1] == 1 == a[0]
a[2] == 2 == a[0] + a[1]
a[3] == 4 == a[0] + a[1] + a[2]
a[4] == 8 == a[0] + a[1] + a[2] + a[4]

And {1, 1, 2, 5, 9} is not cumulative because a[3] == 5 != a[0] + a[1] + a[2] Write a function named isCumulative that accepts an array of integers and returns 1 if the array is cumulative and 0 otherwise. The function signature is

```
int isCumulative(int a[ ], int len)
```

Where `len` is the number of elements in the array. Use file *lab5o.cpp* to test and submit your solution.

Scenario

If the input array is	isCumulative should return
{1}	0 (array must contain at least 2 elements)
{0,0,0,0,0,0}	1
{1, 1, 1, 1, 1, 1}	0
{3, 3, 6, 12, 24}	1
{-3, -3, -6, -12, -24}	1
{-3, -3, 6, 12, 24}	0

9. Value of the Polynomial

Write a function named *eval* that returns the value of the polynomial

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0 .$$

The function's signature is

```
double eval(double x, int a[], int len)
```

Where `len` is the number of elements in the array `a` which is the list off coefficient values in ***reverse order***. Use file *lab5p.cpp* to test and submit your solution.

Scenario

if x is	if the input array is	this represents	eval should return
1.0	{0, 1, 2, 3, 4}	$4x^4 + 3x^3 + 2x^2 + x + 0$	10.0
2.0	{3, 2, 1}	$x^2 - 2x + 3$	18.0
3.0	{3, -2, -1}	$-x^2 - 2x + 3$	-5.0
-3.0	{3, 2, 1}	$x^2 + 2x + 3$	6.0
2.0	{3, 2}	$2x + 3$	7.0
2.0	{4, 0, 9}	$9x^2 + 4$	40.0
2.0	{10}	10	10.0
10.0	{0, 1}	x	10.0

10. Centered array

An array with an odd number of elements is said to be centered if all elements (except the middle one) are strictly greater than the value of the middle element. Note that only arrays with an odd number of elements have a middle element (a[a.length/2]). Write a function named *isCentered* that accepts an integer array and returns 1 if it is a centered array, otherwise it returns 0. The function signature is:-

```
int isCentered(int a[ ], int len)
```

Where `len` is the number of elements in the array. Use file *lab5q.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 2, 3, 4, 5}	0 (the middle element 3 is **not** strictly less than all other elements)
{3, 2, 1, 4, 5}	1 (the middle element 1 is strictly less than all other elements)
{3, 2, 1, 4, 1}	0 (the middle element 1 is **not** strictly less than all other elements)
{3, 2, 1, 1, 4, 6}	0 (no middle element since array has even number of elements)
{}	0 (no middle element)
{1}	1 (satisfies the condition vacuously)

11. Layered array

An array is called layered if its elements are in ascending order and each element appears two or more times. For example, {1, 1, 2, 2, 2, 3, 3} is layered but {1, 2, 2, 2, 3, 3} and {3, 3, 1, 1, 1, 2, 2} are not. Write a function named isLayered that accepts an integer array and returns 1 if the array is layered, otherwise it returns 0. The function signature is

```
int isLayered(int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5r.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 1, 2, 2, 2, 3, 3}	1
{3, 3, 3, 3, 3, 3, 3}	1
{1, 2, 2, 2, 3, 3}	0 (because there is only one occurence of the value 1)
{2, 2, 2, 3, 3, 1, 1}	0 (because values are not in ascending order)
{2}	0
{}	0

12. Dual array

An array is said to be dual if it has an even number of elements and each pair of consecutive even and odd elements sum to the same value. Write a function named isDual that accepts an array of integers and returns 1 if the array is dual, otherwise it returns 0. The function signature is

```
int isDual(int a[ ], int len)
```

Where `len` is the number of elements in the array. Use file *lab5s.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 2, 3, 0}	1 (because 1+2 == 3+0 == 3)
{1, 2, 2, 1, 3, 0}	1 (because 1+2 == 2+1 == 3+0 == 3)
{1, 1, 2, 2}	0 (because 1+1 == 2 != 2+2)
{1, 2, 1}	0 (because array does not have an even number of elements)
{}	1

13. Daphne array

A **Daphne array** is defined to be an array with alternating positive and negative numbers. It may start with either a positive or negative number but after that the positive and negative numbers alternate. For example, {1, -3, 5, -6}, {-3, 5, -6}, {1, -3, 5}, {-3, 5}, {-3} and {5} are all Daphne arrays. However, the arrays {1, 1, -3, 5}, {-2, -2, -2} and {1, -1, 2, 2} are not Daphne arrays because they either have two consecutive positive or two consecutive negative numbers.

Write a function named ***isDaphneArray*** that returns 1 if its array argument is a Daphne array, otherwise it returns 0. The function signature is

```
int isDaphneArray(int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5t.cpp* to test and submit your solution.

14. Cucumber array

A **Cucumber array** is defined to be an array in which exactly one pair of numbers sum to 15. For example, {3, 8, 12} is a Cucumber array because only 3 and 12 sum to 15. The array {1, 3, 15} is **not** a Cucumber array because no two numbers sum to 15. The array {3, 4, 12, 11} is **not** a Cucumber array because both 3+12 and 4+11 sum to 15 Write a function named ***isCucumber*** that returns 1 if its array argument is a Cucumber array, otherwise it returns 0. The function signature is

```
int isCucumber(int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5u.cpp* to test and submit your solution.

15. Distinct Integers

Write a function `countOfDistinct` that accepts an array of integers and returns the number of distinct integers in the array. The function signature is:

```
int countOfDistinct (int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5v.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 2, 3, 10}	4
{5, 5, 5, 5}	1
{8, 9, 321, 9, 321, 9, 8}	3
{}	0

16. Character filtering

Write a function that takes as input an array of positive integers and an array of characters. The integers in the first array are indexes into the array of characters. The function should return an array of characters containing the characters referenced by the integer array. For example, if the inputs are {0, 4, 7} and {'h', 'a', 'p', 'p', 'i', 'n', 'e', 's', 's'}, the function should return the character array {'h', 'i', 's'} because 'h' is the 0^{th} element of the character array, 'i' is the 4^{th} element and 's' is the 7^{th} element. If an integer in the integer array is not a legal index in the character array, the function should detect this and return null. So, if the integer array in the previous example was {0, 4, 12}, the function would return null because there is no 12^{th} element of the character array. The signature of the function is:-

```
char[] filterChars(int[] index, char[] a, int iLen, int aLen)
```

Where `iLen` and `aLen` are the number of elements in the array index and a, respectively. Use file *lab5w.cpp* to test and submit your solution.

Scenario

if input arrays are	return the array
{1} and {'d', 'i', 'g'}	{'i'}
{2, 1, 0} and {'r', 'a', 'm'}	{'m', 'a', 'r'}
{3, 1, 0} and {'r', 'a', 'm'}	Null
{2, -1, 0} and {'r', 'a', 'm'}	Null
{1, 1, 1} and {'c', 'a', 't'}	{'a', 'a', 'a'}
3 and {'a'}	{'a'}
{} and {'c', 'a', 't'}	{}
3 and {}	Null

17. Chained Iteration

Write a function that will iterate through an array *a* as follows. Start at a[0]. If a[0] is -1 return -1. If a[0] is less than -1 or greater than or equal to the length of the array (i.e., it can't be used to index an element of the array), return 1. Otherwise visit a[a[0]] and repeat these steps. This could potentially result in an infinite loop. If an infinite loop is detected the function should return a 0.

To summarize:

1. Iterate through the array using the value of an element as the index to the next element (like in a linked list)
2. return -1 if a -1 encountered
3. return 1 if a value less than -1 or greater than or equal to the size of the array is encountered.
4. return 0 if an infinite loop is detected.

The function signature is:-

```
int isInfinite(int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5x.cpp* to test and submit your solution.

Scenario

input array	Traversal	Return
{1, 2, -1, 5}	visit a[0], a[1], a[2]	-1 (because -1 is encountered before the 5 is encountered)
{1, 2, 4, -1}	visit a[0], a[1], a[2]	1 (because 4, which is too big to be an index, is encountered before the -1)
{5, 3, 4, -1, 1, 2}	visit a[0], a[5], a[2], a[4], a[1], a[3]	-1 (because a[3] is -1)
{3}	visit a[0]	1 (because 3, which is too big to be an index, is encountered.)
{3, 2, 3, 1}	visit a[0], a[3], a[1], a[2], a[3], ...	0
{0}	visit a[0], a[0], ...	0
{-1}	visit a[0]	-1

18. Sum of Negative Values

Write a C++ function called `negSum` that returns the sum of all negative elements in an array of integers. The main function will call `negSum` by passing an array of integers as a parameter, which `negSum` will go over and return the sum of only the negative numbers.

The function signature is:-

```
int negSum(int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5y.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 2, -1, 5}	-1
{1, -2, 4, -1}	-3
{5, -3, 4, -1, 1, 2}	-4
{-5, -2, -3, -1}	-11
{3, 2, 3, 1}	0
{193,-193}	-193
{-1}	-1
{3}	0
{0}	0

19. Range

Write a C++ function called `range` that returns the difference between the largest and smallest elements in an array.

The function signature is:-

```
int range (int a[], int len)
```

Where `len` is the number of elements in the array. Use file *lab5z.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{1, 2, -1, 5}	6
{1, -2, 4, -1}	6
{5, -3, 4, -1, 1, 2}	8
{-5, -2, -3, -1}	4
{3, 2, 3, 1}	2
{20,-40}	60
{3}	0
{}	0

20. Sum of Values in 2D array

Write a function called `sum2D` that returns the sum of all elements in a 2-dimensional array.

The function signature is:-

```
int sum2D (int a[ ][], int lenX, lenY)
```

Where `lenX` and `lenY` are the dimensions of the array. Use file *lab5za.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{{1, 2, 3, 4},{5, 6, 7, 8}, {3, 2, 1, 0}}	
{{10, 12, -11, 5},{13, 8, 9, 14}, {13, 2, 0, 15}, {48, 20, 94, 51}}	
{{8, 4, 3, 9, 5},{0, 9, 5, 4, 8},{9, 5, 4, 7, 6}}	
{{2, 8, 9}, {2, 6, 7}, {2, 3, 4}, {5, 6, 7}, {3, 4, 6}, {7, 6, 4}, {3, 5, 9}, {2, 8, 6}}	
{{342, 223, 189, 294}}	
{{1}}	1
{ }	0

21. Biggest Entry in 2D array

Write a function called `biggestEntry` that uses a two dimensional array of positive integers and two parameters representing the row and column capacities. The function should return the value of the biggest entry in the array.

The function signature is:-

```
int biggestEntry (int a[ ][], int lenX, lenY)
```

Where `lenX` and `lenY` are the dimensions of the array. Use file *lab5zb.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{{1, 2, 3, 4},{5, 6, 7, 8}, {3, 2, 1, 0}}	8
{{10, 12, -11, 5},{13, 8, 9, 14}, {13, 2, 0, 15}, {48, 20, 94, 51}}	94
{{8, 4, 3, 9, 5},{0, 9, 5, 4, 8},{9, 5, 4, 7, 6}}	9
{{2, 8, 9}, {2, 6, 7}, {2, 3, 4}, {5, 6, 7}, {3, 4, 6}, {7, 6, 4}, {3, 5, 9}, {2, 8, 6}}	9
{{342, 223, 189, 294}}	342
{{1}}	1
{ }	0

22. Number of Sixes in a 2D array

Write a function called `sixCount` that returns a count of the number of entries that are equal to 6 in a 2-dimensional array. The function should use a parameter to specify the array and parameters for the row count and column count

The function signature is:-

```
int sixCount(int a[][], int lenX, lenY)
```

Where `lenX` and `lenY` are the dimensions of the array. Use file *lab5zc.cpp* to test and submit your solution.

Scenario

if the input array is	Return
{{1, 2, 3, 4},{5, 6, 7, 8}, {3, 2, 1, 0}}	1
{{10, 12, -11, 5},{13, 8, 9, 14}, {13, 2, 0, 15}, {48, 20, 94, 51}}	0
{{8, 4, 3, 9, 5},{0, 9, 5, 4, 8},{9, 5, 4, 7, 6}}	1
{{2, 8, 9}, {2, 6, 7}, {2, 3, 4}, {5, 6, 7}, {3, 4, 6}, {7, 6, 4}, {3, 5, 9}, {2, 8, 6}}	5
{{342, 223, 189, 294}}	0
{{1}}	0
{ }	0