Chapter 5 - Arrays / Vectors

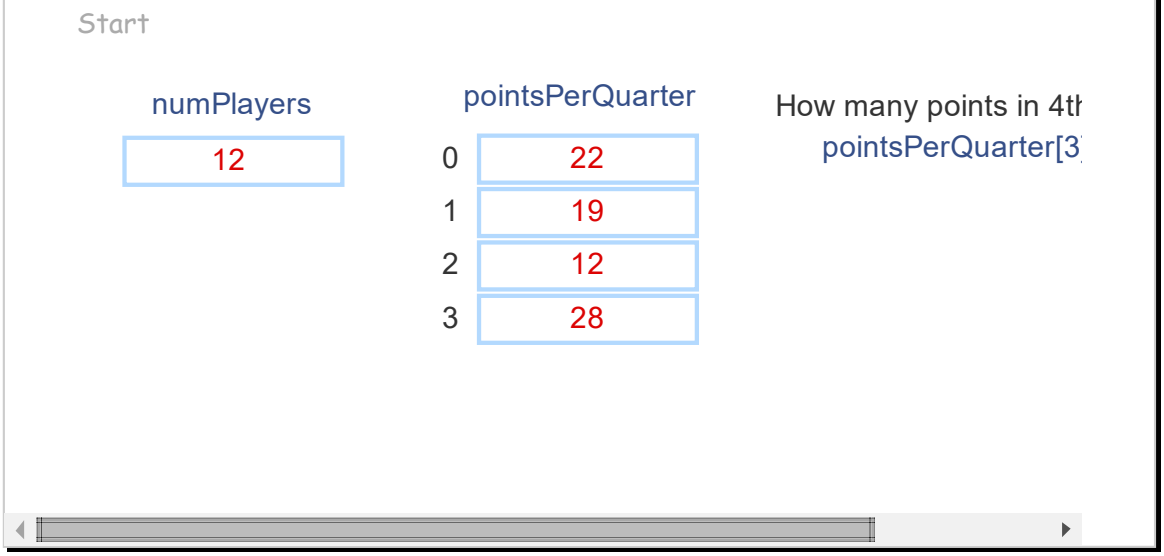
Section 5.1 - Array/vector concept

Note_language_neutral

A typical variable stores one data item, like the number 59 or the character 'a'. Instead, sometimes a *list* of data items should be stored. Ex: A program recording points scored in each quarter of a basketball game needs a list of 4 numbers. Requiring a programmer to define 4 variables is annoying; 200 variables would be ridiculous. An ***array*** is a special variable having one name, but storing a list of data items, with each item directly accessible. Some languages use a construct similar to an array called a ***vector***. Each item in an array is known as an ***element***.

Participation Activity

5.1.1: Sometimes a variable should store a list, or array, of data items.



You might think of a normal variable as a truck, and an array variable as a train. A truck has just one car for carrying "data", but a train has many cars each of which can carry data.

Figure 5.1.1: A normal variable is like a truck, whereas an array variable is like a train.


(Source for above images: Truck, Train)

In an array, each element's location number is called the ***index***; myArray[2] has index 2. An array's key feature is that the index enables direct access to any element, as in myArray[2]; different languages may use different syntax, like myArray(3) or myVector.at(3). In many languages, indices start with 0 rather than 1, so an array with 4 elements has indices 0, 1, 2, and 3.

Participation Activity

5.1.2: Update the array's data values.

Start

Update myItems with the given code.

myItems

0	**6**
1	**38**
2	**68**
3	**43**
4	**26**
5	**96**
6	**92**

1	2	3	4	5	6

Check Next

Participation Activity

5.1.3: Array basics.

Array peoplePerDay has 365 elements, one for each day of the year. Valid accesses are peoplePerDay[0], [1], ..., [364].

#	Question	Your answer
1	Which assigns element 0 with the value 250?	peoplePerDay[250] = 0
		peoplePerDay[0] = 250
		peoplePerDay = 250
2	Which assigns element 1 with the value 99?	peoplePerDay[1] = 99
		peoplePerDay[99] = 1

3	Given the following statements:
	`peoplePerDay[9] = 5;`
	`peoplePerDay[8] = peoplePerDay[9] - 3;`
	What is the value of peoplePerDay[8]?
		8

		5
		2

4	Assume N is initially 1. Given the following:
	`peoplePerDay[N] = 15;`
	`N = N + 1;`
	`peoplePerDay[N] = peoplePerDay[N - 1] * 3;`
	What is the value of peoplePerDay[2]?
		15

		2
		45

Participation Activity

5.1.4: Arrays with element numbering starting with 0.

Array scoresList has 10 elements with indices 0 to 9, accessed as scoresList[0] to scoresList[9].

#	Question	Your answer
1	Assign the first element in scoresList with 77.	
2	Assign the second element in scoresList with 77.	
3	Assign the last element with 77.	
4	If that array instead has 100 elements, what is the last element's index?	
5	If the array's last index was 499, how many elements does the array have?	

(*Note_language_neutral) This section is mostly language neutral

Section 5.2 - Vectors

Previously-introduced variables could each only store a single item. Just as people often maintain lists of items like a grocery list or a course roster, a programmer commonly needs to maintain a list of items. A construct known as vector can be used for this purpose. A ***vector*** is an ordered list of items of a given data type. Each item in a vector is called an ***element***.

Construct 5.2.1: Vector definition.

```
vector<dataType> identifier(numElements);
```

The above statement defines a vector with the specified number of elements, each element of the specified data type.

The following shows how to read and assign values within a vector. The program creates a variable named vals with 3 elements, each of data type int. Those three elements are in fact each a separate variable that is accessed using the syntax vals.at(0), vals.at(1), and vals.at(2). Note that the 3 elements are (some might say unfortunately) numbered 0 1 2 and not 1 2 3. In a vector access, the number within .at() is called the ***index*** of the corresponding element.

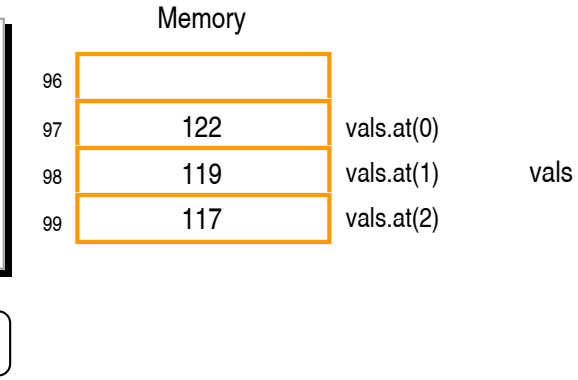
P Participation Activity

5.2.1: A vector definition creates multiple variables in memory, each accessible using .at().

Start

```
vector<int> vals(3);

vals.at(0) = 122;
vals.at(1) = 119;
vals.at(2) = 117;

cout << vals.at(1);
```

119

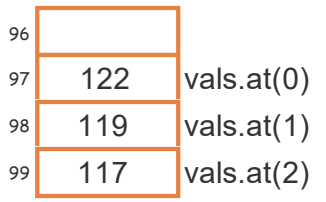


Participation Activity

5.2.2: A vector definition creates multiple variables in memory, each accessible using .at().

Start

```
vector vals(3);

vals.at(0) = 122;
vals.at(1) = 119;
vals.at(2) = 117;

cout << vals.at(1);
```



If you have studied arrays, then know that a vector was added to C++ as a safer and more powerful form of arrays; more later.

Participation Activity

5.2.3: Vector basics.

```
vector<int> yearsList(4);
```

Given:
```
yearsList.at(0) = 1999;
yearsList.at(1) = 2012;
yearsList.at(2) = 2025;
```

#	Question	Your answer
1	How many elements does the vector definition create?	0 1

		3
		4
2	What value is assigned into yearsList.at(1)?	1
		1999
		2012
3	What value does `curr = yearsList.at(2)` assign to curr?	2
		2025
		Invalid index
4	Is `curr = yearsList.at(4)` a valid assignment?	Yes, it accesses the fourth element.
		No, yearsList.at(4) does not exist.
5	What is the proper way to access the *first* element in vector yearsList?	yearsList.at(1)
		yearsList.at(0)
6	What are the contents of the vector if the above code is followed by the statement: yearsList.at(0) = yearsList.at(2)?	1999, 2012, 1999, 0
		2012, 2012, 2025, 0
		2025, 2012, 2025, 0
7	What is the index of the *last* element for the following vector: `vector<int> prices(100);`	99
		100
		101

Besides reducing the number of variables a programmer must define, a powerful aspect of vectors is that the index is an expression. So an access could be written as vals.at(i), where i is an int variable. As such, a vector is useful to easily lookup the Nth item in a list. Consider the following program that allows a user to print the age of the Nth oldest person known to have ever lived.

Figure 5.2.1: Vector's ith element can be directly accessed using .at(i): Oldest people program.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> oldestPeople(5); // Source: Wikipedia.org
   int nthPerson = 0;           // User input, Nth oldest person

   oldestPeople.at(0) = 122; // Died 1997 in France
   oldestPeople.at(1) = 119; // Died 1999 in U.S.
   oldestPeople.at(2) = 117; // Died 1993 in U.S.
   oldestPeople.at(3) = 117; // Died 1998 in Canada
   oldestPeople.at(4) = 116; // Died 2006 in Ecuador

   cout << "Enter N (1..5): ";
   cin  >> nthPerson;

   if ((nthPerson >= 1) && (nthPerson <= 5)) {
      cout << "The " << nthPerson << "th oldest person lived ";
      cout << oldestPeople.at(nthPerson - 1) << " years." << endl;
   }

   return 0;
}
```

```
Enter N (1..5)
The 1th oldest

...

Enter N (1..5)
The 4th oldest

...

Enter N (1..5)

...

Enter N (1..5)

...

Enter N (1..5)
The 5th oldest
```

The program can quickly access the Nth oldest person's age using `oldestPeople.at(nthPerson - 1)`. Note that the index is `nthPerson - 1` rather than just `nthPerson`, because a vector's indices start at 0. So the 1st age is at index 0, the 2nd at index 1, etc.

Participation Activity

5.2.4: Nth oldest person example (vector).

#	Question	Your answer

1	What is the purpose of this check in the above code:
	`if ((nthPerson >= 1) && (nthPerson <= 5)) {`
	`  ...`
	`}`
		To avoid overflow because nthPerson's data type can only store values from 1 to 5.

		To ensure only valid vector elements are accessed because vector oldestPeople only has 5 elements.

A vector's index must be an integer type. The index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

A key advantage of vectors becomes evident when used in conjunction with loops. To illustrate, the following program allows a user to enter 8 integer values, then prints those 8 values:

Figure 5.2.2: Vectors combined with loops are powerful together: User-entered numbers.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_VALS = 8;          // Number of elements in vector
   vector<int> userVals(NUM_VALS); // User values
   int i = 0;                       // Loop index

   cout << "Enter " << NUM_VALS << " integer values..." << endl;
   for (i = 0; i < NUM_VALS; ++i) {
      cout << "Value: ";
      cin >> userVals.at(i);
   }

   cout << "You entered: ";
   for (i = 0; i < NUM_VALS; ++i) {
      cout << userVals.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```

```
Enter 8 intege
Value: 5
Value: 99
Value: -1
Value: -44
Value: 8
Value: 555555
Value: 0
Value: 2
You entered: 5
```

Consider how the program would have been written if using 8 separate variables. The program would have repeated variable definitions, cout statements, and cin statements. Now consider that program for NUM_VALS equal to 100, 1000, or more; requiring a huge amount of code. With vectors and loops, the code would be the same as above. Only the constant literal 8 would be changed.

A vector's elements are automatically initialized to 0s during the variable definition. All a vector's elements may be initialized to another single value as follows: `vector<int> myVector(3, -1);`, which creates myVector with three elements each with value -1. Initializing each element with different values has a more complex syntax, requiring use of an array. Details can be found at cplusplus.com. The new C++11 standard has a simpler vector initialization syntax, but is not yet widely supported by compilers.

A common error is to forget the #include <vector> at the top of the file when using vectors. Trying to then define a vector variable may yield a strange compiler error message, such as:

Figure 5.2.3: Strange error message when forgetting to include the vector library.

```
$ g++ -Wall testfile.cpp
testfile.cpp:12: error: ISO C++ forbids declaration of vector with no type
testfile.cpp:12: error: expected ; before < token
```

The same error message may be seen if the vector library is included but the namespace std is not used.

Try 5.2.1: Internet search for clues of error message cause.

Do an Internet search by copy-pasting the following error message (from the second line of the above figure): error: ISO C++ forbids declaration of vector with no type

Examine at least the first 3 search results, focusing on replies, to find clues to the error message cause.

Participation Activity

5.2.5: Vector definition and use.

#	Question	Your answer
1	Define a vector named testVctr that stores 10 items of type int.	
2	Assign the value stored at index 8 of vector testVctr to a variable x.	
3	Assign the value 555 to the element at index 2 of vector testVctr.	
4	Define an int vector testVctr with 50 elements each initialized to 10.	

C Challenge Activity

5.2.1: Enter the output for the vector.

Start

Enter the output of the following program.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 3;
   vector<int> userVals(NUM_ELEMENTS);
   int i = 0;

   userVals.at(0) = 1;
   userVals.at(1) = 4;
   userVals.at(2) = 7;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals.at(i) << endl;
   }

   return 0;
}
```

```
1
4
7
```

1	2	3	4	5

Check Next

C Challenge Activity

5.2.2: Printing vector elements.

Write three statements to print the first three elements of vector runTimes. Follow each with

```
800
775
790
```

Note: These activities may test code with different test values. This activity will perform two second with a 4-element vector (vector<int> runTimes(4)). See How to Use zyBooks.

Also note: If the submitted code tries to access an invalid vector element, such as runTime the test may crash and report "Program end never reached", in which case the system doe

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     vector<int> runTimes(5);
7
8     // Populate vector
9     runTimes.at(0) = 800;
10    runTimes.at(1) = 775;
11    runTimes.at(2) = 790;
12    runTimes.at(3) = 805;
13    runTimes.at(4) = 808;
14
15    /* Your solution goes here  */
16
17    return 0;
18 }
```

Run

Challenge Activity

5.2.3: Printing vector elements with a for loop.

Write a for loop to print all NUM_VALS elements of vector courseGrades, following each w
newline. Ex: If courseGrades = {7, 9, 11, 10}, print:

```
7 9 11 10
10 11 9 7
```

Hint: Use two for loops. Second loop starts with i = NUM_VALS - 1.

Note: These activities may test code with different test values. This activity will perform two
second with a 2-element vector (vector<int> courseGrades(2)). See How to Use zyBooks.

Also note: If the submitted code tries to access an invalid vector element, such as courseG
Or the test may crash and report "Program end never reached", in which case the system

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_VALS = 4;
   vector<int> courseGrades(NUM_VALS);
   int i = 0;

   courseGrades.at(0) = 7;
   courseGrades.at(1) = 9;
   courseGrades.at(2) = 11;
   courseGrades.at(3) = 10;

   /* Your solution goes here  */

   return 0;
}
```

Run

Section 5.3 - Array/vector iteration drill

The following activities can help one become comfortable with iterating through arrays or vectors, before learning to code such iteration.

Participation Activity

5.3.1: Find the maximum value in the array.

Click "Store value" if a new maximum value is seen.

Start

X	X	X	X	X	X	X

Stored value

-1

Next value Store value

Time - Best time - Clear best

Participation Activity

5.3.2: Negative value counting in array.

Click "Increment" if a negative value is seen.

Start

X	X	X	X	X	X	X

Counter

0

Next value Increment

Time - Best time - Clear best

Participation Activity

5.3.3: Array sorting largest value.

Move the largest value to the right-most position. Click "Swap values" if the larger of the two current values is on the left.

Start

X	X	X	X	X	X	X

Next value Swap values

Time - Best time - Clear best

Section 5.4 - Iterating through vectors

Iterating through vectors using loops is commonplace and is an important programming skill to master.

Because vector indices are numbered 0 to N - 1 rather than 1 to N, programmers commonly use this for loop structure:

Figure 5.4.1: Common for loop structure for iterating through a vector.

```
// Iterating through myVector
for (i = 0; i < numElements; ++i) {
   // Loop body accessing myVector.at(i)
}
```

Note that index variable i is initialized to 0, and the loop expression is i < N rather than i <= N. If N were 5, the loop's iterations would set i to 0, 1, 2, 3, and 4, for a total of 5 iterations. The benefit of the loop structure is that each vector element is accessed as vctr.at(i) rather than the more complex vctr.at(i - 1).

Participation Activity

5.4.1: Iterating through a vector.

#	Question	Your answer

1	Complete the code to print all items for the given vector, using the above common loop structure.
	`vector<int> daysList(365);`
		`for (i = 0; ; ++i) {`
		`  cout << daysList.at(i) << endl;`
		`}`

2	Given that this loop iterates over all items of the vector, how many items are in the vector?
	`for (i = 0; i < 99; ++i) {`
	`  cout << someVector.at(i) << endl;`
	`}`	

Programs commonly iterate through vectors to determine some quantity about the vector's items. The following example computes the sum of a vector's element values:

Figure 5.4.2: Iterating through a vector example: Program that finds the sum of a vector's elements.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 8;          // Number of elements in vector
   vector<int> userVals(NUM_ELEMENTS); // User values
   int i = 0;                           // Loop index
   int sumVal = 0;                      // For computing sum

   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals.at(i);
   }

   // Determine sum
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      sumVal = sumVal + userVals.at(i);
   }
   cout << "Sum: " << sumVal << endl;

   return 0;
}
```

```
Enter 8 ir
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: -1
Sum: 920
```

Iterating through a vector for various purposes is an important programming skill to master. The following is another example program, which determines the maximum value in a user-entered list.

Figure 5.4.3: Iterating through a vector example: Program that finds the max item.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_VALS = 8;          // Number of elements in vector
   vector<int> userVals(NUM_VALS); // User values
   int i = 0;                       // Loop index
   int maxVal = 0;                  // Computed max

   cout << "Enter " << NUM_VALS << " integer numbers..." << endl;
   for (i = 0; i < NUM_VALS; ++i) {
      cout << "Value: ";
      cin >> userVals.at(i);
   }

   // Determine largest (max) number
   maxVal = userVals.at(0);         // Largest so far
   for (i = 0; i < NUM_VALS; ++i) {
      if (userVals.at(i) > maxVal) {
         maxVal = userVals.at(i);
      }
   }
   cout << "Max: " << maxVal << endl;

   return 0;
}
```

```
Enter 8 intege
Value: 3
Value: 5
Value: 23
Value: -1
Value: 456
Value: 1
Value: 6
Value: 83
Max: 456

...

Enter 8 intege
Value: -5
Value: -10
Value: -44
Value: -2
Value: -27
Value: -9
Value: -27
Value: -9
Max: -2
```

If the user enters numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output "max: 55". The bottom part of the code iterates through the vector to determine the maximum value. The main idea of that code is to use a variable maxVal to store the largest value seen thus far as the program iterates through the vector. During each iteration, if the vector's current element value is larger than the max seen thus far, the program writes that value to maxVal (akin to being able to carry only one item as you walk through a store, replacing the current item by a better item whenever you see one).

Before entering the loop, maxVal must be initialized to some value because maxVal will be compared with each vector element's value. A logical error would be to initialize maxVal to 0, because 0 is not in fact the largest value seen so far, and would result in incorrect output (of 0) if the user entered all negative numbers. Instead, the program peeks at a vector element (in this case the first element, though any element could be used) and initializes maxVal to that element's value.

Participation Activity

5.4.2: Iterating through vectors.

Complete the code provided to achieve the desired goal.

#	Question	Your answer

1	Find the minimum element value in vector `valsVctr`.	`tempVal = valsVctr.at(0);`
		`for (i = 0; i < NUM_VALS; ++i) {`
		`  if (valsVctr.at(i) <` [ ] `)`
		`  tempVal= valsVctr.at(i);`
		`  }`
		`}`

2	Find the sum of all elements in vector `valsVctr`.	`valSum =` [ ] `;`
		`for (i = 0; i < NUM_VALS; ++i) {`
		`  valSum += valsVctr.at(i);`
		`}`

3	Count the number of negative-valued elements in vector `valsVctr`.	`numNeg = 0;`
		`for (i = 0; i < NUM_VALS; ++i) {`
		`  if (valsVctr.at(i) < 0) {`
		`  numNeg =` [ ] `;`
		`  }`
		`}`

Participation Activity

5.4.3: Computing the average of a vector's element values.

Complete the code to compute the average of the vector's element values. The result should be 16.

```
1
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7     const int VALS_SIZE = 6;
8     vector<int> valsVctr(VALS_SIZE);
9     int i = 0;
10    int sumVal = 0;
11    int avgVal = 0;
12
13    valsVctr.at(0) = 30;
14    valsVctr.at(1) = 20;
15    valsVctr.at(2) = 20;
16    valsVctr.at(3) = 15;
17    valsVctr.at(4) = 5;
18    valsVctr.at(5) = 10;
19
20    sumVal = 0;
21    /* FIXME: Write for loop to iterate through vector */
22
```

Run

A common error is to try to access a vector with an index that is out of the vector's index range, e.g., to try to access v.at(8) when v's valid indices are 0-7. Care should be taken whenever a user enters a number that is then used as a vector index, and when using a loop index as a vector index also, to ensure the array index is within a vector's valid index range. Accessing an index that is out of range causes the program to automatically abort execution, typically with an error message being automatically printed. For example, for the definition `vector nums(8)`, accessing nums.at(8), or nums.at(i) where i is 8, yields the following error message when running the program compiled with g++:

Figure 5.4.4: Sample error message when running a program that tries to access an out of range index of a vector.

```
terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check
Abort
```

Participation Activity

5.4.4: Loop expressions.

Run the program, which prints the contents of the vals vector. Modify the program's loop expression to be `i <= VALS_SIZE` rather than `i < VALS_SIZE`, and observe that the program aborts.

```
1
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7     const int VALS_SIZE = 6;
8     vector<int> myVals(VALS_SIZE);
9     int i = 0;
10
11    myVals.at(0) = 30;
12    myVals.at(1) = 20;
13    myVals.at(2) = 20;
14    myVals.at(3) = 15;
15    myVals.at(4) = 5;
16    myVals.at(5) = 10;
17
18    for( i=0; i < VALS_SIZE; ++i){
19       cout << "myVals.at(" << i << ") = " << myVals.at(i) <
20    }
21
22
```

Run

Participation Activity

5.4.5: Iterating through a vector.

Given the following code snippet,

```
const int NUM_ELEMENTS = 5;
vector<int> myVctr(NUM_ELEMENTS);
int i = 0;
```

#	Question	Your answer
1	The normal for loop structure iterates as long as: `i <= NUM_ELEMENTS`	True
		False
2	To compute the sum of elements, a reasonable statement preceding the for loop is: `sumVal = 0;`	True
		False
3	To find the maximum element value, a reasonable statement preceding the for loop is: `maxVal = 0;`	True
		False

Challenge Activity

5.4.1: Enter the output for the vector.

Start

Enter the output of the following program.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 3;
   vector<int> userVals(NUM_ELEMENTS);
   int i = 0;

   userVals.at(0) = 2;
   userVals.at(1) = 8;
   userVals.at(2) = 6;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals.at(i) << endl;
   }

   return 0;
}
```

```
2
8
6
```

1	2	3	4	5	6

Check Next

Challenge Activity

5.4.2: Finding values in vectors.

Set numMatches to the number of elements in userValues (having NUM_VALS elements) 2}, then numMatches = 3.

```
3  using namespace std;
4
5  int main() {
6     const int NUM_VALS = 4;
7     vector<int> userValues(NUM_VALS);
8     int i = 0;
9     int matchValue = 0;
10    int numMatches = -99; // Assign numMatches with 0 before your for
11
12    userValues.at(0) = 2;
13    userValues.at(1) = 2;
14    userValues.at(2) = 1;
15    userValues.at(3) = 2;
16
17    matchValue = 2;
18
19    /* Your solution goes here  */
20
21    cout << "matchValue: " << matchValue << ", numMatches: " << numMa
22
23    return 0;
24 }
```

Run

Challenge Activity

5.4.3: Populating a vector with a for loop.

Write a for loop to populate vector userGuesses with NUM_GUESSES integers. Read inte
userGuesses is {9, 5, 2}.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int NUM_GUESSES = 3;
7     vector<int> userGuesses(NUM_GUESSES);
8     int i = 0;
9
10    /* Your solution goes here  */
11
12    return 0;
13 }
```

Run

Challenge Activity

5.4.4: Vector iteration: Sum of excess.

Vector testGrades contains NUM_VALS test scores. Write a for loop that sets sumExtra to extra credit. Ex: If testGrades = {101, 83, 107, 90}, then sumExtra = 8, because 1 + 0 + 7 +

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_VALS = 4;
   vector<int> testGrades(NUM_VALS);
   int i = 0;
   int sumExtra = -9999; // Assign sumExtra with 0 before your for l

   testGrades.at(0) = 101;
   testGrades.at(1) = 83;
   testGrades.at(2) = 107;
   testGrades.at(3) = 90;

   /* Your solution goes here  */

   cout << "sumExtra: " << sumExtra << endl;
   return 0;
}
```

Run

Challenge Activity

5.4.5: Printing vector elements separated by commas.

Write a for loop to print all NUM_VALS elements of vector hourlyTemp. Separate elements

```
90, 92, 94, 95
```

Note that the last element is not followed by a comma, space, or newline.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int NUM_VALS = 4;
7     vector<int> hourlyTemp(NUM_VALS);
8     int i = 0;
9
10    hourlyTemp.at(0) = 90;
11    hourlyTemp.at(1) = 92;
12    hourlyTemp.at(2) = 94;
13    hourlyTemp.at(3) = 95;
14
15    /* Your solution goes here  */
16
17    cout << endl;
18
19    return 0;
20 }
```

Run

Section 5.5 - Multiple vectors

Programmers commonly use multiple same-sized vectors to store related lists. For example, the following program maintains a list of country names, and another list indicating average minutes of TV watched per day in each corresponding country.

Figure 5.5.1: Multiple vector example: TV watching time program.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
   // Source: www.statista.com, 2010
   const int NUM_CTRY = 5;              // Num countries supported
   vector<string> ctryNames(NUM_CTRY); // Country names
   vector<int>    ctryMins(NUM_CTRY);  // Mins TV watched daily
   string userCountry;                 // User defined country
   bool foundCountry = false;          // Match to country supported
   int i = 0;                          // Loop index

   // Define array contents
   ctryNames.at(0) = "China";
   ctryMins.at(0) = 158;

   ctryNames.at(1) = "India";
   ctryMins.at(1) = 119;

   ctryNames.at(2) = "Russia";
   ctryMins.at(2) = 226;

   ctryNames.at(3) = "UK";
   ctryMins.at(3) = 242;

   ctryNames.at(4) = "USA";
   ctryMins.at(4) = 283;

   // Prompt user for country name
   cout << "Enter country name: ";
   cin >> userCountry;

   // Find country's index and avgerage TV time
   foundCountry = false;
   for (i = 0; i < NUM_CTRY; ++i) {
      if (ctryNames.at(i) == userCountry) {
         foundCountry = true;
         cout << "People in " << userCountry << " watch ";
         cout << ctryMins.at(i) << " mins of TV daily." << endl;
         break;
      }
   }
   if (!foundCountry) {
      cout << "Country not found; try again." << endl;
   }

   return 0;
}
```

```
Enter countr
People in US

...

Enter countr
People in Ch

...

Enter countr
Country not
```

The statement `if (ctryNames.at(i) == userCountry)` compares the current ctryNames element with the user-entered country name; if the names match, the program prints the ctryMins element having that same index.

Once the country is found, there's no need to keep iterating through the vector. The code uses a break statement to exit the for loop. Another option is to update the loop expression to `((i < NUM_CTRY) && (!foundCountry))`. The output is the same, but the break approach may be clearer, and faster for large lists.

The program's numbers aren't made up, by the way: Americans watch nearly 5 hours of TV per day on average.

Participation Activity

5.5.1: Multiple vectors.

Consider the above TV watching program involving multiple vectors.

#	Question	Your answer
1	Multiple vectors saved memory over using one larger vector.	True
		False
2	Each vector should be the same data type.	True
		False
3	Each vector should have the same number of elements.	True
		False

Participation Activity

5.5.2: Improve the TV watching time program.

Modify the program such that if a user types a country name that isn't found, print a list of known countries.

```
1
2  #include <iostream>
3  #include <vector>
4  #include <string>
5  using namespace std;
6
7  int main() {
8     // Source: www.statista.com, 2010
9     const int NUM_CTRY = 5;
10    vector<string> ctryNames(NUM_CTRY);// Country names
11    vector<int>    ctryMins(NUM_CTRY); // Mins TV watched da
12    string country;
13    bool found = false;
14    int i = 0;
15
16    ctryNames.at(0) = "China";
17    ctryMins.at(0) = 158;
18
19    ctryNames.at(1) = "India";
20    ctryMins.at(1) = 119;
21
22
```

USA

Run

Challenge Activity

5.5.1: Printing the sum of two vector elements.

Add each element in origList with the corresponding value in offsetAmount. Print each sum
{5, 7, 3, 0}, print:

```
45 57 63 70
```

```
5  int main() {
6     const int NUM_VALS = 4;
7     vector<int> origList(NUM_VALS);
8     vector<int> offsetAmount(NUM_VALS);
9     int i = 0;
10
11    origList.at(0) = 40;
12    origList.at(1) = 50;
13    origList.at(2) = 60;
14    origList.at(3) = 70;
15
16    offsetAmount.at(0) = 5;
17    offsetAmount.at(1) = 7;
18    offsetAmount.at(2) = 3;
19    offsetAmount.at(3) = 0;
20
21    /* Your solution goes here  */
22
23    cout << endl;
24
25    return 0;
26 }
```

Run

Challenge Activity

5.5.2: Multiple vectors: Key and value.

For any element in keysList with a value greater than 100, print the corresponding value in itemsList = {10, 20, 30, 40}, print:

```
20 30
```

Since keysList[1] and keysList[2] have values greater than 100, the value of itemsList[1] ar

```
5  int main() {
6     const int SIZE_LIST = 4;
7     vector<int> keysList(SIZE_LIST);
8     vector<int> itemsList(SIZE_LIST);
9     int i = 0;
10
11    keysList.at(0) = 42;
12    keysList.at(1) = 105;
13    keysList.at(2) = 101;
14    keysList.at(3) = 100;
15
16    itemsList.at(0) = 10;
17    itemsList.at(1) = 20;
18    itemsList.at(2) = 30;
19    itemsList.at(3) = 40;
20
21    /* Your solution goes here  */
22
23    cout << endl;
24
25    return 0;
26 }
```

Run

Section 5.6 - Vector resize

Commonly, the size of a list of items is not known during a program's compile time. Thus, a vector's size need not be specified in the vector's definition. Instead, a vector's size can be set or changed while a program is executing using ***vctr.resize(N)***.

Figure 5.6.1: Vector resize.

```
vctr.resize(N);  // Allocates N elements for vector vctr.
```

The notation vctr.resize() indicates that resize() is a function that operates on the vector; sections on C++ *classes* discuss such notation further.

The following program asks a user to indicate the number of values the user will enter, and allocates that number of elements for a vector.

Figure 5.6.2: Resizing a vector based on user input.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> userVals; // No elements yet
   int numVals = 0;
   int i = 0;

   cout << "Enter number of integer values: ";
   cin >> numVals;

   userVals.resize(numVals); // Allocate elements

   cout << "Enter " << numVals << " integer values..." << endl;
   for (i = 0; i < numVals; ++i) {
      cout << "Value: ";
      cin >> userVals.at(i);
   }
   cout << "You entered: ";
   for (i = 0; i < numVals; ++i) {
      cout << userVals.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```

```
Enter number of i
Enter 7 integer v
Value: -5
Value: -99
Value: 0
Value: 13
Value: 7
Value: -22
Value: 1
You entered: -5 -
```

resize() can be called again. If the new size is larger, resize() adds elements at the end. If smaller, resize() deletes elements from the end. If vctr has size 3 (elements 0, 1, 2), vctr.resize(2) would delete element 2, leaving elements 0 and 1; a subsequent access to vctr.at(2) would result in an error.

A vector's current size can be accessed using the function ***vctr.size()***. Thus, the above loop statements could be replaced by:

```
for (i = 0; i < values.size(); ++i) {
   ...
}
```

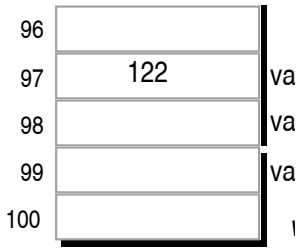
The following animation illustrates vector resize. When initially defined, a vector values has a size of 0. Accessing any element, such as values.at(0), would result in a runtime error. The values.resize(3) call allocates 3 elements (0, 1, 2). As before, an access to an out-of-range index like values.at(3) causes a runtime error.

Participation Activity

5.6.1: Vector resize.

Start animation

```
vector<int> values;

// values.at(0) = 122; Would be out-of-range error

values.resize(3);

values.at(0) = 122;
values.at(3) = 555;  // Index 3 out-of-range
```




```
terminate called after throwing an instance of 'std::out_of_range'
what(): vector::_M_range_check
Abort
```

Participation Activity

5.6.2: Vector resize and size functions..

Given the vector definition: `vector<int> agesVctr;`

#	Question	Your answer
1	Immediately after the definition, agesVctr has only 1 element.	True
		False
2	agesVctr.size(4) allocates 4 elements for agesVctr.	True
		False
3	Given agesVctr has 3 elements, agesVctr.resize(4) adds 4 more elements, totalling 7 elements.	True
		False
4	Given agesVctr has 3 elements with values 22, 18, and 19, agesVctr.resize(2) changes agesVctr to have 2 elements with values 22 and 18.	True
		False
5	After agesVctr.resize(5) and agesVctr.at(0) = 99, agesVctr.size() evaluates to 1.	True
		False

Challenge Activity

5.6.1: Determining the size of a vector.

Assign the size of vector sensorReadings to currentSize.

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     vector<int> sensorReadings(4);
7     int currentSize = 0;
8
9     sensorReadings.resize(10);
10
11    /* Your solution goes here  */
12
13    cout << "Number of elements: " << currentSize << endl;
14
15    return 0;
16 }
```

Run

Challenge Activity

5.6.2: Resizing a vector.

Resize vector countDown to have newSize elements. Populate the vector with integers {ne
1}, and the sample program outputs:

```
3 2 1 Go!
```

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> countDown(0);
   int newSize = 0;
   int i = 0;

   newSize = 3;

   /* Your solution goes here  */

   for (i = 0; i < newSize; ++i) {
      cout << countDown.at(i) << " ";
   }
   cout << "Go!" << endl;

   return 0;
}
```

Run

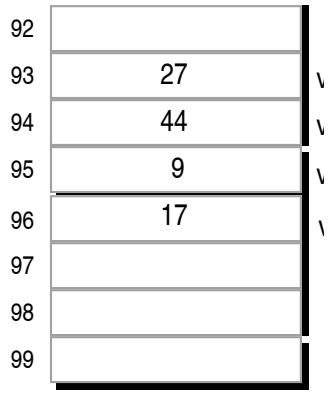
Section 5.7 - Vector push_back

A programmer commonly wishes to append a new element to the end of an existing vector. ***vctr.push_back(value)*** creates a new element at the end of vector and assigns the given value to that element, thus increasing the vector's size by 1.

Participation Activity

5.7.1: The vector push_back() function.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   unsigned int i = 0;
   vector<int> vctr;

   cout << "Size before: " << vctr.size();
   vctr.push_back(27);
   vctr.push_back(44);
   vctr.push_back(9);
   vctr.push_back(17);
   cout << ", after: " << vctr.size() << endl;

   cout << "Contents:" << endl;
   for (i = 0; i < vctr.size(); ++i) {
      cout << " " << vctr.at(i) << endl;
   }

   return 0;
}
```




```
Size before: 0, afte
Contents:
 27
 44
 9
 17
```

The notation myVector.push_back() indicates that push_back() is a function that operates on myVector; sections on "classes" discuss such functions further.

The following summarizes three functions related to the back of a vector.

Table 5.7.1: Functions dealing with a vector's back.

Shown for vector<int>, but applies to other types.

push_back()	`void push_back(const int newVal);`
	
	Append new element having value newVal.
		`// playersList initially !`
		`playersList.push_back(77);`
		`// playersList is now !`

back()	`int back();`
	
	Returns value of vector's last element. Vector is unchanged.
		`// playersList initially !`
		`cout << playersList.back(`
		`// playersList is still !`

pop_back()	`void pop_back();`
	
	Removes the last element.
		`// playersList is 55, 99,`
		`playersList.pop_back();`
		`// playersList now 55, 99`
		
		`cout << playersList.back(`
		`playersList.pop_back();`
		`// Prints 99. playersList`
		
		`cout << playersList.pop_ba`

Below is a simple grocery list example. The program defines a vector groceryList, which is initially empty. As the user enters grocery items one at a time, the program uses push_back() to append the items to the list. When done, the user can go shopping, and is presented one list item at a time (which the user presumably finds and places in a shopping cart). The program uses back() to get each item from the list and pop_back() to remove the item from the list. When the list is empty, shopping is finished. Note that because the program removes items from the end of the list, the items are presented in reverse order.

Figure 5.7.1: Using push_back(), back(), and pop_back(): A grocery list example.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

int main() {
   vector<string> groceryList; // Vector storing shopping list
   string groceryItem;         // Individual grocery items
   string userCmd;             // User input

   // Prompt user to populate shopping list
   cout << "Enter grocery items or type done." << endl;
   cin >> groceryItem;
   while (groceryItem != "done") {
      groceryList.push_back(groceryItem);
      cin >> groceryItem;
   }

   // Display shopping list
   cout << endl << "Enter any key for next item." << endl;
   while (groceryList.size() > 0) {
      groceryItem = groceryList.back();
      groceryList.pop_back();
      cout << groceryItem << "   ";
      cin >> userCmd;
   }
   cout << endl << "Done shopping." << endl;

   return 0;
}
```

```
Enter grocery iten
Oranges
Apples
Bread
Juice
done

Enter any key for
Juice   a
Bread   a
Apples   a
Oranges   a

Done shopping.
```

Participation Activity

5.7.2: Vector push_back(), back(), and pop_back() functions.

#	Question	Your answer
1	If vector `itemPrices` has two elements with values 45, 48, what does itemPrices.size() return?	
2	If itemPrices has element values 45, 48, then after itemPrices.push_back(38), what are itemPrices' element values? Type answer as: 50, 60, 70	
3	If itemPrices has elements 45, 22, 38, what does price = itemPrices.pop_back() assign to price? Type error if appropriate.	
4	If itemPrices has elements 45, 22, 38, what does price = itemPrices.back() assign to price? Type error if appropriate.	
5	If itemPrices has elements 45, 22, 38, what is the vector after itemPrices.back() is called? Type answer as: 50, 60, 70	
6	If itemPrices has elements 45, 22, 38, then after itemPrices.pop_back(), what does itemPrices.at(2) return? Type error if appropriate.	

C Challenge Activity

5.7.1: Appending a new element to a vector.

Append newValue to the end of vector tempReadings. Ex: If newValue = 67, then tempRea

```
3  using namespace std;
4
5  int main() {
6     vector<int> tempReadings(3);
7     int newValue = 0;
8     unsigned int i = 0;
9
10    tempReadings.at(0) = 53;
11    tempReadings.at(1) = 57;
12    tempReadings.at(2) = 60;
13
14    newValue = 67;
15
16    /* Your solution goes here  */
17
18    for (i = 0; i < tempReadings.size(); ++i) {
19       cout << tempReadings.at(i) << " ";
20    }
21    cout << endl;
22
23    return 0;
24 }
```

Run

Challenge Activity

5.7.2: Removing an element from the end of a vector.

Remove the last element from vector ticketList.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   vector<int> ticketList(3);
   unsigned int i = 0;

   ticketList.at(0) = 5;
   ticketList.at(1) = 100;
   ticketList.at(2) = 12;

   /* Your solution goes here  */

   for (i = 0; i < ticketList.size(); ++i) {
      cout << ticketList.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```

Run

C Challenge Activity

5.7.3: Reading the vector's last element.

Write a statement to print "Last mpg reading: " followed by the value of mpgTracker's last e

```
Last mpg reading: 20
```

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     vector<int> mpgTracker(3);
7
8     mpgTracker.at(0) = 17;
9     mpgTracker.at(1) = 19;
10    mpgTracker.at(2) = 20;
11
12    /* Your solution goes here  */
13
14    return 0;
15 }
```

Run

Section 5.8 - Loop-modifying or copying/comparing vectors

Sometimes a program changes some elements' values or moves elements while iterating through a vector. The following uses a loop to convert any negative vector element values to 0.

Figure 5.8.1: Modifying a vector during iteration example: Converting negatives to 0.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 5;          // Number of elements
   vector<int> userVals(NUM_ELEMENTS); // User values
   int i = 0;                           // Loop index

   // Prompt user to populate vector
   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals.at(i);
   }

   // Convert negatives to 0
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      if (userVals.at(i) < 0) {
         userVals.at(i) = 0;
      }
   }

   // Print numbers
   cout << "New values:";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << userVals.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Enter 5 inte
Value: 67
Value: -5
Value: -99
Value: 4
Value: 22
New values:
```

Participation Activity

5.8.1: Modifying a vector in a loop.

What is the resulting vector contents, assuming each question starts with a vector of size 4 having contents -55, -1, 0, 9?

#	Question	Your answer

1	`for (i = 0; i < 4; ++i) {`
	`  itemsList.at(i) = i;`
	`}`
		-54, 0, 1, 10

		0, 1, 2, 3
		1, 2, 3, 4

2	`for (i = 0; i < 4; ++i) {`
	`  if (itemsList.at(i) < 0) {`
	`  itemsList.at(i) = itemsList.at(i) * -1;`
	`  }`
	`}`
		-55, -1, 0, -9
		55, 1, 0, -9
		55, 1, 0, 9

3	`for (i = 0; i < 4; ++i) {`
	`  itemsList.at(i) = itemsList.at(i+1);`
	`}`
		-1, 0, 9, 0
		0, -55, -1, 0
		Error (program aborts)

4	`for (i = 0; i < 3 ; ++i) {`
	`  itemsList.at(i) = itemsList.at(i+1);`
	`}`
		-1, 0, 9, 9
		Error (program aborts)
		-1, 0, 9, 0

5	`for (i = 0; i < 3 ; ++i) {`
	`  itemsList.at(i+1) = itemsList.at(i);`
	`}`
		-55, -55, -55, -55
		0, -55, -1, 0
		Error (program aborts)

Participation Activity

5.8.2: Modifying a vector during iteration example: Doubling element values.

Complete the following program to double each number in the vector.

```
1
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7     const int NUM_ELEMENTS = 5;          // Number of element
8     vector<int> userVals(NUM_ELEMENTS); // User values
9     int i = 0;                           // Loop index
10
11    // Prompt user to populate vector
12    cout << "Enter " << NUM_ELEMENTS << " integer values..."
13    for (i = 0; i < NUM_ELEMENTS; ++i) {
14       cout << "Value: " << endl;
15       cin >> userVals.at(i);
16    }
17
18    // Convert negatives to 0
19    for (i = 0; i < NUM_ELEMENTS; ++i) {
20       if (userVals.at(i) < 0) {
21          userVals.at(i) = 0;
22
```

67 -5 -

Run

In C++, the = operator conveniently performs an element-by-element copy of a vector, called a ***vector copy operation***. The operation vctrB = vctrA resizes vctrB to vctrA's size, appending or deleting elements as needed. vctrB commonly has a size of 0 before the operation.

Figure 5.8.2: Using = to copy a vector: Original and sale prices.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int   NUM_ELEMENTS = 4;          // Number of elements
   vector<int> origPrices(NUM_ELEMENTS); // Original prices
   vector<int> salePrices(NUM_ELEMENTS); // Sale prices
   int i = 0;                            // Loop index

   // Assign original prices
   origPrices.at(0) = 10;
   origPrices.at(1) = 20;
   origPrices.at(2) = 30;
   origPrices.at(3) = 40;

   // Copy original prices to sales prices
   salePrices = origPrices;

   // Update salePrices. Note: does not affect origPrices
   salePrices.at(2) = 27;
   salePrices.at(3) = 35;

   // Output original and sale prices
   cout << "Original prices: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << origPrices.at(i);
   }
   cout << endl;

   cout << "Sale prices:     ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << salePrices.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Original prices:  10 20 30 40
Sale prices:      10 20 27 35
```

Participation Activity

5.8.3: Vector copy operation.

Given: vector<int> userVals(4), with element values 44, 55, 66, 77.
Given: vector<int> newVals.

#	Question	Your answer
1	What is newVals after: `newVals = userVals;` Type answer as: 10, 20, 30, 40 If appropriate type: Error	

2	What is newVals after:
	`newVals = userVals;`
	`userVals.at(0) = 33;`
	Type answer as: 10, 20, 30, 40
	If appropriate type: Error	

3	Given: vector<int> otherVals(9).
	What *size* is newVals after:
	`newVals = userVals;`
	`...`
	`newVals = otherVals;`
	If appropriate type: Error	

In C++, the == operator conveniently compares vectors element-by-element, called a ***vector equality operation***, with vctrA == vctrB evaluating to true if the vectors are the same size AND each element pair is equal.

Participation Activity

5.8.4: Vector copying.

Assume vectors have been defined as follows and have been initialized as indicated in the comments:

```
vector<int> vctrX(2); // {3,4}
vector<int> vctrY(5); // {3,4,0,7,8}
vector<int> vctrZ(5); // {3,4,0,6,8}
```

#	Question	Your answer
1	(vctrX == vctrY) will evaluate to:	True
		False
2	Given: vctrX = vctrY; (vctrX == vctrY) will evaluate to:	True
		False
3	(vctrZ == vctrY) will evaluate to:	True
		False
4	(vctrZ.size() == vctrY.size()) will evaluate to:	True
		False

Challenge Activity

5.8.1: Decrement vector elements.

Write a loop that subtracts 1 from each element in lowerScores. If the element was already becomes {4, 0, 1, 0}.

```
#include <vector>
using namespace std;

int main() {
   const int SCORES_SIZE = 4;
   vector<int> lowerScores(SCORES_SIZE);
   int i = 0;

   lowerScores.at(0) = 5;
   lowerScores.at(1) = 0;
   lowerScores.at(2) = 2;
   lowerScores.at(3) = -3;

   /* Your solution goes here  */

   for (i = 0; i < SCORES_SIZE; ++i) {
      cout << lowerScores.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```

Run

Challenge Activity

5.8.2: Copy and modify vector elements.

Write a loop that sets newScores to oldScores shifted once left, with element 0 copied to th
10}.

```
3  using namespace std;
4
5  int main() {
6     const int SCORES_SIZE = 4;
7     vector<int> oldScores(SCORES_SIZE);
8     vector<int> newScores(SCORES_SIZE);
9     int i = 0;
10
11    oldScores.at(0) = 10;
12    oldScores.at(1) = 20;
13    oldScores.at(2) = 30;
14    oldScores.at(3) = 40;
15
16    /* Your solution goes here  */
17
18    for (i = 0; i < SCORES_SIZE; ++i) {
19       cout << newScores.at(i) << " ";
20    }
21    cout << endl;
22
23    return 0;
24 }
```

Run

Challenge Activity

5.8.3: Modify vector elements using other elements.

Write a loop that sets each vector element to the sum of itself and the next element, excep beyond the last element. Ex:

```
Initial scores:        10, 20, 30, 40
Scores after the loop: 30, 50, 70, 40
```

The first element is 30 or 10 + 20, the second element is 50 or 20 + 30, and the third eleme

```
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int SCORES_SIZE = 4;
7     vector<int> bonusScores(SCORES_SIZE);
8     int i = 0;
9
10    bonusScores.at(0) = 10;
11    bonusScores.at(1) = 20;
12    bonusScores.at(2) = 30;
13    bonusScores.at(3) = 40;
14
15    /* Your solution goes here  */
16
17    for (i = 0; i < SCORES_SIZE; ++i) {
18       cout << bonusScores.at(i) << " ";
19    }
20    cout << endl;
21
22    return 0;
23 }
```

Run

Challenge Activity

5.8.4: Modify a vector's elements.

Double any element's value that is less than minVal. Ex: If minVal = 10, then dataPoints =

```
int main() {
   const int NUM_POINTS = 4;
   vector<int> dataPoints(NUM_POINTS);
   int minVal = 0;
   int i = 0;

   dataPoints.at(0) = 2;
   dataPoints.at(1) = 12;
   dataPoints.at(2) = 9;
   dataPoints.at(3) = 20;

   minVal = 10;

   /* Your solution goes here  */

   for (i = 0; i < NUM_POINTS; ++i) {
      cout << dataPoints.at(i) << " " ;
   }
   cout << endl;

   return 0;
}
```

Run

Challenge Activity

5.8.5: Comparing and copying vectors.

If the vector oldData is the same as the vector newData, print "Data matches!" ended with a
12, 18} and newData = {25, 27, 29, 23}, then oldData becomes {25, 27, 29, 23}.

```
   vector<int> newData(4);
   unsigned int i = 0;

   oldData.at(0) = 10;
   oldData.at(1) = 12;
   oldData.at(2) = 18;

   newData.at(0) = 25;
   newData.at(1) = 27;
   newData.at(2) = 29;
   newData.at(3) = 23;

   /* Your solution goes here  */

  for (i = 0; i < oldData.size(); ++i) {
      cout << oldData.at(i) << " ";
  }
   cout << endl;

   return 0;

}
```

Run

Section 5.9 - Swapping two variables

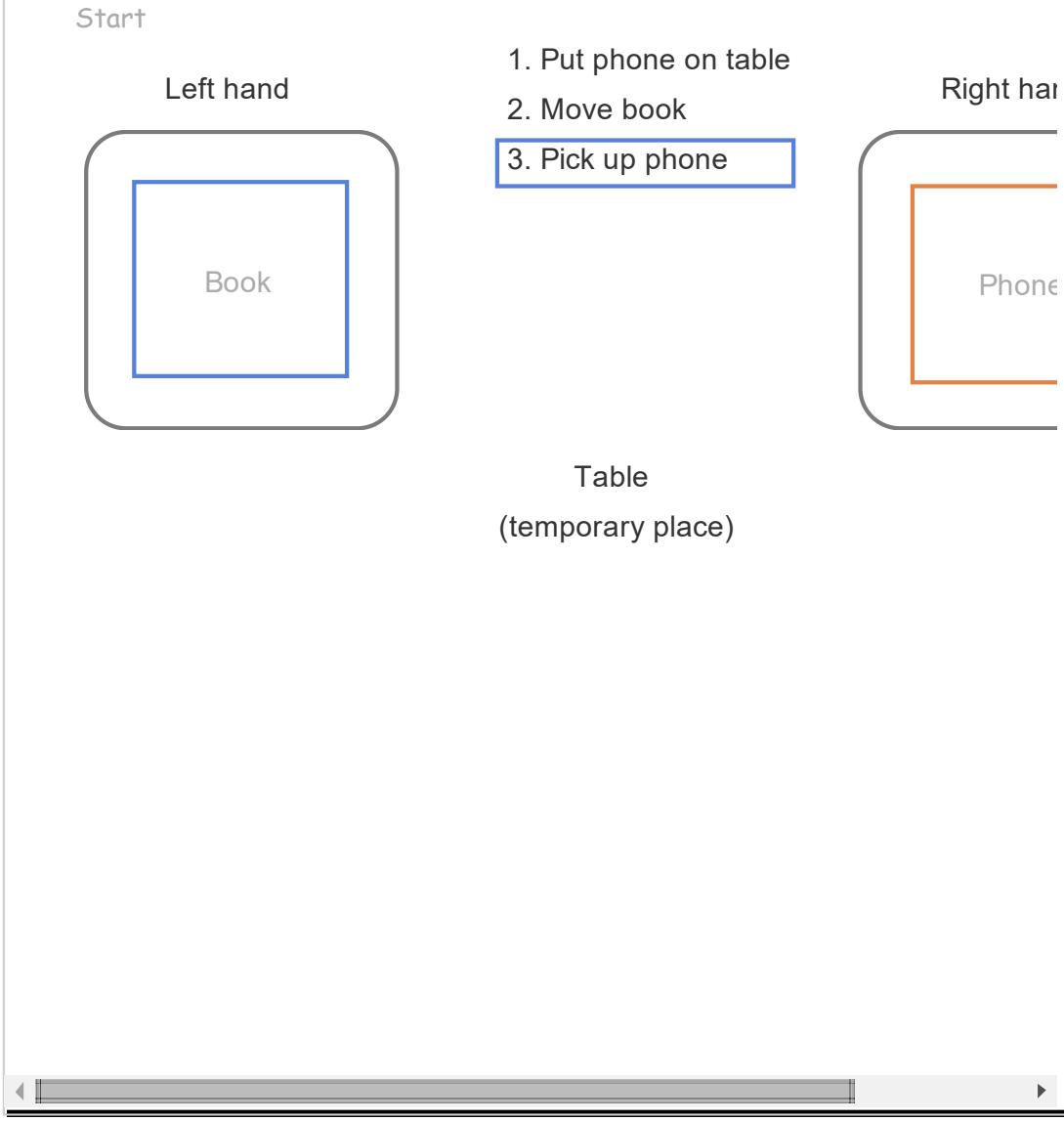
Note_language_neutral2

Sometimes a program must swap values among two variables. ***Swapping*** two variables x and y means to assign y's value to x, and x's value to y. If x is 33 and y is 55, then after swapping x is 55 and y is 33.

Swapping requires a temporary third variable. To understand the intuition of such temporary storage, consider a person holding a book in one hand and a phone in the other, wishing to swap the items. The person can temporarily place the phone on a table, move the book to the other hand, then pick up the phone.

Participation Activity

5.9.1: Swap idea: Use a temporary location.



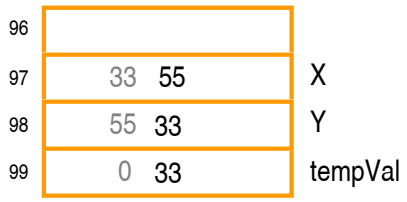
Similarly, swapping two variables uses a third variable to temporarily hold one value while the other value is copied over.

Participation Activity

5.9.2: Swapping two variables requires a third temporary variable.

Start

```
int X = 33;
int Y = 55;
int tempVal = 0;
tempVal = X;
X = Y;
Y = tempVal;

// Print X and Y
```



Store X in tempVal first, swap succeeds

```
X: 55, Y: 33
```

Participation Activity

5.9.3: Swap.

Given x = 22 and y = 99. What are x and y after the given code?

#	Question	Your answer
1	`x = y;` `y = x;`	x is 99 and y is 22. x is 22 and y is 99. x is 99 and y is 99.

2	`x = y;`
	`y = x;`
	`x = y;`
		x is 99 and y is 22.
		x is 99 and y is 99.
		x is 22 and y is 22.

3	`tempVal = x;`
	`x = y;`
	`y = x;`
		x is 99 and y is 22.
		x is 99 and y is 99.

4	`tempVal = x;`
	`x = y;`
	`y = tempVal;`
		x is 99 and y is 22.
		x is 99 and y is 99.

If you have studied arrays or vectors (or other kinds of lists), know that most swaps are actually performed between two list elements. For example, reversing a list with N elements can be achieved by swapping element 1 and N, element 2 and N-1, element 3 and N-2, etc. (stopping at the middle of the list).

Participation Activity

5.9.4: Reversing a list using swaps.

Start

11 77 22 55 33

Participation Activity

5.9.5: Reversing a list using swaps.

#	Question	Your answer
1	Using the above approach, how many swaps are needed to reverse this list: 999 888 777 666 555 444 333 222	

(*Note_language_neutral2) This section is mostly language neutral

---

Section 5.10 - Debugging example: Reversing a vector

A common vector modification is to reverse a vector's elements. One way to accomplish this goal is to perform a series of swaps. For example, starting with a vector of numbers 10 20 30 40 50 60 70 80, we could first swap the first item with the last item, yielding 80 20 30 40 50 60 70 10. We could next swap the second item with the second-to-last item, yielding 80 70 30 40 50 60 20 10. The next swap would yield 80 70 60 40 50 30 20 10, and the last would yield 80 70 60 50 40 30 20 10.

With this basic idea of how to reverse a vector, we can attempt to write a program to carry out such reversal.

Below we develop such a program but we make common mistakes along the way, to aid learning from examples of what not to do.

A first attempt to write a program that reverses a vector appears below.

Figure 5.10.1: First program attempt to reverse vector: Aborts due to invalid access of vector element.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 8;         // Number of elements
   vector<int> revVctr(NUM_ELEMENTS); // User values
   int i = 0;                          // Loop index

   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> revVctr.at(i);
   }

   // Reverse
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      revVctr.at(i) = revVctr.at(NUM_ELEMENTS - i); // Swap
   }

   // Print values
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << revVctr.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Enter 8 inte
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80
libc++abi.dy
of type std:
```

Something went wrong: The program aborted (exited abnormally). The reported message indicates an "out of range" problem related to a vector, meaning the program tried to access a vector element that doesn't exist. Let's try to find the code that caused the problem.

The first and third for loops are fairly standard, so let's initially focus attention on the middle for loop that does the reversing. The swap statement inside that loop is `revVctr.at(i) = revVctr.at(NUM_VALUES - i)`. When i is 0, the statement will execute `revVctr.at(0) = revVctr.at(8)`. However, revVctr has size 8 and thus valid indices are 0..7. revVctr.at(8) does not exist. The program should actually swap elements 0 and 7, then 1 and 6, etc. Thus, let's change the right-side index to `NUM_VALUES - 1 - i`. The revised program is shown below.

Figure 5.10.2: Revised vector reversing program: Doesn't abort, but still a problem.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 8;         // Number of elements
   vector<int> revVctr(NUM_ELEMENTS); // User values
   int i = 0;                          // Loop index

   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> revVctr.at(i);
   }

   // Reverse
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      revVctr.at(i) = revVctr.at(NUM_ELEMENTS - 1 - i); // Swap
   }

   // Print values
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << revVctr.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Enter 8 inte
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:
```

The program didn't abort this time, but the last four elements are wrong. To determine what went wrong, we can manually (i.e., on paper) trace the loop's execution.

- i is 0: revVctr.at(0) = revVctr.at(7). Vector now: 80 20 30 40 50 60 70 80.
- i is 1: revVctr.at(1) = revVctr.at(6). Vector now: 80 70 30 40 50 60 70 80.
- i is 2: revVctr.at(2) = revVctr.at(5). Vector now: 80 70 60 40 50 60 70 80.
- i is 3: revVctr.at(3) = revVctr.at(4). Vector now: 80 70 60 50 50 60 70 80.
- i is 4: revVctr.at(4) = revVctr.at(3). Vector now: 80 70 60 50 50 60 70 80. *Uh-oh, where did 40 go?*

We failed to actually swap the vector elements, instead the code just copies values in one direction. We need to add code to properly swap. We add a variable tmpValue to temporarily hold `revVctr.at(NUM_VALUES - 1 - i)` so we don't lose that element's value.

Figure 5.10.3: Revised vector reversing program with proper swap: Output isn't reversed.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 8;         // Number of elements
   vector<int> revVctr(NUM_ELEMENTS); // User values
   int i = 0;                          // Loop index
   int tmpValue = 0;                   // Placeholder

   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> revVctr.at(i);
   }

   // Reverse
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      tmpValue = revVctr.at(i);  // These 3 statements swap
      revVctr.at(i) = revVctr.at(NUM_ELEMENTS - 1 - i);
      revVctr.at(NUM_ELEMENTS - 1 - i) = tmpValue;
   }

   // Print values
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << revVctr.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Enter 8 inte
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:
```

The new values are not reversed. Again, let's manually trace the loop iterations.

- i is 0: revVctr.at(0) = revVctr.at(7). Vector now: 80 20 30 40 50 60 70 10.
- i is 1: revVctr.at(1) = revVctr.at(6). Vector now: 80 70 30 40 50 60 20 10.
- i is 2: revVctr.at(2) = revVctr.at(5). Vector now: 80 70 60 40 50 30 20 10.
- i is 3: revVctr.at(3) = revVctr.at(4). Vector now: 80 70 60 50 40 30 20 10. *Looks reversed.*
- i is 4: revVctr.at(4) = revVctr.at(3). Vector now: 80 70 60 40 50 30 20 10. *Why are we still swapping?*

Tracing makes clear that the for loop should not iterate over the entire vector. The reversal is completed halfway through the iterations. The solution is to set the loop expression to `i < (NUM_VALUES / 2)`.

Figure 5.10.4: Vector reversal program with correct output.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int NUM_ELEMENTS = 8;         // Number of elements
   vector<int> revVctr(NUM_ELEMENTS); // User values
   int i = 0;                          // Loop index
   int tmpValue = 0;                   // Placeholder

   cout << "Enter " << NUM_ELEMENTS << " integer values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> revVctr.at(i);
   }

   // Reverse
   for (i = 0; i < (NUM_ELEMENTS / 2); ++i) {
      tmpValue = revVctr.at(i);  // These 3 statements swap
      revVctr.at(i) = revVctr.at(NUM_ELEMENTS - 1 - i);
      revVctr.at(NUM_ELEMENTS - 1 - i) = tmpValue;
   }

   // Print values
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << " " << revVctr.at(i);
   }
   cout << endl;

   return 0;
}
```

```
Enter 8 inte
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:
```

We should ensure the program works if the number of elements is odd rather than even. Suppose the vector has 5 elements (0-4) with values 10 20 30 40 50. NUM_VALUES / 2 would be 5 / 2 = 2, meaning the loop expression would be i < 2. The iteration when i is 0 would swap elements 0 and 4 (5-1-0), yielding 50 20 30 40 10. The iteration for i=1 would swap elements 1 and 3, yielding 50 40 30 20 10. The loop would then not execute again because i is 2. So the results are correct for an odd number of elements, because the middle element will just not move.

The mistakes made above are each very common when dealing with loops and vectors, especially for beginning programmers. An incorrect (in this case out-of-range) index, an incorrect swap, and an incorrect loop expression. The lesson is that loops and vectors require attention to detail, greatly aided by manually executing the loop to determine what is happening on each iteration. Ideally, a programmer will take more care when writing the original program, but the above mistakes are quite common.

Participation Activity

5.10.1: Find the error in the vector reversal code.

#	Question

1	for (i = 0; i < numPrices; ++i) {
	tmp = prices.at(i);
	prices.at(i) = prices.at(numPrices - 1 - i);
	prices.at(numPrices - 1 - i) = tmp;
	}

2	for (i = 0; i < (numPrices / 2); ++i) {
	tmp = prices.at(i);
	prices.at(i) = prices.at(numPrices - i);
	prices.at(numPrices - i - 1) = tmp;
	}

3	for (i = 0; i < (numPrices / 2); ++i) {
	tmp = prices.at(i);
	prices.at(numPrices - i - 1) = tmp;
	prices.at(i) = prices.at(numPrices - 1 - i);
	}

Section 5.11 - Arrays vs. vectors

C++ supports two kinds of ordered list types.

- *Arrays:* defined as `int myList[10]`, accessed as `myList[i]`.
- *Vectors:* defined as `vector<int> myList(10)`, accessed as `myList.at(i)`.

Arrays have a simpler syntax than vectors, but vectors are safer to use. Thus, using vectors rather than arrays is good practice.

Vectors are safer because the access v.at(i) is checked during execution to ensure the index is within the vector's valid range. An array access a[i] involves no such check. Such checking is important; trying to access an array with an out-of-range index is a very common error, and one of the hardest errors to debug.

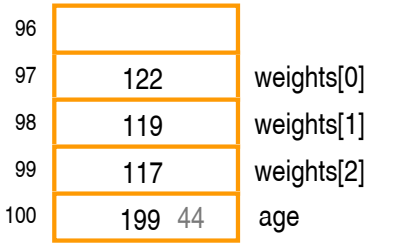
Participation Activity

5.11.1: Assigning to an out-of-range index of an array causes a nasty error that is hard to debug.

Start

```
int weights[3];
int age = 44;

weights[0] = 122;
weights[1] = 119;
weights[2] = 117;
weights[3] = 199; // (Problematic)

// Print age
```

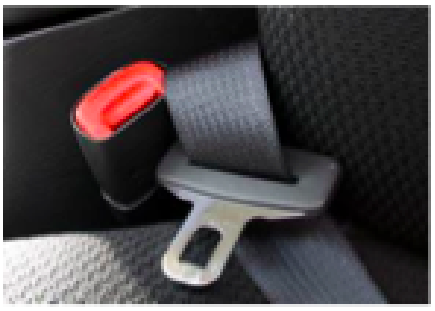
199



int variable age is allocate
location in memory immedi
the array weights.

Assigning to weights[3] wi
the memory location for v

Incorrect value for age is



As shown above, assigning with an out-of-range index can mysteriously change some other variable's value. Debugging such an error can be a nightmare.

Vectors have more advantages, like resizing during runtime, easy insertion of items at the front or rear, determining vector size, etc., discussed later. Arrays have minor benefits that don't really outweigh drawbacks. Like choosing to not wear seatbelts, choosing to not use vectors may be quite risky.

C++ allows vectors to be accessed using brackets [], but brackets involve no range checking, so a good practice is to use .at() to access vector elements.

Participation Activity

5.11.2: Arrays and vectors.

Given:

```
int arrayList[5];
vector<int> vectorList(5);
```

#	Question	Your answer
1	arrayList[6] = 777 will yield a compiler error.	True
		False
2	vectorList.at(6) = 777 will yield a compiler error.	True
		False
3	arrayList[6] = 777 will execute without an error message.	True
		False
4	vectorList.at(6) = 777 will execute without an error message.	True
		False
5	vectorList[6] = 777 will execute without an error message.	True
		False
6	`while ( i < arrayList.size() )` loops while i is less than the array's size.	True
		False

Section 5.12 - Two-dimensional arrays

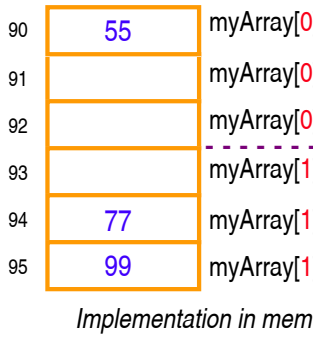
An array can be defined with two dimensions. `int myArray[R][C]` represents a table of int variables with R rows and C columns, so R*C elements total. For example, `int myArray[2][3]` creates a table with 2 rows and 3 columns, for 6 int variables total. Example accesses are `myArray[0][0] = 33;` or `num = myArray[1][2]`.

Participation Activity

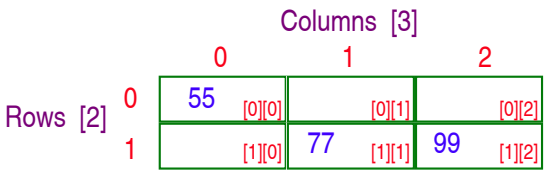
5.12.1: Two-dimensional array.

Start

```
// Define array with size [2][3]

// Write to some elements
myArray[0][0] = 55;
myArray[1][1] = 77;
myArray[1][2] = 99;
```



Implementation in mem



Conceptually a table

Conceptually, a two-dimensional array is a table with rows and columns. The compiler maps two-dimensional array elements to one-dimensional memory, each row following the previous row, known as ***row-major order***.

Figure 5.12.1: Using a two-dimensional array: A driving distance between cities example.

```
#include <iostream>
using namespace std;

/* Direct driving distances between cities, in miles */
/* 0: Boston  1: Chicago  2: Los Angeles */

int main() {
   int cityA = 0;              // Starting city
   int cityB = 0;              // Destination city
   int DrivingDistances[3][3]; // Driving distances

   // Initialize distances array
   DrivingDistances[0][0] = 0;
   DrivingDistances[0][1] = 960;  // Boston-Chicago
   DrivingDistances[0][2] = 2960; // Boston-Los Angeles
   DrivingDistances[1][0] = 960;  // Chicago-Boston
   DrivingDistances[1][1] = 0;
   DrivingDistances[1][2] = 2011; // Chicago-Los Angeles
   DrivingDistances[2][0] = 2960; // Los Angeles-Boston
   DrivingDistances[2][1] = 2011; // Los Angeles-Chicago
   DrivingDistances[2][2] = 0;

   cout << "0: Boston  1: Chicago  2: Los Angeles" << endl;

   cout << "Enter city pair (Ex: 1 2) -- ";
   cin >> cityA;
   cin >> cityB;

   cout << "Distance: " << DrivingDistances[cityA][cityB];
   cout << " miles." << endl;

   return 0;
}
```

```
0: Boston  1: Chicago
Enter city pair (Ex:
Distance: 2011 miles.

...

0: Boston  1: Chicago
Enter city pair (Ex:
Distance: 2960 miles.

...

0: Boston  1: Chicago
Enter city pair (Ex:
Distance: 0 miles.
```

A programmer can initialize a two-dimensional array's elements during definition using nested braces, as below. Multiple lines make the rows and columns more visible.

Construct 5.12.1: Initializing a two-dimensional array during definition.

```
// Initializing a 2D array
int numVals[2][3] = { {22, 44, 66}, {97, 98, 99} };

// Use multiple lines to make rows more visible
int numVals[2][3] = {
   {22, 44, 66}, // Row 0
   {97, 98, 99}  // Row 1
};
```

Arrays of three or more dimensions can also be defined, as in `int myArray[2][3][5]`, which defines a total of 2*3*5 or 30 elements. Note the rapid growth in size -- an array defined as `int myArray[100][100][5][3]` would have 100*100*5*3 or 150,000 elements. A programmer should make sure not to unnecessarily occupy available memory with a large array.

Participation Activity

5.12.2: Two-dimensional arrays.

#	Question	Your answer
1	Define a two dimensional array of integers named dataVals with 4 rows and 7 columns.	
2	How many total elements are in an array with 4 rows and 7 columns?	
3	How many elements are in the array defined as: char streetNames[20][50];	
4	Write a statement that assigns 99 into the fifth row, third column of array dataVals. Note: the first row/column is at index 0, not 1.	

Challenge Activity

5.12.1: Find 2D array max and min.

Find the maximum value and minimum value in milesTracker. Assign the maximum value t given program:

```
Min miles: -10
Max miles: 40
```

```
#include <iostream>
using namespace std;

int main( ) {
   const int NUM_ROWS = 2;
   const int NUM_COLS = 2;
   int milesTracker[NUM_ROWS][NUM_COLS];
   int i = 0;
   int j = 0;
   int maxMiles = -99; // Assign with first element in milesTracker
   int minMiles = -99; // Assign with first element in milesTracker

   milesTracker[0][0] = -10;
   milesTracker[0][1] = 20;
   milesTracker[1][0] = 30;
   milesTracker[1][1] = 40;

   /* Your solution goes here  */

    cout << "Min miles: " << minMiles << endl;
    cout << "Max miles: " << maxMiles << endl;
}
```

Run

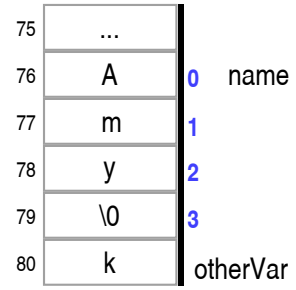
Section 5.13 - Char arrays / C strings

A programmer can use an array to store a sequence of characters, known as a ***string***. Char arrays were the only kinds of strings in C++'s predecessor language C, and thus are sometimes called ***C strings*** to distinguish them from C++'s string type. They are still needed in some C++ code and thus good to learn. An example is: `char movieTitle[20] = "Star Wars";`. Because a string can be shorter than the character array, a string in a char array must end with a special character known as a ***null character***, written as '\0'. Given a string literal like "Star Wars", the compiler automatically appends a null character.

Participation Activity

5.13.1: A char array definition with a compiler-added null character.

```
char userName[4] = "Amy";
```



A char array of size 20 can store strings of lengths 0 to 19. The longest string is 19, not 20, since the null character must be stored.

If a char array is initialized when declared, then the char array's size may be omitted, as in `char userName[] = "Hellen";` . The compiler determines the size from the string literal, in this case 6 + 1 (for the null character), or 7.

An array of characters ending with a null character is known as a ***null-terminated string***.

Output streams automatically handle null-terminated strings, printing each character until reaching the null character that ends the string.

Figure 5.13.1: Printing stops when reaching the null character at each string's end.

```
#include <iostream>
using namespace std;

int main() {
   char cityName[20] = "Forest Lake"; // Compiler appends null char

   // In each cout, printing stops when reaching null char
   cout << "City:" << endl;           // Compiler appends null char to "City:"
   cout << cityName << endl;

   return 0;
}
```

Participation Activity

5.13.2: Char array strings.

Indicate whether the array definition and initialization are appropriate.

#	Question	Your answer
1	`char firstName[10] = "Henry";`	True / False
2	`char lastName[10] = "Michelson";`	True / False
3	`char favoriteMuseum[10] = "Smithsonian";`	True / False
4	Given: `char catBreed[20] = "Persian";` Printing catBreed will print 19 characters.	True / False

After a string is defined, a programmer may not later assign the string as in `movieTitle = "Indiana Jones";`. That statement tries to assign a value to the char array variable itself, rather than copying each character from the string on the right into the array on the left. Functions exist to *copy* strings, such as strcpy(), discussed elsewhere.

A programmer can traverse a string using a loop that stops when reaching the null character.

A common error is to loop for the string's array size rather than stopping at the null character. Such looping visits unused array elements beyond the null character. An even worse common error is to loop beyond the last valid element, which visits memory locations that are not part of the array. These errors are illustrated below. Notice the strange characters that are output as the contents of other memory locations are printed out; the program may also crash.

Figure 5.13.2: Traversing a C string.

```
#include <iostream>
using namespace std;

int main() {
   char  userStr[20] = "1234567890123456789"; // Input string
   int   i = 0;

   // Prompt user for string input
   cout << "Enter string (<20 chars): ";
   cin >> userStr;

   // Print string
   cout << endl << userStr << endl;

   // Look for '@'
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (userStr[i] == '@') {
         cout << "Found '@'." << endl;
      }
   }
   cout << endl;


   // The following is an ERROR.
   // May print chars it shouldn't.
   // Problem: doesn't stop at null char.
   cout << "\""; // Print opening "
   for (i = 0; i < 20; ++i) { // Print each char
      cout << userStr[i];
   }
   cout << "\"" << endl;      // Print closing "


   // The following is an even WORSE ERROR.
   // Accesses beyond valid index range.
   // Program may crash.
   cout << "\""; // Print opening "
   for (i = 0; i < 30; ++i) {
      cout << userStr[i];
   }
   cout << "\"" << endl; // Print closing "

   return 0;
}
```

```
Enter string (<20 c

test@gmail.com
Found '@'.

"test@gmail.com6789
"test@gmail.com6789
```

The above output is machine and compiler dependent. Also, some values aren't printable so don't appear in the output.

Participation Activity

5.13.3: C string errors.

Given the following char array declaration, which of the following code snippets are bad?

```
char userText[10] = "Car";
```

#	Question	Your answer

1	`for (i = 0; userText != '\0'; ++i) {`
	`  // Print userText[i]`
	`}`
		OK
		Bad

2	`for (i = 0; userText[i] != '\0'; ++i) {`
	`  // Print userText[i]`
	`}`
		OK
		Bad

3	`for (i = 0; i < 10; ++i) {`
	`  // Print userText[i]`
	`}`
		OK
		Bad

4	`for (i = 0; i < 4; ++i) {`
	`  // Print userText[i]`
	`}`
		OK
		Bad

5	`userText = "Bus";`	OK Bad

Yet another common error with C strings is for the program user to enter a string larger than the character array. That may cause the input statement to write to memory locations outside the array's locations, which may corrupt other parts of program or data, and typically causes the program to crash.

Participation Activity

5.13.4: Reading in a string too large for a C string.

Run the program, which simply reads an input string and prints it one character at a time. Then, lengthen the input string beyond 10 characters, and run again. The program *might* work, if the extra memory locations being assigned don't matter. Try larger and larger strings, and see if the program fails (be sure to scroll to the bottom of the output to look for erroneous output or an error message).

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     char  userStr[10]; // Input string
6     int   i = 0;
7
8     // Prompt user for string input
9     cout << "Enter string (<10 chars): ";
10    cin >> userStr;
11
12    // Print 1 char at a time
13    cout << endl;
14    for (i = 0; userStr[i] != '\0'; ++i) {
15       cout << userStr[i] << endl;
16    }
17    cout << endl;
18
19    return 0;
20 }
21
```

Hello

Run

C string usage is fraught with common errors. C++ introduced its own string type, as in `string myString;` and accessible after `#include <string>`, to reduce those errors and increase programmer convenience. Yet legacy code uses C strings, and some commonly-used C++ functions only accept C strings as parameters (e.g., `ofstream myfile; myfile.open ("somefile.txt");`), and thus the C++ programmer should be comfortable with C strings. C++ provides common functions for handling C strings, which can be used by including the following: `#include <cstring>`.

The following program is for illustration, showing how a string is made up of individual character elements followed by a null character. Normally a programmer would not create a string that way.

Figure 5.13.3: A C string is an array of characters, ending with the null character.

```
#include <iostream>
using namespace std;

int main() {
   char nameArr[5];

   nameArr[0] = 'A';
   nameArr[1] = 'l';
   nameArr[2] = 'a';
   nameArr[3] = 'n';
   nameArr[4] = '\0'; // Null character

   cout << nameArr << endl;

   nameArr[4] = '!';         // Oops, overwrote null char
   cout << nameArr << endl; // *Might* still work

   return 0;
}
```

```
Alan
Alan!
```

When printing a string stored within a character array, each character within the array will be printed until the null character is reached. If the null character is omitted, the program would print whatever values are found in memory after the array, until a null character happens to be encountered. Omitting the null character is a serious logical error.

It just so happens that the null character '\0' has an ASCII encoding of 0. Many compilers initialize memory to 0s. As such, omitting the '\0' in the above program would not always cause erroneous execution. Like a nail in the road, that bug in your code is just waiting to wreak havoc.

Participation Activity

5.13.5: C string without null character.

Given:

```
char userText[10];
userText[0] = 'C';
userText[1] = 'a';
userText[2] = 'r';
userText[3] = '\0';
...
userText[3] = 's';
```

#	Question	Your answer
1	The first four characters in userText are now: Cars.	True
		False
2	The compiler generates an error, because element 3 is the null character and can't be overwritten.	True
		False
3	Printing userText should work fine because the new string is 4 characters, which is still much less than the array size of 10.	True
		False

Section 5.14 - String library functions

C++ provides functions for working with C strings, presented in the ***cstring*** library. To use those functions, the programmer starts with: `#include <cstring>`.

Some C string functions for *modifying* strings are summarized below.

Table 5.14.1: Some C string modification functions.

Given:

```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including null character) to destStr.
		`strcpy(targetText, user`
		
		`strcpy(targetText, orgNa`
		
		`targetText = orgName;`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters.	`strncpy(orgName, userTe`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr (up to and including null character) to *end* of destStr (starting at destStr's null character).	`strcat(orgName, userTex`
strncat()	`strncat(destStr, sourceStr, numChars)` Copies up to numChars characters to destStr's end, then appends null character.	`strcpy(targetText, "abc"` `strncat(targetText, "12`

For strcpy(), a common error is to copy a source string that is too large, causing an out-of-range access in the destination string. Another common error is to call strcpy with the source string first rather than the destination string, which copies in the wrong direction.

Note that string assignment, as in targetText = orgName, does not copy the string and should not be used. The exception is during initialization, as in `char userText[20] = "UNICEF";`, for which the compiler copies the string literal's characters into the array.

Participation Activity

5.14.1: String modification functions.

Given: char userStr[5];
Do not type quotes in your answers.
If appropriate, type: Error

#	Question	Your answer
1	What is userStr after: strcpy(userStr, "Bye");	
2	If userStr is initially "Hi", what is userStr after: strcpy(userStr, "Bye");	
3	What is userStr after: strcpy(userStr, "Goodbye");	
4	What is userStr after: strncpy(userStr, "Goodbye", 4);	
5	If userStr is initially "Hi", what is userStr after: strcat(userStr, '!');	
6	If userStr is initially "Hi", what is userStr after: strcat(userStr, "!");	
7	If userStr is initially "Hi", what is userStr after: strncat(userStr, "?!$#@%", 2);	

Several C string functions that get *information* about strings are summarized below.

Table 5.14.2: Some C string information functions.

Given:

```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the cstring library.
		`if (strchr(orgName, 'U') != NULL)`
		`  ...  // 'U' exists in "United N`
		`}`
		`if (strchr(orgName, 'u') != NULL)`
		`  ...  // 'u' doesn't exist (case`
		`}`

strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);  // Assigns`
		`x = strlen(userText);  // Assigns`
		`x = strlen(targetText); // Error:`

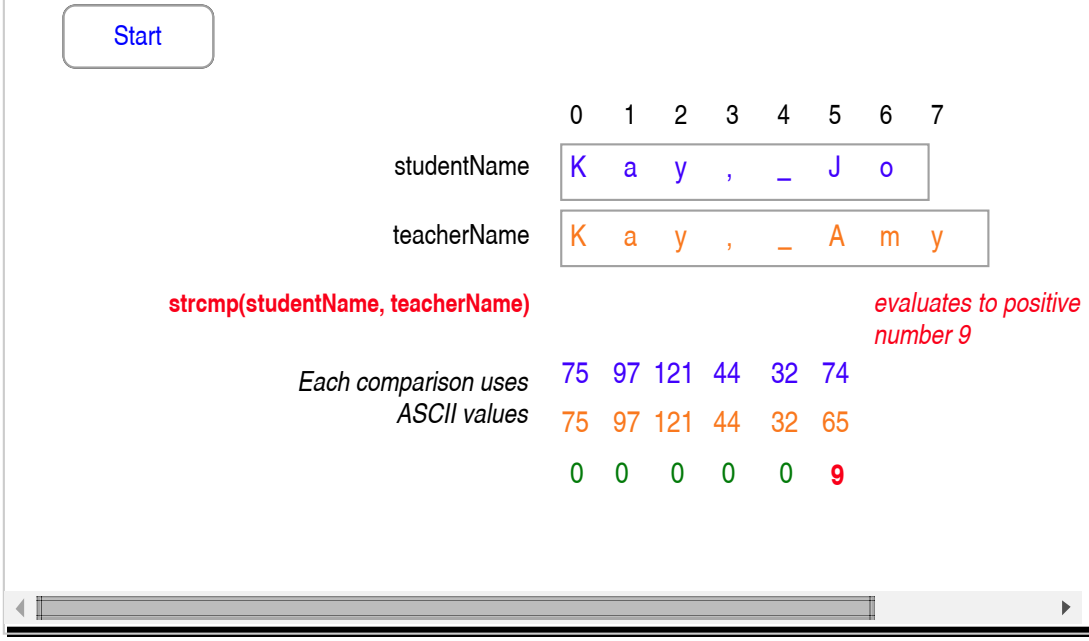
strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName, "United Nation`
		`  ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) == 0`
		`  ... // Not equal, branch not ta`
		`}`

strcmp() is usually used to compare for equality, returning 0 if the strings are the same length and have identical characters. A common error is to use == when comparing C strings, which does not work. str1 == str2 compares the strings' addresses, not their contents. Because those addresses will usually be different, str1 == str2 will evaluate to 0. This is not a syntax error, but clearly a logic error. Another common error is to forget to compare the result of strcmp with 0, as in `if (strcmp(str1, str2)) {...}`. The code is not a syntax error, but is a logic error because the if condition will be false (0) when the strings are equal. The correct condition would instead be `if (strcmp(str1, str2) == 0) {...}`. Although strcmp returns 0, a good practice is to avoid using `if (!strcmp(str1,str2)) {...}` because that 0 does not represent "false" but rather is encoding a particular situation.

strcmp(str1, str2) returns a negative number if str1 is less than str2, and a positive number if str1 is greater than str2. Evaluation first compares the character pair at element 0, then at element 1, etc., returning as soon as a pair differs.

Participation Activity

5.14.2: String comparison.



strlen is often used to iterate through each string character in a loop.

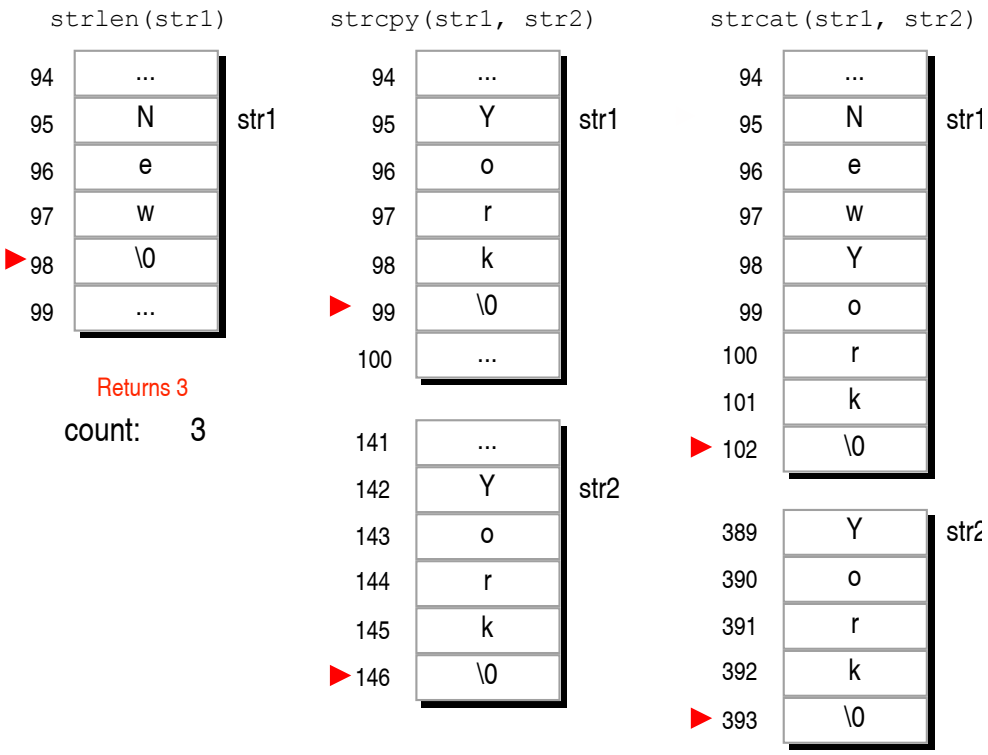
Figure 5.14.1: Iterating through a C string using strlen.

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
   char userName[15] = "Alan Turing";
   int i = 0;

   cout << "Before: " << userName << endl;

   for (i = 0; i < strlen(userName); ++i) {
      if (userName[i] == ' ') {
         userName[i] = '_';
      }
   }
   cout << "After:  " << userName << endl;

   return 0;
}
```

```
Before: Alan Turing
After:  Alan_Turing
```

Participation Activity

5.14.3: Some C string library functions.



Participation Activity

5.14.4: String information functions.

Given:

```
char str1[10] = "Earth";
char str2[20] = "Earthlings";
char str3[15] = "Mars";
```

Answer the following questions. If appropriate, type: Error

#	Question	Your answer
1	What does strlen(str3) return?	

2	Is the branch taken? (Yes/No/Error)
	
	`if (strchr(str1, '@') != NULL) {`
	`  // Print "Found @"`
	`}`	

3	Is the branch taken? (Yes/No/Error)
	
	`if (strchr(str1, 'E') != NULL) {`
	`  // Print "Found E"`
	`}`	

4	Is the branch taken? (Yes/No/Error)
	
	`if (strchr(str2, "Earth") != NULL) {`
	`  // Print "Found Earth"`
	`}`	

5	Is the branch taken? (Yes/No/Error)
	
	`if (strcmp(str1, str2) == 0) {`
	`  // Print "strings are equal"`
	`}`	

6	Is the branch taken? (Yes/No/Error)
	
	`if (str1 == str3) {`
	`  // Print "strings are equal"`
	`}`	

7	Finish the code to take the branch if str1 and str3 are equal.	`if (strcmp(str1, str3)  ) {`
		`  // Strings are equal`
		`}`

Exploring further:

- More C string functions from cplusplus.com

Section 5.15 - Char library functions: ctype

C++ provides common functions for working with characters, presented in the cctype library, the ctype short for "character type", and the first c indicating the library is a C language standard library. To use those functions, the programmer adds the following at the top of a file: `#include <cctype>`

Commonly-used cctype functions are summarized below; a complete reference is found at http://www.cplusplus.com/reference/cctype/.

Character checking functions

The following functions check whether a character is of a given category, returning either false (0) or true (non-zero). The examples below assume the following string declaration.

```
char myString[30] = "Hey9! Go";
```

- ***isalpha(c)*** -- Returns true if c is alphabetic: a-z or A-Z.
  - `isalpha('A'); // Returns true`
  - `isalpha(myString[0]);` // Returns true because 'H' is alphabetic
  - `isalpha(myString[3]);` // Returns false because '9' is not alphabetic
- ***isdigit(c)*** -- Returns true if c is a numeric digit: 0-9.
  - `isdigit(myString[3]);` // Returns true because '9' is numeric
  - `isdigit(myString[4]);` // Returns false because ! is not numeric
- ***isalnum(c)*** -- Returns true if c is alphabetic or a numeric digit. Thus, returns true if either isalpha or isdigit would return true.
- ***isspace(c)*** -- Returns true if character c is a whitespace.
  - `isspace(myString[5]);` // Returns true because that character is a space ' '.
  - `isspace(myString[0]);` // Returns false because 'H' is not whitespace.

- ***islower(c)*** -- Returns true if character c is a lowercase letter a-z.
    - `islower(myString[0]);` // Returns false because 'H' is not lowercase.
    - `islower(myString[1]);` // Returns true because 'e' is lowercase.
    - `islower(myString[3]);` // Returns false because '9' is not a lowercase letter.

- ***isupper(c)*** -- Returns true if character c is an uppercase letter A-Z.

- ***isblank(c)*** -- Returns true if character c is a blank character. Blank characters include spaces and tabs.
    - `isblank(myString[5]);` // Returns true because that character is a space ' '.
    - `isblank(myString[0]);` // Returns false because 'H' is not blank.

- ***isxdigit(c)*** -- Returns true if c is a hexadecimal digit: 0-9, a-f, A-F.
    - `isxdigit(myString[3]);` // Returns true because '9' is a hexadecimal digit.
    - `isxdigit(myString[1]);` // Returns true because 'e' is a hexadecimal digit.
    - `isxdigit(myString[6]);` // Returns false because 'G' is not a hexadecimal digit.

- ***ispunct(c)*** -- Returns true if c is a punctuation character. Punctuation characters include: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
    - `ispunct(myString[4]);` // Returns true because '!' is a punctuation character.
    - `ispunct(myString[6]);` // Returns false because 'G' is not a punctuation character.

- ***isprint(c)*** -- Returns true if c is a printable character. Printable characters include alphanumeric, punctuation, and space characters.

- ***iscntrl(c)*** -- Returns true if c is a control character. Control characters are all characters that are not printable.

Character conversion functions

The following functions return a character representing a converted version of the input character. The examples below assume the following string declaration.

```
char myString[30] = "Hey9! Go";
```

- ***toupper(c)*** -- If c is a lowercase alphabetic character (a-z), returns the uppercase version

(A-Z). If c is not a lowercase alphabetic character, just returns c.

- `toupper(myString[0]);` // Returns 'H' (no change)
- `toupper(myString[1]);` // Returns 'E' ('e' converted to 'E')
- `toupper(myString[3]);` // Returns '9' (no change)
- `toupper(myString[5]);` // Returns ' ' (no change)

- ***tolower(c)*** -- If c is an uppercase alphabetic character (A-Z), returns the lowercase version (a-z). If c is not an uppercase alphabetic character, just returns c.

The following example illustrates some of the cctype functions.

Figure 5.15.1: Use of some functions in cctype.

```
#include <iostream>
#include <cctype>
using namespace std;

int main() {
   const int MAX_LEN = 30;       // Max string length
   char  userStr[MAX_LEN] = ""; // User defined string
   int   i = 0;

   // Prompt user to enter string
   cout << "Enter string (<"
   << MAX_LEN << " chars): ";
   cin >> userStr;

   cout << "Original: " << userStr << endl;

   cout << "isalpha:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      cout << isalpha(userStr[i]);
   }
   cout << endl;

   cout << "isdigit:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      cout << isdigit(userStr[i]);
   }
   cout << endl;

   cout << "isupper:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      cout << isupper(userStr[i]);
   }
   cout << endl;

   for (i = 0; userStr[i] != '\0'; ++i) {
      userStr[i] = toupper(userStr[i]);
   }
   cout << "After toupper: " << userStr << endl;

   return 0;
}
```

```
Enter string (<30 chars):
Original: ABC123$%&def
isalpha:  111000000111
isdigit:  000111000000
isupper:  111000000000
After toupper: ABC123$%&DE
```

Participation Activity

5.15.1: Character type functions.

Enter the value to which each function evaluates, using 1 for true, 0 for false. Assume str is "Hi 321!".

#	Question	Your answer
1	isalpha(str[0])	
2	isdigit(str[4])	
3	isalnum(str[2])	
4	isspace(str[2])	
5	islower(str[6])	
6	tolower(str[0])	
7	tolower(str[1])	

To compare two strings without paying attention to case, one technique is to first convert (a copy of) each string to lowercase (using a loop, discussed elsewhere) and then comparing.

Section 5.16 - C++ example: Salary calculation with vectors

Participation Activity

5.16.1: Various tax rates.

Vectors are useful to process tabular information. Income taxes are based on annual salary, usually with a tiered approach. Below is an example of a simple tax table:

Annual Salary	Tax Rate
0 to 20000	10%
Above 20000 to 50000	20%
Above 50000 to 100000	30%
Above 100000	40%

The below program uses a vector salaryBase to hold the cutoffs for each salary level and a parallel vector taxBase that has the corresponding tax rate.

1. Run the program and enter annual salaries of 40000 and 60000, then enter 0.
2. Modify the program to use two parallel vectors named annualSalaries and taxesToPay, each with 10 elements. Vectors annualSalaries holds up to 10 annual salaries entered; vector taxesToPay holds up to 10 corresponding amounts of taxes to pay for those annual salaries. Print the total annual salaries and taxes to pay after all input has been processed.
3. Run the program again with the same annual salary numbers as above.

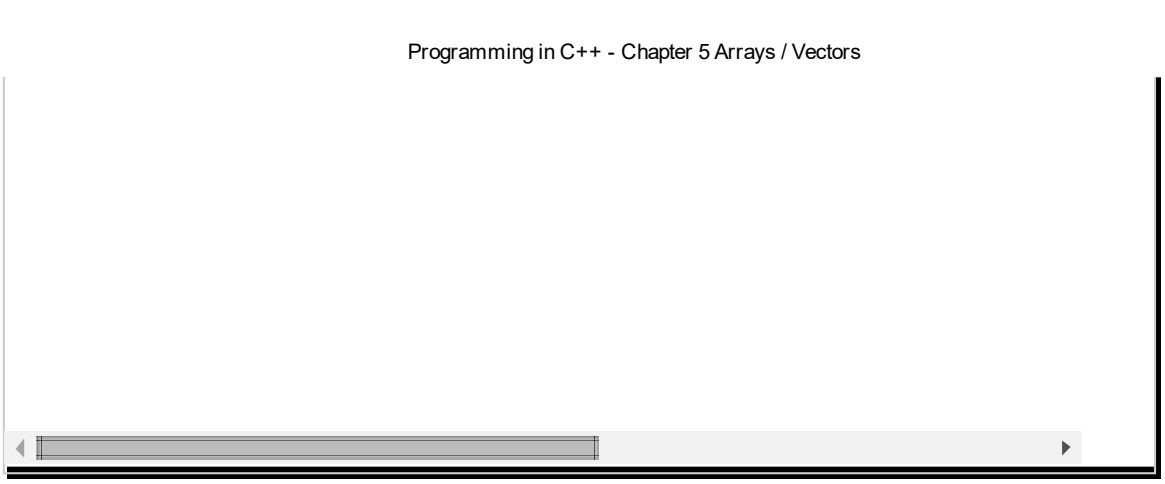
The following program calculates the tax rate and tax to pay based on annual income.

Reset

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int MAX_ELEMENTS = 10;
   int annualSalary       = 0;
   double taxRate         = 0.0;
   int taxToPay           = 0;
   int numSalaries        = 0;
   bool keepLooking       = true;
   int i = 0;
   vector<int> salaryBase(5);
   vector<double> taxBase(5);

   salaryBase.at(0) = 0;
   salaryBase.at(1) = 20000;
   salaryBase.at(2) = 50000;
   salaryBase.at(3) = 100000;
   salaryBase.at(4) = 99999999;

   taxBase.at(0) = 0.0;
```

40000 60000 0

Run



A solution to above problem follows.

Participation Activity

5.16.2: Solution to salary calculation with vectors.

Reset

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
   const int MAX_ELEMENTS = 10;
   int annualSalary       = 0;
   double taxRate         = 0.0;
   int taxToPay           = 0;
   int totalSalaries      = 0;
   int totalTaxes         = 0;
   int numSalaries        = 0;
   bool keepLooking       = true;
   int i = 0;

   vector<int> salaryBase(5);
   vector<double> taxBase(5);
   vector<int> annualSalaries(MAX_ELEMENTS);
   vector<int> taxesToPay(MAX_ELEMENTS);

   salaryBase.at(0) = 0;
   salaryBase.at(1) = 20000;
```

```
40000 60000 0
```

Run

Section 5.17 - C++ example: Domain name validation with vectors

Participation Activity

5.17.1: Validate domain names with vectors.

Vectors are useful to process lists.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***restricted top-level domain*** is a TLD that is either .biz, .name, or .pro. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com.

The following program repeatedly prompts for a domain name, and indicates whether that domain name consists of a second-level domain followed by a core gTLD. Valid core gTLD's are stored in a vector. For this program, a valid domain name must contain only one period, such as apple.com, but not support.apple.com. The program ends when the user presses just the Enter key in response to a prompt.

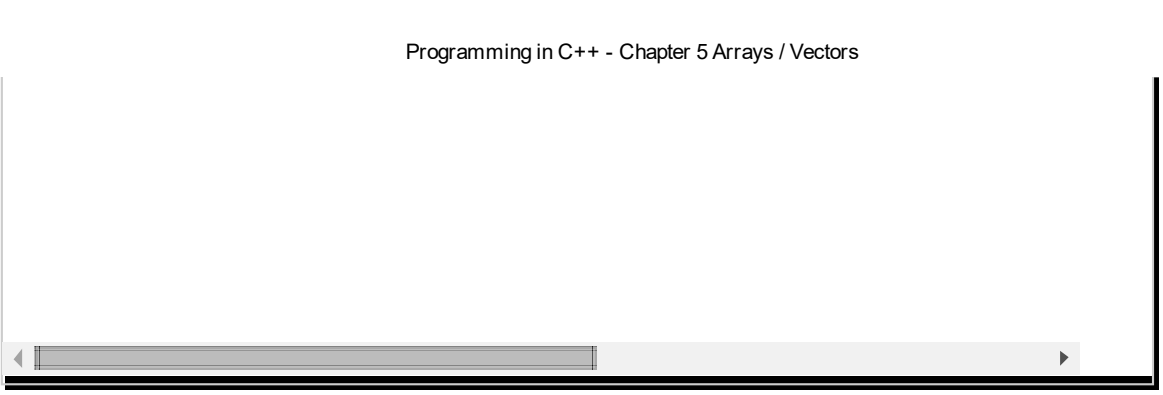
1. Run the program and enter domain names to validate.
2. Extend the program to also recognize restricted TLDs using a vector, and statements to validate against that vector. The program should also report whether the TLD is a core gTLD or a restricted gTLD. Run the program again.

Reset

```
#include <iostream>
#include <vector>
using namespace std;

int main() {

   // Define the list of valid core gTLDs
   const int NUM_ELEMENTS = 4;
   vector<string> validCoreGtld(NUM_ELEMENTS);
   // FIXME: Define a vector named validRestrictedGtld that has the
   //        of the restricted domains, .biz, .name, and .pro
   string  inputName      = "";
   string  searchName     = "";
   string  theGtld        = "";
   bool isValidDomainName = false;
   bool isCoreGtld        = false;
   bool isRestrictedGtld  = false;
   int periodCounter      = 0;
   int periodPosition     = 0;
   int i = 0;

   validCoreGtld.at(0) = ".com";
```

```
apple.com
APPLE.com
apple.comm
```

Run



Participation Activity

5.17.2: Validate domain names with vectors (solution).

A solution to the problem posed above follows.

Reset

```
#include <iostream>
#include <vector>
using namespace std;

int main() {

   // Define the list of valid core gTLDs
   const int NUM_ELEMENTS_CORE = 4;
   vector<string> validCoreGtld(NUM_ELEMENTS_CORE);
   const int NUM_ELEMENTS_RSTR = 3;
   vector<string> validRestrictedGtld(NUM_ELEMENTS_RSTR);
   string  inputName      = "";
   string  searchName     = "";
   string  theGtld        = "";
   bool isValidDomainName = false;
   bool isCoreGtld        = false;
   bool isRestrictedGtld  = false;
   int periodCounter      = 0;
   int periodPosition     = 0;
   int i = 0;

   validCoreGtld.at(0) = ".com";
```

apple.com
APPLE.com
apple.comm

Run