6.1 Vectors

Vector declaration and accessing elements

A programmer commonly needs to maintain a list of items, just as people often maintain lists of items like a grocery list or a course roster. A ***vector*** is an ordered list of items of a given data type. Each item in a vector is called an ***element***. A programmer must include the statement `#include <vector>` at the top of the file when planning to use vectors.

Construct 6.1.1: Vector declaration.

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

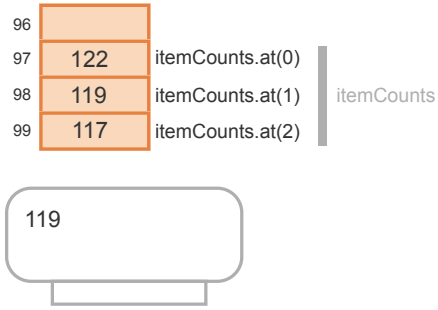
The statement above declares a vector with the specified number of elements, each element of the specified data type. The type of each vector element is specified within the angle brackets (<>). The number of vector elements is specified within parentheses following the vector name. Ex: `vector<int> gameScores(4);` declares a vector gameScores with 4 integer elements.

Terminology note: { } are ***braces***. < > are ***angle brackets***, or ***chevrons***. In a vector access, the number in .at() parentheses is called the ***index*** of the corresponding element. The first vector element is at index 0.

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

PARTICIPATION ACTIVITY | 6.1.1: A vector declaration creates multiple variables in memory, each accessible using .at().

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

int main() {
    vector<int> itemCounts(3);

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

    cout << itemCounts.at(1);

    return 0;
}
```



Animation content:

Static figure:
Begin C++ code:
#include <iostream>
#include <vector>
using namespace std;

```
int main() {
   vector<int> itemCounts(3);

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

   cout << itemCounts.at(1);

   return 0;
}
```

End C++ code.

The elements of the vector itemCounts are displayed with the elements' locations in memory, as well as the elements' corresponding indices. Integer element 122 is located at memory location 97 and is at index 0. Integer element 119 is located at memory location 98 and is at index 1. Integer element 117 is located at memory location 99 and is at index 2. A monitor with the integer 119 on the monitor is displayed, as well.

Step 1: A vector named itemCounts is declared. The vector consists of 3 elements, each of data type int. The statement: vector<int> itemCounts(3); declares the vector itemCounts with storage for 3 integer elements.

Step 2: An element is accessed with the at() function. The number in parentheses is the index of the corresponding element. The statement: itemCounts.at(1) = 119; accesses the element at index 1 and assigns the element with the integer 119 using the at() function. The statement: cout << itemCounts.at(1); accesses the element at index 1 and displays the integer 119 on the monitor.

Animation captions:

1. A vector named itemCounts is declared. The vector consists of 3 elements, each of data type int.
2. An element is accessed with the at() function. The number in parentheses is the index of the corresponding element.

PARTICIPATION ACTIVITY 6.1.2: Vector basics.

Given:

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

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


1) How many elements does the vector declaration create?

- 0
- 1
- 3
- 4

2) With what value is yearsList.at(1) assigned?

- 1
- 1999
- 2012

3) With what value does `currYear = yearsList.at(2)` assign currYear?

- 2
- 2025
- Invalid index

4) Is `currYear = yearsList.at(4)` a valid assignment?

- Yes, the fourth element is accessed.
- 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> pricesList(100);`

- 99
- 100
- 101

Using an expression for a vector index

A powerful aspect of vectors is that the index is an expression. Ex: userNums.at(i) uses the value held in the int variable i as the index. As such, a vector is useful to easily lookup the Nth item in a list.

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

The program below allows a user to print the age of the Nth oldest known person to have ever lived. The program quickly accesses 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.

Figure 6.1.1: Vector's ith element can be directly accessed using .at(i - 1): Oldest people program.

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

int main() {
   vector<int> oldestPeople(5);
   int nthPerson;               // 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 << " oldest
person lived ";
      cout << oldestPeople.at(nthPerson - 1) << "
years." << endl;
   }

   return 0;
}
```

```
Enter N (1..5): 1
The 1th oldest person lived
122 years.

...

Enter N (1..5): 4
The 4th oldest person lived
117 years.

...

Enter N (1..5): 9

...

Enter N (1..5): 0

...

Enter N (1..5): 5
The 5th oldest person lived
116 years.
```


PARTICIPATION ACTIVITY 6.1.3: Nth oldest person program.

1) In the program above, what is the purpose of this check:

```
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 the vector oldestPeople only has 5 elements.

PARTICIPATION ACTIVITY 6.1.4: Vector declaration and accesses.

1) Declare a vector named myVals that stores 10 items of type int.

Check Show answer

2) Assign x with the value stored at index 8 of vector myVals.

Check Show answer

3) Given myVals has 10 elements, assign the last element in myVals with the value 555.

Check Show answer

4) Assign myVals element at the index held in currIndex with the value 777.

Check Show answer

5) Assign tempVal with the myVals element at the index one after the value held in variable i.

Check Show answer

Loops and vectors

A key advantage of vectors becomes evident when used in conjunction with loops. The program below uses a loop to allow a user to enter 8 integer values, storing those values in a vector, and then printing those 8 values.

A vector's ***size()*** function returns the number of vector elements. Ex: In the program below, userVals.size() is 8 because the vector was declared with 8 elements.

Figure 6.1.2: 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
   unsigned int i;                  // Loop index

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

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

   return 0;
}
```

```
Enter 8 integer values...
Value: 5
Value: 99
Value: -1
Value: -44
Value: 8
Value: 555555
Value: 0
Value: 2
You entered: 5 99 -1 -44 8
555555 0 2
```


PARTICIPATION ACTIVITY 6.1.5: Vector with loops.

Refer to the program above.

1) How many times does each for loop iterate?
   - 1
   - 8
   - Unknown

2) Which one line of code can be changed to allow the user to enter 100 elements?
   - `const int NUM_VALS = 8;`
   - `for (i = 0; i < userVals.size(); ++i) {`

Vector initialization

A vector's elements are automatically initialized to 0s during the vector declaration.

All of a vector's elements may be initialized to another single value. Ex: `vector<int> myVector(3, -1);` creates a vector named myVector with three elements, each with value -1.

A programmer may initialize each vector element with different values by specifying the initial values in braces {} separated by commas. Ex: `vector<int> carSales = {5, 7, 11};` creates a vector of three integer elements initialized with values 5, 7, and 11. Such vector declaration and initialization does not require specifying the vector size, because the vector's size is automatically set to the number of elements within the braces. For a larger vector, initialization may be done by first declaring the vector, and then using a loop to assign vector elements.

PARTICIPATION ACTIVITY 6.1.6: Vector initialization.

1) Write a single statement to declare a vector of ints named salesGoals with 4 elements each initialized to 10.

Check Show answer

2) Given the following, what is maxScores.at(3)?

```
vector<int> maxScores = {20,
20, 100, 50};
```

Check Show answer

Common error: Forgetting to include <vector>

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

```
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.

CHALLENGE ACTIVITY 6.1.1: Enter the output for the vector.

539740.3879454.qx3zqy7

Start

Type the program's output

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

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

   userVals.at(0) = 3;
   userVals.at(1) = 6;
   userVals.at(2) = 8;

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

   return 0;
}
```

```
3
6
8
```


1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 6.1.2: Vectors.

539740.3879454.qx3zqy7

Start

Two integers are read from input as variables member1 and member2. Declare a vector of integers named memberListings initialize the elements with the variables member1 and member2 in the opposite order the input integers are read.

Ex: If the input is 21 26, then the output is:

```
26 21
```

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     int member1;
7     int member2;
8     unsigned int i;
9
10    cin >> member1;
11    cin >> member2;
12
13    /* Your code goes here */
14
15    for (i = 0; i < memberListings.size(); ++i) {
```


1 2 3 4

Check Next level

6.2 Iterating through vectors

Iterating through vectors using loops

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 6.2.1: Common for loop structure for iterating through a vector.

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

Note that index variable i is initialized to 0, and the loop expression is i < myVector.size() rather than i <= myVector.size(). If myVector.size() 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 myVector.at(i) rather than the more complex myVector.at(i - 1).

PARTICIPATION ACTIVITY 6.2.1: Iterating through a vector.

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;
}
```

Check Show answer

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;
}
```


Check Show answer

Determining a quantity about a vector's items

Iterating through a vector for various purposes is an important programming skill to master. Programs commonly iterate through vectors to determine some quantity about the vector's items. The example below computes the sum of a vector's element values.

Figure 6.2.2: 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
   unsigned int i;                      // Loop index
   int sumVal;                          // For computing
sum

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

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

   return 0;
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: -1
Sum: 920
```

Finding the maximum value in a vector

The program below determines the maximum value in a user-entered list. If the user enters numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output "max: 55". The program uses the 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 assigns maxVal with the current vector element.

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 with that element's value.

Figure 6.2.3: 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
   unsigned int i;                  // Loop index
   int maxVal;                      // Computed max

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

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

   return 0;
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 23
Value: -1
Value: 456
Value: 1
Value: 6
Value: 83
Max: 456

...

Enter 8 integer
values...
Value: -5
Value: -10
Value: -44
Value: -2
Value: -27
Value: -9
Value: -27
Value: -9
Max: -2
```


PARTICIPATION ACTIVITY 6.2.2: Iterating through vectors.

Complete the code provided to achieve the desired goal.

1) Find the minimum element value in vector `valsVctr`.

```
tempVal = valsVctr.at(0);
for (i = 0; i <
valsVctr.size(); ++i) {
   if (valsVctr.at(i) <
                        ) {
      tempVal= valsVctr.at(i);
   }
}
```

Check Show answer

2) Find the sum of all elements in vector `valsVctr`.

```
valSum =                    ;
for (i = 0; i <
valsVctr.size(); ++i) {
   valSum += valsVctr.at(i);
}
```

Check Show answer

3) Count the number of negative-valued elements in vector `valsVctr`.

```
numNeg = 0;
for (i = 0; i <
valsVctr.size(); ++i) {
   if (valsVctr.at(i) < 0) {
      numNeg =
                ;
   }
}
```

Check Show answer

zyDE 6.2.1: 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.

Load default template... Run

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int VALS_SIZE =
7     vector<int> valsVctr(
8     unsigned int i;
9     int sumVal;
10    int avgVal;
11
12    valsVctr.at(0) = 30;
13    valsVctr.at(1) = 20;
14    valsVctr.at(2) = 20;
15    valsVctr.at(3) = 15;
```

Common error: Accessing out of range vector element

A common error is to try to access a vector with an index that is out of the vector's index range. Ex: Trying to access highScores.at(8) when highScores 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 declaration `vector highScores(8)`, accessing highScores.at(8), or highScores.at(i) where i is 8, yields the following error message when running the program compiled with g++:

Figure 6.2.4: Sample error message when accessing an out of range vector index.

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

zyDE 6.2.2: 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.

Load default template... Run

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int VALS_SIZE =
7     vector<int> myVals(VA
8     unsigned int i;
9
10    myVals.at(0) = 30;
11    myVals.at(1) = 20;
12    myVals.at(2) = 20;
13    myVals.at(3) = 15;
14    myVals.at(4) = 5;
15    myVals.at(5) = 10;
```

PARTICIPATION ACTIVITY 6.2.3: Iterating through a vector.

Given the following code:

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

1) The normal for loop structure iterates as long as: `i <= myVctr.size()`
   - ○ True
   - ○ False

2) To compute the sum of elements, a reasonable statement preceding the for loop is: `int sumVal = 0;`
   - ○ True
   - ○ False

3) To find the maximum element value, a reasonable statement preceding the for loop is: `int maxVal = 0;`

- True
- False

CHALLENGE ACTIVITY 6.2.1: Enter the output for the vector.

539740.3879454.qx3zqy7

Start

Type the program's output

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

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

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

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

   return 0;
}
```

```
2
8
6
```

1 2 3 4 5 6

Check Next

CHALLENGE ACTIVITY 6.2.2: Iterating through vectors.

539740.3879454.qx3zqy7

Start

Integer numElements is read from input. Given the integer vector dailyMiles with the size of numElements, write a for loop tc output the integers in the second half of dailyMiles. Separate the integers with a comma followed by a space (", ").

Ex: If the input is:


```
8
65 75 92 94 101 119 122 130
```

then the output is:

```
101, 119, 122, 130
```

Note: The vector size is always even.

```
1 #include <iostream>
2 #include <vector>
```

```
using namespace std;

int main() {
   int numElements;
   unsigned int i;

   cin >> numElements;

   // Creates a vector of size numElements and initializes all values to 0
   vector<int> dailyMiles(numElements);

   for (i = 0; i < dailyMiles.size(); ++i) {
      cin >> dailyMiles.at(i);
```


1 2 3

Check Next level

6.3 Array/vector iteration drill

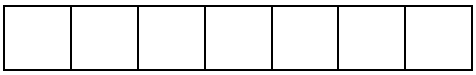
The following activities can help one become comfortable with iterating through arrays or vectors, before learning to code such iteration.

PARTICIPATION ACTIVITY 6.3.1: Find the maximum value in the array.

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

Start

max



Next value Store value

Time - Best time -

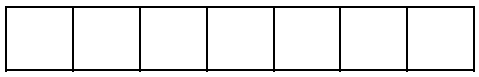
Clear best

PARTICIPATION ACTIVITY 6.3.2: Negative value counting in array.

Click "Increment" if a negative value is seen.

Start

Counter



0

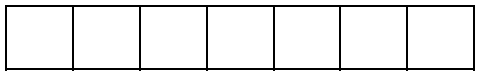
Next value Increment

Time - Best time -
Clear best

PARTICIPATION ACTIVITY 6.3.3: 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



Next value Swap values

Time - Best time -
Clear best

6.4 Using a loop to modify, copy, or compare vectors

Modifying vector elements

A program may need to modify elements while iterating through a vector. The program below uses a loop to convert any negative vector element value to 0.

Figure 6.4.1: 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
   unsigned int i;                      // Loop index

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

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

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

   return 0;
}
```

```
Enter 5 integer
values...
Value: 67
Value: -5
Value: -99
Value: 4
Value: 22
New values: 67 0 0 4
22
```


PARTICIPATION ACTIVITY 6.4.1: 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?

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
- -55, -55, -1, 0
- Error (program aborts)

zyDE 6.4.1: Modifying a vector during iteration example: Doubling element values.

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

Load default template...

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

int main() {
   const int NUM_ELEMENT
   vector<int> userVals(
   unsigned int i;

   // Prompt user to pop
   cout << "Enter " << N
   for (i = 0; i < userV
      cout << "Value: "
      cin >> userVals.at
   }
```

67 -5 -99 4 22

Run

Element by element vector copy

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

Figure 6.4.2: 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
   unsigned int i;                         // 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 < origPrices.size(); ++i) {
      cout << " " << origPrices.at(i);
   }
   cout << endl;

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

   return 0;
}
```

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


PARTICIPATION ACTIVITY 6.4.2: Vector copy operation.

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

```
vector<int> userVals(4); // {44, 55, 66, 77}
vector<int> newVals;     // No elements yet
```

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

Check Show answer

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

Check Show answer

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

Check Show answer

Element by element vector comparison

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

PARTICIPATION ACTIVITY 6.4.3: Vector comparing.

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

```
vector<int> vectorX(2); // {3,4}
vector<int> vectorY(5); // {3,4,0,7,8}
vector<int> vectorZ(5); // {3,4,0,6,8}
```

1) (vectorX == vectorY) will evaluate to:

- True
- False

2) Given: vectorX = vectorY; (vectorX == vectorY) will evaluate to:

- True
- False

3) (vectorZ == vectorY) will evaluate to:

- True
- False

4) (vectorZ.size() == vectorY.size()) will evaluate to:

- True
- False

CHALLENGE ACTIVITY 6.4.1: Loop-modifying or copying/comparing vectors.

539740.3879454.qx3zqy7

Start

Integer numWages is read from input, representing the number of integers to be read next. Then, the remaining integers are and stored into vector wagesList. For each element in wagesList that is less than 140:

- Output the element, followed by ": set to twice the current value" and a newline.
- Assign the element with twice the element's current value.

Ex: If the input is:

```
3
160 135 90
```

then the output is:

```
Raw wages: 160 135 90
135: set to twice the current value
90: set to twice the current value
Adjusted wages: 160 270 180
```

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

int main() {
   int numWages;
   unsigned int i;
   vector<int> wagesList;

   cin >> numWages;

   wagesList.resize(numWages);

   for (i = 0; i < wagesList.size(); ++i) {
      cin >> wagesList.at(i);
```


1 2 3

Check Next level

6.5 Multiple vectors

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

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 at the same index.

The loop's expression `(i < ctryNames.size()) && (!foundCountry)` depends on the value of the variable foundCountry. This expression prevents the loop from iterating through the entire vector once the correct country is found.

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

Figure 6.5.1: Multiple vector example: TV watching time program.

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

int main() {
   // Source: www.statista.com, 2015
   const int NUM_COUNTRIES = 5;              // Num
countries supported
   vector<string> ctryNames(NUM_COUNTRIES); //
Country names
   vector<int>    ctryMins(NUM_COUNTRIES);  // Mins
TV watched daily
   string userCountry;                       // User
defined country
   bool foundCountry = false;                //
Match to country supported
   unsigned int i;                           // Loop
index

   // Fill vector contents
   ctryNames.at(0) = "China";
   ctryMins.at(0) = 155;

   ctryNames.at(1) = "Sweden";
   ctryMins.at(1) = 154;

   ctryNames.at(2) = "Russia";
   ctryMins.at(2) = 246;

   ctryNames.at(3) = "UK";
   ctryMins.at(3) = 216;

   ctryNames.at(4) = "USA";
   ctryMins.at(4) = 274;

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

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

   return 0;
}
```

```
Enter country name: USA
People in USA watch 274
mins of TV daily.

...

Enter country name: Sweden
People in Sweden watch 154
mins of TV daily.

...

Enter country name: Brazil
Country not found; try
again.
```


©zyBooks 01/31/24 17:48 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 6.5.1: Multiple vectors.

Consider the above TV watching program involving multiple vectors.

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

zyDE 6.5.1: 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.

Load default template...

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

int main() {
   // Source: www.statis
   const int NUM_COUNTRI
   vector<string> ctryNa
   vector<int>    ctryMi
   string userCountry;
   bool foundCountry = f
   unsigned int i;

   // Fill vector conte
```

USA

Run

CHALLENGE ACTIVITY 6.5.1: Printing the sum of two vector elements.

Add each element in origList with the corresponding value in offsetAmount. Print each sum followed by a space. Ex: If origList = {40, 50, 60, 70} and offsetAmount = {5, 7, 3, 0}, print:

```
45 57 63 70
```


Learn how our autograder works

539740.3879454.qx3zqy7

```
#include <iostream>
#include <vector>
```

```
using namespace std;

int main() {
   const int NUM_VALS = 4;
   vector<int> origList(NUM_VALS);
   vector<int> offsetAmount(NUM_VALS);
   unsigned int i;

   for (i = 0; i < origList.size(); ++i) {
      cin >> origList.at(i);
   }

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


Run

CHALLENGE ACTIVITY 6.5.2: Multiple vectors: Key and value.

For any element in keysList with a value greater than 100, print the corresponding value in itemsList, followed by a space. Ex: If keysList = {42, 105, 101, 100} and itemsList = {10, 20, 30, 40}, print:

```
20 30
```

Since keysList.at(1) and keysList.at(2) have values greater than 100, the value of itemsList.at(1) and itemsList.at(2) are printed.

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int SIZE_LIST = 4;
   vector<int> keysList(SIZE_LIST);
   vector<int> itemsList(SIZE_LIST);
   unsigned int i;

   for (i = 0; i < keysList.size(); ++i) {
      cin >> keysList.at(i);
   }

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


Run

CHALLENGE ACTIVITY 6.5.3: Multiple vectors.

539740.3879454.qx3zqy7

Start

Integer inputSize is read from input. The remaining input consists of inputSize strings followed by inputSize integers. Declare string vector purchaseList and integer vector labelList, each with inputSize elements. Then:

- Read and store inputSize strings into purchaseList.
- Read and store inputSize integers into labelList.

Ex: If the input is:

```
3
avocado truffle bread
94 52 30
```


Then the output is:

```
Purchase: avocado, Label: 94
Purchase: truffle, Label: 52
Purchase: bread, Label: 30
```

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

int main() {
   int inputSize;
   unsigned int i;

   cin >> inputSize;

   /* Your code goes here */

   for (i = 0; i < purchaseList.size(); ++i) {
      cout << "Purchase: " << purchaseList.at(i) << ", ";
      cout << "Label: " << labelList.at(i) << endl;
```

1 2 3

Check Next level

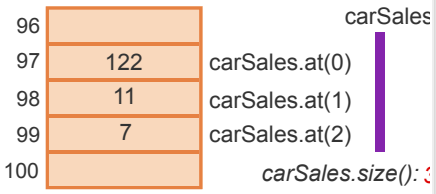
6.6 Vector resize

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 declaration. Instead, a vector's size can be set or changed while a program executes using ***resize(N)***. Ex: `highScore.resize(10)` resizes the highScores vector to have 10 elements.

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

PARTICIPATION ACTIVITY 6.6.1: Vector resize.

```
vector<int> carSales;

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

carSales.resize(3);

carSales.at(0) = 122;
carSales.at(1) = 11;
carSales.at(2) = 7;
```



Animation content:

Static figure:
Begin C++ code:
vector<int> carSales;

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

carSales.resize(3);

carSales.at(0) = 122;
carSales.at(1) = 11;
carSales.at(2) = 7;
End C++ code.
After the program's execution, elements 122, 11, and 7 are stored sequentially in carSales.

Animation captions:

1. When initially declared, the vector carSales has a size of 0.
2. Accessing any element, such as carSales.at(0), would result in a runtime error.
3. carSales.resize(3) allocates 3 elements with indices 0, 1, and 2. Elements can now be accessed using the .at() function.

The program below asks a user to indicate the number of values the user will enter, allocates that number of elements for a vector, assigns the vector's elements with user-entered values, and then displays the vector's elements.

Figure 6.6.1: Resizing a vector based on user input.

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

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

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

   userVals.resize(numVals); // Allocate elements

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

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

   return 0;
}
```

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


PARTICIPATION ACTIVITY 6.6.2: Vector resize and size functions.

Given the vector declaration:

```
vector<int> agesVctr;
```

1) Immediately after the declaration, 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 6.6.1: Vector resize.

539740.3879454.qx3zqy7

Start

Given the integer vector valElements with five elements and the input integer removeSize, resize valElements to remove removeSize elements from the vector.

Ex: If the input is `1`, then the output is:

```
Vector start 11 10 8 9 7 Vector end
Vector start 11 10 8 9 Vector end
```

Note: Assume that removeSize is a non-negative integer less than or equal to the vector's size.

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

int main() {
   vector<int> valElements(5);
   int valElementsSize = valElements.size();
   int removeSize;
   int i;

   valElements.at(0) = 11;
   valElements.at(1) = 10;
   valElements.at(2) = 8;
   valElements.at(3) = 9;
   valElements.at(4) = 7;
```

1 2 3

Check Next level

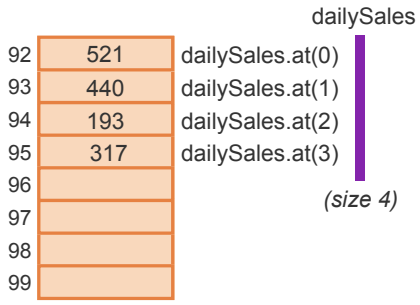
6.7 Vector push_back()

Appending items to a vector

A programmer can append a new element to the end of an existing vector using a vector's ***push_back()*** function. Ex: `dailySales.push_back(521)` creates a new element at the end of the vector dailySales and assigns that element with the value 521.

PARTICIPATION ACTIVITY 6.7.1: The vector push_back() function.

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

int main() {
   unsigned int i;
   vector<int> dailySales;

   cout << "Size before: " << dailySales.size();

   dailySales.push_back(521);
   dailySales.push_back(440);
   dailySales.push_back(193);
   dailySales.push_back(317);

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

   return 0;
}
```




```
Size before: 0 , after: 4
Contents:
521
440
193
317
```

Animation content:

Static figure:
Begin C++ code:
#include <iostream>
#include <vector>
using namespace std;

int main() {
  unsigned int i;
  vector<int> dailySales;

  cout << "Size before: " << dailySales.size();

  dailySales.push_back(521);
  dailySales.push_back(440);
  dailySales.push_back(193);
  dailySales.push_back(317);

  cout << ", after: " << dailySales.size() << endl;
  cout << "Contents:" << endl;

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

  return 0;
}
```

End C++ code.

Step 1 : When initially declared, the vector dailySales has a size of 0.
Size before: 0 is output.
Step 2: The push_back() function appends a new element to the vector.
Elements 521, 440, 193, and 317 are appended to the vector sequentially and , after: 4 is output followed by
Contents:
521
440
193
317

Animation captions:

1. When initially declared, the vector dailySales has a size of 0.
2. The push_back() function appends a new element to the vector.

PARTICIPATION ACTIVITY 6.7.2: Vector push_back().

1) If vector itemPrices has two elements with values 45, 48, what does `itemPrices.size()` return?

Check Show answer

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

Check Show answer

Vector pop_back() and back()

The following table summarizes a few common functions dealing with the back (or last element) of a vector.

Table 6.7.1: Functions on the back of a vector.

push_back()	`void push_back(const int newVal);`
	
	Append new element having value newVal.
		`// playersList initially 55, 99, 44 (size is 3)`
		`playersList.push_back(77);`
		`// Appends new element 77`
		`// playersList is now 55, 99, 44, 77 (size is 4)`

back()	`int back();`
	
	Returns vector's last element. Vector is unchanged.
		`// playersList initially 55, 99, 44`
		`cout << playersList.back(); // Prints 44`
		`// playersList is still 55, 99, 44`

pop_back()	`void pop_back();`
	
	Removes the last element.
		`// playersList is 55, 99, 44 (size 3)`
		`playersList.pop_back(); // Removes last element`
		`// playersList now 55, 99 (size 2)`
		
		`cout << playersList.back(); // Common combination of back()`
		`playersList.pop_back();`
		`// followed by pop_back()`
		`// Prints 99. playersList becomes just 55`
		
		`cout << playersList.pop_back(); // Common error:`
		
		`// pop_back() returns void`

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

The program below declares 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 6.7.1: 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 items or
type done.
Oranges
Apples
Bread
Juice
done

Enter any key for next
item.
Juice    a
Bread    a
Apples    a
Oranges    a

Done shopping.
```


PARTICIPATION ACTIVITY 6.7.3: Vector back() and pop_back() functions.

1) If itemPrices has elements 45, 22, 38, what does price = itemPrices.pop_back() assign to price? Type Error if appropriate.

Check Show answer

2) If itemPrices has elements 45, 22, 38, what does price = itemPrices.back() assign to price? Type Error if appropriate.

Check Show answer

3) If itemPrices has elements 45, 22, 38, what is the vector after itemPrices.back() is called? Type answer as: 50, 60, 70

Check Show answer

4) If itemPrices has elements 45, 22, 38, then after itemPrices.pop_back(), what does itemPrices.at(2) return? Type Error if appropriate.

Check Show answer

CHALLENGE ACTIVITY 6.7.1: Making and using vectors.

539740.3879454.qx3zqy7

Start

Write a while loop to read positive integers from input until a non-positive integer is read. For each positive integer read befo non-positive integer, add the positive integer times five to vector inputVector.

Ex: If the input is `2 5 7 1 -100`, then the output is:

```
10
25
35
5
```

Note: Positive integers are greater than 0.

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

int main() {
   vector<int> inputVector;
   int value;
   int i;

   /* Your code goes here */

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

```


1 2 3 4

Check Next level

CHALLENGE ACTIVITY 6.7.2: Removing an element from the end of a vector.

Remove the last element from vector ticketList.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int NUM_ELEMENTS = 3;
7     vector<int> ticketList(NUM_ELEMENTS);
8     unsigned int i;
9
10    for (i = 0; i < ticketList.size(); ++i) {
11       cin >> ticketList.at(i);
12    }
13
14    /* Your solution goes here  */
15
```

Run

CHALLENGE ACTIVITY 6.7.3: Reading the vector's last element.

Write a statement to print "Last mpg reading: " followed by the value of mpgTracker's last element. End with newline. Ex: If mpgTracker = {17, 19, 20}, print:

```
Last mpg reading: 20
```

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6     const int NUM_ELEMENTS = 3;
7     vector<int> mpgTracker(NUM_ELEMENTS);
8     int i;
9
10    for (i = 0; i < mpgTracker.size(); ++i) {
11       cin >> mpgTracker.at(i);
12    }
13
14    /* Your solution goes here  */
15
```

Run

6.8 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 6.8.1: 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
   unsigned int i;                     // Loop index

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

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

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

   return 0;
}
```

```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80
libc++abi.dylib: terminating with uncaught exception
of type std::out_of_range: vector
```


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(revVctr.size() - 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 `revVctr.size() - 1 - i`. The revised program is shown below.

Figure 6.8.2: 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
   unsigned int i;                     // Loop index

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

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

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

   return 0;
}
```

```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  80 70 60 50
50 60 70 80
```

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(i)` so we don't lose that element's value.

Figure 6.8.3: 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
   unsigned int i;                     // Loop index
   int tmpValue;                       // Placeholder

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

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

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

   return 0;
}
```

```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  10 20 30 40
50 60 70 80
```


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 < (revVctr.size() / 2)`.

Figure 6.8.4: 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
   unsigned int i;                     // Loop index
   int tmpValue;                       // Placeholder

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

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

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

   return 0;
}
```

```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  80 70 60 50
40 30 20 10
```


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. revVctr.size() / 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 6.8.1: Find the error in the vector reversal code.

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

  prices.at(prices.size() - 1 - i) = tmp;

}
```

2)
```
for (i = 0; i < (prices.size() / 2);
++i) {
  tmp = prices.at(i);
  prices.at(i) = prices.at(prices.size() - i);

  prices.at(prices.size() - i - 1)
= tmp;
}
```

3)
```
for (i = 0; i < (prices.size() / 2);
++i) {
  tmp = prices.at(i);

  prices.at(prices.size() - i - 1) = tmp;

prices.at(i) = prices.at(prices.size() - 1 - i);

}
```


6.9 Arrays vs. vectors

C++ supports two kinds of ordered list types.

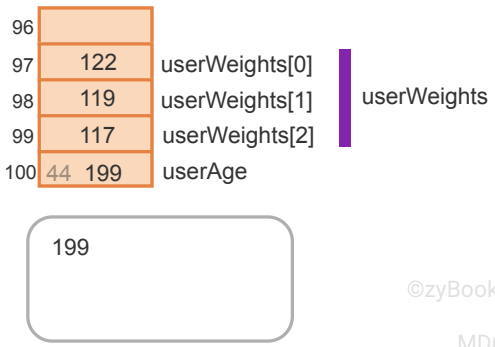
- *Arrays:* declared as `int myList[10]`, accessed as `myList[i]`.
- *Vectors:* declared as `vector<int> myList(10)`, accessed as `myList.at(i)`.

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 6.9.1: Writing to an out-of-range index using an array.

```
int userWeights[3];
int userAge;

userAge = 44;

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

// Print userAge
```



Animation content:

Static figure:
Begin C++ code:
int userWeights[3];
int userAge;

```
userAge = 44;

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

// Print userAge
End C++ code.
```


The elements of userWeights and userAge are shown as their representations in memory.
userWeights[0] is in position 97 in memory, and is set to 122.
userWeights[1] is in position 98 in memory, and is set to 119.
userWeights[2] is in position 99 in memory, and is set to 117.
userAge] is in position 100 in memory, and is set to 199.
199 is printed to the screen.

Step 1: Variable userAge is allocated a location in memory immediately after the array userWeights. userAge is assigned with 44.
The line of code reading "userAge = 44" is highlighted, and userAge is set to the value of 44 at position 100 in memory.
Step 2: Each element of array userWeights is assigned a value.
The line of code reading "userWeights[0] = 122;" is highlighted, and position 97 in memory is set to 122. The code reading "userWeights[1] = 119;" is highlighted, and position 98 in memory is set to 119. The code reading "userWeights[2] = 117;" is highlighted, and position 99 in memory is set to 117.
Step 3: 3 is out of userWeights's index range, and results in overwriting userAge's value.
The code reading "userWeights[3] = 199; // (Problematic)" is highlighted, and position 100 in memory is set to 199, overwriting userAge's original value of 44. userAge is printed to be 119.

Animation captions:

1. Variable userAge is allocated a location in memory immediately after the array userWeights. userAge is assigned with 44.
2. Each element of array userWeights is assigned a value.
3. 3 is out of userWeights's index range, and results in overwriting userAge's value.

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 6.9.2: Arrays and vectors.

Given:

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

1) arrayList[6] = 777 will yield a compiler error.
   - True
   - False

2) vectorList[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