16.1 Range-based for loop

Range-based for loop

The ***range-based for loop*** is a for loop that iterates through each element in a vector or container. A range-based for loop is also known as a ***for each loop***. The range-based loop declares a new variable that will be assigned with each successive element of a container, such as a vector, from the first element to the last element.

PARTICIPATION ACTIVITY 16.1.1: The range-based for loop declares a new variable and assigns the variable with each successive element of a container.

Animation captions:

1. The teamRoster is initialized with several players's names. A range-based for loop can be used to iterate through each vector element.
2. The range-based for loop declares a variable playerName that will be assigned with each element of the vector teamRoster.

PARTICIPATION ACTIVITY 16.1.2: Range-based for loop.

Refer to the animation above.

1) What variable is assigned with each vector element?
   - string
   - playerName
   - teamRoster

2) How many times does the for loop iterate?
   - 1
   - 2
   - 3

3) What is the value of playerName during the second iteration of the for loop?
   - Mike

- Scottie
- Toni

Improved code readability

Compared to a regular for loop, a range-based for loop decreases the amount of code needed to iterate through containers, thus enhancing code readability and clearly demonstrating the loop's purpose. A range-based for loop also prevents a programmer from writing code that incorrectly accesses elements outside of the container's range.

Figure 16.1.1: Range-based for loop decreases the amount of code needed to iterate through a vector.

Regular for loop:

```
vector<string> teamRoster;
string playerName;
unsigned int i;

// Adding player names
teamRoster.push_back("Mike");
teamRoster.push_back("Scottie");
teamRoster.push_back("Toni");

cout << "Current roster: " << endl;

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

Range-based for loop:

```
vector<string> teamRoster;

// Adding player names
teamRoster.push_back("Mike");
teamRoster.push_back("Scottie");
teamRoster.push_back("Toni");

cout << "Current roster: " << endl;

for (string playerName : teamRoster) {
   cout << playerName << endl;
}
```

PARTICIPATION ACTIVITY 16.1.3: Using range-based for loops.

Using a range-based for loop, complete the code to achieve the stated goal.

1) Calculate the sum of all values within the sensorReadings vector.

```
double sumVal = 0.0;
for (double readingVal :
sensorReadings) {
   sumVal =
                          ;
}
```


Check Show answer

2) Print each double within the vector interestRates.

```
for (                    :
interestRates) {
   cout << theRate << "%" <<
endl;
}
```


Check Show answer

3) Compute the product of all integer elements within the vector primeNumbers.

```
int primeProduct = 1;
for (int theNumber
                    ) {
   primeProduct = primeProduct
* theNumber;
}
```

Check Show answer

Modifying vector using range-based for loop

To modify a vector's elements using a range-based for loop, a programmer must declare the for loop's variable as a reference. The reference variable will refer to each vector element as the for loop iterates through the vector elements. Assigning the reference variable with a new value assigns the corresponding vector's element with that value. In the code example below, gradeVal will refer to each vector element, so the statement `gradeVal = userGrade;` assigns the vector elements with userGrade.

Figure 16.1.2: Modifying a vector using a range-based for loop.

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

int main() {
   vector<double> examGrades(4); // User's exam grades
   double averageGrade;

   // Prompt user for exam grades
   for (double &gradeVal : examGrades) {
      double userGrade;

      cin >> userGrade;
      gradeVal = userGrade;
   }

   averageGrade = 0.0;
   for (double gradeVal : examGrades) {
      averageGrade += gradeVal;
   }
   averageGrade = averageGrade / examGrades.size();

   cout << "Average grade: " << averageGrade << endl;

   return 0;
}
```

```
90.3
85.5
97
88.2
Average grade: 90.25
```


PARTICIPATION ACTIVITY 16.1.4: Range-based for loop modifying loops vector.

Given: `vector<double> sensorReadings(3)`, with element values -4.0, 5.5, 7.75. What are the resulting vector contents after executing the code below?

1)
```
for (double &sensorVal :
sensorReadings) {
   sensorVal += 10.0;
}
```

- -4.0, 5.5, 7.75
- 6.0, 15.5, 17.75
- Does not compile

2)
```
for (int &sensorVal :
sensorReadings) {
   if( sensorVal < 0.0 ) {
      sensorVal = sensorVal * -1.0;
   }
}
```


- -4.0, 5.5, 7.75
- Does not compile

3)

```
for (double sensorVal :
sensorReadings) {
   sensorVal = sensorVal - 2.0;
}
```

- -4.0, 5.5, 7.75
- -6.0, 3.5, 5.75

CHALLENGE ACTIVITY 16.1.1: Range-based for loop.

Start

Type the program's output

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

int main() {
   vector<string> vacationSpots;
   vacationSpots.push_back("Sweden");
   vacationSpots.push_back("Canada");

   cout << "Country:" << endl;

   for (string countryName : vacationSpots) {
      cout << countryName << endl;
   }
}
```

1 2 3

Check Next

Range-based for loop with auto

Programmers commonly use the auto type specifier to declare a range-based for loop's variable. In the code example below, the compiler determines gradeVal is of type double because the elements of the vector examGrades are of type double.

Figure 16.1.3: Range-based for loop with auto.

```
vector<double> examGrades;
double averageGrade;

...

averageGrade = 0.0;
for (auto gradeVal : examGrades) {
   averageGrade += gradeVal;
}
averageGrade = averageGrade / examGrades.size();
```


PARTICIPATION ACTIVITY 16.1.5: Using auto with range-based for loops.

1) Using the auto type specifier, complete the code to print each double within the vector interestRates.

```
for (                          :
interestRates) {
   cout << theRate << "%" <<
endl;
}
```

Check Show answer

2) What type will the compiler deduce for the variable pointsVal?

```
vector<int> gamePoints;

...

double avgPointsPerGame = 1;
for (auto pointsVal : gamePoints) {
   avgPointsPerGame =
avgPointsPerGame + pointsVal;
}
```

Check Show answer

16.2 List

List container

The ***list*** class defined within the C++ Standard Template Library (STL) defines a container of ordered elements, i.e., a sequence. The list class supports functions for inserting, modifying, and removing elements.

The list type is an ADT implemented as a templated class that supports different types of elements. A C++ list is implemented as a doubly-linked list. A list can be declared and created as `list<T> newList;` where T represents the list's type, such as int or string. `#include <list>` enables use of a list within a program.

A list supports insertion of elements either at the end of the list or at the beginning of the list. The push_back() function adds an element to the end of a list. Ex: `authorList.push_back("Martin");` adds "Martin" as the last element of the list. The push_front() function adds an element to the front of a list. Ex: `authorList.push_front("Rowling");` adds "Rowling" as the first list element.

PARTICIPATION ACTIVITY | 16.2.1: push_back() adds elements to end of list and push_front() add element to front of the list.

Animation captions:

1. The push_back() function adds an element, such as a String, to the end of the list.
2. The push_front() function adds an element, to the front of the list.

PARTICIPATION ACTIVITY | 16.2.2: list's push_back() and push_front() functions.

Given the following code that creates and initializes a list:

```
list<string> wordsFromFile;
wordsFromFile.push_back("fowl");
wordsFromFile.push_back("is");
wordsFromFile.push_back("the");
wordsFromFile.push_back("term");
```

1) At what index does the following statement insert the word "end"? Assume the first list element is located at index 0. Enter a number.

```
wordsFromFile.push_back("end");
```


Check Show answer

2) Given the original list initialization above, how many elements does wordsFromFile contain after the following statement?

```
wordsFromFile.push_front("fifth");
```

Check Show answer

3) Given the original list initialization above, write a statement to add the word "The" before the element "fowl".

```
wordsFromFile.
;
```

Check Show answer

Common list functions

The ***front()*** function returns the first element of a list.. The ***back()*** function returns the last element of a list. Ex: If the list exList contains the strings "four", "five", and "six", exList.front() returns the element "four" and exList.back() returns the element "six".

The ***pop_front()*** and ***pop_back()*** functions remove the first and last elements of a list, respectively.

The ***remove()*** function removes specific existing elements from the list. To identify the elements, the remove() function will start at the first list element and then traverse the list element by element until all occurrences of the specified value are found and removed.

PARTICIPATION ACTIVITY 16.2.3: pop_front() removes the first element and remove() removes specific elements from the list.

Animation captions:

1. The front( ) function returns first element in the list. The pop_front( ) function removes the first element from the list.
2. The remove( ) function removes all occurrences of a specific value from a list.

The list class implements several functions, for getting information about a list and accessing and removing a list's elements.

Table 16.2.1: Common list functions.

front()	`front()` Returns element at the front of the list.	`// Assume list is: 6, 3, 1` `exList.front();  // returns 6`
back()	`back()` Returns element at the back of the list.	`// Assume list is: 6, 3, 1` `exList.back(); // returns 1`

push_back()	`void push_back(newElement)`
	
	Adds newElement to the end of the list. List's size is increased by one.
		`// Assume list is empty`
		`exList.push_back(4);    // List is: 4`
		`exList.push_back(5);    // List is: 4, 5`

push_front()	`void push_front(newElement)`
	
	Adds newElement to the beginning of the list. List's size is increased by one.
		`// Assume list is empty`
		`exList.push_front(7);    // List is: 7`
		`exList.push_front(9);    // List is: 9, 7`

size()	`size()`
	
	Returns the number of elements in the List.
		`// Assume list is empty`
		`exList.size(); // returns 0`
		
		`exList.push_back(25);`
		`exList.push_back(13);`
		`// List is now: 25, 13`
		
		`exList.size(); // returns 2`

pop_back()	`void pop_back()`
	
	Removes element from the end of the list. List's size is decreased by one.
		`// Assume list is: 3, 11, 6, 6`
		`exList.pop_back();    // List is: 3, 11, 6`
		`exList.pop_back();    // List is: 3, 11`

pop_front()	`void pop_front()`
	
	Removes element from the front of the list. List's size is decreased by one.
		`// Assume list is: 3, 11, 6, 6`
		`exList.pop_front();    // List is: 11, 6, 6`
		`exList.pop_front();    // List is: 6, 6`

remove()	`void remove(existingElement)`
	
	Removes all occurrences of elements which are equal to existingElement. List's size is decreased by number of existingElement occurrences.
		`// Assume list is: 3, 11, 6, 6`
		`exList.remove(6); // List is now: 3, 11`
		`exList.remove(11); // List is now: 3`

PARTICIPATION ACTIVITY | 16.2.4: Using list's front(), back(), pop_front(), pop_back(), and remove() functions.

Given the following code that creates and initializes a list:

```
list<double> accelerometerValues;
accelerometerValues.push_back(9.8);
accelerometerValues.push_back(10.2);
accelerometerValues.push_back(15.4);
```


1) Complete the statement to print the first list element.

```
cout << accelerometerValues.
                            ;
```

Check Show answer

2) Complete the statement to assign currentValue with the last list element.

```
currentValue =
accelerometerValues.
                            ;
```

Check Show answer

3) Complete the statement to remove the last element from the list.

```
accelerometerValues.
                            ;
```

Check Show answer

4) Write a statement to remove the value 9.8 from the list.

```
accelerometerValues.
                            ;
```


Check Show answer

Iterating through a list

Iteration through a list necessitates keeping track of the current position in the list without using an index. An ***iterator*** is an object that points to a location in a list and can be used to traverse the list bidirectionally. The iterator allows for elements to be added, moved, and removed from any position in the list more efficiently than other containers, such as arrays and vectors. The statement `list<int>::iterator iter;` declares a list iterator for a list of integers.

A list has two iterators that point to beginning and end of the list. The ***begin()*** function returns an iterator to the first element in the list. The ***end()*** function returns an iterator to a position after the last element in the list. To iterate through the list from the beginning, a loop compares the current list position to the end position. If the end has not been reached, the position can be advanced.

Elements pointed to by the iterator are dereferenced to reveal their value. Ex: `*iter` dereferences the iterator and evaluates to the element to which the iterator referred.

PARTICIPATION ACTIVITY 16.2.5: The list class provides functions to access elements of a list using an iterator.

Animation captions:

1. The iterator, iter, is an object for iterating through list elements.
2. To start at the top, iter is assigned the value of the list's begin( ) function, which is an iterator pointing to the first element's position.
3. The list's end( ) function returns an iterator pointing to after the last element. If iter is not pointing to end, the loop condition is true. Otherwise, the condition is false.
4. Dereferencing iter reveals the string at the current position. Incrementing iter advances the iterator to the next position.

Table 16.2.2: Common list functions that use an iterator.

begin()	`begin()`
	
	Returns an iterator that points to the first element in the list.
		`// Assume exList is:  1 2`
		`exIterator = exList.begin();`
		`// exList is now:  1 2`
		`// exIterator position:  ^`

end()	`end()` Returns	

	an iterator that points to after the last element of the list.	`// Assume exList is: 1 2`
		`exIterator = exList.end();`
		`// exList is now: 1 2`
		`// exIterator position: ^`

PARTICIPATION ACTIVITY 16.2.6: Using list's begin() and end() functions to traverse and modify a list.

Answer the questions given the following code that creates and initializes a list and a list iterator.

```
list<int> numbersList;
list<int>::iterator numberIterator;

numbersList.push_back(3);
numbersList.push_back(1);
numbersList.push_back(4);

numberIterator = numbersList.begin();
```

1) What does the following condition evaluate to? (true or false)

```
numberIterator == numbersList.end();
```

Check Show answer

2) Given the original list initialization above, what is the value of numVal after executing the following statements?

```
++numberIterator;
++numberIterator;
numVal = *numberIterator;
```


Check Show answer

Inserting and removing elements using iterators

Using iterators allows for inserting and removing elements from positions other than the front and back of the list. The ***insert()*** function adds one or more elements before the specified iterator position. The ***erase()*** function can remove one or multiple elements starting at the specified iterator position. If erase() is given two iterators as arguments to dictate a range of elements to remove, the element at the first iterator is removed and all elements up until the second iterator are removed. The element at the second iterator is not removed.

Table 16.2.3: list functions for inserting and removing elements using iterators.

insert()	`insert(iteratorPosition, newElement)`
	
	Inserts newElement into the list before the iteratorPosition.
		`// Assume exList is: 1 2`
		`exIter = ++exList.begin();`
		`// exList is now: 1 2`
		`// exIter position: ^`
		
		`exList.insert(exIter, 4);`
		`// exList is now: 1 4 2`
		`// exIter position: ^`

erase()	`erase(iteratorPosition)`
	
	Removes a single element at iteratorPosition.
	
	`erase(iteratorFirst, iteratorLast)`
	
	Removes a range of elements from iteratorFirst (inclusive) to iteratorLast (exclusive).
		`// Assume exList is: 1 2 3 4`
		`exIter = exList.begin();`
		`// exList is now: 1 2 3 4`
		`// exIter position: ^`
		
		`exIter = exList.erase(exIter);`
		`// exList is now: 2 3 4`
		`// exIter position: ^`
		
		`exIter = exList.erase(exIter, --exList.end());`
		`// exList is now: 4`
		`// exIter position: ^`

PARTICIPATION ACTIVITY 16.2.7: Using list's insert() and erase() functions to modify a list.

Answer the questions given the following code that creates and initializes a list and a list iterator.

```
list<int> numbersList;
list<int>::iterator numberIterator;

numbersList.push_back(3);
numbersList.push_back(1);
numbersList.push_back(4);

numberIterator = numbersList.begin();
```

1) Given the original list and code above, what are the list contents after executing the following statement?

```
numbersList.insert(numberIterator,
6);
```

Check Show answer

2) What are the list contents after executing the following statements?

```
numberIterator++;
numbersList.erase(numberIterator);
```

Check Show answer

3) What are the list contents after executing the following statement?

```
numbersList.erase(numbersList.begin(),
--numbersList.end());
```

Check Show answer

Traversing a list with range-based for loop

A range-based for loop can be used to traverse a list without an iterator. Elements of a list can be accessed and modified with a range-based for loop.

Figure 16.2.1: Using a range-based for loop with a list.

```
list<string> teamRoster;

// Adding player names
teamRoster.push_back("Mike");
teamRoster.push_back("Scottie");
teamRoster.push_back("Toni");

cout << "Current roster: " << endl;

for (string& playerName : teamRoster) {
   cout << playerName << endl;
}
```


PARTICIPATION ACTIVITY 16.2.8: Range-based for loop list traversal.

1) When using a range-based for loop, an iterator is used to traverse a list.

   - True
   - False

2) The following code changes the values of all elements in authorList.

```
for (string& authorName :
authorList) {
   authorName = "Something";
}
```

   - True
   - False

Exploring further:

- The list class from Cplusplus.com

CHALLENGE ACTIVITY 16.2.1: List.

Start

Type the program's output

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

int main() {
   list<char> letters;
   char letter;
   int i;
   int size;

   letters.push_back('a');
   letters.push_back('b');
   letters.push_back('c');
   letters.push_back('d');

   letters.push_front('e');
   letters.pop_back();
   letters.push_back('g');
   letters.remove('c');

   size = letters.size();

   for (i = 0; i < size; ++i) {
      letter = letters.front();
      cout << letter;
      letters.pop_front();
   }
   cout << endl;

   return 0;
}
```


1 2

Check Next

16.3 Pair

The pair class

The ***pair*** class in the C++ Standard Template Library (STL) defines a container that consists of two data elements. `#include <utility>` enables use of the pair class. Many STL container classes internally use or return pair objects. Ex: a map contains key-value pairs.

The pair type is an ADT implemented as a templated class (discussed elsewhere) that supports different types values. A pair can be declared as `pair<F, S> newPair;` where F represents the pair's first element type and S represents the pair's second element type. Ex:

`pair<int, string> newPair;` declares a pair with an integer first element and a string second element.

The ***make_pair()*** function creates a pair with the specified first and second values, with which a pair variable can be assigned. Each element in a pair can be accessed and modified using the pair's `first` and `second` members.

Figure 16.3.1: Creating a pair and accessing the pair's elements.


```
#include <iostream>
#include <string>
#include <utility>

using namespace std;

int main() {
   pair<string, int> caPair;

   // 2013 population data from census.gov
   caPair = make_pair("California", 38332521);

   cout << "Population of " << caPair.first << " in 2013 was "
        << caPair.second << "." << endl ;

   // 2010 population data from census.gov
   caPair.second = 37253965;

   cout << "Population of " << caPair.first << " in 2010 was "
        << caPair.second << "." << endl ;

   return 0;
}
```

```
Population of California in 2013 was 38332521.
Population of California in 2010 was 37253965.
```

PARTICIPATION ACTIVITY 16.3.1: STL pair objects.

1) Declare a pair playAttempts with an integer as the first element and a double as the second element. Do not initialize the pair.

Check Show answer

2) Which class member accesses a pair's first element?

Check Show answer

3) Complete the statement to assign playerJim with a pair consisting of the values 25 and "Jim".

```
playerJim =
                ;
```


Check Show answer

4) Write a statement to assign teamCaptain with playerJim's second element.

Check Show answer

16.4 Map

Map container

A programmer may wish to lookup values or elements based on another value, such as looking up an employee's record based on an employee ID. The ***map*** class within the C++ Standard Template Library (STL) defines a container that associates (or maps) keys to values. `#include <map>` enables use of a map.

The map type is an ADT implemented as a templated class (discussed elsewhere) that supports different types of keys and values. Generically, a map can be declared and created as `map<K, V> newMap;` where K represents the map's key type and V represents the map's value type.

The emplace() function associates a key with the specified value. If the key does not already exist, a new entry within the map is created. If the key already exists, the associated map entry is not updated. Thus, a map associates at most one value for a key.

The at() function returns the value associated with a key, such as `statePopulation.at("CA")`.

A map entry can be updated by assigning the entry with a new value. Ex:

```
statePopulation.at("CA") = 39776830;
```

PARTICIPATION ACTIVITY 16.4.1: A map allows a programmer to map keys to values.

Animation captions:

1. The emplace() function associates a key with the specified value.
2. The at() function returns the value associated with a key.

PARTICIPATION ACTIVITY 16.4.2: Updating the value associated with a key.

Animation content:

```
undefined
```

Animation captions:

1. A map entry can be updated by assigning the entry with a new value. statePopulation.at("AZ") accesses the entry for "AZ", and the element is assigned with 6871809.

PARTICIPATION ACTIVITY 16.4.3: Basic map operations: emplace() and at().

Given the following code that creates and initializes a map:

```
map<string, int> playerScores;
playerScores.emplace("Jake", 14);
playerScores.emplace("Pat", 26);
playerScores.emplace("Hachin", 60);
playerScores.emplace("Michiko", 21);
playerScores.emplace("Pat", 31);
```

1) Write a statement to add a mapping for a player named Kira with a score of 1.

Check Show answer

2) Write a statement to assign Hachin's score to a variable named highScore.

Check Show answer

3) What value will playerScores.at("Pat") return.

Check Show answer

4) Write a single statement to update Jake's score to the value 34.

Check Show answer

Determining if a key exists

If the map does not contain the specified key, the at() function throws an out_of_range exception. A programmer can use the count() function to check if a map contains the specific key. count() returns 1 if the key exists and 0 otherwise. In the program above, `statePopulation.count("NY")` would return 0.

zyDE 16.4.1: Use count() to check if the map contains the use specified key.

This program uses a map to associate a race distance in kilometers with a runner's race
If a race distance does not exist, the program throws an exception. Modify the program
the count() function to check if the runner has run a race of the specified distance, and
a message of "No race of the specified distance exists."

RunDistTimeMap.cpp Load default templ

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main () {
   map<int, double> raceTimes;
   int userDistKm;

   raceTimes.emplace(5, 23.14);
   raceTimes.emplace(15, 78.5);
   raceTimes.emplace(25, 120.75);

   cout << "Enter race distance in km (0 to exit): " << endl;
   cin >> userDistKm;

   while(userDistKm != 0) {
```


5 15 10 0

Run

PARTICIPATION ACTIVITY 16.4.4: Determining if map contains a key.

Given the code below, determine the result of each expression.

```
map<int, double> raceTimes;

raceTimes.emplace(5, 23.14);
raceTimes.emplace(15, 78.5);
raceTimes.emplace(25, 120.75);
```

1) raceTimes.count(10)

- 1
- 0

2) raceTimes.count(5)

- 1
- 0

3) raceTimes.at(7)

- 0
- exception

Common map functions

The map class implements several functions for accessing and modifying map entries.

Table 16.4.1: Common map functions.

emplace()	`emplace(key, value)` Associates key with specified value. If key already exists, the map entry is not changed.	`// map originally empty`
		`exMap.emplace("Tom", 14);`
		`// map now: Tom->14,`
		`exMap.emplace("John", 86);`
		`// Map now: Tom->14, John->86`

at()	`at(key)`
	Returns the entry associated with key. If key does not exist, throws an out_of_range exception.
		`// Assume map is: Tom->14, John->86, Mary->13`
		`exMap.at("Mary") = 25;`
		`// Map now: Tom->14, John->86, Mary->25`
		`exMap.at("Tom")  // returns 14`
		`exMap.at("Bob")  // throws exception`

count()	`count(key)`
	Returns 1 if key exists, otherwise returns 0.
		`// Assume map is: Tom->14, John->86, Mary->13`
		`exMap.count("Tom")  // returns 1`
		`exMap.count("Bob")  // returns 0`

erase()	`erase(key)`
	Removes the map entry for the specified key if the key exists.
		`// Assume map is: Tom->14, John->86, Mary->13`
		`exMap.erase("John");`
		`// map is now: Tom->14, Mary->13`

clear()	`clear()`
	Removes all map entries.
		`// Assume map is: Tom->14, John->86, Mary->13`
		`exMap.clear();`
		`// map is now empty`

PARTICIPATION ACTIVITY 16.4.5: Common map functions.

Given the following code that creates and initializes a map:

```
map<string, int> exMap;
exMap.emplace("Tom", 14);
exMap.emplace("John", 26);
exMap.emplace("Mary", 13);
exMap.emplace("Hans", 90);
exMap.emplace("Franz", 88);
```

Which of the following operations modify the map?

1) exMap.emplace("Mary", 17);

- ○ True
- ○ False

2) exMap.emplace("Frederique", 36);

- ○ True
- ○ False

3) exMap.erase("Tom");

- True
- False

4) exMap.erase("john");

- True
- False

5) exMap.clear();

- True
- False

CHALLENGE ACTIVITY 16.4.1: Map functions.

Start

Type the program's output

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main () {
   map<string, string> airportCode;

   airportCode.emplace("LNZ", "Linz, Austria");
   airportCode.emplace("ALB", "Albany, USA");
   airportCode.emplace("LOH", "Loja, Ecuador");

   cout << "LNZ: " << airportCode.at("LNZ") << endl;

   airportCode.emplace("LOH", "Lome, Togo");

   cout << "LOH: " << airportCode.at("LOH") << endl;
}
```

1 2

Check Next

[ ] operator

The map class' [ ] operator can be used to add map entries (Ex: ***`exMap["Dan"] = 25;`****) and access map entries (Ex:* ***`exMap["Dan"];`****). However, if a map does not contain the specified key, the [ ] operator creates a new entry in the map with the key and the default value for the map's value type. Ex: If the key "Bob" does not exist in the map,* ***`myVal = exMap["Bob"];`*** *creates a new map entry with key "Bob" and value 0. When using the [ ] operator, if creating a map entry with a default value is not desired, a programmer should first check that the key exists before accessing the map with that key. This material uses emplace() and at() to add and access map entries, but the examples above can be modified to use the [ ] operator.*

emplace()

Like many C++ containers, the map class has both an insert() function and an emplace() function to add new entries to a map. insert() requires that an entry be explicitly constructed before insertion. emplace() does not have this requirement, and constructs entries of the correct type upon insertion by automatically calling an appropriate constructor. This material uses emplace() for the map class as an intuitive way of adding a pair entry to a map without explicitly constructing the pair beforehand.

Exploring further:

- The map class from Cplusplus.com.

16.5 Set

Set container

The ***set*** class defined within the C++ Standard Template Library (STL) defines a collection of unique elements. The set class supports functions for adding and removing elements, as well as querying if a set contains an element. For example, a programmer may use a set to store employee names and use that set to determine which customers are eligible for employee discounts.

The set type is an ADT implemented as a templated class that supports different types of elements. A set can be declared as `set<T> newSet;` where T represents the set's type, such as int or string. `#include <set>` enables use of a set within a program.

PARTICIPATION ACTIVITY | 16.5.1: A set's insert(), erase(), and count() functions add an item, remove an item, and check if an item exists within the set.

Animation content:

`undefined`

Animation captions:

1. The insert() function adds an item such as a string to a set.
2. The count() function returns 1 if an item is in the set, and otherwise returns 0.
3. The erase() method removes an item from a set.

PARTICIPATION ACTIVITY | 16.5.2: set operations: insert(), erase(), and count().

Given the following code that creates and initializes a set:

```
set<int> employeeIDs;
employeeIDs.insert(1001);
employeeIDs.insert(1002);
employeeIDs.insert(1003);
```

1) Write a statement that adds an employee ID 1337.

Check Show answer

2) Write a statement that removes the employee ID 1002.

Check Show answer

3) What value will employeeIDs.count(1001) return?

Check Show answer

insert() and erase() functions

The insert() function does not add duplicate elements to a set. If a programmer tries to add a duplicate element, the insert() function returns a pair object containing an iterator at the existing element's location and false. Otherwise, insert() returns a pair object with an iterator at the added element location and true.

The erase() function only removes elements that exist within the set. If the element exists, the erase() function removes the element and returns 1. Otherwise, erase() returns 0.

PARTICIPATION ACTIVITY 16.5.3: A set's insert() and erase() functions return a value to indicate the operation's failure or success.

Animation content:

undefined

Animation captions:

1. The insert() function adds an item only if the item does not already exist.
2. insert() returns a pair consisting of an iterator at the existing element's location and a boolean value indicating if the item was added to the set.
3. The erase() function returns 1 if item removal was successful, otherwise returns 0.

PARTICIPATION ACTIVITY 16.5.4: Return value of set's insert() and erase() functions.

Given the following code that creates and initializes a set:

```
set<char> guessedLetters;
guessedLetters.insert('e');
guessedLetters.insert('t');
guessedLetters.insert('a');
```

1) What value will guessedLetters.erase('s') return?

Check Show answer

2) What value will (guessedLetters.insert('e')).second return?

Check Show answer

3) Write a single statement that adds to guessedLetters the variable favoriteLetter only if favoriteLetter does not already exist in the set.

Check Show answer

zyDE 16.5.1: Use only erase() to check if the user's guess is in the set.

The program below uses a set to store the numbers a user has to guess to win. The pro uses both count() and erase() to remove correct guesses from the set, and the game en when user guesses all numbers (i.e., the set is empty). Modify the program to only use e and not count(). Hint: consider the return value of erase().

GuessTheNumbers.cpp Load default templ

```
#include <iostream>
#include <set>
#include <string>

using namespace std;

int main() {
   set<int> numbersToGuess;
   int userGuess;

   numbersToGuess.insert(3);
   numbersToGuess.insert(5);
   numbersToGuess.insert(1);

   cout << "Enter a number between 1 to 10 (0 to exit): ";
   cin >> userGuess;

   while (userGuess != 0) {
```


1 2 5 3

Run

Common set operations

The set class implements several functions for modifying and querying the status of the set.

Table 16.5.1: Common set functions.

insert()	`insert(element)`
	If element does not exist, adds element to the set and returns pair object (iterator to element location, true). If element already exists, returns pair object (iterator to element location, false).
		`// Set originally empty`
		`exSet.insert("Kasparov"); // returns pair (iterator at element "Kasparov", true)`
		`// Set is now: Kasparov`
		`exSet.insert("Kasparov"); // returns pair (iterator at element "Kasparov", false) (element "Kasparov" already exists)`
		`// Set is unchanged`

erase()	`erase(element)`
	If element exists, removes element from the set and returns 1. If the element does not exist, returns 0.
		`// Assume Set is: Kasparov, Fisher`
		`exSet.erase("Fisher");  // returns 1 (element "Fisher" exists)`
		`// Set is now: Kasparov`
		`exSet.erase("Carlsen"); // returns 0 (element "Carlsen" does not exist)`
		`// Set is unchanged`

count()	`count(element)`
	Returns 1 if element exists, otherwise returns 0.
		`// Assume Set is: Kasparov, Fisher, Carlsen`
		`exSet.count("Carlsen")  // returns 1`
		`exSet.count("Anand")  // returns 0`

size()	`size()`
	Returns the number of elements in the set.
		`// Set originally empty`
		`exSet.size(); // returns 0`
		`exSet.insert("Nakamura");`
		`exSet.insert("Carlsen");`
		`// Set is now: Nakamura, Carlsen`
		`exSet.size(); // returns 2`

Exploring further:

- The set class from Cplusplus.com

CHALLENGE ACTIVITY 16.5.1: Using a set to define unique elements.

Start

Type the program's output

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

int main() {
   set<string> astronauts;
   string userInput;

   cin >> userInput;

   while (userInput != "done") {
      if (astronauts.insert(userInput).second) {
            cout << "i" << endl;
      }
      else {
            cout << "n" << endl;
      }
      cin >> userInput;
   }

   cout << "Size: " << astronauts.size() << endl;

   return 0;
}
```

Input

```
Ride
Jemison
Ride
Glenn
Lovell
Shepard
Yang
done
```

Output

1 2

Check Next

16.6 Queue

queue class

The ***queue*** class defined within the C++ Standard Template Library (STL) defines a container of ordered elements that supports element insertion at the tail and element retrieval from the head.

The queue type is an ADT implemented as a templated class that supports different types of elements. A queue can be declared as `queue<T> newQueue;` where T represents the element's type, such as int or string. `#include <queue>` enables use of a queue within a program.

A queue's ***push()*** function adds an element to the tail of the queue and increases the queue's size by one. A queue's ***front()*** function returns the element at the head of the queue. And, a queue's ***pop()*** function removes the element at the head of the queue. If a queue is empty, front() results in undefined behavior, thus one should check a queue's size before using front().

PARTICIPATION ACTIVITY 16.6.1: queue: push() adds an element to the tail of the queue, front() returns element at the head of the queue, and pop() removes the element at the head of the queue.

Animation captions:

1. The push() functions adds an element, such as a string, to the tail of the queue.
2. The front() function returns the element at the head of the queue. The pop() function removes the element at the head of the queue.

PARTICIPATION ACTIVITY 16.6.2: Using queue's push(), front(), and pop() functions to insert and retrieve elements.

Answer the questions given the following code that creates and initializes a queue.

```
queue<int> ordersQueue;

ordersQueue.push(351);
ordersQueue.push(352);
ordersQueue.push(353);
```

1) Complete the statement to add the value 354 to the tail of the queue.

```
ordersQueue.
;
```

Check Show answer

2) Complete the statement to print the element at the head of the queue.

```
cout << ordersQueue.
                    << endl;
```

Check Show answer

3) Given the original queue initialization above, what is the size of the queue *after* three calls to ordersQueue.pop()?

Check Show answer

Common queue functions

The queue class has several functions for inserting and removing elements and examining the contents of a queue.

Table 16.6.1: Common queue functions.

push()	`push(newElement)`
	
	Adds newElement element to the tail of the queue. The queue's size increases by one.
		`// Assume exQueue is: "down" "right" "A"`
		`exQueue.push("B");`
		`// exQueue is now: "down" "right" "A" "B"`

pop()	`pop()`
	
	Removes the element at the head of the queue.
		`// Assume exQueue is: "down" "right" "A"`
		`exQueue.pop(); // Removes "down"`
		`// exQueue is now: "right" "A"`

front()	`front()`
	
	Returns, but does not remove, the element at the head of the queue. If the queue is empty, the behavior is not defined.
		`// Assume exQueue is: "down" "right" "A"`
		`exQueue.front(); // Returns "down"`
		`// exQueue is still: "down" "right" "A"`

back()	`back()`
	
	Returns, but does not remove, the element
		`// Assume exQueue is: "down" "right" "A"`
		`exQueue.back(); // Returns "A"`
		`// exQueue is still: "down" "right" "A"`

	at the tail of the queue.	

size()	`size()`
	
	Returns the size of the queue.
		`// Assume exQueue is: "down"`
		`exQueue.size(); // Returns 1`
		
		`exQueue.pop();  // exQueue is empty`
		`exQueue.size(); // Returns 0`

CHALLENGE ACTIVITY 16.6.1: Queue.

Start

Type the program's output

```
#include <iostream>
#include <queue>
#include <string>

using namespace std;

int main(){
   queue<string> colors;

   colors.push("yellow");
   colors.push("red");
   colors.push("brown");
   colors.pop();
   colors.pop();

   cout << colors.front() << endl;

   return 0;
}
```

1 2

Check Next

Exploring further:

- The queue class from Cplusplus.com

16.7 Deque

deque class

The ***deque*** (pronounced "deck") class defined within the C++ Standard Template Library (STL) defines a container of ordered elements that supports element insertion and removal at both ends (i.e., at the head and tail of the deque).

A deque can be declared as `deque<T> newDeque;` where T represents the element's type, such as int or string. `#include <deque>` enables use of a deque within a program.

deque's ***push_front()*** function adds an element at the head of the deque and increases the deque's size by one. The push_front() function shifts elements in the deque to make room for the new element. deque's ***front()*** function returns the element at the head of the deque. And, deque's ***pop_front()*** function removes the element at the head of the deque. If the deque is empty, front() results in undefined behavior.

The push_front(), front(), and pop_front() functions allow a deque to be used as a stack. A ***stack*** is an ADT in which elements are only added or removed from the top of a stack.

PARTICIPATION ACTIVITY 16.7.1: deque: push_front() function adds an element at the head, front() function returns the element at the head, and pop_front() function removes the element at the head.

Animation captions:

1. The push_front() function adds an element, such as a string, at the head of the deque.
2. The front() function returns the element at the head of the deque. The pop_front() function removes the element at the head of the deque.

PARTICIPATION ACTIVITY 16.7.2: Use deque's push_front(), front(), and pop_front() functions to insert and retrieve elements.

Given the following code that creates and initializes a deque.


```
deque<string> jobsDeque;

jobsDeque.push_front("Filter");
jobsDeque.push_front("Download");
jobsDeque.push_front("Process");
```

1) What is the value of the element at the head of the deque?

Check Show answer

2) What is the value of the element at the tail of the deque?

Check Show answer

3) Complete the statement to add the value "Draw" at the head of the deque.

```
jobsDeque.          ;
```

Check Show answer

4) Complete the statement to print the element at the head of the deque.

```
cout << jobsDeque.
                    << endl;
```

Check Show answer

5) Complete the statement to remove the element at the head of the deque.

```
cout << jobsDeque.
                    ;
```

Check Show answer

Common deque functions

The deque class has functions for inserting and removing elements at both ends of the deque and examining the contents of the deque.

Table 16.7.1: Common deque functions.

push_front()	`push_front(newElement)`	`// Assume exDeque is: 3 5 6`
		`exDeque.push_front(1);`
		`// exDeque is now:  1 3 5 6`

	Adds newElement element at the head of the deque. The deque's size increases by one.	

push_back()	`push_back(newElement)`
	
	Adds newElement element at the tail of the deque. The deque's size increases by one.
		`// Assume exDeque is: 3 5 6`
		`exDeque.push_back(7);`
		`// exDeque is now: 3 5 6 7`

pop_front()	`pop_front()`
	
	Removes the element at the head of the deque.
		`// Assume exDeque is: 3 5 6`
		`exDeque.pop_front();`
		`// exDeque is now: 5 6`

pop_back()	`pop_back()`
	
	Removes the element at the tail of the deque.
		`// Assume exDeque is: 3 5 6`
		`exDeque.pop_back();`
		`// exDeque is now: 3 5`

front()	`front()`
	
	Returns, but does not remove, the element at the head of the deque. If the deque is empty, the behavior is not defined.
		`// Assume exDeque is: 3 5 6`
		`exDeque.front(); // Returns 3`
		`// exDeque is still: 3 5 6`

back()	`back()`
	
	Returns, but does not remove, the element at the tail of the deque. If the deque is empty, the behavior is not defined.
		`// Assume exDeque is: 3 5 6`
		`exDeque.back(); // Returns 6`
		`// exDeque is still: 3 5 6`

size()	`size()`
	
	Returns the size of the deque
		`// Assume exDeque is: 3`
		`exDeque.size(); // Returns 1`
		
		`exDeque.pop_front(); // exDeque is empty`
		`exDeque.size(); // Returns 0`

CHALLENGE ACTIVITY 16.7.1: Enter the output for deque.

Start

Type the program's output

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

int main() {
   deque<string> groceries;

   groceries.push_front("cereal");
   groceries.push_front("milk");
   groceries.push_front("lettuce");

   cout << "1. " << groceries.front() << endl;
   groceries.pop_front();
   cout << "2. " << groceries.front() << endl;

   return 0;
}
```


1 2

Check Next

Exploring further:

- The deque class from Cplusplus.com

16.8 find() function

The find() function

The ***find()*** function seeks a specific value in a range of elements. find() is defined in the algorithms library of the Standard Template Library (STL) , thus adding `#include <algorithm>` enables use of find(). find() can be used with different STL containers such as vectors and lists. The container elements can be any C++ data type that supports use of the equality operator (==).

The function's notation is `find(iteratorFirst, iteratorLast, value)`:

- iteratorFirst: An iterator to the range's first element
- iteratorLast: An iterator to one element past the range's last element
- value: The value of the element to be found within the range

Note that iteratorFirst's element is checked, but iteratorLast's element is not.

find() returns an iterator to the first element in the range that is equal to value. If value is not found, find() returns iteratorLast.

PARTICIPATION ACTIVITY 16.8.1: The find() function.

Animation captions:

1. find() takes 3 arguments: a beginning iterator, an ending iterator, and the value being sought.
2. find() searches the range sequentially and returns an iterator to the first element in the range equal to the value.
3. If an element with the sought value is not found, find( ) returns an iterator to the end of the range.

PARTICIPATION ACTIVITY 16.8.2: Using find() to locate elements in a list.

Given the following code that creates and initializes a list.

```
list<string> firstNames;

firstNames.push_back("Johnny");
firstNames.push_back("Katie");
firstNames.push_back("Daniel");
firstNames.push_back("Kushal");
firstNames.push_back("Damian");
firstNames.push_back("Katie");
```

1) Complete the statement to find where Johnny exists in the list.

```
iter =
find(firstNames.begin(),
firstNames.end(),
                    );
```

Check Show answer

2) What is returned by `find(firstNames.begin(), firstNames.end(), "Daniel");`?

```
firstNames.begin() +
```

Check Show answer

3) What is returned by `find(firstNames.begin(), firstNames.end(), "Katie");`?

```
firstNames.begin() +
```

Check Show answer

4) What is returned by `find(firstNames.begin(), firstNames.end(), "Alex");`?

```
firstNames.
;
```

Check Show answer

The find_if() function

The ***find_if()*** function is a variation of the find() function, defined in the STL algorithms library, that searches a range for an element that satisfies a boolean condition. Adding `#include <algorithm>` enables the use of find_if().

The function's notation is `find_if(iteratorFirst, iteratorLast, boolFunction)`:

- iteratorFirst: An iterator to the range's first element
- iteratorLast: An iterator to one element past the range's last element
- boolFunction: A function that takes one argument (of the container type) and returns true or false

find_if() returns an iterator to the first element in the range that causes boolFunction to return true. If no element satisfies the condition, find_if() returns iteratorLast. find_if() goes through the range sequentially and passes each element to boolFunction one-by-one.

Figure 16.8.1: Using the find_if() function to find the first string formatted like ##.#.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

using namespace std;

bool isFormattedNum(string num) {
   bool isMatch;

   isMatch = false;

   if (num.size() == 4) {
      if (isdigit(num.at(0)) && isdigit(num.at(1)) &&
          isdigit(num.at(3)) && num.at(2) == '.') {
         isMatch = true;
      }
   }

   return isMatch;
}

int main() {
   vector<string> numberStrings;
   vector<string>::iterator iter;

   // Adding strings
   numberStrings.push_back("fifty two point one");
   numberStrings.push_back("2.42");
   numberStrings.push_back("63.2");
   numberStrings.push_back("89.0");

   iter = find_if(numberStrings.begin(), numberStrings.end(), isFormattedNum);

   if (iter != numberStrings.end()) {
      cout << "The first ##.# formatted string is " << *iter << "." << endl;
   }
   else {
      cout << "No ##.# formatted strings found." << endl;
   }

   return 0;
}
```

```
The first ##.# formatted string is 63.2.
```


PARTICIPATION ACTIVITY 16.8.3: The find_if() function.

Refer to the find_if() example above.

1) How is the boolean function isFormattedNum() passed to the find_if() function?

isFormattedNum

- isFormattedNum()

2) What element would `find_if(numberStrings.begin() + 2, numberStrings.end(), isFormattedNum)` point to?

- 63.2
- 89.0

3) If another defined function called noDigits() existed that identified strings with no digits (so "xyz" would match, but "xy2" would not), what would be returned by `find_if(numberStrings.begin(), numberStrings.end(), noDigits)`?

- numberStrings.begin()
- numberStrings.end()

16.9 sort() function

The sort() function

The ***sort()*** function arranges a container's elements in ascending order. sort() is defined in the Standard Template Library's (STL) algorithms library. Adding `#include <algorithm>` enables the use of sort(). sort() can be used with numerous STL containers, such as vectors and arrays. The container elements can be any C++ data type that supports use of the less than operator (<).

sort() takes two parameters to specify the range of elements to be sorted, calling sort() as: `sort(iteratorFirst, iteratorLast)`:

- iteratorFirst: An iterator to the range's first element
- iteratorLast: An iterator to one element past the range's last element

PARTICIPATION ACTIVITY 16.9.1: Using sort().

Animation captions:

1. sort() takes 2 arguments, iterators to the beginning and end of a range.
2. sort( ) arranges elements in ascending order.

PARTICIPATION ACTIVITY 16.9.2: Using sort() to sort a vector's elements.

Given the following code that creates and initializes a vector.


```
vector<string> firstNames;

firstNames.push_back("Johnny");
firstNames.push_back("Katie");
firstNames.push_back("Daniel");
firstNames.push_back("Kushal");
firstNames.push_back("Damian");
firstNames.push_back("Katherine");
```

1) Complete the statement to arrange the vector in ascending order.

```
[ ]
(firstNames.begin(),
firstNames.end());
```

Check Show answer

2) What is the last element of the vector after `sort(firstNames.begin(), firstNames.end());`?

Check Show answer

3) Complete the statement to arrange the first four elements of the vector in ascending order.

```
sort(firstNames.begin(),
firstNames.begin() +
[ ]);
```


Check Show answer

4) Write the first element in the range that was sorted of the vector after `sort(firstNames.begin() + 2, firstNames.end());`

Check Show answer

Passing custom comparison functions to sort()

A programmer can use a custom comparison function when sorting a container's elements. The comparison function is passed as the third argument to the sort() function. Ex: `sort(numsVec.begin(), numsVec.end(), compareFunction)`. compareFunction is a function that takes two elements as arguments and compares those elements. The function returns true if the first element should appear before the second element when sorted, and false otherwise. Ex: To sort a vector's element in descending order, the compare function would return true if the function's first argument is greater than the second argument.

PARTICIPATION ACTIVITY 16.9.3: Using sort() with a custom comparison function to sort in descending order.

Animation captions:

1. A custom comparison function can be passed to sort( ) as the first argument.
2. sortDescending is passed to sort(). sort() uses sortDescending to compare the vector's elements within the sort algorithm.
3. The sortDescending function returns true if the first parameter is greater than the second, indicating the first parameter should be ordered before the second parameter when sorted.

PARTICIPATION ACTIVITY 16.9.4: sort() with a custom comparison function.

Refer to the animation above.

1) What is the return type of function sortDescending()?
   - double
   - bool

2) For a custom comparison function, if the first argument should appear before the second argument when sorted, what should the function return?
   - true
   -

false

Sorting objects of custom data types

To compare objects of custom class types, a programmer can overload the less than operator for comparing two objects of the class. Good practice is to define the less operator so objects are sorted in ascending order by default.

A program can also define custom comparison functions, allowing objects to be sorted in multiple ways. Ex: For a Student class, multiple comparison functions can be defined so a list of Students can be sorted by first name, last name, or ID number. In this case, the sort() call takes three arguments with the compare function being the third argument.

Figure 16.9.1: Using the sort() function and custom comparison functions to sort a vector of students in different ways.

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

using namespace std;

class Student {
   public:
      Student(string name1, string name2, int num);
      void SetStudentVals(string name1, string name2, int num);
      string GetFirstName();
      string GetLastName();
      int GetIdNum();
      void Print();
   private:
      string firstName;
      string lastName;
      int idNum;
};

Student::Student(string name1, string name2, int num) {
   firstName = name1;
   lastName = name2;
   idNum = num;
}

void Student::SetStudentVals(string name1, string name2, int num) {
   firstName = name1;
   lastName = name2;
   idNum = num;
}

string Student::GetFirstName() {
   return firstName;
}

string Student::GetLastName() {
   return lastName;
}

int Student::GetIdNum() {
   return idNum;
```

```
}

void Student::Print() {
   cout << firstName << " " << lastName << " (" << idNum << ")" << endl;
}


bool sortByFirstName (Student student1, Student student2) {
   return student1.GetFirstName() < student2.GetFirstName();
}

bool sortByLastName (Student student1, Student student2) {
   return student1.GetLastName() < student2.GetLastName();
}

bool sortByIdNum (Student student1, Student student2) {
   return student1.GetIdNum() < student2.GetIdNum();
}


int main() {
   unsigned int i;
   vector<Student> studentsVector;

   Student newStudent ("Sammy", "Hill", 1357);
   studentsVector.push_back(newStudent);

   newStudent.SetStudentVals("Jack", "Casella", 2468);
   studentsVector.push_back(newStudent);

   newStudent.SetStudentVals("Greta", "Phillips", 1928);
   studentsVector.push_back(newStudent);
   // studentsVector: Sammy Hill (id: 1357), Jack Casella (id: 2468), Greta Phillips
(id: 1928)

   cout << "Sorted by first name:"<< endl;
   sort(studentsVector.begin(), studentsVector.end(), sortByFirstName);
   // studentsVector: Greta Phillips (id: 19283), Jack Casella (id: 2468), Sammy Hill
(id: 1357)

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

   cout << "Sorted by last name:" << endl;
   sort(studentsVector.begin(), studentsVector.end(), sortByLastName);
   // studentsVector: Jack Casella (id: 2468), Sammy Hill (id: 1357), Greta Phillips
(id: 1928)

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

   cout << "Sorted by id number:" << endl;
   sort(studentsVector.begin(), studentsVector.end(), sortByIdNum);
   // studentsVector: Sammy Hill (id: 1357), Greta Phillips (id: 1928), Jack Casella
(id: 2468)

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

   return 0;
}
```

```
Sorted by first name:
Greta Phillips (1928)
Jack Casella (2468)
Sammy Hill (1357)

Sorted by last name:
Jack Casella (2468)
Sammy Hill (1357)
Greta Phillips (1928)

Sorted by id number:
Sammy Hill (1357)
Greta Phillips (1928)
Jack Casella (2468)
```


PARTICIPATION ACTIVITY 16.9.5: Using sort() with custom data types.

Given `Student newStudent ("Katrin", "Coggs", 2472);` has been added to the `studentsVector` in the example above.

1) At which index will student Katrin Coggs be after executing the following statement?

```
sort(studentsVector.begin(),
studentsVector.end(),
sortByLastName);
```

- 0
- 1

2) At which index will student Katrin Coggs be after executing the following statement?

```
sort(studentsVector.begin(),
studentsVector.end(),
sortByFirstName);
```

- 1
- 2

3) At which index will student Jack Casella be after executing the following statement?

```
sort(studentsVector.begin(),
```

```
studentsVector.end(),
sortByIdNum);
```

- 2
- 3

16.10 LAB: Grocery shopping list (list)

Given a **ListItem** class, complete main() using the built-in list type to create a linked list called **shoppingList**. The program should read items from input (ending with -1), adding each item to shoppingList, and output each item in shoppingList using the PrintNodeData() function.

Ex. If the input is:

```
milk
bread
eggs
waffles
cereal
-1
```

the output is:

```
milk
bread
eggs
waffles
cereal
```

LAB ACTIVITY | 16.10.1: LAB: Grocery shopping list (list) | 0 / 10

Current file: **main.cpp** Load default template...


```
1  #include "ListItem.h"
2  #include <string>
3  #include <list>
4  #include <iostream>
5
6  using namespace std;
7
8  int main (int argc, char* argv[]) {
9     // TODO: Declare a list called shoppingList of type ListItem
10
```

```
11    string item;
12
13    // TODO: Read inputs (items) and add them to the shoppingList list
14    //       Read inputs until a -1 is input
15
16
17    // TODO: Print the shoppingList list using the PrintNodeData() function
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Signature of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

16.11 LAB: Student grades (map)

Given a map pre-filled with student names as keys and grades as values, complete main() by reading in the name of a student, outputting their original grade, and then reading in and outputting their new grade.

Ex: If the input is:

```
Quincy Wraight
73.1
```

the output is:

```
Quincy Wraight's original grade: 65.4
Quincy Wraight's new grade: 73.1
```

LAB ACTIVITY | 16.11.1: LAB: Student grades (map) 0 / 10

main.cpp

Load default template...

```
#include <map>
#include <string>
#include <iostream>

using namespace std;

int main (int argc, char* argv[]) {
   string studentName;
   double studentGrade;

   map<string, double> studentGrades;

   // Students' grades (pre-entered)
   studentGrades.emplace("Harry Rawlins", 84.3);
   studentGrades.emplace("Stephanie Kong", 91.0);
   studentGrades.emplace("Shailen Tennyson", 78.6);
   studentGrades.emplace("Quincy Wraight", 65.4);
   studentGrades.emplace("Janine Antinori", 98.2);
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Signature of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

16.12 LAB: Ticketing service (queue)

Given main(), complete the program to add people to a queue. The program should read in a list of people's names including "You" (ending with -1), adding each person to the `peopleInQueue` queue. Then, remove each person from the queue until "You" is at the head of the queue. Include print statements as shown in the example below.

Ex. If the input is:

```
Zadie Smith
Tom Sawyer
You
Louisa Alcott
-1
```

the output is:

```
Welcome to the ticketing service...
You are number 3 in the queue.
Zadie Smith has purchased a ticket.
You are now number 2
Tom Sawyer has purchased a ticket.
You are now number 1
You can now purchase your ticket!
```

LAB ACTIVITY | 16.12.1: LAB: Ticketing service (queue) 0 / 10

main.cpp Load default template...

```
1  #include <queue>
2  #include <iostream>
3
4  using namespace std;
5
6  int main (int argc, char* argv[]) {
7     string personName = "";
8     int counter = 0;
9     int youPosition;
10
11    queue<string> peopleInQueue;
12
13    getline(cin, personName);
14    while (personName != "-1") {
15       // TODO: Add personName to peopleInQueue
```

```
16    //       determine position of "You" (youPosition)
17
```

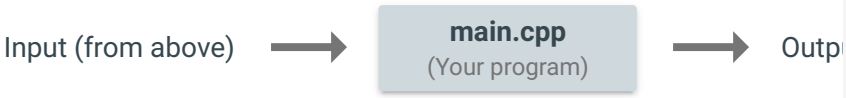
Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Signature of your work What is this?

History of your effort will appear here once you begin working on this zyLab.