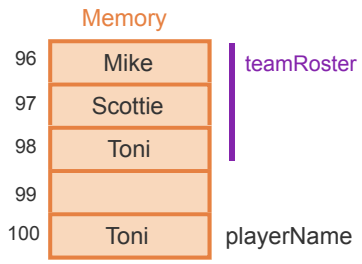
19.1 Range-based for loop

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

```
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;
}
```




```
Current roster:
Mike
Scottie
Toni
```

Animation content:

Static figure:
Begin C++ code.
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;
}
```
End C++ code.

A box labeled "Memory" contains five empty memory addresses, labeled from 96 to 100, consecutively. The output console is empty.

Step 1: The teamRoster is initialized with several players's names. A range-based for loop can be used to iterate through each vector element. The line of code, vector<string> teamRoster;, is highlighted and memory location 96 is labeled "teamRoster". The line of code, teamRoster.push_back("Mike");, is highlighted and Mike is stored into memory location 96. The line of code, teamRoster.push_back("Scottie");, is highlighted and Scottie is stored into memory location 97. The line of code, teamRoster.push_back("Toni");, is highlighted and Toni is stored into memory location 98. Memory locations 96, 97, and 98 are all labeled teamRoster. The line of code, cout << "Current roster: " << endl;, is highlighted and the output console now contains one line of output:
Current roster:

Step 2: The range-based for loop declares a variable playerName that will be assigned with each element of the vector teamRoster. The line of code, for (string playerName : teamRoster) {, is highlighted and memory location 100 is labeled playerName. Memory location 96, which contains Mike, is also highlighted. The line of code, cout << playerName << endl;, is highlighted and Mike is stored into memory location 100, then output. The output console now contains two lines of output:
Current roster:
Mike

The for loop repeats this process for the remaining players. The line of code, for (string playerName : teamRoster) {, is highlighted along with memory location 97, which contains Scottie. Then, the line of code, cout << playerName << endl;, is highlighted and Scottie is stored into memory location 100, then output. The line of code, for (string playerName : teamRoster) {, is highlighted along with memory location 98, which contains Toni. Then, the line of code, cout << playerName << endl;, is highlighted and Toni is stored into memory location 100, then output. The output console now contains four lines of output:
Current roster:
Mike
Scottie
Toni

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 19.1.2: 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 19.1.1: 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 19.1.3: 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 *= 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 19.1.2: 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 19.1.4: 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 *= -1.0;
   }
}
```

- -4.0, 5.5, 7.75
- Does not compile

3)
```
for (double sensorVal :
sensorReadings) {
   sensorVal -= 2.0;
}
```

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

CHALLENGE ACTIVITY 19.1.1: Range-based for loop.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   vector<string> shelterPets;
   shelterPets.push_back("Moby");
   shelterPets.push_back("Spot");

   cout << "Pets:" << endl;

   for (string petName : shelterPets) {
      cout << petName << endl;
   }
}
```

```
Pets:
Moby
Spot
```

1 2 3

Check Next

Range-based for loop with auto

Using auto further simplifies the amount of code needed to iterate through containers. auto causes the compiler to automatically determine the variable type needed in the range-based for loop. In the

example below, examGrades is a vector of type double. Using auto to declare variable gradeVal in the for loop causes the compiler to determine that gradeVal is a double.

Figure 19.1.3: Using a range-based for loop with auto to output a vector's elements.


```
vector<double> examGrades = {45.7, 72.5,
53.2};

for (auto gradeVal : examGrades) {
   cout << gradeVal << endl;
}
```

auto is used with & to declare a reference variable if the for loop modifies the elements in the container. In the example below, a reference variable is needed to modify the elements of vector examGrades. Thus, auto& is used to declare gradeVal.

Figure 19.1.4: Using a range-based for loop with auto and & to modify a vector's elements.

```
vector<double> examGrades = {45.7, 72.5,
53.2};

for (auto& gradeVal : examGrades) {
   gradeVal += 0.5;
}
```

zyDE 19.1.1: Explore auto.

Change the variable type of vector v from int to: double, char, or string.
Provide three input values of the new type to discover how auto works.

Load default template...

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

int main() {
  vector<int> v(3);  //

  // Modifying the eleme
  for (auto& a : v) {
    cin >> a;
  }

  // Outputting elements
  for (auto a : v) {
    cout << a << endl;
```

3 8 2

Run

The following example shows three common uses of the range-based for loop with auto. The first for loop sums up the elements in vector v to find the average. Since the elements are not modified, & is not needed. The second for loop modifies each element of v and thus requires auto&. The third for loop outputs the elements and does not need &.

Figure 19.1.5: Using a range-based for loop with auto to compute average and to modify and output a vector's elements.

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

int main() {
   vector<double> examGrades = {45.7, 72.3, 53.6}; // User's exam grades
   double averageGrade;

   // Find the average of three values
   averageGrade = 0.0;
   for (auto gradeVal : examGrades) {
      averageGrade += gradeVal;
   }
   averageGrade = averageGrade / examGrades.size();
   cout << "Average grade: " << averageGrade << endl;

   // Increase each value by 0.5
   for (auto& gradeVal : examGrades) {
      gradeVal += 0.5;
   }

   cout << "Adjusted grades:" << endl;
   // Output adjusted values
   for (auto gradeVal : examGrades) {
      cout << gradeVal << endl;
   }

   return 0;
}
```

```
Average grade: 57.2
Adjusted grades:
46.2
72.8
54.1
```


PARTICIPATION ACTIVITY 19.1.5: Using auto with range-based for loops.

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

```
double sumEarnings = 0.0;
for (                    :
earningsData) {
   sumEarnings += earningsVal;
}
```

Check Show answer

2) Using the auto type specifier, complete the code to add 0.2 to each double within the vector interestRates.

```
for (                    :
interestRates) {
   theRate += 0.2;
}
```

Check Show answer

3) What type will the compiler determine for the variable pointsVal?

```
vector<int> gamePoints;

...

double totalPointsPerGame =
0;
for (auto pointsVal :
gamePoints) {
   totalPointsPerGame +=
pointsVal;
}
```

Check Show answer

19.2 List

List container

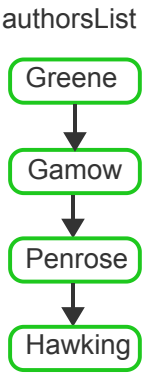
The ***list*** class defined within the C++ Standard Template Library (STL) defines a container of ordered elements, i.e., a sequence. The list class supports functions for inserting, modifying, and removing elements.

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("Butler");` adds "Butler" as the first list element.

PARTICIPATION ACTIVITY | 19.2.1: push_back() adds elements to end of list and push_front() add element to front of the list.


```
list<string> authorsList;

authorsList.push_back("Gamow");
authorsList.push_back("Penrose");
authorsList.push_back("Hawking");

authorsList.push_front("Greene");
```



Animation content:

Static figure:
Begin code.
list<string> authorsList;

authorsList.push_back("Gamow");
authorsList.push_back("Penrose");
authorsList.push_back("Hawking");

authorsList.push_front("Greene");
End code.

The code in the static figure creates a list object called "authorsList" which holds strings.

Step 1: The push_back() function adds an element, such as a String, to the end of the list. The push_back() function is used three times to append the following three strings to the end of authorList: "Gamow", "Penrose", "Hawking". The order of authorList is the same order the three strings were added: ["Gamow", "Penrose", "Hawking"].

Step 2: The push_front() function adds an element to the front of the list. The push_front() function is used to append "Greene" to the front of authorList. "Greene" is inserted before "Gamow", shifting each element to the right one index. Thus, the final order of authorList is: ["Greene", "Gamow", "Penrose", "Hawking"].

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 19.2.2: 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 | 19.2.3: pop_front() removes the first element and remove() removes specific elements from the list.

```
cout << authorsList.front() << endl;

authorsList.pop_front();

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

authorsList.remove("Greene");

cout << authorsList.front() << endl;
```

authorList

Penrose

Gamow
Greene
Penrose

Animation content:

Static figure:
Begin code.

```
cout << authorsList.front() << endl;

authorsList.pop_front();

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

authorsList.remove("Greene");

cout << authorsList.front() << endl;
```
End code.

The initial elements of authorList are ["Gamow", "Greene", "Penrose", "Greene"]. The output display box is empty.

Step 1: The front( ) function returns the first element in the list. The pop_front( ) function removes the first element from the list. The line of code, "cout << authorsList.front() << endl;" is highlighted and "Gamow", which is currently at the front of authorsList, is output. The display box now contains one line of output:
Gamow

The line of code, "authorsList.pop_front(); " is highlighted and "Gamow", which is currently at the front of authorsList, is removed from authorsList. The elements of authorList are now ["Greene", "Penrose", "Greene"].

The line of code, "cout << authorsList.front() << endl;" is highlighted and "Greene", which is currently at the front of authorsList, is output. The display box now contains two lines of output:
Gamow
Greene

Step 2: The remove( ) function removes all occurrences of a specific value from a list. The line of code, "authorsList.remove("Greene");" is highlighted and all instances of "Greene" in authorList are removed. The elements of authorList are now ["Penrose"].
The line of code, "cout << authorsList.front() << endl;", is highlighted and "Penrose", which is currently at the front of the list, is output. The final output of the display box contains three lines:
Gamow
Greene
Penrose

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 19.2.1: 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 19.2.4: 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

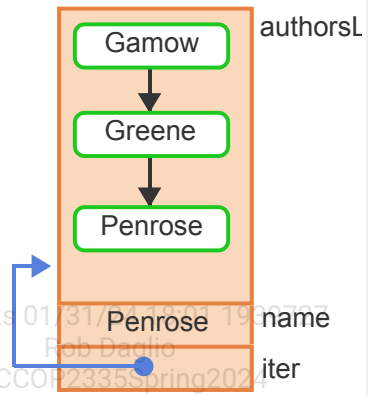
Iteration through a list necessitates keeping track of the current position in the list without using an index. An ***iterator*** is an object that points to a location in a list and can be used to traverse the list bidirectionally. The iterator allows for elements to be added, moved, and removed from 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 19.2.5: The list class provides functions to access elements of a list using an iterator.

```
list<string> authorsList;
string name;
list<string>::iterator iter;

authorsList.push_back("Gamow");
authorsList.push_back("Greene");
authorsList.push_back("Penrose");

for (iter = authorsList.begin(); iter != authorsList.end(); ++iter) {
   name = *iter;
   cout << name << endl;
}
```



Gamow
Greene
Penrose

Animation content:

Static figure:
Static figure:
Begin code.

```
list<string> authorsList;
string name;
list<string>::iterator iter;

authorsList.push_back("Gamow");
authorsList.push_back("Greene");
authorsList.push_back("Penrose");

for (iter = authorsList.begin(); iter != authorsList.end(); ++iter) {
   name = *iter;
   cout << name << endl;
}
```

End code.

A memory box contains 3 empty, segmented boxes: top, middle, and bottom. The output display box is empty.

Step 1: The iterator, iter, is an object for iterating through list elements. The line of code, "list<string> authorsList;", is highlighted and the top memory box is labeled "authorsList". The line of code, "string name;", is highlighted and the middle memory box is labeled "name". The line of code, "list<string>::iterator iter;", is highlighted and the bottom memory box is labeled "iter".

The next three lines of code, each containing push_back(), are highlighted and the elements of authorsList are now: ["Gamow", "Greene", "Penrose"]. This list is placed in the top memory box, labeled "authorsList".

Step 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. The code, "iter = authorsList.begin()" is highlighted. An arrow stems from the bottom memory box, "iter", and points to "Gamow", the first element of authorsList, in the top memory box.

Step 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. The code, "iter != authorsList.end();", is highlighted and returns true since iter is pointing to "Gamow", which is not the end of the list.

Step 4: Dereferencing iter reveals the string at the current position. Incrementing iter advances the

iterator to the next position.
The line of code, "name = *iter;", is highlighted and "Gamow" is stored into the middle memory box, labeled "name". Then, the line of code, "cout << name << endl;", is highlighted and the display box contains one line of output:
Gamow

The code, "++iter", is highlighted and the for loop continues. The iter now points to "Greene", the next element of the list. Since "Greene" is not the end of the list, the loop condition returns true, "Greene" is stored into the memory box labeled "name", and output. The for loop continues to the next element, "Penrose". Since "Penrose" is not at the end of the list, the loop condition returns true, "Penrose" is stored into the memory box labeled "name", and output.
The display box contains three lines of output:
Gamow
Greene
Penrose

Then, the iter points to the end of the list, thus the loop condition returns false and the loop terminates.

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 19.2.2: 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 19.2.6: 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 19.2.3: 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 19.2.7: 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 19.2.1: 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 19.2.8: 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 19.2.1: List.

539740.3879454.qx3zqy7

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

   size = letters.size();

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

   return 0;
}
```


gabd

1 2

Check Next

19.3 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 19.3.1: 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 19.3.1: 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

19.4 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.

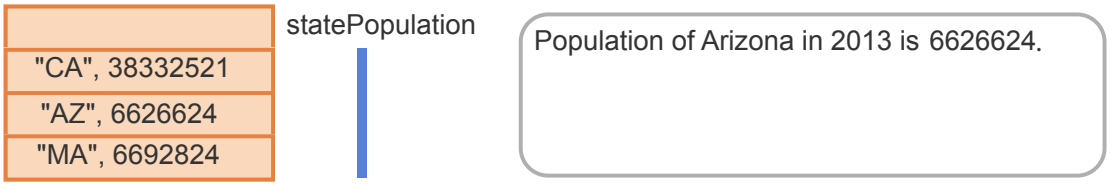
The at() function returns the value associated with a key, such as `statePopulation.at("CA")`.

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

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

PARTICIPATION ACTIVITY 19.4.1: A map allows a programmer to map keys to values.


```
map<string, int> statePopulation;

// 2013 population data from census.gov
statePopulation.emplace("CA", 38332521);
statePopulation.emplace("AZ", 6626624);
statePopulation.emplace("MA", 6692824);

cout << "Population of Arizona in 2013 is "
     << statePopulation.at("AZ")
     << "." << endl;
```



Animation content:

Static figure:
Begin C++ code:
map<string, int> statePopulation;

// 2013 population data from census.gov
statePopulation.emplace("CA", 38332521);
statePopulation.emplace("AZ", 6626624);
statePopulation.emplace("MA", 6692824);

cout << "Population of Arizona in 2013 is "
<< statePopulation.at("AZ")
<< "." << endl;
End C++ code.
map statePopulation has three entries, "CA", 38332521, "AZ", 6626624, and "MA", 6692824. The output console contains:
Population of Arizona in 2013 is 6626624.

Step 1: The emplace() function associates a key with the specified value. Statement map<string, int> statePopulation; declares and creates map statePopulation with string as the map's key type and int

as the map's value type. Statement statePopulation.emplace("CA", 38332521); associates "CA" with 38332521 and adds the entry to the map. Statement statePopulation.emplace("AZ", 6626624); associates "AZ" with 6626624 and adds the entry to the map. Statement statePopulation.emplace("MA", 6692824); associates "MA" with 6692824 and adds the entry to the map.

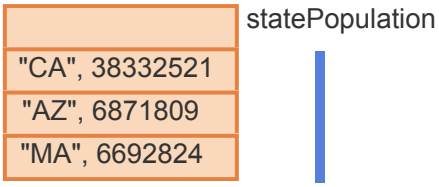
Step 2: The at() function returns the value associated with a key. Statement cout << "Population of Arizona in 2013 is " outputs Population of Arizona in 2013 is to the output console. Statement << statePopulation.at("AZ") uses the at() function to return the value associated with key "AZ", so 6626624 is output to the output console. The output console now contains:
Population of Arizona in 2013 is 6626624.

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 19.4.2: Updating the value associated with a key.

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

using namespace std;

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

   // 2013 population data from census.gov
   statePopulation.emplace("CA", 38332521);
   statePopulation.emplace("AZ", 6626624);
   statePopulation.emplace("MA", 6692824);

   cout << "Population of Arizona in 2013 is "
        << statePopulation.at("AZ")
        << "." << endl;

   // 2014 estimated population
   statePopulation.at("AZ") = 6871809;

   cout << "Population of Arizona in 2014 is "
        << statePopulation.at("AZ")
        << "." << endl;

   return 0;
}
```



Population of Arizona in 2013 is 6626624.
Population of Arizona in 2014 is 6871809.

Animation content:

Static figure:
Begin C++ code:

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

using namespace std;

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

   // 2013 population data from census.gov
   statePopulation.emplace("CA", 38332521);
   statePopulation.emplace("AZ", 6626624);
   statePopulation.emplace("MA", 6692824);

   cout << "Population of Arizona in 2013 is " << statePopulation.at("AZ") << "." << endl;

   // 2014 estimated population
   statePopulation.at("AZ") = 6871809;

   cout << "Population of Arizona in 2014 is " << statePopulation.at("AZ") << "." << endl;

   return 0;
}
```

End C++ code.
map statePopulation has three entries, "CA", 38332521, "AZ", 6871809, and "MA", 6692824. The output console contains:
Population of Arizona in 2013 is 6626624.
Population of Arizona in 2014 is 6871809.

Step 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. Statement map<string, int> statePopulation; declares and creates map statePopulation with string as the map's key type and int as the map's value type. Statement statePopulation.emplace("CA", 38332521); associates "CA" with 38332521 and adds the entry to the map. Statement statePopulation.emplace("AZ", 6626624); associates "AZ" with 6626624 and adds the entry to the map. Statement statePopulation.emplace("MA", 6692824); associates "MA" with 6692824 and adds the entry to the map. Statement cout << "Population of Arizona in 2013 is " << statePopulation.at("AZ") << "." << endl; outputs Population of Arizona in 2013 is 6626624., followed by a newline to the output console. Statement statePopulation.at("AZ") = 6871809; updates the entry "AZ", 6626624 by assigning the

entry with a new value, 6871809. The updated entry is now "AZ", 6871809. Statement cout << "Population of Arizona in 2014 is " << statePopulation.at("AZ") << "." << endl; outputs Population of Arizona in 2014 is 6871809., followed by a newline to the output console. The output console now contains:
Population of Arizona in 2013 is 6626624.
Population of Arizona in 2014 is 6871809.

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 19.4.3: 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 19.4.1: 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 time. If a race distance does not exist, the program throws an exception. Modify the program to use the count() function to check if the runner has run a race of the specified distance, and display a message of "No race of the specified distance exists."

RunDistTimeMap.... Load default template...

```
#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): "
```


5 15 10 0

Run

PARTICIPATION ACTIVITY 19.4.4: 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 19.4.1: 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 19.4.5: 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 19.4.1: Map functions.

539740.3879454.qx3zqy7

Start

Type the program's output


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

using namespace std;

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

   airportCode.emplace("GRX", "Granada, Spain");
   airportCode.emplace("MLN", "Melilla, Spain");
   airportCode.emplace("IWJ", "Iwami, Japan");

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

   airportCode.emplace("MLN", "Mendi, Ethiopia");

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

```
GRX: Granada,
MLN: Melilla,
```

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.

19.5 Set

Set container

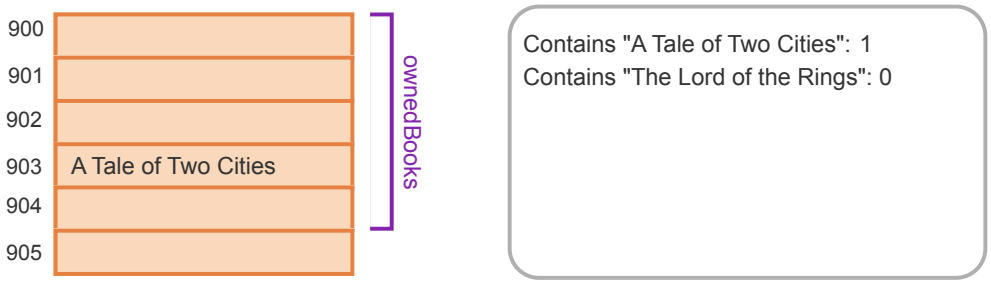
The ***set*** class defined within the C++ Standard Template Library (STL) defines a collection of unique elements. The set class supports functions for adding and removing elements, as well as querying if a set contains an element. For example, a programmer may use a set to store employee names and 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 19.5.1: A set's insert(), erase(), and count() functions add an item, remove an item, and check if an item exists within the set.


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

int main() {
   set<string> ownedBooks;
```

```
   ownedBooks.insert("A Tale of Two Cities");
   ownedBooks.insert("The Lord of the Rings");

   cout << "Contains \"A Tale of Two Cities\": "
        << ownedBooks.count("A Tale of Two Cities") << endl;

   ownedBooks.erase("The Lord of the Rings");

   cout << "Contains \"The Lord of the Rings\": "
        << ownedBooks.count("The Lord of the Rings") << endl;

   return 0;
}
```

Animation content:

Static figure:
Begin C++ code:

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

int main() {
   set<string> ownedBooks;

   ownedBooks.insert("A Tale of Two Cities");
   ownedBooks.insert("The Lord of the Rings");

   cout << "Contains \"A Tale of Two Cities\": "
        << ownedBooks.count("A Tale of Two Cities") << endl;

   ownedBooks.erase("The Lord of the Rings");

   cout << "Contains \"The Lord of the Rings\": "
        << ownedBooks.count("The Lord of the Rings") << endl;

   return 0;
}
```

End C++ code.

set ownedBooks is stored from memory location 900 to memory location 904. String "A Tale of Two Cities" is stored at memory location 903. The output console contains:
Contains "A Tale of Two Cities": 1
Contains "The Lord of the Rings": 0

Step 1: The insert() function adds an item such as a string to a set. Statement set<string> ownedBooks; declares set ownedBooks and the set's type as string. Statement ownedBooks.insert("A Tale of Two Cities"); uses the insert() function to add the string "A Tale of Two Cities" to the set at memory location 903. Statement ownedBooks.insert("The Lord of the Rings"); uses the insert() function to add the string "The Lord of the Rings" to the set at memory location 901.

Step 2: The count() function returns 1 if an item is in the set, and otherwise returns 0. Statement cout << "Contains \"A Tale of Two Cities\": " outputs Contains "A Tale of Two Cities": to the output console. Statement << ownedBooks.count("A Tale of Two Cities") << endl; uses the count() function to return 1 and is output to the output console, followed by a newline. The output console now contains:
Contains "A Tale of Two Cities": 1

Step 3: The erase() method removes an item from a set. Statement ownedBooks.erase("The Lord of the Rings"); uses the erase() method to remove the string "The Lord of the Rings" from the set. Statement cout << "Contains \"The Lord of the Rings\": " outputs Contains "The Lord of the Rings": to the output console. The output console now contains:
Contains "A Tale of Two Cities": 1
Contains "The Lord of the Rings": . Statement << ownedBooks.count("The Lord of the Rings") << endl; uses the count() to return 0 and is output to the output console, followed by a newline. The output console now contains:
Contains "A Tale of Two Cities": 1
Contains "The Lord of the Rings": 0

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 19.5.2: 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

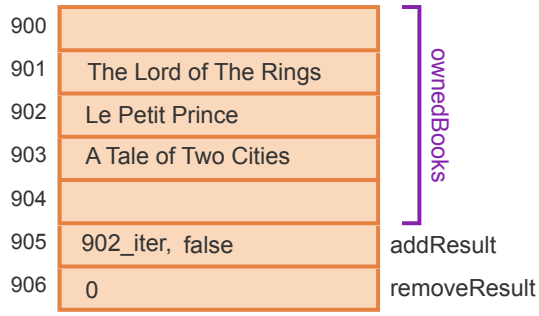
The insert() function does not add duplicate elements to a set. If a programmer tries to add a duplicate element, the insert() function returns a pair object containing an iterator at the existing element's location and false. Otherwise, insert() returns a pair object with an iterator at the added element 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 19.5.3: A set's insert() and erase() functions return a value to indicate the operation's failure or success.

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

int main() {
   set<string> ownedBooks;
   pair<set<string>::iterator, bool> addResult;
   int removeResult;

   ownedBooks.insert("A Tale of Two Cities");
   ownedBooks.insert("The Lord of the Rings");
   ownedBooks.insert("Le Petit Prince");
```

```
    addResult = ownedBooks.insert("Le Petit Prince");
    cout << "Added \"Le Petit Prince\" again: ";
    std::cout << std::boolalpha << addResult.second << endl;

    removeResult = ownedBooks.erase("The Hobbit");
    cout << "Removed \"The Hobbit\": " << removeResult << endl;

    return 0;
}
```



Added "Le Petit Prince" again: false
Removed "The Hobbit": 0

Animation content:

Static figure:
Begin C++ code:

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

int main() {
   set<string> ownedBooks;
   pair<set<string>::iterator, bool> addResult;
   int removeResult;

   ownedBooks.insert("A Tale of Two Cities");
   ownedBooks.insert("The Lord of the Rings");
   ownedBooks.insert("Le Petit Prince");

   addResult = ownedBooks.insert("Le Petit Prince");
   cout << "Added \"Le Petit Prince\" again: ";
   std::cout << std::boolalpha << addResult.second << endl;

   removeResult = ownedBooks.erase("The Hobbit");
   cout << "Removed \"The Hobbit\": " << removeResult << endl;

   return 0;
}
```

End C++ code.

set ownedBooks is stored from memory location 900 to memory location 904. String "The Lord of the Rings" is stored at memory location 901. String "Le Petit Prince" is stored at memory location 902. String "A Tale of Two Cities" is stored at memory location 903. The output console contains:
Added "Le Petit Prince" again: false
Removed "The Hobbit": 0

Step 1: The insert() function adds an item only if the item does not already exist. Statement ownedBooks.insert("A Tale of Two Cities"); uses the insert() function to add string "A Tale of Two Cities" to the set at memory location 903. Statement ownedBooks.insert("The Lord of the Rings"); uses the insert() function to add string "The Lord of the Rings" to the set at memory location 901. Statement ownedBooks.insert("Le Petit Prince"); uses the insert() function to add string "Le Petit Prince" to the set at memory location 902.

Step 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. Statement addResult = ownedBooks.insert("Le Petit Prince"); uses insert() to assign addResult with a pair consisting of an iterator at memory location 902 and boolean value false. Statements cout << "Added \"Le Petit Prince\" again: "; and std::cout << std::boolalpha << addResult.second << endl; output Added "Le Petit Prince" again: false, followed by a newline to the output console.

Step 3: The erase() function returns 1 if item removal was successful, otherwise returns 0. Statement removeResult = ownedBooks.erase("The Hobbit"); uses the erase() function to remove the string "The Hobbit" from the set and assigns removeResult with 0. Statement cout << "Removed \"The Hobbit\": " << removeResult << endl; outputs Removed "The Hobbit": 0 to the output console. The output console now contains:
Added "Le Petit Prince" again: false
Removed "The Hobbit": 0

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 19.5.4: 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 19.5.1: 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 program uses both count() and erase() to remove correct guesses from the set, and the game ends when user guesses all numbers (i.e., the set is empty). Modify the program to only use erase(), and not count(). Hint: consider the return value of erase().

GuessTheNumber... Load default template...

```
#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
```


1 2 5 3

Run

Common set operations

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

Table 19.5.1: 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 19.5.1: Using a set to define unique elements.

539740.3879454.qx3zqy7

Start

Type the program's output

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

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

   cin >> userInput;

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

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

   return 0;
}
```

Input

```
Beloved
Ulysses
Middlemarch
Ulysses
Ivanhoe
Beowulf
Dracula
done
```


Output

```
i
i
i
n
i
i
i
Size: 6
```

1 2

Check Next

19.6 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 19.6.1: 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.


```
queue<string> waitList;

waitList.push("Neumann party of 1");
waitList.push("Amdahl party of 2");
waitList.push("Flynn party of 4");

while (waitList.size() > 0) {
   cout << "Serving: " << waitList.front() << endl;
   waitList.pop();
}
```

waitList

Serving: Neumann party of 1
Serving: Amdahl party of 2
Serving: Flynn party of 4

waitList size = 0

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 19.6.2: 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 19.6.1: 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 at the tail of the queue.
		`// Assume exQueue is: "down" "right" "A"`
		`exQueue.back(); // Returns "A"`
		`// exQueue is still: "down" "right" "A"`

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 19.6.1: Queue.

Start

Type the program's output

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

using namespace std;

int main(){
   queue<string> fruits;

   fruits.push("banana");
   fruits.push("grape");
   fruits.push("pear");
   fruits.pop();

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

   return 0;
}
```

grape

1 2

Check Next

Exploring further:

- The queue class from Cplusplus.com

19.7 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 19.7.1: 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.

```
deque<string> tripRoute;

cout << "Depart Tokyo" << endl;
tripRoute.push_front("Tokyo");

cout << "Transfer at Osaka" << endl;
tripRoute.push_front("Osaka");

cout << "Arrive in Nara" << endl;
tripRoute.push_front("Nara");

cout << "\nReturn trip: " << endl;
cout << "Depart " + tripRoute.front() << endl;
tripRoute.pop_front();

cout << "Transfer at " + tripRoute.front() << endl;
tripRoute.pop_front();

cout << "Arrive in " + tripRoute.front() << endl;
tripRoute.pop_front();
```

tripRoute


```
Depart Tokyo
Transfer at Osaka
Arrive in Nara

Return trip:
Depart Nara
Transfer at Osaka
Arrive in Tokyo
```

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 19.7.2: 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.


```
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 19.7.1: Common deque functions.

push_front()	`push_front(newElement)`
	
	Adds newElement element at the head of the deque. The deque's size increases by one.
		`// Assume exDeque is: 3 5 6`
		`exDeque.push_front(1);`
		`// exDeque is now: 1 3 5 6`

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 19.7.1: Enter the output for deque.

539740.3879454.qx3zqy7

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("steak");
   groceries.push_front("apples");

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

   return 0;
}
```

```
1. apples
2. steak
```

1 2

Check Next

Exploring further:

- The deque class from Cplusplus.com

19.8 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)`:

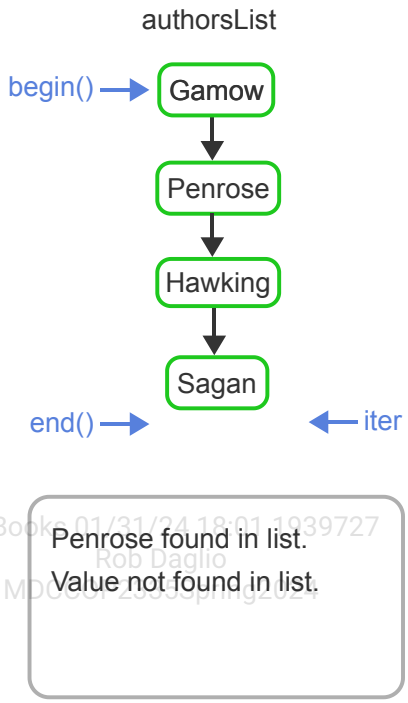
- iteratorFirst: An iterator to the range's first element
- iteratorLast: An iterator to one element past the range's last element
- value: The value of the element to be found within the range

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 19.8.1: The find() function.

```
list<string> authorsList;
authorsList.push_back("Gamow");
authorsList.push_back("Penrose");
authorsList.push_back("Hawking");
authorsList.push_back("Sagan");

list<string>::iterator iter;

iter = find(authorsList.begin(), authorsList.end(), "Penrose");

if (iter != authorsList.end()) {
   cout << *iter << " found in list." << endl;
}
else {
   cout << "Value not found in list." << endl;
}

iter = find(authorsList.begin(), authorsList.end(), "Austin");

if (iter != authorsList.end()) {
   cout << "Value " << *iter << " found in list." << endl;
}
else {
   cout << "Value not found in list." << endl;
}
```



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 19.8.2: 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 19.8.1: Using the find_if() function to find the first string formatted like ##.#.

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

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 19.8.3: 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()

19.9 sort() function

The sort() function

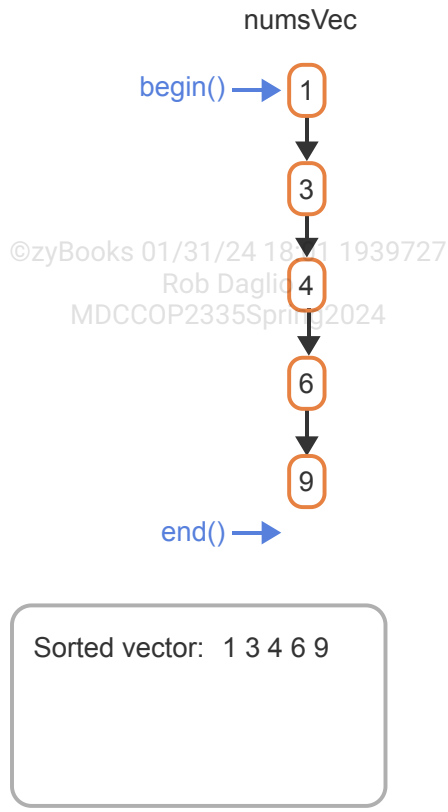
The ***sort()*** function arranges a container's elements in ascending order. sort() is defined in the Standard Template Library's (STL) algorithms library. Adding `#include <algorithm>` enables the use of sort(). sort() can be used with numerous STL containers, such as vectors and arrays. The container 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 19.9.1: Using sort().

```
int main() {
   unsigned int i;
   vector<int> numsVec;
   numsVec.push_back(3);
   numsVec.push_back(9);
   numsVec.push_back(4);
   numsVec.push_back(1);
   numsVec.push_back(6);

   sort(numsVec.begin(), numsVec.end());

   cout << "Sorted vector: ";
   for (i = 0; i < numsVec.size(); ++i) {
      cout << numsVec.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```



Animation content:

Static Figure:
Being C++ code:

```
int main() {
  unsigned int i;
  vector <int> numsVec;
  numsVec.push_back(3);
  numsVec.push_back(9);
  numsVec.push_back(4);
  numsVec.push_back(1);
  numsVec.push_back(6);

  sort(numsVec.begin(), numsVec.end());

  cout << "Sorted vector: ";
  for (i = 0; i < numsVec.size(); ++i) {
    cout << numsVec.at(i) << " ";
  }
  cout << endl;

  return 0;
}
```

End C++ code.
A integer vector numsvec with values 1, 3, 4, 6, 9. numsvec.begin() returns an iterator to the first element, 1. numsvec.end() returns an iterator to one element past the last element, 9. The output "Sorted vector: 1 3 4 6 9" appears in the console.
Step 1: sort() takes 2 arguments, iterators to the beginning and end of a range. The lines of code
numsVec.push_back(3);
  numsVec.push_back(9);
  numsVec.push_back(4);
  numsVec.push_back(1);
  numsVec.push_back(6);
are highlighted. The integer vector numsVec now contains values 3, 9, 4, 1, 6.
Step 2: sort( ) arranges elements in ascending order. The line of code sort(numsVec.begin(), numsVec.end()); is highlighted. The integer vector numsVec now contains values 1, 3, 4, 6, 9. The lines of code cout << "Sorted vector: ";
  for (i = 0; i < numsVec.size(); ++i) {
    cout << numsVec.at(i) << " ";
  }
  cout << endl
are highlighted. The output "Sorted vector: 1 3 4 6 9" appears in the console.

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 19.9.2: 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

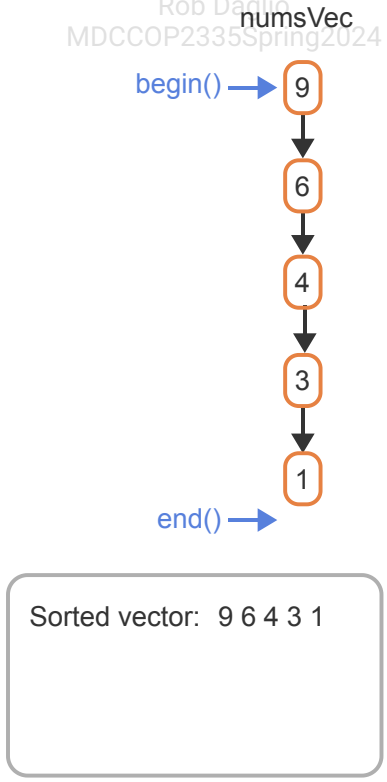
Passing custom comparison functions to sort()

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 | 19.9.3: Using sort() with a custom comparison function to sort in descending order.


```
bool sortDescending(int i, int j) {
   // Return true if i is greater than j,
   // indicating i should appear before j when sorted
   return i > j;
}

int main() {
   unsigned int i;
   vector<int> numsVec;
   numsVec.push_back(3);
   numsVec.push_back(9);
   numsVec.push_back(4);
   numsVec.push_back(1);
   numsVec.push_back(6);

   sort(numsVec.begin(), numsVec.end(), sortDescending);

   cout << "Sorted vector: ";
   for (i = 0; i < numsVec.size(); ++i) {
      cout << numsVec.at(i) << " ";
   }
   cout << endl;

   return 0;
}
```



Animation content:

Static figure:
Begin C++ code:

```
bool sortDescending(int i, int j) {
  // Return true if i is greater than j,
  // indicating i should appear before j when sorted
  return i > j;
}

int main() {
  unsigned int i;
  vector<int> numsVec;
  numsVec.push_back(3);
  numsVec.push_back(9);
  numsVec.push_back(4);
```

```
  numsVec.push_back(1);
  numsVec.push_back(6);

  sort(numsVec.begin(), numsVec.end(), sortDescending);

  cout << "Sorted vector: ";
  for (i = 0; i < numsVec.size(); ++i) {
    cout << numsVec.at(i) << " ";
  }
  cout << endl;

  return 0;
}
```


End C++ code. A integer vector numsVec with values 9, 6, 4, 3, 1. numsVec.begin() returns an iterator to the first element, 9. numsVec.end() returns an iterator to one element past the last element, 1. The output "Sorted vector: 9 6 4 3 1" appears in the console.

Step 1: A custom comparison function can be passed to sort( ) as the first argument. The lines of code int main() {

```
  unsigned int i;
  vector<int< numsVec;
  numsVec.push_back(3);
  numsVec.push_back(9);
  numsVec.push_back(4);
  numsVec.push_back(1);
  numsVec.push_back(6);
```

are highlighted. The integer vector numsVec now contains values 3, 9, 4, 1, 6.

Step 2: sortDescending is passed to sort(). sort() uses sortDescending to compare the vector's elements within the sort algorithm. The line of code sort(numsVec.begin(), numsVec.end(), sortDescending);

is highlighted.

Step 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. The integer vector numsVec now contains values 9, 6, 4, 3, 1. The lines of code cout << "Sorted vector: ";

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

are highlighted. The output "Sorted vector: 9 6 4 3 1" appears in the console.

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 19.9.4: 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 than 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 19.9.1: 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 compareByFirstName (Student student1, Student student2) {
   return student1.GetFirstName() < student2.GetFirstName();
}

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

bool compareByIdNum (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(), compareByFirstName);
   // 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(), compareByLastName);
   // 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(), compareByIdNum);
   // 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;
}
```


©zyBooks 01/31/24 18:01 1939727
Rob Daglio
MDCCOP2335Spring2024

```
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 19.9.5: 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(),
compareByLastName);
```

- 0
- 1

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

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

- 1
- 2

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

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

- 2
- 3

19.10 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
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 19.10.1: 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 () {
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
```


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) → Output

Program output displayed here

Coding trail of your work What is this?

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

19.11 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 | 19.11.1: LAB: Student grades (map) | 0 / 10

main.cpp — Load default template...

```
1  #include <map>
2  #include <string>
3  #include <iostream>
4
5  using namespace std;
6
7  int main () {
8     string studentName;
9     double studentGrade;
10
11    map<string, double> studentGrades;
12
13    // Students' grades (pre-entered)
14    studentGrades.emplace("Harry Rawlins", 84.3);
15    studentGrades.emplace("Stephanie Kong", 91.0);
```

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) → Output

Program output displayed here

Coding trail of your work What is this?

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


19.12 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 19.12.1: LAB: Ticketing service (queue) 0 / 10

main.cpp Load default template...

```
1 #include <queue>
2 #include <iostream>
3
4 using namespace std;
```

```
int main () {
   string personName = "";
   int counter = 0;
   int youPosition;

   queue<string> peopleInQueue;

   getline(cin, personName);
   while (personName != "-1") {
      // TODO: Add personName to peopleInQueue
```


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) → Output

Program output displayed here

Coding trail of your work What is this?

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

19.13 LAB: Palindrome (deque)

A palindrome is a string that reads the same backwards and forwards. Use a deque to implement a program that tests whether a line of text is a palindrome. The program reads a line, then outputs whether the input is a palindrome or not.

Ex: If the input is:

```
senile felines!
```

the output is:

```
Yes, "senile felines!" is a palindrome.
```

Ex: If the input is:


```
rotostor
```

the output is:

```
No, "rotostor" is not a palindrome.
```

Ignore punctuation and spacing. Assume all alphabetic characters will be lowercase.

Special case: A one-character string is a palindrome.

LAB ACTIVITY | 19.13.1: LAB: Palindrome (deque) | 0 / 10

main.cpp — Load default template...

```
#include <iostream>
#include <deque>

using namespace std;

int main() {
   string line;
   bool result;

   /* Type your code here. */

   return 0;
}
```


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) → Output

Program output displayed here

Coding trail of your work What is this?

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

19.14 LAB: Unique random integers (set)

Given integer inputs howMany and maxNum, generate a vector of howMany unique random integers from 0 to maxNum (exclusive).

The structure of the program is:

- main() calls UniqueRandomInts() with arguments howMany, and maxNum.
- UniqueRandomInts() returns a vector of howMany unique random integers.
- The required output is already provided in main() and PrintNums().

Complete UniqueRandomInts(), which generates random integers until howMany unique integers have been collected in vector nums.

Hint: If a generated number is new, add the number to vector nums and set alreadySeen. If the number has been seen before, increment the global variable retries and generate another random integer.

Note: For testing purposes, the random number generator is seeded with a fixed value (641) in main().

Ex: When the input is:

```
5 8
```

the output is

```
5 4 0 1 6   [2 retries]
```

LAB ACTIVITY | 19.14.1: LAB: Unique random integers (set) 0 / 10

main.cpp Load default template...


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

// Print the integers in vector nums separated by a space
void PrintNums(vector<int> nums, int size) {
   for (int i = 0; i < size; ++i) {
      cout << nums.at(i) << " ";
   }
}

// Used in uniqueRandomInt, printed in main()
int retries;
```

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) → Output

Program output displayed here

Coding trail of your work What is this?

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