Activity #28: Containers
Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

a) Model 1, Question #3

b) Model 2, Question #7

c) Model 3, Question #11

d) Model 4, Question #14

Activity #28: Containers

In this activity, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to C++ containers beyond the vector.

Content Learning Objectives

After completing this activity, students should be able to:

- Use a range-based for loop to iterate over a container
- Distinguish between `list`, `pair`, `map`, `set`, `queue`, and `deque`
- Select the appropriate container for a given programming task

Process Skill Goals

During the activity, students should make progress toward:

- Write C++ code using range-based for loops with value and reference variables
- Write C++ code that stores and retrieves data using `map` and `set`

Model 1 Range-Based For Loops

Traditional For Loop

```
vector<int> scores =
    {92, 78, 85, 91, 63};

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

```

Range-Based For Loop

```
vector<int> scores =
    {92, 78, 85, 91, 63};

for (int score : scores) {
    cout << score << " ";
}
cout << endl;

```

Output: `92 78 85 91 63`

Refer to Model 1 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

1. Compare the traditional for loop on the left with the range-based for loop on the right.

a) What three parts appear in the traditional for loop header (before the opening brace), separated by semicolons?

b) In the range-based for loop, what symbol separates the loop variable from the container being iterated?

c) How many times does the body of each loop execute?

d) What is the value of `score` during the third iteration of the range-based for loop?

2. The range-based for loop declares a variable that takes on the value of each element during each iteration.

a) In the range-based for loop above, what is the declared type and name of this variable?

b) If `scores` were a `vector<string>`, what type should the loop variable be declared as?

c) Write a range-based for loop that prints each element of a `vector<double>` named `temps` on its own line.

3. Consider the two loops below.

Loop A

```
1 for (int score : scores) {
2     score += 5;
3 }
4
```

Loop B

```
1 for (int& score : scores) {
2     score += 5;
3 }
4
```

a) What is the only syntactic difference between Loop A and Loop B?

b) In Loop A, `score` is a copy of each element. Does modifying `score` inside the loop change the original vector?

c) The `&` in Loop B declares `score` as a *reference*, meaning it refers directly to the element in the vector. Does modifying `score` in Loop B change the original vector?

d) The file `activity28a.cpp` contains both loop styles. Run it and record the output of the last two lines.

4. Write a range-based for loop that computes the sum of all elements in a `vector<int>` named `values`. Use a reference when appropriate.

Model 2 List and Pair

```cpp
#include <list>
#include <utility>

list<string> tasks =
    {"breakfast", "class", "lunch"};

tasks.push_front("wake up");
tasks.push_back("dinner");
tasks.pop_front();

for (string task : tasks) {
    cout << task << endl;
}
```

After push_front("wake up"):

wake up	breakfast	class	lunch

After push_back("dinner"):

wake up	breakfast	class	lunch	dinner

After pop_front():

breakfast	class	lunch	dinner

Output:

```
breakfast
class
lunch
dinner
```

Operation	vector	list
`push_back(val)`	yes	yes
`pop_back()`	yes	yes
`push_front(val)`	no	yes
`pop_front()`	no	yes
`at(i)`	yes	no
range-based for	yes	yes

```cpp
pair<string, int> student = {"Alice", 95};
cout << student.first << " scored ";
cout << student.second << endl;
```

Output:

```
Alice scored 95
```

Refer to Model 2 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

5. What `#include` header is required to use each of the following?

a) `list`

b) `pair`

6. Using the operations table in the model, answer the following.

a) Write a line of C++ to access the 3rd element (index 2) of a `vector<int>` named `v`.

b) Can you do the same operation with a `list`? Explain why or why not.

c) A `vector` does not support `push_front`. Why might adding an element to the front of a vector be slow compared to a list?

7. When would you choose a `list` over a `vector`?
When would you choose a `vector` over a `list`?

8. A `pair` stores two related values of potentially different types and is accessed via `.first` and `.second`.

a) In the pair declaration above, what is the type of `student.first`?

b) What is the type of `student.second`?

c) Write one line of C++ to declare a `pair` that stores a city name (string) and its population (int), with the city `"Bellingham"` and population `90000`.

d) The file `activity28b.cpp` contains the list and pair examples. Run it and verify the output matches the model. What does `tasks.front()` return?

Model 3 Map and Set

```
#include <map>

map<string, int> population;
population["Seattle"] = 750000;
population["Bellingham"] = 90000;
population["Olympia"] = 52000;
population["Seattle"] += 1000;

for (pair<string,int> city : population) {
    cout << city.first << ": ";
    cout << city.second << endl;
}
cout << population.count("Seattle");
cout << population.count("Portland");

```

population map:

key	value
"Bellingham"	90000
"Olympia"	52000
"Seattle"	751000

Output:

```
Bellingham:  90000
Olympia:  52000
Seattle:  751000
1
0
```

```
#include <set>

set<int> scores = {85, 92, 78, 92, 85, 100};
scores.insert(73);
scores.insert(92);

for (int score : scores) {
    cout << score << endl;
}
cout << "size: " << scores.size();

```

scores set (sorted, unique):

73	78	85	92	100

Output:

```
73
78
85
92
100
size:  5
```

Refer to Model 3 above as your team develops consensus answers to the questions below.

Questions (20 min) Start time:

9. A `map` stores key-value pairs, where each key maps to exactly one value. Answer the following about the `map` example above.

a) What is the type of the *key* in `population`?

b) What is the type of the *value* in `population`?

c) What syntax is used to insert or update a value in the map?

d) What happens when `population["Seattle"]` is assigned a second time? Does it create a new entry or modify the existing one?

10. What does `map.count(key)` return? What are the two possible return values and what does each mean?

11. When iterating over a `map` or `set` with a range-based for loop, the elements are visited in a particular order.

a) In what order does the `for` loop print the cities from the `map`?

b) In what order does the `for` loop print the values from the `set`?

c) Both `map` and `set` keep their elements *sorted*. How does this differ from `vector` and `list`?

12. The `set` in the model is initialized with six values, two of which are duplicates (92 appears twice, 85 appears twice).

a) How many elements does the set actually contain?

b) After `scores.insert(92)`, does the set size increase?

c) When would a `set` be more useful than a `vector` for storing a collection of values?

13. The file `activity28c.cpp` contains both examples. Run it and verify the output. Then write C++ code to count the number of times each word appears in the sentence `"the cat sat on the mat"`. Use a `map<string,int>`.

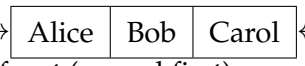
Model 4 Queue and Deque

```cpp
#include <queue>

queue<string> waitlist;
waitlist.push("Alice");
waitlist.push("Bob");
waitlist.push("Carol");

while (!waitlist.empty()) {
    cout << waitlist.front() << endl;
    waitlist.pop();
}
```

waitlist queue (FIFO):



front (served first) back (added here)

Output:

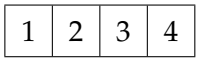
```
Alice
Bob
Carol
```

```cpp
#include <deque>

deque<int> dq;
dq.push_back(3);
dq.push_front(2);
dq.push_front(1);
dq.push_back(4);

for (int val : dq) {
    cout << val << " ";
}
cout << endl;
dq.pop_front();
dq.pop_back();
```

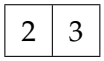
dq after all push operations:



Output:

```
1 2 3 4
```

dq after pop_front and pop_back:



Container	push_front	push_back	pop_front	pop_back	at(i)	range-for
`vector`	no	yes	no	yes	yes	yes
`list`	yes	yes	yes	yes	no	yes
`queue`	no	yes	yes	no	no	no
`deque`	yes	yes	yes	yes	yes	yes

Refer to Model 4 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

14. A `queue` models a waiting line: items are added at the back and removed from the front. This behavior is called *FIFO* (First In, First Out).

a) Who was the first person pushed onto the `waitlist`?

b) Who was printed (served) first?

c) If you called `waitlist.push("Dave")` before the `while` loop, what would be the last name printed?

d) Why can't you use a range-based for loop to print the contents of a `queue`? (Hint: look at the table.)

15. Using the operations table, compare `queue` and `deque`.

a) What does the "d" in `deque` stand for?

b) List two operations that `deque` supports but `queue` does not.

c) A `deque` supports `at(i)` but a `queue` does not. What does this tell you about which container supports random access?

16. For each scenario below, choose the most appropriate container from `vector`, `list`, `map`, `set`, `queue`, or `deque`. Justify each choice.

a) Storing a list of student exam scores that you need to access by index and sort.

b) Keeping track of customers waiting in line at a support desk (serve in the order they arrived).

c) Storing a dictionary mapping English words to their definitions.

d) Recording which usernames have already been taken (no duplicates, fast lookup).

e) Simulating a deck of cards where cards can be drawn or added from either end.

17. The file `activity28d.cpp` contains the queue and deque examples. Run it and verify the output matches the model. Then write C++ code using a `deque<int>` that adds the values 1, 2, 3 to the front (in order) and 4, 5, 6 to the back, then prints all values.