Module 12: Iterators

Learning Objectives

1. Trace code that uses iterators.
2. Write code that uses iterators as parameters to perform a behavior.

Process Skills

1. Information processing. Extract structural patterns from the sample code.
2. Critical thinking. Transfer understanding of iterators from one context to a new context.

Please fill in the roles for each member of your team. Take a look at the description of each role to see its responsibilities. If there are only three people in the group, please assign the same person to the **Presenter** and **Reflector** role. It is a good idea to select roles that you have not recently taken.

Team name: _____ Date: _____

Role	Team Member Name
Manager. Keeps track of time and makes sure everyone contributes appropriately.	
Presenter. Talks to the facilitator and other teams.	
Reflector. Considers how the team could work and learn more effectively.	
Recorder. Records all answers and questions and makes the necessary submission.	

For virtual activities: Once you select your roles, change your Zoom name using the format and example below.

Format: *Group X: First name, Last name initial / Role*

Example: *Group 1: Paul I / Presenter*

Model 1. Vector iterators (12 min) Start time: _____

Iterators allow you to access each element of any container (e.g., std::vector, std::map) using a general set of operations. Regardless of the underlying implementation, we use the same iterator operations. The table below lists the most commonly used operations. **Note: Iterators are implemented as objects that store pointers to specific elements of a container*

Iterator operation	**Description**
`container::begin()`	Return iterator pointing to the container's first element.
`container::end()`	Return iterator pointing to the element following the container's last element. This element acts as a placeholder and will return an error when dereferenced.
`operator++`	Update the iterator so it points to the next element.
`operator==`	Test if two iterators point to the same element.
`operator*`	Dereference the element of the container.

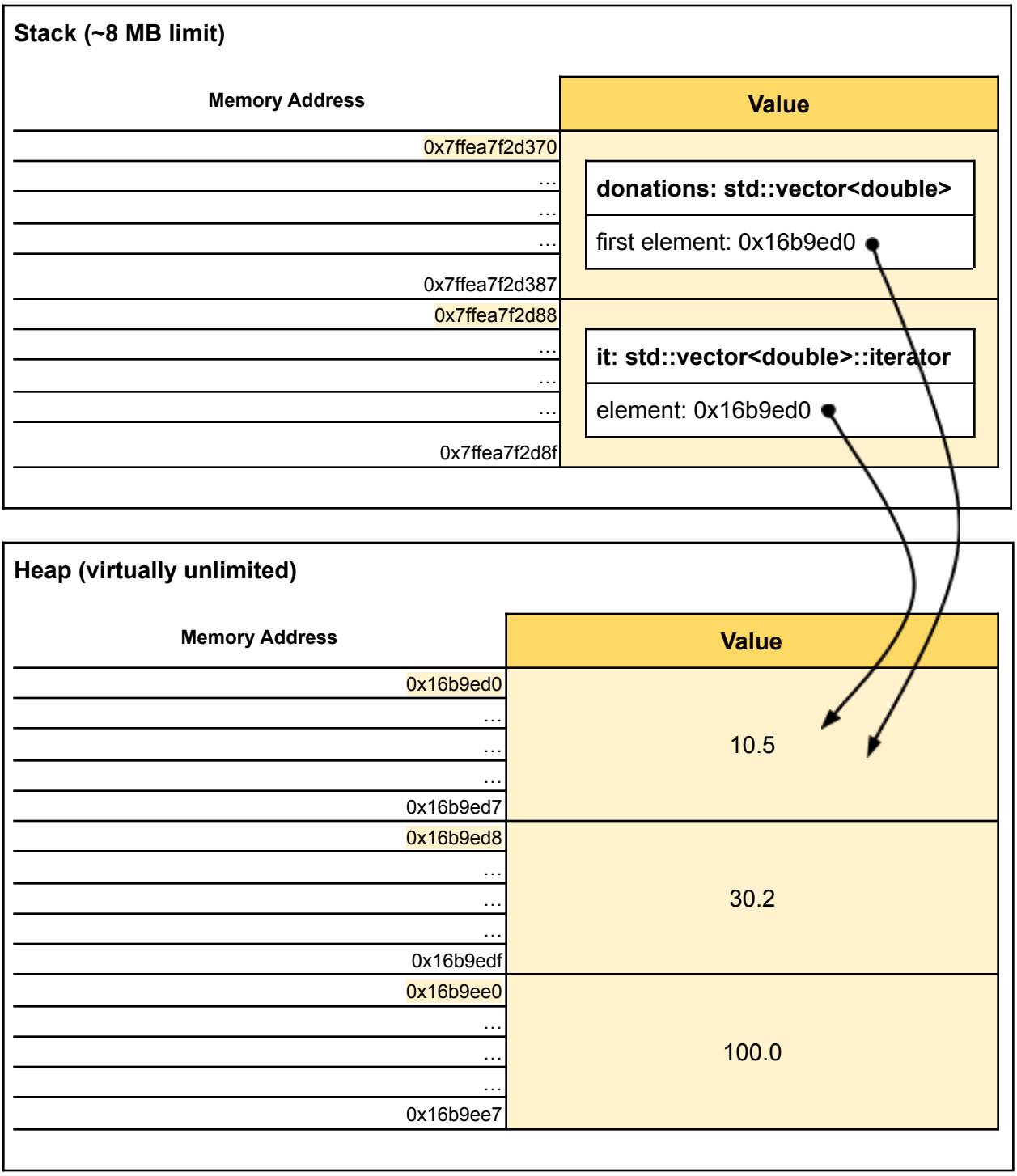
```
int main() {
 std::vector<double> donations {10.5, 30.2, 100.0};

 std::vector<double>::iterator it = donations.begin();
 std::cout << *it << "\n";

 it++;
 std::cout << *it << "\n";

 if (it == donations.end()) {
   std::cout << "All donations viewed.\n";
 } else {
   std::cout << "We have more donations.\n";
 }
 return 0;
}
```

Screen output:
10.5
30.2
We have more donations.



1. Which operation gives us back an iterator that points to the first element of a container (e.g., std::vector)?

container::begin

2. Which element of the donations std::vector did the iterator object `it` initially point to? Provide the value of the element.

10.5

3. What operation do we use to retrieve the value of the element pointed to by the iterator object `it`? Place a check (✓) beside your answer.

   a. operator* ✓
   b. operator++
   c. operator==

4. Why do we get the value 30.2 when we dereference and display the iterator `it` after performing the expression it++? Place a check (✓) beside your answer.

   a. Incrementing the value of the element pointed to by the iterator object `it` gives back 30.2
   b. operator* should have been used instead of operator++
   c. operator++ updates the iterator object `it` so it stores the address of the next element in the vector, 30.2. ✓

5. What address do you think the iterator object `it` stores after performing the expression it++? Place a check (✓) beside your answer.

   a. 0x16b9ed0
   b. 0x16b9ed8 ✓
   c. 0x16b9ee0
   d. 0x7ffea7f2d8f

6. If we add another it++ expression, what address do you think the iterator object `it` will store? Place a check (✓) beside your answer.

   a. 0x16b9ed0
   b. 0x16b9ed8
   c. 0x16b9ee0 ✓
   d. 0x7ffea7f2d8f

7. According to the code, when will the body of the if condition get executed? Place a check (✓) beside your answer.

   a. When the iterator object is updated to store the address of the element following the last element. ✓
   b. When the iterator object is updated to store the address of the first element.
   c. When the iterator object is first instantiated using the begin operation
   d. It will never get executed.

8. Predict the screen output of the code below.

```
int main() {
  std::vector<double> donations {10.5, 30.2, 100.0};
  for (std::vector<double>::iterator it =
       donations.begin(); it != donations.end(); it++) {
    std::cout << *it << " ";
  }
  return 0;
}
```

10.5 30.2 100.0

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 2. Map iterators (9 min) Start time: _____

std::map iterators work the same way as std::vector iterators, except their elements are stored in std::pairs to distinguish keys and values. *Note: std::map elements may not be stored in the same order they were inserted into the map.*

```
int main() {
  std::map<int, std::string> products {
    {111, "milk"},
    {112, "banana"},
    {113, "water"},
  };

  std::map<int, std::string>::iterator it = products.begin();
  std::cout << it->first << ": " << it->second << "\n";

  return 0;
}
```

Screen output
111: milk

9. What operation did we use to get an iterator for products that pointed to the first element? Place a check (✓) beside your answer.

   a. products.begin() ✓
   b. it++
   c. products.end()

10. Why do we use the expression it->first instead of it.first? Place a check (✓) beside your answer.

    a. We need to dereference the iterator object `it` to access and call the std::pair object's `first` member variable. ✓
    b. The code will be complete if we use it.first.
    c. We need to use the arrow operator to access the first element of the map.

11. What expression should we use if we want to access the next element of the map using the iterator object `it`? Place a check (✓) beside your answer.

    a. products.begin()
    b. it++ ✓
    c. products.end()

12. What expression should we use to get an iterator for products that point to the element following its last element? Place a check (✓) beside your answer.

    a. products.begin()
    b. it++
    c. products.end() ✓

13. Complete the code to display all names and quantities in the fridge_contents std::map.

```
int main() {
  std::map<std::string, int> fridge_contents {
    {"milk", 4},
    {"banana", 5},
    {"water", 10}
  };
  // TODO: Create an iterator for fridge_contents called `it`
  // pointing to the first element of the container.
 std::map<std::string, int>::iterator it = fridge_contents.begin();

  // TODO: Check if iterator object `it` is not yet pointing to the
  // element following the last element of the map
  while(it != fridge_contents.end()) {
    // TODO: Display the key of the current map element
    std::cout << it->first;
    std::cout << " x ";
    // TODO: Display the value of the current map element
    std::cout << it->second;
    std::cout << "\n";
    // TODO: Move it to the next map element
    it++;
  }
  return 0;
}
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Model 3. Iterators as parameters (12 min) Start time: _____

Several classes and functions in the std library use iterators as parameters. The table below lists some of the commonly used ones.

Function/member function	**Description**
iterator std::vector::erase(iterator pos);	Removes the element at pos
iterator std::vector::insert(const_iterator pos, const T& value);	Inserts value before pos
iterator std::find(iterator first, iterator last, const T& value);	Finds element that matches value starting from first to last. Returns an iterator pointing to the element that matches value, or returns iterator pointing to last if no matches were found.

```
#include <iostream>
#include <vector>
#include <algorithm> // required to use std::find

void DisplayVector(const std::vector<std::string> &vec) {
  for (std::string elem : vec) {
    std::cout << elem << " ";
  }
  std::cout << "\n";
}

int main() {
  std::vector<std::string> product_names {"milk", "banana",
"apple",
                                          "water", "bread"};
  DisplayVector(product_names);

  product_names.erase(product_names.begin());

  DisplayVector(product_names);

  product_names.erase(product_names.begin() + 2);
  DisplayVector(product_names);

  std::vector<std::string>::iterator it =
```

```
      std::find(product_names.begin(), product_names.end(),
"bread");
  product_names.erase(it);

  DisplayVector(product_names);

  return 0;
}
```

Screen output:
milk banana apple water bread
banana apple water bread
banana apple bread
banana apple

14. Which element did product_names.begin() point to before the first call to erase? Place a check (✓) beside your answer.

    a. milk ✓
    b. banana
    c. apple
    d. water
    e. bread

15. What element was removed after the first call to erase? Place a check (✓) beside your answer.

    a. milk ✓
    b. banana
    c. apple
    d. water
    e. bread

16. After erasing the first element, what element does product_names.begin() point to? Place a check (✓) beside your answer.

    a. milk
    b. banana ✓
    c. apple
    d. water
    e. bread

17. After erasing the first element, what element does product_names.begin() + 2 point to?
Note: Adding an integer n to an iterator will give back a new iterator pointing to the element that is n positions after the iterator. Place a check (✓) beside your answer.

    a. milk
    b. banana
    c. apple
    d. water ✓
    e. bread

18. How many iterators do we pass to the std::find function?

2

19. Analyze the call to std::find. What element does the returned iterator point to? Place a check (✓) beside your answer.

    a. milk
    b. banana
    c. apple
    d. water
    e. bread ✓

20. Analyze the std::vector::insert member function declaration. Use std::vector::insert to complete the code for inserting "Tuffy" between "Mildred" and "Willie" in the volunteer_names std::vector.

The function declaration and description for std::insert is pasted below for your convenience:

Function/member function	Description
iterator std::vector::insert(const_iterator pos, const T& value);	Inserts value before pos

```
void DisplayVector(const std::vector<std::string> &vec) {
  for (std::string elem : vec) {
    std::cout << elem << " ";
  }
  std::cout << "\n";
}

int main() {
  std::vector<std::string> volunteer_names {
    "Fram", "Mildred", "Willie", "Milton", "Jewel"
  };
  DisplayVector(volunteer_names);
  std::cout << "\nAfter insertion\n";

  // TODO: Create an iterator for volunteer_names called `it`
  // pointing to the first element of the container.
  std::vector<std::string>::iterator it = volunteer_names.begin();


  // TODO: Insert Tuffy between Mildred and Willie
  volunteer_names.insert(it + 2, "Tuffy");

  DisplayVector(volunteer_names);
  return 0;
}
```

🛑 **STOP HERE AND WAIT FOR FURTHER INSTRUCTIONS**

Reflector questions

1. What was the most useful thing your team learned during this session?

2. At this point, reflect and rate how confident you are about your skills in learning to use classes, member function, and functions given sample code (e.g., creating an iterator, using operator++ on an iterator, calling erase). Write a number between 1 - 5, 1 being not confident and 5 being very confident.

3. What previous activities helped you understand how to use the std::vector::insert member function?