Module 4: Vectors

Learning Objectives

1. Write code to call member functions given member function declarations.
2. Write code that creates and uses vectors.

Process Skills

1. Information processing. Extract structural patterns from sample code.
2. Critical thinking. Interpret member function declarations.

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. std::vector (6 min) Start time: ______

The class diagram below shows a simplified version of the std::vector class that highlights common methods. You can find the full vector class at https://en.cppreference.com/w/cpp/container/vector.

std::vector
// member variables not // shown here

vector();
vector(size_type count);
vector(const std::vector& other);
vector(std::initializer_list<T> init);
reference at(size_type pos);
void push_back(const T& value);
void clear();
size_type size() const;
bool empty() const;
 	
// other member functions not
// shown here

std::vector is a template class, where we provide the contained elements' data type during construction. We use T to refer to the template data type.

size_type is a data type used in std::vector that can hold positive non-fractional numbers

Member functions
vector(); - constructs an empty container.

vector(const std::vector& other); - constructs a container and copies the contents of other; also called the *copy constructor*

vector(size_type count); - constructs container with count instances using default values for the elements.

vector(std::initializer_list<T> init); - constructs a container with the contents of the initializer list init

reference at(size_type pos);- returns a reference to the element at specified location pos, with bounds checking. reference gives access to the element stored in the vector.

void push_back(const T& value); - appends the given element value at the end of the container. T refers to the vector's template data type so it expects value to have the same data type.

void clear(); - erases all elements from the container. After this call, size() returns zero.

size_type size(); - returns the number of elements in the container.

bool empty() const; - checks if the container has no elements.

1. How many constructors are shown in the std::vector class diagram?

4

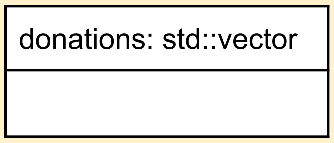
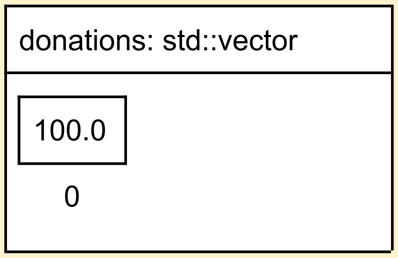
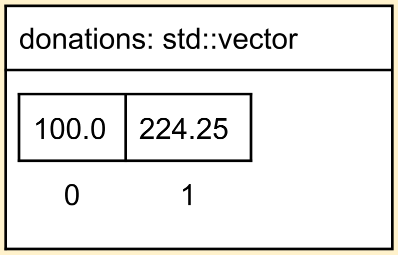
2. Which member function appends values at the end of the std::vector container? Write the name of the function below.

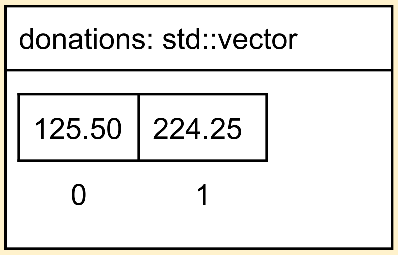
push_back

3. Which member function can tell us if the std::vector container does not contain any values? Write the name of the function below.

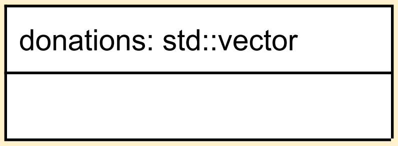
empty

Model 2. Using std::vector (15 min) Start time: _______

Line	Code	Visualization
01	`std::vector<double> donations;`	
02	`donations.push_back(100.0);`	
03	`donations.push_back(224.25);`	

04	`donations.at(0) = 125.50;`	
05	`std::cout << donations.at(1);`	Screen output: 224.25

06
 	
07
08
	`std::size_t donation_count =`
	`  donations.size();`
	`std::cout << donation_count`
	`          << "\n";`
		Screen output: 2

09	`donations.clear();`	

10
11
12
13
14
15
16
	`bool is_empty = donations.empty();`
	`if (is_empty) {`
	`  std::cout << "No donations.\n";`
	`} else {`
	`  std::cout << "Received "`
	`            << "donations.\n";`
	`}`
		Screen output: No donations

4. Which std::vector constructor did we use in line 01? Write the constructor's function declaration below.

The default constructor vector();

5. Complete the table below to indicate the number of elements in the std::vector container after performing code in the specified line numbers

Line	Number of elements in donations
01	0
02	1
03	2
04	2

6. Where in the std::vector container does the push_back member function add an element? Place a check (✓) beside your answer.

   a. Front of the container
   b. Middle of the container
   c. End of the container ✓

7. Line 04 passes 0 as the argument to the at member function. Which element in the container does it change? We use the term *position* or *index* to refer to an element's location in the container. *position* or *index* starts at 0. Place a check (✓) beside your answer.

   a. First element ✓
   b. Second element
   c. None of the elements

8. The at member function can access values inside a std::vector container. Complete the table below to specify the argument you would pass to the at member function to access the corresponding element in a std::vector container. Assume we have a std::vector that contains 100 elements.

std::vector container element	argument passed to the at member function
First element	0
30th element	29
Last element	99

9. What does the size member function return? Place a check (✓) beside your answer.

    a. The number of elements in the std::vector container. ✓
    b. The position of the last element in the std::vector container.
    c. The position of the first element in the std::vector container.

10. When using a std::vector, what values do we expect to get from the size and empty member functions after calling the clear member function? Complete the table below with the expected values.

member function	value after donations.clear()
donations.size()	0
donations.empty()	true

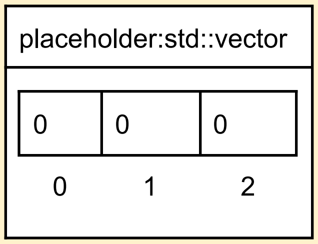
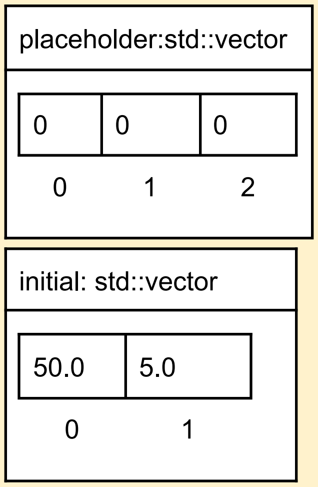
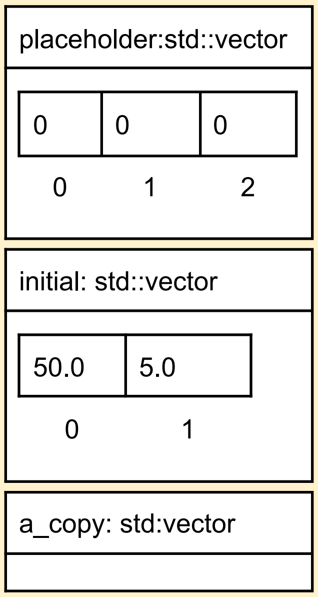
11. Write code to create a std::vector of customers using the default constructor. Each element represents the number of visitors per hour. Specifically, the first element contains the number of visitors in the first hour of operation, the second element for the second hour, and so on. Add the following hours to the vector in the same order: 2, 4, 1, 4, 3, 2, 1, 1. Get the number of times that an hour was logged to the std::vector and store it in a variable called hours_logged. Get the number of visitors in the fourth hour and store it in a variable called midday_visitors.

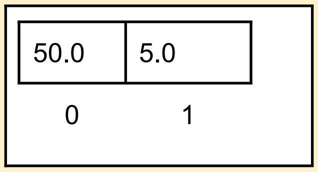
```
int main() {
  std::vector<int> customers_per_hour;
  customers_per_hour.push_back(2);
  customers_per_hour.push_back(4);
  customers_per_hour.push_back(1);
  customers_per_hour.push_back(4);
  customers_per_hour.push_back(3);
  customers_per_hour.push_back(2);
  customers_per_hour.push_back(1);
  customers_per_hour.push_back(1);
  std::size_t hours_logged = customers_per_hour.size();
  int midday_visitors = customers_per_hour.at(3);

  std::cout << "Logged hours: " << hours_logged << "\n";
  std::cout << "Visitors in the middle of the day: " << midday_visitors << "\n";
  return 0;
}
```

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

Model 3. std::vector initialization (6 min) Start time: ______

Line	Code	Visualization
01	`std::vector<int> placeholder(3);`	
02	`std::vector<double> initial {50.0, 5.0};`	 placeholder:std::vector 0 0 0 0 1 2 initial: std::vector 50.0 5.0 0 1
03	`std::vector<double> a_copy(initial);`	 placeholder:std::vector 0 0 0 0 1 2 initial: std::vector 50.0 5.0 0 1 a_copy: std:vector

12. Match the constructor's declaration with the most likely code that used it for instantiating the std::vector. Write the line number of the corresponding code in the table below.

Constructor declaration	**Line # (1, 2, or 3)**
vector(size_type count);	01
vector(const std::vector& other);	03
vector(std::initializer_list<T> init);	02

13. Rewrite the code we used to create a std::vector with the number of visitors per hour. We will use the constructor taking an *initialization list* parameter (see line 02 for an example) to shorten the code. Specifically, instantiate the std::vector with the following values during construction: 2, 4, 1, 4, 3, 2, 1, 1. No need to write code to display information on the screen.

```
int main() {
  std::vector<int> customers_per_hour {2, 4, 1, 4, 3, 2, 1, 1};

  return 0;
}
```

14. *copy constructors* allow us to create new std::vectors whose values are copied from an existing std::vector. Write code that will copy the contents of an existing std::vector (hours_placeholder) into a new std::vector using a *copy constructor* (see line 03 for an example). You can use any name for your new std::vector.

```
int main() {
  std::vector<int> hours_placeholder(8);
  std::vector<int> actual_hours(hours_placeholder);


  return 0;
}
```

Reflector questions

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

2. What previous discussion helped in learning about creating and using std::vectors?

3. What is the importance of member function declarations?

4. What did the team do well in this activity?

5. What challenges did the team experience in this activity?

6. If your team experienced challenges, what strategies can you try in the next activity?