Chapter 7 Arrays

7.1 What are arrays?

Part One: Definition
Most variables only hold one value at a time.

Ex: int count = 4;
-This stores one value, 4

Ex2: float price = 499;
-This stores one value, 499

Arrays work like a variable that can store several values. The values are stored together like a file.

Part Two: What are the parts of an array?
One: To make an array you need the data-type, then the name of the array, then the size of the array (how many variables will be in the array) in brackets, like these [ ]
Format: dataType arrayName [sizeofArray]

Ex: int days [6]

Term 1: Size Declaration

Size Declaration: indicates the number of elements the array can hold.

1. It's the value put in brackets.
2. Can be a literal (like 6) or a constant

Literal:
```
int days [6];
```

Constant:
```
numberofDays = 6;
Int days [numberofDays];
```

*In both cases the days array has a size of 6. But one used a constant, and one used a literal value.

Term 2: Elements

Elements: the values that are stored in the array.

Two: The value of the arrays are stored in the computer together as a group. This is what arrays look like where it is stored as a group in the memory of a computer:

1st value	2nd value	3rd value	4th value	5th value	6th value
Element 0	Element 1	Element 2	Element 4	Element 4	Element 5

Notes:
The individual values of the array (the elements) can be specifically named by what order they are in the array.
Array numbering starts with 0, so the first value is element [0].

Part 3: Memory Requirements of Arrays:

The amount of memory used for an array can be found by this formula

Array memory = (size of element)(bytes taken up for the here and now)

Ex: int days [6];

- 6 is the size of the array
- Int is the data type that takes up size four amounts of space.

Uses 4 bytes	Uses 4 bytes	Uses 4 bytes	Uses 4 bytes	Uses 4 bytes	Uses 4 bytes
Element 0	Element 1	Element 2	Element 3	Element 4	Element 5

6 * 4=24, so the days array takes up 24 bytes of memory.

7.2 Accessing Array Elements

How can I access/use specific values contained in an array?
Part 1: Each element in an array has a subscript, that categorizes the element from first to last. The elements can be accessed and used as a variable by accessing that specific element within the array.

This is the1st value	This is the 2nd value	This is the 3rd value	This is the 4th value	This is the 5th value	This is the 6th value
0	1	2	3	4	5

Point Two:Remember: subscripts always start at zero, not one. So an array of element 6 has subscript number 0-5, not 1-6.

Point three: how to access an element.
This is the format to access an element.

arrayName[number]

Once an element is accessed on its own, you can treat it the same way as if it were a variable. For example, if I were to assign the value 45 to all elements of the array int days = [6], the elements would look like this:

```
Int days[0]=45;
Int days[1]=45;
Int days[2]=45;
Int days[3]=45;
Int days[4]=45;
Int days[5]=45;
```

Point four:The array size declarator is not the same thing as the array subscript.

Array size declarator: the brackets around 5 Int hours [5]; The brackets here determine the size of the array.	Array subscript: the brackets around 5 Hours [5]; The brackets here help define the 6th variable in the array order (because arrays start with the number zero).

Point 5: Default values for arrays.
If the values in an array aren't initialized in a globally defined array, then the all the elements are automatically zero. If the values in an array aren't defined in a locally defined array, then there is no initialization to the values.

7.3 No Bounds Checking in C++

Point One: C++ doesn't have area bounds checking, which means that a person could write a program with a subscript of an array that doesn't actually exist within the size of the array. This is common when people forget that the numbering of an array starts with zero, not one.

Point Two: A program trying to write into memory outside the array will most likely cause the program to crash, so it's really important that you don't try to access a variable that doesn't exist within an array.

Example:

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

int main()
{
const int SIZE = 3; // Constant for the array size
int values[SIZE]; // An array of 3 integers
int count; // Loop counter variable

for (count = 0; count < 5; count++)
values[count] = 100;

Return 0;
}
```

What's Wrong:

1. The program tries to assign the value 100 to the elements of the array values.
2. But the problem is that the values array only has 3 elements
3. This means that there is only 3 spaces for the array for its memory
4. The test part of the for loop has the loop go all the way up to values [3] and values[4], both of which don't exist.
5. The computer likely crashes

Point Three:

	values[0]	*values[1]*	*Values [2]*		

The only values that the array values has to store memory in is values [0] to [2].

	Values [0]	*values[1]*	*values[2]*	**values[3]**	**Values [4]**

But if the program tries to store memory in values [3] and values[4], which don't exist, then the computer is overriding data that already exists in that place, like other variables and arrays that could be there.

Point four: off by one errors are very common: people often forget about the implication that the numbering of array elements starts with the number zero, not one.

Example:

```
// This code has an off-by-one error.

const int SIZE = 100;
int numbers[SIZE];
for (int count = 1; count <= SIZE; count++)
numbers[count] = 0;
```

What's Wrong:

1. The range of the element's names in the numbers array goes from [0] to [99]
2. The for loop starts with one and repeats 100 times because of the constant SIZE in the test part of the loop, so the for loop is assigning values to numbers [1]- [100]
3. [100] doesn't exist, so the computer would probably crash because the variable is off by one and the computer would overwrite the computer's memory for other things not related to the array

7.4 Array Initialization

How can I enter in values to an array all at once?

Point One: You can initialize arrays similar to how you initialize variables. Intead of just one value like variables, you put curly brackets around the group of numbers.

Example:
numWeeks = 7;
int dayofTheWeekRating[numWeeks] = {10,9, 8, 7, 6, 5, 4}

Point 2: Initialization List
Initialization list: the list of numbers that you initialize an array with.

- The values are entered in the array in the order that they are entered in the initialization

Example:

dayofTheWeekRating[0] = 10;
dayofTheWeekRating[1] = 9;
dayofTheWeekRating[2] = 8;
dayofTheWeekRating[3] = 7;
dayofTheWeekRating[4] = 6;
dayofTheWeekRating[5] = 5;
dayofTheWeekRating[6] = 4;

Point 3: Initializing the array not completely is okay. Any values that are not initialized on purpose will initialize to zero by default. However, you can't skip values

Example:
num_people = 8;
peopleHungerMeter[num_people] = {4,6,4}

These are the values of peopleHungerMeter: 4 6 4 0 0 0 0 0

*Note: you can't skip values for intializing arrays:

Example: int numbers[6] = {2, 4, , 8, , 12};
-This is not legal, and wont work

Point 4: Implicit Array Sizing.
You don't have to enter the size of the array if there is an initialization list for the array that has ALL the values.

Example:

double ratings[] = {1.0, 1.5, 2.0, 2.5, 3.0};

-This array ends up having a size of 5 because there are 5 values included in the initialization list and no size declarator

7.5 The Range-Based for Loop

How can I develop a code that lets me interact with every element in an array automatically?
Point 1: The range-based for loop makes processing entire arrays easier. It enables the programmer to interact with a whole array automatically.

Point 2: here is the format of the range based for loop:

```
for ( dataType rangeVariable : array )
        Statement;
```

dataType is the data type of the range variable. It must be the same as the data type of the array elements, or a type that the elements can automatically be converted to.
rangeVariable is the name of the range variable. This variable will receive the value of a different array element during each loop iteration. During the first loop iteration, it receives the value of the first element; during the second iteration, it receives the value of the second element, and so forth.
array is the name of an array on which you wish the loop to operate. The loop iterates once for every element in the array.
statement is what the loop uses the array for. If you need to execute more than one statement in the loop, enclose the statements in a set of braces.

Example:

Here is an array:

```
int numbers[] = { 3, 6, 9 };
```

Here is a range based loop that will print out each value:

```
for (int val : numbers)
cout << val << endl;
```

- Int is the data type,
- val is the range variable that prints out the values of the numbers array,
- numbers is the array,
- and the statement is the second line (cout).

Point 3: Changing an array with a range based for loop
You can change an array using range based for loops by making the range variable into a reference for the array's values so that whatever happens to the range variable happens to the array's values.
You can make the range variable into a reference by putting an ampersand in front of the range variable in the loop header.

Example:

```
1 // This program uses a range-based for loop to
 2 // modify the contents of an array.
 3 #include <iostream>
 4 using namespace std;
 5
 6 int main()
 7 {
 8 const int SIZE = 5;
 9 int numbers[5];
 10
 11 // Get values for the array.
 12 for (int &val : numbers)
 13 {
 14 cout << "Enter an integer value: ";
 15 cin >> val;
 16 }
 17
 18 // Display the values in the array.
 19 cout << "Here are the values you entered:\n";
20 for (int val : numbers)
 21 cout << val << endl;
 22
23 return 0;
24 }
```

Because val is a reference variable, any changes made to the range variable will be made to the array element it references.

Point 4: Range Based For Loop vs. Regular for Loop
Range Based for loops are helpful where you need to step through every element of the array without using element subscripts. But they won't work when you need the element subscript for some purpose; then the normal for loop is needed.

Range Based For Loops: **Helpful where you don't need to use subscripts**	**For Loops:** **When you need to use subscripts for some purpose**

7.6 Processing Array Contents

Array elements can be processed like any other variable.

Point 1: You can perform math operations on array elements like variables
Ex:
pay = hours[3] * rate;

Point 2: You can use increment/decrement operators on array elements

Ex:
int score[5] = {7, 8, 9, 10, 11};
++score[2]; // Pre-increment operation on the value in score[2]
score[4]++; // Post-increment operation on the value in score[4]

Point 3: You can use relational expressions on array elements

Ex: These array elements are used in the heading of an if loop:
if (cost[20] < cost[0])

Point 4: The only way to assign one array to another is to assign the individual elements (usually loops work well for this) of one array to the individual elements of the other array; you can't just equalize two different arrays.

Why? When an array is used without brackets or subscripts, it is seen by the computer as the array's beginning memory address (the point where the array is stored in the computer). Because the name of the array stands for the starting memory address, each array has a different number.

Example:
Say the array NewPrices of element size 4 exists. That means it's stored somewhere, the memory address is 2048.This means that

newPrices = 2048

Another array OldPrices of element size 4 also exists, but it is stored in a different spot, its memory address is 2206. this means that
OldPrices = 2206

You can't do this:

```
const int SIZE = 4;
int oldValues[SIZE] = {10, 100, 200, 300};
int newValues[SIZE];
```

```
newValues = oldValues
```

Because the last expression says:

2048 = 2206

Which isn't true. A for loop works better, to assign **individual** elements:

```
for (int count = 0; count < SIZE; count++)
        newValues[count] = oldValues[count];
```

Point 5: You can Print the contents of an array by printing each array element individually

```
const int SIZE = 5;
int numbers [SIZE] = {10, 20, 30, 40, 50};
```

WRONG	RIGHT

cout << numbers << endl; //Wrong!
*This prints the memory location of the numbers array, not the elements of the array
	for (int count = 0; count < SIZE; count++)
	cout << numbers[count] << endl;
	*You need to use a loop to print out the numbers one by one

Point 6: You can add all the values of an array together by adding each element together

Accumulator Variable: A variable that the loop adds to each iteration so that the total sum of all the values equals the accumulator variable by the end of the loop

Example:

```
const int NUM_UNITS = 24;
int units[NUM_UNITS];
int total = 0;
 // Initialize accumulator
for (int count = 0; count < NUM_UNITS; count++)
        total += units[count];
```

Point 7: You can take the average of a set of elements in an array by finding the sum and then dividing the sum by the size of the array as shown by the size declarator

```
const int NUM_SCORES = 10;
double scores[NUM_SCORES];
double total = 0;
 // Initialize accumulator double average;
 // Will hold the average
for (int count = 0; count < NUM_SCORES; count++)
        total += scores[count];
```

```
average = total / NUM_SCORES;
```

Point 8: You can find the highest and lowest values in an array

Instructions:

1. copy the value in the first array element to the variable highest
2. loop compares all of the remaining array elements, beginning at subscript 1, to the value in highest
3. Each time it finds a value in the array that is greater than highest, it copies that value to highest.
4. When the loop has finished, highest will contain the highest value in the array.
5. The process is the same for the lowest value.

Example:

```
const int SIZE = 50;
int numbers[SIZE];
int count;
 int highest;
 highest = numbers[0];
 for (count = 1; count < SIZE; count++)
 { if (numbers[count] > highest)
highest = numbers[count]; }
```

Point 9: If you don't know the exact number of elements needed for an array, then you can always make a really large array so that it will always be big enough. However, this leads to only a partially filled array.

Instructions:

1. Make an accompanying integer variable that holds the number of items stored in the array
2. Each time we add an item to the array, we must increment count
3. Each iteration of this sentinel-controlled loop allows the user to enter a number to be stored in the array, or −1 to quit
4. When the user enters −1, or count exceeds a certain number, the loop stops

Example:

```
const int SIZE = 100;
int numbers[SIZE];
int count = 0;
int num; cout << "Enter a number or −1 to quit: ";

cin >> num;
while (num != −1 && count < SIZE)
{
 count++; numbers[count − 1] = num;
cout << "Enter a number or −1 to quit: ";
cin >> num;
}
```

Point 10: Comparing Arrays: to compare the content of two arrays, you have to compare the elements of the two arrays individually.

```
const int SIZE = 5;
 int firstArray[SIZE] = { 5, 10, 15, 20, 25 };
 int secondArray[SIZE] = { 5, 10, 15, 20, 25 };
bool arraysEqual = true;
// Flag variable int count = 0;
// Loop counter variable // Determine whether the elements contain the same data.

while (arraysEqual && count < SIZE)
{
          If (firstArray[count] != secondArray[count])
                    arraysEqual = false;
          count++;
 }
 if (arraysEqual)
          cot << "The arrays are equal.\n";
Else
          cout << "The arrays are not equal.\n";
```

7.7 Focus on Software Engineering: Using Parallel Arrays

Point 1: sometimes its useful to store data in 2 or more arrays especially when the data is of unlike datatypes.
Example:

```
1 // This program uses two parallel arrays: one for hours
2 // worked and one for pay rate.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 int main()
8 {
9 const int NUM_EMPLOYEES = 5; // Number of employees
10 int hours[NUM_EMPLOYEES]; // Holds hours worked
11 double payRate[NUM_EMPLOYEES]; // Holds pay rates
12
13 // Input the hours worked and the hourly pay rate.
14 cout << "Enter the hours worked by " << NUM_EMPLOYEES
15 << " employees and their\n"
16 << "hourly pay rates.\n";
17 for (int index = 0; index < NUM_EMPLOYEES; index++)
18 {
19 cout << "Hours worked by employee #" << (index+1) << ": ";
20 cin >> hours[index];
21 cout << "Hourly pay rate for employee #" << (index+1) << ": ";
22 cin >> payRate[index];
23 }
24
25 // Display each employee's gross pay.
26 cout << "Here is the gross pay for each employee:\n";
27 cout << fixed << showpoint << setprecision(2);
28 for (int index = 0; index < NUM_EMPLOYEES; index++)
29 {
30 double grossPay = hours[index] * payRate[index];
31 cout << "Employee #" << (index + 1);
32 cout << ": $" << grossPay << endl;
33 }
34 return 0;
35 }
```

The loops used the same subscript to access both arrays. This is because the data stored in the arrays have the same subscript for each employee.

7.8 Arrays as Function Arguments

Passing the array elements by value:	**Pass by reference**
● This is when the elements of the array are passed to the function one by one, using iterations of loops Process of 1. The function is called. 2. a copy of an array element is passed to the parameter variable 3. The function displays the contents of the parameter variable, not the array itself. The array passes it's value to the parameter variable, hence "pass by value"	● This is when the enitre array is passed to the function at once by passing the name of the array and therefore just passing the reference to the starting point of the memory location of the array ● Useful for really large arrays that would take a long time to pass by value

Example of passing by value:

```
//This program demonstrates that an array element is passed
//to a function like any other variable.
#include <iostream>
using namespace std;
void showValue(int); // Function prototype

int main()
{
const int SIZE = 8;
int numbers[SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};

for (int index = 0; index < SIZE; index++)
showValue(numbers[index]);
return 0;
}

//*****************************************
// Definition of function showValue. *
// This function accepts an integer argument. *
// The value of the argument is displayed. *
//*****************************************
void showValue(int num)
{
```

```
cout << num << " ";
}
```

Example of passing by reference:

```
// This program demonstrates an array being passed to a function.
2 #include <iostream>
3 using namespace std;
4
5 void showValues(int [], int); // Function prototype
6
7 int main()
8 {
9 const int ARRAY_SIZE = 8;
10 int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
11
12 showValues(numbers, ARRAY_SIZE);
13 return 0;
14 }
15
16 //**************************************************
17 // Definition of function showValue. *
18 // This function accepts an array of integers and *
19 // the array's size as its arguments. The contents *
20 // of the array are displayed. *
21 //**************************************************
22
23 void showValues(int nums[], int size)
24 {
25 for (int index = 0; index < size; index++)
26 cout << nums[index] << " ";
27 cout << endl;
28 }
```

Using "Const" array parameters:
You can prevent a function from making changes to an array by using the "const" key word in the parameter declaration. You should always use "const" array parameters in any function not intended to modify the original array. The function will fail to compile if you accidentaly write code that would change the array.

Ex:

```
void showValues(const int nums[], int size)
 {
 for (int index = 0; index < size; index++)
```

```
 cout << nums[index] << " ";
cout << endl;
}
```

7.9 Two-Dimensional Arrays

Part 1: What is a 2 dimensional array?

A two dimensional array is like putting several identical arrays together.

One dimensional array: holds one set of data

Two dimensional Array: can hold multiple sets of data

Part 2: How to define a two dimensional array:

int someTwoDimensionalArray[a][b];

*a= rows, b= columns

Part 3: Cycling Through a Two-Dimensional Array

Programs that cycle through 2 dimensional arrays usually do so with nested loops

...(Scroll to next page)

Example:

```
// This program demonstrates a two-dimensional array.
2 #include <iostream>
3 #include <iomanip>
4 using namespace std;
5
6 int main()
7 {
8 const int NUM_DIVS = 3; // Number of divisions
9 const int NUM_QTRS = 4; // Number of quarters
10 double sales[NUM_DIVS][NUM_QTRS]; // Array with 3 rows and 4 columns.
11 double totalSales = 0; // To hold the total sales.
12 int div, qtr; // Loop counters.
13
14 cout << "This program will calculate the total sales of\n";
15 cout << "all the company's divisions.\n";
16 cout << "Enter the following sales information:\n\n";
17
18 // Nested loops to fill the array with quarterly
19 // sales figures for each division.
20 for (div = 0; div < NUM_DIVS; div++)
21 {
22 for (qtr = 0; qtr < NUM_QTRS; qtr++)
23 {
24 cout << "Division " << (div + 1);
25 cout << ", Quarter " << (qtr + 1) << ": $";
26 cin >> sales[div][qtr];
27 }
28 cout << endl; // Print blank line.
29 }
30
31 // Nested loops used to add all the elements.
32 for (div = 0; div < NUM_DIVS; div++)
33 {
34 for (qtr = 0; qtr < NUM_QTRS; qtr++)
35 totalSales += sales[div][qtr];
36 }
37
38 cout << fixed << showpoint << setprecision(2);
39 cout << "The total sales for the company are: $";
40 cout << totalSales << endl;
41 return 0;
42 }
```

Part 4: initialization
To initialize some values to a 2D array, enclose the numbers in brackets, like this:

{3, 4, 5, 35, 34, 23, 34, 23, 1}

It helps (but is optional) to enclose each row's initialization list in a set of braces.

Example: int hours[3][2] = {{8, 5}, {7, 9}, {6, 3}}

Part 5: Passing two dimensional array to functions
When passing an array the parameter for the array MUST have a defined number of columns. As long as an array has a defined number of columns, the array can be accepted by the function.

Why: The computer can store the values in the array by storing each element in a row that is a specified length. The only thing the computer needs to do to change the number of values in an array is increase the amount of rows. The computer stores values in the 2D array by replicating the format of the row.

Example:
void showArray(const int numbers[][COLS], int rows

*This is the header for a function called showArray.

Summing the elements in a 2D array:
You can use nested loops to sum the elements in an array.

Example:
(...scroll down)

```
const int NUM_ROWS = 5; // Number of rows
const int NUM_COLS = 5; // Number of columns
int total = 0; // Accumulator
int numbers[NUM_ROWS][NUM_COLS] = {{2, 7, 9, 6, 4},
 {6, 1, 8, 9, 4},
 {4, 3, 7, 2, 9},
 {9, 9, 0, 3, 1},
 {6, 2, 7, 4, 1}};
// Sum the array elements.
for (int row = 0; row < NUM_ROWS; row++)
 {
 for (int col = 0; col < NUM_COLS; col++)
 total += numbers[row][col];
 }
// Display the sum.
cout << "The total is " << total << endl;
```

You can sum the specific amoutn of each row by using a loop to all the elements in one row.

Example:

```
const int NUM_STUDENTS = 3; // Number of students
const int NUM_SCORES = 5; // Number of test scores
double total; // Accumulator is set in the loops
double average; // To hold each student's average
double scores[NUM_STUDENTS][NUM_SCORES] = {{88, 97, 79, 86, 94},
 {86, 91, 78, 79, 84},
 {82, 73, 77, 82, 89}};
// Get each student's average score.
for (int row = 0; row < NUM_STUDENTS; row++)
 {
 // Set the accumulator.
 total = 0;
 // Sum a row.
 for (int col = 0; col < NUM_SCORES; col++)
 total += scores[row][col];
 // Get the average.
 average = total / NUM_SCORES;
 // Display the average.
 cout << "Score average for student "
 << (row + 1) << " is " << average <<endl;
 }
```

*total is an accumulator variable that is set to zero at the beginning of each iteration of the inner loop. It catches the amount in each loop.)

7.10 Arrays with Three or More Dimensions

You can create any number of dimensions for arrays in C++.
Example: Three dimensional array
double seats[3][5][8];

Note: When writing functions that accept multi-dimensional arrays as arguments, all but the first dimension must be explicitly stated in the parameter list.

7.12 If You Plan to Continue in Computer Science: Introduction to the STL vector

Part 1: The Standard Template Library
Standard Template Library(STL): a collection of data types and algorithms that you may use in your programs.

-not part of the C++ language, created in language along with data types
data types called *containers* because the STL data types organize and collect data
Two types:

1. Sequence containers: organizes data in a sequential fashion (like an array)
2. Associative containers: organize data with keys, which allows random and quick access to elements in the container

Vector data type is a sequence container (like an array, it organizes data sequentially)

Part 2: vectors vs. arrays

Similarities to arrays	Differences from arrays
1. A vector holds a sequence of values, or elements. 2. A vector stores its elements in contiguous memory locations. 3. You can use the array subscript operator [] to read the individual elements in the vector.	1. You do not have to declare the number of elements that the vector will have. 2. If you add a value to a vector that is already full, the vector will automatically increase its size to accommodate the new value. 3. vectors can report the number of elements they contain.

Part 3: defining a vector
To use vectors you need the heading:

#include <vector>
*Note: You need namespace std to use vectors in program

format:
vector<int> numbers;

Note you don't need to include the size of the vector because vectors have no predetermined size and can extend the size if needed

If you wanted the vector to have a specific size:
vector<int> numbers(10);
*note: parentheses, not brackets

Part 4: initialization

1. You can initialize a vector by adding a second number in the parenthesesvector<int> numbers(10, 2);
2. You can initialize a vector with the values of another vector:vector<int> set2(set1); The vector set2 is a copy of set1.
3. You can initialize a group of numbers for a vector using an intialization list: vector<int> numbers { 10, 20, 30, 40 }; this creates a vector with the values 10, 20, 30, and 40

Part 5: Storing and Retrieving Values in a Vector
To store and retrieve a value in a vector, you can use [].

Example:

```
// This program stores, in two vectors, the hours worked by 5
2 // employees, and their hourly pay rates.
3 #include <iostream>
4 #include <iomanip>
5 #include <vector> // Needed to define vectors
6 using namespace std;
7
8 int main()
9 {
10 const int NUM_EMPLOYEES = 5; // Number of employees
11 vector<int> hours(NUM_EMPLOYEES); // A vector of integers
12 vector<double> payRate(NUM_EMPLOYEES); // A vector of doubles
13 int index; // Loop counter
14
15 // Input the data.
16 cout << "Enter the hours worked by " << NUM_EMPLOYEES;
17 cout << " employees and their hourly rates.\n";
18 for (index = 0; index < NUM_EMPLOYEES; index++)
19 {
20 cout << "Hours worked by employee #" << (index + 1);
21 cout << ": ";
22 cin >> hours[index];
23 cout << "Hourly pay rate for employee #";
24 cout << (index + 1) << ": ";
25 cin >> payRate[index];
26 }
27
28 // Display each employee's gross pay.
```

```
29 cout << "\nHere is the gross pay for each employee:\n";
30 cout << fixed << showpoint << setprecision(2);
31 for (index = 0; index < NUM_EMPLOYEES; index++)
32 {
33 double grossPay = hours[index] * payRate[index];
34 cout << "Employee #" << (index + 1);
35 cout << ": $" << grossPay << endl;
36 }
37 return 0;
38 }
```

Part 6: Using the Range Based For Loop

- You can use the range based for loop to step through elements of a vector

Example:

```
vector numbers { 10, 20, 30, 40, 50 };
for (int val : numbers)
cout << val << endl;
```

- You can use a reference variable for a range based for loop to store items in a vector

```
vector<int> numbers(5);
```

Example:

```
for (int &val : numbers)
{
cout << "Enter an integer value: ";
cin >> val;
}
```

1. The function defines numbers as a vector of int with 5 elements
2. There is an ampersand in front of val that declares val as a reference variable
3. This causes the loop to use val as a reference variable for the vector elements

Part 7: Member Functions

Push_back member function

You can't use the [ ] operator for an element that doesn't already exist.
But, you can use the push_back function to:

1. store a value in a vector that doesn't have a starting size.
2. Store a value in a vector that is already full.

What the push_back function does:

The push_back function accepts a value as an argument and stores that value after the very last element in that vector

Example:

```
numbers.push_back(25);
```

The statement stores 25 as the last element. If the vector that it is stored in, the vector numbers, has all 25 elements already storing a value, then a new element is added to the vector (a 2nd element). The value 25 is then stored there, in that added element.

Pop_Back member function

Use the pop_back function to remove elements from a vector.

Format:

```
collection.pop_back();
```

*collection is a vector

Determining the Size of a Vector

Vectors can report the number of elements that they contain using the size member function.

Example:

```
numValues = set.size();
```

numValues is an integer number
Set is a vector
After executing the statement, numValues will then contain the number of elements in set

The size member function is helpful when writing vectors to be used as arguments. If you know the size of a vector, then you don't have to create a second argument for the size in addition to the first argument, the vector's name.

Example:

```
void showValues(vector vect)
{
for (int count = 0; count < vect.size(); count++)
cout << vect[count] << endl;
}
```

Example 2:

```
// This program demonstrates the vector size
2 // member function.
3 #include <iostream>
4 #include <vector>
5 using namespace std;
```

```
6
7 // Function prototype
8 void showValues(vector<int>);
9
10 int main()
11 {
12 vector<int> values;
13
14 // Put a series of numbers in the vector.
15 for (int count = 0; count < 7; count++)
16 values.push_back(count * 2);
17
18 // Display the numbers.
19 showValues(values);
20 return 0;
21 }
22
23 //************************************************
24 // Definition of function showValues. *
25 // This function accepts an int vector as its *
26 // argument. The value of each of the vector's *
27 // elements is displayed. *
28 //************************************************
29
30 void showValues(vector<int> vect)
31 {
32 for (int count = 0; count < vect.size(); count++)
33 cout << vect[count] << endl;
34 }
```

Here is a list of member functions that you can use for vectors:

Member function	**Description**	**Example**
at(element)	Returns the value of the element located at element in the vector.	x = vect.at(5); *This statement assigns the value of the fifth element of vect to x.
capacity( )	Returns the maximum number of elements that may be stored in the	x = vect.capacity();

	vector without additional memory being allocated. (This is not the same value as returned by the size member function).	*This statement assigns the capacity of vect to x.
clear( )	Clears a vector of all its elements. Example: vect.clear(); This statement removes all the elements from vect.	vect.clear(); *This statement removes all the elements from vect.
empty( )	Returns true if the vector is empty. Otherwise, it returns false..	if (vect.empty()) cout << "The vector is empty."; *This statement displays the message if vect is empty
pop_back( )	Removes the last element from the vector.	vect.pop_back(); *This statement removes the last element of vect, thus reducing its size by 1
push_back(value)	Stores a value in the last element of the vector. If the vector is full or empty, a new element is created.	vect.push_back(7); *This statement stores 7 in the last element of vect.
reverse( )	Reverses the order of the elements in the vector. (The last element becomes the first element, and the first element becomes the last element.)	vect.reverse(); *This statement reverses the order of the element in vect.
resize(someElementSize, someValue)	Resizes a vector by elements. Each of the new elements is initialized with the value in value.	vect.resize(5, 1); *This statement increases the size of vect by five elements. The five new elements are initialized to the value 1.
swap(someVector)	Swaps the contents of the vector with	vect1.swap(vect2);

	the contents of vector2.	*This statement swaps the contents of vect1 and vect2