Chapter 8 - Arrays

The big idea behind this chapter is ....

It relates to the previous chapter how ...

The main purpose of this chapter is ...

The key questions are ...

Why:

When:

How:

Why is this material at this point in the class?

You'll know this material when ...

Main assumptions are ...

Opening Thoughts. *Write any thoughts or questions you have before reading this material. See if you can find the answers while you read.*

Key Ideas. *Record major points from the chapter.*

Idea	Notes

List of Figures

Demonstration Code.

```cpp
// 8-1.cpp -- Intro to arrays
#include <iostream>
using namespace std;

int main() {
    const int NUM_EMPLOYEES = 6;
    int hours[NUM_EMPLOYEES];

    cout << "Enter hours worked by "
        << NUM_EMPLOYEES << " employees: ";
    cin >> hours[0];
    cin >> hours[1];
    cin >> hours[2];
    cin >> hours[3];
    cin >> hours[4];
    cin >> hours[5];

    cout << "The hours you entered are: ";
    cout << " " << hours[0];
    cout << " " << hours[1];
    cout << " " << hours[2];
    cout << " " << hours[3];
    cout << " " << hours[4];
    cout << " " << hours[5] << endl;

    return 0;
}
```

Figure 1: §8.3 Array use case. Source file: 8-1.cpp

```cpp
// 8-2.cpp -- Array with loops
#include <iostream>
using namespace std;

int main() {
    const int NUM_EMPLOYEES = 6;
    int hours[NUM_EMPLOYEES];

    cout << "Enter the hours worked by "
        << NUM_EMPLOYEES << " employees: ";

    for (int i = 0; i < NUM_EMPLOYEES; i++)
        cin >> hours[i];

    cout << "The hours you entered are: ";

    for (int i = 0; i < NUM_EMPLOYEES; i++)
        cout << " " << hours[i];

    cout << endl;
    return 0;
}
```

Figure 2: §8.3 Array with loop demo. Source file: 8-2.cpp

```cpp
// 8-4.cpp -- Demo of out of bounds errors
#include <iostream>
using namespace std;

int main() {
    const int SIZE = 3;
    int A[SIZE] = { 1, 1 ,1};
    int B[SIZE];

    cout << "Original data in in array A\n";
    for (int count = 0; count < 0; count++)
        cout << A[count] << "\t";

    cout << "\n\n Now store 7 elements in the second 3 element array\n";
    for (int count = 0; count < 7; count++)
        B[count] = 5;

    cout << "If the program did not crash, try to display the array\n";
    for (int count = 0; count < 7; count++)
        cout << B[count] << "\t";

    cout << "\nDisplay array A.\n";
    for (int count = 0; count < 3; count++)
        cout << A[count] << "\t";

    cout << "\n\nArray A's values were overwritten by \n"
        << "the values overflowing array B\n";

    return 0;
}
```

Figure 3: §8.3 Demo out of bounds error. Source file: 8-4.cpp

```cpp
//8-9.cpp -- range based for loop demo
#include <iostream>
#include <string>
using namespace std;

int main() {
    string planets[] = {    "Mercury", "Venus", "Earth", "Mars",
                            "Jupiter", "Saturn", "Uranus",
                            "Neptune", "Pluto"};

    cout << "Here are the planets\n";
    for (string planet : planets)
        cout << planet << endl;

    return 0;
}
```

Figure 4: §8.5 Range based loop example. Source file: 8-9.cpp

```cpp
#include <iostream>

int main() {
  int num;
  std::cout << "Enter size: ";
  std::cin >> num;
  int quizzes[num];
  std::cout << "Hey! I compile!" << std::endl;
}
```

Figure 5: §8.x1 Demo of a variable based array. Source file: noName.cpp

```cpp
// arrStuff.cpp -- stuff to do with an array
#include <iostream>
using namespace std;

int main() {
  const int SIZE = 5;
  int aArray[] = {10, 20, 30, 40, 50};
  int bArray[] = {0, 0, 0, 0, 0};

  for (int e : aArray) cout << e << ", ";
  cout << endl;
  for (int e : bArray) cout << e << ", ";
  cout << endl;

  cout << "Array copy\n";
  // Can't do bArray = aArray
  for (int i = 0; i < SIZE; i++)
    bArray[i] = aArray[i];

```

Figure 6: §8.6 How to process array elements. Source file: arrStuff.cpp

```cpp
// 8-14(mod).cpp -- process a string as an array
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

int main() {
  char ch;
  int vowelCount = 0;
  string sentence;

  cout << "Enter a sentence" << endl;
  getline(cin, sentence);

  for (int pos = 0; pos < sentence.length(); pos++) {
    ch = toupper(sentence[pos]);

    switch(ch) {
      case 'A':
      case 'E':
      case 'I':
      case 'O':
      case 'U': vowelCount++;
    }
  }

  cout << "There are " << vowelCount << " vowels in this sentence.\n";

  return 0;
}
```

Figure 7: §8.6 Process string as an array demo. Source file: 8-14(mod).cpp

```cpp
// paraMo.cpp -- C++ parallel arrays
#include <iostream>

using namespace std;

int main() {
    string names[] = {"BAD", "January", "February", "March", "April", "May", "June",
            "July", "August", "September", "October", "November",
            "December"};
    int days[] = {0, 30, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    int mo = 0;
    cout << "Enter a month number (1-12): ";
    cin >> mo;

    cout << names[mo] << " has " << days[mo] << " days in it\n";
//  cout << names[mo+1] << " has " << days[mo+1] << " days in it\n";

    return 0;
}
```

Figure 8: §8.7 Parallel array demo. Source file: paraMo.cpp

```cpp
// 8-16.cpp -- pass array elements
#include <iostream>
using namespace std;

void ShowValue(int);

int main() {
  int collection[]{5, 10, 15, 20, 25, 30, 35, 40};

  for (int c : collection)
    ShowValue(c);
  cout << endl;

  return 0;
}

void ShowValue(int num) {
  cout << num << " ";
}
```

Figure 9: §8.9 Example of passing an array. Source file: 8-16.cpp

```cpp
// 8-17.cpp -- pass entire array
#include <iostream>
using namespace std;

void ShowValue(int intArray[], int size);

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

  ShowValue(collection, SIZE);
  cout << endl;

  return 0;
}

void ShowValue(int nums[], int size) {
  for (int i = 0; i < size; i++)
    cout << nums[i] << " ";
  cout << endl;
}
```

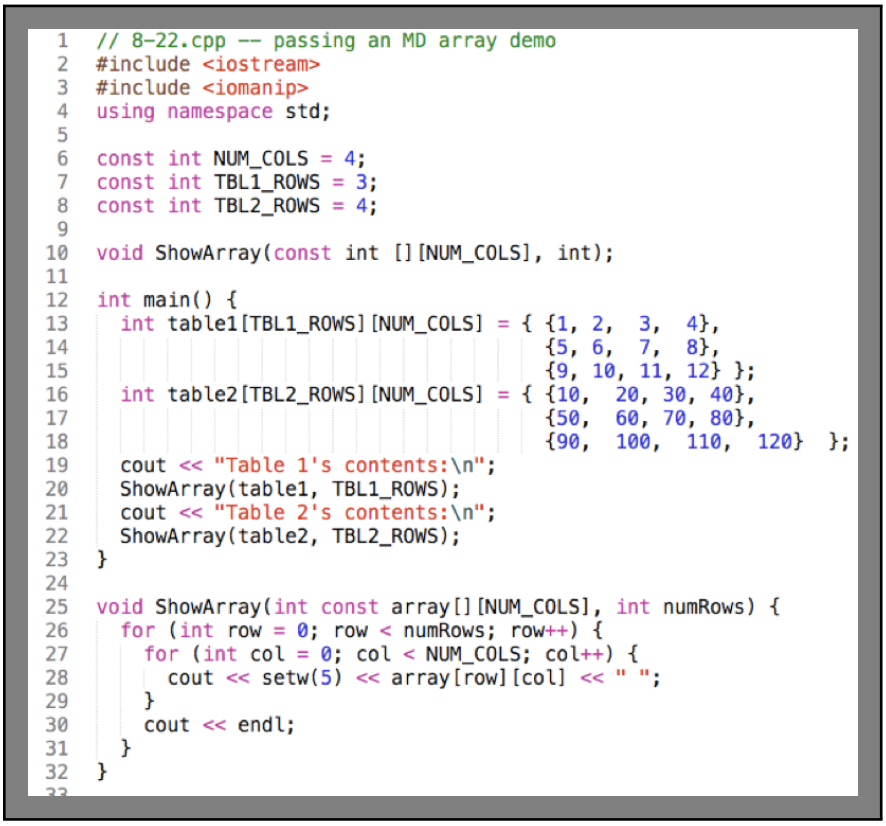
Figure 10: §8.9 Pass by reference. Source file: 8-17.cpp

```cpp
// 8-18.cpp -- array and typedef
#include <iostream>
using namespace std;

typedef int arrayType[];

void ShowValue(arrayType, int);

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

  ShowValue(collection, SIZE);

  return 0;
}

void ShowValue(arrayType nums, int size) {
  for (int i = 0; i < size; i++)
    cout << nums[i] << " ";
  cout << endl;
}
```

Figure 11: §8.9 TypeDef and arrays. Source file: 8-18.cpp

```cpp
// mdArray.cpp -- MD array demo
#include <iostream>
#include <iomanip>
using namespace std;

const int NUM_COLS = 4;
const int TBL1_ROWS = 3;


int main() {
  int table1[TBL1_ROWS][NUM_COLS] = { {1, 2,  3,  4},
                                      {5, 6,  7,  8},
                                      {9, 10, 11, 12} };
  cout << "Table 1's contents:\n";
  for (int row = 0; row < TBL1_ROWS; row++) {
    for (int col = 0; col < NUM_COLS; col++) {
      cout << setw(5) << table1[row][col] << " ";
    }
    cout << endl;
  }
}
```

Figure 12: §8.10 Multidimensional array demo. Source file: mdArray.cpp

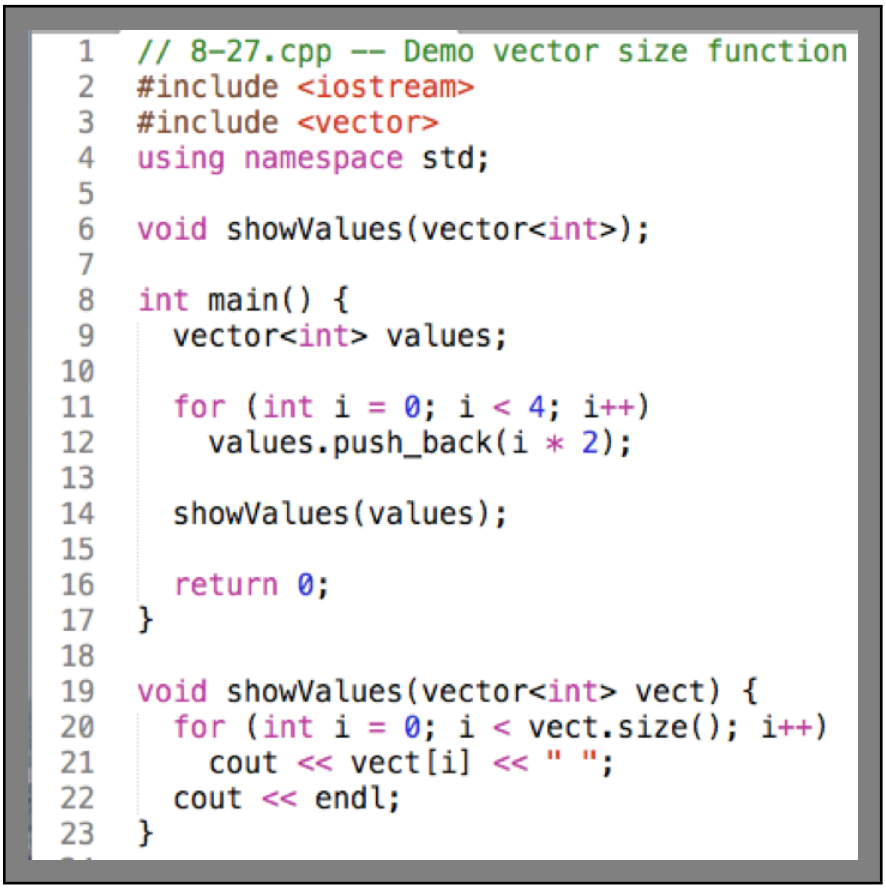


```cpp
// 8-22.cpp -- passing an MD array demo
#include <iostream>
#include <iomanip>
using namespace std;

const int NUM_COLS = 4;
const int TBL1_ROWS = 3;
const int TBL2_ROWS = 4;

void ShowArray(const int [][NUM_COLS], int);

int main() {
  int table1[TBL1_ROWS][NUM_COLS] = { {1, 2,  3,  4},
                                      {5, 6,  7,  8},
                                      {9, 10, 11, 12} };
  int table2[TBL2_ROWS][NUM_COLS] = { {10,  20, 30, 40},
                                      {50,  60, 70, 80},
                                      {90,  100,  110,  120}  };
  cout << "Table 1's contents:\n";
  ShowArray(table1, TBL1_ROWS);
  cout << "Table 2's contents:\n";
  ShowArray(table2, TBL2_ROWS);
}

void ShowArray(int const array[][NUM_COLS], int numRows) {
  for (int row = 0; row < numRows; row++) {
    for (int col = 0; col < NUM_COLS; col++) {
      cout << setw(5) << array[row][col] << " ";
    }
    cout << endl;
  }
}
```

Figure 13: §8.10 How to pass a MD array. Source file: 8-22.cpp

```cpp
// 8-26(mod).cpp -- Demo adding elements to vector
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> numbers;
  int num = 0;
  int tempNum = 0;

  cout << "How many numbers to enter: ";
  cin >> num;

  for (int i = 0; i < num; i++) {
    cout << "Enter number " << (i + 1) << ": ";
    cin >> tempNum;
    numbers.push_back(tempNum);
  }

  cout << "Here's the values: \n";

  for (int val : numbers)
    cout << val << "  ";
  cout << endl;
  return 0;
}
```

Figure 14: §8.12 Vector demo. Source file: 8-26(mod).cpp



```cpp
// 8-27.cpp -- Demo vector size function
#include <iostream>
#include <vector>
using namespace std;

void showValues(vector<int>);

int main() {
  vector<int> values;

  for (int i = 0; i < 4; i++)
    values.push_back(i * 2);

  showValues(values);

  return 0;
}

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

Figure 15: §8.12 Vector size() function.
Source file: 8-27.cpp

```cpp
// 8-28(mod).cpp -- Vector demo size, push_back, pop_back
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> values;
  values.push_back(1);
  values.push_back(2);
  values.push_back(3);
  cout << "The size of values is " << values.size() << endl;

  cout << "Popping a value\n";
  values.pop_back();
  for (int i = 0; i < values.size(); i++)
    cout << values[i] << " ";
  cout << endl;

  cout << "Popping another value\n";
  values.pop_back();
  for (int i : values)
    cout << i << " ";
  cout << endl;

  cout << "Popping the last value\n";
  values.pop_back();
  if (values.empty())
    cout << "Vector is empty";
  else
    for (int i : values)
      cout << i << " ";
  cout << endl;

  return 0;
}
```

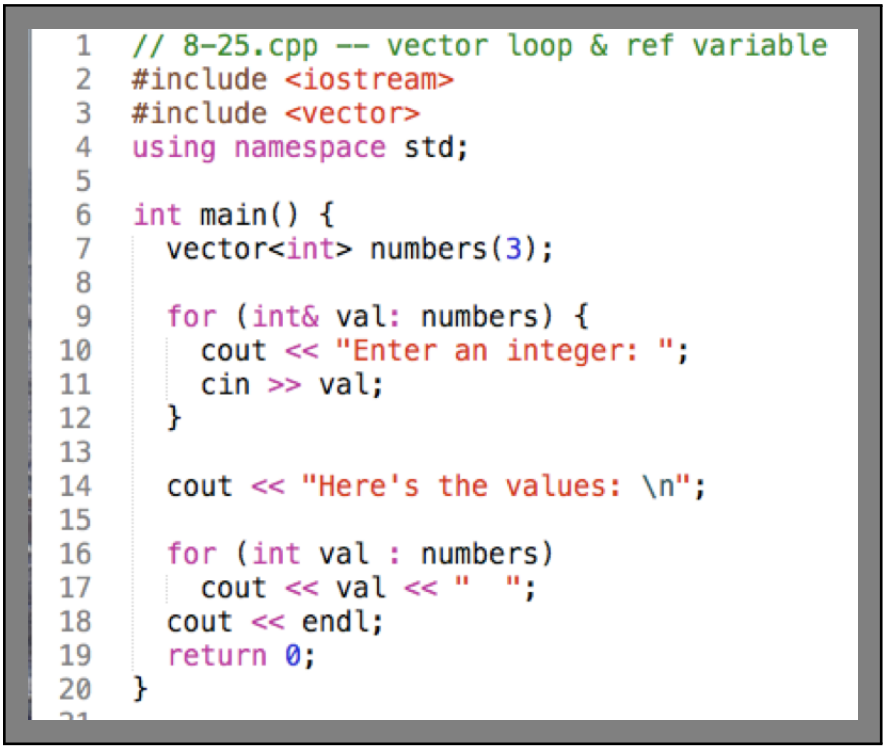
Figure 16: §8.12 Push() and Pop() demo.
Source file: 8-28(mod).cpp

```cpp
// 8-29.cpp -- Demo of vector clear function
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> values(100);

  cout << "The values vector has "
    << values.size() << " elements.\n";
  cout << "I now call the clear method.\n";
  values.clear();
  cout << "The values vector has "
    << values.size() << " elements.\n";

  return 0;
}
```

Figure 17: §8.12 Vector clear() demo..
Source file: 8-29.cpp

```cpp
// 8-24.cpp -- Parallel vectors
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;

int main() {
  const int NUM_EMPS = 5;
  vector <int> hours(NUM_EMPS);
  vector <double> payRate(NUM_EMPS);
  double grossPay;

  cout << "Enter the hours worked and the hourly pay"
    << " rates of " << NUM_EMPS << " employees.\n";
  for (int i = 0; i < NUM_EMPS; i++) {
    cout << "\nHours worked by employee #"
      << (i + 1) << ": ";
    cin >> hours[i];
    cout << "Hourly pay rate for this employee: $";
    cin >> payRate[i];
  }

  cout << "\nGross pay per employee:\n";
  cout << fixed << showpoint << setprecision(2);
  for (int i = 0; i < NUM_EMPS; i++) {
    grossPay = hours[i] * payRate[i];
    cout << "Employee #" << (i + 1);
    cout << ": $" << setw(7) << grossPay << endl;
  }
  return 0;
}
```

Figure 18: §8.12 Parallel vectors. Source file: 8-24.cpp



```
// 8-25.cpp -- vector loop & ref variable
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> numbers(3);

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

  cout << "Here's the values: \n";

  for (int val : numbers)
    cout << val << "  ";
  cout << endl;
  return 0;
}
```

Figure 19: §8.12 Vector loop with reference variable. Source file: 8-25.cpp