Array

Array

- Part 1

- Accessing array
- Array and loop

arrays must be used with loops

- Array and bound checking

Be careful of array bound: invalid subscripts => corrupt memory; cause bugs

- Array initialization
- Processing array contents
- Arrays and operators; arrays assignment

Array

Array Part 2

Array operations

– Printing array content; Sum and average value; Find highest and lowest value; comparing two arrays

- Parallel Array
- Array and Function

Array Part 3

- 2-dimensional arrays
- 2-dimensional array operations
- Arrays of Strings
- N-dimensional arrays

Printing, sum and average array elements, finding highest and lowest values, partial filled arrays, comparing arrays

ARRAY OPERATIONS

Printing the Contents of an Array

- You can display the contents of a *character* array by sending its name to cout:

```
char fName[] = "Henry";
cout << fName << endl;
```

But, this ONLY works with character arrays!

Printing the Contents of an Array

- For other types of arrays, you must print element-by-element:

```
for (i = 0; i < ARRAY_SIZE; i++)
   cout << tests[i] << endl;
```

Summing and Averaging Array Elements

- Use a simple loop to add together array elements:

```
int tnum;
double average, sum = 0;
for(tnum = 0; tnum < SIZE; tnum++)
      sum += tests[tnum];
```

- Once summed, can compute average:

```
average = sum / SIZE;
```

Finding the Highest Value in an Array

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

When this code is finished, the `highest` variable will contain the highest value in the `numbers` array.

Finding the Lowest Value in an Array

```
int count;
int lowest;
lowest = numbers[0];
for (count = 1; count < SIZE; count++)
{
   if (numbers[count] < lowest)
      lowest = numbers[count];
}
```

When this code is finished, the `lowest` variable will contain the lowest value in the `numbers` array.

Partially-Filled Arrays

- If it is unknown how much data an array will be holding:
  - Make the array large enough to hold the largest expected number of elements.
  - Use a counter variable to keep track of the number of items stored in the array.

Comparing Arrays

- To compare two arrays, you must compare element-by-element:

```
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
// Compare the two arrays.
while (arraysEqual && count < SIZE)
{
   if (firstArray[count] != secondArray[count])
      arraysEqual = false;
   count++;
}
if (arraysEqual)
   cout << "The arrays are equal.\n";
else
   cout << "The arrays are not equal.\n";
```

In-Class Exercise

- Given the following array definition:

```
int values[] = {2,6,10,14};
```

What does each of the following display?

a)
```
cout<<values[2];
```

b)
```
cout<<++values[0];
```

c)
```
cout<< values[1]++;
```

d)
```
x = 2;
cout<<values[++x];
```

In-Class Exercise

- Do Lab 12, Exercise 2, No. 1 (pg. 176)
- Do Lab 12, Exercise 2, No. 2 (pg. 176)
- Do Lab 12, Exercise 2, No. 3 (pg. 177)
- Do Lab 12, Exercise 2, No. 4 (pg. 178)
- Declare an integer array named `names` with 20 elements. Write a loop that prints each element of the array.

In-Class Exercise

- Write a program that lets the user enter 10 values into an array. The program should then display the largest and smallest values stored in the array.
- Write a program that lets the user enter the total rainfall for each of 12 months into an array of doubles. The program should then calculate and display the total rainfall for the year, the average monthly rainfall, and the months with the highest and lowest amounts.

  Input Validation: Do not accept negative numbers for monthly rainfall figures.

PARALLEL ARRAY

Using Parallel Arrays

- Parallel arrays: two or more arrays that contain related data
- A subscript is used to relate arrays: elements at same subscript are related
- Arrays may be of different types

Parallel Array Example

```
const int SIZE = 5;    // Array size
int id[SIZE];          // student ID
double average[SIZE]; // course average
char grade[SIZE];      // course grade
...
for(int i = 0; i < SIZE; i++)
{
    cout << "Student ID: " << id[i]
         << " average: " << average[i]
         << " grade: " << grade[i]
         << endl;
}
```

Parallel Array Example

Program 7-12

```
1  // This program stores, in an array, the hours worked by 5
2  // employees who all make the same hourly wage.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9     const int NUM_EMPLOYEES = 5;
10    int hours[NUM_EMPLOYEES];         // Holds hours worked
11    double payRate[NUM_EMPLOYEES];    // Holds pay rates
12
13    // Input the hours worked.
14    cout << "Enter the hours worked by " << NUM_EMPLOYEES;
15    cout << " employees and their\n";
16    cout << "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
```


(Program Continues)

Parallel Array Example

Program 7-12 *(Continued)*

```
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 (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   }
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 5 employees and their
hourly pay rates.
Hours worked by employee #1: 10 [Enter]
Hourly pay rate for employee #1: 9.75 [Enter]
Hours worked by employee #2: 15 [Enter]
Hourly pay rate for employee #2: 8.62 [Enter]
Hours worked by employee #3: 20 [Enter]
Hourly pay rate for employee #3: 10.50 [Enter]
Hours worked by employee #4: 40 [Enter]
Hourly pay rate for employee #4: 18.75 [Enter]
Hours worked by employee #5: 40 [Enter]
Hourly pay rate for employee #5: 15.65 [Enter]
```

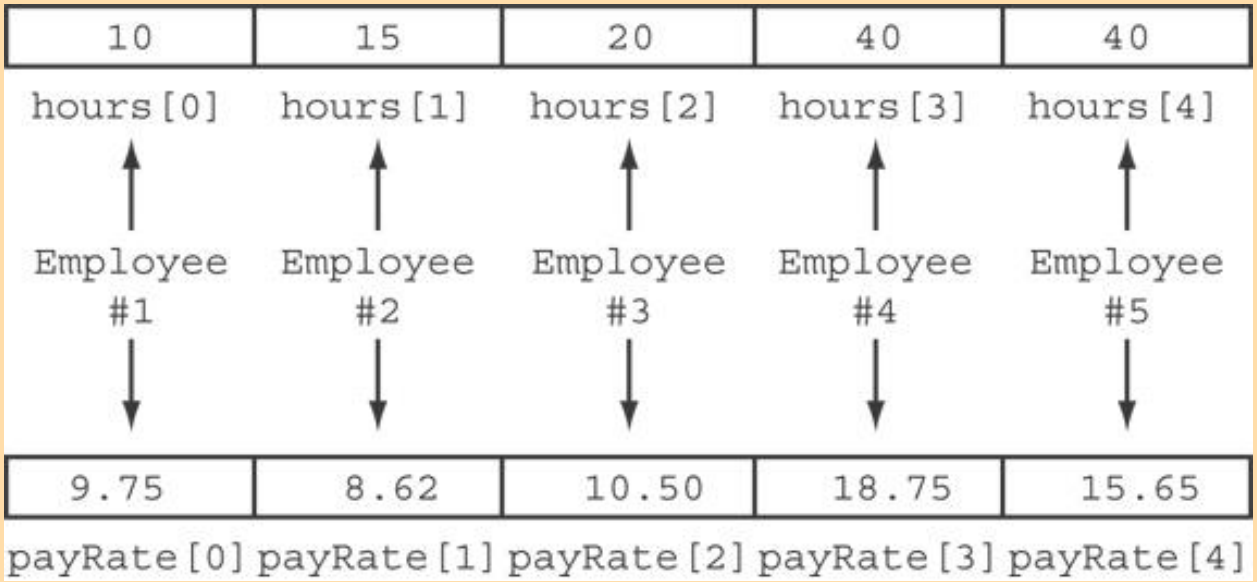
(program output continues)

Parallel Array Example

Program 7-12 *(continued)*

```
Here is the gross pay for each employee:
Employee #1: $97.50
Employee #2: $129.30
Employee #3: $210.00
Employee #4: $750.00
Employee #5: $626.00
```

The `hours` and `payRate` arrays are related through their subscripts:



In-Class Exercise

- What is the output of the following code? (You may need to use a calculator.) .

```
const int SIZE = 5;
int time[SIZE] = {1, 2, 3, 4, 5},
speed[SIZE] = {18, 4, 27, 52, 100},
dist[SIZE];

for (int count = 0; count < SIZE; count++)
      dist[count] = time[count] * speed[count];
for (int count = 0; count < SIZE; count++) {
      cout << time[count] << " ";
      cout << speed[count] << " ";
      cout « dist[count] << endl;
}
```

In-Class Exercise

- Write a program that store the populations of 12 countries. Define 2 arrays that may be used in parallel to store the names of the countries and their populations. Write a loop that uses these arrays to print each country's name and its population.

ARRAY AND FUNCTION

Arrays as Function Arguments

- To pass an array to a function, just use the array name:

```
showScores(tests);
```

- To define a function that takes an array parameter, use empty `[]` for array argument:

```
void showScores(int []); // function prototype
void showScores(int tests[])// function header
```

Arrays as Function Arguments

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:

```
showScores(tests, ARRAY_SIZE);
```

- Array size must also be reflected in prototype, header:

```
void showScores(int [], int);
                // function prototype
void showScores(int tests[], int
  size)
                // function header
```

Arrays as Function Arguments - example

Program 7-14

```
1  // 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
```

(Program Continues)

Arrays as Function Arguments - example

Program 7-14 *(Continued)*

```
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  }
```

Program Output

```
5 10 15 20 25 30 35 40
```

Modifying Arrays in Functions

- Array names in functions are like reference variables – changes made to array in a function are reflected in actual array in calling function
- Need to exercise caution that array is not inadvertently changed by a function

In-Class Exercise

- The following program skeleton, when completed, will ask the user to enter 10 integers which are stored in an array. The function `avgArray`, which you must write, is to calculate and return the average of the numbers entered.

```
#include <iostream>
//Write your function prototype here
int main() {
        const int SIZE = 10;
        int userNums[SIZE];
        cout << "Enter 10 numbers: ";
        for (int count = 0; count < SIZE; count++){
                cout << "#" « (count + 1) << " ";
                cin >> userNums[count];
        }
        cout <<  "The average of those numbers is ";
        cout << avgArray(userNUms, SIZE) << endl;
        return 0;
}
//Write the function avgArray here.
```

In-Class Exercise

```
#include <iostream>
using namespace std;
void Test(int []);
int main()
{
int myArr [4]={3,4,5,6};
   for(int i=0;i<5;i++)
 cout<<myArr[i]<<" ";
   cout<<endl;
   Test(myArr);
   cout<<endl;
 for(int i=0;i<4;i++)
   cout<<myArr[i]<<" ";
      system("pause");
      return 0;}
```

```
void Test(int z[])
{
  int temp=z[3];
   z[3]=z[0];
   z[0]=temp;

   for(int
  j=0;j<4;j++)
      cout<<z[j]<<"
  ";
}
```

In-Class Exercise

```
#include <iostream>
using namespace std;
void Test(int , int,int[]);
int main()
{  int x = 1;
   int y[3];
   y[0]=1;
   Test(x,y[0],y);
   cout<<"x is: " << x<<
   endl;
   cout<<"y[0] is: " <<y[0]
   << endl;
   for(int i=0;i<3;i++)
          cout<<y[i]<<endl;
      system("pause");
      return 0;}
```

```
void Test(int num, int num1,
   int z[])
{
   num=1001;
   num1=290;
   z[1]=34;
   z[2]=35;
}
```

In-Class Exercise

- Each of the following definitions and program segments has errors. Locate as many as you can and correct the errors.

a)
```
void showValues(int nums)
   {
        for(int i = 0; i<8; i++)
                cout<<nums[i];
   }
```

b)
```
void showValues(int nums [4])
{
    for(int i = 0; i<8; i++)
            cout<<nums[i];
}
```

In-Class Exercise

- Consider the following function prototypes:

```
void funcOne(int [], int);
int findSum(int, int);
```

And the declarations:

```
int list[50];
int num;
```

Write a C++ statements that:

a) Call the function `funcOne` with the actual parameters, `list` and 50 respectively.

b) Print the value returned by the function `funcSum` with the actual parameters, 50, and the fourth element of `list` respectively.

c) Print the value returned by the function `funcSum` with the actual parameters, the thirtieth and tenth elements of `list`, respectively.

In-Class Exercise

- Write a program that has two overloaded functions that return the average of an array with the following headers:

```
int average(int array[], int size)
double average(int array[], int size
```

Use `{1,2,3,4,5,6}` and `{6.0,4.4,1.9,2.9,3.4,3.5}` to test the functions.

In-Class Exercise

- Write a program that has a function that returns the index of the smallest element in an array of integers. If there are more than one such elements, return the smallest index. Use `{1,2,4,5,10,100,2,-22}` to test the function.