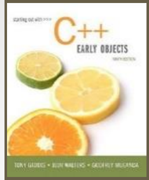
COP2334

Introduction to Object Oriented Programming with C++

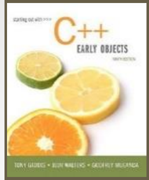
D. Singletary

<u>Module 6</u>

Ch. 8 Arrays



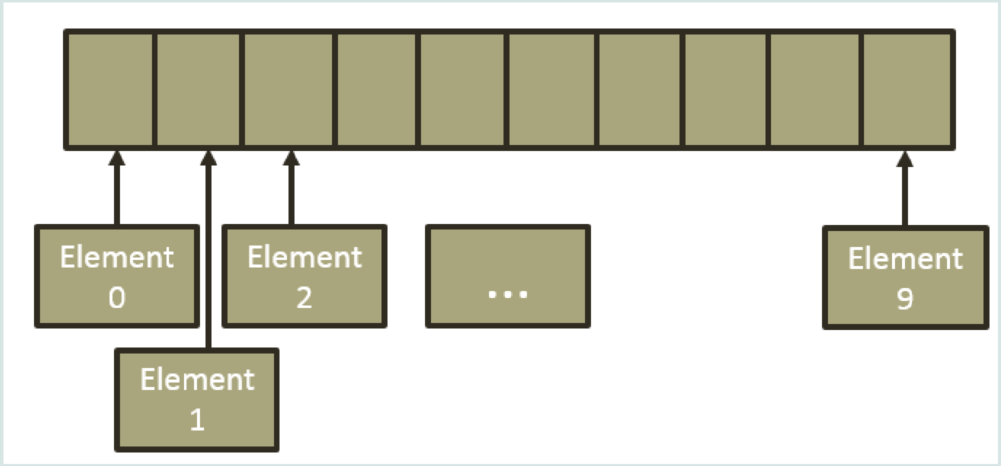
Ch. 8 Arrays

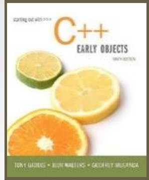


- An array stores multiple values of the same data type

int iArray[10];

- declares an integer array of length 10 named iArray

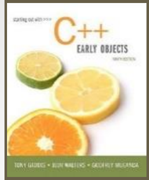




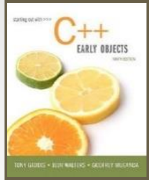
```
// arraydemo.cpp
#include <iostream>
using namespace std;

int main()
{
   const int NUM_INTS = 10;
   int iArray[NUM_INTS];

   iArray[0] = 0;
   iArray[1] = 10;
   iArray[2] = 20;
   iArray[3] = 30;
   iArray[4] = 40;
   iArray[5] = 50;
   iArray[6] = 60;
   iArray[7] = 70;
   iArray[8] = 80;
   iArray[9] = 90;

   for (int i = 0; i < NUM_INTS; i++)
      cout << "iArray[" << i << "] = " << iArray[i] << endl;

   return 0;
}
```

```
iArray[0] = 0
iArray[1] = 10
iArray[2] = 20
iArray[3] = 30
iArray[4] = 40
iArray[5] = 50
iArray[6] = 60
iArray[7] = 70
iArray[8] = 80
iArray[9] = 90
```

```
// arraydemo.cpp
#include <iostream>
using namespace std;

int main()
{
    const int NUM_INTS = 10;
    int iArray[NUM_INTS];

    // shortcut initialization – use an initializer list
    int iArray[NUM_INTS] = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 };

    for (int i = 0; i < NUM_INTS; i++)
        cout << "iArray[" << i << "] = " << iArray[i] << endl;

    return 0;
}
```

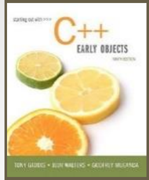


```
// arraydemo.cpp
#include <iostream>
using namespace std;

int main()
{
    const int NUM_INTS = 10;
    int iArray[NUM_INTS];

    // if we use an initializer list, we don't have to provide
    // a size declaratory, the compiler will count our items for us.
    int iArray[] = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 };

    for (int i = 0; i < NUM_INTS; i++)
        cout << "iArray[" << i << "] = " << iArray[i] << endl;

    return 0;
}
```

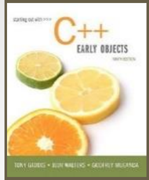


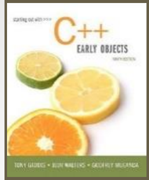
```
// montharray.cpp
// displays number of days in each month
#include <iostream>
#include <iomanip>
using namespace std;

enum MONTHS { MON_JAN, MON_FEB, MON_MAR, MON_APR, MON_MAY, MON_JUN,
                   MON_JUL, MON_AUG, MON_SEP, MON_OCT, MON_NOV, MON_DEC };

int main()
{
   const int NUM_MONTHS = 12;
   int days[NUM_MONTHS];

   days[MON_JAN] =  31;     // January
   days[MON_FEB] =  28;     // February
   days[MON_MAR] =  31;     // March
   days[MON_APR] =  30;     // April
   days[MON_MAY] =  31;     // May
   days[MON_JUN] =  30;     // June
   days[MON_JUL] =  31;     // July
   days[MON_AUG] =  31;     // August
   days[MON_SEP] =  30;     // September
   days[MON_OCT] =  31;     // October
   days[MON_NOV] = 30;      // November
   days[MON_DEC ] = 31;     // December

   for (int month = 0; month < NUM_MONTHS; month++)
      cout << "Month " << setw(2) << (month+1) << " has " <<
            days[month] << " days." << endl;

   return 0;
}
```

```
Month  1 has 31 days.
Month  2 has 28 days.
Month  3 has 31 days.
Month  4 has 30 days.
Month  5 has 31 days.
Month  6 has 30 days.
Month  7 has 31 days.
Month  8 has 31 days.
Month  9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```



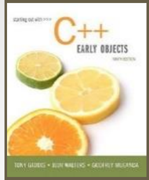
```
// calcpay.cpp
// use an array to store hours worked for employees
// and calculate gross/total pay
#include <iostream>
#include <iomanip>
using namespace std;

const int NUM_WORKERS = 5; // number of employees
const string STR_HOURSPROMPT = "Enter the hours worked by ";
const string STR_RATEPROMPT = "Enter hourly pay rate for everyone: $";
const string STR_EMPNUM = "Employee #";
const string STR_GROSS = "Gross pay";
const string STR_TOTALPAY = "Total pay";
const string STR_COLSPACE = ": ";
const string STR_COLSPACEDOLLAR = ": $";

int main() {
```

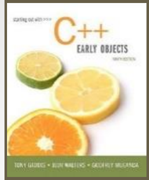




```
int hours[NUM_WORKERS];     // array to store hours
double payRate = 0.0;       // hourly pay rate
double grossPay = 0.0;      // employee's gross pay
double totalPay = 0.0;

// input hours worked by each
cout << STR_HOURSPROMPT << endl;
for (int worker = 0; worker < NUM_WORKERS; worker++)
{
        cout << STR_EMPNUM << (worker + 1) << STR_COLSPACE;
        cin >> hours[worker];
}

// input hourly pay rate for all
cout << STR_RATEPROMPT;
cin >> payRate;
```

```
    // display each employee's gross payRate
    cout << STR_GROSS << endl;
    cout << fixed << showpoint << setprecision(2);

    for (int worker = 0; worker < NUM_WORKERS; worker++)
    {
        grossPay = hours[worker] * payRate;
        totalPay += grossPay;
        cout << STR_EMPNUM << (worker + 1) <<
            STR_COLSPACEDOLLAR <<
            setw(7) << grossPay << endl;
    }

    cout << STR_TOTALPAY << STR_COLSPACEDOLLAR << totalPay << endl;

    return 0;
}
```



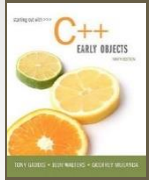


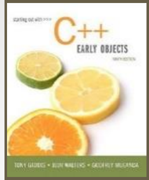
```
Enter the hours worked by
Employee #1: 30
Employee #2: 40
Employee #3: 50
Employee #4: 45
Employee #5: 35
Enter hourly pay rate for everyone: $15.55
Gross pay:
Employee #1: $ 466.50
Employee #2: $ 622.00
Employee #3: $ 777.50
Employee #4: $ 699.75
Employee #5: $ 544.25
Total pay: $3110.00
```

Parallel Arrays

- Two arrays of the same size which can use the same subscript for processing are <u>parallel</u> arrays

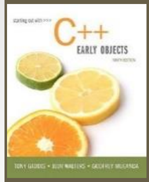
```
string monthName[NUM_MONTHS] = {
        "January",    "February", "March",      "April",
        "May",        "June",      "July",       "August",
        "September", "October",  "November", "December" };

    int monthDays[NUM_MONTHS] = {
                31, 28, 31, 30,
                31, 30, 31, 31,
                30, 31, 30, 31 };
```

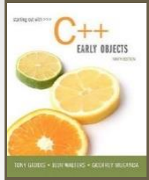




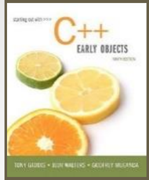
```
// monthday.cpp
// Allows the user to select a month and
// displays how many days are in that month.
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

const string STR_PROG_DESCRIP = "This program will tell you how many
days are in any month.\n\n";
const string STR_SPACER = "  ";
const string STR_PROMPT = "\nEnter the number of the month you want: ";
const string STR_MONTHOF = "The month of ";
const string STR_HAS = " has ";
const string STR_DAYS = " days\n";

int main() {
```

```
const int NUM_MONTHS = 12;
string monthName[NUM_MONTHS] = {
    "January",    "February", "March",    "April",
    "May",        "June",     "July",     "August",
    "September",  "October",  "November", "December" };

int monthDays[NUM_MONTHS] = {
                    31, 28, 31, 30,
                    31, 30, 31, 31,
                    30, 31, 30, 31 };

cout << STR_PROG_DESCRIP;
```

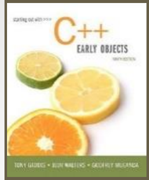




```
        // Display the months
        for (int month = 1; month <= NUM_MONTHS; month++)
                cout << setw(2) << month << STR_SPACER <<
                    monthName[month-1] << endl;

        cout << STR_PROMPT;

        int choice = 0;
        cin  >> choice;

        // Use the choice the user entered to get the name of
        // the month and its number of days from the arrays.
        cout << STR_MONTHOF << monthName[choice-1] << STR_HAS
                << monthDays[choice-1]  << STR_DAYS;

        return 0;
}
```



```
This program will tell you how many days are
in any month.

 1  January
 2  February
 3  March
 4  April
 5  May
 6  June
 7  July
 8  August
 9  September
10  October
11  November
12  December

Enter the number of the month you want: 9
The month of September has 30 days
```



- Strings in C++ can be treated as arrays

```
// stringsAsArray.cpp
// demonstrates using strings as arrays

#include <iostream>
using namespace std;

int main()
{
   string hStr = "Hello";

   cout << hStr << endl;
   cout << hStr[0] << hStr[1] << hStr[2] <<
              hStr[3] << hStr[4] << endl;

   return 0;
}
```

```
Hello
Hello
```

- Arrays can be initialized at declaration using braces:

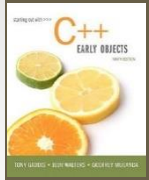
```
const int SIZE = 6;
int arrayA[SIZE] = { 10, 20, 30, 40, 50, 60 };
int arrayB[SIZE] = { 2, 4, 6, 8, 10, 12 };
```

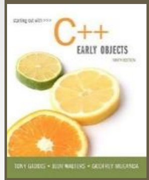
- Arrays cannot be copied by simple assignment:

```
arrayA = arrayB;
```

  - Instead, a for loop must be used:

```
for (int index = 0; index < SIZE; index++)
    arrayA[index] = arrayB[index];
```



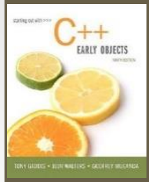


• Passing Arrays to Functions

```
// arraysToFunctions.cpp
// demonstrates passing arrays to functions

#include <iostream>
using namespace std;

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

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

   cout << "The array contains the values" << endl;
   showValues(collection, ARRAY_SIZE);

   return 0;
}
```

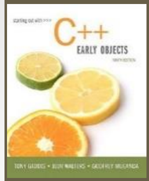
```
// displays contents of integer array
void showValues(int nums[], int size)
{
   for (int index = 0; index < size; index++)
      cout << nums[index] << " ";
   cout << endl;
}
```

```
The array contains the values
5 10 15 20 25 30 35 40
```



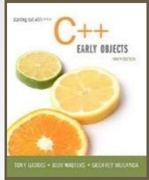
- Although arrays are not explicitly declared as pass-by-reference, they are treated as such

```
void showValues(int intArray[], int size);
void incrValues(int intArray[], int size);

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

   cout << "The array contains the values" << endl;
   showValues(collection, ARRAY_SIZE);

   incrValues(collection, ARRAY_SIZE);

   cout << "After increment, the array contains the values" << endl;
   showValues(collection, ARRAY_SIZE);

   return 0;
}
```



```
// displays contents of integer array
void showValues(int nums[], int size)
{
   for (int index = 0; index < size; index++)
      cout << nums[index] << " ";
   cout << endl;
}

// increments contents of integer array
void incrValues(int nums[], int size)
{
   for (int index = 0; index < size; index++)
      nums[index]++;
}
```

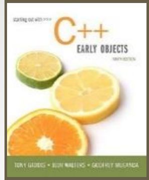
```
The array contains the values
5 10 15 20 25 30 35 40
After increment, the array contains the values
6 11 16 21 26 31 36 41
```

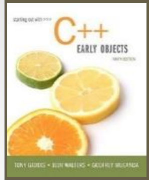


Range-Based For Loops

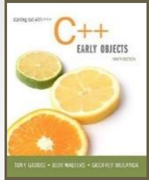
- C++ 11 introduced range-based for loops
  - same concept as "for-each" loops in Java
- Each time the loop iterates, an element from the array is copied to a variable

```
int numbers[] = { 3, 6, 9};
for (int val : numbers)
{
        cout << "The next value is ";
        cout << val << endl;
}
```



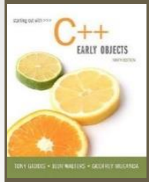


```
// rbf.cpp
// range-based for loop examples

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

int main()
{
   int numbers[] = { 3, 6, 9};
   for (int val : numbers)
   {
      cout << "The next value is ";
      cout << val << endl;
   }
   cout << endl;
```



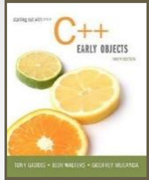
```
double values[] = { 1.5, 2.5, 30.5, 90.75, 212.0 };
    for (double dval : values)
    {
        cout << "The next value is ";
        cout << dval << endl;
    }
    cout << endl;

    string strings[] = { "Hello, ", "World ", "\n", "How ", "are ", "you?" };
    for (string s : strings)
    {
        cout << s;
    }
    cout << endl;

    return 0;
}
```

Limitations of RBF Loops



- No subscript/index (if you're counting)
- Won't work with parallel arrays
- Cannot modify array elements without a reference declaration

```
int numbers[] = { 3, 6, 9};
for (int val : numbers)
        val++;  // this only modifies the copy
```



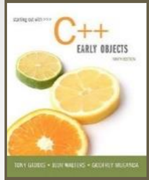
```
int numbers[] = { 3, 6, 9};
for (int val : numbers)
{
   cout << "The next value is ";
   cout << val << endl;
}
cout << endl;

for (int val : numbers)
   val++; // this only modifies the copy

for (int val : numbers)
{
   cout << "The next value is ";
   cout << val << endl;
}
cout << endl;
```

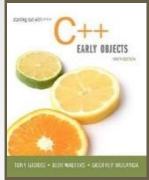
```
The next value is 3
The next value is 6
The next value is 9

The next value is 3
The next value is 6
The next value is 9
```

- Declare the range variable as a reference variable in order to modify the array

```
for (int &val : numbers)
    val++;
```

```
The next value is 3
The next value is 6
The next value is 9

The next value is 4
The next value is 7
The next value is 10
```





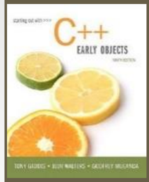
Two-Dimensional Arrays

- A two-dimensional array holds multiple sets of data
  - Usually thought of as a table with rows and columns as elements

```
double score[2][3];  // [rows][columns]
```

- Individual elements are referenced in a similar fashion as single dimensional arrays:

```
score[0][0]
score[0][1]
score[0][2]
score[1][0]
score[1][1]
score[1][2]
```



- 2D arrays can be initialized when declared:

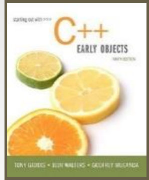
```
int hours[3][2] = { { 8, 5 }, { 7, 9 }, { 6, 3 } };
```

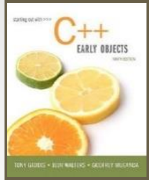
- 2D arrays can be passed to functions:
  - The number of columns (size of the second dimension) must be specified!

```
#define ROWS 2
#define COLS 3

int main() {
	int table1[ROWS][COLS] = { { 1, 2, 3}, { 4, 6, 8 } };
	showArray(table1, TBL1_ROWS);
	return 0;
}

void showArray(int const array[][COLS], int numRows)
{ …
```

- Nested loops are typically used to navigate a 2D array
- We can use const to prevent an array from being modified by a function

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





We can use range-based for loops to process 2D arrays, but it’s a bit tricky.

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

int main()
{
    const int SIZE = 3;
    int arr[SIZE][SIZE] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

    // display the array contents, with a newline after each row
    for (auto &row : arr)
    {
        for (auto col : row)
            cout << col << " ";
        cout << endl;
    }
    return 0;
}
```

To process a 2-dimensional array, the first dimension must be declared as a reference, otherwise there is an implicit conversion to a pointer that occurs that breaks the code. Note also the use of auto; C++ prefers this to int for declaring the reference type as the local variable.

In this example we also want to modify the array, so in the first loop we also declare the second variable (inner loop) as a reference so we can modify the element.

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

int main()
{
    const int SIZE = 3;
    int arr[SIZE][SIZE] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

    int count = 0 ;

    // set each array element to the incremented value of count * 100
    for(auto &row : arr)
        for(auto &col : row)
            col = ++count * 100;

    // display the array contents, with a newline after each row
    for (auto &row : arr)
    {
        for (auto col : row)
            cout << col << " ";
        cout << endl;
    }

    return 0;
}
```