

Laboratory 09B

CS-102

Spring 2022

Laboratory 09B: Duplicate and Reverse an Array Using Pointers

- Write a program that accepts, as input, a file of int, (“Gradelist.txt”), and stores it as an array. Use 50 for the Size of the resulting array.
- Display the original array.
- Write a pointer function that accepts the int array, as well as the array’s size as arguments.
- The function should create a copy of the array, except that the element values should be in reverse order in the copy.
- The function should return a pointer to the **new** array.
- Finally, display the **new** array showing that it is indeed in reverse order.
- In each case, use Dynamic Memory Allocation to create your duplicate array which will be in reverse order.
- Note: You will likely find the functions in Programs 9-12, 9-13 and 9-16 helpful in the execution of this laboratory.

Laboratory 09B: Duplicate and Reverse an Array Using Pointers

- The program that calls the function that will reverse the direction of the array in the file, GradeList.txt, is supplied for you (see following pages).
- It will be your job to write the function reverseArray(), that will take the contents of array1, and reverse its direction.
- You will also need to write the function displayArray(), which will display the original array, array1, as well as the duplicate array, dup1, which contains the data of array1 in reversed order.

```
#include <iostream>
#include <fstream>
using namespace std;
// Function prototype
int *reverseArray(const int *, int);
void displayArray(const int[], int);
const int SIZE1 = 50;
int main()
{
    // Array of unsorted values
    int array1[SIZE1];
    int count = 0;
    int arraySize, maxIndex;
    ifstream inputFile;
    inputFile.open("GradeList.txt");
    while (count < SIZE1 && inputFile >> array1[count])
        count++;
    inputFile.close();
    // Define pointer for the reversed array.
    int *dup1 = nullptr;
```

Function main() of
the program,
ReverseArray.cpp,
that reverses the
array supplied by
the data in the file
gradelist.txt .

-

Part 1 of 2

```

// Reverse the array.
dup1 = reverseArray(array1, count);

// Display the original array.
cout << "Here is the original array contents:\n";
displayArray(array1, count);

// Display the new array.
cout << "\nHere is the reversed array:\n";
displayArray(dup1, count);

cin.ignore();
cin.get();

// Free the dynamically allocated memory and
// set the pointer to 0.
delete [] dup1;
dup1 = nullptr;
return 0;
}

```

Function main()
of the program
that reverses the
array supplied by
the data in the
file gradelist.txt .

-

Part 2 of 2

Laboratory 09B – Part 1: Duplicate and Reverse an Array Using Pointers and Array Notation

- There are four ways you will be writing the function that will reverse the direction of the array.
 1. Using Array notation in the for loop of the function.
 2. Using static Pointer notation in the for loop of the function.
 3. Using dynamic Pointer notation in the for loop of the function.
 4. In this case, the original data should be put into a Vector, then dynamic Pointer notation should be used to populate the **new** array you will be constructing.
- For part 1, use Array notation (like that shown in Program 9-16).
- Call your program: *YourName-Lab09B-1.cpp*

Laboratory 09B – Part 1: Duplicate and Reverse an Array Using Pointers and Array Notation

- When you have your array notation function solution working,:
 - If you are doing this Lab synchronously, call over the instructor so your work can be assessed.
 - If you are doing this Lab asynchronously, please submit your program to Canvas for assessment.

Laboratory 09B – Part 2: Duplicate and Reverse an Array Using Pointers and Static Pointer Notation

- Now do the function using static pointers.
 - The use of static pointers is demonstrated in Program 9-13.
 - If **int *numbers** is the pointer, pointing to the beginning of the **new** array, and **count** is the loop variable, then ***(numbers + count)** will point to the particular element of the array as designated by **count**.
- Call your program: *YourName-Lab09B-2.cpp*
- When you have the static pointer version running:
 - If you are doing this Lab synchronously, call over the instructor so your work can be assessed.
 - If you are doing this Lab asynchronously, please submit your program to Canvas for assessment.

Laboratory 09B – Part 3: Duplicate and Reverse an Array Using Pointers and Dynamic Pointer Notation

- Now do the function using dynamic pointers.
 - The use of dynamic pointers is demonstrated in Program 9-12.
 - If **double *arr** is the pointer to the array of sales, and **arr** is the pointer to a particular element number which is currently being addressed, then **arr++** will advance the pointer to the next element in the array.
 - Call your program: *YourName-Lab09B-3.cpp*
- When you have the dynamic pointer version running:
 - If you are doing this Lab synchronously, call over the instructor so your work can be assessed.
 - If you are doing this Lab asynchronously, please submit your program to Canvas for assessment.

Laboratory 09B – Part 4: Duplicate and Reverse an Array Using Dynamic Pointers Notation and Vectors

- Now the sequence housing the original data will be a Vector, and you are asked to use dynamic pointers to populate the **new** array which will hold the data in the reverse direction.
 - In doing so, you will be combining array notation, which is used by Vectors, with the dynamic pointer notation which we will use to populate the **new** array which will hold the data supplied by the Vector in reverse order.
 - This means that you will need to rewrite the function `main()` so that it can accommodate vectors
 - You may also need to write a separate `displayVector()` function to display the elements of the original vector, which would slightly differ from `displayArray()`.
- Call your program: *YourName-Lab09B-4.cpp*

Laboratory 09B – Part 4: Duplicate and Reverse an Array Using Dynamic Pointers Notation and Vectors

- When you have the vector/dynamic pointer version running:
 - If you are doing this Lab synchronously, call over the instructor so your work can be assessed.
 - If you are doing this Lab asynchronously, please submit your program to Canvas for assessment.

Laboratory 09B – Part 5: Extra Credit

- Do this part only if you have the time.
 - Don't start this part if less than ½ hour of scheduled time remains for the class.
- In the program ReverseArray.cpp, that you are given, you were told you could assume that the file, GradeList.txt, had 50 integers.
- Rewrite ReverseArray.cpp, so that it cannot make this assumption. In other words, your program will have to determine the size that your array should be in order to accommodate the contents of the entire file, GradeList.txt, regardless of what size it might be.
- As you don't know, apriori, what the size of the file will be, you will need to make your initial array using Dynamic Memory Allocation also.
- Call your program: *YourName-Lab09B-5.cpp*

Laboratory 09B – Part 5: Extra Credit

- When you have your program working properly:
 - If you are doing this Lab synchronously, call over the instructor so your work can be assessed.
 - If you are doing this Lab asynchronously, please submit your program to Canvas for assessment.

Program 9-16: Duplicates Arrays

```
// This program uses a function to duplicate
// an int array of any size.
#include <iostream>
using namespace std;

// Function prototype
int *duplicateArray(const int *, int);
void displayArray(const int[], int);

int main()
{
    // Define constants for the array sizes.
    const int SIZE1 = 5, SIZE2 = 7, SIZE3 = 10;

    // Define three arrays of different sizes.
    int array1[SIZE1] = { 100, 200, 300, 400, 500 };
    int array2[SIZE2] = { 10, 20, 30, 40, 50, 60, 70 };
    int array3[SIZE3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    // Define three pointers for the duplicate arrays.
    int *dup1 = nullptr, *dup2 = nullptr, *dup3 = nullptr;
```

```
// Duplicate the arrays.
dup1 = duplicateArray(array1, SIZE1);
dup2 = duplicateArray(array2, SIZE2);
dup3 = duplicateArray(array3, SIZE3);
// Display the original arrays.
cout << "Here are the original array contents:\n";
displayArray(array1, SIZE1);
displayArray(array2, SIZE2);
displayArray(array3, SIZE3);
// Display the new arrays.
cout << "\nHere are the duplicate arrays:\n";
displayArray(dup1, SIZE1);
displayArray(dup2, SIZE2);
displayArray(dup3, SIZE3);
// Free the dynamically allocated memory and
// set the pointers to 0.
delete [] dup1;
delete [] dup2;
delete [] dup3;
dup1 = nullptr;
dup2 = nullptr;
dup3 = nullptr;
return 0;
}
```

Program 9-16 Cont.

```

//*****
// The duplicateArray function accepts an int array  *
// and an int that indicates the array's size. The  *
// function creates a new array that is a duplicate  *
// of the argument array and returns a pointer to the *
// new array. If an invalid size is passed the      *
// function returns null.                           *
//*****
int *duplicateArray(const int *arr, int size)
{
    int *newArray = nullptr;
    // Validate the size. If 0 or a negative
    // number was passed, return a null pointer.
    if (size <= 0)
        return nullptr;
    // Allocate a new array.
    newArray = new int[size];
    // Copy the array's contents to the
    // new array.
    for (int index = 0; index < size; index++)
        newArray[index] = arr[index];

    // Return a pointer to the new array.
    return newArray;
}

```

Program 9-16: duplicateArray Function

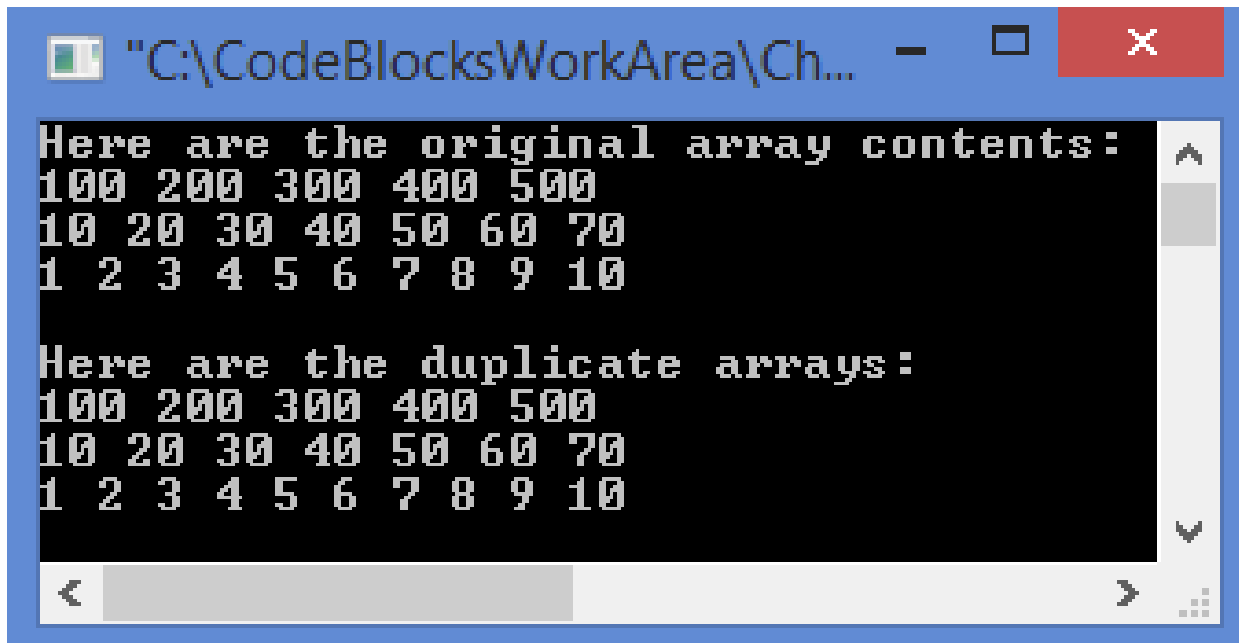
-

Note that in this function, that standard array notation is used to populate the newly created array, newArray.

Program 9-16: displayArray Function

```
/**  
 *  
 * The displayArray function accepts an int array *  
 * and its size as arguments and displays the *  
 * contents of the array. *  
 */
```

```
void displayArray(const int arr[], int size)  
{  
    for (int index = 0; index < size; index++)  
        cout << arr[index] << " ";  
    cout << endl;  
}
```



The screenshot shows a terminal window titled "C:\CodeBlocksWorkArea\Ch...". The output of the program is displayed in a monospaced font. It first shows the original array contents: 100 200 300 400 500, 10 20 30 40 50 60 70, and 1 2 3 4 5 6 7 8 9 10. Then it shows the duplicate arrays: 100 200 300 400 500, 10 20 30 40 50 60 70, and 1 2 3 4 5 6 7 8 9 10. The terminal has a blue title bar and a red close button.

```
"C:\CodeBlocksWorkArea\Ch...  
Here are the original array contents:  
100 200 300 400 500  
10 20 30 40 50 60 70  
1 2 3 4 5 6 7 8 9 10  
  
Here are the duplicate arrays:  
100 200 300 400 500  
10 20 30 40 50 60 70  
1 2 3 4 5 6 7 8 9 10
```

// This program demonstrates that a pointer may be used as a parameter to accept the address of an array.

```
#include <iostream>
```

```
#include <iomanip>
```

```
using namespace std;
```

```
// Function prototypes
```

```
void getSales(double *, int);
```

```
double totalSales(double *, int);
```

```
int main()
```

```
{
```

```
    const int QTRS = 4;
```

```
    double sales[QTRS];
```

```
    // Get the sales data for all quarters.
```

```
    getSales(sales, QTRS);
```

```
    // Set the numeric output formatting.
```

```
    cout << fixed << showpoint << setprecision(2);
```

```
    // Display the total sales for the year.
```

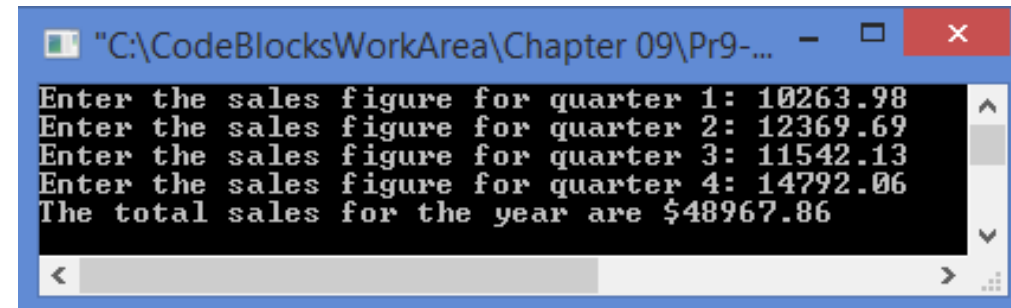
```
    cout << "The total sales for the year are $";
```

```
    cout << totalSales(sales, QTRS) << endl;
```

```
    return 0;
```

```
}
```

Program 9-12



The screenshot shows a Windows-style window titled "C:\CodeBlocksWorkArea\Chapter 09\Pr9-...". The window contains a black terminal area with white text. The text shows the program's execution: it prompts for sales figures for four quarters, receives inputs of 10263.98, 12369.69, 11542.13, and 14792.06, and then displays the total sales for the year as \$48967.86. The window has a blue title bar, a red close button, and a scrollbar on the right.

```
"C:\CodeBlocksWorkArea\Chapter 09\Pr9-... - [X]  
Enter the sales figure for quarter 1: 10263.98  
Enter the sales figure for quarter 2: 12369.69  
Enter the sales figure for quarter 3: 11542.13  
Enter the sales figure for quarter 4: 14792.06  
The total sales for the year are $48967.86
```

```
// Definition of getSales. This function uses a pointer to accept the address of an array of doubles.
```

```
// The function asks the user to enter sales figures and stores them in the array.
```

```
//*****
```

```
void getSales(double *arr, int size)
```

```
{  
    for (int count = 0; count < size; count++)
```

```
{
```

```
    cout << "Enter the sales figure for quarter ";
```

```
    cout << (count + 1) << ": ";
```

```
    cin >> arr[count];
```

```
}
```

```
}
```

```
//*****
```

```
// Definition of totalSales. This function uses a pointer to accept the address of an array.
```

```
// The function returns the total of the elements in the array.
```

```
double totalSales(double *arr, int size)
```

```
{
```

```
    double sum = 0.0;
```

```
    for (int count = 0; count < size; count++)
```

```
{
```

```
        sum += *arr;
```

```
        arr++;
```

```
}
```

```
    return sum;
```

```
}
```

Program 9-12

-

Note that in the function, totalSales(), that the pointer double *arr, marches through the array, dynamically, changing the element to which it points.

```
// This program demonstrates a pointer to const parameter
#include <iostream>
using namespace std;
void displayValues(const int *, int);
int main()
{
    // Array sizes
    const int SIZE = 6;
    // Define an array of const ints.
    const int array1[SIZE] = { 1, 2, 3, 4, 5, 6 };
    // Define an array of non-const ints.
    int array2[SIZE] = { 2, 4, 6, 8, 10, 12 };
    // Display the contents of the const array.
    displayValues(array1, SIZE);
    // Display the contents of the non-const array.
    displayValues(array2, SIZE);
    return 0;
}
```

Program 9-13
Using of
static
pointers to
process an
array.
Page 1 of 2

```

//*****
// The displayValues function uses a pointer to      *
// parameter to display the contents of an array.    *
//*****

void displayValues(const int *numbers, int size)
{
    // Display all the values.
    for (int count = 0; count < size; count++)
    {
        cout << *(numbers + count) << " ";
    }
    cout << endl;
}

```

Program 9-13

Using of static pointers to process an array.

Page 2 of 2

-

It is in this function that static pointers are used. Note that the pointer, numbers, never changes its value. It remains static. Only count changes.