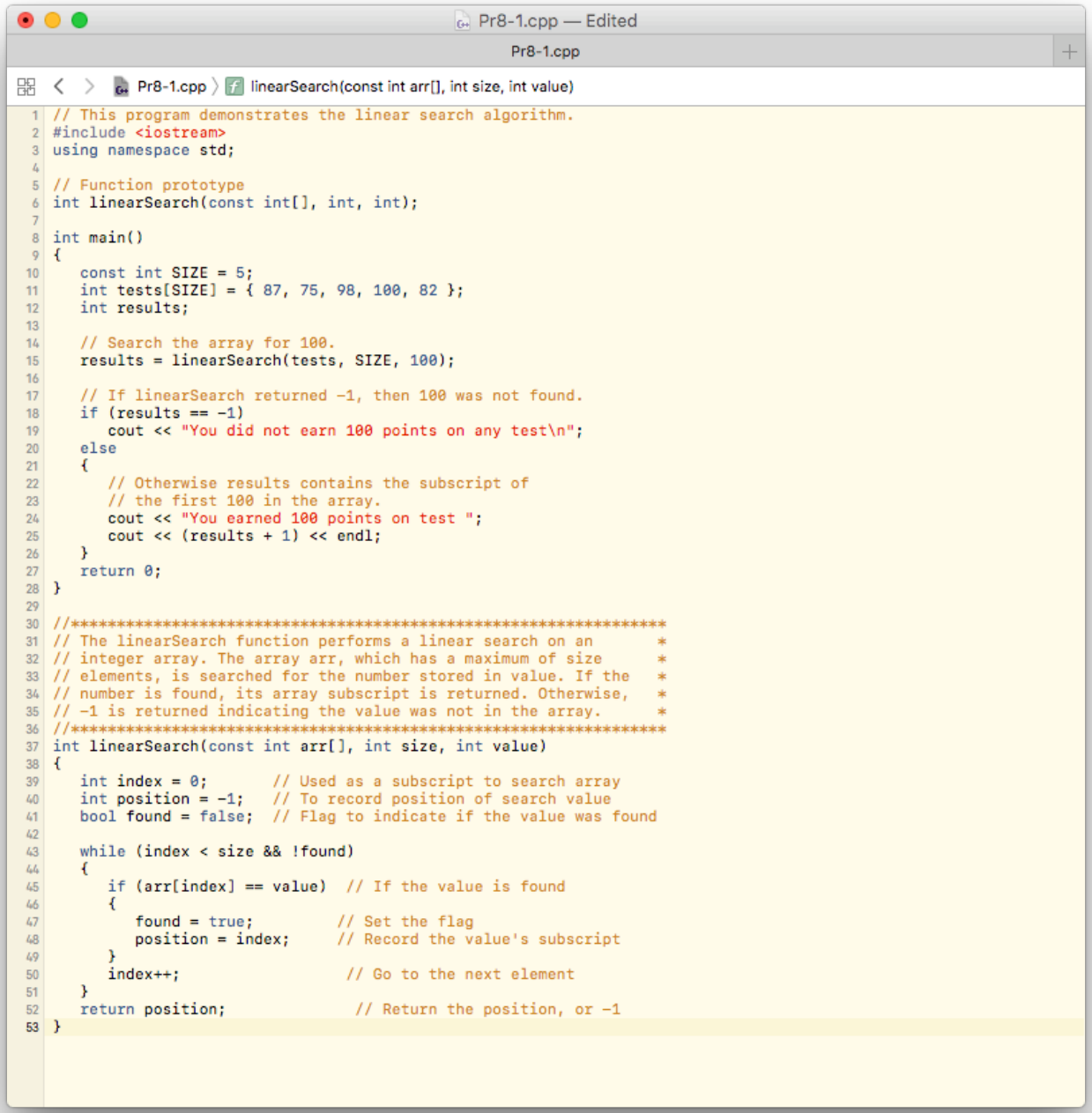



```cpp
// This program demonstrates the linear search algorithm.
#include <iostream>
using namespace std;

// Function prototype
int linearSearch(const int[], int, int);

int main()
{
   const int SIZE = 5;
   int tests[SIZE] = { 87, 75, 98, 100, 82 };
   int results;

   // Search the array for 100.
   results = linearSearch(tests, SIZE, 100);

   // If linearSearch returned -1, then 100 was not found.
   if (results == -1)
      cout << "You did not earn 100 points on any test\n";
   else
   {
      // Otherwise results contains the subscript of
      // the first 100 in the array.
      cout << "You earned 100 points on test ";
      cout << (results + 1) << endl;
   }
   return 0;
}

//*************************************************************
// The linearSearch function performs a linear search on an       *
// integer array. The array arr, which has a maximum of size      *
// elements, is searched for the number stored in value. If the   *
// number is found, its array subscript is returned. Otherwise,   *
// -1 is returned indicating the value was not in the array.      *
//*************************************************************
int linearSearch(const int arr[], int size, int value)
{
   int index = 0;       // Used as a subscript to search array
   int position = -1;   // To record position of search value
   bool found = false;  // Flag to indicate if the value was found

   while (index < size && !found)
   {
      if (arr[index] == value)  // If the value is found
      {
         found = true;          // Set the flag
         position = index;      // Record the value's subscript
      }
      index++;                  // Go to the next element
   }
   return position;             // Return the position, or -1
}
```

Pr8-2.cpp

Pr8-2.cpp

Pr8-2.cpp 〉 No Selection

```cpp
// This program demonstrates the binarySearch function, which
// performs a binary search on an integer array.
#include <iostream>
using namespace std;

// Function prototype
int binarySearch(const int [], int, int);
const int SIZE = 20;

int main()
{
   // Array with employee IDs sorted in ascending order.
   int idNums[SIZE] = {101, 142, 147, 189, 199, 207, 222,
                       234, 289, 296, 310, 319, 388, 394,
                       417, 429, 447, 521, 536, 600};
   int results;   // To hold the search results
   int empID;     // To hold an employee ID

   // Get an employee ID to search for.
   cout << "Enter the employee ID you wish to search for: ";
   cin >> empID;

   // Search for the ID.
   results = binarySearch(idNums, SIZE, empID);

   // If results contains -1 the ID was not found.
   if (results == -1)
      cout << "That number does not exist in the array.\n";
   else
   {
      // Otherwise results contains the subscript of
      // the specified employee ID in the array.
      cout << "That ID is found at element " << results;
      cout << " in the array.\n";
   }
   return 0;
}

//*************************************************************
// The binarySearch function performs a binary search on an     *
// integer array. array, which has a maximum of size elements,  *
// is searched for the number stored in value. If the number is *
// found, its array subscript is returned. Otherwise, -1 is     *
// returned indicating the value was not in the array.          *
//*************************************************************

int binarySearch(const int array[], int size, int value)
{
   int first = 0,             // First array element
       last = size - 1,       // Last array element
       middle,                // Mid point of search
       position = -1;         // Position of search value
   bool found = false;        // Flag

   while (!found && first <= last)
   {
      middle = (first + last) / 2;     // Calculate mid point
      if (array[middle] == value)      // If value is found at mid
      {
         found = true;
         position = middle;
      }
      else if (array[middle] > value)  // If value is in lower half
         last = middle - 1;
      else
         first = middle + 1;           // If value is in upper half
   }
   return position;
}
```

Pr8-3.cpp

Pr8-3.cpp

Pr8-3.cpp 〉 No Selection

```cpp
// Demetris Leadership Center (DLC) product lookup program
// This program allows the user to enter a product number
// and then displays the title, description, and price of
// that product.
#include <iostream>
#include <string>
using namespace std;

const int NUM_PRODS = 9;        // The number of products produced
const int MIN_PRODNUM = 914;    // The lowest product number
const int MAX_PRODNUM = 922;    // The highest product number

// Function prototypes
int getProdNum();
int binarySearch (const int [], int, int);
void displayProd(const string [], const string [], const double [], int);

int main()
{
    // Array of product IDs
    int id[NUM_PRODS] = {914, 915, 916, 917, 918, 919, 920,
                         921, 922};

    // Array of product titles
    string title[NUM_PRODS] =
            { "Six Steps to Leadership",
              "Six Steps to Leadership",
              "The Road to Excellence",
              "Seven Lessons of Quality",
              "Seven Lessons of Quality",
              "Seven Lessons of Quality",
              "Teams Are Made, Not Born",
              "Leadership for the Future",
              "Leadership for the Future"
            };

    // Array of product descriptions
    string description[NUM_PRODS] =
            { "Book", "Audio CD", "DVD",
              "Book", "Audio CD", "DVD",
              "Book", "Book", "Audio CD"
            };

    // Array of product prices
    double prices[NUM_PRODS] = {12.95, 14.95, 18.95, 16.95, 21.95,
                                31.95, 14.95, 14.95, 16.95};

    int prodNum;    // To hold a product number
    int index;      // To hold search results
    char again;     // To hold a Y or N answer

    do
    {
        // Get the desired product number.
        prodNum = getProdNum();

        // Search for the product number.
        index = binarySearch(id, NUM_PRODS, prodNum);

        // Display the results of the search.
        if (index == -1)
            cout << "That product number was not found.\n";
        else
            displayProd(title, description, prices, index);

        // Does the user want to do this again?
        cout << "Would you like to look up another product? (y/n) ";
        cin >> again;
    } while (again == 'y' || again == 'Y');
    return 0;
}
```

Pr8-3.cpp

Pr8-3.cpp

Pr8-3.cpp 〉 No Selection

```
72
73  //*************************************************
74  // Definition of getProdNum function              *
75  // The getProdNum function asks the user to enter a *
76  // product number. The input is validated, and when *
77  // a valid number is entered, it is returned.     *
78  //*************************************************
79
80  int getProdNum()
81  {
82      int prodNum; // Product number
83
84      cout << "Enter the item's product number: ";
85      cin >> prodNum;
86      // Validate input
87      while (prodNum < MIN_PRODNUM || prodNum > MAX_PRODNUM)
88      {
89          cout << "Enter a number in the range of " << MIN_PRODNUM;
90          cout <<" through " << MAX_PRODNUM << ".\n";
91          cin >> prodNum;
92      }
93      return prodNum;
94  }
95
96  //*************************************************************
97  // Definition of binarySearch function                        *
98  // The binarySearch function performs a binary search on an   *
99  // integer array. array, which has a maximum of numElems      *
100 // elements, is searched for the number stored in value. If the *
101 // number is found, its array subscript is returned. Otherwise, *
102 // -1 is returned indicating the value was not in the array.  *
103 //*************************************************************
104
105 int binarySearch(const int array[], int numElems, int value)
106 {
107     int first = 0,           // First array element
108         last = numElems - 1, // Last array element
109         middle,              // Midpoint of search
110         position = -1;       // Position of search value
111     bool found = false;      // Flag
112
113     while (!found && first <= last)
114     {
115         middle = (first + last) / 2;  // Calculate midpoint
116         if (array[middle] == value)   // If value is found at mid
117         {
118             found = true;
119             position = middle;
120         }
121         else if (array[middle] > value) // If value is in lower half
122             last = middle - 1;
123         else
124             first = middle + 1;         // If value is in upper half
125     }
126     return position;
127 }
128
129 //*************************************************************
130 // The displayProd function accepts three arrays and an int. *
131 // The arrays parameters are expected to hold the title,     *
132 // description, and prices arrays defined in main. The index *
133 // parameter holds a subscript. This function displays the   *
134 // information in each array contained at the subscript.     *
135 //*************************************************************
136
137 void displayProd(const string title[], const string desc[],
138                  const double price[], int index)
139 {
140     cout << "Title: " << title[index] << endl;
141     cout << "Description: " << desc[index] << endl;
142     cout << "Price: $" << price[index] << endl;
143 }
```

Pr8-4.cpp

Pr8-4.cpp

Pr8-4.cpp 〉 No Selection

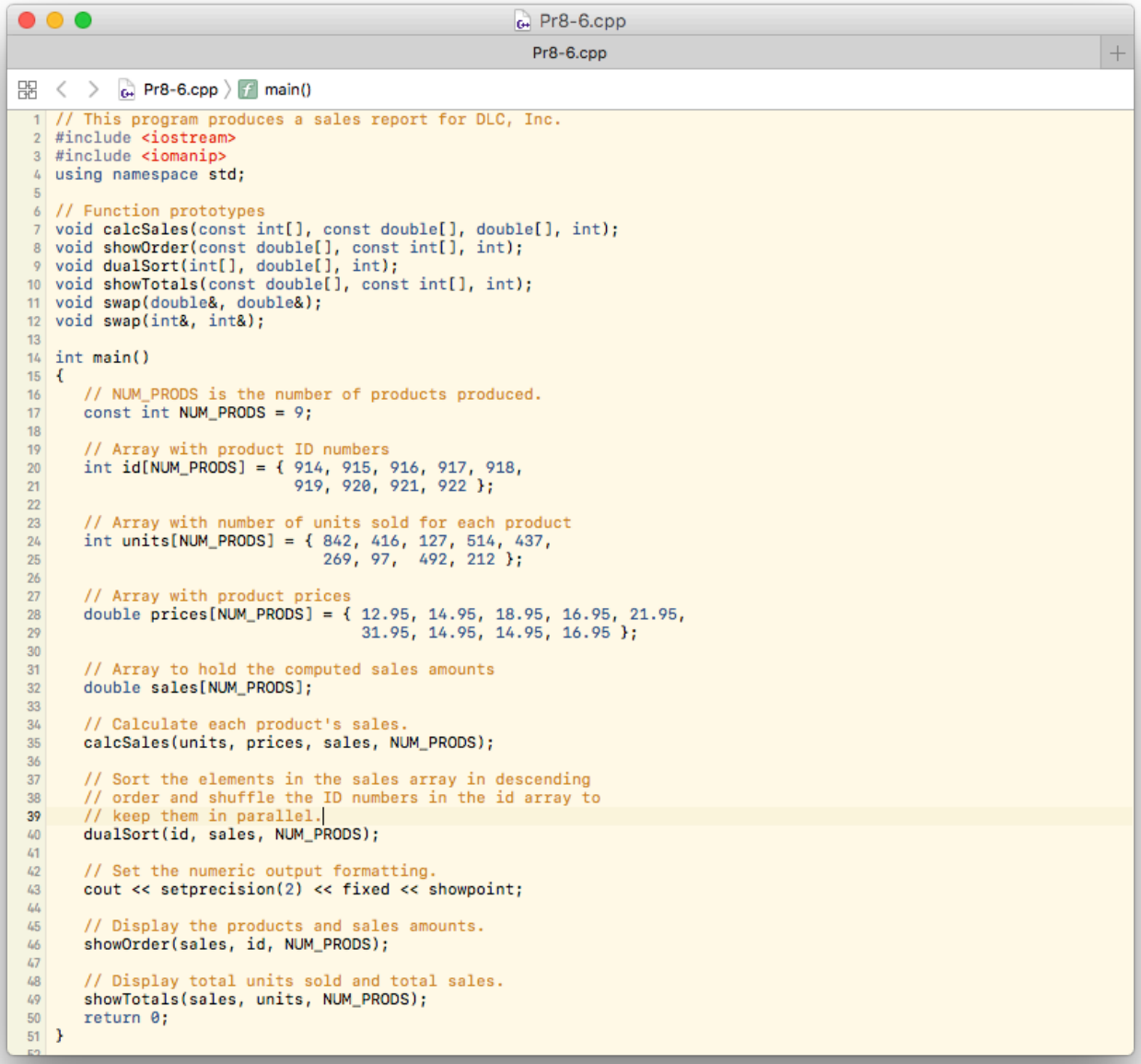
```cpp
// This program demonstrates the Bubble Sort algorithm.
#include <iostream>
using namespace std;

// Function prototypes
void bubbleSort(int[], int);
void swap(int &, int &);

int main()
{
   const int SIZE = 6;

   // Array of unsorted values
   int values[SIZE] = { 6, 1, 5, 2, 4, 3 };

   // Display the unsorted array.
   cout << "The unsorted values:\n";
   for (auto element : values)
      cout << element << " ";
   cout << endl;

   // Sort the array.
   bubbleSort(values, SIZE);

   // Display the sorted array.
   cout << "The sorted values:\n";
   for (auto element : values)
      cout << element << " ";
   cout << endl;

   return 0;
}

//*************************************************************
// The bubbleSort function sorts an int array in ascending order. *
//*************************************************************
void bubbleSort(int array[], int size)
{
   int maxElement;
   int index;

   for (maxElement = size - 1; maxElement > 0; maxElement--)
   {
      for (index = 0; index < maxElement; index++)
      {
         if (array[index] > array[index + 1])
         {
            swap(array[index], array[index + 1]);
         }
      }
   }
}

//*****************************************************
// The swap function swaps a and b in memory.       *
//*****************************************************
void swap(int &a, int &b)
{
   int temp = a;
   a = b;
   b = temp;
}
```

Pr8-5.cpp

Pr8-5.cpp

Pr8-5.cpp 〉 No Selection

```cpp
// This program demonstrates the Selection Sort algorithm.
#include <iostream>
using namespace std;

// Function prototypes
void selectionSort(int[], int);
void swap(int &, int &);

int main()
{
   const int SIZE = 6;

   // Array of unsorted values
   int values[SIZE] = { 6, 1, 5, 2, 4, 3 };

   // Display the unsorted array.
   cout << "The unsorted values:\n";
   for (auto element : values)
      cout << element << " ";
   cout << endl;

   // Sort the array.
   selectionSort(values, SIZE);

   // Display the sorted array.
   cout << "The sorted values:\n";
   for (auto element : values)
      cout << element << " ";
   cout << endl;

   return 0;
}

//*************************************************************
// The selectionSort function sorts an int array in ascending order. *
//*************************************************************
void selectionSort(int array[], int size)
{
   int minIndex, minValue;

   for (int start = 0; start < (size - 1); start++)
   {
      minIndex = start;
      minValue = array[start];
      for (int index = start + 1; index < size; index++)
      {
         if (array[index] < minValue)
         {
            minValue = array[index];
            minIndex = index;
         }
      }
      swap(array[minIndex], array[start]);
   }
}

//*************************************************
// The swap function swaps a and b in memory.       *
//*************************************************
void swap(int &a, int &b)
{
   int temp = a;
   a = b;
   b = temp;
}
```


Pr8-6.cpp
Pr8-6.cpp
Pr8-6.cpp
main()
// This program produces a sales report for DLC, Inc.
#include <iostream>
#include <iomanip>
using namespace std;
// Function prototypes
void calcSales(const int[], const double[], double[], int);
void showOrder(const double[], const int[], int);
void dualSort(int[], double[], int);
void showTotals(const double[], const int[], int);
void swap(double&, double&);
void swap(int&, int&);
int main()
{
// NUM_PRODS is the number of products produced.
const int NUM_PRODS = 9;
// Array with product ID numbers
int id[NUM_PRODS] = { 914, 915, 916, 917, 918,
919, 920, 921, 922 };
// Array with number of units sold for each product
int units[NUM_PRODS] = { 842, 416, 127, 514, 437,
269, 97, 492, 212 };
// Array with product prices
double prices[NUM_PRODS] = { 12.95, 14.95, 18.95, 16.95, 21.95,
31.95, 14.95, 14.95, 16.95 };
// Array to hold the computed sales amounts
double sales[NUM_PRODS];
// Calculate each product's sales.
calcSales(units, prices, sales, NUM_PRODS);
// Sort the elements in the sales array in descending
// order and shuffle the ID numbers in the id array to
// keep them in parallel.
dualSort(id, sales, NUM_PRODS);
// Set the numeric output formatting.
cout << setprecision(2) << fixed << showpoint;
// Display the products and sales amounts.
showOrder(sales, id, NUM_PRODS);
// Display total units sold and total sales.
showTotals(sales, units, NUM_PRODS);
return 0;
}

Pr8-6.cpp — Edited

Pr8-6.cpp

Pr8-6.cpp 〉 showOrder(const double sales[], const int id[], int num)

```cpp
//***************************************************************
// Definition of calcSales. Accepts units, prices, and sales    *
// arrays as arguments. The size of these arrays is passed      *
// into the num parameter. This function calculates each        *
// product's sales by multiplying its units sold by each unit's *
// price. The result is stored in the sales array.              *
//***************************************************************

void calcSales(const int units[], const double prices[], double sales[], int num)
{
   for (int index = 0; index < num; index++)
      sales[index] = units[index] * prices[index];
}

//***************************************************************
// Definition of function dualSort. Accepts id and sales arrays *
// as arguments. The size of these arrays is passed into size.  *
// This function performs a descending order selection sort on  *
// the sales array. The elements of the id array are exchanged  *
// identically as those of the sales array. size is the number  *
// of elements in each array.                                   *
//***************************************************************

void dualSort(int id[], double sales[], int size)
{
   int start, maxIndex, tempid;
   double maxValue;

   for (start = 0; start < (size - 1); start++)
   {
      maxIndex = start;
      maxValue = sales[start];
      tempid = id[start];
      for (int index = start + 1; index < size; index++)
      {
         if (sales[index] > maxValue)
         {
            maxValue = sales[index];
            tempid = id[index];
            maxIndex = index;
         }
      }
      swap(sales[maxIndex], sales[start]);
      swap(id[maxIndex], id[start]);
   }
}

//*****************************************************
// The swap function swaps doubles a and b in memory. *
//*****************************************************
void swap(double &a, double &b)
{
   double temp = a;
   a = b;
   b = temp;
}

//*****************************************************
// The swap function swaps ints a and b in memory.    *
//*****************************************************
void swap(int &a, int &b)
{
   int temp = a;
   a = b;
   b = temp;
}

//***************************************************************
// Definition of showOrder function. Accepts sales and id arrays *
// as arguments. The size of these arrays is passed into num.   *
// The function first displays a heading, then the sorted list  *
// of product numbers and sales.                                *
//***************************************************************

void showOrder(const double sales[], const int id[], int num)
{
   cout << "Product Number\tSales\n";
   cout << "--------------------------------\n";
   for (int index = 0; index < num; index++)
   {
      cout << id[index] << "\t\t$";
      cout << setw(8) << sales[index] << endl;
   }
   cout << endl;
}

//***************************************************************
// Definition of showTotals function. Accepts sales and id arrays *
// as arguments. The size of these arrays is passed into num.   *
// The function first calculates the total units (of all        *
// products) sold and the total sales. It then displays these   *
// amounts.                                                     *
//***************************************************************

void showTotals(const double sales[], const int units[], int num)
{
   int totalUnits = 0;
   double totalSales = 0.0;

   for (int index = 0; index < num; index++)
   {
      totalUnits += units[index];
      totalSales += sales[index];
   }
   cout << "Total units Sold:  " << totalUnits << endl;
   cout << "Total sales:      $" << totalSales << endl;
}
```