9.1 The Array Data Type

- Array elements have a common name
  - The array as a whole is referenced through the common name
- Array elements are of the same type — the base type
- Individual elements of the array are referenced by sub scripting the group name

Arrays

- Analogies
  - Egg carton
  - Apartments
  - Cassette carrier
- More terminology
  - Ability to refer to a particular element
    - Indexing or sub scripting
  - Ability to look inside an element
    - Accessing value

Language restrictions

- Subscripts are denoted as expressions within brackets: [ ]
- Base type can be any fundamental, library-defined, or programmer-defined type

Array Declaration

```
BaseType Id [ SizeExp ] ;
```

- The index type is integer and the index range must be 0 ... n-1
  - where n is a programmer-defined constant expression.
- Parameter passing style
  - Always call by reference (no indication necessary)
Sample Declarations

* Suppose
  
  ```
  const int N = 20;
  const int M = 40;
  const int MaxStringSize = 80;
  const int MaxListSize = 1000;
  ```

Then the following are all correct array declarations.

```
int A[10];
char B[MaxStringSize];
float C[M*N];
int Values[MaxListSize];
Rational D[N-15];
```

Subscripting

* Suppose
  
  ```
  int A[10]; // array of 10 ints
  ```

* To access an individual element we must apply a subscript to array name A
  
  - A subscript is a bracketed expression
  - The expression in the brackets is known as the index
  - First element of A has index 0
  - `A[0]`

Incorrect indexing is a common error

Array Elements

* Suppose
  
  ```
  int A[10]; // array of 10 uninitialized ints
  ```

* To access an individual element we must apply a subscript to array name A

```
```

Array Element Manipulation

* Given the following:

```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[A[j]] = 12;
```
Array Element Manipulation

```cpp
cin >> A[k]; // where the next input value is 3
```

Inputting Into An Array

```cpp
int A[MaxListSize];
int n = 0;
int CurrentInput;
while((n < MaxListSize) && (cin >>
    CurrentInput))
{
    A[n] = CurrentInput;
    ++n;
}
```

Displaying An Array

```cpp
// List A of n elements has
// already been set
for (int i = 0; i < n; ++i)
{
    cout << A[i] << " ";
}
```

Remember

- Arrays are always passed by reference – Artifact of C
- Can use const if array elements are not to be modified
- You do not need to include the array size within the brackets when defining an array parameter
- Initialize array with 0 or some other known value

9.2 Sequential Access to Array Elements

- Random Access
  - Access elements is random order
- Sequential Access
  - Process elements in sequential order starting with the first
  - `ShowDiff.cpp` a program that looks at values and calculates a difference between the element and the average

ShowDiff.cpp

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    const int MAX_ITEMS = 8;
    float x[MAX_ITEMS];
    float average;
    float sum;
```
// Enter the data.
cout << "Enter " << MAX_ITEMS << " numbers: ";
for (int i = 0; i < MAX_ITEMS; i++)
    cin >> x[i];

// Compute the average value.
sum = 0.0;
for (int i = 0; i < MAX_ITEMS; i++)
    sum += x[i];
average = sum / MAX_ITEMS;

// Display the difference between each item
// and the average.
cout << "The average value is " << average << endl << endl;

cout << "Table of differences between x[i] and the average." << endl;
cout << setw (4) << "i" << setw (8) << "x[i]" << setw (14) << "difference" << endl;
for (int i = 0; i < MAX_ITEMS; i++)
    cout << setw (4) << i << setw (8) << x[i] << setw (14) << (x[i] - average) << endl;

return 0;
}

9.3 Array Arguments

- Use <, ==, >, +, - to test and modify array elements
- At times it might benefit you to pass an entire array to a function
- Can pass array elements to functions
  - actual function call
    exchange (s[3], s[5]);
- Examples follow
Arrays as Function Arguments

- Remember arrays are pass by reference
  - Passing the array address
- Remember these points when passing arrays to functions
  - The formal array argument in a function is not itself an array but rather is a name that represents an actual array argument. Therefore in the function definition, you need only inform the compiler with [[]] that the actual argument will be an array.

SameArray.cpp example

```cpp
bool sameArray (float a[], float b[], const int size) {
  int i;
  i = 0;
  while ((i < size-1) && (a[i] == b[i])) 
    i++;
  return (a[i] == b[i]);
}
```

AddArray.cpp

```cpp
void addArray (int size, const float a[], const float b[], float c[]) 
{ // Add corresponding elements of a and b and store in c:
  for (int i = 0; i < size; i++)
    c[i] = a[i] + b[i];
} // end addArray
```

9.4 Reading Part of an Array

- Sometimes it is difficult to know how many elements will be in an array
- Scores example
  - 150 students
  - 200 students
- Always allocate enough space at compile time
- Remember to start with index [0]
// File: ReadScoresFile.cpp
// Reads an array of exam scores for a lecture section of up to max_size students.

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

#define inFile "Scores.txt"

void readScoresFile (ifstream& ins, int scores[], const int MAX_SIZE, int& sectionSize)
{
    int tempScore;
    // Read each array element until done.
    sectionSize = 0;
    ins >> tempScore;
    while (!ins.eof() && (sectionSize < MAX_SIZE))
    {
        scores[sectionSize] = tempScore;
        sectionSize++;
        ins >> tempScore;
    }
    // End while
    if (!ins.eof())
    {
        cout << "Array is filled!" << endl;
        cout << tempScore << " not stored" << endl;
    }
}

int main()
{
    int scores[100];
    int size;
    ifstream ins;
    ins.open(inFile);
    if (ins.fail())
    {
        cout << "Error" << endl;
        return 1;
    }
    readScoresFile(ins, scores, 5, size);
    for (int i = 0; i < size; i++)
    {
        cout << scores[i] << "  " ;
    }
    cout << endl;
    return 0;
}
9.5 Searching and Sorting Arrays

- Look at 2 common array problems
  - Searching
  - Sorting
- How do we go about finding the smallest number in an array?
  - Assume 1st is smallest and save its position
  - Look for one smaller
  - If you locate one smaller save its position

ArrayOperations.cpp

```cpp
int findIndexOfMin(const float x[], int startIndex, int endIndex) {
    // Local data ...
    int minIndex;
    int i;
    // Assume the first element of subarray is smallest and check the rest.
    // minIndex will contain subscript of smallest examined so far.
    minIndex = startIndex;
    for (i = startIndex + 1; i <= endIndex; i++)
        if (x[i] < x[minIndex])
            minIndex = i;
    return minIndex;
}
```

Strings and Arrays of Characters

- String object uses an array whose elements are type `char`
- First position of a string object is 0
  - example string find function ret of position 0
- Can use the find function to locate or search an array
- We will study some various search functions
Linear Search

- The idea of a linear search is to walk through the entire until a target value is located
- If the target is not located some type of indicator needs to be returned

ArrayOperations.cpp

```
// Searches an integer array for a given element
// (the target)
// Array elements ranging from 0 to size - 1 are searched for an element equal to target.
// Pre: The target and array are defined.
// Post: Returns the subscript of target if found; otherwise, returns -1.

int linSearch (const int items[], int target, int size)
{
    for (int i = 0; i < size; i++)
    {
        if (items[next] == target)
            return next;
        // All elements were tested without success.
        return -1;
    } // end linSearch
```

ArrayOperations.cpp

```
// Sorts an array (ascending order) using selection sort algorithm
// Uses exchange and findIndexOfMin
// Sorts the data in array items [items[0] through items[n-1]).
// Pre: items is defined and n <= declared size of actual argument array.
// Post: The values in items[0] through items[n-1] are in increasing order.

void selSort(int items[], int n)
{
    // Local data ...
    int minSub;
    for (int i = 0; i < n-1; i++)
    {
        // Find index of smallest element in unsorted section of items.
        minSub = findIndexOfMin(items, i, n-1);
    }
}
```
ArrayOperations.cpp

```c++
// Exchange items at position minSub and i
exchange(items[minSub], items[i]);
```

9.7 Analyzing Algorithms

**Big O Notation**

- How to compare efficiency of various algorithms
- A mathematical measuring stick to do quantitative analysis on algorithms
- Typically sorting and searching
- Based on looping constructs and placed into categories based on their efficiency
- Most algorithms have BigO published

---

**Analyzing Algorithms**

**Big O Notation**

- Run time efficiency is in direct proportion to the number of elementary machine operations
  - Compares
  - Exchanges

---

**Analyzing Algorithms**

**Big O Notation**

- Two independent loops
  - Sum of the loops is efficiency
  - $n/2 + n^2$ is Big O($N^2$)
  
  **Example:**
  ```c++
  for (k=1; k<=n/2; ++k)
  {
  }
  for (j=1; j<=n*n; ++j)
  {
  }
  ```

---

**Analyzing Algorithms**

**Big O Notation**

- Two nested loops
  - Product of the loops is efficiency
  - $n/2 * n^2 = n^3/2$ is Big O($N^3$)
  
  **Example:**
  ```c++
  for (k=1; k<=n/2; ++k)
  {
    for (j=1; j<=n*n; ++j)
    {
    }
  }
  ```

---

**9.7 The Struct Data Type**

- struct used to store related data items
- Individual components of the struct are called its members
- Each member can contain different types of data
- Employee example
// Definition of struct employee
struct employee
{
    string id;
    string name;
    char gender;
    int numDepend;
    money rate;
    money totWages;
};

// Members are accessed using the member access operator, a period
// For struct variable s and member variable m to access m you would use the following:
    cout << s.m << endl;
// Can use all C++ operators and operations on structs

organist.id = 1234;
organist.name = “Noel Goddard”;
organist.gender = ‘F’;
organist.numDepend = 0;
organist.rate = 6.00;
organist.totWages += organist.rate * 40.0;

// How to do arithmetic and other operations using structs
// Process entire struct using programmer defined functions
// Often better to pass an entire structure rather than individual elements
// struct copies
organist = janitor;

// Grading program example
// Keep track of students grades
// Prior to our learning structs we needed to store each item into a single variable
// Group all related student items together
// Pass struct by const reference if you do not want changes made

// FILE: ExamStat.h
struct examStats
{
    string stuName;
    int scores[3];
    float average;
    char grade;
};
PrintStats.cpp

void printStats(examStats stuExams)
{
    cout << "Exam scores for " << stuExams.stuName << " : 
    cout << "Average score: " << stuExams.average << endl;
    cout << "Letter grade : " << stuExams.grade << endl;
}

ReadEmp.cpp

#include <string>
#include <iostream>

void readEmployee(employee& oneEmployee)
{
    cout << "Enter a name terminated with the symbol # : ");
    getline(cin, oneEmployee.name, '#');
    cout << "Enter an id number: ");
    cin >> oneEmployee.id;
    cout << "Enter gender (F or M): ");
    cin >> oneEmployee.gender;
    cout << "Enter number of dependents: ");
    cin >> oneEmployee.numDepend;
    cout << "Enter hourly rate: ");
    cin >> oneEmployee.rate;
}

9.9 Common Programming Errors

- Watch non int subscripts (ASCII value)
- Enumerated types can be used
- Out of range errors
  - C++ no range error checking
- Lack of subscript to gain access
- Subscript reference to non-array variable
- Type mixing when using with functions
- Initialization of arrays