Data Structures
Arrays and Structs

Chapter 9

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

Arrays

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

Arrays

- 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)

Array Declaration

BaseTypeId [SizeExp] ;

Type of values in list

Name of list

Bracketed constant expression indicating number of elements in list

Sample Declarations

- Suppose

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

7

Sample Declarations

- 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];
```

8

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]

9

Subscripting

  - Second element of A has index 1, and so on

    A[1]
  - Last element has an index one less than the size of the array

    A[9]
- Incorrect indexing is a common error

10

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

--	--	--	--	--	--	--	--	--	--
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

11

Array Element Manipulation

- Given the following:

```
int i = 7, j = 2, k = 4;
A[0] = 1;
A[i] = 5;
A[j] = A[i] + 3;
A[j+1] = A[i] + A[0];
A[A[j]] = 12;
```

12

Array Element Manipulation

1	--	8	6	3	--	--	5	12	--
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

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

Inputting Into An Array

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


Displaying An Array

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


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

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
   const int MAX_ITEMS = 8;
   float x[MAX_ITEMS];
   float average;
   float sum;
```

ShowDiff.cpp

```
   // 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;
```


ShowDiff.cpp

```
   cout << "The average value is " <<
                  average << endl << endl;

   // Display the difference between each item
  // and the average.
   cout << "Table of differences between x[i]
            and the average." << endl;
   cout << setw (4) << "i" << setw (8) << "x[i]"
        << setw (14) << "difference" << endl;
```


ShowDiff.cpp

```
   for (int i = 0; i < MAX_ITEMS; i++)
      cout << setw (4) << i << setw (8) << x[i]
           << setw (14) << (x[i] - average) <<
                                   endl;


   return 0;
}
```


ShowDiff.cpp

Program Output

Enter 8 numbers: **16 12 6 8 2.5 12 14 -54.5**

The average value is 2.0

Table of differences between x[i] and the average

I	x[I]	difference
0	16.0	14.0
1	12.0	10.0
2	6.0	4.0
3	8.0	6.0

etc etc

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

Exchange.cpp

```
// FILE: Exchange.cpp
// Exchanges two type float values

void exchange (float& a1, float& a2)
{
   float temp;

   temp = a1;
   a1 = a2;
   a2 = temp;
}
```

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

25

Arrays as Function Arguments

- Remember these points when passing arrays to functions
  - Formal array arguments that are not to be altered by a function should be specified using the reserved word ***const***. When this specification is used, any attempt to alter the contents will cause the compiler generate an error message
- SameArray.cpp example

26

SameArray.cpp

```
// FILE: SameArray.cpp
// COMPARES TWO FLOAT ARRAYS FOR EQUALITY BY
// COMPARING CORRESPONDING ELEMENTS

// Pre:  a[i] and b[i] (0 <= i <= size-1) are
// assigned values.
// Post: Returns true if a[i] == b[i] for all I
// in range 0 through size - 1; otherwise,
// returns false.

bool sameArray (float a[], float b[],
                        const int size)
```

27

SameArray.cpp

```
{
   // Local data ...
   int i;
   i = 0;

   while ((i < size-1) && (a[i] == b[i]))
      i++;
   return (a[i] == b[i]);
}
```

28

AddArray.cpp

```
// Array elements with subscripts ranging from
// 0 to size-1 are summed element by element.
// Pre:  a[i] and b[i] are defined
// (0 <= i <= size-1
// Post: c[i] = a[i] + b[i] (0 <= i <= size-1)
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
```

29

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]

30

ReadScoresFile.cpp

```
// 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"
```

31

ReadScoresFile.cpp

```
void readScoresFile (ifstream& ins,int scores[],
       const int MAX_SIZE, int& sectionSize);

int main()
{
  int scores[100];
  int size;
  ifstream ins;

  ins.open(inFile);
```

32

ReadScoresFile.cpp

```
  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;
}
```

33

ReadScoresFile.cpp

```
// File: ReadScoresFile.cpp
// Reads an array of exam scores for a lecture
// section of up to MAX_SIZE students from a
// file.

// Pre:  None
// Post: The data values are read from a file
// and stored in array scores.
// The number of values read is stored in
// sectionSize.(0 <= sectionSize < MAX_SIZE).
```

34

ReadScoresFile.cpp

```
void readScoresFile (ifstream& ins, int scores[],
            const int MAX_SIZE, int& sectionSize)
{
   // Local data ...
   int tempScore;

   // Read each array element until done.
   sectionSize = 0;
   ins >> tempScore;
   while (!ins.eof() && (sectionSize < MAX_SIZE))
   {
      scores[sectionSize] = tempScore;
```

35

ReadScoresFile.cpp

```
      sectionSize++;
      ins >> tempScore;
   }  // end while

   // End of file reached or array is filled.
   if (!ins.eof())
   {
      cout << "Array is filled!" << endl;
      cout << tempScore << " not stored" << endl;
   }
}
```

36

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

37

ArrayOperations.cpp

```
// File: arrayOperations.cpp
// Finds the subscript of the smallest value in a
// subarray.

// Returns the subscript of the smallest value
// in the subarray consisting of elements
// x[startindex] through x[endindex]
// Returns -1 if the subarray bounds are invalid.
// Pre:  The subarray is defined and 0 <=
// startIndex <= endIndex.
// Post: x[minIndex] is the smallest value in
// the array.
```

38

ArrayOperations.cpp

```
int findIndexOfMin(const float x[],
                   int startIndex, int endIndex)
{
   // Local data ...
   int minIndex;
   int i;

   // Validate subarray bounds
   if ((startIndex < 0) || (startIndex >
                                   endIndex))
   {
```

39

ArrayOperations.cpp

```
      cerr << "Error in subarray bounds" << endl;
      return -1;
   }

// 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;
```

40

ArrayOperations.cpp

```
// All elements are examined and minIndex is
//    the index of the smallest element.
   return minIndex;
} // end findIndexOfMin
```

41

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

42

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

43

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++)
```

44

ArrayOperations.cpp

```
      if (items[next] == target)
          return next;

   // All elements were tested without success.
   return -1;
}  // end linSearch
```

45

Sorting in Ascending Order Selection Sort

- Idea of the selection sort is to locate the smallest value in the array
- Then switch positions of this value and that in position [0]
- We then increment the index and look again for the next smallest value and swap
- Continue until sorted

46

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.
```

47

ArrayOperations.cpp

```
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);
```

48

ArrayOperations.cpp

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

49

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

50

Analyzing Algorithms Big O Notation

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

51

Analyzing Algorithms Big O Notation

- Two independent loops
  - Sum of the loops is efficiency
  - n/2 + n^2 is Big O(N^2)

Example:

```
for (k=1; k<=n/2; ++k)
{
}
for (j=1; j<=n*n; ++j)
{
}
```

52

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:

```
for (k=1; k<=n/2; ++k)
{
    for (j=1; j<=n*n; ++j)
    {
    }
}
```

53

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

54

Struct Employee

```
// Definition of struct employee

struct employee
{
   string id;
   string name;
   char gender;
   int numDepend;
   money rate;
   money totWages;
};
```

55

Accessing Members of a struct

- 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

56

Accessing Members of a struct

organist.id = 1234;
organist.name = "Noel Goddard";
organist.gender = 'F';
organist.numDepend = 0;
organist.rate = 6.00;
organist.totWages += organist.rate * 40.0;

57

9.8 Structs as Operands and Arguments

- 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;

58

Passing struct as an Argument

- 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

59

ExamStat.h

```
// FILE: ExamStat.h

struct examStats
{
   string stuName;
   int scores[3];
   float average;
   char grade;
};
```

60

PrintStats.cpp

```
// File: printStats.cpp
// Prints the exam statistics

// Pre:  The members of the struct variable
// stuExams are assigned values.
// Post: Each member of stuExams is displayed.

void printStats(examStats stuExams)
{
   cout << "Exam scores for " <<
             stuExams.stuName << ": "
```

61

PrintStats.cpp

```
   cout << stuExams.scores[0] << ' ' <<
            stuExams.scores[1]<< ' ' <<
                  stuExams.scores[2] << endl;
   cout << "Average score: " <<
                   stuExams.average << endl;
   cout << "Letter grade : " <<
                     stuExams.grade << endl;
}
```

62

ReadEmp.cpp

```
// File: ReadEmp.cpp
// Reads one employee record into oneemployee

#include <string>
#include <iostream>

// Pre:  None
// Post: Data are read into struct oneEmployee
void readEmployee(employee& oneEmployee)
{
   cout << "Enter a name terminated with the
                         symbol # : ";
```

63

ReadEmp.cpp

```
   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;
}
```

64

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

65

Common Programming Errors

- No prefix to reference a struct member
- Incorrect prefix reference to a struct member
- Missing ; following definition of struct
- Initialization of struct members

66