Arrays

- Arrays are collections of like items which are referenced by position, not by name
  - One-Dimensional Arrays
  - Arrays of Records
  - Two-Dimensional Arrays

One-Dimensional Arrays

- A one-dimensional array is a collection of variables – all of the same type – in which the first part of the variable name is the same, and the last part is an index value enclosed in square brackets. Example: `string`
- The declaration of a one-dimensional array is similar to the declaration of a simple variable with one exception, you must also declare the size of the array.
- Example: `int value[1000];` The variables are `value[0]` – `value[999]`

Syntax Templates

ArrayDeclaration

DataType ArrayName [ConstIntExpression];

ArrayComponentAccess

ArrayName [Index Expression]

The index expression may be as simple as a constant or a variable name or as complex as a combination of variables, operators, and function calls.

Array Examples

Consider the following declarations:

```
float angle[4];
int testScore[10];
```

and the assignments:

```
angle[0] = 4.93;        angle[2] = 0.5;
angle[1] = -15.2;       angle[3] = 1.67;
```

Each array component can be treated exactly the same as any simple variable of type `float`.

```
Ex:   angle[2] = 9.6;
      cin >> angle[2];
      cout << angle[2];
      y = sqrt(angle[2]);
      x = 6.8 * angle[2] + 7.5;
```


Out-of-Bounds Array Indexes

- Problem: Given the declaration `float alpha[100],` valid indexes for the array range from 0 – 99. What happens if you access `alpha[105]`, for example?
- Answer: C++ does not check for out-of-bounds indexes either at compile time or at run time. The memory location is destroyed. It is entirely the programmer's responsibility to make sure that an array does not step off either end of the array.

More About Arrays

- Arrays can be initialized in their declarations.
  Example: `int age[5] = {23, 10, 16, 37, 12};`
- C++ does not allow aggregate operations on arrays. To copy one array into another, you must do it yourself, element by element. You also may not do an aggregate comparison of arrays. It's not possible to return an entire array as the value of a value-returning function. You can pass an entire array as an argument to a function.

Examples of Declaring and Accessing Arrays

```
const int BUILDING_SIZE = 350;
int occupants[BUILDING_SIZE];
int totalOccupants;
int counter;
totalOccupants = 0;
for (counter = 0; counter < BUILDING_SIZE; counter++)
   totalOccupants = totalOccupants + occupants[counter];
```


Examples of Declaring and Accessing Arrays

```
const int BUILDING_SIZE = 350;
int occupants[BUILDING_SIZE];
int totalOccupants;
int counter;
totalOccupants = 0;
for (counter = 0; counter < BUILDING_SIZE; counter++)
   totalOccupants = totalOccupants + occupants[counter];


enum Drink {ORANGE, COLA, ROOT_BEER, GINGER_ALE, CHERRY,
   LEMON};
float salesAmt[6];
for (flavor = ORANGE; flavor <= LEMON; flavor =
   Drink(flavor + 1))
    cout << salesAmt[flavor] << endl;
```


Passing Arrays as Arguments

- By default, C++ simple variables are always passed by value.
- Arrays are always passed by reference.
  - Example:

```
void ZeroOut (/* out */ float arr[],
              /* in */ int numElements)
{
   int i;
   for (i = 0; i < numElements; i++)
       arr[i] = 0.0
}
```


Preventing the Function from Modifying an Array

- Use the reserved word `const`.
- Example:

```
void Copy(/* out */      int destination[],
          /* in */ const int source[],
          /* in */       int size)
{
int i;
for (i = 0; i < size; i++)
      destination[i] = source[i];
}
```


Arrays of Records

- Many applications require a collection of records.
- Example:

```
const int MAX_STUDENTS = 150;
enum GradeType {A, B, C, D, F};
struct StudentRec
{
   string stuName;
   float  gpa;
   int examScore[4];
   GradeType courseGrade;
};

StudentRec gradeBook[MAX_STUDENTS];
int count;
```


Arrays of Records

- Valid assignment statements given the declaration of `gradeBook`

```
gradeBook[55].gpa = 4.0;
gradeBook[103].stuName = “Harold Schmerdlap”;
gradeBook[149].examScore[0] = 83;
gradebook[0].GradeType = B;

for (count = 0; count < MAX_STUDENTS; count++)
   cout << gradeBook[count].stuName << endl;
```

Two-Dimensional Arrays

- A two-dimensional array is used to represent items in a table with rows and columns of the same data type item.
- Each component is accessed using two indices that represent the position in each dimension.
- ArrayDeclaration

  DataType Array Name [ConstInt Expression] [ConstIntExpression]… ;
- ArrayComponentAccess
  - Ø ArrayName[Index Expression]
  - Ø [Index Expression] …;

Two-Dimensional Array Examples

```
const int NUM_ROWS = 100;
const int NUM_COLS = 9;
float alpha[NUM_ROWS][NUM_COLS];
```


Two-Dimensional Array Examples

```
const int NUM_ROWS = 100;
const int NUM_COLS = 9;
float alpha[NUM_ROWS][NUM_COLS];

int hiTemp[52][7];
```


Two-Dimensional Array Examples

```
const int NUM_ROWS = 100;
const int NUM_COLS = 9;
float alpha[NUM_ROWS][NUM_COLS];

int hiTemp[52][7];

enum Colors {RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO,
     VIOLET};
enum Makes {FORD, TOYOTA, HYUNDAI, JAGUAR, CITROEN,
     BMW, FIAT, SAAB};
float crashRating[7][8];

crashRating[BLUE][JAGUAR] = 0.83;
crashRating[RED][FORD] = 0.19;
```


Processing Two-Dimensional Arrays

- Process the rows.
- Process the columns.
- Initialize the array.
- Print the array.