Intro to arrays

- so far all our data types have held simple, primitive values: single integers, single real numbers, single characters, etc
- if we had to hold hundreds or thousands of separate values we would need to declare variables for each: not very effective
- we'd like a way to declare collections of values, e.g. a collection of 1000 student numbers, a collection of 100,000 temperatures we measured over time, a collection of 17 item prices on a bill
- most programming languages support “arrays” as one means of creating such collections

Arrays in C++

- to define an array in C++ we give three things:
  - a name for the array
  - the number of items it can hold
  - the type of item it holds (e.g. int, float, etc)

```
int myArray[200]; // myArray holds 200 ints
```

- the array is organized into positions, each acting as a storage spot for one item
- when we want to access an item we specify the array name and position, with the position in [ ]

```
x = arr[10]; // copy what's in position #10 into x
arr[17] = 205; // store 205 in position #17
```

Array indexing and elements

- the values stored in the array are called elements, the positions are called indices
- for an array of size N the positions are numbered 0..N-1
- for example, here we read values into all 5 positions then print them out:

```
float values[5];
cout << “Enter 5 numbers” << endl;
cin >> values[0];
cin >> values[1];
cin >> values[2];
cin >> values[3];
cin >> values[4];
```

```
cout << “The five values were:” << endl;
cout << values[0];
cout << values[1];
cout << values[2];
cout << values[3];
cout << values[4];
```

Array sizes

- the size of an array must be a positive integer, and must be a constant (or expression of constants), e.g.

```
const int Size = 10;
const int N = 3;
int arrayOne[Size];
int arrayTwo[N];
int arrayThree[N * Size];
```

- *(some compilers allow the use of variables for a size, but this isn't universally supported, we'll look at alternatives later)*

Array elements

- an array element (e.g. arr[3]) can be used anywhere that its data type is valid: in expressions, passed as parameters, etc

```
const int size = 10;
float arr[size];
cin >> arr[0]; // get and store first element, read from user
cin >> arr[1]; // get and store second element
arr[2] = arr[1] + arr[0];  // add two elements, store in third
arr[3] = pow(arr[2], arr[0]); // 4th is 3rd to power of 1st elem
```

Initializing array contents

- like variables, the contents of the array could be any values until/unless we specifically store something

```
int x;
cout << x; // x could be any random integer
int arr[3];
cout << arr[0]; // again, could be any random integer
```

- we say the variables/array in such cases are uninitialized
- the first time we store a value in a variable or array position we are *initializing* that variable/array element
- we always want to initialize items before attempting to use their contents

Initializing arrays at declaration

- we can assign initial values to an array as part of the declaration (this is the only place we can assign multiple values to an array at once):

```
int x[5] = { 10, 20, 30, 40, 50 };
```

- the number of elements inside the { } must match the size of the array
- while sometimes handy for small arrays, this is impractical for larger arrays

Setting variable values with loops

- We'll often use loops to process the contents of an array, e.g. setting the value of every element (one at a time), printing each element, updating each element, etc

```
// set every element in an array to value 101
const int size = 10;
float data[size];
// will use local variable pos to track current position
for (int pos=0; pos<size; pos++) {
    data[pos] = 101; // set value in current element
}
```

Get user data to fill array

```
const int ArrSize = 50;
double array[ArrSize];

// fill with user data
for (int p=0; p<ArrSize; p++) {
    cout << “Enter a number” << endl;
    cin >> array[p];
}

// print it all out again
for (int pos=0; pos<ArrSize; pos++) {
     cout << “The value in position “ << pos;
     cout << “ is “ << array[pos] << endl;
}
```

More array examples

More array examples

```
// still using our array from the previous slide

// compute the sum of all elements
float sum = 0;
for (int p=0; p<ArrSize; p++) {
    sum += array[p];
}
cout << “sum of all values is “ << sum << endl;

// find smallest element
float smallest = array[0]; // first is smallest so far
for (int pos=1; pos<ArrSize; pos++) {
    if (array[pos] < smallest) {
       smallest = array[pos]; // we've found a new smallest so far
    }
}
cout << “Smallest value is “ << smallest << endl;
```