Week 11: Functions and Arrays

Gaddis: 7.8

CS 1428
Fall 2014

Jill Seaman

Functions and Array Elements

- An **array element** can be passed to any parameter of the same (or compatible) type:

```
double square (double);

int main() {
   double numbers[5] = {2.2, 3.3, 5.11, 7.0, 3.2};

   for (int i=0; i<5; i++)
      cout << square(numbers[i]) << " ";
   cout << endl;
   return 0;
}

double square (double x) {
   return x * x;
}
```

Output:

```
4.84 10.89 26.1121 49 10.24
```


Functions and Array Elements

- An **array element** can be passed by r**eference**. What is output by this program?

```
void changeMe(int &myValue) {
   myValue = 200;
}

int main() {
   int numbers[5] = {2, 3, 5, 7, 3};

   for (int i=0; i<5; i++)
      changeMe(numbers[i]);

   for (int i=0; i<5; i++)
      cout << numbers[i] << " ";
   cout << endl;
}
```


7.8 Arrays as Function Arguments

- An entire **array** can be passed to a function that has an array parameter

```
void showArray(int[], int);

int main() {
   int numbers[5] = {2, 3, 5, 7, 3};
   showArray(numbers,5);
   return 0;
}

void showArray(int values[], int size) {
   for (int i=0; i<size; i++)
      cout << values[i] << " ";
   cout << endl;
}
```

Output:

```
2 3 5 7 3
```

Passing arrays to functions

- In the function definition, the parameter type is a variable name with an empty set of brackets: [ ]
  - Do NOT give a size for the parameter

```
void showArray(int values[], int size)  {…}
```

- In the prototype, empty brackets go after the element datatype.

```
void showArray(int[], int);
```

- In the function call, use the variable name for the array.

```
showArray(numbers, 5);
```


Passing arrays to functions

- An array is **always** passed by **reference**.
- The parameter name is an alias to the array being passed in, even though it has no &.
- Changes made to the array (elements) inside the function **DO** affect the array in the function call.

Passing arrays to functions

- Changing an array inside a function:

```
void incrArray(int[], int);
void showArray(int[], int);

int main() {
   int numbers[5] = {2, 3, 5, 7, 3};
   incrArray(numbers,5);
   showArray(numbers,5);
   return 0;
}

void incrArray(int values[], int size) {
   for (int i=0; i<size; i++)
      (values[i])++;          //values[i]=values[i]+1;
}
```

Output:

```
3 4 6 8 4
```


Passing arrays to functions

- Usually functions that have an array parameter also have an int parameter for the count of the number of valid elements in the array.
  - so the function knows how many elements to process.
- The count parameter is just a regular int parameter and must be listed in the parameter list and included in the function call.

Passing partially filled arrays to functions

```
const int MAX_STUDENTS = 100;  // max number of students
int sumArray (int[], int);     // func to sum elems, returns int

int main() {
   int scores[MAX_STUDENTS];   // array of scores
   int count = 0;              // number of scores in array
   ifstream infile;
   infile.open("students.txt");
   while (count<MAX_STUDENTS && infile >> scores[count])
      count++;
   cout << "Total is: " << sumArray(scores, count) << endl;
}

int sumArray (int arr[], int numElems) {
  int total = 0;
  for (int x = 0; x < numElems; x++)
     total = total + arr[x];
  return total;
}
```