Arrays (and Efficiency)

CSC 1230
AUTUMN 2014
MWF 11AM – 12:20 PM
OMH 244

EXCERPTS FOR
CSC 2430 – WIN2016

DR. AARON DINGLER
OMH 242
(206) 281-2943
DINGLERA@SPU.EDU

Outline

▪ Arrays & Array Processing	Malik 7th: Chapter 8
▪ Arrays & Array Processing	520 – 542, 558 – 559
▪ C-Strings (Char arrays)	550 – 557
▪ Parallel Arrays	558
▪ 2-dimensional arrays	559 – 575

Arrays

- Arrays are "lists" of data in contiguous (adjacent) memory locations
  - Arrays can be used to store any primitive data type (`int`, `double`) or object (`string`, etc.)
  - Share *some* properties with strings
- Some examples:

```
int     myArray[100];      // individual array elements are uninitialized
int     arrayInts[5] = {1, 2, 3, 4, 5};      // initialize at declaration
float   gasMileages[20] = {23.5, 19.0, 36.8, 11.3 };  // rest of elts
uninitialized
string  myFriends[5] = { "Mike", "Sherry", "Fred", "Russell", "Laurie" };
string  theGroup[10];
```

- Operations? Individual element manipulation, usually inside loops

```
theGroup[0] = "Doug";
theGroup[1] = "Marshawn";

cout << arrayInts << endl;   // Error – Doesn't work.  Must specify subscripts

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

Creating Arrays

1. Declare a `const` for array maximum size
2. Declare variable(s) where needed
   - In `main()` if to be passed into functions
   - Local to a function if "temporary" (for that function's use)

- Example: in `main()`

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

const int MAX = 100;  //Static array size

int main()
{
     float floatArray[MAX];
     int countOfValues;
       //ready to process...
     return 0;
}
```

Initializing Array Elements

- If it's a small array, can initialize "by hand" as in our first example:

```
const int MAX = 5;
int arrayInts[MAX] = {1, 2, 3, 4, 5};
```

  - Can also initialize partially:

```
int arrayInts[MAX] = {1, 2, 3};
```

    - Final two elements are initialized to 0
- Typically, one works with either:
  - Much larger arrays
  - Arrays where don't know values ahead of time
- And thus we will need to:
  - Initialize arrays using a loop
  - Initial values from user (keyboard), or from a file, from a function, etc.

Details on Accessing Arrays

- As seen in our first example, access using the indexing operator [ ]

```
for(int i = 0; i < MAX; i++)
   cout << arrayInts[i] << endl;
```

  - Similar to accessing individual characters in a string
    - However, we **cannot** use .at() with arrays – that is a *method* of the string class
    - An array is not a class/object in C++!
      - Again, an array is simply defined as a series of adjacent elements of the same type in memory
- **Be careful** of accessing elements out of the array's bounds
  - This will likely cause your program to crash
    - Officially, the behavior is "undefined" (likely system-dependent)

Initializing Arrays

- Let's write a function to initialize every array element to 0 (in an array of floats)

```
const int  MAXSIZE = 10;
. . .
void initarray(float a[], int numelts) // arrays pass by reference
{
   for(int i = 0; i < numelts; i++) {
      a[i] = 0;                          // a[i] = i;  ???
   }
}

void main() {
   float  myarray[MAXSIZE];
   initarray( myarray, MAXSIZE );

   // print out the array
   for(int i=0; i<MAXSIZE; ++i) {
      cout << "myarray[" << i << "] = " << myarray[i] << endl;
   }
}
```

Input into Arrays

- Let's look at ways to *input* data into our arrays
  - i.e., from a source outside the C++ program
- Data can be read and stored until
  - Sentinel or end-of-file is reached, **or**
  - Array is full
- Functions often have array and MAXSIZE as parameters **and** return the number of values read-and-stored

1. Read from keyboard
2. Read from file (while not eof)
   - File has been opened elsewhere (i.e., in `main`)
3. Read string values from file (with getline)
   - Reading strings
   - File has been opened elsewhere (i.e., in `main`)

Reading from the Keyboard

- Let's write a function to read floats from the keyboard and store each inputted element in an array
  - We will use *cin* (since we are getting *floats*)
  - After each input, we will ask the user if there is more data
    - If 'y', ask for a new input, if 'n', stop
  - If we have filled up the array, we will also stop asking for input

```
int getArray(float a[ ], int maxsize)
{
   int  i;
   float val;
   for(i = 0; i < maxsize; i++) {
      cout << "Enter float value: ";
      cin >> val;
      a[ i ] = val;

      cout << "More (y)? ";
      char yesno;
      cin >> yesno;
      if(yesno != 'y') break; // early exit from loop
   }

   return (i + 1); // return number of items read
}
```

Using Arrays for More Complex Operations

- We can use arrays to perform some more complex operations
  - These operations will usually iterate over the entire array,
  - And calculate and return some value as a result

- For example:
  - A function to sum each element in an array, return the sum
  - A function to find and return the largest value in an array
  - A function to find and return the *location* of the largest value in an array
  - Putting the three functions together to output "statistics" for the array
  - Search for *specific* elements, or sort the array