## cis1.5 spring2009 lecture IV.3

today we are going to talk about...
doing things with arrays and strings

- passing arrays/strings to/from functions
- 2-dimensional arrays



## using arrays as global variables

- the example below shows the use of an array declared as a global variable and used inside a function
- note the use of a constant: `const int MAX=5;` to declare the size of the array; `MAX` is also used throughout the program whenever we need to refer to the number of elements in the array
- this makes it easier to change the number of elements in the array
- it also makes the code look cleaner; whenever we see `MAX`, we know that it is being used to refer to the size of the array



- example:

```
/*
 * array0.cpp     (sklar/05-apr-2009)
 */

#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
using namespace std;

// declare global variables
const int MAX = 5; // declare size of array as a constant
int myArray[MAX]; // declare array, globally

void printArray() {
  for ( int i=0; i<MAX; i++ ) {
    cout << i << "-th element = " << myArray[i] << endl;
  } // end for i
} // end printArray()

int main() {
  srand( time( NULL ));
  for ( int i=0; i<MAX; i++ ) {
    myArray[i] = ( rand() % 10 ) + 1;
  }
  printArray();
} // end of main()
```



## passing arrays to functions

- the example below shows how to use an array that is declared locally, for example inside `main()`, and then is passed to a function for use inside the function
- it is like passing a non-compound variable (like an `int` or a `char`) where you specify in the function header the name and data type of the parameter
- but you also have to specify that the variable is an array by using the square brackets (`[]`) after the name
- you can include the size of the array in the definition, for example:
  `void printArray( int myArray[MAX] );`
- but you don't have to specify the size, for example:
  `void printArray( int myArray[] );`
- it is common to include the size of the array as another argument, which makes the function more useful, because then it can be called with an array of any size, for example:
  `void printArray( int myArray[], int numArray );`

- example:

```
/*
 * array1.cpp     (sklar/05-apr-2009)
 */

#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
using namespace std;

// declare global variable
const int MAX = 5; // declare size of array as a constant

void printArray( int myArray[], int numArray ) {
  for ( int i=0; i<numArray; i++ ) {
    cout << i << "-th element = " << myArray[i] << endl;
  } // end for i
} // end printArray()

int main() {
  int myArray[MAX]; // declare array locally; scope = main()
  srand( time( NULL ));
  for ( int i=0; i<MAX; i++ ) {
    myArray[i] = ( rand() % 10 ) + 1;
  }
  printArray( myArray, MAX );
} // end of main()
```



## passing arrays to functions as reference parameters

- the example below shows how to pass an array as a reference parameter to a function—where the value of the array elements can change inside the function and retain their new values after exiting the function, for example:
  `void initArray( int (&myArray)[MAX], int numArray )`
- note the syntax: you have to put the reference operator (`&`) in parentheses grouped with the name of the array (`myArray`)
- also note that in this form, you MUST specify the size of the array
- however, because an array is a type of compound variable, you do not need to specify the reference operator, so the following also works, for example:
  `void initArray( int myArray[], int numArray );`



- example:

```
/*
 * array2.cpp     (sklar/05-apr-2009)
 */

#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
using namespace std;

// declare global variable
const int MAX = 5; // declare size of array as a constant

void printArray( int myArray[], int numArray ) {
  for ( int i=0; i<numArray; i++ ) {
    cout << i << "-th element = " << myArray[i] << endl;
  } // end for i
} // end printArray()

void initArray( int myArray[], int numArray ) {
  for ( int i=0; i<numArray; i++ ) {
    myArray[i] = ( rand() % 10 ) + 1;
  } // end for i
} // end of initArray()

int main() {
  int myArray[MAX]; // declare array locally; scope = main()
  srand( time( NULL ));
  initArray( myArray, MAX );
  printArray( myArray, MAX );
} // end of main()
```



## using strings as global variables

- you can also declare strings as global variables and use them inside functions
- example:

```
/*
 * string0.cpp     (sklar/05-apr-2009)
 */

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

// declare global variable
string myString;

void printString() {
  cout << "your string = [" << myString << "]" << endl;
} // end printString()

int main() {
  cout << "enter a string: ";
  getline( cin, myString );
  printString();
} // end of main()
```

## passing strings to functions

- as with an array, you can also pass a string as an argument to a function, for example:
  `void printString( string myString );`
- note that you don't have to worry about the size of the string, since the `string` data type (defined in <string>) takes care of that for you
- example:

```
/*
 * string1.cpp     (sklar/05-apr-2009)
 */

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

void printString( string myString ) {
  cout << "your string = [" << myString << "]" << endl;
} // end printString()

/**
 * define main() function
 */
int main() {
  string myString; // declare as a local variable inside main()
  cout << "enter a string: ";
  getline( cin, myString );
  printString( myString );
} // end of main()
```



## passing strings to functions as reference parameters

- as with an array, you can also pass a string to a function as a reference parameter, where you can change the value of the string inside the function and retain the new value outside the function, for example:
  `void initString( string &myString );`
- note that with a string, you have to use the reference operator (&), otherwise the new value set inside the function will not be retained when you exit the function



- example:

```
/*
 * string2.cpp     (sklar/05-apr-2009)
 */

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

void printString( string myString ) {
  cout << "your string = [" << myString << "]" << endl;
} // end printString()

void initString( string &myString ) {
  cout << "enter a string: ";
  getline( cin, myString );
} // end of initString()

int main() {
  string myString;   // declare local variable
  initString( myString );
  printString( myString );
} // end of main()
```



## returning a string as the value of a function

- because a `string` is a special kind of data type (called an `object`), you can also define functions that have `string`s as their data type, which means that they return a `string`, for example:
  `string initString();`
- note that you cannot do this with arrays in the same way (there are other ways, but that is beyond the scope of this class)

- example:

```
/*
 * string3.cpp     (sklar/05-apr-2009)
 */

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

void printString( string myString ) {
  cout << "your string = [" << myString << "]" << endl;
} // end printString()

string initString() {
  string localString;
  cout << "enter a string: ";
  getline( cin, localString );
  return( localString );
} // end of initString()

int main() {
  string myString;
  myString = initString();
  printString( myString );
} // end of main()
```



## using arrays of strings

- because a `string` is a special kind of data type (called an `object`), you can also define arrays of `strings`, for example: `string myArray[MAX];`
- an array of `strings` is handled basically just like an array of a simple data type (like an array of `ints`)



- example:

```
/*
 * arrayString.cpp     (sklar/05-apr-2009)
 */

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

// declare global variable
const int MAX = 5; // declare size of array as a constant

void printArray( string myArray[], int numArray ) {
  for ( int i=0; i<numArray; i++ ) {
    cout << i << "-th element = " << myArray[i] << endl;
  } // end for i
} // end printArray()

void initArray( string myArray[], int numArray ) {
  cout << "please enter 5 words: ";
  for ( int i=0; i<numArray; i++ ) {
    cin >> myArray[i];
  } // end for i
} // end of initArray()

int main() {
  string myArray[MAX];
  initArray( myArray, MAX );
  printArray( myArray, MAX );
} // end of main()
```



## 2-dimensional arrays

- so far, we have only talked about *one-dimensional* arrays, where you can think of the data as being stored in EITHER a single row OR a single column in spreadsheet
- but we can also define *two-dimensional* arrays, where you can think of the data as being stored in BOTH rows and columns
- you define a two-dimensional array by adding another set of square brackets, with a value in between indicating the size of the dimension, for example:
  `int myArray[3][4];`
  which declares a two-dimensional array with 3 *rows* and 4 *columns*
- we will refer to the two dimensions as rows and columns so that you can visualize a spreadsheet, like this:

| | | | | |
|---|---|---|---|---|
| 3 | 1 | 1 | 9 | *there are 3 rows, going horizontally* |
| 2 | 5 | 6 | 8 | *there are 4 columns, going vertically* |
| 5 | 7 | 4 | 4 | |

- C++ defines arrays as *"row major"*, which means that the row dimension comes first, then the column dimension

- note that you can define multi-dimensional arrays (i.e., more than 2 dimensions), but that is beyond the scope of this class
- when you pass two-dimensional arrays to functions, it is a bit more complex because you need to include the size of the second dimension, otherwise the function gets confused about how many elements (i.e., columns) are in a row, for example:

```
void printArray( int myArray[][WIDTH], int length, int width );
```

- as with one-dimensional arrays, you can also pass a two-dimensional array to a function as a reference parameter, where any content changed inside the function is retained outside the function after the function exits, for example:

```
void initArray( int myArray[][WIDTH], int length, int width );
```



- example:

```
/*
 * array2D.cpp     (sklar/05-apr-2009)
 */

#include <stdlib.h>
#include <sys/time.h>
#include <iostream>
using namespace std;

// declare global variables
const int LENGTH = 3;
const int WIDTH  = 4;

void printArray( int myArray[][WIDTH], int length, int width ) {
  for ( int y=0; y<length; y++ ) {
    for ( int x=0; x<width; x++ ) {
      cout << "(" << x << "," << y << ") = " << myArray[y][x] << "   ";
    } // end for x
    cout << endl;
  } // end for y
} // end printArray()

void initArray( int myArray[][WIDTH], int length, int width ) {
  for ( int y=0; y<length; y++ ) {
    for ( int x=0; x<width; x++ ) {
      myArray[y][x] = ( rand() % 10 ) + 1;
    } // end for x
  } // end for y
} // end of initArray()

int main() {
  int myArray[LENGTH][WIDTH];
  srand( time( NULL ));
  initArray( myArray, LENGTH, WIDTH );
  printArray( myArray, LENGTH, WIDTH );
} // end of main()
```