CSC 215

Math and Computer Science

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

A Legacy of Excellence

What are Arrays

- Means of storing multiple values of the same data type
- Accessed using one variable name
- All elements in the array are in contiguous memory

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY

Why Use Arrays

```
int student1, student2, student3;
cout << “Enter grade for each student”;
cin >> student1 >> student2 >> student3;
```

Easy for three students. Now do the for all csc 150 students

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Declaration

Syntax: dataType variableName[quantity];

Example: int exam1Scores[200];

double averages[20];

Quantity

- Must be an integer constant greater than 0
- Can use an integer constant (global) to set the quantity

```
const int MAX = 200;
int exam1Scores[ Max ];
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Usage

- Each element (member) is access with an index
- Index is an integer value starting at 0 thru (size)-1
    - Referred to as "Zero-based Indexing"
    - Also called a subscript, lvalue, offset
    - An index thus refers to the (index+1)th element of the array
- All elements of an array are the same data type

Components of the array

Assume   int data[10];

- Array name by itself contains a memory address
  - Data is just an address
- Index (subscript) represents a position in the array

- Together the provide access to a single item in the array

  data[3] is a single integer

Overall Picture

Assume: `int data[10];`

index	0	1	2	3	4	5	6	7	8	9
data										

`data[3] = 4;  // would assign 4 to the fourth element in the array`

index	0	1	2	3	4	5	6	7	8	9
data				4						

Accessing Array Elements

```
cout << data;
```

Outputs the address of the array

```
        forward                                 backwards
for( i=0; i<SIZE; i++)                  for( i=SIZE-1; i>=0; i--)
        cout << data[i];                        cout << data[i];


for( i=0; i<SIZE; i++)
        data[i] = sqrt( data[i] );
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Dangers

Assume: int data[10];

- Compiler does not warn you about an invalid index
- Data[10] is not within the array.
  - 10 is an invalid index.
  - Access the next memory after the array and changes it
  - It is likely to be another variable
- This is called overstepping the bound

Assignment

```
int data1[5], data2[5];
```

- You can not directly assign the contents of one array to another
  - `data2 = data1;`
  - That is because data1 and data2 are just addresses
- Must copy data from one array to another element by element using a loop structure

```
for( i=0; i<5; i++)
        data2[i] = data1[i];
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Initializer Lists

int data[5] = { 1, 3, 11, 13, 21 };

- Copies values in at declaration time

index	0	1	2	3	4
data	1	3	11	13	21

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Initializer Lists - Continued

int data[5] = { 1, 3, 11 };

- If you do not provide enough, the value 0 is used

index	0	1	2	3	4
data	1	3	11	0	0

int data[5] = { 0 };

index	0	1	2	3	4
Data	0	0	0	0	0

Initializer Lists - Continued

int data[5] = { 1, 3, 11, 13, 21, 23 };

- If you provide to many, you get a compiler err

int data[5];

- Uninitialized list contain random values.

index	0	1	2	3	4
data	??	??	??	??	??

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Initializer List - Continued

```
int data[5] = {1};
```

index	0	1	3	4	5
data	1	0	0	0	0

```
for( i=0; i<5; i++)
        data[i] = 1;
```

index	0	1	3	4	5
data	1	1	1	1	1

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Initializer Lists - Continued

- Let the list determine the size of the array

```
int data[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

- The size of the data array is 10 elements

Index	0	1	2	3	4	5	6	7	8	9
Data	1	2	3	4	5	6	7	8	9	10

- Not recommended, you don't see what size the array is when debugging.

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Array Uses

- Store large number of related values
  - Student exams
  - Hourly temperatures for the entire year
- Store data for easy lookup (tables)
- Store data to be used repetitively in a calculation
  - Standard Deviation – must compute average of numbers before equation

$$\sqrt{\frac{\sum_{i=0}^{n-1}(average - example_i)^2}{n-1}}$$

SOUTH DAKOTA
SCHOOL OF MINES & TECHNOLOGY

Standard Deviation Example

```
const int NUM_SCORES = 10;
int main ()
{
   int  i;
   double  scores[NUM_SCORES];
   double  avg = 0.0, sum = 0.0, stDev = 0.0;

   for ( i = 0; i < NUM_SCORES; i++ )
   {
      cin >> scores[i];
      sum += scores[i];
   }
   avg = sum / NUM_SCORES;

   sum = 0;
   for ( i = 0; i < NUM_SCORES; i++ )
   {
             sum += pow ( avg - scores[i], 2 );
   }

   stDev = sqrt ( sum / ( NUM_SCORES - 1 ) );

   cout << "The average score is: " << avg << endl;
   cout << "The Standard Deviation is: "
          << stDev << endl << endl;
   return 0;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Passing Arrays to Function

- Function Prototype

  returnType functionName(dataType arrayName[], …);

  - [] indicate that an array of anysize may be passed to this function
  - arrayName is the address of the first element
  - Size of the array must be explicitly passed as a separate parameter if needed. (This is a good practice for writing flexible code)

EX: void printArray(int data[], int size);

Passing Arrays to Function

- Function Header / Definition
- Same as the prototype but without the semicolon
- Arrays are pass by reference only.

Pass by Reference Example

```
void incArray( int data[], int size );
int main()
{
    int my_array[5] = {1,2,3,4,5};
    int k;

    incArray( my_array, 5 );
    for( k = 0; k < 5; k++ )
        cout << my_array[k] << "   "  ;

    cout << endl;

    return 0;
} //output is 2 3 4 5 6
```

```
void incArray( int theArray[], int size )
{
    int i;

    for( i = 0; i < size; i++ )
        theArray[i]++;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Passing an Array Element

- Follows the same rules as for a non array variable

```
void function1( int num );          // pass by value function
void function2( int & value );      // pass by reference function
int a=5,  array[3]={1,2,3};

function1(a);                       // a passed by value
function1(array[2]);                // array[2] passed by value
function2(a);                       // a passed by reference
function2(array[1]);                // array[1] passed by reference
```

Filling an Array

```
int i;
double lowTemps[365];
:
for( i=0; i<365; i++)
{
        cout << "Enter the low temperature";
        cin >> lowTemps[i];
}
```

Retrieving an element

- Assume array from previous slide

```
int spot;
cout << "Which day would you like to retrieve: ";
cin >> spot;

if( spot >=0 && spot <365)
    cout << "Temperature for that day is: " << lowTemps[spot] << endl;
else
    cout << "Invalid position" << endl;
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Finding Minimum Value

```
int findMinimum( double array1[], int size )
{
    int i;
    double min;
    min = array1[0];
    for( i=1; i<size; i++)
    {
       if(array1[i] < min )
              min = array1[i];
    }
    return min;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Finding an element as a function

```
int findItem( int array1[], int size,
              int targetElement)
{
     int i;
     for( i=0; i<size; i++)
     {
          if( array1[i] == targetElement)
               return i;
     }
     return -1;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Counting Function

```
int countBelowZero( double temperatures[], int size )
{
    int i;
    int count = 0;
    for( i=0; i<size; i++)
    {
        if( temperatures[i] < 0 )
            count++;
    }
    return count;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY