The C++ Language

Arrays

Structure Data Types

- Collections of component items
  - Each item can be accessed individually
- In C++
  - Arrays
  - Structs
  - Unions
  - Classes
Motivation

- Suppose you wanted to read in 100 temperatures. How would you store them all?

```c
double temp00 = 0.0,
          temp01 = 0.0,
          temp02 = 0.0,
          ...
          temp99 = 0.0;
```

- What if you had to print all of them out? How would you do it?

Arrays

- A collection of elements of the same type
- Each element is accessed by indexing
- Mathematics
  ```
  \text{temp}_0 \text{ temp}_1 \text{ temp}_2 \ldots \text{ temp}_{99}
  ```
- C++
  ```
  \text{temp}[0] \text{ temp}[1] \text{ temp}[2] \ldots \text{ temp}[99]
  ```
Arrays

- Memory
  
  | Memory | temp[0] |
  |        |        |
  |        | temp[1]|
  |        | temp[2]|
  |        | temp[3]|
  |        | temp[4]|
  |        | ...   |
  |        | temp[98]|
  |        | temp[99]|
**Declaration Examples**

```c++
// Function prototypes

// Constants
const int MAX_NAMES = 50;
const int MAX_GRADES = 100;

// Main program
void main() {
    int grades[MAX_GRADES];
    string names[MAX_NAMES];

    // computations in main()
}

// Function definitions and documentation
```

Array sizes should always be named constants!

---

**Element Access**

- Individual elements in an array are access by **indexing**
  - An *index* is an integer expression

**Syntax**

```
ArrayIdentifier[IntegerExpression]
```

- Indices always start at 0 in C++!
void main() {
    int grades[MAX_GRADES];
    string names[MAX_NAMES];
    int i;

    // read in grades
    for (i = 0; i < MAX_GRADES; i++) {
        cin >> grades[i];
    }
    // Exercise: read in names
    // Print the 20th name
    cout << names[19] << endl;
}

void main() {
    int grades[MAX_GRADES];
    string names[MAX_NAMES];
    int i;

    // Initialize in reverse order
    for (i = 1; i <= MAX_GRADES; i++) {
        grades[MAX_GRADES - i] = i;
    }
}
**Array Definitions**

- *Capacity*, the largest number of elements the array can legally store.
- *Usage*, the largest number of elements actually in use.

```
a[0] a[1] a[2] … a[57] … a[99]
```

| 23 | -10 | 59 | … | 71 | … |

Unassigned

**Aggregate Operations**

- Although you really want some things to work, they don't.

```cpp
const int MAX_SIZE = 50;
int x[MAX_SIZE];
int y[MAX_SIZE];
x = y;    // No, does not copy arrays
x == y;  // No, does not compare elements in arrays
cout << x; // No, you'll get an error (most of the time)
x + y;    // No, doesn't add elements in arrays
return x; // No, does not return a copy of the array
someFunc(x); // YES, we can pass arrays as parameters
```
Aggregate Operations

- Compare two arrays for equality
  
  ```cpp
  bool areEqual = (xUsage == yUsage);
  for (int i = 0; areEqual && i < xUsage; i++) {
    areEqual = (areEqual && x[i] == y[i]);
  }
  ```

- What about printing an array?
- Great candidates for turning into functions!
  - We will return to arrays and functions later

Common Mistake

- The most common mistake is to access an array out of bounds...

```cpp
const int MAX_SCORES = 100;
int scores[MAX_SCORES];

for (int i = 0; i <= MAX_SCORES; i++) {
  scores[i] = 0;
}
```

- Assigns 0 to `scores[100]`! What happens?
C++ allows you to index an array out of bounds
- This is evil, but checking array bounds can be costly

![But this is not space assigned to the scores array!]

| scores[0] |   |
| scores[1] |   |
| scores[2] |   |
| ...       | ...|
| scores[99]|   |
| scores[100]|   |

The results of accessing arrays out of bounds are unpredictable
- May appear to run just fine
- If you're lucky, will cause a runtime error
- May corrupt another variable/value in your program
  - Unexpected infinite loops
  - Unexpected changes in values output
- If your program runs on your machine but not on the Curator, look for array out of bounds errors!!!
Array Initialization

- Most commonly use a loop to initialize arrays

```c
const int MAX_SCORES = 100; // Capacity of scores array
int scores[MAX_SCORES];

for (int i = 0; i < MAX_SCORES; i++) {
    scores[i] = 0; // change 0 to whatever initial value
}
```

- Make this a function!!!!

Initialization at Declaration

- You can also initialize arrays at declaration
  - When you know size and values of array beforehand

```c
const int MAX_HEIGHTS = 3;
int heights[MAX_HEIGHTS] = {5, 7, 8};
```

- Unassigned elements are initialized to 0

```c
int heights[MAX_HEIGHTS] = {3}; // heights[1] == 0
  // heights[2] == 0
```
Initialization at Declaration

- Providing too many initial values is a compile-time error.

// ERROR!
int heights[MAX_HEIGHTS] = {3, 7, 8, 9};

Properties of Array Elements

- Once you specify an array element, it can be used just like any other variable of the specified data type:

int scores[MAX_SCORES];

scores[0] = 90;
total = total + scores[i];
cout << "Your score is " << scores[38] << endl;
Swap(scores[3], scores[10]);
Properties of Array Elements

void Swap(int& x, int& y) {
    int temp;
    temp = x;
    x = y;
    y = temp;
}

Arrays As Parameters

- Recall that entire arrays can be passed as parameters.
- Arrays are always pass by reference
  - Can use the const keyword to enforce input only parameter
- Array name without brackets is used as the actual parameter (argument)
- Formal parameter uses brackets to specify passing an array
Arrays As Parameters Example

```cpp
const string INIT_STRING = "";
void main() {
    string names[MAX_NAMES];
    InitStringArray(names, MAX_NAMES, INIT_STRING);
}

// Initialize an array of strings to the specified value
void InitStringArray(string array[], int size,
                      const string& value)
{
    for (int i = 0; i < size; i++) {
        array[i] = value;
    }
}
```

Arrays As Parameters Example 2

```cpp
void AddIntArrays(const int x[], const int y[],
                  int x[], int usage);
void PrintIntArray(ostream& Out, const int x[], int usage);
void main() {
    int x[MAX_ELEM], y[MAX_ELEM];
    int i = 0;
    // read until input failure or max elements
    cin >> x[i] >> y[i];
    while (cin && i < MAX_ELEM) {
        i++;
        if (i < MAX_ELEM) {
            cin >> x[i] >> y[i];
        }
    }
    AddIntArrays(x, y, x, i);
    PrintIntArray(cout, x, i);
}
```
Arrays As Parameters Example 2

```cpp
// Add two arrays together and store in third parameter
void AddIntArrays(const int x[], const int y[],
                  int z[], int usage)
{
    for (int i = 0; i < usage; i++) {
        z[i] = x[i] + y[i];
    }
}

// Print an integer array
void PrintIntArray(ostream& Out, const int x[], int usage) {
    for (int i = 0; i < usage; i++) {
        Out << x[i] << " ";
    }
    Out << endl;
}
```

Parallel Arrays

- Sometimes you want to associate different types of data to one logical item
- Example
  - Automobile parts in a store
    - Price and name
- But, arrays can only store one type of data
- Use multiple arrays
  - The index refers to the same logical item in each array
### Parallel Arrays Example

```c
const int MAX_ITEMS = 100;
void main() {
    string ItemName[MAX_ITEMS];
    double ItemPrice[MAX_ITEMS];
    int item = 0;
    // Make sure to initialize your arrays (code not included)
    getline(cin, ItemName[item], '\t');
    cin >> ItemPrice[item];
    while (cin && item < MAX_ITEMS) {
        item++;
        if (item < MAX_ITEMS) {
            getline(cin, ItemName[item], '\t');
            cin >> ItemPrice[item];
        }
    }
}
```

### Parallel Arrays Example

<table>
<thead>
<tr>
<th>Item (Index)</th>
<th>ItemName</th>
<th>ItemPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Shocks&quot;</td>
<td>59.99</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Brake Pads&quot;</td>
<td>79.95</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Wipers&quot;</td>
<td>2.38</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Parallel Arrays

- To access a logical item
  - Access each parallel array with the same index
  - Index identifies the logical item of interest
- Write functions to encapsulate parallel operations
  - Initialization
  - Printing
  - Access

Multidimensional Arrays

- Arrays are often used to store data for two or more dimension
  - Matrices
  - Images
  - Graphs
  - etc.
Multidimensional Arrays Example

```c
const int MAX_ROWS = 5;
const int MAX_COLS = 10;
void InitMatrix(int matrix[][MAX_COLS], int rows,
    int cols, int value);

void main() {
    int matrix[MAX_ROWS][MAX_COLS];

    InitMatrix(matrix, MAX_ROWS, MAX_COLS, 0);

    // Access fifth row, third column
    matrix[4][2] = 10;
}
```
Multidimensional Arrays Example

0 1 2 ... 9
0
1
2
3
4