Arrays

- Basic use of 1-Dim arrays
- Passing as parameters
Allen Telescope Array (ATA)
Wind Turbines
The Power of Wind
The Power of Michigan
int x0, x1, x2, ..., x99;

cin >> x0;
cin >> x1;
cin >> x2;
...
cin >> x99;
Why Arrays?

```cpp
int x[100];

cin >> x[0];
cin >> x[1];
cin >> x[2];
...
cin >> x[99];
```
Why Arrays?

```cpp
int x[100];

for (int i=0; i<100; i++)
{
    cin >> x[i];
}
```
Why Arrays?

- Read a sequence of values and
- Print out the sequence
- Mark the largest value

32
54
67.5
29
115 ➔ largest value
44.5
100
65
Issues

- Don't know how many values

- Largest could be
  - first
  - last
  - in the middle -- somewhere
Options

- declare more variables than you think you need
  - not practical
  - lots of duplicated code
Best choice

- use an array
- excellent choice for storing a sequence of values
double data [ 10 ];
double data [ 10 ];
Declaration of Array

double data [ 10 ];

basetype

size
(# of elements)
double data [ 10 ];

- basetype
- identifier
- size (# of elements)
double data[10];
double data[10];

10 consecutive memory locations
Declaration of Array

double data[10];

10 consecutive memory locations

index is offset of base address
double data[10];

10 consecutive memory locations

Index is offset of base address
double data[10];
data[3] = 29;
double data[10];
data[0] = 32;
data[1] = 54;
data[2] = 67.5;
data[3] = 29;
data[4] = 35;
data[5] = 80;
data[6] = 115;
data[7] = 44.5;
data[8] = 100;
data[9] = 65;
const int SIZE = 10;
double data[SIZE];
Won't Work

```c
int SIZE = 10;
double data[SIZE];  // this won't work
```

can only use a constant
cannot use a variable
can only use a constant
cannot use a variable

cout << “Enter a size”;
cin >> size;

double data [size]; // won’t work
double data[10] = {32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65};
double data[ ] = {32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65};
double data[10] = {32, 54, 67.5, 29};
double data[10] = {32, 54, 67.5, 29};

Some compilers will put 0’s
Others won’t put in the 0’s
double data[10];

int count = 0;
int temp;

cin >> temp;  // 32
while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp;
}

double data[10];

int count = 0;
int temp;

cin >> temp;

while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp;
}

Read values in from keyboard
double data[10];

int count = 0;
int temp;

cin >> temp;

while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp;
}

32
double data[10];

int count = 0;
int temp;

cin >> temp;
while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp; // 54
}

32 1 54
double data[10];

int count = 0;
int temp;
cin >> temp;
while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp;
}

double data[10];

int count = 0;
int temp;

cin >> temp;

while (temp != 0) {
    data[count] = temp;
    count++;
    cin >> temp;
}

// Partially filled array
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max
Find Max

double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
  if (data[i] > max) {
    index = i;
    max = data[i];
  }
}

// Example data:
<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

// Example count:
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

// Example index:
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max

32 54 67.5 29 35 80 115 44.5 8 32
0 54 67.5 29 35 80 115 44.5 8 32
[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

data count max index i
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max
Find Max

double data[10];
double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

32 54 67.5 29 35 80 115 44.5 8 32 0 1
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max

32
54
67.5
29
35
80
115
44.5
8
32
1

data
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max: fast forward
double data[10];

double max = data[0];
int index = 0;

for (int i = 0; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max
double data[10];

double max = data[0];
int index = 0;

for (int i = 1; i < count; i++) {
    if (data[i] > max) {
        index = i;
        max = data[i];
    }
}

Find Max

<table>
<thead>
<tr>
<th>i</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>67.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[0]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[1]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[2]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[3]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[4]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[5]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[6]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[7]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[8]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data[9]</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value"
    }
    cout << endl;
}

Print Results

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>54</td>
<td>67.5</td>
<td>29</td>
<td>35</td>
<td>80</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 54 67.5 29 35 80 115 44.5 8 115 6 0

count max index
double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value"
    }
    cout << endl;
}

Print Results

double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value";
    }
    cout << endl;
}

32
54
67.5
29
35
80
115
44.5
8
115
6
0

data

0
1
2
3
4
5
6
7
8
9

count

max

index

i
double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value"
    }
    cout << endl;
}

32
54
67.5
29
35
80
115
44.5
8
115
6
0
double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value"
    }
    cout << endl;
}

Print Results

<table>
<thead>
<tr>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>67.5</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>115</td>
</tr>
<tr>
<td>44.5</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>115</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
double data[10];

for (int i = 0; i < count; i++) {
    cout << data[i];
    if (i == index) {
        cout << " <-- largest value";
    }
    cout << endl;
}

Print Results:  etc

data

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>54</td>
<td>67.5</td>
<td>29</td>
<td>35</td>
<td>80</td>
<td>115</td>
<td>44.5</td>
<td>8</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
find average of an array where values are read from file

../sampleCode/avg_of_array.cpp
Errors:

Maybe engineering is the pursuit of an unattainable perfection.

Maybe it's impossible to create something bug-free.

Maybe the tyranny of Murphy is the penalty for hubris.

But I just can't shake the feeling.

With all those supplies, I could have caught that Roadrunner.
Range Errors

- C++ does NOT check array indices for validity

- Range errors can:
  - overwrite data
C++ does NOT check array indices for validity

Range errors can:
- overwrite data

```c
data[10] = 42;
```
const int SIZE_DATA = 10;
int data[SIZE_DATA], count;

ifstream inFile;

for (count=0; count<=SIZE_DATA; count++)
inFile >> scores[count];
OFF-BY-ONE Error

const int SIZE_DATA = 10;
int data[SIZE_DATA], count;

ifstream inFile;

for (count=0; count<=SIZE_DATA; count++)
inFile >> scores[count];
const int SIZE = 5;
int arr1[SIZE], arr2[SIZE];

for (int i=0; i<SIZE; i++)
{
    cin >> arr1[i]; // compile error
}

for (int i=0; i<SIZE; i++)
{
    cin >> arr1[i];
}
Assignment

const int SIZE = 5;

int arr1[SIZE], arr2[SIZE];

arr1 = arr2;

for (int i=0; i<SIZE; i++)
{
    arr1[i] = arr2[i];
}
const int SIZE = 5;
int arr1[SIZE], arr2[SIZE];

arr1 = arr2 + 6;     // compile error

for (int i=0; i<SIZE; i++)
{
    arr1[i] = arr2[i] + 6;
}
const int SIZE = 5;
int arr1[SIZE], arr2[SIZE];

if (arr1 < arr2) // compile error
{
    for (int i=0; i<SIZE; i++)
    {
        if (arr1[i] < arr2[i]) {
            ...
        }
    }
}
const int SIZE = 5;
int arr1[SIZE], arr2[SIZE];

void printArray(int arr[], int SIZE);

pass by reference
Return from a function

```c
const int SIZE = 5;
int arr1[SIZE], arr2[SIZE];
int [ ] foo(int arr[ ], int SIZE);
// compile error

void foo(int arr[ ], int SIZE);

pass by reference
no need to "return" the array
```
C++ arrays are:
- always passed by reference
- saves memory space
  - array is not copied
- saves processing time
  - array elements are not copied
strings can be treated as arrays

string str = "hello";
str[0] = 'j';
cout << str << endl;
strings can be treated as arrays

```cpp
string str = "hello";
str[0] = 'j';
cout << str << endl;
```

prints:

```
hello  \rightarrow  jello
```

```cpp
j
```
strings can be treated as arrays

string str = "hello";
str[0] = 'y';
str[5] = 'w';  // shouldn't do
              // past end
cout << str << endl;
strings can be treated as arrays

string str = "hello";
str[0] = 'y';
str[5] = 'w';  // shouldn't do
              // past end
cout << str << endl;

hello \[y\\]w \[yellow\]  
but will cause issues
strings can be treated as arrays

string str = "hello";
str[1] = 'o';
str[0] = 'p';

str = 'a' + str;
cout << str << endl;
strings can be treated as arrays

```cpp
string str = "hello";
str[1] = 'o';
str[0] = 'p';

str = 'a' + str;
cout << str << endl;
```

```
// ahello  ➔  apollo
```
Example Call (assume main is caller):
readScoreList ( data, n );

Possible Corresponding Prototypes:
void readList(int data[], int size);
void readList(int data[SIZE], int size);
Want: "pass by value" for arrays

void printResults (int data[ ]);  
  allows function to alter data

Remember all arrays are passed by reference
Passing by \texttt{const} reference

- Want: "pass by value" for arrays

- \texttt{void printResults (int data[ ])};
  - allows function to alter data

- \texttt{void printResults(\texttt{const} int data[ ])};
Zero out all elements of array

- Declare array with 100 elements, and initialize all elements to zero
Zero out all elements of array

- Declare array with 100 elements, and initialize all elements to zero

```cpp
int numbers[100];
for (int i = 0; i < 100; i++) {
    numbers[i] = 0;
}
```
Arrays are very useful for set of like data
- 0-based
- Don't write past end of array (range error)
- ALL elements must be the same datatype
Arrays are always passed by reference

- Can be modified in called function

```
//prototype
void readScoreList(int scores[]);

//call
readScoreList (scores);
```
Arrays *may* be passed by *const ref* – very useful
- Cannot be modified in called function

```c++
// prototype
void calcMeanScore(const int scores[]);

// call
calcMeanScore(scores);
```
Additional Information
The name of an array
- refers to the entire data structure
- memory address of 1st array element

```c
```
The name of an array
- refers to the entire data structure
- memory address of 1st array element

```c
```
The name of an array
- is a *pointer constant*

<table>
<thead>
<tr>
<th>data type</th>
<th>const int *</th>
</tr>
</thead>
</table>

address operator

```
score == & score[0]
```
The name of an array
  - is a *pointer constant*

```c
int score = & score[0];
```

- data type `const int *`
- address operator
- address of 1st elem of array