Arrays are Groups of Values Named Using an Index

In MP4,
- you have to handle four regions
- with sizes given as \( r_1, r_2, r_3, \) and \( r_4 \).

In some larger nonograms, a row or column may have 20 regions.
So for MP6 ...

... let’s talk about arrays!
(Instead of \( r_1, r_2, \ldots, r_{20} \).)

Arrays are Contiguous in Memory; Indices Use Brackets

In **C**, one declares an array of 20 **ints** as

```c
int region[20];
```

The compiler **allocates memory**
- to hold 20 **ints**
- called **region[0]** through **region[19]**,
- as illustrated to the right.

Pointer Arithmetic Moves Among Array Elements

```c
int region[20];
```

The expression **region**
- has type **int**
- and points to **region[0]**.

The expression **region + N**
- points to **region[N]**
- (called **pointer arithmetic**).

```c
region + 19
```

```c
region
```
Brackets are Shorthand for Add and Dereference

```c
int region[20];
```

Thus,

- `region[N]` and
- `*(region + N)`
- are equivalent in C.

Use Brackets for Reading and Writing Array Elements

```c
int region[20];
int a;
```

Thus

- `a = region[19];`
- reads the value at address `region + 19`.

And

- `region[19] = a;`
- stores bits to address `region + 19`.

Pointer Arithmetic Generally Involves Multiplication

```c
int region[20];
```

Say that `region` is address `0x12345000`.

What is `region + 5`?  `0x12345005`? Not necessarily.

The answer depends on

- the size of an `int` and
- the addressability of memory.

The amount added is the number of addresses required for 5 ints.

Pass Array Arguments as Pointers

Let’s do an example. Our task:

- write a subroutine
- to find the minimum value
- in an array of `int32_t`.

How can we pass the array to the subroutine?

Copy it onto the stack? Expensive!

Instead, pass a pointer to the first element!
An Address Does Not Define an Array Length

```c
int32_t min_value = same as int32_t const* values
    (int32_t const values[]);
```

Look good?

How can `min_value` know the array size?

As shown, it cannot.

So ...?

Add a second parameter for the length.

---

Finding Minimum Value with a C Function

```c
int32_t min_value
    (int32_t const values[], int32_t n_values)
{
    int32_t min = values[0];
    int32_t check;
    for (check=1; n_values > check; check++) {
        if (min > values[check]) {
            min = values[check];
        }
    }
    return min;
}
```

Assume first value is smallest.

In loop body, check goes from 1 to `n_values - 1`.

---

Using Our Minimum Value Function

How do we use the function?

```c
int32_t my_nums[4] = [93, 100, 79, 42];
int32_t least;
least = min_value (my_nums, 4);
```

Initializes array to values shown.

Holds 42 after assignment.

Length of `my_nums`
Note that values (type int const*) takes only one memory location.

Strings Can Be Stored in Arrays of chars

Strings are like (and can be stored in) arrays of chars. Unfortunately, we must choose a size for an array.

```c
char name[20];
printf("Hi, what is your name?\n");
if (1 == scanf("%s", name)) {
    printf("Hello, %s!\n", name);
}
```

Array Bounds are Not Checked in C

```c
char name[20];
printf("Hi, what is your name?\n");
if (1 == scanf("%s", name)) {
    printf("Hello, %s!\n", name);
}
```
Buffer Overrun Attacks Used to Dominate Vulnerabilities

This type of attack is:
- a buffer overrun attack,
- the dominant software vulnerability
- for many years.

Microsoft went through 50 million lines of code to try to eliminate them.

Recent OS changes have also helped:
- reduced ability to execute code on stack, and
- randomization of code location.

Use Field Width to Make scanf Safe

```c
char name[20];
printf("Hi, what is your name?\n");
if (1 == scanf("%19s", name)) {
    printf("Hello, %s!\n", name);
}
```

Use field width 19* to limit input to 19 characters (need 1 char for NUL).

*Solutions (such as this one) that require humans to maintain them are error-prone.

Impromptu Survey on Phone Books

How many of you have ...

- ...used a phone book?
- ...seen a phone book?
- ...heard of phone books?

Just wondered.

How Do We Search When Values are Sorted?

Imagine that you have
- an array of integers
- sorted in numerically increasing order.

**How do you check whether**
- a particular integer
- appears in the array?

Let’s write a C function and return either
- the index of the desired value, or
- -1 if the value is not in the array.
Parameters and Local Variables for Binary Search

```c
int32_t binary_search(int32_t array[], int32_t len, int32_t value)
{
    int32_t low = 0;
    int32_t high = len - 1;
    int32_t mid;
    // initialize search bounds [low,high] to [0,len - 1]

    // a sorted array of integers
    // length of array
    // number to find in array
```

Main Iteration: Look Once, Then Adjust Bounds

```c
while (high >= low) {
    mid = low + (high - low) / 2;
    // look at one value
    // and adjust bounds
    }
return -1;
```

Bugs Can Be Subtle and Hide for Decades

Are these two expressions equivalent?

- \( \text{low} + \frac{(\text{high} - \text{low})}{2} \)
- \( \frac{\text{low} + \text{high}}{2} \)

No ...
- but for 20+ years,
- library code for binary search
- used the second expression.

Sums Can Overflow, Producing Negative Array Indices

Consider the following:
- \( 0 \leq \text{low} < 2^{31} \) and \( 0 \leq \text{high} < 2^{31} \), and
- \( \text{low} \leq \text{high} \), so \( 0 \leq \frac{\text{high} - \text{low}}{2^{31}} \)

What about \( \text{low} + \text{high} \)?

Overflow can produce \( \text{mid} < 0 \)!

*Technically, one needs to couple this argument with a proof by induction.*
Loop Body: Find Value, or Adjust One Bound

```java
while (high >= low) {
    mid = low + (high - low) / 2;
    if (value == array[mid]) {
        return mid; // Found!
    }
    if (value < array[mid]) {
        high = mid - 1;
    } else { // value > array[mid]
        low = mid + 1;
    }
}
```

Can Use Arrays of Arrays for Multidimensional Data

One can create arrays of arrays. For example,

```java
int32_t blk[3][5];
```

allocates an array of 3 arrays of 5 `int32_t`

The expressions `blk` and `blk[0]`

- have the same value
- but `blk[0]` has type `int32_t*`
- while `blk` has type `int32_t (*)[5]`
  - (pointer to an array of 5 `int32_t`

Remember: Pointer Arithmetic Depends on Size

What happens when we add 1...

- `blk` and `blk[0]`
- `blk[0] + 1`
- ... to `blk[0]`?
  - (same as before)
- ... to `blk`?
  - (points to next array of 5)

Often Need to Map Multidimensional Data by Hand

You will use multidimensional arrays in MP6.

In later MPs,
- array dimensions are not known in advance,
- so your program must
- perform the mapping into 1D.

For example, in MP8,
- an image is `height × width` pixels,
- but `height` and `width` are variables.
Multiply “Larger” Dimension by Size of Smaller

We choose which dimension to map first.
Here, row index $y$ is multiplied by $\text{width}$:

$$(x, y) \rightarrow y \times \text{width} + x$$

Let's Have Some Fun

Time for some fun...

Let's play cards!

You can teach me...

...how to sort my hand.

Parameters and Local Variables for Insertion Sort

```c
void insertion_sort(int32_t values[], int32_t num_vals) {
  int32_t sorted, current, index;
  number sorted (after loop body)
  card to insert
  position to insert
}
```

Main Loop: Sort One Card at a Time Until All are Sorted

```c
for (sorted = 2; num_vals >= sorted; sorted++) {
  // insert one more card
  Sort one card at a time.
}
```
After inner loop, index is correct position for new card.

New card is at position $\text{sorted} - 1$.

```c
current = values[\text{sorted} - 1];
// find place to insert new card
values[index] = current;
```

Position Check Simply Compares Card Values

```c
if (current \geq values[index - 1]) {
    break;
}
```

If the new card's value is at least as great as the card in the previous position, stop searching.
Let's Try It! Start with Some Arguments

void insertion_sort(int32_t values[], int32_t num_vals)

Our Local Variables are Not Initialized

int32_t sorted, current, index;

What's in these variables?

Bits!

First Loop Iteration (First Card is Sorted Already)

for (sorted = 2; num_vals >= sorted; sorted++) {

Time to Insert the 4 at the Right Position

current = values[sorted - 1];

Huh? What's in values[1]?

Still 4, but, logically, it's 'empty.'
Find the Right Place for the 4

for (index = sorted - 1; 0 < index; index--) {
  if (current >= values[index - 1]) {
    break;
  }
}

Copy 12 from Position 0 to Position 1

values[index] = values[index - 1];

Update and Test Again

for (index = sorted - 1; 0 < index; index--) {
  if (0 < index) {  // No. This loop is done.
    break;
  }
}

Compare 4 with 12

if (current >= values[index - 1]) {
  break;
}

No.
Place New Card in “Blank” Position (index)

\[ \text{values[index]} = \text{current}; \]

<table>
<thead>
<tr>
<th>values</th>
<th>4</th>
<th>12</th>
<th>9</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

num_vals is 5

Update and Test Again (First 2 Cards Now Sorted)

for (sorted = 2; num_vals >= sorted; sorted++) {
   Is num_vals >= sorted? Yes.

<table>
<thead>
<tr>
<th>values</th>
<th>4</th>
<th>12</th>
<th>9</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time to Insert the 9 at the Right Position

\[ \text{current} = \text{values[sorted - 1]}; \]

<table>
<thead>
<tr>
<th>values</th>
<th>4</th>
<th>12</th>
<th></th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Find the Right Place for the 9

for (index = sorted - 1; 0 < index; index--) {
   Is 0 < index? Yes!

<table>
<thead>
<tr>
<th>values</th>
<th>4</th>
<th>12</th>
<th></th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>current</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compare 9 with 12

if (current >= values[index – 1]) {
    break;  // Is 9 >= 12? No.
}

values

0 1 2 3 4
4 12 1 8

9

current

index sorted

num_vals is 5

Copy 12 from Position 1 to Position 2

values

0 1 2 3 4
4 12 1 8

9

current

index sorted

num_vals is 5

Update and Test Again

for (index = sorted – 1; 0 < index; index--) {
    // Is 0 < index? Yes.
}

values

0 1 2 3 4
4 12 1 8

9

current

index sorted

num_vals is 5

Compare 9 with 4

if (current >= values[index – 1]) {
    break;  // Is 9 >= 4? Yes, so break.
}

values

0 1 2 3 4
4 12 1 8

9

current

index

sorted

num_vals is 5
Place New Card in “Blank” Position (index)

values[index] = current;

values

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
4 & 9 & 12 & 1 & 8 \\
\end{array}
\]

current

index

sorted

num_vals is 5

Update and Test Again (First 3 Cards Now Sorted)

for (sorted = 2; num_vals >= sorted; sorted++) {
Is num_vals >= sorted? Yes.

values

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
4 & 9 & 12 & 3 & 8 \\
\end{array}
\]

current

index

sorted

num_vals is 5

Time to Insert the 1 at the Right Position

current = values[sorted - 1];

values

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
4 & 9 & 12 & 1 & 8 \\
\end{array}
\]

current

index

sorted

num_vals is 5

Find the Right Place for the 1

for (index = sorted - 1; 0 < index; index--) {
Is 0 < index? Yes!

values

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 \\
4 & 9 & 12 & 3 & 8 \\
\end{array}
\]

current

index

sorted

num_vals is 5
Compare 1 with 12

\[
\text{if (current >= values[index - 1]) { break; \quad \text{Is 1 >= 12? No.}}}
\]

values
\[
\begin{array}{cccc}
0 & 4 & 9 & 12 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

current
\[
1
\]  
num_vals is 5

Copy 12 from Position 2 to Position 3

\[
\text{values[index] = values[index - 1];}
\]

values
\[
\begin{array}{cccc}
0 & 4 & 9 & 12 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

current
\[
1
\]  
num_vals is 5

Update and Test Again

\[
\text{for (index = sorted - 1; 0 < index; index--) { \quad \text{Is 0 < index? Yes.}}}
\]

values
\[
\begin{array}{cccc}
0 & 4 & 9 & 12 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

current
\[
1
\]  
num_vals is 5

Compare 1 with 9

\[
\text{if (current >= values[index - 1]) { break; \quad \text{Is 1 >= 9? No.}}}
\]

values
\[
\begin{array}{cccc}
0 & 4 & 9 & 12 \\
1 & 2 & 3 & 4 \\
\end{array}
\]

current
\[
1
\]  
num_vals is 5
Copy 9 from Position 1 to Position 2

values[index] = values[index - 1];

Update and Test Again

for (index = sorted - 1; 0 < index; index--)

Is 0 < index? Yes.

values[index] = values[index - 1];

Copy 4 from Position 0 to Position 1

values[index] = values[index - 1];
Update and Test Again

for (index = sorted - 1; 0 < index; index--) {
Is 0 < index? No. This loop is done.
}

Place New Card in “Blank” Position (index)

values[index] = current;

Update and Test Again (First 4 Cards Now Sorted)

for (sorted = 2; num_vals >= sorted; sorted++) {
Is num_vals >= sorted? Yes.
}

Time to Insert the 8 at the Right Position

current = values[sorted - 1];
Find the Right Place for the 8

for (index = sorted - 1; 0 < index; index--) {
  Is 0 < index? Yes!
}

values

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>12</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Copy 12 from Position 3 to Position 4

values[index] = values[index - 1];

values

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>12</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Compare 8 with 12

if (current >= values[index - 1]) {
  break; Is 8 >= 12? No.
}

values

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>12</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Update and Test Again

for (index = sorted - 1; 0 < index; index--) {
  Is 0 < index? Yes.
}

values

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>9</th>
<th>12</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Compare 8 with 9

```c
if (current >= values[index - 1]) {
    break;  // 8 >= 9?  No.
}
```

values: 1 4 9 12

<table>
<thead>
<tr>
<th>current</th>
<th>sorted is 5</th>
<th>num_vals is 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copy 9 from Position 2 to Position 3

```c```
values[index] = values[index - 1];
```

<table>
<thead>
<tr>
<th>current</th>
<th>sorted is 5</th>
<th>num_vals is 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Update and Test Again

```c```
for (index = sorted - 1; 0 < index; index--) {
    // 0 < index? Yes.
}
```

<table>
<thead>
<tr>
<th>current</th>
<th>sorted is 5</th>
<th>num_vals is 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compare 8 with 4

```c```
if (current >= values[index - 1]) {
    break;  // 8 >= 4?  Yes, so break.
}
```

<table>
<thead>
<tr>
<th>current</th>
<th>sorted is 5</th>
<th>num_vals is 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Place New Card in “Blank” Position (**index**)

```
values[index] = current;
```

Current: sorted is 5  num_vals is 5

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Update and Test Again (First 5 Cards Now Sorted)

```
for (sorted = 2; num_vals >= sorted; 
    sorted++) {
    Is num_vals >= sorted? No. We're done.
```

Current: sorted is 5  num_vals is 5

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time for Another Think-Pair-Share

As before, let’s do a group exercise in lecture.

The process:
1. I give you a problem.
2. You form groups of 3-4 people.
3. Talk about ways to solve the problem.
4. Once enough of the groups have finished, one group volunteers to share their answer.
5. We go over the group’s answer together.

The Task: Calculate Average of a List of Numbers

The task...
- Allow a user to enter up to 5 numbers (not -1).
- End the list by typing -1 (not include in list).
- Find and print average of user’s numbers.

One half of the class writes **main**.
The other half writes **calc_avg**.
Let’s use

```
int32_t calc_avg
    (int32_t const array[], int32_t len);
```