Simple Arrays

Steven R. Bagley

Recap

- Program = Algorithms + Data
- Data stored in variables
- Have type and name
- Hold a single value…

What happens when we want to store more than one value?

Temperatures

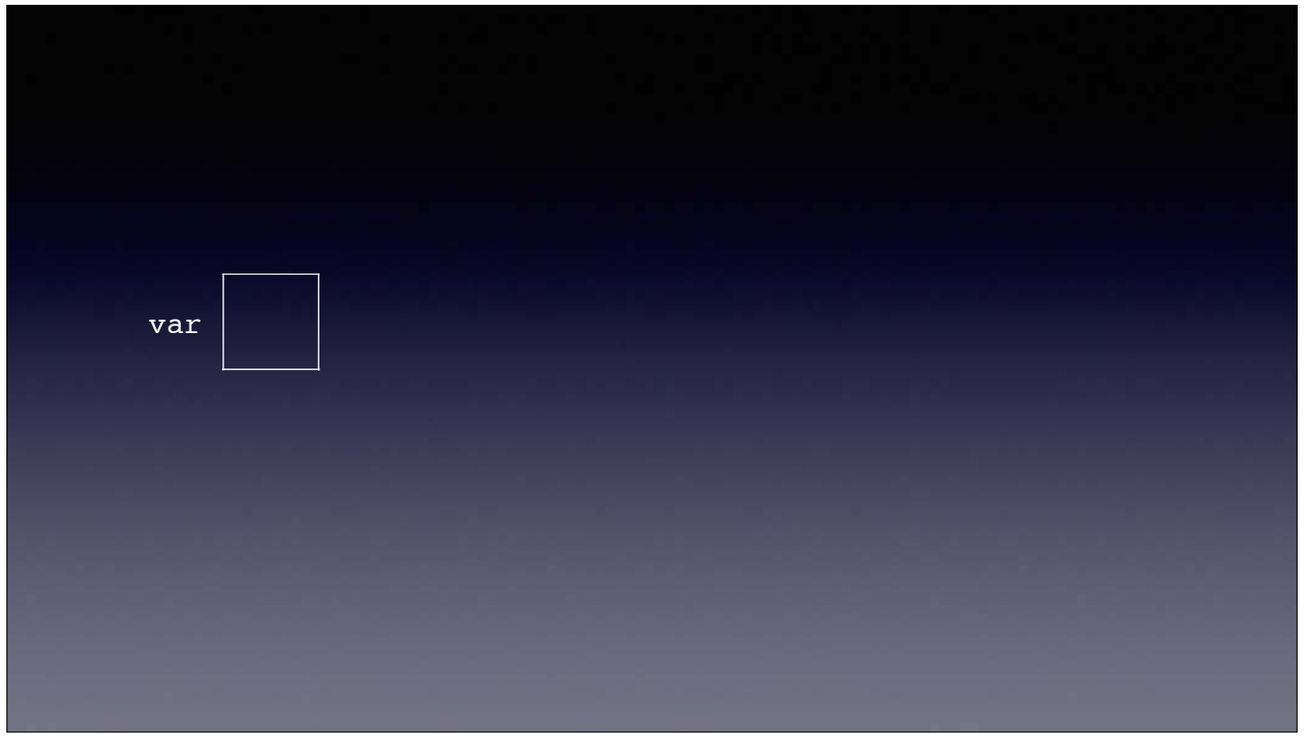
- How would we store the temperatures for a week?
- Multiple variables? — one for each day of the week
  `double monTemp, tuesTemp, wedTemp, thursTemp, … ;`
- Works, but cumbersome…
- Data related, but variables aren't…
- Be nicer if we can store all the values together

Have to manually add up all seven variables to take an average (can't use a loop). Fine for seven, but a pain for more. Can't at runtime decide which to add up (without lots of ifs)

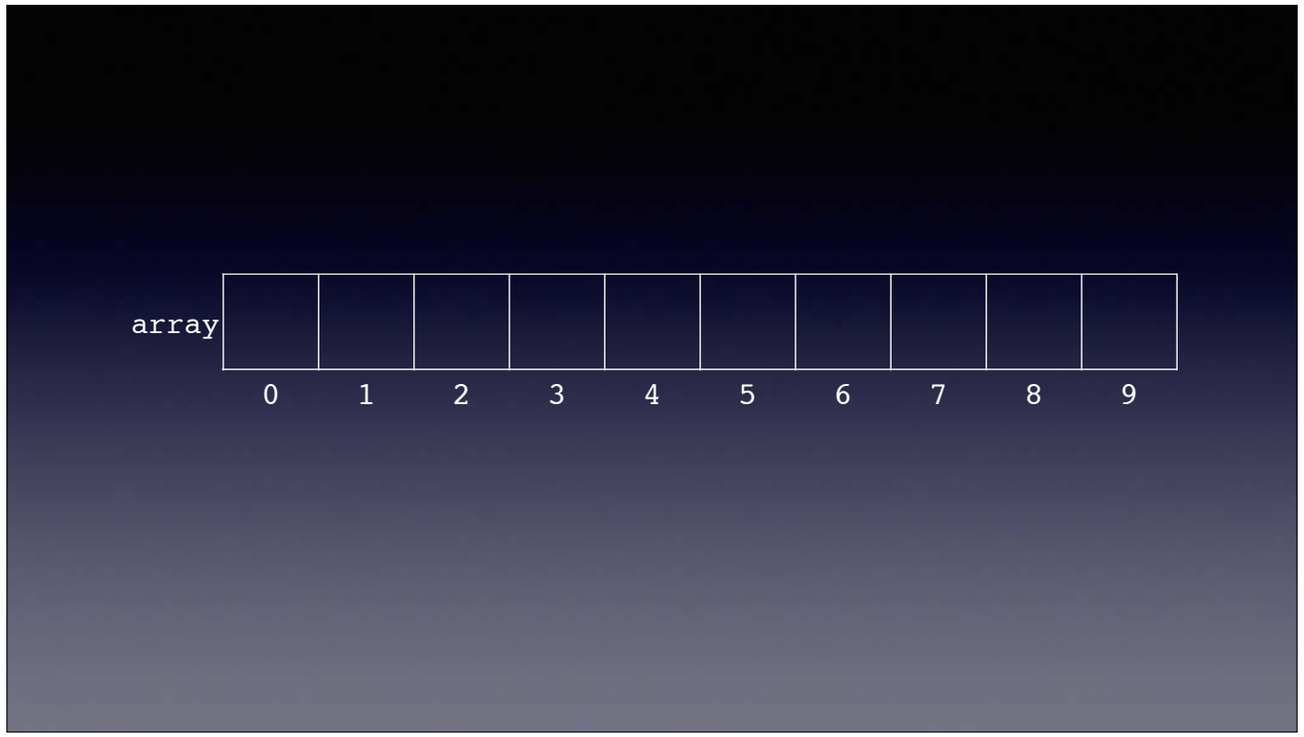
The Array

- C provides such a facility — the Array
- Data structure that stores multiple values of the same type
- Has a fixed size specified at compile-time
- Individual values accessed by a numeric index (from 0 to size)
- Again has a name
- Constant time access for each item

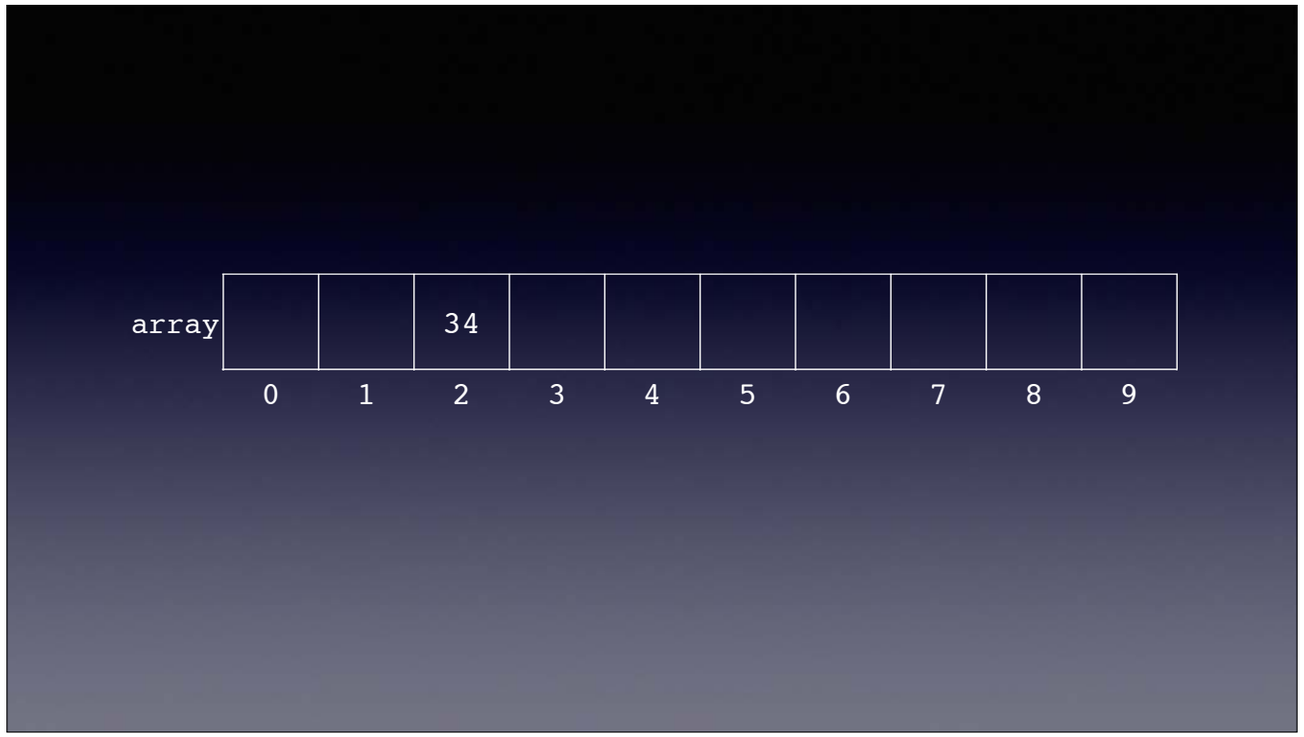
Will see later how to create them dynamically
Values have an order...

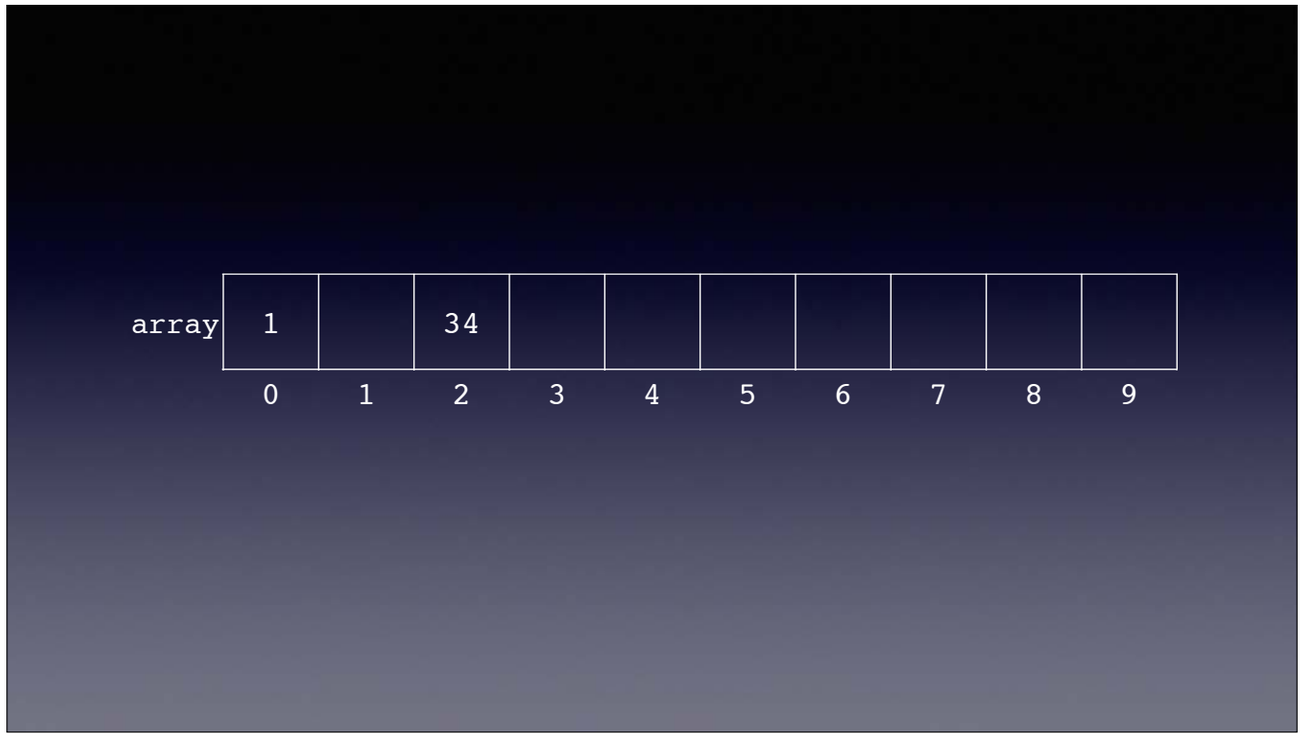


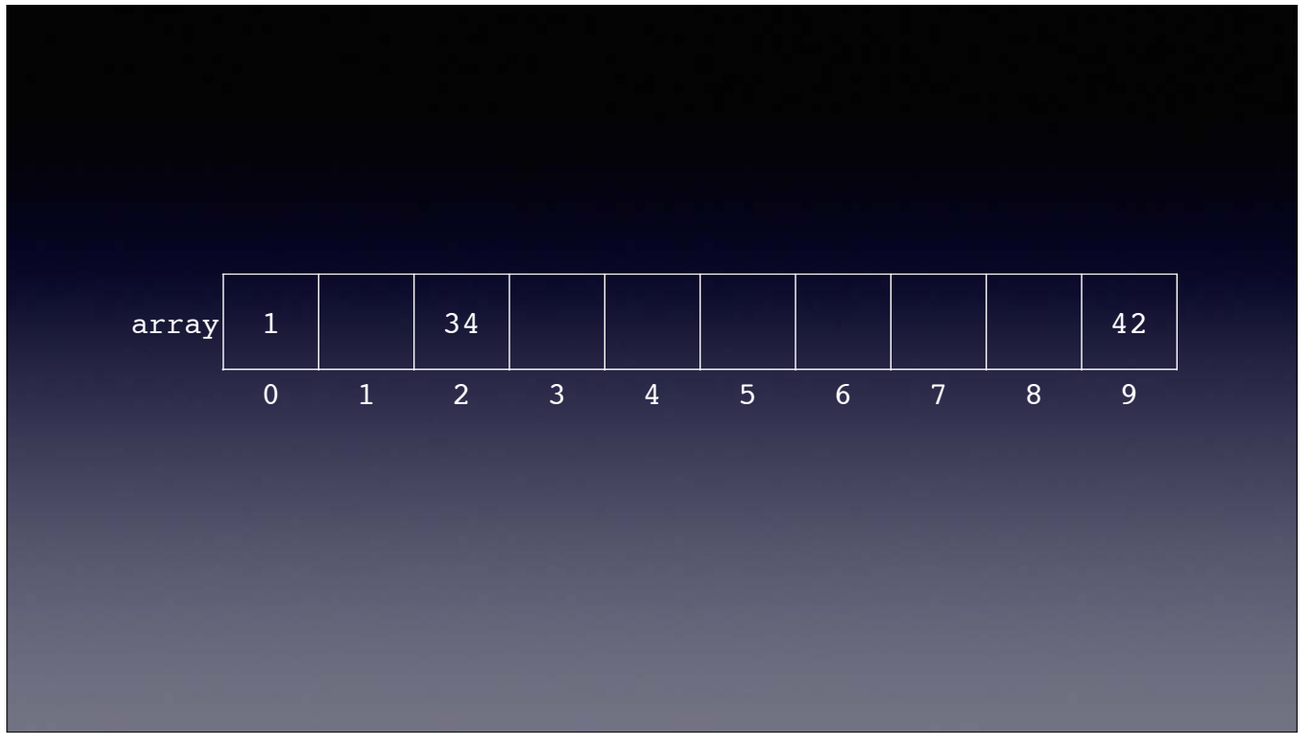
just like variables but accessed by name and index, rather than just name
Unknown what the default value will be



just like variables but accessed by name and index, rather than just name
Unknown what the default value will be


array
34
0
1
2
3
4
5
6
7
8
9


array
1
34
0
1
2
3
4
5
6
7
8
9


array
1
34
42
0
1
2
3
4
5
6
7
8
9

Creating and using an Array

- C uses `[]` operator to signify an array
- Created like a normal variable with the `[]` after the name
  `int values[42];` – Create an array of 42 `int`s
- Array called `values`
- Size placed inside square brackets
- This creates an array with 42 items, numbered from 0 to 41

```
int    daysOfMonths[12];  /* Array of 12 ints */

double weeksTemps[7];     /* Array of 7 doubles */

char   text[256];         /* Array of 256 char */

int   *pInts[10];         /* Array of 10 pointers to ints */
```

Some example arrays

Accessing Array Items

- Each array item has a numeric index…
- To access an item, index is put in square brackets after array name

```
values[0] = 42;
printf(“%d”, values[42]);
```

- Can be used just like a variable, including getting a pointer to it
- Can use literals, variables, functions as an index, providing its an `int`
- Very powerful – especially if the variable is in a loop

Pointer useful for scanf(), e.g. scanf(“%d”, &values[4])
Even another array look-up…

Accessing Array Items

- A word of warning…
- C is very liberal when it comes to arrays
- It performs no bounds-checking when you access an index
- Even if the index is negative…
- In fact, it only keeps track of where the array starts in memory
- Assumes you, the programmer, know what you are doing…

The next couple of lectures will show why this is valid, if bizarre!

Recap

- Data stored in variables or arrays
- Variables hold a single value
- Variables accessed by a name
- Arrays can store a number of values
- Arrays accessed by a name *and* a numeric index

Arrays

- Arrays can be created anywhere variables can
- Local arrays — local to a function
- Global Arrays — accessible everywhere
- Can be slightly more acceptable to have global arrays than global variables…

Go for some demos -- simple ones
modify temperature thing to read in values to an array

Initial Values

- Like variables, the initial value of an item in an array is unknown
- Assign useful values into them
- But can also create arrays with an initial set of values
- When creating the array we specify a list of values
- These are assigned *in order* to each item in the array
- List specified using `{ … }`

Use commas to separate items

```
int daysInMonth[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
printf("January has %d days\n", daysInMonth[0]);

double weeksTemp[7] = { 8.6, 6.4, 9.0, 8.2, 5.2, 4.9, 6.1 };
```

Creating an array with some initial values
Write a program to calculate the average temperature

Initial Values

- Raw values stored in your program file
- When a local array is created, memory is allocated for it
- Values copied in each time the function is run
- Even if they were changed the last time the function is run

Initial Values

- For global arrays, things are slightly different
- Values are stored in the program file
- When you access them you go directly to those values
- No need to copy them…
- Why global arrays are acceptable for data that is unlikely to change e.g. an array of number of days in a month

Initial Values

- Only possible to set an array like this when array is initially defined
- Compile error if you try to do it at any other point
- Note that you can leave out the size of the array, if you are specifying the initial values
- Compiler will work out the size of the array automatically

```
int daysInMonth[] = {31,28,31,30,31,30,31,31,30,31,30,31};
printf("January has %d days\n", daysInMonth[0]);

double weeksTemp[] = { 8.6, 6.4, 9.0, 8.2, 5.2, 4.9, 6.1 };
```

Creating an array with some initial values

Passing Arrays to Functions

- Can pass arrays into functions
- Does not make a copy of the array, rather it passes a pointer to it
- This means a function can alter the values…

Arrays and pointers are in extricably linked in C
look at main -- takes an array of char * and its size

Passing Arrays to Functions

- Declare the array as a parameter
- No need to declare the size, won't make any difference if you do
- Because arrays have no size in C
- So declare like so
  ```
  double AverageArray(double array[]);
  ```
- Can't return arrays (easily)

Except for multidimensional arrays, need to pass the size in for them to work

Passing Arrays to Functions

- How does the function know how big the array is?
- It doesn't, needs a mechanism to find out
- One method is as we have already seen with `main()` declared as `int main(int argc, char *argv[])`
- Takes an array of arguments (`argv`), and a count of how many there were (`argc`)
- Or have some value that means the end

Come across this again when doing IO
Demo command line arguments
Strings in C use a zero-byte to mean end of array

```
double average(double array[], int size)
{
    double sum = 0.0;
    int i;

    for(i = 0; i < size; i++)
    {
        sum += array[i];
    }

    return sum / size;
}

double weeksTemp[7] = { 8.6, 6.4, 9.0, 8.2, 5.2, 4.9, 6.1 };

double avgTemp = average(weeksTemp);
```

A function that can generate an average of any array of doubles.

```
void ClearArray(int array[], int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
        array[i] = 0;
    }
}

int daysInMonth[] = {31,28,31,30,31,30,31,31,30,31,30,31};
ClearArray(daysInMonth);
```

This function will clear an array of ints.
Note there are better ways of doing this

Copying Array

- No support to copy an array…

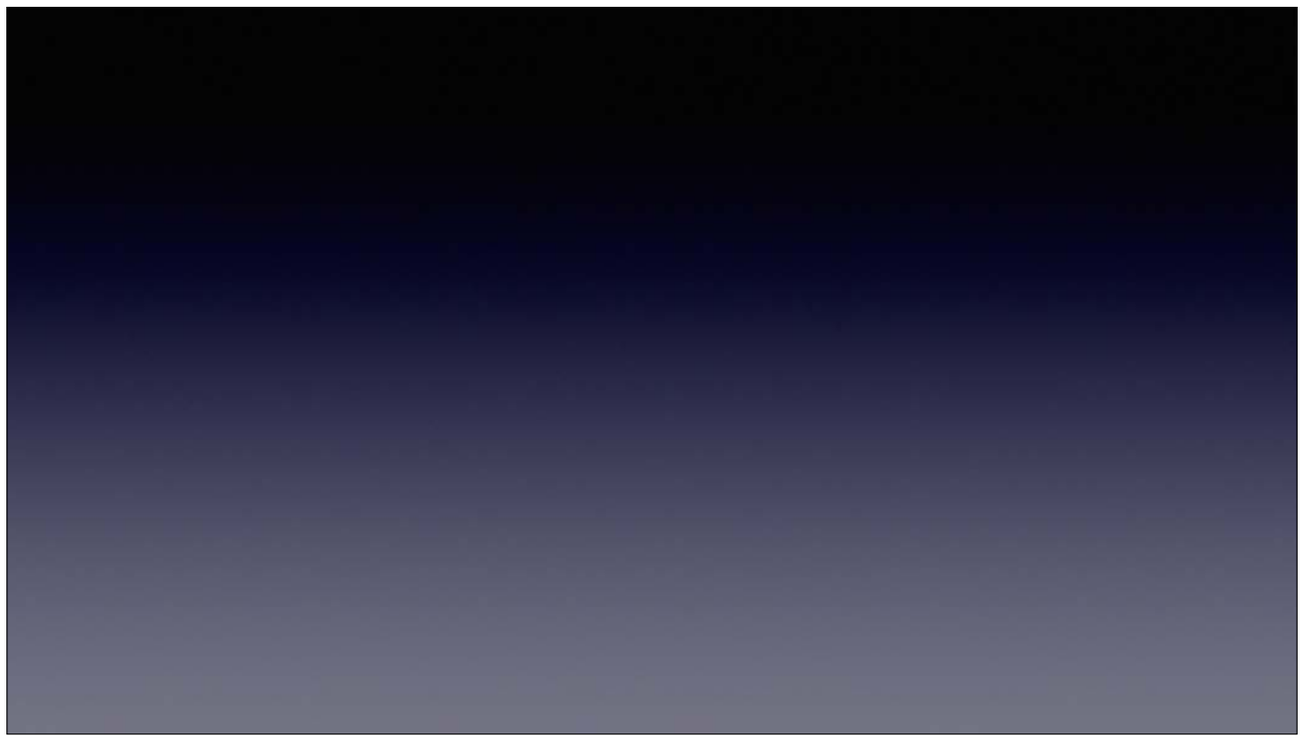
```
int a[42], b[36];
a = b; /* This won't work */
```

- Not possible for C to do so either since it doesn't keep track of how big they are
- Can only change the items in an array, not the array as a whole
- If you need to copy values from one array to another, then you have to do it manually, e.g. with a `for` loop

Strings or Character arrays

- In C, strings are just arrays of characters in memory
- Terminated by a null character (ASCII code 0, `'\0'`)
- Can create strings by creating an array of characters
  `char string[12];`
- Creates space for a string containing up to 11 characters and null character
- Always remember to leave space for the null character…

Incredibly easy to forget and only leads to errors in a quarter of cases...



Using “” to initialize puts the null in
Can use %s to print out a string

```
char mesg1[12] = {‘A’, ‘v’, ‘e’, ‘r’, ‘a’, ‘g’, ‘e’, ‘\0’ };
char mesg2[] = “Average”;
```

Using “” to initialize puts the null in
Can use %s to print out a string

Much easier to do it like this

```
char mesg1[12] = {‘A’, ‘v’, ‘e’, ‘r’, ‘a’, ‘g’, ‘e’, ‘\0’ };
char mesg2[] = “Average”;
```

Using “” to initialize puts the null in
Can use %s to print out a string

Much easier to do it like this

```
char mesg1[12] = {‘A’, ‘v’, ‘e’, ‘r’, ‘a’, ‘g’, ‘e’, ‘\0’ };
char mesg2[] = “Average”;
```

Using “” to initialize puts the null in
Can use %s to print out a string

Much easier to do it like this

```
char mesg1[12] = {‘A’, ‘v’, ‘e’, ‘r’, ‘a’, ‘g’, ‘e’, ‘\0’ };
char mesg2[] = “Average”;
```

Using “” to initialize puts the null in
Can use %s to print out a string

Strings

- Character arrays can be initialized like other arrays too but you need to specify the null character explicitly if you want to use it as a string
- Otherwise, its just an array of `chars`
- Easier to use the second method and put the string in quote marks
- Again, this only works at initialization
- Need to copy strings between arrays at other times

Show how perfectly possible to manipulate string characters

```
char secretMessage[] = “This is a secret message”;
int i;

for(i = 0; secretMessage[i] != ‘\0’; i++)
{
    printf(“%s\n”, secretMessage);
    secretMessage[i] = ‘*’;
}
```

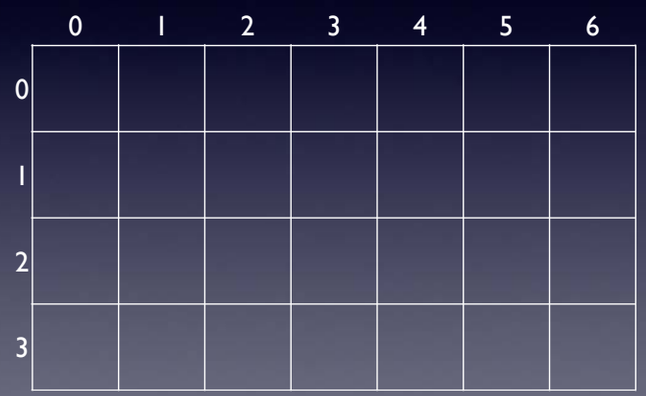
Can manipulate the characters in a string
Here we stop when we find the null character

Two-Dimensional Arrays

- Not limited to one dimensional arrays
- Can have two (or more) indices into the array
  `int novemberTemps[4][7];`
- Defines an array of 4x7 `int`s
- First index is the row, second is the column
- When initializing the values go across the columns first then onto the next row…

```
int novemberTemps[4][7] = { 9, 8,13,13,11,11, 6,
                           10, 9,10,10, 9, 9, 9,
                           10,11,12, 8, 8, 8, 9,
                            7, 4, 9, 9, 9, 9, 8 };
```

```
int novemberTemps[4][7] = { 9, 8,13,13,11,11, 6,
                           10, 9,10,10, 9, 9, 9,
                           10,11,12, 8, 8, 8, 9,
                            7, 4, 9, 9, 9, 9, 8 };
```



novemberTemps

```
int novemberTemps[4][7] = { 9, 8,13,13,11,11, 6,
                           10, 9,10,10, 9, 9, 9,
                           10,11,12, 8, 8, 8, 9,
                            7, 4, 9, 9, 9, 9, 8 };
```

	0	1	2	3	4	5	6
0	9	8	13	13	11	11	6
1	10	9	10	10	9	9	9
2	10	11	12	8	8	8	9
3	7	4	9	9	9	9	8

novemberTemps

```
int novemberTemps[4][7] = { 9, 8,13,13,11,11, 6,
                           10, 9,10,10, 9, 9, 9,
                           10,11,12, 8, 8, 8, 9,
                            7, 4, 9, 9, 9, 9, 8 };
```

	0	1	2	3	4	5	6
0	9	8	13	13	11	11	6
1	10	9	10	10	9	9	9
2	10	11	12	8	8	8	9
3	7	4	9	9	9	9	8

novemberTemps

Two Dimensional Arrays

- Accessed by giving the two indexs (row and column, in that order) in the form `a[i][j]`, e.g.
  `novemberTemps[2][5] = 12;`
- Nested for loops are the natural way to iterate over each array element in turn

Notes that its not a[i,j] or a(i,j) as in maths or basic but a[i][j]

Passing 2D Arrays to Functions

- C needs the 'size' of a 2D array to generate code to access it
- More specifically, it needs to know the number of columns
- When passing a 2D array to a function, the parameter declaration must specify how many columns there are
- So `void func(int td[][3])` is fine, but `void func(int td[][])` is not…
- Need to pass a 2D array of the same dimension…

Explain how it finds the offset

Array I/O

- Easy to read/write from an array, treat each value like a variable

```
printf(“%d”, array[0]);
scanf(“%d”, &tdarray[1][4]);
```

- Use a loop, with a variable as an index, to process the whole array
- But need to make sure we don’t go out of bounds…
- Make sure your array is big enough, its fine to only part fill it…
- And keep track of how many values you read in

fscanf/fprintf to handle reading writing from a file
Will see later how to allocate an array’s size dynamically

Array I/O

- Need to know how many values to read
- Might be obvious (e.g. 7 days in a week)
- At other times, you may need a mechanism to find out
  - Could specify in the file beforehand
  - Or use a terminating value…

terminating value -- some value that doesn't make sense as an actual value

Array I/O

- Loop conditions when reading values from a file become trickier
- Need to check for
  - End of File
  - End condition
  - Array size
- Can get tricky…

Give a demo — generalize the temperature average program