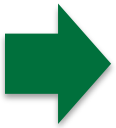
CS 50:
Software Design and Implementation

Command line args and arrays

Agenda



1. Arrays
2. Strings as arrays of chars
3. Command line arguments
4. Activity

Arrays declare a contiguous block of memory to hold several items

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is memory when run
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array)
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

- **Reserves a contiguous block of memory big enough to hold specified number of elements (5 here) times size of each element (4 bytes for integers) = 20 bytes**
- **Indices are 0…SIZE-1**
- **Global and static arrays are initialized to 0 (or null)**
- **Local arrays get whatever happens to be in memory**
- **Define a constant integer (old style uses #define directive)**

Global and static arrays are initialized to 0 or null

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is mem
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

- **global_array is initialized to 0**
- **Indices are 0…4**

```
$ mygcc -o array_int_test array_int_test.c
$ ./array_int_test
0 0
1 0
2 0
3 0
4 0
0 9
1 0
2 1720063584
3 32757
4 829015432
0 0
1 2
2 4
3 6
4 8
6 15775231
int size 4
array size 20
```

Local arrays are not initialized, holds whatever happens to be in memory

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is mem
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

- **Local array filled with whatever is left in memory**

```
$ mygcc -o array_int_test array_int_test.c
$ ./array_int_test
0 0
1 0
2 0
3 0
4 0
0 9
1 0
2 1720063584
3 32757
4 829015432
0 0
1 2
2 4
3 6
4 8
6 15775231
int size 4
array size 20
```

Values can be assigned by index

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is mem
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

- Assign array values to i*2 and print
- Array points to base address of array
- Remember from CS10, element at index *idx* memory address is:
  - base + idx*sizeof(element)

```
$ mygcc -o array_int_test array_int_test.c
$ ./array_int_test
0 0
1 0
2 0
3 0
4 0
0 9
1 0
2 1720063584
3 32757
4 829015432
0 0
1 2
2 4
3 6
4 8
6 15775231
int size 4
array size 20
```

C does not do bounds checking, can reference memory outside array

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is mem
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

Check value at index 6 (outside bounds of array by 2 values)

We will see this is a common cause of bugs

```
$ mygcc -o array_int_test array_int_test.c
$ ./array_int_test
0 0
1 0
2 0
3 0
4 0
0 9
1 0
2 1720063584
3 32757
4 829015432
0 0
1 2
2 4
3 6
4 8
6 15775231
int size 4
array size 20
```

Get size of values and arrays with `sizeof()` function

array_int_test.c

```c
#include<stdio.h>
int global_array[5];
const int SIZE = 5;

int main() {
        int a[SIZE];

        //global array is initialized to 0
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, global_array[i]);
        }

        //local array is filled with whatever is mem
        for (int i=0; i < SIZE; i++) {
                printf("%d %d\n",i, a[i]);
        }

        //assign values
        for (int i=0; i < SIZE; i++) {
                a[i] = i*2;
                printf("%d %d\n",i, a[i]);
        }

        //get value out of bounds (past end of array
        printf("%d %d\n",SIZE+1,a[SIZE+1]);

        //return size of int array
        printf("int size %lu\n",sizeof(int));
        printf("array size %lu\n",sizeof(a));

        return 0;
}
```

Integers are 4 bytes wide
Array is 20 bytes (4 bytes * 5 elements)

```
$ mygcc -o array_int_test array_int_test.c
$ ./array_int_test
0 0
1 0
2 0
3 0
4 0
0 9
1 0
2 1720063584
3 32757
4 829015432
0 0
1 2
2 4
3 6
4 8
6 15775231
int size 4
array size 20
```

Arrays can be initialized when declared

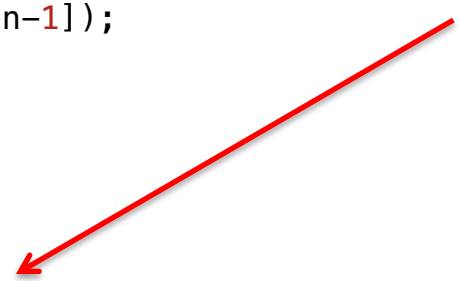
array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

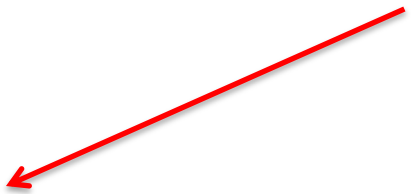


Declare array with constant size
Or
With initial elements (size not required)

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

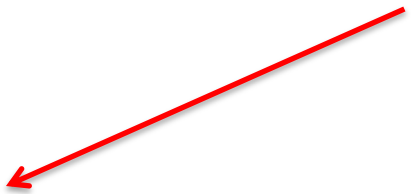


Declare array with constant size
Or
With initial elements (size not required)
Or
With size and initial elements (others initialize to 0)

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```



Declare array with constant size
Or
With initial elements (size not required)
Or
With size and initial elements (others initialize to 0)
If you provide more elements than size, extra elements ignored (e.g., d[2] = {1,2,3}, then 3 ignored)

Arrays can be initialized when declared

array_print.c

```c
#include <stdio.h>
const int SIZE=5;

void print_array(int arr[], int len) {
   for (int i = 0; i < len-1; i++) {
      printf("%d, ",arr[i]);
   }
   printf("%d\n",arr[len-1]);
}


int main() {
   int a[SIZE];
   int b[4];
   int c[] = {1,2,3};
   int d[5] = {1,2,3};

   printf("printing a\n");
   print_array(a, sizeof(a)/sizeof(int));
   printf("printing b\n");
   print_array(b, sizeof(b)/sizeof(int));
   printf("printing c\n");
   print_array(c, sizeof(c)/sizeof(int));
   printf("printing d\n");
   print_array(d, sizeof(d)/sizeof(int));

   return 0;
}
```

Local array a just happens to hold zero (not initialized, got lucky all zeros!)

```
$ mygcc -o array_print array_print.c
$ ./array_print
printing a
0, 0, 0, 0, 0
printing b
1, 0, -1277859539, 22087
printing c
1, 2, 3
printing d
1, 2, 3, 0, 0
```

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

Local array a just happens to hold zero (not initialized, got lucky all zeros!)

```
$ mygcc -o array_print array_print.c
$ ./array_print
printing a
0, 0, 0, 0, 0
printing b
1, 0, -1277859539, 22087
printing c
1, 2, 3
printing d
1, 2, 3, 0, 0
```

Local array b was not as lucky

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

Compiler can determine size of local array c to be three elements

Here we initialize each element

```
$ mygcc -o array_print array_print.c
$ ./array_print
printing a
0, 0, 0, 0, 0
printing b
1, 0, -1277859539, 22087
printing c
1, 2, 3
printing d
1, 2, 3, 0, 0
```

Arrays can be initialized when declared

array_print.c

```
14 #include <stdio.h>
15 const int SIZE=5;
16
17 void print_array(int arr[], int len) {
18    for (int i = 0; i < len-1; i++) {
19       printf("%d, ",arr[i]);
20    }
21    printf("%d\n",arr[len-1]);
22 }
23
24
25 int main() {
26    int a[SIZE];
27    int b[4];
28    int c[] = {1,2,3};
29    int d[5] = {1,2,3};
30
31    printf("printing a\n");
32    print_array(a, sizeof(a)/sizeof(int));
33    printf("printing b\n");
34    print_array(b, sizeof(b)/sizeof(int));
35    printf("printing c\n");
36    print_array(c, sizeof(c)/sizeof(int));
37    printf("printing d\n");
38    print_array(d, sizeof(d)/sizeof(int));
39
40    return 0;
41 }
```

Here we said to allocate room for 5 elements, and initialized the first three Remaining elements get whatever is in memory

```
$ mygcc -o array_print array_print.c
$ ./array_print
printing a
0, 0, 0, 0, 0
printing b
1, 0, -1277859539, 22087
printing c
1, 2, 3
printing d
1, 2, 3, 0, 0
```

Agenda

1. Arrays
2. Strings as arrays of chars
3. Command line arguments
4. Activity

Strings in C are a contiguous block of characters terminated with null

array_string_test.c

```c
#include<stdio.h>


int main() {
   char b[] = "hello";

   //char array
   printf("char array %s\n",b);
   printf("char size %lu\n",sizeof(char));
   printf("array size %lu\n",sizeof(b));
   printf("index 0: %c %d\n",b[0], b[0]);
   printf("index 4: %c %d\n",b[4], b[4]);
   printf("index 5: %c %d\n",b[5], b[5]);
   printf("index 6: %c %d\n",b[6], b[6]);
   printf("index 7: %c %d\n",b[7], b[7]);

   return 0;
}
```

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

Strings in C are a contiguous block of characters terminated with null

array_string_test.c

```c
#include<stdio.h>


int main() {
    char b[] = "hello";

    //char array
    printf("char array %s\n",b);
    printf("char size %lu\n",sizeof(char));
    printf("array size %lu\n",sizeof(b));
    printf("index 0: %c %d\n",b[0], b[0]);
    printf("index 4: %c %d\n",b[4], b[4]);
    printf("index 5: %c %d\n",b[5], b[5]);
    printf("index 6: %c %d\n",b[6], b[6]);
    printf("index 7: %c %d\n",b[7], b[7]);

    return 0;
}
```

C adds a null terminator ('\0') to this string

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

Print strings with “%s”

array_string_test.c

```c
#include<stdio.h>


int main() {
   char b[] = "hello";

   //char array
   printf("char array %s\n",b);
   printf("char size %lu\n",sizeof(char));
   printf("array size %lu\n",sizeof(b));
   printf("index 0: %c %d\n",b[0], b[0]);
   printf("index 4: %c %d\n",b[4], b[4]);
   printf("index 5: %c %d\n",b[5], b[5]);
   printf("index 6: %c %d\n",b[6], b[6]);
   printf("index 7: %c %d\n",b[7], b[7]);

   return 0;
}
```

Print strings with %s

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

Each character takes one byte of memory

array_string_test.c

```c
#include<stdio.h>


int main() {
   char b[] = "hello";

   //char array
   printf("char array %s\n",b);
   printf("char size %lu\n",sizeof(char));
   printf("array size %lu\n",sizeof(b));
   printf("index 0: %c %d\n",b[0], b[0]);
   printf("index 4: %c %d\n",b[4], b[4]);
   printf("index 5: %c %d\n",b[5], b[5]);
   printf("index 6: %c %d\n",b[6], b[6]);
   printf("index 7: %c %d\n",b[7], b[7]);

   return 0;
}
```

Each character is one byte

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

`sizeof` returns the number of bytes and counts the null terminator

array_string_test.c

```c
#include<stdio.h>


int main() {
    char b[] = "hello";

    //char array
    printf("char array %s\n",b);
    printf("char size %lu\n",sizeof(char));
    printf("array size %lu\n",sizeof(b));
    printf("index 0: %c %d\n",b[0], b[0]);
    printf("index 4: %c %d\n",b[4], b[4]);
    printf("index 5: %c %d\n",b[5], b[5]);
    printf("index 6: %c %d\n",b[6], b[6]);
    printf("index 7: %c %d\n",b[7], b[7]);

    return 0;
}
```

Local array b is 6 characters (plus one for null terminator) * 1 byte per characters = 6 bytes

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

Chars can be printed as character with "%c" or integer with "%d"

array_string_test.c

```c
#include<stdio.h>


int main() {
    char b[] = "hello";

    //char array
    printf("char array %s\n",b);
    printf("char size %lu\n",sizeof(char));
    printf("array size %lu\n",sizeof(b));
    printf("index 0: %c %d\n",b[0], b[0]);
    printf("index 4: %c %d\n",b[4], b[4]);
    printf("index 5: %c %d\n",b[5], b[5]);
    printf("index 6: %c %d\n",b[6], b[6]);
    printf("index 7: %c %d\n",b[7], b[7]);

    return 0;
}
```

Each character is an entry in the array, starting at index 0

Characters are stored as integers, print their ASCII values

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

The string terminator comes after the last character

array_string_test.c

```c
#include<stdio.h>


int main() {
   char b[] = "hello";

   //char array
   printf("char array %s\n",b);
   printf("char size %lu\n",sizeof(char));
   printf("array size %lu\n",sizeof(b));
   printf("index 0: %c %d\n",b[0], b[0]);
   printf("index 4: %c %d\n",b[4], b[4]);
   printf("index 5: %c %d\n",b[5], b[5]);
   printf("index 6: %c %d\n",b[6], b[6]);
   printf("index 7: %c %d\n",b[7], b[7]);

   return 0;
}
```

Null terminator is at index 5

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

C does not check bounds! You can read and write beyond the end of the string!

array_string_test.c

```c
#include<stdio.h>


int main() {
   char b[] = "hello";

   //char array
   printf("char array %s\n",b);
   printf("char size %lu\n",sizeof(char));
   printf("array size %lu\n",sizeof(b));
   printf("index 0: %c %d\n",b[0], b[0]);
   printf("index 4: %c %d\n",b[4], b[4]);
   printf("index 5: %c %d\n",b[5], b[5]);
   printf("index 6: %c %d\n",b[6], b[6]);
   printf("index 7: %c %d\n",b[7], b[7]);

   return 0;
}
```

C does not check array bounds; you can read and write past the end of an array without complaints

Here indices 6 and 7 are beyond the end of the array We see whatever was in memory

```
$ mygcc -o array_string_test array_string_test.c
$ ./array_string_test
char array hello
char size 1
array size 6
index 0: h 104
index 4: o 111
index 5:  0
index 6:  0
index 7: c 99
```

Its up to you to check array bounds, be careful, especially when writing!

String.h provides a number helpful string functions including `strlen` and `strcmp`

array_string_test2.c

```
1 #include<stdio.h>
2 #include<string.h>
3
4 int main() {
5    char b[] = "hello"; char c[] = {"hello"};
6    char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7    char f[strlen(b)+1];
8
9    //sizeof vs strlen
10   printf("b sizeof: %lu\n",sizeof(b));
11   printf("b strlen: %lu\n",strlen(b));
12
```

String.h provides a number helpful string functions including `strlen` and `strcmp`

array_string_test2.c

b and c are the same and are null terminated

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main() {
5     char b[] = "hello"; char c[] = {"hello"};
6     char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7     char f[strlen(b)+1];
8
9     //sizeof vs strlen
10    printf("b sizeof: %lu\n",sizeof(b));
11    printf("b strlen: %lu\n",strlen(b));
12
```

String.h provides a number helpful string functions including `strlen` and `strcmp`

array_string_test2.c

```
1 #include<stdio.h>
2 #include<string.h>
3
4 int main() {
5    char b[] = "hello"; char c[] = {"hello"};
6    char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7    char f[strlen(b)+1];
8
9    //sizeof vs strlen
10   printf("b sizeof: %lu\n",sizeof(b));
11   printf("b strlen: %lu\n",strlen(b));
12
```

d is not null terminated, but e is

strlen gives the number of characters without counting the null terminator

array_string_test2.c

```
1 #include<stdio.h>
2 #include<string.h>
3
4 int main() {
5    char b[] = "hello"; char c[] = {"hello"};
6    char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7    char f[strlen(b)+1];
8
9    //sizeof vs strlen
10   printf("b sizeof: %lu\n",sizeof(b));
11   printf("b strlen: %lu\n",strlen(b));
12
```

sizeof counts the null terminator

strlen does not!

```
$ mygcc -o array_string_test2
array_string_test2.c
$ ./array_string_test2
b sizeof: 6
b strlen: 5
```

strcmp compares two strings, returns 0 if equivalent

array_string_test2.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main() {
5     char b[] = "hello"; char c[] = {"hello"};
6     char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7     char f[strlen(b)+1];
8
    <snip>
12
13    //compare strings
14    if (!strcmp(b,c)) printf("b and c are the same\n");
15    else printf("b and c are not the same\n");
16
17    if (!strcmp(b,d)) printf("b and d are the same\n");
18    else printf("b and d are not the same\n");
19
20    if (!strcmp(b,e)) printf("b and e are the same\n");
21    else printf("b and e are not the same\n");
22
```

strcmp returns 0 if two strings are the same

Matches up to null terminator

```
$ mygcc -o array_string_test2 array_string_test2.c
$ ./array_string_test2
b sizeof: 6
b strlen: 5
b and c are the same
```

Notice the !

strcmp returns 0 if same

strcmp compares two strings, returns 0 if equivalent

array_string_test2.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main() {
5     char b[] = "hello"; char c[] = {"hello"};
6     char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7     char f[strlen(b)+1];
8
    <snip>
12
13    //compare strings
14    if (!strcmp(b,c)) printf("b and c are the same\n");
15    else printf("b and c are not the same\n");
16
17    if (!strcmp(b,d)) printf("b and d are the same\n");
18    else printf("b and d are not the same\n");
19
20    if (!strcmp(b,e)) printf("b and e are the same\n");
21    else printf("b and e are not the same\n");
22
```

d has no null terminator, not the same as b

```
$ mygcc –o array_string_test2 array_string_test2.c
$ ./array_string_test2
b sizeof: 6
b strlen: 5
b and c are the same
b and d are not the same
```

strcmp compares character by character until it hits the null terminator

Does not find null terminator in d

strcmp compares two strings, returns 0 if equivalent

array_string_test2.c

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main() {
5     char b[] = "hello"; char c[] = {"hello"};
6     char d[] = {'h','e','l','l','o'}; char e[] = {'h','e','l','l','o','\0'};
7     char f[strlen(b)+1];
8
     <snip>
12
13    //compare strings
14    if (!strcmp(b,c)) printf("b and c are the same\n");
15    else printf("b and c are not the same\n");
16
17    if (!strcmp(b,d)) printf("b and d are the same\n");
18    else printf("b and d are not the same\n");
19
20    if (!strcmp(b,e)) printf("b and e are the same\n");
21    else printf("b and e are not the same\n");
22
```

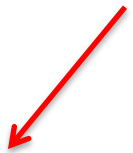
e has a null terminator, same as b

```
$ mygcc –o array_string_test2
array_string_test2.c
$ ./array_string_test2
b sizeof: 6
b strlen: 5
b and c are the same
b and d are not the same
b and e are the same
```

Be very careful to make sure strings are null terminated!

overflow.c

a has no null terminator, b does



```
14 #include <stdio.h>
15 #include <string.h>
16
17 int main(int argc, char *argv[]) {
18     char a[] = {'h','e','l','l','o'}; //no null terminator
19     char b[] = "world"; //C includes a null terminator
20     char c[sizeof(a)+sizeof(b)+1];
21
22     printf("Size of a is %lu\n",sizeof(a));
23     printf("Size of b is %lu\n",sizeof(b));
24
25     printf("strlen of a is %lu\n",strlen(a));
26     printf("strlen of b is %lu\n",strlen(b));
27
28     strcpy(c,a);
29     printf("%s\n",c);
30
31
32     return 0;
33 }
```

`sizeof` returns the number of bytes (characters are 1 byte)

overflow.c

```
14 #include <stdio.h>
15 #include <string.h>
16
17 int main(int argc, char *argv[]) {
18     char a[] = {'h','e','l','l','o'}; //no null terminator
19     char b[] = "world"; //C includes a null terminator
20     char c[sizeof(a)+sizeof(b)+1];
21
22     printf("Size of a is %lu\n",sizeof(a));
23     printf("Size of b is %lu\n",sizeof(b));
24
25     printf("strlen of a is %lu\n",strlen(a));
26     printf("strlen of b is %lu\n",strlen(b));
27
28     strcpy(c,a);
29     printf("%s\n",c);
30
31
32     return 0;
33 }
```

Sizes are as expected

```
$ mygcc overflow.c
$ ./a.out
Size of a is 5
Size of b is 6
```

strlen returns the number of characters but does not count the terminator

strlen counts until it hits a \0 terminator
a is not null terminated
b is allocated right after a in memory
strlen reads until hits b's terminator

overflow.c

```
14 #include <stdio.h>
15 #include <string.h>
16
17 int main(int argc, char *argv[]) {
18     char a[] = {'h','e','l','l','o'}; //no null terminator
19     char b[] = "world"; //C includes a null terminator
20     char c[sizeof(a)+sizeof(b)+1];
21
22     printf("Size of a is %lu\n",sizeof(a));
23     printf("Size of b is %lu\n",sizeof(b));
24
25     printf("strlen of a is %lu\n",strlen(a));
26     printf("strlen of b is %lu\n",strlen(b));
27
28     strcpy(c,a);
29     printf("%s\n",c);
30
31
32     return 0;
33 }
```

strlen of a is strange!

```
$ mygcc overflow.c
$ ./a.out
Size of a is 5
Size of b is 6
strlen of a is 10
strlen of b is 5
```

strcpy copies from one string into another until it hits a terminator

overflow.c

strcpy copies until it hits a \0 terminator
a is not null terminated
b is allocated right after a in memory
strpy copies until hits b's terminator

```
14 #include <stdio.h>
15 #include <string.h>
16
17 int main(int argc, char *argv[]) {
18     char a[] = {'h','e','l','l','o'}; //no null terminator
19     char b[] = "world"; //C includes a null terminator
20     char c[sizeof(a)+sizeof(b)+1];
21
22     printf("Size of a is %lu\n",sizeof(a));
23     printf("Size of b is %lu\n",sizeof(b));
24
25     printf("strlen of a is %lu\n",strlen(a));
26     printf("strlen of b is %lu\n",strlen(b));
27
28     strcpy(c,a);
29     printf("%s\n",c);
30
31
32     return 0;
33 }
```

strcpy copies until it hits a \0 terminator!

Strings that are not null terminated often cause bugs!

```
$ mygcc overflow.c
$ ./a.out
Size of a is 5
Size of b is 6
strlen of a is 10
strlen of b is 5
helloworld
```

Agenda

1. Arrays
2. Strings as arrays of chars
3. Command line arguments
4. Activity

argc and argv give the number and parameters from the command line

params.c

```
 1 #include <stdio.h>
 2
 3 int main(int argc, char *argv[])
 4 {
 5   int i;
 6
 7   printf("\nThe number of items on the command line is %d\n\n",argc);
 8   for (i = 0; i < argc; i++) {
 9     printf("argument %d is \"%s\"\n", i, argv[i]);
10     printf("The address stored in argv[%d] is %p\n", i, argv[i]);
11     printf("The first character pointed to there is \'%c\'\n", argv[i][0]);
12   }
13
14   return 0;
15 }
```

Count of parameters

Array of strings

Each parameter is a string in the array

Index 0 is the name of the program

argc and argv give the number and parameters from the command line

params.c

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5    int i;
6
7    printf("\nThe number of items on the command line is %d\n\n",argc);
8    for (i = 0; i < argc; i++) {
9      printf("argument %d is \"%s\"\n", i, argv[i]);
10     printf("The address stored in argv[%d] is %p\n", i, argv[i]);
11     printf("The first character pointed to there is \'%c\'\n", argv[i][0]);
12   }
13
14   return 0;
15 }
```

Escape double quotes with \" to print "

%p prints the memory address of a variable

Escape single quotes with \' to print '

argc and argv give the number and parameters from the command line

params.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  printf("\nThe number of items on the command line is %d\n\n",argc);
  for (i = 0; i < argc; i++) {
    printf("argument %d is \"%s\"\n", i, argv[i]);
    printf("The address stored in argv[%d] is %p\n", i, argv[i]);
    printf("The first character pointed to there is \'%c\'\n", argv[i][0]);
  }

  return 0;
}
```

```
$ mygcc -o params params.c
$ ./params 1 2 five 7

The number of items on the command line is 5

argument 0 is "./params"
The address stored in argv[0] is 0x7ffdbdc87e29
The first character pointed to there is '.'
argument 1 is "1"
The address stored in argv[1] is 0x7ffdbdc87e32
The first character pointed to there is '1'
argument 2 is "2"
The address stored in argv[2] is 0x7ffdbdc87e34
The first character pointed to there is '2'
argument 3 is "five"
The address stored in argv[3] is 0x7ffdbdc87e36
The first character pointed to there is 'f'
argument 4 is "7"
The address stored in argv[4] is 0x7ffdbdc87e3b
The first character pointed to there is '7'
```

5 parameters passed (include program name)

First parameter at index 0 of argv

Parameter memory address

Parameters are strings, sometimes we want to convert them other types

Options to convert parameters to integers

1) Use atoi (or atof for float)[1]
    - This is probably the fastest if you're using it in performance-critical code
    - Does no error reporting.
    - If the string does not begin with an integer, it will return 0
    - If the string contains junk after the integer, it will convert the initial part and ignore the rest
    - If the number is too big to fit in int, the behavior is unspecified

2) Use sscanf
    - Can specify format of input
    - Returns the number of successful conversions

[1] https://stackoverflow.com/questions/3420629/what-is-the-difference-between-sscanf-or-atoi-to-convert-a-string-to-an-integer

sscanf can convert multiple types from a string in a known format

sscanf.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
   int day, year, count;
   char weekday[20], month[20], dtm[100];

   strcpy( dtm, "Wednesday Sept 21 2022");
   count = sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );
   printf("Number of values converted: %d\n",count);
   return 0;
}
```


```
$ mygcc -o sscanf_test sscanf.c
$ ./sscanf_test
Sept 21, 2022 = Saturday
Number of values converted: 4
```


Adapted from https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm

sscanf can convert multiple types from a string in a known format

sscanf.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main () {
6     int day, year, count;
7     char weekday[20], month[20], dtm[100];
8
9     strcpy( dtm, "Wednesday Sept 21 2022");
10    count = sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );
11
12    printf("%s %d, %d = %s\n", month, day, year, weekday );
13    printf("Number of values converted: %d\n",count);
14    return 0;
15 }
```

Use sscanf (string scanf) to parse dtm looking for: string, string, integer, integer

sscanf returns the number of successful conversions

If count == 4, got all conversions

Use scanf (one s in scanf) to scan from stdin

```
$ mygcc -o sscanf_test sscanf.c
$ ./sscanf_test
Sept 21, 2022 = Saturday
Number of values converted: 4
```


Adapted from https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm

sscanf can convert multiple types from a string in a known format

sscanf.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
   int day, year, count;
   char weekday[20], month[20], dtm[100];

   strcpy( dtm, "Wednesday Sept 21 2022");
   count = sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );
   printf("Number of values converted: %d\n",count);
   return 0;
}
```

What is wrong with this code?
Should use strncpy not strcpy!

```
$ mygcc -o sscanf_test sscanf.c
$ ./sscanf_test
Sept 21, 2022 = Saturday
Number of values converted: 4
```


Adapted from https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm

sscanf can convert multiple types from a string in a known format

sscanf.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
   int day, year, count;
   char weekday[20], month[20], dtm[100];

   // strcpy( dtm, "Wednesday Sept 21 2022");
   strncpy( dtm, "Wednesday Sept 21 2022",100 );
   count = sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );

   printf("%s %d, %d = %s\n", month, day, year, weekday );
   printf("Number of values converted: %d\n",count);
   return 0;
}
```

What is wrong with this code?
Should use strncpy not strcpy!

```
$ mygcc -o sscanf_test sscanf.c
$ ./sscanf_test
Sept 21, 2022 = Saturday
Number of values converted: 4
```


Adapted from https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm

Agenda

1. Arrays
2. Strings as arrays of chars
3. Command line arguments
4. Activity