Strings and Pointers

July 18, 2005

Announcements

- Lab 4
  - Why are you taking this course?
- Lab 5
  - #7, 8: Reading in data from file using *fscanf*
- Quiz

Quiz

Strings

- Special character arrays
  - End in **null character**, '\0'

char hi[6];

H	i	!	\0		
0	1	2	3	4	5

char hello[20];

H	e	l	l	o	,		m	y		n	a	m	e		i	s	\0		

Null character--not a space--
marks end of string

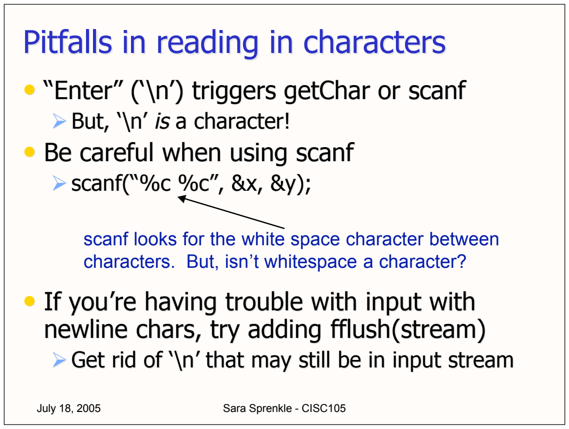
Initializing Strings

- char hello[] = {"hello"};
  - Implicitly, the null character is added to the end
  - The length of the array is set to 6
- char hi[] = "hi";
  - Null characted appended
  - Length of the array is 3
- char greetings[10] = {'g','r','e','e','t','i','n','g'}
  - Just a character array, not a string because no null character

Character I/O

- Have to understand how to manipulate characters to manipulate strings
- scanf
  - scanf("%c", &x);
- getChar
  - Returns a character from stdin (terminal)
  - Use:

```
char x;
…
x = getChar();
```

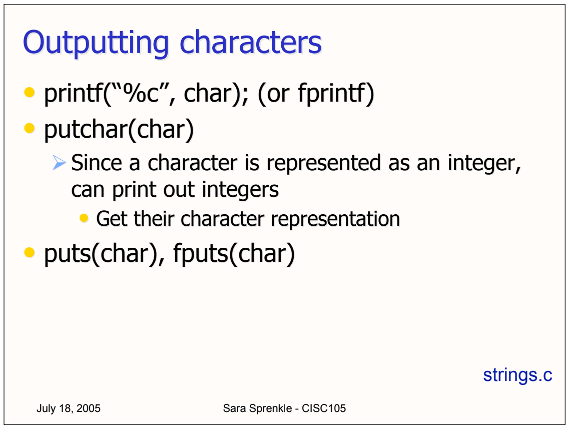


Pitfalls in reading in characters

- “Enter” (‘\n’) triggers getChar or scanf
  - But, ‘\n’ *is* a character!
- Be careful when using scanf
  - scanf(“%c %c”, &x, &y);

scanf looks for the white space character between characters. But, isn’t whitespace a character?

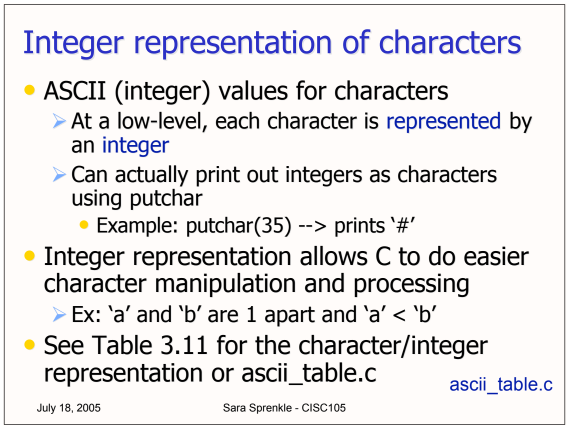
- If you’re having trouble with input with newline chars, try adding fflush(stream)
  - Get rid of ‘\n’ that may still be in input stream



Outputting characters

- printf(“%c”, char); (or fprintf)
- putchar(char)
  - Since a character is represented as an integer, can print out integers
    - Get their character representation
- puts(char), fputs(char)

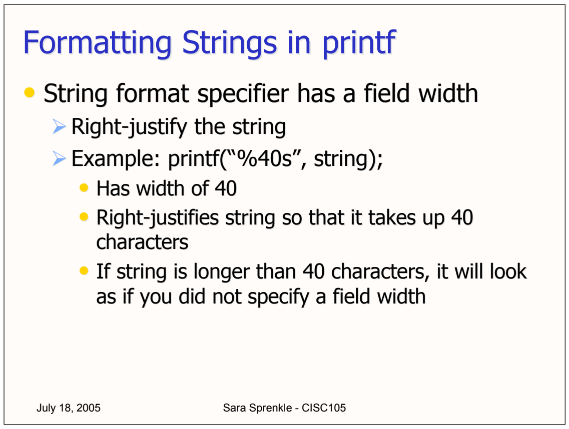
strings.c



Integer representation of characters

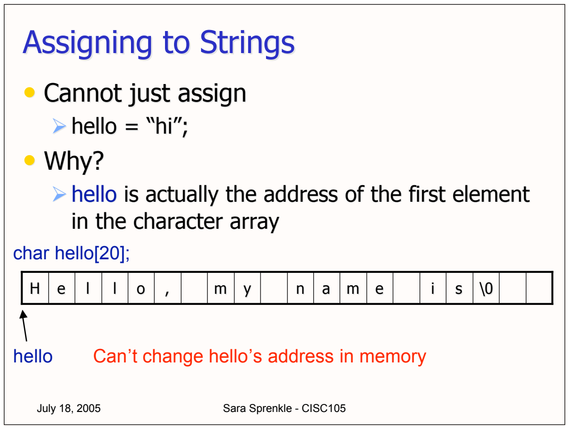
- ASCII (integer) values for characters
  - At a low-level, each character is represented by an integer
  - Can actually print out integers as characters using putchar
    - Example: putchar(35) --> prints ‘#’
- Integer representation allows C to do easier character manipulation and processing
  - Ex: ‘a’ and ‘b’ are 1 apart and ‘a’ < ‘b’
- See Table 3.11 for the character/integer representation or ascii_table.c

ascii_table.c



Formatting Strings in printf

- String format specifier has a field width
  - Right-justify the string
  - Example: printf(“%40s”, string);
    - Has width of 40
    - Right-justifies string so that it takes up 40 characters
    - If string is longer than 40 characters, it will look as if you did not specify a field width



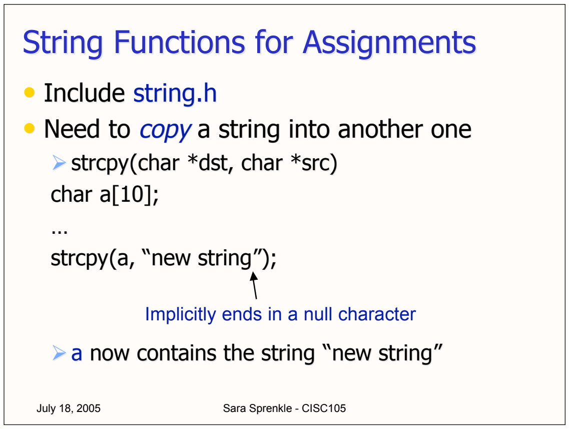
Assigning to Strings

- Cannot just assign
  - hello = “hi”;
- Why?
  - hello is actually the address of the first element in the character array

char hello[20];

H	e	l	l	o	,		m	y		n	a	m	e		i	s	\0		

hello Can’t change hello’s address in memory



String Functions for Assignments

- Include string.h
- Need to *copy* a string into another one
  - strcpy(char *dst, char *src)

```
char a[10];
…
strcpy(a, “new string”);
```

Implicitly ends in a null character

  - a now contains the string “new string”

String Functions: Determining Size

- sizeof(char_array_name)
  - Returns amount of memory allocated to the array
- strlen(char_array_name)
  - Returns length of the string
- Example:
  - char example[10] = "example";
  - sizeof: 10 Bytes
  - strlen: 7

Why don't we have a similar "length" function for numeric arrays?

String I/O: Input

- scanf("%s", char_array_name);
  - Don't need the [] or & when passing the argument, char_array_name
  - Reads in one word (until whitespace) from stdin
  - fscanf works similarly
- gets(char_array_name);
  - Reads until newline character
  - Stored in char_array_name
  - Newline character is not included in the string
  - Appends a null character to the string

Why is gets unsafe?

stringio.c

String I/O: Input

- fgets(char_array_name, length, stream)
  - Reads until newline character or length-1 characters from the input stream, e.g., file stream
  - Stored in char_array_name
  - Newline character *is* included in the stored string
  - Appends a null character to the string (after the \n)
- Why is fgets the preferred way to read character input?

stringio.2.c

String Input Functions Summary

Name	Notes
scanf, fscanf	Reads in one word
gets	Reads in one line, does not include newline
fgets	Reads in one line until some limit, includes newline character

String I/O: Output

- printf("%s", char_array_name)
  - Unlike scanf, prints the whole string, regardless of white space
- puts(char_array_name)
  - Prints the string out
  - Appends a newline character to output
- fputs(char_array_name, outputstream)
  - Prints out the string

String Output Functions Summary

Name	Notes
printf, fprintf	Prints whole string
puts	Prints string, appends newline
fputs	Prints string to stream

Manipulating Characters

- Character Functions: Table 7.1 in Book
- Handle textual input
- Determine character type
  - isalnum: alpha-numeric character
  - isalpha: alphabetic character
  - iscntrl: control character (like newline, tab)
  - …

Manipulating Characters

- tolower
  - Return the lowercase version of the character
  - Could we implement this function?
- toupper
  - Return the uppercase version of the character

More string functions

- Common string functions: Table 7.3
- atoi: returns the int value of a string
- strcat: combine two strings into one string
- strcmp: compares two strings
  - Use to sort strings (using algorithms from last week)
- Lots more!

2-D Character Arrays

- Similar to 2-D numeric arrays
- Example, a dictionary

```
/* 100 words, at most 39 characters long (plus the
   null character) */
char dictionary[100][40];
```

- dictionary[1] is "cat", which is a char[]
- dictionary[1][2] is 't', which is a char

0	"boy"
1	"cat"
2	"dog"
…	…
99	"zebra"

2-D Character Arrays

- How would you print each word?

```
for(i=0; i < 100; i++)
    printf("%s", dictionary[i]);
```

- How would you print the second character in each word?

```
for(i=0; i < 100; i++)
    printf("%c", dictionary[i][1]);
```

Pointers

- Points to a location in memory

int x = 7; x is stored in first available memory location, x0000

int *ptr=&x;

Means that ptr is of type pointer to an int.

Memory Location	Value
x0000 x	7
x0004 ptr	x0000
x0008	
…	
x9996	

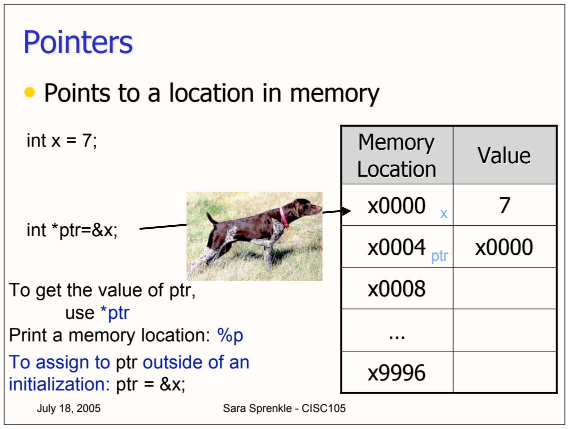
"ptr points to the location of x"

New Operators:

&: "address of"

*: "value of" (or *dereferencing*)

pointerex.c



Pointers

- Points to a location in memory

int x = 7;

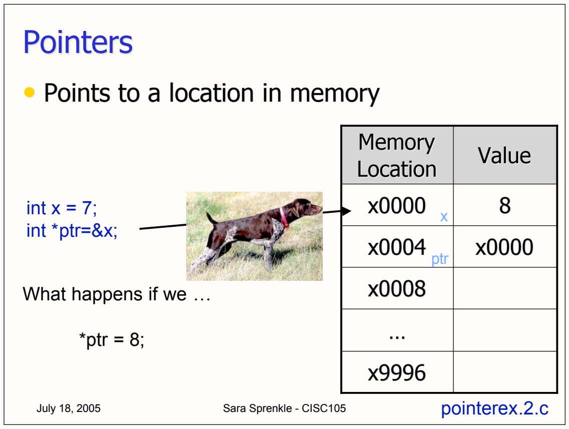
int *ptr=&x;

Memory Location	Value
x0000 x	7
x0004 ptr	x0000
x0008	
…	
x9996	

To get the value of ptr, use *ptr

Print a memory location: %p

To assign to ptr outside of an initialization: ptr = &x;



Pointers

- Points to a location in memory

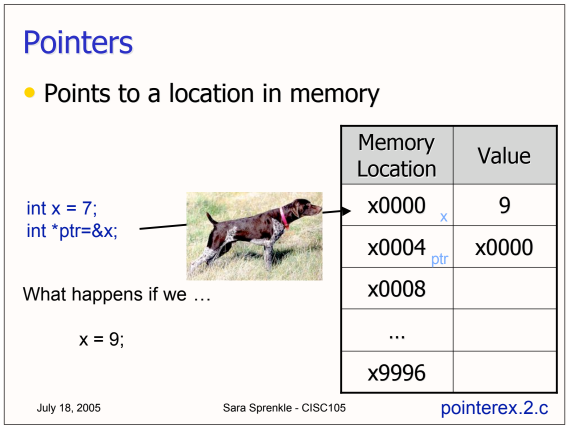
int x = 7;
int *ptr=&x;

Memory Location	Value
x0000 x	8
x0004 ptr	x0000
x0008	
…	
x9996	

What happens if we …

*ptr = 8;

pointerex.2.c



Pointers

- Points to a location in memory

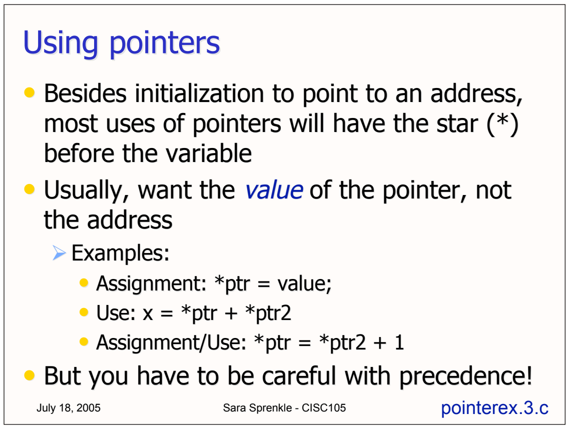
int x = 7;
int *ptr=&x;

Memory Location	Value
x0000 x	9
x0004 ptr	x0000
x0008	
…	
x9996	

What happens if we …

x = 9;

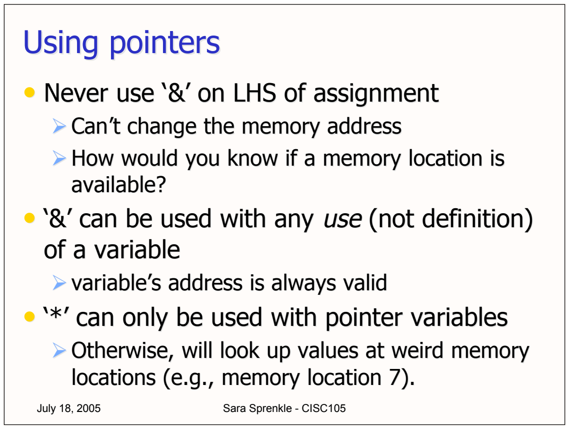
pointerex.2.c



Using pointers

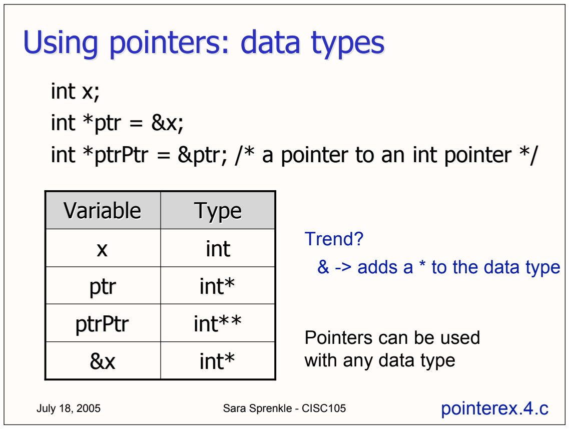
- Besides initialization to point to an address, most uses of pointers will have the star (*) before the variable
- Usually, want the *value* of the pointer, not the address
  - Examples:
    - Assignment: *ptr = value;
    - Use: x = *ptr + *ptr2
    - Assignment/Use: *ptr = *ptr2 + 1
- But you have to be careful with precedence!

pointerex.3.c



Using pointers

- Never use '&' on LHS of assignment
  - Can't change the memory address
  - How would you know if a memory location is available?
- '&' can be used with any *use* (not definition) of a variable
  - variable's address is always valid
- '*' can only be used with pointer variables
  - Otherwise, will look up values at weird memory locations (e.g., memory location 7).



Using pointers: data types

```
int x;
int *ptr = &x;
int *ptrPtr = &ptr; /* a pointer to an int pointer */
```

Variable	Type
x	int
ptr	int*
ptrPtr	int**
&x	int*

Trend?

& -> adds a * to the data type

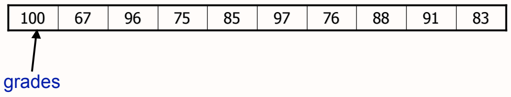
Pointers can be used with any data type

pointerex.4.c

We have been using pointers!

- Array identifiers are pointers to the first element in the array

int grades[10];



100	67	96	75	85	97	76	88	91	83

- In scanf, we had to use &var notation
  - So we could *modify* the value in var
  - Now, you know why the ampersand!

Pointers and Functions

- Instead of pass-by-value, we can pass-by-reference
  - We discussed pass-by-ref briefly last week
  - Using pointers allows us to modify the parameters passed in (also called *reference parameters*)
    - Example: swapping numbers
    - Recall that before, our swap function did not change the variables after returning to the main function

swap.*.c

Pointers and Functions

- Two different functions to square a number
  - int square(int n);
  - void square(int *n);

square.2.c

Pass By Reference

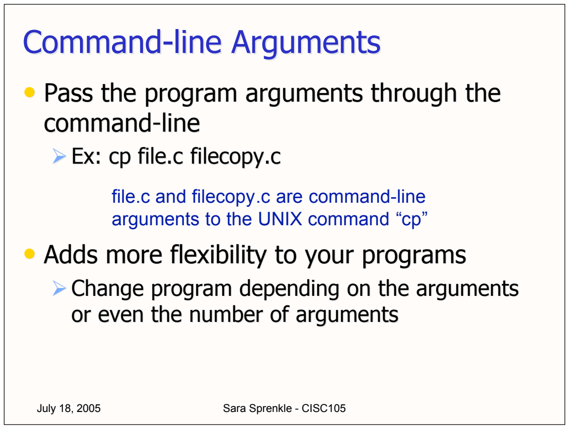
- Allows us to “return” more than one variable from a function
  - Can change values in multiple variables
- Passing arrays is pass by reference
  - Because arrays are pointers
  - Why we could modify arrays

Arrays passed to functions

- The following two function prototypes are equivalent:
  - void modArray(int array[]);
  - void modArray(int *array);

Strings as character pointers

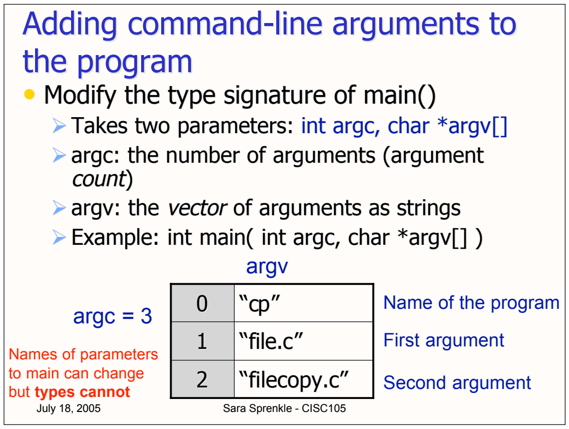
- Since arrays and pointers are equivalent
  - string == character array == character pointer
    - However, once you’ve chosen either the array or pointer, must be consistent in use in function headers and passing parameters
- Why would we choose one representation over another?
  - Character arrays have an associated length
    - May not want to deal with a fixed length



Command-line Arguments

- Pass the program arguments through the command-line
  - Ex: cp file.c filecopy.c

    file.c and filecopy.c are command-line arguments to the UNIX command "cp"
- Adds more flexibility to your programs
  - Change program depending on the arguments or even the number of arguments



Adding command-line arguments to the program

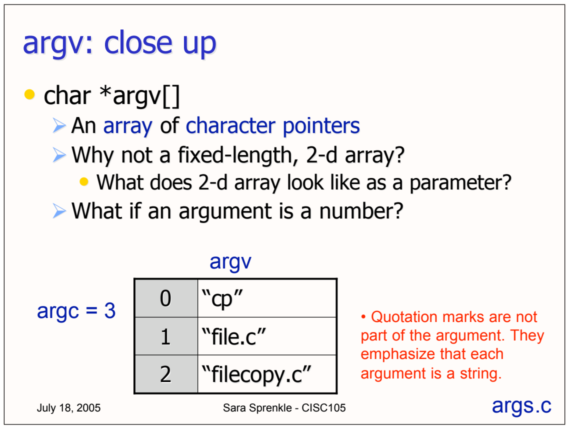
- Modify the type signature of main()
  - Takes two parameters: int argc, char *argv[]
  - argc: the number of arguments (argument *count*)
  - argv: the *vector* of arguments as strings
  - Example: int main(int argc, char *argv[])

argv

argc = 3

0	"cp"	Name of the program
1	"file.c"	First argument
2	"filecopy.c"	Second argument

Names of parameters to main can change but **types cannot**



argv: close up

- char *argv[]
  - An array of character pointers
  - Why not a fixed-length, 2-d array?
    - What does 2-d array look like as a parameter?
  - What if an argument is a number?

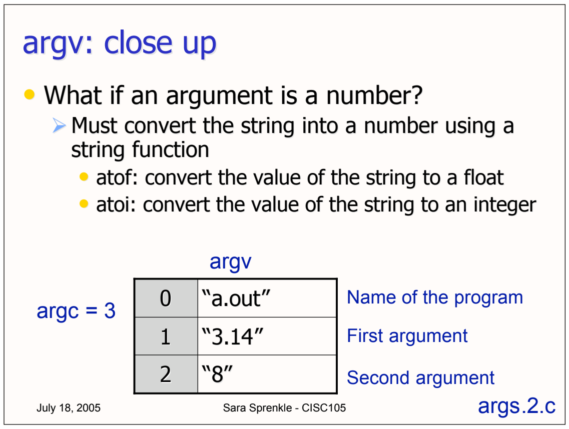
argv

argc = 3

0	"cp"
1	"file.c"
2	"filecopy.c"

- Quotation marks are not part of the argument. They emphasize that each argument is a string.

args.c



argv: close up

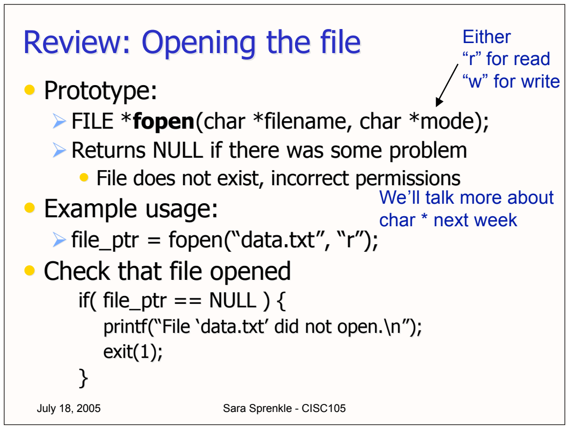
- What if an argument is a number?
  - Must convert the string into a number using a string function
    - atof: convert the value of the string to a float
    - atoi: convert the value of the string to an integer

argv

argc = 3

0	"a.out"	Name of the program
1	"3.14"	First argument
2	"8"	Second argument

args.2.c



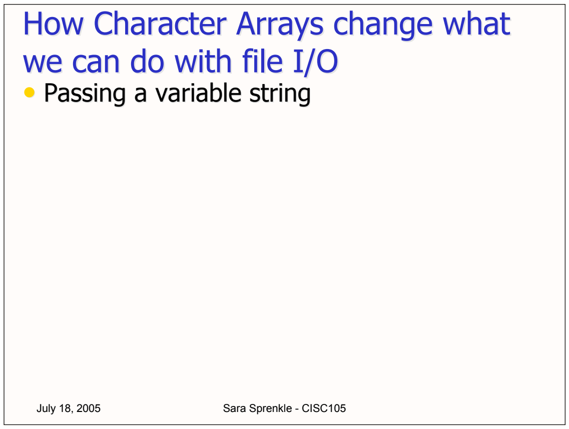
Review: Opening the file

Either
"r" for read
"w" for write

- Prototype:
  - FILE ***fopen**(char *filename, char *mode);
  - Returns NULL if there was some problem
    - File does not exist, incorrect permissions
- Example usage:
  - file_ptr = fopen("data.txt", "r");

  We'll talk more about char * next week
- Check that file opened

```
if( file_ptr == NULL ) {
    printf("File 'data.txt' did not open.\n");
    exit(1);
}
```

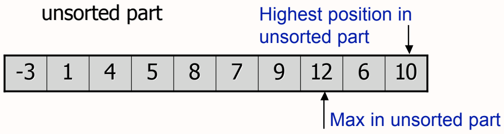


How Character Arrays change what we can do with file I/O

- Passing a variable string

Review: Selection Sort

- Key idea: keep track of the sorted and unsorted parts of the array
- Algorithm:
  - While array is not sorted
    - Find the maximum of the unsorted part
    - Swap with element in the highest position in the unsorted part



Highest position in unsorted part

-3	1	4	5	8	7	9	12	6	10

Max in unsorted part

Extend Selection Sort to Strings

- While array is not sorted
  - Find the maximum of the unsorted part
  - Swap with element in the highest position in the unsorted part
- What is the maximum?
- How do we swap?

0	1	2	3	4	5	6	7	8	9
car	bat	zebra	kite	lamb	ball	man	fish	soap	arm

stringsort.1.c