Intro to Programming II

Strings in C

Chris Brooks

Department of Computer Science

University of San Francisco
Remember that C does not have a String data type.

Instead, a string is just an array of characters.

This means that we can access characters in a string just like we do elements of an array.
So to iterate through a string and convert all the characters to upper case, we could do:

```c
#include <ctype.h>
int main(void) {
    char str[80];
    int i;
    scanf(``%s'', str);
    for (i = 0; i < 80; i++) {
        str[i] = toupper(str[i]);
    }
    printf(``%s'', str);
}
```
C has a number of useful functions built into the string.h library.

- `strlen`
- `strcpy/strncpy`
- `strcat/strncat`
- `strdup`
- `strcmp/strncmp`
### 27-3: strlen

**strlen** takes a string as input and returns an int representing the length of the string.

```c
int countXs(char *instr) {
    int i, count = 0;
    for (i = 0; i < strlen(instr); i++) {
        if ( instr[i] == 'x' || instr[i] == 'X' ) {
            count++;
        }
    }
    return count;
}
```
Exercise

6 What if strlen didn’t exist? How would we write it?
**27-5: `strcpy`**

1. `strcpy` takes two arguments: dest and src.
2. `strcpy` makes a copy of src in dest.

```c
char in[80] = "hello world";
char out[80];

strcpy(out, in);
printf("%s", out);
```
What if strcpy didn’t exist? How would we write it?
Most string functions also have an 'n' version.

- `strncpy`

This takes a third argument which represents the number of characters to copy.

`strcpy(out,in,20)` will copy the first 20 characters of `in` into `out`. 
6. `strcmp(s1,s2)` compares two strings, `s1` and `s2`.
6. Semantics are the same as `compareTo`.
   - `s1 < s2`: return `< 0`.
   - `s1 == s2`: return `0`
   - `s1 > s2`: return `> 0`.
char *name1, *name2;

printf("enter name 1:");
scanf("%s", name1);
printf("enter name 2:");
scanf("%s", name2);
if (strcmp(name1, name2) < 0) {
    printf("name 1 comes first.");
} else if (strcmp(name1, name2) > 0) {
    printf("name 2 comes first.");
} else {
    printf("equal.");
}
What if strcmp didn’t exist? How would we write it?
strcat(dest, src);

strcat appends a copy of the characters in src to the string dest.

dest must have enough room for all of the characters in src.
char s1[80] = "world ";
char s2[80] = "hello ";
strcat(s2,s1);
printf("%s",s2);
6 What if strcat didn’t exist? How would we write it?
strdup returns a pointer to a copy of the input string.

These pointers refer to different memory locations.
char *s1 = "Hello world";
char *s2 = strdup(s1);
s2[5] = 'Q';
printf("%s %s", s1, s2);
27-16: *Hints for strings*

- When processing a string character by character, be careful not to run off the end.
- When creating your own strings by hand, be sure to add a null character at the end.
- Use `string.h` when you can.
Let’s say we need to keep track of whether 16 computers are on or off.

How should we represent this?
Let’s say we need to keep track of whether 16 computers are on or off.

How should we represent this?

Sixteen separate variables? ick.

An array of 16 ints? Better, but wasteful.

Let’s do it with a single int.
Recall that an int is represented by (usually) 32 bits.

Let’s use the lower 16 bits to represent our computers.

Start out with all zeros.

As a computer is turned on, we’ll set that bit to 1.

As it’s turned off, we’ll set that bit to zero.

How to do this?
Let’s call the integer representing our current set of ’on’ machines the state.

We can OR this state with an integer that has a 1 in the place we want to ’turn on’, and 0 in all others.

```c
unsigned int state = 0;
unsigned int mask = 4; /* turn on computer 3 */
state |= mask;
```
If we want to turn off a machine, we AND the state with an integer that has a zero in that place, and 1s everywhere else.

```c
mask = 65531; /* 1111111111111011 = 65531 */
state = state & mask;
```
C also allows you to *shift* an integer left or right.

11001 shifted left becomes 110010

11001 shifted right becomes 1100

- Shifting left is equivalent to multiplying by two.
- Shifting right is equivalent to dividing by two.
Let’s assume that we’re going to store the state of all 16 computers in a single int.

We’ll prompt the user for a computer number, and whether it’s used or not.

We need functions to:
- Mark a computer as used.
- Mark a computer as unused.
- Print out the state of a computer.