Pointer arithmetic, dynamic memory, and structs

9/23/15
Which of the following lines has an error?

int x = 3;               //A
char[10] str;            //B
char* p = str;           //C
scanf("%d", &x);        //D
scanf("%s", &str);      //E
Which of the following lines has an error?

```c
int x = 3; //A
char[10] str; //B
char* p = str; //C

scanf("%d", &x); //D
scanf("%s", &str); //E (warning, but bad)
```
From lab: Find first non-space

//line stores what I read

int i = 0; //index into line
while(line[i] == ' ') //advance i to 1st non-space
    i++;
From lab: Copying a substring

//line stores what I read, word will store 1\textsuperscript{st} word
//i is index (in line) of beginning of word

int j = 0;  //index in word
while((line[i] != ' ') && (line[i] != 0))  //while in word
  word[j] = line[i];  //copy next char
  i++;  //advance indices
  j++;
}
word[j] = 0;  //add terminating 0
Yuck!

• Both use string[index] construct
  – potentially lots of indices
  – hard to print part of a word (requires a copy)
  – arguably, lots of typing
Alternative: Pointer arithmetic

• Use a pointer into the array

• Pointer itself moves: \( \text{ptr}++ \) advances it
  – Can also use other arithmetic
  – adding “1” moves address by 1 cell (\textbf{not} 1 byte)

• Access value at pointer’s location with \(*\text{ptr}\)
Finding first non-space revisited

//line stores what I read

char* ptr = line; //pointer into line
while(*ptr == ' ') //advance i to 1st non-space
  ptr++;
If some is good, more must be better...

void strcpy(char* a, char* b) {
    //copy string b into a
    while(*a++ = *b++);
}

What does this do?

```c
int f(char* a) {
    char* b = a;
    while(*b++);
    return b-a-1;
}
```
What about this?

```c
int g(char* a, char* b) {
    while(*a++ == *b++)
        if(*(a-1) == 0)
            return 0;
    return *(a-1) - *(b-1);
}
```
Dynamic memory allocation

• Can’t do with stack-allocated memory:
  – Return pointer to it
  – Get variable-sized part of it (sometimes actually possible)
Dynamic memory allocation

• Can’t do with stack-allocated memory:
  – Return pointer to it
  – Get variable-sized part of it (sometimes actually possible)

• Need memory allocated from the heap:
  
  ```
  int* ptr = (int*) malloc(10*sizeof(int));
  ```

  cast; malloc returns void*
gives number of bytes in an int
Other notes on malloc

• Need to include `<stdlib.h>`

• Need to explicitly free dynamically-allocated memory
  – pass pointer once to free:
    ```c
    free(ptr);
    ```
Which of the following lines does not do as its comment says?

#include <stdio.h>       //for malloc (line 1)
...
int* p = (int*) malloc(10); //allocate 10 ints (line 2)
char* s = (char*) malloc(6); //allocate room for “David” (line 3)

A. Line 1 does not match its comment
B. Line 2 does not match its comment
C. Line 3 does not match its comment
D. More than 1 line does not match its comment
E. All lines match their comments
Which of the following lines does not do as its comment says?

```c
#include <stdio.h>  //for malloc (line 1)
...
int* p = (int*) malloc(10);  //allocate 10 ints (line 2)
char* s = (char*) malloc(6);  //allocate room for “David” (line 3)
```

A. Line 1 does not match its comment
B. Line 2 does not match its comment
C. Line 3 does not match its comment
D. More than 1 line does not match its comment (1, 2, and arguably 3)
E. All lines match their comments