CS61C: Belief, Buffers & Pointers

CS61C Fall2007 - Discussion #3

Greg Gibeling

Stump the TA

- Goal
  - A problem Greg can't solve
  - A question Greg can't answer
- Rules
  - No deliberate obfuscation
    - The problem/question may be complex
    - Your explanation of it must be as clear as possible
  - No detailed reference information
    - I'm not going to spend 20 minutes looking up Ann Margaret's pant size

Course Newsgroup

- Access
  - news.berkeley.edu from "on campus"
  - authnews.berkeley.edu from home
    - Login using your CalNET ID
    - Requries SSL
- Update your Gecos information
  - Use your real name not "Class Account"
  - `finger | grep <login>`
  - `ssh update`

Belief & Debugging (1)

- Questions
  - Are you religious? (Don't answer out loud)
  - Do your beliefs (or lack thereof) affect your performance & abilities in this class?

Belief & Debugging (2)

- Answers
  - Are you religious? (Don't answer out loud)
    - It was a trick question
    - It doesn't really matter
  - Do your beliefs (or lack thereof) affect your performance & abilities in this class?
    - Your beliefs affect everything you do, particularly when you're debugging
    - We'll talk about how in a minute…

Belief & Debugging (3)

- Questions
  - When you run a program and it doesn't work as expected, what's next?
  - Are you afraid to answer because I tricked you last time?

Belief & Debugging (4)

- Questions
  - When you run a program and it doesn't work as expected, what's next?
    - Most people would say "debugging"
    - Shouldn't you stop to wonder about your expectations before you blame the program?
  - Are you afraid to answer because I tricked you last time?
    - This is good!
    - You are questioning yourself…

Belief & Debugging (5)

- Belief: You believe you know what your program does
  - You think you understand it
  - You think you know what the library calls do
- Fact: You can read what it actually does
  - Computers are as close to perfect as possible
  - A computer error or fault is very unlikely
- Consequence
  - A mismatch means your beliefs are wrong
  - Always assume that you are **dead wrong**
    - It's possible the bug is a typo

Buffer Overflows (1)

- Buffer Overflow
  - Write n+x bytes to an n byte buffer
  - Results in crash (we hope)
- Common causes
  - Fixed length buffers
  - Off-by-one errors
  - Misplaced belief
- Fixes
  - Use strncpy
    - Don't forget to worry about concurrency
  - **Always validate all arguments**

Buffer Overflows (2)

- An example with strncpy

```
void foo(char* string) {
    int length = strlen(string);
    char* buffer = (char*)malloc((length+1)*sizeof(char));
    strncpy(buffer, string, length);
    // etc…
}
```

- A bug in dirmain.c

```
    char cmd[6];
    // etc…
    sscanf(line, "%6s", cmd);
```

  - Why doesn't this work?
  - Why didn't we notice this until yesteday?

Buffer Overflows (3)

- Who cares?
  - Every employer you will ever interview with
    - Buffer Overflows are one of the largest sources of software cracks ever
    - Visual Studio issues warnings for use of strcpy!
  - You
    - Countless student hours wasted on debugging
    - No one is immune, our code contained an error!

A Smarter Free

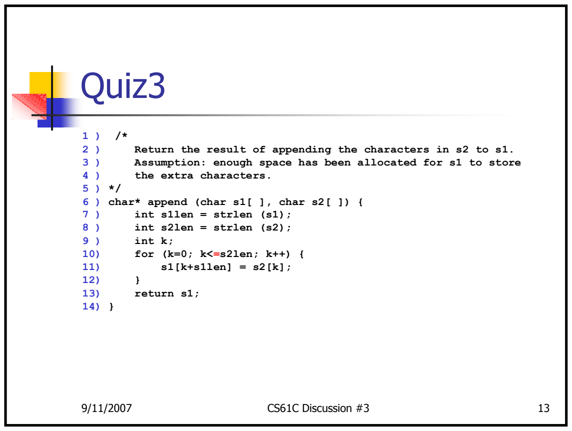
- The macro

```
#define FREE(x) { if (x) free(x); x = NULL; }
```

  - Cheap, easy to remember and use
  - Prevents all kinds of errors (double free() calls)
- The function

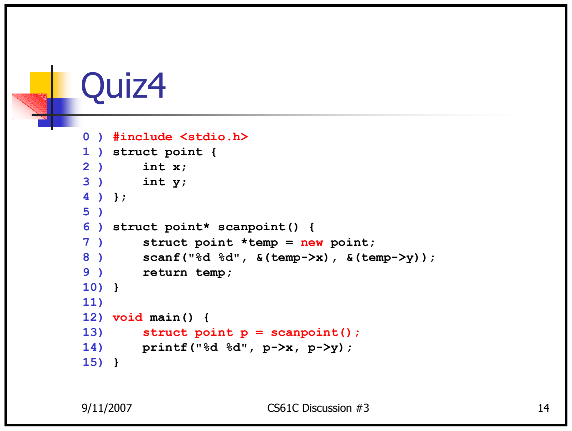
```
void FREE(void**x) {
    if (x) { if (*x) free(*x); (*x) = NULL; }
}
```

  - A little more expensive (maybe)
  - More versatile
- When don't these work?
- Why aren't they always a good idea?



Quiz3

```
 /*
    Return the result of appending the characters in s2 to s1.
    Assumption: enough space has been allocated for s1 to store
    the extra characters.
*/
char* append (char s1[ ], char s2[ ]) {
    int s1len = strlen (s1);
    int s2len = strlen (s2);
    int k;
    for (k=0; k<=s2len; k++) {
        s1[k+s1len] = s2[k];
    }
    return s1;
}
```

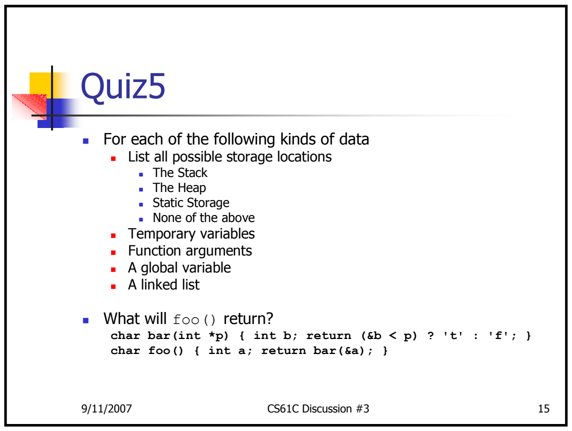


Quiz4

```
#include <stdio.h>
struct point {
    int x;
    int y;
};

struct point* scanpoint() {
    struct point *temp = new point;
    scanf("%d %d", &(temp->x), &(temp->y));
    return temp;
}

void main() {
    struct point p = scanpoint();
    printf("%d %d", p->x, p->y);
}
```

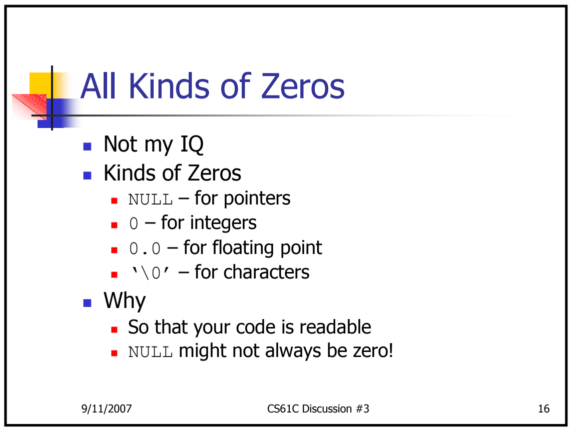


Quiz5

- For each of the following kinds of data
  - List all possible storage locations
    - The Stack
    - The Heap
    - Static Storage
    - None of the above
  - Temporary variables
  - Function arguments
  - A global variable
  - A linked list

- What will `foo()` return?

```
char bar(int *p) { int b; return (&b < p) ? 't' : 'f'; }
char foo() { int a; return bar(&a); }
```



All Kinds of Zeros

- Not my IQ
- Kinds of Zeros
  - `NULL` – for pointers
  - `0` – for integers
  - `0.0` – for floating point
  - `'\0'` – for characters
- Why
  - So that your code is readable
  - `NULL` might not always be zero!