PURDUE UNIVERSITY®

CS 240: Programming in C

Lecture 11: More Pointers Debugging Introduction

Prof. Jeff Turkstra

Announcements

- The linter(s) is a tool to help you adhere to the code standard
  - It is your job to ensure that you are following the code standard
  - The linter also helps with grading, but the graders ultimately determine your grade
- Start your homework early :-)

Announcements

- Attend lab and office hours earlier
  - We cannot guarantee help will be available at the last minute
- Remember, you can attend any lab section!
  - This should be your primary source for help, NOT office hours

Homeworks 2 and 3

- Homework 2 Format String
  2024-10-25,"LWSN B148",turkstra,53.56,45,"vi",2.69
- Homework 3 Format String
  John Doe|2023-06-16|2023-07-24|
  1HGCM82633A001234,Honda,Accord|120.35|8.5|
  80.45|20.00

Feasting with Faculty

- Lunch this Thursday!
  - 12pm – 1pm
  - ERHT Private Dining Room B
- No lunch next week!

Notes

- Homework 5
  - Don’t close the file pointer!
- Takehome Quiz 4

Midterm Exam 1

- Monday, March 3, 8pm - 10pm
- Rooms TBA with seating chart
- Accommodated exams will overlap the same time in a different room
  - We'll send you an email
- More information Monday

Reading

- Read Chapter 8.7 and Chapter 11 in Beej's

Definition vs indirection

```
int x = 7;
int *p = &x; // this * means p is a pointer

*p = 5; // this * means dereference p
```

A complete example

```
#include <stdio.h>

int main() {
   int x = 0;
   int *p = 0;

   p = &x;
   *p = 5;

   printf("x = %d\n", x);
   return 0;
}
```

p now "points" to x

*p is equivalent to x

Another pointer example

```
int main() {
  int ctr = 0;
  int *ptr = 0;
  int int4 = 18;
  int int3 = 11;
  int int2 = 10;
  int int1 = 7;

  ptr = &int1;

  for (ctr = 0; ctr < 7; ctr++) {
    printf("Value at address %p: 0x%x (%d)\n",
           ptr, *ptr, *ptr);
    ptr++;
  }

  return 0;
}
```

More pointer basics

- Pointers can be used just as arrays
- Arrays are equivalent to pointers
- “Address-of” (&) can be used on array elements
- …and this is the same as “pointer arithmetic”
- “Address-of” can be used for structs

Pointers can be used as arrays

- When you obtain the address of a variable…

```
ptr = &x;
```

- You can “dereference” it two ways:

```
y = *ptr;   /* treat as pointer   */
z = ptr[0]; /* or 1 element array */
```

- The effect and meaning are exactly the same

Using a pointer as an array

```
#include <stdio.h>

void inc(int *ptr) {
   ptr[0]++; /* use as array */
}

int main() {
   int num = 0;
   inc(&num);  /* pass as a pointer */
   printf("num = %d\n", num);
   return 0;
}
```

Arrays are equivalent to pointers

- When you assign an array (not one of its elements) to something, you're assigning a pointer…

```
ptr = array;
```

- When you pass an array (not one of its elements) to something, you're passing a pointer:

```
strcpy(array1, array2);
```

- When you return an array, you return a pointer…

```
return array;
```

Example

```
#include <stdio.h>

int *zap(int *ptr) {
   ptr[0] = 0;
   return ptr;
}

int main() {
   int array[100];
   int *ptr = 0;
   ptr = zap(array);
}
```

Differences between arrays and pointers

- You can assign something new to a pointer, but an array always points to the same thing…

```
ptr = array;   /* OK */
array = ptr;   /* Not allowed! */
```

- An array definition allocates space for all the elements – but not the “pointer”!
- A pointer definition allocates space only for the pointer value (address).
  - Here, 8 bytes
- A function parameter defined as an array is really just a pointer

Array function parameter

```
int sum(int array[2]) {
  int s = 0;

  for (int i = 0; i < 50; i++) {
    s = s + array[i];  /* legal? */
  }
  return s;
}
```

Address-of can be used on array elements

- Since an array is already an address, it makes no sense to find the address of an array…

```
ptr = &array;  /* Error */
```

- But you can find the address of an element…

```
ptr = &array[3];  /* Great! */
```

A different way to get the address of an element…

- You could also say:

```
ptr = array;  /* Address of array[0] */
ptr = ptr + 3; /* go to 3rd element */
```

- Or even…

```
ptr = &array[0];
ptr++;
ptr++;
ptr++;
```

This is called pointer arithmetic

- You can add or subtract constants:

```
ptr = ptr + 1;   ptr = ptr – 12;
```

- You can increment/decrement:

```
ptr++;   ptr--;   ++ptr;   --ptr;
```

- You can even subtract one pointer from another!

```
int arr[100], *ptr1, *ptr2
long diff;
ptr1 = &arr[10];
ptr2 = &arr[20];
diff = ptr1 – ptr2;
```

- This looks a little dangerous…

  Does your brain hurt yet?

Pointers are dangerous…

- One of the characteristics of a useful computer language is that it should protect the programmer from potential disaster
- C is not like that.
  ”When C is the only gun you have, everything looks like a foot.”
- What happens when you have a pointer problem?

Takehome Quiz #6

1. a. Assuming sizeof(char) is 1, what will the following two lines display to the screen?

```
char name[] = “Artskrut”;
printf(“sizeof = %d\n”, sizeof(name));
```

b. Given the struct declaration and variable definition:

```
struct bug {
  char id;
  short num_legs;
} my_insect;
```

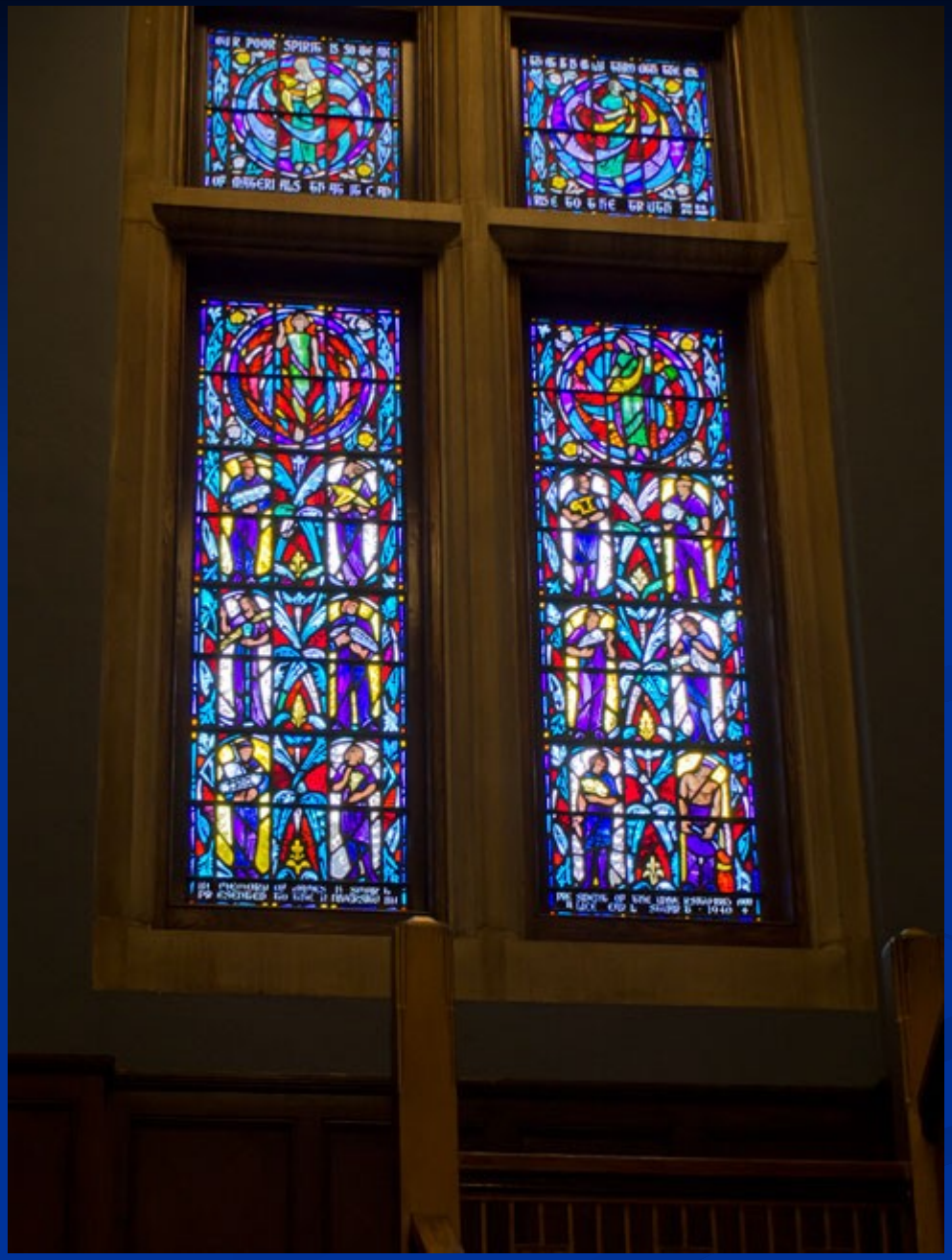
How many bytes will the following fread() call read, if successful?

```
fread(&my_insect, sizeof(struct bug), 3, input_fp);
```

2. Write a function called swap() that takes two arguments – each a pointer to an integer. When called, exchange/swap the two integer values. The function should not return a value.

Purdue Trivia

- PMU includes a stained glass window above the southwest entrance
  - Dedicated to Purdue president James H. Smart (1883 – 1900)
  - Donated by Alice Earl Stuart in 1940
- “Our poor spirit is so weak that it is only through the use of materials that it can rise to the truth”
- Four figures – Mother Earth, Sister Water, Brother Fire, and Brother Wind



When good pointers go bad

```
#include <stdio.h>

int main() {
   int *ptr = 0;
   int array[] = { 5, 6, 7, 8, 9 };

   printf("Before: %d\n", *ptr);
   ptr = &array[2];
   printf("After: %d\n", *ptr);
   return 0;
}
```

How to find the problem?

- You can design your code right in the first place
  ”Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.” – Brian W. Kernighan
- You can carefully examine every statement in your program until you understand what happened.
- You can insert print statements in your code until you narrow down where the problem is.
  - ...and probably `fflush(NULL)` a lot
- Or you can admit defeat and use a debugger

Basic debugger (*nix)

- gdb is the root of all UNIX debuggers
- Very useful in determining where the segmentation fault occurred
  - Not necessarily what caused it

- How to use? Easiest is a 5 step procedure:

```
$ gcc -g file.c -o file # -g flag important!
$ gdb file
(gdb) run  (if problem, will stop at error line)
(gdb) bt (backtrace problem, can provide more info)
(gdb) quit
```

More on gdb

- GDB HOWTO on course website
- https://beej.us/guide/bggdb/

Address-of structures

- You can get the address of anything that stores a value...including a structure:

```
struct coord c = { 5, 12 };
struct coord *p = 0;

p = &c;
(*p).x = 1;
(*p).y = (*p).x;
```

A note about precedence

- It's a little verbose to have to say `(*p).x`
- If the parentheses are omitted, the natural precedence is `*(p.x)` which means something really different
- Wouldn't it be nice if we had an operator that could be used to refer to a field x within a structure pointed to by p???
- It exists! `p->x`

Example

```
#include <stdio.h>

struct coord { int x; int y; };

int main() {
   struct coord c = { 12, 14 };
   struct coord *p = 0;
   p = &c;
   p->x = 4;
   printf("c.x = %d\n", c.x);
   printf("c.y = %d\n", p->y);
   return 0;
}
```

Structures containing pointers

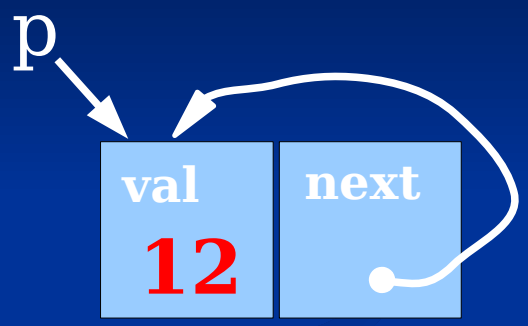
- We mentioned several weeks ago that a structure can contain any definition (except a function)
- A pointer definition can be placed in a structure declaration
- In fact, we can define a pointer to the type of struct that we're presently declaring!

Example of internal pointer…

```
#include <stdio.h>

struct node {
   int val;
   struct node *next;
};

struct node g_node = { 12, NULL };
```

Example continued…

```
void subroutine() {
   struct node *p = 0;
   p = &g_node;

   p->next = p;
   printf("%d\n", p->val);
   printf("%d\n", p->next->val);
   printf("%d\n", p->next->next->val);
}
```

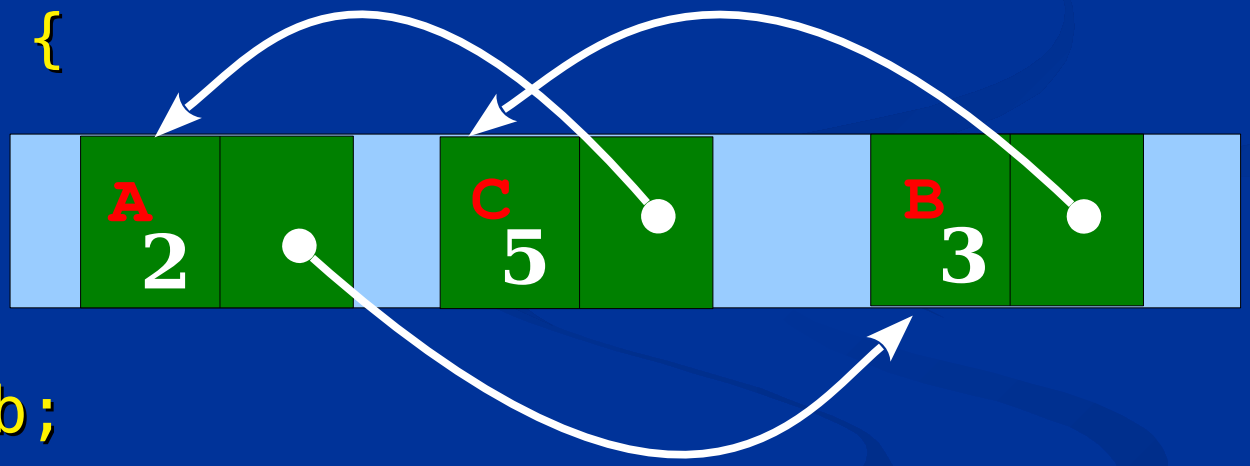


What's the point?

- There's not a lot of use creating a structure that contains a pointer to itself, other than for demonstration
- What if we had several structures?
- What if we set them up to point to each other?
- Better yet, what if we organized them into a list?

Nodes in a ring...

```
struct node a;
struct node b;
struct node c;

void setup() {
  a.val = 2;
  b.val = 3;
  c.val = 5;

  a.next = &b;
  b.next = &c;
  c.next = &a;
}
```



What's the point?

- Still not much use for this except in setting up "state machines"
- We still have the same number of node structures
- What if we could create new node structures dynamically?

End Exam 1 Material

For next lecture

- More pointers, linked lists
- Study the examples in this lecture at home
- Practice the examples
- Modify the examples
- Read Chapter 8.7
  - Chapter 11 in Beej's

Boiler Up!