Welcome to Section 3!

This is CS50.

Pset2 wrap-up

- Compile errors
- Late days
- Design improvements

Last week…

1. **Recursion**

2. **Complexity**

3. **Sorting**

Today

1. Pointers

2. Dynamic Memory Allocation

3. Practice!

Today

1. Pointers

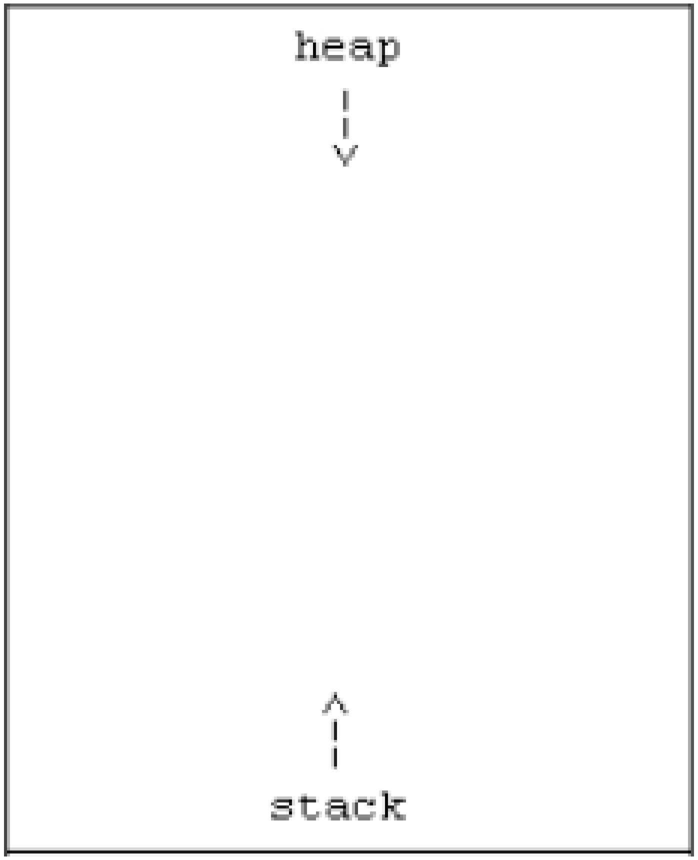
2. Dynamic Memory Allocation

3. Practice!

Memory

Memory - Review

- Stack can smash
- into
- Heap



Memory

- Different types have different sizes in memory
  - int = 4 bytes, char = 1 byte, long long = 8 bytes

- 32-bit machine works with 32-bit addresses
  - 64-bit uses 64-bit address
  - Why is this important?

History…

http://assets.thequietus.com/images/articles/1000/oldcomputer_1232113879_crop_450x411.jpg

Why pointers? (historical)

- C wanted efficiency, control of data storage that existed in assembly language

```
08048940 <_start>:
 8048940:   31 ed                      xor    %ebp,%ebp
 8048942:   5e                         pop    %esi
 8048943:   89 e1                      mov    %esp,%ecx
 8048948:   50                         push   %eax
 8048949:   54                         push   %esp
 804894a:   52                         push   %edx
 804894b:   68 40 95 04 08             push   $0x8049540
 8048950:   68 50 95 04 08             push   $0x8049550
```

Why pointers? (CS50)

- You can use “swap”!
- Allow data structures to be shared
- Dynamic memory allocation

Really understand what’s going on “under the hood”!

Pointers are easy

- Seriously! It’s just boxes and arrows. Learn the syntax and it won’t scare you.

&	(address)
*	(dereference … gets value at an address)
int *p	(declare a pointer to an int)
char *argv[]	(declare a pointer to an array of chars)
int **x	(yikes! a pointer to a pointer to an int)

Creating Pointers

- You've already been using them…
  - In your main's declaration:

```
//char *argv[] is a ptr to an array of chars
int main(int argc, char *argv[])
```

  - Whenever you declare an array

```
//triple is ptr to the first element in array
double triple[3];
```

Creating Pointers

- Declaring
  - `char *p;     //declares ptr to a char`
  - `int *pa, *pb, *pc;     //declares ptrs to ints`

- Careful!
  - `int *px, y, z;`
  - (One pointer to an integer and two integers)

* … gets the value

- The setup…

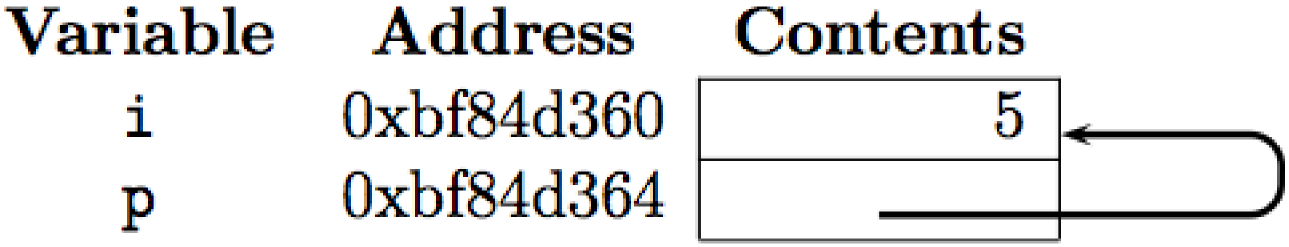
```
int *p;
//(code in between to initialize p)
*p = 1;
```

- Note the difference!
  - (type * name) declares the pointer
  - (*name) "dereferences" … accesses value at location that pointer points to

& … gets the address

```
int i;
int *p;  //p is now defined, but not initialized

i = 5;
p = &i;  //p now points to the location of i
```



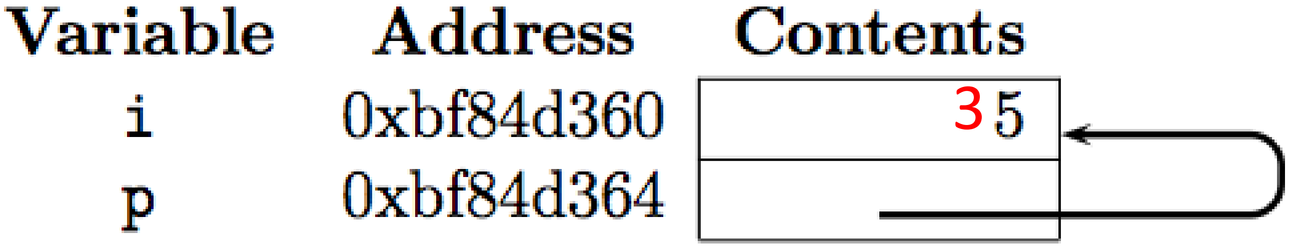
Variable	Address	Contents
i	0xbf84d360	5
p	0xbf84d364	

What if `*p = 35?`

& … gets the address

```
int i;
int *p;  //p is now defined, but not initialized

i = 5;
p = &i;  //p now points to the location of i
```



Variable	Address	Contents
i	0xbf84d360	35
p	0xbf84d364	

What if `p = 35?`

Call by reference

```
void
swap (int *p1, int *p2)
{
  int temp;

  temp = *p1;
  *p1 = *p2;
  *p2 = temp;
}
```

Pointers and arrays

- `int a[3]` declares array of 3 ints
- Just `a` acts like `&a[0]` … a pointer to first element in array

Pointer Arithmetic

- Adding a number to an pointer actually adds based on the size of the pointer’s type
  - int *a … then a + 1 moves 4 bytes because sizeof(int) == 4
  - char *b … then b + 1 moves 1 byte because sizeof(char) == 1
- a[1]?

```
(a + 1) // address of 2nd element
*(a+1) // value of 2nd element
```

- a[n]?

Today

1. Pointers

2. Dynamic Memory Allocation

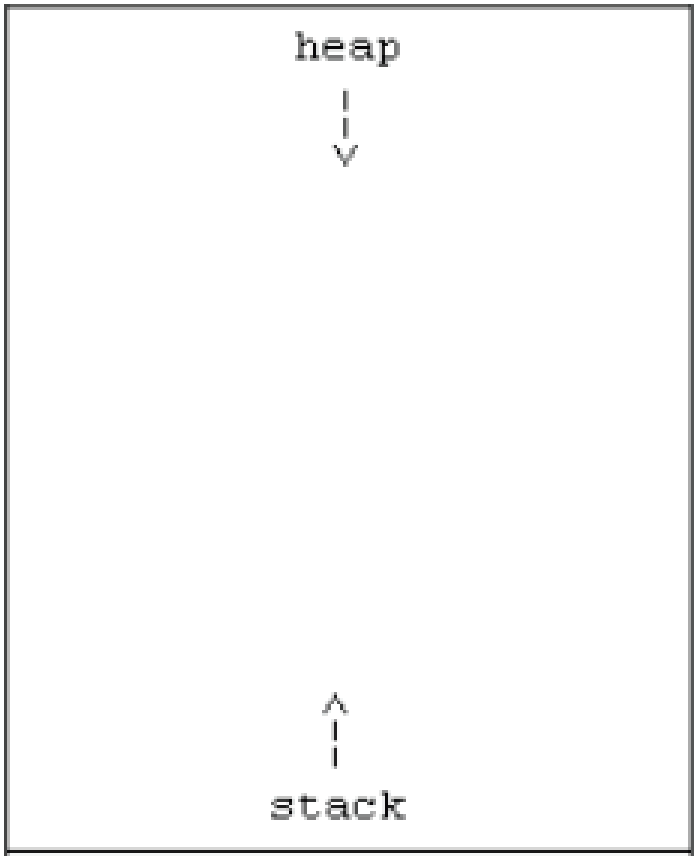
3. Practice!

Malloc

- malloc()
  - Allocates memory to the heap (dynamic! while program is running)
- Related commands
  - sizeof()
  - free()

Memory

- Stack can be smashed by function calls



Malloc

```
int *pa = malloc(sizeof(int))

if(pa == NULL)
{
   printf(“error - out of memory.\n”);
   return 1;
}
```

- Always check pointer returned by malloc, else you risk segmentation faults!

Free

- All memory that you malloc must be freed!
  - Memory leaks: see windows, firefox, word, etc…

- Only free memory that you malloc’ed

Today

1. Pointers

2. Dynamic Memory Allocation

3. Practice!

Today

1. Pointers

2. Dynamic Memory Allocation

3. Practice!

That’s all folks!

http://3.bp.blogspot.com/_xdN0QQwsP1A/Sfspxgi_67I/AAAAAAAABrE/EYSZ_syFwUg/s400/Porky_pig_thats_all_folks.jpg