C Programming
and
Debugging

CS 35L
Spring 2018 - Lab 3

Assignment 7 Reminder Beaglebone Wireless

For assignment 7, you will need a Seeed Studio BeagleBone Green Wireless Development Board

Get it sooner rather than later!

See the specs for assignment 7 for details: https://web.cs.ucla.edu/classes/spring18/cs35L/assign/assign7.html

Lab clarification

- You must specify which ls to use. Only the coreutils-with-bug version of ls will demonstrate the bug.
- "Try to reproduce the problem in your home directory, instead of the $tmp directory. How well does SEASnet do?"
  - Timestamps represented as seconds since Unix Epoch
    - Seconds or nanoseconds elapsed since January 1st 00:00 1970
  - SEASnet NFS filesystem has **unsigned** time stamps
  - Local File System on Linux server (in tmp) has **signed** time stamps
  - If you touch the files on the NFS filesystem it will return timestamp around 2054

Pointers review

- Variables that store memory addresses

- **Declaration:** <variable_type> *<name>;
  - int *ptr;   //declare ptr as a pointer to int
  - int var = 77;   // define an int variable
  - ptr = &var;   // let ptr point to the variable var

(De)Referencing

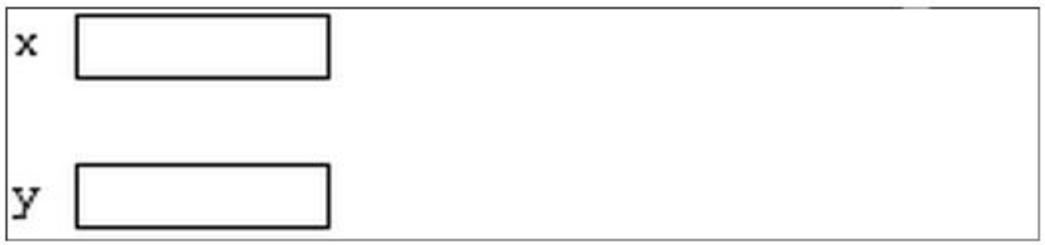
- **Referencing**: get the address of a variable
- **Dereferencing**: getting the value that the pointer is currently pointing to

- Example:
  - double x, *ptr;
  - ptr = **&**x;   //referencing: let ptr point to x
  - *****ptr = 7.8;   //dereferencing: assign 7.8 to x

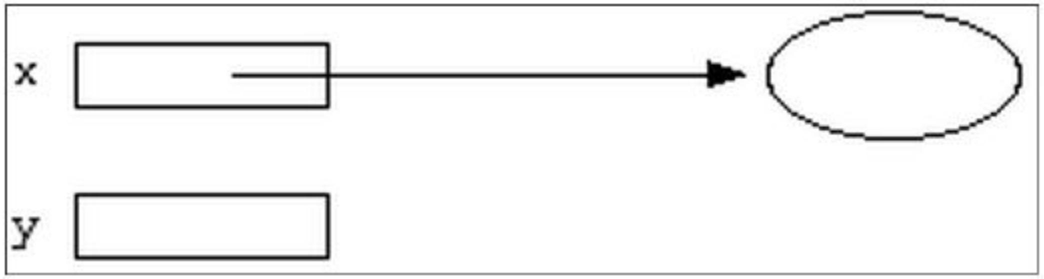
Pointer Example

int *x;

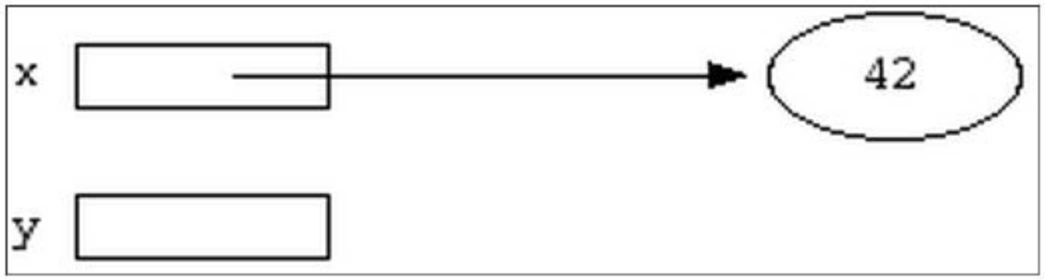
int *y;



int var;   x = &var;

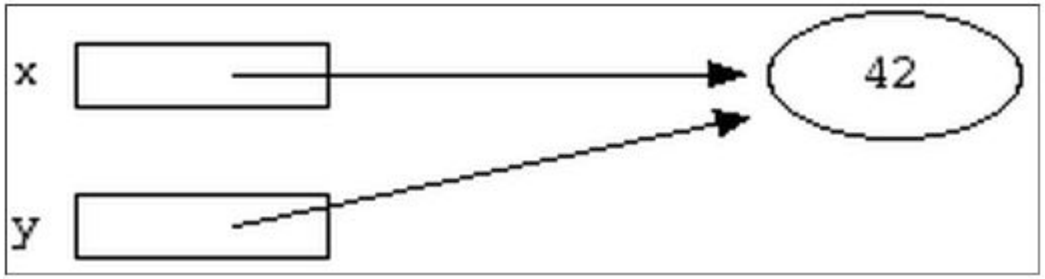


*x = 42;

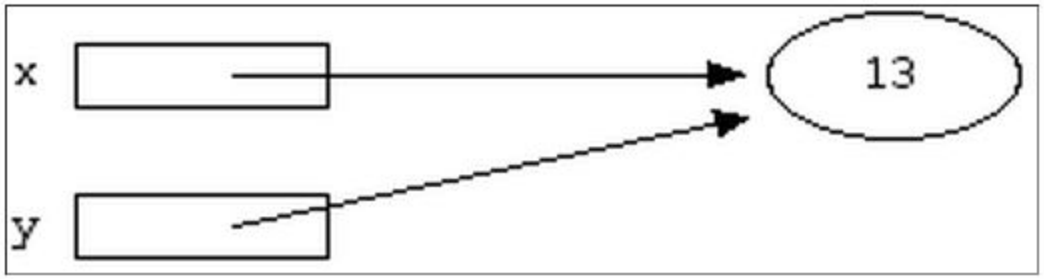


Pointer Example

```
y = x;
```



```
*x = 13;     or
*y = 13;
```



Pointers to Functions

- Also known as: **function pointers** or **functors**
- Goal: write a sorting function
  - Has to work for ascending and descending sorting order + other
- How?
  - Write multiple functions
  - Provide a flag as an argument to the function
  - Polymorphism and virtual functions
  - Use function pointers!!

Pointers to Functions

- Declaration

```
double (*func_ptr) (double, double);
func_ptr = &pow;
func_ptr = pow;
```

- Usage:

```
double result = (*func_ptr)( 1.5, 2.0 );
double result = func_ptr( 1.5, 2.0 );
```

qsort Example

```
int compare (const void * a, const void * b) {
   return ( *(int*)a - *(int*)b );
}

int main () {
   int values[] = { 40, 10, 100, 90, 20, 25 };
   qsort (values, 6, sizeof(int), compare);
   int n;
   for (n = 0; n < 6; n++)
      printf ("%d ",values[n]);
   return 0;
}
```

Dynamic Memory

- Memory that is allocated at runtime
- Allocated on the **heap**

void *malloc (size_t size);

- Allocates *size* bytes and returns a pointer to the allocated memory

void *realloc (void *ptr, size_t size);

- Changes the size of the memory block pointed to by *ptr* to *size* bytes

void free (void *ptr);

- Frees the block of memory pointed to by *ptr*

Valgrind

- Powerful dynamic analysis tool
- Useful to detect memory leaks

Example:

```
$ valgrind --leak-check=full
    ./sfrob < foo.txt
```

```
88 (...) bytes in 1 blocks are definitely lost ...
   at 0x........: malloc (vg_replace_malloc.c:...)
   by 0x........: mk (leak-tree.c:11)
   by 0x........: main (leak-tree.c:25)
```

Homework 4

- Implement a C function **`frobcmp`**
  - Takes two arguments **`a`** and **`b`** as input
  - Each argument is of type char const *
  - **`a`**,**`b`** point to array of non-space bytes
  - Returns an int result that is:
    - Negative if: **`a`** < **`b`**
    - Zero if: **`a`** == **`b`**
    - Positive if: **`a`** > **`b`**
    - Where each comparison is a lexicographic comparison of the unforbnicated bytes

Homework 4

- Then, write a C program called *sfrob*
  - Reads stdin byte-by-byte **(getchar)**
    - Consists of records that are newline-delimited
    - Read until end of file
  - Each byte is frobnicated
    - frobnicated - bitwise XOR (**^**) with dec 42
    - Sort records without decoding (**qsort, frobcmp**)
    - Output in frobnicated text to stdout **(fprintf, putchar)**
  - Dynamic memory allocation (**malloc, realloc, free**)
  - Program should work on empty and large files too

Example 1

- $ cat 'sybjre obl’ > foo.txt
- Input: contents of foo.txt
  - $ ./sfrob < foo.txt
- Read the strings from stdin: sybjre, obl
- Compare strings using *frobcmp* function
- Use *frobcmp* as compare function in *qsort*
- Output: obl sybjre

Example 2

- Input: printf 'sybjre obl’
  - $ printf 'sybjre obl ' | ./sfrob
- Read the strings from stdin: sybjre, obl
- Compare and sort as in example 1
- Output: obl sybjre

Homework Hints

- Assignment 5 **requires** having a solid handle on assignment 4, so this is important!
- Use *exit*, not *return* when exiting with error
- Consider: 1-D vs. 2-D array(s)
- Test output with `od -c` or `od -a` (man od)
- Your code must do thorough error checking, and print an appropriate message on errors.
- Plug all memory leaks! (I'll be checking . . .)