CS 35L

Week 5

TA: Tomer Weiss
Feb-04-2016

goo.gl/8oLMx4

Slides

Announcements

- Student presentations today:
  - Driverless Cars Are Colliding With the Creepy Trolley Problem
- Next week:
  - Write your topic here
  - Not registering you topic beforehand may result in rescheduling of your presentation

Lab clarification

- Test the bug

```
$ touch -d '1918-11-11 11:00 GMT' /tmp/wwi-armistice-yourname
$ touch /tmp/now-yourname
$ sleep 1
$ touch now1-yourname
$ ls -lt /tmp/wwi-armistice-yourname /tmp/now-yourname /tmp/now1-yourname

-rw-r--r-- 1 eggert csfac 0 1918-11-11 11:00:00.000000000 +0000 wwi-armistice
-rw-r--r-- 1 eggert csfac 0 2013-04-29 16:42:48.358233532 +0000 now1
-rw-r--r-- 1 eggert csfac 0 2013-04-29 16:42:47.355197103 +0000 now
```

Lab clarification

- “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 32-bit time stamps
  - Local File System on Linux server has 64-bit time stamps
  - If you touch the files on the NFS filesystem it will return timestamp around 2054
  - => files have to be touched on local filesystem (df –l)

Structs

- No classes in C
- Used to package related data (variables of different types) together
- Single name is convenient

```
struct Student {
      char name[64];
      char UID[10];
      int age;
      int year;
};
struct Student s;
```

```
 typedef struct {
char name[64];
char UID[10];
  int age;
int year;
  } Student;
Student s;
```

C structs vs. C++ classes

- C structs cannot have member functions
- There's no such thing as access specifiers in C
- C structs don't have constructors defined for them

- C++ classes can have member functions
- C++ class members have access specifiers and are **private** by default
- C++ classes must have at least a default constructor

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

Dereferencing Pointers

- Accessing the value that the pointer points to

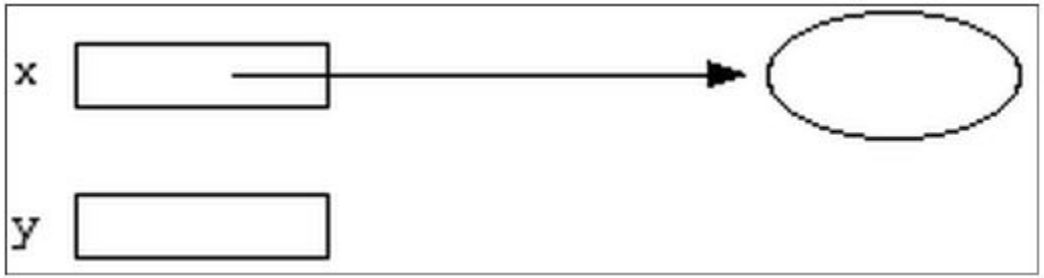
- Example:
  - double x, *ptr;
  - ptr = **&**x; // let ptr point to x
  - *****ptr = 7.8; // assign the value 7.8 to x

Pointer Example

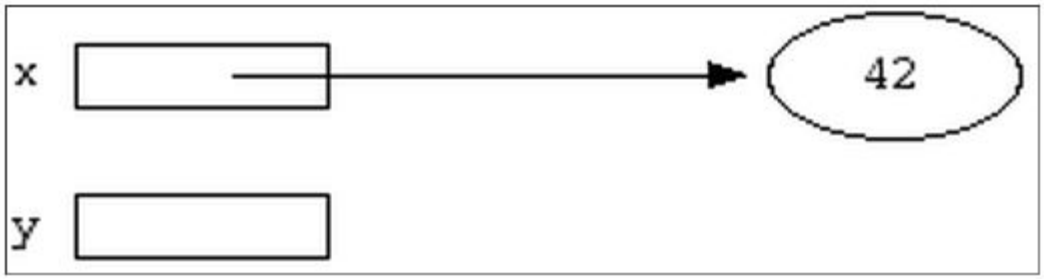
int *x;

int *y;

int var; x = &var;

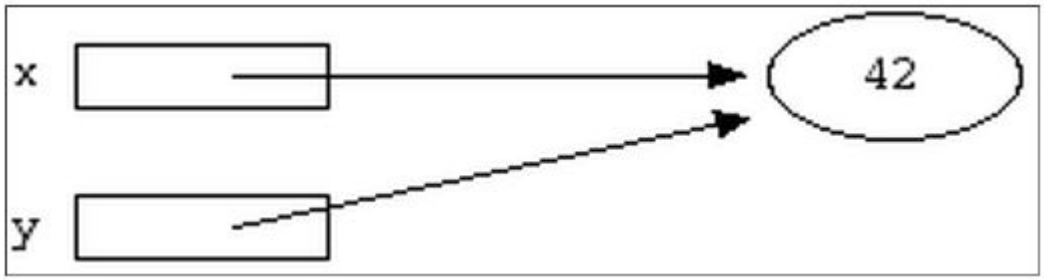


*x = 42;

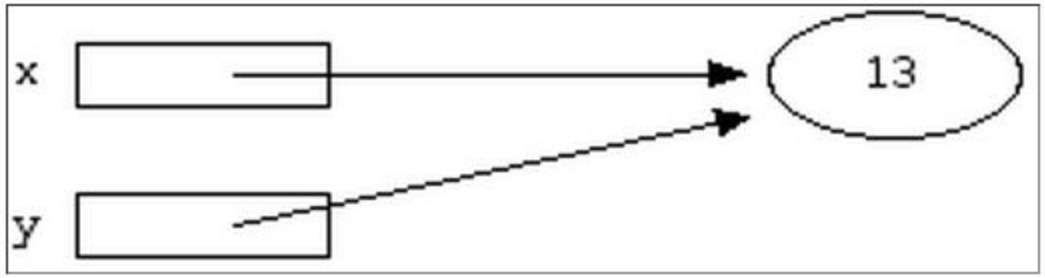


Pointer Example

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

- User can pass in a function to the sort function
- Declaration
  - double (*func_ptr) (double, double);
  - func_ptr = [&]pow; // func_ptr points to pow()
- Usage
  - // Call the function referenced by func_ptr

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

  - // The same function call

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

qsort Example

```
#include <stdio.h>
#include <stdlib.h>

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*

Reading/Writing Characters

- **int getchar();**
  - Returns the next character from stdin
- **int putchar(int character);**
  - Writes a character to the current position in stdout

Formatted I/O

- int fprintf(FILE * fp, const char * format, …);
- int fscanf(FILE * fp, const char * format, …);
  - FILE *fp can be either:
    - A file pointer
    - stdin, stdout, or stderr
  - The format string
    - int score = 120; char player[] = “Mary”;
    - printf(**“%s has %d points.\n”, player, score**);

Homework 5

- Implement a C function frobcmp
  - takes two arguments a and b as input
  - returns an int result that is negative, zero, or positive depending on whether a is less than, equal to, or greater than b. Each argument is of type char const *.
  - a,b point to array of non-space bytes

Homework 5

- 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 (XOR with dec 42)
    - frobnicated - encoded with memfrob
    - Sort records without decoding (**qsort, frobcmp**)
    - Output result in frobnicated encoding to stdout **(putchar)**
  - Error checking (**fprintf**)
  - Dynamic memory allocation (**malloc, realloc, free**)
  - Program should work on empty and large files too

Example

- Input: printf 'sybjre\nobl’
  - $ printf 'sybjre\nobl**\n**' | ./sfrob
- Read the records: sybjre, obl
- Compare records using *frobcmp* function
- Use *frobcmp* as compare function in *qsort*
- Output: obl

  sybjre

Homework Hints

- Start as soon as possible
- Use ***gdb***
- Use *exit*, not *return* when exiting with error
- 1-D vs. 2-D array
- Test your code with od –c
  - what is od?
    - man od

Lab

web.cs.ucla.edu/classes/winter16/cs35L/assign/assign5.html