PURDUE
UNIVERSITY®

CS 240: Programming in C

Lecture 20: Callbacks
Midterm 2 Review

Prof. Jeff Turkstra

Announcements

- Midterm Exam 2 Thursday!
  - Check the seating charts!
  - 8:00pm – 10:00pm
  - Do not enter the room until told to do so
- Homework 9 due tonight

4/9 Lecture

- We will have lecture next Wednesday
- Instead will not have lecture on 4/21
    - An important appointment came up that I cannot miss

Feasting with Faculty

- Tomorrow at 12pm in Earhart!
- Don’t have to wait for an invitation
- Can come if you’ve already been invited
- Can come again if you’ve already attended before

Homework 6

```
597 scores total…
100+:  (0)
 100: ====================== (331)
  90: == (23)
  80: === (34)
  70: === (33)
  60: ==== (49)
  50: === (33)
  40: == (32)
  30: = (10)
  20: = (6)
  10: = (2)
   0: === (44)
Average: 79.6298157453936348
```

Homework 7

```
595 scores total...
100+:  (0)
 100: ===================== (343)
  90: ==== (52)
  80: ==== (54)
  70: ==== (56)
  60: == (27)
  50: = (5)
  40: = (6)
  30: = (7)
  20: = (3)
  10: = (3)
   0: === (39)
Average: 84.8050420168067227
```

Homework 8

```
595 scores total...
100+:  (0)
 100: ====================== (471)
  90: == (45)
  80: = (10)
  70: = (8)
  60: = (5)
  50: = (4)
  40: = (4)
  30: = (4)
  20:  (0)
  10: = (1)
   0: == (43)
Average: 90.0453781512605042
```

The void type

- There is a type in C that represents nothing
- It is used in only two cases:
  - To represent a function that has no return value:

```
void no_value(int x) {
  printf(“Value is %d\n”, x);
  return;
}
```

  - A pointer to something opaque:

```
void *pointer = NULL;
int *i_ptr = NULL;
int *i_arr = malloc(sizeof(int) * 15);
pointer = i_arr;
i_ptr = (int *) pointer;
```

What you can do to a void *

- You can assign any pointer type to a void * variable without a cast
- A void * type will hold (almost) any other first-class data type
  - E.g., double, int, long
  - This isn't guaranteed to be portable
- You can later assign the void * type to a usable type again with a cast
- You may not dereference a void * type
- You should not perform pointer arithmetic on a void * type

When to use void *

- Use the void * type to serve as a conveyor of opaque data or data whose type is not yet known
- Example: our friend, the free() function:
  `void free(void *ptr);`
  - free() does not care what type of pointer we pass it. It only needs to know where it points to.
  - This allows you to free any type of pointer

Another application: callbacks

- Suppose I set up some kind of function that accepted a pointer to a function and a value to pass to that function:

```
void setup_cb(void (*callback)(int),
              int callback_value) {
  callback(callback_value);
}
```

- This function allows the user to pass a function to call and the integer value to call it with
  - What if we wanted to use more than integers?

Generalize callback arguments using void *

- Change the functions to use void * instead...

```
void setup_cb(void (*callback)(void *),
              void *callback_value) {
  callback(callback_value);
}
```

- Now we can pass various pointer types in addition to integers and other first-class types

A generic mechanism to run something periodically…

```
#include <signal.h>
#include <sys/time.h>

void *callback_data;
void (*callback)(void *);

void signal_handler(int x) {
  callback(callback_data);
}
void setup_timer(int rate, void (*cb)(void *),
                 void *cb_data) {
  struct itimerval i = { {rate, 0}, {rate, 0} };
  callback = cb;
  callback_data = cb_data;
  setitimer(ITIMER_REAL, &i, NULL);
  signal(SIGALRM, signal_handler);
}
```

And something to use it…

- Now we have a main() function that demonstrates it…

```
void print_msg(void *arg) {
  char *msg = (char *) arg;
  printf("%s\n", msg);
}

int main() {
  setup_timer(1, print_msg, “Sample Message”);
  while (1);
}
```

Full example of a callback

- In this example, we set up a “clock” structure and then use an asynchronous callback mechanism to update it:

  ```
  struct clock {
     volatile char hours;
     volatile char minutes;
     volatile char seconds;
  };
  ```

- Then we define a routine used to update it…

update_clock()

```
void update_clock(void *v_ptr) {
  struct clock *c_ptr = (struct clock *) v_ptr;
  c_ptr->seconds++;
  if (c_ptr->seconds == 60) {
    c_ptr->seconds = 0;
    c_ptr->minutes++;
    if (c_ptr->minutes == 60) {
      c_ptr->minutes = 0;
      c_ptr->hours++;
      if (c_ptr->hours == 13) {
        c_ptr->hours = 1;
      }
    }
  }
}
```

And something to use it…

- Now we have a main() function that sets everything up and demonstrates it…

```
int main() {
  struct clock *clk = NULL;
  clk = calloc(1, sizeof(struct clock));
  setup_timer(1, update_clock, clk);
  while (1) {
    printf(“Hit return!”);
    getchar();
    printf(“Time: %02d:%02d:%02d\n”,
           clk->hours, clk->minutes,
           clk->seconds);
  }
}
```

Purdue Trivia

- The Purdue Exponent was established on December 15, 1889
  - Student organization until 1969
  - Now one of a handful of independent student newspapers
  - Run their own printing press
  - Indiana’s largest collegiate newspaper
  - Alumni have won six Pulitzers, six Emmys, and two Peabodys

Midterm 2

- Thursday, April 10
  - 8:00pm – 10:00pm
  - New seating charts
- Bring your Purdue ID (optional)
- Bring a pencil
- Bring nothing else
- Seating chart soon

Midterm 2

- Look at and understand the example questions
- Review your homeworks
  - Write them out on paper
  - Diagram the data structures
  - Understand them
- Take the sample exam
  - Time yourself
  - Review your answers

Midterm 2 topics

- Pointers! (surprise!)
  - Obtaining the address of variables (&)
  - Dereferencing (getting contents of) pointers (*)
  - Using pointers as arrays
  - Pointers to array elements
  - Pointer arithmetic
  - Passing variables by pointer
- Debugging
  - Approaches, gdb

Midterm 2 topics (cont)

- Dynamic memory allocation
  - malloc(), calloc()
  - free()
- Pointers to structures
  - Use of the -> operator
  - Linked lists (singly-linked lists) and operations
  - Doubly-linked lists and operations
  - Trees and operations

Midterm 2 topics (cont)

- Pointers to pointers
  - Re-writing list operations to use pointers to pointers
- Pointers inside structures (internal pointers)
  - E.g.: structure fields that point to dynamically allocated strings
- Pointers to functions
  - Passing a function name as an argument
  - Calling a passed function within a function
- Recursion

Midterm 2 topics (cont)

- Multidimensional, dynamically allocated arrays
- Types
  - Qualifiers, storage classes
- C Preprocessor

For next lecture

- Efficiency
- Libraries
- Large-scale development

Boiler Up!