Pointers, Structures, and Arrays

CS 350: Computer Organization & Assembly Language Programming

Due Fri Nov 15

A. Why?

- Pointers are an efficient way to share large memory objects without copying them.
- In C, pointers are used to simulate call-by-reference, and array references can be written as pointer dereferencing operations (and vice versa).
- In C, structures define data records (but don't support constructors, methods, inheritance, or interfaces).

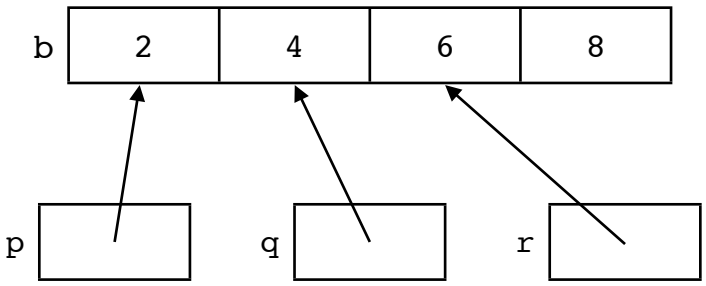
B. Outcomes

After this lecture lab, you should

- Take a C expression or assignment that uses arrays, pointers, and structures and determine its value or action given a state of memory.
- Translate between C code that uses array references and pointer references.

C. Written Problems [65 points total]

1. [15 pts] Write some C declarations and code to establish the memory diagram below. (There are multiple right answers.) `p`, `q`, and `r` should be pointers to integers.



2. [15 = 5 * 3 pts] Using the memory diagram for Problem 1, answer the following question for each of the expressions below: Does it cause a compile-time warning or error (and if so, which one), or does it cause a runtime error (and if so, which one), or does it evaluate to true or false?

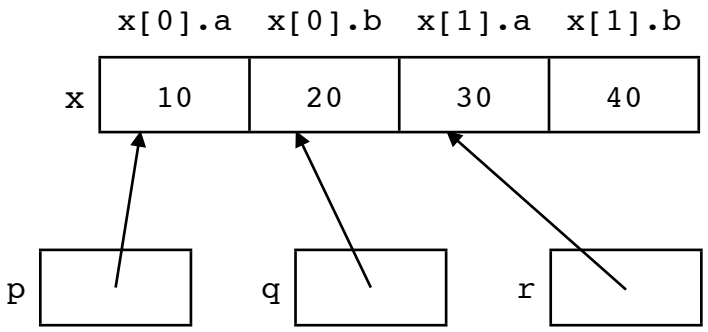
   (a) `p == b` (b) `q == b+1` (c) `q == (&b)+1` (d) `*q == *(r-1)` (e) `p[1] == r[-1]`

3. [20 = 2 * 10 pts] Consider the C declarations and code below. (a) Draw a memory diagram that shows the state at position 1. (b) Draw a memory diagram that shows the state of memory at position 2.

```
int b[4] = {12, 13, 14, 15};
int u = 20, v = 30, *x = &u, *y, *z;
y = &u;
z = &b[2];
// <----- Position 1
++ *x;  // (i.e., *x = *x + 1)
y = &v;
--z;
z[1] = 20;
// <----- Position 2
```

4. [15 = 5 * 3 pts] The code below declares an array of `Pairs` `x`, a pointer-to-`Pair` `p` and pointers-to-int `q` and `r`, and it uses assignments to establish the memory diagram below.

```
typedef struct {int a, b;} Pair;
Pair x[2];
Pair *p;
int *q, *r;
x[0].a = 10;
x[0].b = 20;
x[1].a = 30;
x[1].b = 40;
p = &x[0];
q = &x[0].b;
r = &x[1].a;
printf("%d\n", /* See expressions below */);
```



For each of the expressions below, what would happen if we use it as the expression in the `printf` statement above? Would it cause a compile-time warning or error (and if so, which one)? Or would it cause a runtime error (and if so, which one)? Or would it simply evaluate to true or false? (Hint: Try typing the code into a file and compiling and running it.)

(a) `p->a + p->b == x[1].a`

(b) `q == p+1`

(c) `&x[1] == p+1`

(d) `&(x[0].b) == &(x[0].a)+1`

(e) `r == x[1].&a`

D. Programming Problem: Return of the SDC Simulator[1]!! [35 points]

- For this lab, you are to rewrite your solution to Lab 9 (the full SDC simulator) using a `CPU` structure and pointers instead of global variables.
- The skeleton file `Lab11_skeleton.c` declares a `CPU` structure; the main program creates a `CPU` value and a pointer to it. To call a routine that uses the `CPU`, the we pass the pointer as an argument.

```
CPU cpu_value;
CPU *cpu = &cpu_value;
initCPU(cpu);
```

- When declaring the routine, we include the `CPU` pointer as a parameter. In the body of the routine, we access the appropriate `CPU` field using `cpu->ir`, `cpu->pc`, `cpu->reg[`*regnbr*`]`, etc., instead of using the global variables `ir`, `pc`, etc., we used in the earlier lab.

```
void init_CPU(CPU *cpu) {
    …
    cpu->pc = 0;
    …
}
```

- You should be able to take your Lab 9 solution and convert it to use the `CPU` structure fairly straightforwardly.

---

[1] Just when you thought it was safe to sit down in front of your laptop ….

- Your program for this lab should behave just like your program for Lab 9 (unless you had bugs in your Lab 9 solution :-)
- **Point breakdown**: 15 points for a program that uses the CPU structure (and has no syntax errors); 15 points for program correctness; 5 points for commenting and code structure.

