Pointers, Structures, and Arrays

CS 350: Computer Organization & Assembly Language Programming
Due Fri Apr 25

A. Why?

- Pointers are an efficient way to share memory objects without copying them.
- In C, structures define data records (but don't support constructors, methods, inheritance, or interfaces).

B. Outcomes

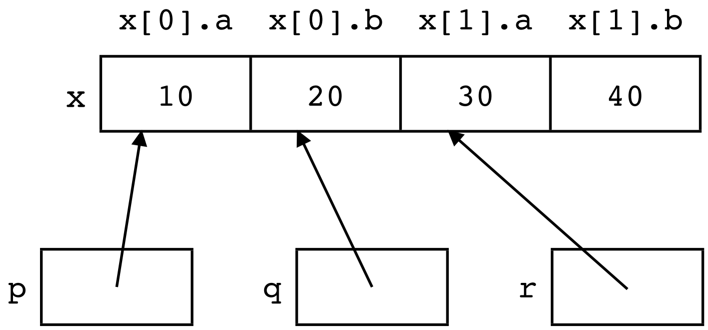
After this lecture lab, you should

- Take a C expression or assignment that uses arrays, pointers, and structures and determine its value or action given a state of memory.

C. Written Problems [60 points total]

1. [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 might 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`

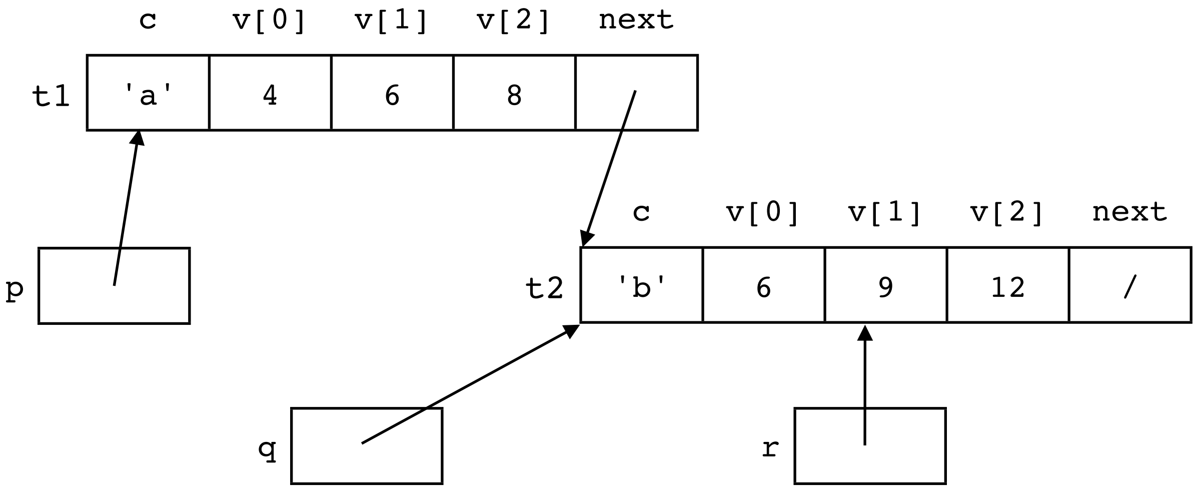
(b) `q == p+1`

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

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

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

2. [30 pts] First, study the following memory diagram. It shows two nodes of a singly-linked list, with pointers to the two nodes and a pointer to an array element within one of the nodes.



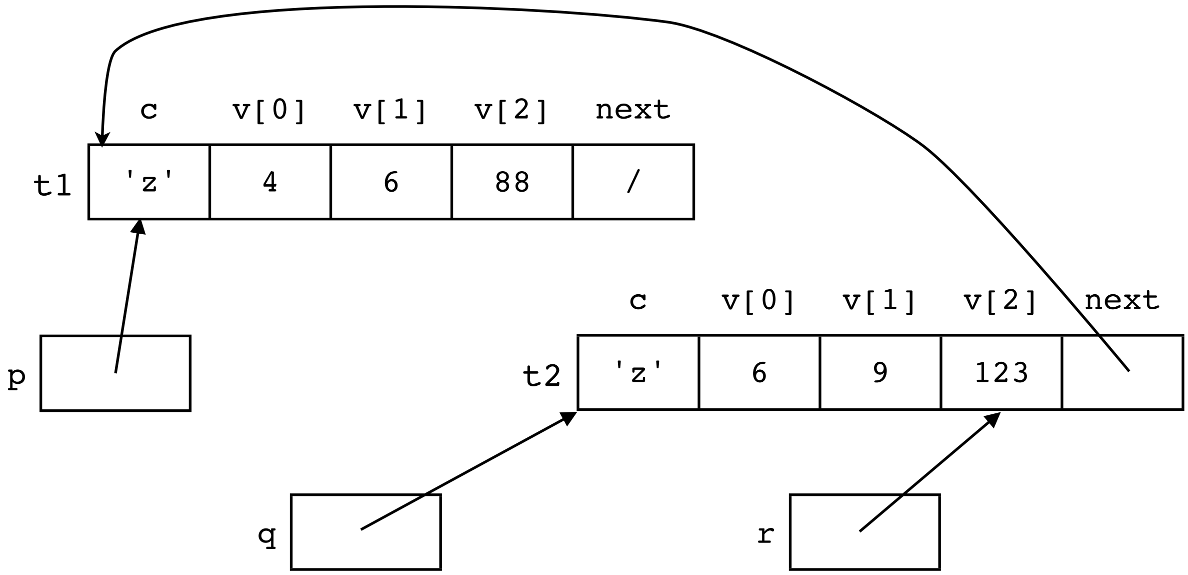
Complete the code below so that at `/* here */`, it establishes the memory diagram. There are multiple right answers, and it's possible to write the code so that `/* part 2 */` is empty.

```
typedef struct Node {
   char c;
    int v[3];
   struct Node *next;
} Node;

main() {
    Node t1, t2, *p, *q;
    int *r;
```

```
        /* Part 1 (replace this with code) */
        p -> next = q;
        q -> next = NULL;
        t1.c = 'a';
        /* Part 2 (replace this with code) */
        /* Here */
    }
```

3. [15 pts] Write code to take the memory diagram of the previous problem to the diagram below. Again, there are multiple right answers.



D. Programming Problem: Return of the SDC Simulator[1]!! [40 points]

- For this lab, you are to rewrite the 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);
```

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

- 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 earlier simulator and convert it to use the `CPU` structure fairly straightforwardly.
- Your program for this lab should behave just like your earlier simulator (unless it had bugs :-)
- **Point breakdown**: 15 points for a program that uses the `CPU` structure (and has no syntax errors); 20 points for program correctness; 5 points for commenting and code structure.