CS 430 Types

Part 1: Pointer Types

Consider the following C program:

```
int main() {
    int a = 123;
    int *b = &a;
    int **c = &b;
    // location A
    *b = 321;
    printf("%d %d %d\n", a, *b, **c);
    return 0;
}
```

1. Draw a picture of stack memory at location A. Label memory locations with their C variable names, and draw pointers as arrows pointing to their targets.

2. What is the name of the "*" operator in C (e.g., on the two lines after location A), and what does it do?

3. How does your diagram change after the line following location A? What is the program's output?

4. Load PythonTutor.com/visualize.html and enter the above code (or use this shortcut: https://goo.gl/vm8BVg). Run the code (make sure you're in C mode) and confirm your answers to #1 and #3 above.

Consider the following C program:

```
int main() {
    char msg[] = "Hello!";
    msg[1] = 'o';
    char *p = msg;
    // location B
    *p = 'P';
    p++;
    *(p+4) = '?';
    printf("%s\n", msg);
    return 0;
}
```

5. Draw a picture of stack memory at location B. Label memory locations with their C variable names, and draw pointers as arrows pointing to their targets. How does your diagram change after each of the three lines following location B? What is the program's output?

6. Confirm your answers to #5 using PythonTutor. (shortcut: https://goo.gl/W5HsnH)

Part 2: Heap Pointers

Consider the following C program:

```
int main() {
    int *a = (int*)malloc(5*sizeof(int));
    int b = 2;
    int *p = &a[1];
    // location C
    a[0] = 777;
    a[b] = 45;
    *p = 9;
    p[2] = 12;
    p += b;
    *(p+1) = *a;
    // location D
    free(a);
    printf("%d\n", *p);
    return 0;
}
```

7. Draw a picture of **both stack and heap** memory at location C. How does your diagram change after each of the six lines following location C? What is the contents of the memory allocated on the first line of code after execution reaches location D? What is the program output?

8. Confirm your answers to #7 using PythonTutor. (shortcut: https://goo.gl/ZYqojG)

9. Does the program leak memory? Are there any dangling pointers?

10. Describe how the *locks & keys* approach described in CPL section 6.11.8.2 would help deal with a problem identified in #9.

11. Describe how *mark-sweep garbage collection* would help deal with a problem identified in #9.

12. Describe how *reference counters* would help deal with a problem identified in #9. How many references are there to each 4-byte memory cell allocated on the heap at location C?

13. What is a *reference type* and how is it different from a *pointer type*? Does C have reference types, pointer types, or both? What about C++, Java, and Ruby?

Part 3: Struct and Union Types

Consider the following C program:

```
struct {
  int a;
  char b;
  double c;
} v;

int main() {
    v.a = 7;
    v.b = 'x';
    v.c = 1.23;
    // location E
    printf("%d %c %f\n", v.a, v.b, v.c);
    return 0;
}
```

14. Draw a picture of memory at location E and confirm your answers using PythonTutor. (shortcut: https://goo.gl/qbUpVy)

15. What is the minimum number of bytes necessary to store the struct variable ("v") in the above program?

16. Suppose that on a particular system there are X different possible integers, Y different possible characters, and Z different possible double-precision numbers. As a function of X, Y, and Z, what is the total number of different possible values for the struct variable ("v")?

17. Change "`struct`" to "`union`" in the first line. How does this change what is stored in memory? How does it change the output? What is the minimum number of bytes required to store the union variable? As a function of X, Y, and Z (as defined in #15), what is the total number of different possible values for the union variable ("v")?

NOTE: PythonTutor doesn't appear to have support for union types, so if you wish to test your answers for #17 you will have to compile and run the code yourself.

BONUS: Trace the code and draw memory for the following programs:

- https://goo.gl/ohJ1DB
- https://goo.gl/RTimVH
- https://goo.gl/JNDZdD
- https://goo.gl/UnqT8V
- https://goo.gl/eYMmHQ
- https://goo.gl/DTdy2j

Part 4: Type Equivalence

Suppose the following declarations have been made in a C-like language. For each context, circle all of the assignments that are valid, and **cross out all that are not valid**.

```
typedef float inches;
typedef float feet;
typedef struct { inches x; feet y; } box;
typedef struct { inches x; feet y; } bin;

inches a, b;
feet c;
float d;
struct { inches x; feet y; } m, n;
box o;
bin p;
```

18. Assume assignments require name equivalence unless at least one type is anonymous, in which case they only require structure equivalence.

`a = b`	`a = c`	`a = d`	`a = m`
`m = n`	`m = o`	`p = o`	`m.x = d`
`m.x = a`	`o.x = a`	`m.x = m.y`	`m.x = o.x`

19. Assume assignments only require structure equivalence, regardless of type aliases.

`a = b`	`a = c`	`a = d`	`a = m`
`m = n`	`m = o`	`p = o`	`m.x = d`
`m.x = a`	`o.x = a`	`m.x = m.y`	`m.x = o.x`

20. Assume assignments require name equivalence for primitive types and their aliases but permit structure equivalence for non-primitive types.

`a = b`	`a = c`	`a = d`	`a = m`
`m = n`	`m = o`	`p = o`	`m.x = d`
`m.x = a`	`o.x = a`	`m.x = m.y`	`m.x = o.x`