Assignment 5

Aleck Zhao (azhao9)
Noah Halpern (nhalper1)

October 10, 2016

Numeric Representations

Convert the following decimal numbers to binary:

1. 3

   Answer. 11

2. 19

   Answer. 10011

3. 372

   Answer. 101110100

4. 2048

   Answer. 100000000000

Convert the following binary numbers to decimal:

5. 1011

   Answer. 11

6. 11000

   Answer. 24

7. 10101010

   Answer. 170

8. 10000000001

   Answer. 1025

Convert the following decimal numbers to hexadecimal:

9. 7

   Answer. 7

10. 12

    Answer. C

11. 97

    Answer. 61

12. 256

    Answer. 100

Code Puzzles

13.

Answer. This prints out

6 24 49 1 -18

14.

Answer. This prints out

```
11  2  3
4  17  6
7  24  9
```

Code Tracing

15.

Answer. This prints out

sdrawrof

16.

Answer. This prints out

hat the in cat the

Code Writing

17.

```
int* copyArray(const int *orig, int size) {

    int* new = malloc(size * sizeof(int));

    for (int i = 0; i < size; i++) {
        *(new + i) = *(orig + i);
    }

    return new;
}
```

18.

```
int isSymmetric(const int *values, int nElements) {

    if (nElements == 0 || nElements == 1) {
        return 1;
    }

    if (values[0] == values[nElements - 1]) {
        return isSymmetric(values + 1, nElements - 2);
    } else {
        return 0;
    }
}
```

Explanation

19. Suppose i is an int variable, and p and q are type point to int variables. For each expression, either explain what the given statement accomplishes, or indicate that the given statement is illegal, and why.

    (a) p = &i;

    Answer. Since p is a pointer type variable, and i is an int variable, &i is the address of an int variable. This sets the value of p to the address of i.

    (b) *p = i;

    Answer. p is an integer type pointer, so by dereferencing it with *p, we set the value of the integer that p points to. This is equal to i.

    (c) *q = &i;

    Answer. This is illegal. *q is the dereferenced pointer, so it is of type integer. However, because i is an int, &i is of type integer pointer, and this is an illegal assignment.

    (d) p = q;

    Answer. Makes the value of p the value of q. Now p points to what q points to.

20. At the end of the following snippet of code, indicate how many bytes of dynamically allocated memory are allocated to the program, and which pointers can safely be dereferenced. Assume that sizeof(int) is 4 bytes, and no allocations failures occur.

```
int size = 10;
int *a = malloc(size * sizeof(int));
int *b = realloc(a, size * 2 * sizeof(int));
```

    Answer. The size of int is 4 bytes, so the first *a allocates a block of memory of size $10 * 4 = 40$ bytes. Then realloc returns a new pointer to a block of memory $10 * 2 * 4 = 80$ bytes, and copies everything from a to b. Since this new block is larger, it is a new spot in memory, and the previous pointer a is freed. Thus at the end, there are 80 bytes of dynamically allocated memory.

21. Explain what the keyword const means in the prototypes for the functions in the **Code Writing** above, and why it makes sense to have it there.

    Answer. The keyword const means that the variable cannot be changed within the function. If we do not wish for the variable to be changed within the function, it is desirable to put this within the prototype because if it is inadvertently changed, we will get a compiler warning. This makes it easier to debug.

22. Explain why types in C are not guaranteed to be the same size on all platforms (eg an int may be 4 bytes on a modern computer, but only 2 bytes on an older one). This should be an explanation of why this choice was made (since it was an intentional choice by the language designers), not just an explanation of what happens.

    Answer. The language designers did not set the size of ints and other types. Certain hardware architectures might benefit from independently defining the sizes of these types, and the designers of C did not want to artificially constrain these hardware from operating optimally.

23. Explain *what* incremental development is, and *why* we use it.

    Answer. Incremental development is creating a program or programs step by step. The benefit of this is that it makes debugging between steps easier, since we can focus on getting one part right before continuing. It allows the programmer to gradually build up to a fully functional program, and is convenient for version control. If there is a working version that was committed, if the next step breaks the program, we can revert to the previous version and not lose anything.

24. Explain why pointers and dynamic allocation are exposed to the programmer in C; how does this choice support the design goals of the C language?

    Answer. Pointers and dynamic allocation are exposed to the programmer in C because the C language was intended to interact with hardware. Directly manipulating memory addresses and directly allocating memory are allowed by this functionality of C, and allow the programmer to do more things. Since C was meant to write operating systems, it is essential that everything runs quickly, and the programmer has liberty to optimize things.

25. Explain why pointers and dynamic allocation are hidden from the programmer in languages like Java and Python; how does this choice support the design goals of these languages?

    Answer. Java and Python are intended to be user friendly languages. Pointers and dynamic memory allocation create opportunities for more bugs and segmentation faults and memory leaks, so the designers of Java and Python kept these hidden from the programmer. These languages were meant to write programs and scripts, not operating systems, so there is no need to directly interact with hardware.