Computer Science 50 **Quiz 0 Practice**

TF: Samuel Green
samuelgreen@college.harvard.edu

October 2015

N.B. This practice quiz is my best guess at what a quiz might look like. I haven't seen Quiz 0, so no guarantees! I'd recommend taking this quiz in the same ~90 minute controlled setting as the real quiz, but also be warned that it may be shorter or longer than the real quiz. It would also be most helpful after you've prepared your one page front-and-back "cheatsheet".

SOLUTIONS LISTED BELOW QUESTIONS

1 Lightning Round

1.1 Short Answer

(A) What's decimal 51 in binary? How about hexadecimal?

```
110011
0x33
```

(B) How many bytes does an `int` occupy? How many bits?

4 bytes, 32 bits

(C) What does it mean to compile code? What are the steps?

Compiling code is the process of turning an text file containing human readable code into a binary file containing machine-executable code. The steps are:
Preprocessing
Complilation
Assembly
Linking.

See Rob's short on compiling for more info.

(D) Who's the best TF in all the land?

Idk my bff jill?

1.2 0 or 1

Please answer the following questions, justifying your answer in the space provided.

(A) A string is really an array.

TRUE. Recall that the `string` datatype is the CS50 library's wrapper around `char*`, and that we can change the values in each "box" in the char just as we would an array. Can you think of any special properties of `char*` strings, however?

(B) The `float` datatype can store infinitely precise decimal numbers.

FALSE. Recall floating point imprecision.

(C) Binary search is always better than linear search.

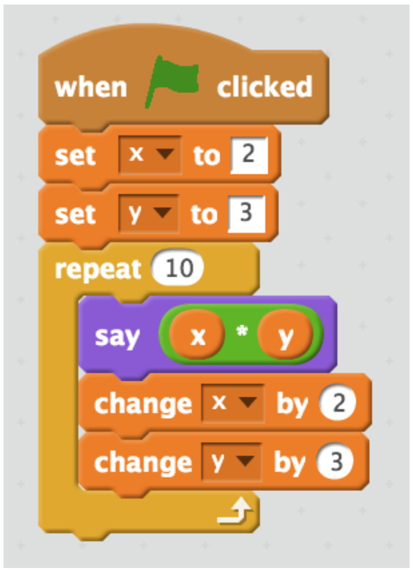
FALSE, for several reasons. First, the data we're searching in may not be sorted, in which case we have sorting time to consider. Linear search can also outperform on sorted data if we search extremely frequently for "low" values and very infrequently for "high" values.

(D) Running time is all that matters when considering the performance of an algorithm.

FALSE. We might also consider space required (all of the extra stack frames or array space for merge sort) or preprocessing to make the algorithm work (like sorting before binary search) in considering the merits of an algorithm.

2 Fun with Scratch

Take a look at the Scratch script below:



In the space below, write a C program that creates "equivalent" output, where `say` is equivalent to `printf` .

```
#include <stdio.h>

int main(void){
  int x = 2;
  int y = 3;

  for (int = 0; i < 10; i++){
    printf("%i\n", x * y);
    x += 2;
    y += 3;
  }
}
```

3 Things that Divide Us

I really enjoy random mathematical facts. As part of my fascination, I have a C program that really, really needs to know some stuff about multiples. Can you help me out?

3.1 Three's Plenty

Complete the function below so that it returns `true` if `n` is a multiple of three, `false` otherwise.

```
bool isMultThree(unsigned int n){
  return n % 3 == 0;
}
```

3.2 More's a Crowd

Turns out, I need a more general function after all.

Complete the function below so that it returns true if `n` is a multiple of `f`.

```
bool isMult(unsigned int n, unsigned int f){
  return n % f == 0;
}
```

4 Uppers and Lowers

You're working on a super cool program called initials, which you hope will let users easily find out what their initials are. For some reason, though, you can't remember the header file in which some handy character functions live, so you've decided to write them again.

Fill in the functions below, maximizing code reusing whenever possible. You may assume that all necessary headers have been included (except the one containing these functions!).

```
bool isupper(char c){
  return 'A' <= c && 'Z' >= c;







}

bool islower(char c){
    return 'a' <= c && 'z' >= c;







}

bool isalpha(char c){
    return isupper(c) || islower(c);







}
```

5 What you doin' in the club on a Thursday?

(A) I can't remember the upper and lower bounds on the running times of my favorite algorithms. Remind me by filling in the following table based on algorithms we have seen in class. Be careful not to reuse any algorithm!

Algorithm	Ω	O
Linear Search	1	n
Merge Sort	$n \log n$	$n \log n$
Bubble Sort	n	n^2
Selection Sort	n^2	n^2
Insertion Sort	n	n^2

(B) What're the lower and upper bound on the running time of binary search? Why does it have those bounds?

- Lower bound $\Omega(1)$
- Upper bound $O(\log n)$
- Constant time lower bound comes from finding the desired value "on the first check", and $\log n$ bound comes from dividing the problem in half with every step.

(C) Would we ever choose an algorithm with a higher upper-bound on its running time over a similarly effective algorithm with a lower upper-bound? Why or why not?

There are several good answers to this question. You might discuss space considerations, preprocessing steps, and the fact that upper-bound running times don't *necessarily* reflect average running times.

6 Stop Copying Me!

Consider the program below:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define ALPHA 26

void change(char* str){
    for (int i = 0, n = strlen(str); i < n; i++){
        if (islower(str[i]))
            str[i] = (str[i] % 'a' + 1) % ALPHA + 'a';
        else if (isupper(str[i]))
            str[i] = (str[i] % 'A' + 1) % ALPHA + 'A';
    }
    printf("%s\n", str);
}

int main(void){
    char myString[] = "Sam's section rockz!";

    change(myString);
    printf("%s\n", myString);
}
```

(A) What does the program above output?
The program above is a "hard-coded" caesar cipher that shifts all alphabetic characters forward by one alphabetic index. It prints:

```
Tbn't tfdujpo spdla!
Tbn't tfdujpo spdla!
```

(B) What does `#define` do, and why does it appear in the program above?
`#define` creates a constant. It allows programmers to make their code more maintainable by allowing all references to some value to be linked to the same place.

(C) What does it mean for a function to have a "side effect"?
A function has a side effect if it performs some operation that "changes the state of the" outside of preparing the value it returns. Some examples of side effects would be: changing the value of a global variable or a persistent pointer or printing out a value.

7 Pointing is Rude

In the code below, you may assume the following values:

&w	0x123
&x	0x4ff
&y	0xf77
&z	0x288

```
#include <stdio.h>

int main(void){
  // comment 1
  int w = 2, x = 3, y = 4, z = 5;
  int* wp = &w;
  int* xp = &x;
  int* yp = &y;
  int* zp = &z;

  // comment 2
  wp = yp;
  yp = zp;
  zp = xp;

  // comment 3
  x = x*2;
  y = x-*yp;
  z = x + y;
  w = *wp + *zp;
  printf("%p\n", yp);
  printf("%i\n", w);
}
```

(A) In the space to the right of the code, please draw two box-and-arrow diagrams (like we did in section!): one showing the program's variables as it "passes through" comment 2, and one showing the program's variables as it "passes through" comment 3.

SEE ATTACHED DIAGRAMS

(B) What does the program output?

```
7
```

8 Up Periscope!

(A) Define scope.

Scope is the “area” within the a program that variable is “visible,” i.e. the area in which its contents can be accessed.

(B) What does the code below output?

```
void func(void){
  int x = 100;
  printf("%i\n", x);
}
int main(void){
  int x = 10;
  func();
  printf("%i\n", x);
}
```

```
100
10
```

(C) Does it matter what names variables are given? Why and why not?

The “and” in “why and why not” should tip you off that a complete answer to this question has more than one part. The name of a variable has no functional effect on its performance (unless multiple variables in the same scope are named the same thing) – the compiler just treats variables as variables and keeps track of them in an abstract form not related to their names. From the human perspective, however, variable names are exceedingly important: they help to ensure that code is readable and maintainable and tha readers aren’t confused by bad naming. For that reason, choosing good variable names is fundamental to high quality programming.

9 Inbox

Respond to the following emails:

(A) Dear TF:

I think something is going wrong with my compiler! I'm certain that my code is correct, but I keep seeing `Segmentation Fault` every time I run my program. What's going on?!?@@@? Please help <3333333

Sincerely, certain this is not an error.

Solution. Segmentation faults occur when a program attempts to access memory that it doesn't have access to. Common examples might including indexing into an illegal index in an array or trying to dereference a pointer to an arbitrary address in memory.

(B) Dear TF:

why do you keep on taking style points off on my problem sets? If code all looks the same to the computer, why does style even matter?

Sincerely, unstylish

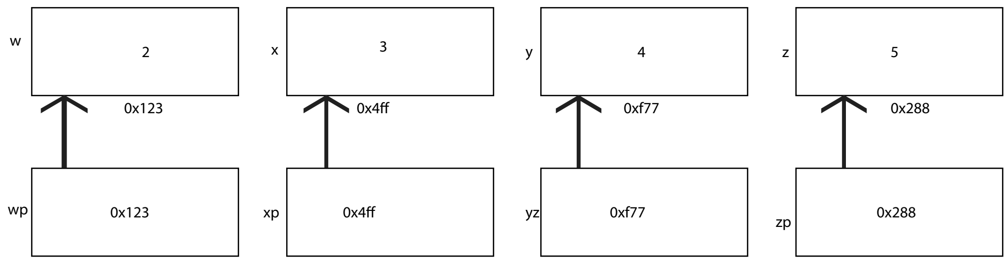
Solution. Programming is about more than writing functional code. Your code has to be readable to others, so that they can understand your work and collaborate with you, or give you constructive feedback. Style is fundamental to making sure that code is clear.

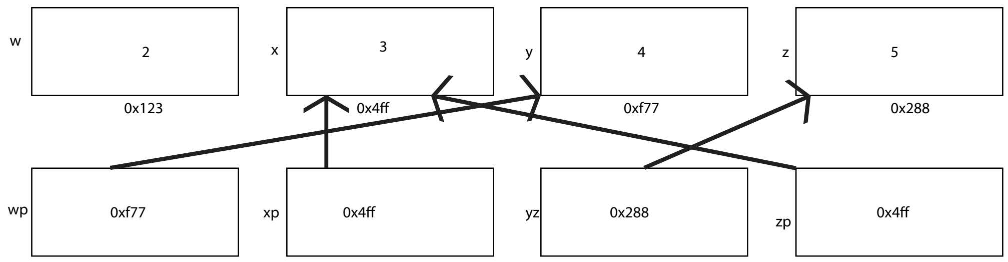
(C) Dear TF:
I really don't get the point of writing functions. Our programs are short, so I might as well just put the code in again! Can you help me understand the benefits?

Thx, functional

Solution. You might be right that the programs in CS50 are super short so far – but that doesn't mean we can should ignore good design! Imagine that you use the same block of code in 10 different places. It wouldn't take you *that* long to make changes to all 10 blocks of code. BUT, imagine that you made a mistake in one version! You could spend hours looking for that kind of a bug. Just one of the reasons that functions should be prefered when possible.

DIAGRAMS FOR “POINTING IS RUDE” ON FOLLOWING PAGE


w
2
x
3
y
4
z
5
0x123
0x4ff
0xf77
0x288
wp
0x123
xp
0x4ff
yz
0xf77
zp
0x288


w
2
x
3
y
4
z
5
0x123
0x4ff
0xf77
0x288
wp
0xf77
xp
0x4ff
yz
0x288
zp
0x4ff