

Problem 1: Generics, char * Hacks, and Function Pointers

```
a. int odds[] = {1, 3, 5, 7, 9, 11, 13, 15, 17};
    size_t count = sizeof(odds) / sizeof(odds[0]);
    int unlucky = 13;
    int *found = bsearch(&unlucky, odds, count, sizeof(odds[0]), cmp);
    assert(found == odds + 6);
```

```
b. char *suits[] = {"Harvey", "Mike", "Rachel", "Donna", "Jessica"};
   size_t count = sizeof(suits) / sizeof(char *);
   qsort(suits, count, sizeof(char *), cmp);
```

```
C. char dna[] = "GCGCAAACATGCACCGACCATTGGGATCGAATACTACAGTCT";
   size_t count = strlen(dna);
   char thymine = 'T';
   char *found = lfind(&thymine, dna, &count, 1, cmp);
   assert(found == dna + 9);
```

```
int cmp(const void *one, const void *two) {  
  
  
  
}
```

```
d. char *abbott[] = {
    "Ava", "Barbara", "Gregory", "Jacob", "Janine", "Melissa"
};
size_t count = sizeof(abbott) / sizeof(char *);
char *newbie = "Dominic";
char **found = bsearch(&newbie, abbott, count, sizeof(char *), cmp);
assert(found == NULL);
```

```
int cmp(const void *one, const void *two) {

}
}
```

```
e. char *abbott[] = {
    "Ava", "Barbara", "Gregory", "Jacob", "Janine", "Melissa"
};
size_t count = sizeof(abbott) / sizeof(char *);
char **found = bsearch("Dom", abbott, count, sizeof(char *), cmp);
// unorthodox, but doable, since key is always the first arg to cmp.
// this requires you to implement a comparison function that knows
// the first argument addresses a char and the second argument
// addresses a char *
assert(found == NULL);
```

```
int cmp(const void *one, const void *two) {

}
}
```

Answers:

```
a. return *(int *)one - *(int *)two;
b. return strcmp((char **)one, (char **)two);
c. return *(char *)one - *(char *)two;
d. return strcmp((char **)one, (char **)two);
e. return strcmp((char *)one, (char *)two);
// my answers strip away constness, because it doesn't matter here.
```

Problem 2: Implementing `count_matches` [5-minute problem]

Implement `count_matches`, which counts how many elements in an array match a supplied key. The function must work with arrays of any element type, so use the following prototype:

```
size_t count_matches(const void *key, const void *arr, size_t count,
                    size_t size, int (*cmp)(const void *, const void *));
```

Assume `cmp` returns 0 if and only if the elements addressed by its arguments match.

Assuming the `int`-specific version of `cmp` from Problem 1 is being used, the following code snippet should execute without drama:

```
int nums[] = {3, 7, 3, 9, 3, 5};
int key = 3;
assert(count_matches(&key, nums, 6, sizeof(nums[0]), cmp) == 3);
```

```
size_t count_matches(const void *key, const void *arr, size_t count,
                    size_t size, int (*cmp)(const void *, const void *)) {

}
}
```

Answer:

```
size_t matches = 0;
for (size_t i = 0; i < count; i++) {
    const char *elem = (const char *)arr + i * size;
    if (cmp(key, elem) == 0) matches++;
}
return matches;
```