Problem 1: strseparate
The most straightforward approach was to step through the C string addressed by \*stringp, one character at a
time, and relying on strchr to tell us when it’s found a delimiter character.

```
static char *strseparate(char **stringp, const char *delimiters) {
    char *front = *stringp;
    char *end = front;
    while (strchr(delimiters, *end) == NULL) end++;
    *stringp = *end == '\0' ? NULL : end + 1;
    *end = '\0';
    return front;
}
```

strchr even finds the ' \0 ' of delimiters if you pass 0 as the second argument, though it was fine to include a special
case for *end != ' \0 ' in the while loop (my first pass at this actually did just that.)

Many students invoke strchr on *stringp for each character in delims instead of the other way around. That
can be made to work, but it requires much more code to determine which delimiter character appears before the others.

Problem 2: The CMultiSet

```
CMap *make_map(CVector *cv) {
    CMap *cm = cmap_create(sizeof(char *), cvec_count(cv), cleanup);
    for (void *cur = cvec_first(cv); cur != NULL; cur = cvec_next(cv, cur)) {
        char *word = *(char **)cur;
        char *sig = strdup(word);
        qsort(sig, strlen(sig), 1, cmpchar);
        cmap_put(cm, word, &sig);
    }
    return cm;
}
```

```
int cmpchar(const void *a, const void *b) {
    return *(char *)a - *(char *)b;
}
```

Problem 3: The accumulate generic

a) This first part was designed to expose basic memory and pointer errors very early on—e.g. to confirm that weren’t
dropping &’s and *’s where they weren’t needed.

```
void accumulate(const void *base, size_t n, size_t elem_size,
       BinaryFunc fn, const void *init, void *result) {
    memcpy(result, init, elem_size);
    for (size_t i = 0; i < n; i++) {
        const void *next = (char *) base + i * elem_size;
        fn(result, next, result);
    }
}
```
b) static void multiply_two_numbers(void *partial, const void *next, void *result) {
  *(int *)result = *(int *)partial * *(const int *)next;
} // preserving the constness in the casts wasn’t necessary

int int_array_product(const int array[], size_t n) {
  int identity = 1, product;
  accumulate(array, n, sizeof(int),
             multiply_two_numbers, &identity, &product);
  return product;
}

Problem 4: Integer Types

a) B3A
b) -19
c) 13
d) 10101001011110111101111

e) 00011100
f) 1C
g) 11011111