CS136 EXERCISE SOLUTIONS: MODULE 9

Memory leaks

```
int *fib_array(int n) {
  assert(n > 1);
  int *buf = malloc(n * sizeof(int));
  buf[0] = 0;
  buf[1] = 1;
  for (int i = 2; i < n; ++i) {
    buf[i] = buf[i - 1] + buf[i - 2];
  }
  return buf;
}
```

Merge Sort

```
// merge(dest, src0, len0, src1, len1)
//     Copy  values from src0 and src1 to dest,
//     so the result is sorted.
// requires: src0 and src1 are already sorted.
//           dest has size at least len0 + len1
void merge(int *dest, const int *src0, int len0,
                      const int *src1, int len1) {
  while (len0 || len1) {
    if (!len1 || (len0 && *src0 < *src1)) {
      *dest = *src0;
      len0--;
      src0++;
    } else {
      *dest = *src1;
      len1--;
      src1++;
    }
    dest++;
  }
}
```

Persistent arrays

```
char *my_strdup(const char *src) {
  char *p = malloc(1 + strlen(src));

  // copy starting at the null-terminator.
  for (int i = strlen(src); i >= 0; --i) {
    p[i] = src[i];
  }
  return p;
}
```

Implementing a Stack ADT

```c
struct stack *stack_create(void) {
  struct stack *s = malloc(sizeof(struct stack));
  s->maxlen = 1;  // any positive value could work.
  s->len = 0;
  s->data = malloc(s->maxlen * sizeof(int));
  return s;
}

void stack_destroy(struct stack *s) {
  free(s->data);
  free(s);
}

void print_stack(const struct stack *s) {
  assert(s);
  assert(s->len <= s->maxlen);
  printf("|");
  for (int i = 0; i < s->len; ++i) printf("%d ", s->data[i]);
  printf("]\n");
}

void stack_push(int v, struct stack *s) {
  assert(s); // *don't* assert that there is enough space.
  if (s->len == s->maxlen) {
    s->maxlen *= 2; // instead, make the space.
    s->data = realloc(s->data, s->maxlen);
  }
  s->data[s->len] = v;
  s->len++;
}

bool stack_is_empty(const struct stack *s) {
  return s->len == 0;
}

int stack_pop(struct stack *s) {
  assert(s);
  assert(!stack_is_empty(s));

  s->len--;
  return s->data[s->len];
}

int main(void) {
  struct stack *s = stack_create();
  stack_push(2, s);
  stack_push(4, s);
  stack_push(6, s);
  stack_push(0, s);
  stack_push(1, s);
  print_stack(s);
  stack_destroy(s);
}
```