CS107 Cynthia Lee
Autumn 2018

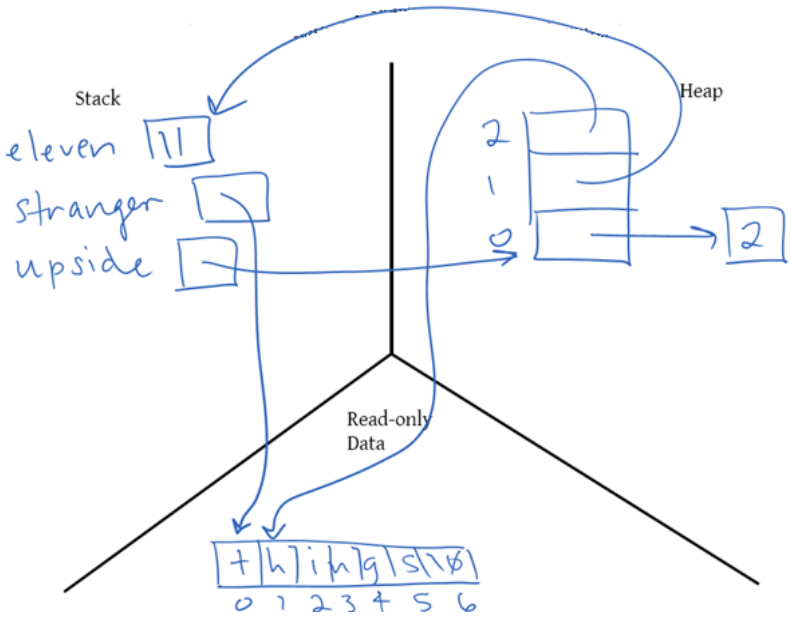
CS107 Midterm Practice Problems SOLUTIONS

Problem 1

(a) -20
(b) 31B or 0x31B
(c) 11101111010101101
(d) 00010101
(e) 11110001

Problem 2

(see diagram at right)



Problem 3

(a) b = 'v', ' ' (space)

(b) Bug fix 1: char **return_array = malloc(2 * sizeof(char*));
Bug fix 2: add +1 to malloc bytes (for null terminating character)
Bug fix 3: make comparison <= or make bound be +1

Problem 4:

```
bool odd_cols(unsigned int n)  {
      unsigned char *ptr = (unsigned char*)&n;
      unsigned char row0, row1, row2, row3;
      row0 = ptr[0];
      row1 = ptr[1];
      row2 = ptr[2];
      row3 = ptr[3];
      return (unsigned char) ~(row0 ^ row1 ^ row2 ^ row3) == 0;
}
```

Problem 5: Strings and Pointers Short Answer

(a) 1000 + 10*8 = 1080, 1000 + 10*1 = 1010, 1000 + 10*4 = 1040

(b) HELLOWORIAMREADYTOPARTY

Because the buggy code mixes up sizes and levels of indirection, it copies sizeof(char*) bytes into the returned strings. In other words, it copies the first 8 bytes of a string, which are the first 8 characters. So HELLOWORLD gets shortened to HELLOWOR, and because TOPARTY's null terminator is the 7th character, that is copied in, and the entire string print in printf ends there, even though more characters were in fact copied after that.

(c) The key conceptual error in original code is thinking that sizeoof on a char* variable is the same as strlen of a char* variable. This must be corrected in the first for loop and in the subsequent concatenation loop.

Here is one solution that corrects the arguments to memcpy:

```
char *multi_concatenate(const char *strs[], size_t num_strs) {
	size_t len = 1;
	for (size_t i = 0; i< num_strs; i++) {
		~~len += sizeof(strs[i]);~~ len += strlen(strs[i]);
	}
	char *result = malloc(len);
	int curr_len = 0;
	for (size_t i = 0; i< num_strs; i++) {
		~~memcpy(result + sizeof(strs[i]) * i, strs[i],~~
			~~sizeof(strs[i]));~~
		memcpy(result + curr_len, strs[i], strlen(strs[i]));
		curr_len += strlen(strs[i]);
	}
	~~result[sizeof(strs[0]) * num_strs] = '\0';~~
	result[curr_len] = '\0'; //need to add null terminator
	return result;
}
```

Here is another solution that replaces memcpy with the more appropriate strcat:

```
char *multi_concatenate(const char *strs[], size_t num_strs) {
	size_t len = 1;
	for (size_t i = 0; i< num_strs; i++) {
		~~len += sizeof(strs[i]);~~ len += strlen(strs[i]);
	}
	char *result = malloc(len);
	result[0] = '\0'; //need to start with null terminator before strcat
	for (size_t i = 0; i< num_strs; i++) {
		~~memcpy(result + sizeof(strs[i]) * i, strs[i],~~
			~~sizeof(strs[i]));~~
		strcat(result, strs[i]);
	}
	~~result[sizeof(strs[0]) * num_strs] = '\0';~~
	return result;
}
```

Problem 6: 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 7: Integer Representation

a) B3A
b) -19
c) 13
d) 10101001011111011101111
e) 0011100
f) 1C
g) 11011111

Problem 8: Integer Representation

(a) 0xCAFE
(b) 00110111
(c) -86
(d) 11110011

Problem 9: Pointers and strings

'I'
'E'

```
*first_key = strdup(cmap_first(cmap));
*values = malloc(nelems * sizeof(int));
(*values)[index++] = *(int*)cmap_get(cmap, cur))
```

Rubric Notes: (*values)[++ index] is not an acceptable answer

Problem 10: Memory Diagram