CS 61C Fall 2022

C and Pointers

Discussion 2

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and if false, correct the statement to make it true:

1.1 True or False: C is a pass-by-value language.

1.2 The following is correct C syntax:

```
int num = 43
```

1.3 In compiled languages, the compile time is generally pretty fast, however the run-time is significantly slower than interpreted languages.

1.4 The correct way of declaring a character array is `char[]` array.

1.5 Bitwise and logical operations result in the same behaviour for given bitstrings.

2 Bit-wise Operations

2.1 In C, we have a few bit-wise operators at our disposal:

- `AND` (`&`)
- `NOT` ($\sim$)
- `OR` (`|`)
- `XOR` ($\wedge$)
- `SHIFT LEFT` (`<<`)
  - Example: `0b0001 << 2 = 0b0100`
- `SHIFT RIGHT` (`>>`)
  - Example: `0b0100 >> 2 = 0b0001`

a	**b**	**a & b**	**a | b**	**a ∧ b**	**∼a**
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

For your convenience, truth tables for the logical operators are provided above. With the binary numbers `a`, `b`, and `c` below, perform the following bit-wise operations:

```
a = 0b1000 1011
b = 0b0011 0101
c = 0b1111 0000
```

(a) `a & b`

(b) `a` ∧ `c`

(c) `a | 0`

(d) `a | (b >> 5)`

(e) ∼ ((`b | c`) `& a`)

3 Pass-by-who?

3.1 Implement the following functions so that they work as described.

(a) Swap the value of two **`int`**s. *Remain swapped after returning from this function.*
Hint: Our answer is around three lines long.

```
void swap(________________, ________________) {
```

(b) Return the number of bytes in a string. *Do not use* `strlen`.
Hint: Our answer is around 4 lines long.

```
int mystrlen(________________) {
```

4 Debugging

4.1 The following functions may contain logic or syntax errors. Find and correct them.

(a) Returns the sum of all the elements in `summands`.

```
1  int sum(int *summands) {
2      int sum = 0;
3      for (int i = 0; i < sizeof(summands); i++)
4          sum += *(summands + i);
5      return sum;
6  }
```

(b) Increments all of the letters in the `string` which is stored at the front of an array of arbitrary length, `n >= strlen(string)`. Does not modify any other parts of the array's memory.

```
1  void increment(char *string, int n) {
2      for (int i = 0; i < n; i++)
3          *(string + i)++;
4  }
```

(c) Copies the string `src` to `dst`.

```
1  void copy(char *src, char *dst) {
2      while (*dst++ = *src++);
3  }
```

(d) Overwrites an input string `src` with "61C is awesome!" if there's room. Does nothing if there is not. Assume that `length` correctly represents the length of `src`.

```
1  void cs61c(char *src, size_t length) {
2      char *srcptr, replaceptr;
3      char replacement[16] = "61C is awesome!";
4      srcptr = src;
5      replaceptr = replacement;
6      if (length >= 16) {
7          for (int i = 0; i < 16; i++)
8              *srcptr++ = *replaceptr++;
9      }
10 }
```