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.

True

1.2 The following is correct C syntax:

```
int num = 43
```

False, 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.

False, Reasonable compilation time, excellent run-time performance. It optimizes for a give processor type and operating system.

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

False   char arr[];

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

False Bitwise and logical operations fundamentally speaking, perform the same operations, just in differencet contexts. Bitwise operations compare and operate on inputs bit-by-bit, from least to most significant bit in the bitstring. Logical operations compare and operate on inputs as a whole, where anything not 0 can be considered to be a 1

2 Bit-wise Operations

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

- AND (&)
- NOT (~)
- OR (|)
- XOR (^)
- 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` 0b 0000 0001

(b) `a ∧ c` 0b 0111 1011

(c) `a | 0` 0b 1000 1011

(d) `a | (b >> 5)` 0b 1000 1011

(e) `∼ ((b | c) & a)` 0b 0111 1110

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(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

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

```
int mystrlen(char *str) {
    int count = 0;
    while( str[count] != '\0'){
        count++;
    }
    return count;
}
```

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`.

```
int sum(int *summands) {
    int sum = 0;
    for (int i = 0; i < sizeof(summands); i++)
        sum += *(summands + i);
    return sum;
}
```

```
int sum(int* summands, size_t n){
  int sum =0;
 for(int i=0; i<n; i++)
     sum += *(summands +i);
 return sum;
}
```

(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.

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

```
void increment(char * string){
 for(int i=0; string[i] != '\0'; i++){
       string[i]++;
 }
}
```

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

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

(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`.

```
void cs61c(char *src, size_t length) {
    char *srcptr, replaceptr;   *replaceptr
    char replacement[16] = "61C is awesome!";
    srcptr = src;
    replaceptr = replacement;
    if (length >= 16) {
        for (int i = 0; i < 16; i++)
            *srcptr++ = *replaceptr++;
    }
}
```