CS61c Summer 2014 Discussion 2 – C

1 C Introduction

C is syntactically very similar to Java, but there are a few key differences of which to be wary:

- C is function oriented, not object oriented, so no objects for you.
- C does not automatically handle memory for you.
  - In the case of stack memory (things allocated in the "usual" way), a datum is garbage immediately after the function in which it was defined returns.
  - In the case of heap memory (things allocated with `malloc` and friends), data is freed only when the programmer explicitly frees it.
  - In any case, allocated memory always holds garbage until it is initialized.
- C uses pointers explicitly. `*p` tells us to use the value that `p` points to, rather than the value of `p`, and `&x` gives the address of `x` rather than the value of `x`.

There are other differences of which you should be aware, but this should be enough for you to get your feet wet.

2 At Least There Are Comments.

Write the following functions so that they perform according to the provided comment.

1.
```
/* The first function you write in any language.
 * Prints the string "Hello World\n" to standard output. */
void hello_world() {
  printf("Hello World\n");
}
```

2.
```
/* Divides and takes the floor of a value exterior to this function by 2^POW.
 * Does not use the division function. */
void div(int *y, unsigned int pow) {
  *y = y[0] >> pow;
}
```

3.
```
/* For each bit position i in [0, sizeof(int)*8) calls hello_world i times
 * iff the ith bit of the value X points to is set. */
void HI_HI_HI_HI(int *x) {
  int i = 0, j = 0, int_bits = sizeof(int) * 8;
  for (i = 0; i < int_bits; i++) {
    if ((x[0] >> i) & 1) {
      for (j = 0; j < i; j++) {
        hello_world();
      }
    }
  }
}
```

4.
```
/* Computes and returns the nth fibonacci number, using an iterative approach. */
int fib_iter(unsigned int n) {
  int fib0 = 0, fib1 = 1, i, swap;
```

```
  for (i = 0; i < n; i++) {
    swap = fib1;
    fib1 += fib0;
    fib0 = swap;
  }
  return fib0;
}
```

3 Uncommented Code? Yuck!

The following functions work correctly (note, this does not mean intelligently), but have no comments. Document the code to prevent it from causing further confusion.

1.
```
/* Returns the sum of the first N elements in ARR. */
int foo(int *arr, size_t n) {
  return n ? arr[0] + foo(arr + 1, n - 1) : 0;
}
```

2.
```
/* Returns -1 times the number of zeroes in the first N elements of ARR. */
int bar(int *arr, size_t n) {
  int sum = 0, i;
  for (i = n; i > 0; i--) {
    sum += !arr[i - 1];
  }
  return ~sum + 1;
}
```

3.
```
/* Does nothing. */
void baz(int x, int y) {
  x = x ^ y;
  y = x ^ y;
  x = x ^ y;
}
```

4 Programming with Pointers

Write the following functions so that they perform according to the provided comment. Not all questions are guaranteed to be soluble.

1.
```
/* Swaps the value of two ints outside of this function. */
void swap(int *x, int *y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}
```

2.
```
/* Increments the value of an int outside of this function by one. */
void plus_plus(int *x) {
  x[0]++;
}
```

3.
```
/* Returns a buffer for N ints. */
// Insoluble using provided machinery. Can of course be done using malloc.
int* allocate_buffer(unsigned int size) {
  return malloc(sizeof(int) * size); //note that this is an unchecked malloc
}
```

4.
```
/* Returns the number of bytes in a string. Does not use strlen. */
int mystrlen(char* str) {
  int count = 0;
  while(*str++) {
    count++;
  }
  return count;
}
```

5.
```
/* Returns the number of elements in an array ARR of ints. */
// insoluble
```

5 Problem?

The following code segments may contain either logic or syntax errors. Find them.

1.
```
/* Returns the sum of all the elements in SUMMANDS. */
int sum(int* summands) {                    //int sum(int* summands, unsigned int n)
  int sum = 0;
  for (int i = 0; i < sizeof(summands); i++) //i < n
    sum += *(summands + i);
  return sum;
}
```

2.
```
/* Increments all the letters in the string STRING, held in an array of length N.
 * Does not modify any other memory which has been previously allocated. */
void increment(char* string, int n) {
  for (int i = 0; i < n; i++) //for (i = 0; string[i] != 0; i++)
    *(string + i)++;          //string[i]++; or (*(string + i))++;

  //consider the corner case of incrementing 0xff
}
```

3.
```
/* Copies the string SRC to DST. */
void copy(char* src, char* dst) {
  while (*dst++ = *src++);
}

/* Parses a numeric character, putting the result into the value of VALUE and returning
 * 1 if it was successful and 0 otherwise. */
int parse_digit(char c, int * value) {
  if(c>='0' && c<='9') // {
     *value = c-'0';
     return 1;         // }
  return 0;
}
```