# CS61C FA20 Quest Solutions

Instructors: Dan Garcia, Borivje Nikolic  Head TAs: Stephan Kaminsky, Cece McMahon

## Q2: Bit Manipulation

**Solution Walkthrough**: https://cs61c.org/fa20/videos?ytvid=zshCbM9AaKw

---

**bitmanip.c**

```c
#include <inttypes.h>

// Replace with values given in question prompt
int GROUP_SIZE = 5;
char *ROT_DIR = "right";
int ROT_AMT = 3;
int ON_BIT = 2;
int OFF_BIT = 1;

unsigned get_bit(uint64_t x, uint64_t n) {
  // n = 2
  //     ABCDE
  //     ABC   (>> n)
  //       C   (& 1)
  return 1 & (x >> n);
}

void set_bit(uint64_t *x, uint64_t n, uint64_t v) {
  // n = 2
  //     ABCDE
  // AND 11011 // n = 2, (1 << 2) = 0b...00100, ~(1 << 2) = 0b...11011
  //     -----
  //     AB0DE
  //  OR 00v00 // (v << 2) = 0b...00v00
  //     -----
  //     ABvDE
  *x = (*x & ~(1 << n)) | (v << n);
}

// Not the most efficient solution, but written for slightly better readability
uint64_t bit_manip(uint64_t num) {
  uint32_t num_groups = 64 / GROUP_SIZE + 1;
  uint64_t mask = 0;
  uint64_t vals[num_groups];

  // Build a mask of GROUP_SIZE 1's
  for (int i = 0; i < GROUP_SIZE; i++) {
    mask = mask | (1 << i);
  }

  // Separate all 64 bits of `num` into groups of GROUP_SIZE bits
  for (int i = num_groups; i > 0; i--) {
```

```
    vals[i - 1] = num & mask;
    num = num >> GROUP_SIZE;
  }

  uint64_t result = 0;

  // For each group
  for (int i = 0; i < num_groups; i++) {
    // Rotate group by ROT_AMT in ROT_DIR ("right" in this case)
    for (int j = 0; j < ROT_AMT; j++) {
      // Rotate 1 bit at a time
      //  ________
      // /        \
      // |  0110  1 = 01101
      // \  \\\\
      //  \  \\\\
      //   \  \\\\
      //    1  0110 = 10110
      unsigned bit_0 = get_bit(vals[i], 0);
      vals[i] = vals[i] >> 1;
      set_bit(&(vals[i]), GROUP_SIZE - 1, bit_0);
    }

    // Set the on/off bits
    set_bit(&(vals[i]), ON_BIT, 1);
    set_bit(&(vals[i]), OFF_BIT, 0);

    // Insert group into result
    // result =                                 01010
    // result << GROUP_SIZE =                   01010 00000
    // (result << GROUP_SIZE) | vals[i] =       01010 00000
    //                                    OR          10110
    //                                          -----------
    //                                          01010 10110
    result = result << GROUP_SIZE;
    result = result | vals[i];
  }
  return result;
}
```

## Q3: Split

**Solution Walkthrough**: https://cs61c.org/fa20/videos?ytvid=zshCbM9AaKw

---

**split.c**

```c
#include "split.h"
#include <string.h>

void *CS61C_malloc(size_t size);
void CS61C_free(void *ptr);

/*
For reference, this is the Node struct defined in split.h:
typedef struct node {
  char *data;
  struct node *next;
} Node;
*/

void split(Node *words, Node **consonants, Node **vowels) {
  Node *consonants_last = NULL;
  Node *vowels_last = NULL;

  // While we haven't reached the list's NULL terminator
  while (words != NULL) {
    // Allocate memory for the node and its data, and copy the data string
    Node *item = (Node *) CS61C_malloc(sizeof(Node));
    item->data = (char *) CS61C_malloc(sizeof(char) * (strlen(words->data) + 1));
    item->next = NULL; // Make sure there is no next item (for now)
    strcpy(item->data, words->data);

    // Check which list the item belongs in
    char first = words->data[0];
    if (first == 'a' || first == 'e' || first == 'i' || first == 'o' || first == 'u') {
      // If this is the first item in `vowels`, set it as the head of the `vowels` list
      if (vowels_last == NULL) {
        *vowels = item;
      // Otherwise, append this item to the end of the list
      } else {
        vowels_last->next = item;
      }
      // Save this item so the next item can be appended after it
      vowels_last = item;
    } else {
      // If this is the first item in `consonants`, set it as the head of the `consonants` list
      if (consonants_last == NULL) {
        *consonants = item;
      // Otherwise, append this item to the end of the list
      } else {
        consonants_last->next = item;
      }
      // Save this item so the next item can be appended after it
      consonants_last = item;
    }

    // Free the current node in the words list, and move to the next node
    Node *temp = words->next;
    CS61C_free(words->data);
```

```
    CS61C_free(words);
    words = temp;
  }
  return;
}
```