Solution

1. (32 points) **Learn C**. Please read the questions carefully. For each question, if you answer the question correctly, you get **+2** points. If you do not answer, you get **0** points. If you answer the question incorrectly, you get **-2** points. For all questions, assume the platform is **64-bit**.

1.1. Assume a is an 8-bit number whose value is 0b00001010, what is its two's complement?

a. 0b11110101
b. 0b11110110
c. 0b11110111
d. None of above

1.2.

```
unsigned char ux = 238;
char sx = ux;
printf("%d", sx);
```

What is the output of this code?

a. -17
b. -18
c. -19
d. -20

1.3. -1 == -1U is true.

a. True
b. False

1.4. -1U > 0 is true

a. True
b. False

1.5. -1 < sizeof(-1) is true

a. True
b. False

1.6.

```
unsigned int a = 0xffffffff;
printf("%d\n", a + 8);
```

What is the output of this code?

a. 7

b. 8
c. 9
d. None of above (due to undefined behaviors)

1.7.

```
int a = 1;
int b = 2;
int c = 3;

printf("%d\n", (a && b) & c);
```

What is the output of this code?

a. 1
b. 2
c. 3
d. None of above

1.8.

```
char a = 16;
a = a << 3;
printf(“%d\n”, a);
```

What is the output of this code?

a. 32
b. 64
c. 128
d. -128

1.9. ('a' > 'c' - 2) is true.

a. True
b. False

1.10.

```
char a[100];
printf("%ld\n", sizeof(a));
```

What is the output of this code?

a. 0
b. 100
c. 101
d. Unknown

1.11.

```
char a[100];
```

```
printf("%ld\n", strlen(a));
```

What is the output of this code?

a. 0
b. 100
c. 101
d. Unknown

1.12.

```
printf("%ld\n", sizeof(unsigned int) + sizeof(short) * sizeof(char*));
```

What is the output of this code?

a. 12
b. 20
c. 36
d. 40

1.13.

```
int a = 1;
int b;

for (b = 0; b < 10; b++)
  a++;

printf("%d\n", ++a);
```

What is the output of this code?

a. 10
b. 11
c. 12
d. None of above

1.14.

```
int a[5] = {1,2};
printf("%d\n", a[4]);
```

What is the output of this code?

a. 1
b. 2
c. 0
d. Unknown

1.15.

```
int a[] = {1,2,3,4,5};
printf("%ld\n", sizeof(a));
```

What is the output of this code?

a. 4
b. 5
c. 20
d. None of above

1.16.

```
char x[] = “abcd”;
printf(“%ld\n”, sizeof(x));
```

What is the output of this code?

a. 4
b. 5
c. 20
d. None of above

1.17.

```
char *x = “abcd”;
x+=2;
printf(“%c\n”, *x);
```

What is the output of this code?

a. a
b. b
c. c
d. None of above (due to a compile error)

1.18.

```
int *(*p)[10];
printf(“%ld\n”, sizeof(p));
```

What is the output of this code?

a. 8
b. 40
c. 80
d. None of above

2. **Function** (6 points)
Write expressions to complete a recursive function that prints binary of a positive integer.

```
void print_binary (unsigned int n) {
  if (n >= [1])
    print_binary(n / [2]);

  putchar([3] == 0 ? '0': '1');
}
```

[1]: 2
[2]: 2
[3]: n % 2

3. **Scope** (8 points)
What is the output of this code?

```
#include <stdio.h>
int i = 0;

int f1(int i)
{
  i+=1;
  return i;
}

int f2() {
  i+=2;
  return i;
}

int f3() {
  int i = 0;
  i+=3;
  return i;
}

int f4() {
  static int i = 0;
  i+=4;
  return i;
}


int main() {
  printf("f1 (1): %d\n", f1(1));
  printf("f1 (2): %d\n", f1(1));
  printf("f2 (1): %d\n", f2());
  printf("f2 (2): %d\n", f2());
  printf("f3 (1): %d\n", f3());
  printf("f3 (2): %d\n", f3());
  printf("f4 (1): %d\n", f4());
  printf("f4 (2): %d\n", f4());
}
```

f1 (1): [2]
f1 (2): [2]
f2 (1): [2]
f2 (2): [4]
f3 (1): [3]
f3 (2): [3]
f4 (1): [4]
f4 (2): [8]

4. **Dynamic Memory (8 points)**

```
// counter.h

#ifndef _COUNTER_H_
#define _COUNTER_H_

#define INIT_CAPACITY 1024

typedef struct {
  int* arr;
  int capacity;
} Counter;

Counter* Counter_init();
int Counter_get(Counter* c, int index);
void Counter_increment(Counter* c, int index);
void Counter_resize(Counter* c, int newCapacity);
void Counter_free(Counter* c);

#endif
```

```
// counter.c

// Assumption
// 1) realloc & malloc are  always successful
// (i.e., No NULL check is okay)
// 2) No assertion would happen at noOutOfBounds()

#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include "counter.h"

void noOutOfBounds(Counter* c, int index) {
  assert(index >= 0 && index < c->capacity);
}

Counter* Counter_init() {
  Counter* c = malloc(sizeof(Counter));
  c->capacity = INIT_CAPACITY;
  c->arr = malloc(sizeof(int) * INIT_CAPACITY);
  return c;
}

int Counter_get(Counter* c, int index) {
  noOutOfBounds(c, index);
  return c->arr[index];
}

void Counter_increment(Counter* c, int index) {
```

```
  noOutOfBounds(c, index);
  c->arr[index]++;
}

void Counter_resize(Counter* c, int newCapacity) {
  c->arr = realloc(c->arr, sizeof(int) * newCapacity);
  c->capacity = newCapacity;
}

void Counter_free(Counter* c) {
  free(c->arr);
  free(c);
}
```

```
// client.c

#include <stdio.h>
#include "counter.h"

enum { FALSE = 0, TRUE = 1 };

…

int testSimple() {
  int res = TRUE;
  Counter* c = Counter_init();
  for (int i = 0; i < INIT_CAPACITY; i++) {
    Counter_increment(c, i);
    if (Counter_get(c, i) != 1) {
      res = FALSE;
      break;
    }
  }
  Counter_free(c);
  return res;
}

int testResize() {
  int res = TRUE;

  Counter* c = Counter_init();
  Counter_resize(c, INIT_CAPACITY * 2);
  for (int i = INIT_CAPACITY; i < INIT_CAPACITY * 2; i++) {
    Counter_increment(c, i);
    if (Counter_get(c, i) != 1) {
      res = FALSE;
      break;
    }
  }
  Counter_free(c);
```

```
  return res;
}

…

int main() {
  …
  printf("testSimple: %s\n", testSimple() ? "Success" : "Failed");
  printf("testResize: %s\n", testResize() ? "Success" : "Failed");
  …
}
```

I implemented a program that tracks the number of increments, called Counter. To test this program, I made a test client and ran the program as follows.

```
$ gcc209 -o client client.c counter.c
$ ./client
…
testSimple: Failed
testResize: Failed
…
```

Unfortunately, as shown above, two tests failed. I found that my counter.c contains two errors. Please identify where errors are and how to fix them (NOTE: You don't have to write a fixed code but explain clearly).

a. Error 1:

- Where (Code): c->arr = malloc(sizeof(int) * INIT_CAPACITY);
- How to fix: Change it to calloc (initialize)

b. Error 2:

- Where (Code): realloc(c->arr, sizeof(int) * newCapacity);
- How to fix: Initialize an expanded part if the size is increased.

5. Pointers and functions (10 points)

```
void swap(int v[], int i, int j) {          // swap: swap the values of v[i] and v[j]
   int temp = v[i]; v[i] = v[j]; v[j] = temp;
}

void qsort(int v[], int left, int right) {   // qsort: sort v[left] … v[right] into increasing order
   int i, last;

   if (left >= right)   // base case: if the group is too small, stop
      return;

   // others: pick pivot as the first-indexed # (i.e., left) in the current group. Partition the
   // group into A and B such that (all elms in A) <= pivot and (all elms in B) > pivot

   last = left;
   for (i = left + 1; i <= right; i++)
      if (v[i] <= v[left])
         swap(v, ++last, i);

   swap(v, left, last);
   qsort(v, left, last-1);      // sort group A [left..last-1]
   qsort(v, last+1, right);   // sort group B [last+1..right]
}
```

Above is the quicksort code that we learned in class (slightly modified, but does the same thing). Please convert the quicksort code to the pointer version by filling in the swap_ptr() and qsort_ptr() functions below. Note: swap_ptr() is the pointer version of swap() and qsort_ptr() is the pointer version of qsort().

```
#include <stdio.h>

void swap_ptr(int *i, int *j) {
   // Your code
}

void qsort_ptr(int *left, int *right) {
   // Your code
}

int main(void) {
  int arr[] = {3, 4, 2, 10, 6, 8, 5, 1, 9, 7};
  qsort_ptr(arr, arr + sizeof(arr)/sizeof(int) - 1);
  return 0;
}
```

```c
#include <stdio.h>

void swap_ptr(int *i, int *j) {
   int temp = *i; *i = *j; *j = temp;
}

void qsort_ptr(int *left, int *right) {
   int *i, *last;
   if(left >= right)
      return;
   last = left;
   for (i = left + 1; i <= right; i++)
      if (*i <= *left)
         swap_ptr(++last, i);
   swap_ptr(left, last);
   qsort_ptr(left, last-1);
   qsort_ptr(last+1, right);
}

int main(void) {
  int arr[] = {3, 4, 2, 10, 6, 8, 5, 1, 9, 7};
  qsort_ptr(arr, arr + sizeof(arr)/sizeof(int) - 1);

  return 1;
}
```

6. Hash table (20 points)

```
enum {BUCKET_COUNT = 1024};

struct Node {
   const char *key;
   int value;
   struct Node *next;
};

struct Table {
   struct Node *array[BUCKET_COUNT];
};

unsigned int hash(const char *x) {
   int i;
   unsigned int h = 0U;
   for (i=0; x[i]!='\0'; i++)
      h = h * 65599 + (unsigned char)x[i];
   return h % 1024;
}

int Table_updateValue(struct Table *t, const char *key, int newValue) {

   // Your code

}

int Table_updateKey(struct Table *t, const char *key, const char *newKey) {

   // Your code

}
```

a (5 points) In the above code, write your own Table_updateValue() function. It finds the node with the matching key (use strcmp), which then updates the value to newValue. Assume unique key.

b. (15 points) In the above code, write your own Table_updateKey() function. It finds the node with the matching key (use strcmp), which then updates the key to newKey. Assume unique key.

```
// Unique Key version

enum {BUCKET_COUNT = 1024};

struct Node {
   const char *key;
   int value;
   struct Node *next;
};

struct Table {
   struct Node *array[BUCKET_COUNT];
};

unsigned int hash(const char *x) {
   int i;
   unsigned int h = 0U;
   for (i=0; x[i]!='\0'; i++)
      h = h * 65599 + (unsigned char)x[i];
   return h % 1024;
}

int Table_updateValue(struct Table *t, const char *key, int newValue)
{

   struct Node *p;
   int h = hash(key);
   for (p = t->array[h]; p != NULL; p = p->next)
      if (strcmp(p->key, key) == 0) {
         p->value = newValue;
         return 1;
      }
   return 0;
}

int Table_updateKey(struct Table *t, const char *key, const char *newKey)
{
   struct Node *p; struct Node *prevp;
   int h = hash(key);
   int newH = hash(newKey);
   p = t->array[h];
   prevp = t->array[h];
   while(p != NULL) {
      if(strcmp(p->key, key) == 0) {
         if(p == t->array[h])
            t->array[h] = p->next;
```

```
        else
           prevp->next = p->next;
        p->key = newKey;
        p->next = t->array[newH];
        t->array[newH] = p;

        return 1;
      }
      prevp = p;
      p = p->next;
    }

    return 0;
}
```

7. Debugging (12 points)

Following is a silly code that prints a sub-string beginning from the input character. This code has 3 errors. Please identify where errors are and how to fix them. **(NOTE: please write the fixed code)**

```
#include <stdio.h>

char *findSubstring(char *string, char target) {
   char *cPtr = string;
   while(*cPtr != '\0') {
      if ((*cPtr) == target)
         return cPtr;
     cPtr++;
   }
   return NULL;
}

int main(void)
{
   char c;
   char *string = "EE209 is a great class";
   char arr[] = "dummy string";

   if(c = getchar() != EOF)
      if((arr = findSubstring(string, c)))
         printf("Substring: %s\n", arr);
      else
         printf("No match\n");
   else
      printf("EOF\n");

   return 0;
}
```

a. Error 1:

- Where (Code): char c (or c = getchar())
- Fixed code: char c -> int c;

b. Error 2:

- Where (Code): char arr[] = “dummy string” (or arr = findSubstring(string, c))
- Fixed code: char arr[] = “dummy string” -> char *arr = “dummy string”

c. Error 3:

- Where (Code): if(c = getchar() != EOF)
- Fixed code: if((c = getchar()) != EOF)