CS 4400 Fall 2017
Final Exam – Practice

Name: ____________________________

Instructions You will have eighty minutes to complete the actual open-book, open-note exam. Electronic devices will be allowed only to consult notes or books from local storage; network use will be prohibited. The actual exam will be shorter than this practice exam.

The first three questions refer to the code template

```
long f(long x, long y) {
  return ....;
}
```

and the code fragments

1. `x*(2+y)`
2. `x+8*y`
3. `((long *)x)[y]`
4. `*(long *)x = y`
6. `x == y`

1. Which of the above code fragments most likely appears in place of `....` if the generated machine code includes the following instruction?

```
leaq (%rdi, %rsi, 8), %rax
```

2. Which of the above code fragments most likely appears in place of `....` if the generated machine code includes the following instruction?

```
movq (%rdi, %rsi, 8), %rax
```

3. Which of the above code fragments most likely appears in place of `....` if the generated machine code includes the following sequence?

```
cmpq %rdi, %rsi
sete %al
movzbq %rax
```

The next four questions refer to the following assembly sequence:

```
        leal (, %eax, 4), %ebx
        shrl $0x2, %ebx
        subl %ebx, %eax
        cmpl $0x0, %eax
        jg   .L1
        andl $0x7FFFFFFF, %eax
    .L1:
        shrl $29, %eax
        ret
```

4. If `%eax` starts as 0, what is the ending value of register `%eax`?

5. If `%eax` starts as -1, what is the ending value of register `%eax`?

6. If `%eax` starts as `0x4FFFFFFF`, what is the ending value of register `%eax`?

7. If `%eax` starts as unknown, what are all the possible ending values of `%eax`?

8. Given that

```
int sum_element(int c,
                long mat1[][M],
                long mat2[][N]) {
  int k;
  long sum = 0;
  for (k = 0; k < c; k++)
    sum += mat1[k][c] - mat2[k][c];
  return sum;
}
```

compiles as

```
    testl   %edi, %edi
    jle     zero
    movslq  %edi, %rax
    subl    $1, %edi
    leaq    0(,%rax,8), %r8
    leaq    (%rax,%rdi,4), %rax
    leaq    (%rsi,%r8), %rcx
    addq    %r8, %rdx
    leaq    32(%rsi,%rax,8), %rsi
    xorl    %eax, %eax
loop:
    addq    (%rcx), %rax
    addq    $32, %rcx
    subq    (%rdx), %rax
    addq    $40, %rdx
    cmpq    %rsi, %rcx
    jne     loop
    ret
zero:
    xorl    %eax, %eax
    ret
```

then what are the values of the constants `M` and `N` among the following possibilities?

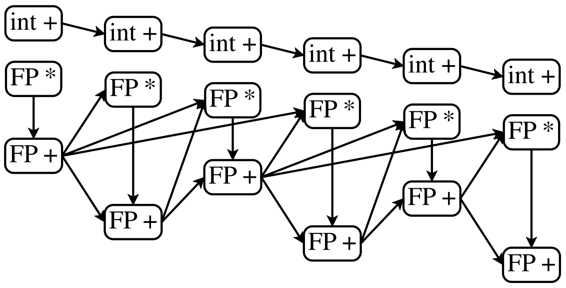
- `M` = 10 and `N` = 6
- `M` = 12 and `N` = 2
- `M` = 3 and `N` = 5
- `M` = 4 and `N` = 5
- `M` = 5 and `N` = 12
- `M` = 8 and `N` = 17

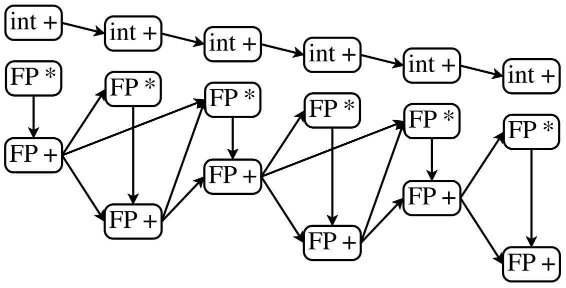
To enable partial credit, show your reasoning for how the assembly code corresponds to elements of the C code.

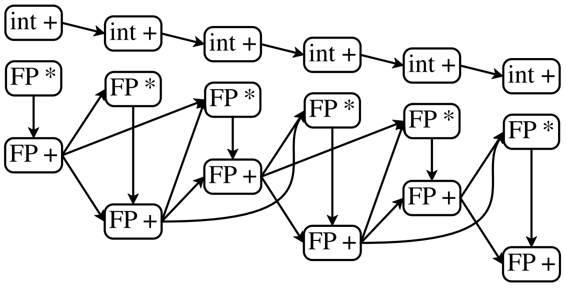
The next two questions refer to the `iterate` function defined as

```
double iterate(double x, double y, double z, int steps) {
  int i;
  for (i = 0; i < steps; i++) {
    z = x * y;
    if (!(i & 0x1))
      x = y + z;
    else
      y = x + z;
  }
  return z;
}
```

9. Which of the following correctly represents the dependency graph of `iterate` over six iterations, where each column corresponds to a single iteration?

- 

- 

- 

10. Based on the dependency graph, how many cycles will `iterate` take (expressed as a multiple of `steps`)?

The following four questions refer to the following declarations:

```
typedef struct jumble {
  struct {
    int i;
  } r;
  double d;
  char c;
  union {
    int i;
    char *s;
  } u;
  short s;
} jumble;

jumble ja[7][5];
```

For the questions, assume that `ja` starts at address `0x10000`.

11. What is `sizeof(jumble)`?

12. What is the address of `ja[0][2].d`?

13. What is the address of `ja[4][2].u.s`?

14. What is the address of `ja[4][2].s`?

The next two questions refer to the following declarations and function:

```
typedef struct {
  int a[2];
} pair;

pair s[32][32];

int sum(int how_many) {
  int i, j, k, a = 0;

  for (i = 0; i < 32; i++)
    for (j = 0; j < 32; j++) {
      for (k = 0; k < how_many; k++) {
        a += s[i][j].a[k];
      }
    }

  return a;
}
```

For each question below, assume a 1kB direct-mapped cache that uses 16-byte blocks, the cache is initially empty, and local variables are in registers. Also assume that the array p is at the address 0x10000 in memory.

15. For each of first five memory accesses via p in sum(1), what is the accessed element, what is the accessed address, and is the access a cache hit or miss?

Access expression	Access address	Hit or miss

16. What is the overall expected cache-miss rate of sum(1)?

17. For each of first five memory accesses via `p` in `sum(2)`, what is the accessed element, what is the accessed address, and is the access a cache hit or miss?

Access expression	Access address	Hit or miss

18. What is the overall expected cache-miss rate of `sum(2)`?

19. What are all of the possible outputs of the following program?

Assume that the program runs on Linux, where the child process created by `fork` does *not* include all the threads of the parent process. The child process includes only the thread that called `fork`.

In case you don't list all of the possible outputs correctly, to improve opportunities for partial credit, show how you arrived at your answer by sketching one or more process/thread graphs.

```
#include "csapp.h"

static sem_t done;

static void *go(void *p) {
  pid_t pid = Fork();
  if (pid == 0) {
    Write(1, "c", 1);
    V(&done);
    exit(0);
  } else {
    Write(1, "p", 1);
    Waitpid(pid, NULL, 0);
    V(&done);
    return NULL;
  }
}

int main() {
  pid_t pid;
  pthread_t th1, th2;

  Sem_init(&done, 0, 0);

  Write(1, "+", 1);
  Pthread_create(&th1, NULL, go, NULL);
  Pthread_create(&th2, NULL, go, NULL);

  P(&done);
  P(&done);
  Write(1, "-", 1);

  return 0;
}
```

20. For this question, a *word* is defined to be 16 bytes, each cell in a diagram represents a word, and an underlined number $\underline{N}$ is a shorthand for *N* times 16.

An allocator produces word-aligned payload pointers, uses a word-sized header and footer for each allocated block, uses a 2-word prolog block and a 1-word terminator block, coalesces unallocated blocks, uses an explicit free list as a singly-linked and `NULL`-terminated list starting from `fp` and using the first word of a block payload for a "next" pointer, adds free blocks to the start of the free list, does not split a block to create an unallocated block with a zero-sized payload, uses a **best-fit** allocation strategy, and is confined to 16 words of memory that is initially filled with 0s. Show a header in a diagram as a value for the block size over a 0 or 1 to indicate the block's allocation status; draw a footer as just the block size. Show the free list as either equal to NULL or as a chain of pointers.

The left-hand column below contains a sequence of `malloc` and `free` calls that are handled by the allocator. Fill in the left-hand column by showing relevant header and footer values and the free list just after each step on the left. The first row of the left column is blank so that you can show the initial state of memory in the first row of the right column.

fp=

p1 = malloc($\underline{2}$) **fp=**

p2 = malloc($\underline{2}$) **fp=**

p3 = malloc($\underline{2}$) **fp=**

p4 = malloc($\underline{2}$) **fp=**

free(p1) **fp=**

free(p3) **fp=**

p5 = malloc($\underline{2}$) **fp=**

The last page contains a server implementation that implements a vote-counting service, except that many parts of the program have be deleted and replaced with blanks. Your task is to fill in each blank with a number that corresponds to one of the following program fragments. Each fragment can be used multiple times, and not all fragments need to be used, but you can only fill in blanks with these fragments. If you use a fragment that itself has blanks, then those blanks must be filled in, too; blanks in unused fragments do not need to be filled in.

The server listens for both TCP connections and UDP connections at a given port. Each message sent to the sever by UDP is received as a vote, which is recorded by incrementing a counter for the message string as stored in a dictionary. For example, a client could send the message `apple` to increasethe `apple` count. For each connection made to the server via TCP, the server sends back vote results so far. Vote results are reported as an `int` for the number of voted-on strings, then, for each voted-on string: each string's length as an `int`, its content as bytes, and its vote count as an `int`.

The server must listen for UDP and TCP connections currently. Since each UDP message can be handled immediately, there's no need to have multiple messages processed concurrently to each other. Each TCP connection may take a while to receive all the votes, however, so the server is meant to support multiple TCP connections currently. Finally, each TCP connection should be sent a snapshot of vote tallies at the time of the connection, and new votes via UDP should be allowed meanwhile.

The server uses a dictionary library with the following property: the dictionary makes it own copy of keys, but when a value is installed into the dictionary with `dictionary_set`, the dictionary takes ownership of the value and is responsible for deallocating it when the value is replaced (or when the dictionary is destroyed, but that never happens in this server). A client *does not* take ownership of a value that is extracted from the dictionary with `dictionary_get`; that value can be deallocated if it is meanwhile replaced.

1.
```
Close(s);
```

2.
```
ls = Open_listenfd(portno);
```

3.
```
struct sockaddr_storage addr;
unsigned int len = sizeof(addr);
s = Accept(ls, (struct sockaddr *)&addr, &len);
```

4.
```
s = Socket(addrs->ai_family, addrs->ai_socktype, addrs->ai_protocol);
```

5.
```
amt = Sendto(s, buffer, MAXBUF-1, 0,
             (struct sockaddr *)&addr, &len);
```

6.
```
char buffer[MAXBUF];
size_t amt;
amt = Recv(s, buffer, MAXBUF-1, 0);
buffer[amt] = 0;
```

7. `free(new_p);`

8. `free(p);`

9.
```
static int make_socket(char *portno) {
  int s;
  struct addrinfo hints, *addrs;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_STREAM;
  Getaddrinfo("localhost", portno, &hints, &addrs);

  ____________a

  Connet(s, addrs->ai_addr, addrs->ai_addrlen);
  Freeaddrinfo(addrs);

  return s;
}
```

10.
```
static int make_socket(char *portno) {
  int s;
  struct addrinfo hints, *addrs;

  memset(&hints, 0, sizeof(struct addrinfo));
  hints.ai_family = AF_INET;
  hints.ai_socktype = SOCK_DGRAM;
  hints.ai_flags = AI_PASSIVE;
  Getaddrinfo(NULL, portno, &hints, &addrs);

  ____________b

  Bind(s, addrs->ai_addr, addrs->ai_addrlen);
  Freeaddrinfo(addrs);

  return s;
}
```

11. `s`

12. `ls`

13. `Sem_init(&db_lock, 0, 1);`

14.
```
static void sigchld_handler(int sig) {
  while (waitpid(-1, NULL, WNOHANG) < 0)
   ;
}
```

15. `Signal(SIGCHLD, sigchld_handler);`

16. `V(&db_lock);`

17. `P(&db_lock);`

18. `Waitpid(-1, NULL, 0);`

19. `exit(0);`

20. `receive_votes`

21. `serve_counts`

22.
```
static void write_entry(dictionary_t *db, int i, int s) {
  const char *key = dictionary_key(db, i);
  int len = strlen(key);
  int votes = *(int *)dictionary_value(db, i);
  Rio_writen(s, &len, sizeof(len));
  Rio_writen(s, (void *)key, len);
  Rio_writen(s, &votes, sizeof(votes));
}
```

23. `/* A million threads isn't cool... */`

21. Fill in the blanks with numbers for fragments on the previous pages:

```
#include "csapp.h"
#include "dictionary.h"

static dictionary_t *db;
static sem_t db_lock;

___________c
___________d
___________e

static void receive_votes(char *portno) {
  int s = make_socket(portno);
  while (1) {
    int *p, *new_p;
    ___________f
    new_p = malloc(sizeof(int));
    *new_p = 1;
    ___________g
    p = dictionary_get(db, buffer);
    if (p) *new_p += *p;
    dictionary_set(db, buffer, new_p);
    ___________h
  }
}

static void *serve_counts(void *portno) {
  int ls, s;
  ___________i
  while (1) {
    ___________j
    ___________k
    if (Fork() == 0) {
      int i, count = dictionary_count(db);
      Rio_writen(s, &count, sizeof(count));
      for (i = 0; i < count; i++)
        write_entry(db, i, s);
      ___________l
    }
    ___________m
    Close(s);
  }
}

int main(int argc, char **argv) {
  pthread_t th;
  if (argc != 2) app_error("need <port>");
  db = make_dictionary(COMPARE_CASE_SENS, free);
  Sem_init(&db_lock, 0, 1);
  ___________n
  Pthread_create(&th, NULL, ___________o, argv[1]);
  Pthread_detach(th);
  ___________p(argv[1]);
  return 0;
}
```

Answers

1. `x+8*y`

2. `((long *)x)[y]`

3. `x == y`

4. 0

5. 2

6. 2

7. 0 and 2

8. `M` = 4 and `N` = 5

```
    testl   %edi, %edi            # check whether c is 0
    jle     zero                  # if c is zero, just return 0
    movslq  %edi, %rax            # copy c to rax
    subl    $1, %edi              # rdi = c-1
    leaq    0(,%rax,8), %r8       # r8 = c*sizeof(long)
    leaq    (%rax,%rdi,4), %rax   # rax = 5*(c-1)
    leaq    (%rsi,%r8), %rcx      # rcx = matrix1 + c*sizeof(long)
    addq    %r8, %rdx             # rdx = matrix2 + c*sizeof(long)
    leaq    32(%rsi,%rax,8), %rsi # rsi = matrix1 + 5*(c-1)*8+32; used as end condition
    xorl    %eax, %eax            # sum = 0
loop:
    addq    (%rcx), %rax          # sum += rcx[0]
    addq    $32, %rcx             # rcx += 4 * sizeof(long) => M = 4
    subq    (%rdx), %rax          # sum -= rdx[0]
    addq    $40, %rdx             # rdx += 5 * sizeof(long) => N = 5
    cmpq    %rsi, %rcx            # rcx == matrix1+c*sizeof(long)?
    jne     loop                  # if we haven't reached last row....
    ret
zero:
    xorl    %eax, %eax
    ret
```

9. The third one

10. 8, since there's a path that goes through both `FP*` and `FP+` at each iteration.

11. 40

```
typedef struct jumble {
  struct {
    int i;      // size 4, alignment 4 => offset 0
  } r;          // also size 4, offset 0
  double d;     // size 8, alignment 8 => offset 8
  char c;       // size 1, alignment 1 => offset 16
  union {
    int i;      // size 4, alignment 2 => offset 20... but 24 for s
    char *s;    // size 8, alignment 8 => offset 24
  } u;          // so, size 8, offset 24
  short s;      // size 2, alignment 2 => offset 32
} jumble;       // size >= 34, alignment 8 => size 40

jumble ja[7][5]; // each row is 40*5 bytes
```

12. 0x10000 + 80 + 8 = 0x10058

13. 0x10000 + 40×5×4 + 80 + 24 = 0x10388

14. 0x10000 + 40×5×4 + 80 + 32 = 0x10390

15.

Access expression	Access address	Hit or miss
s[0][0].a[0]	0x10000	miss
s[0][1].a[0]	0x10008	hit
s[0][2].a[0]	0x10010	miss
s[0][3].a[0]	0x10018	hit
s[0][4].a[0]	0x10020	miss

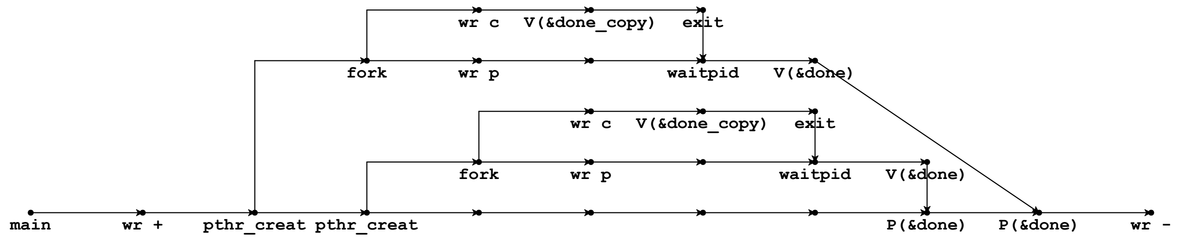
16. 50%

17.

Access expression	Access address	Hit or miss
s[0][0].a[0]	0x10000	miss
s[0][0].a[1]	0x10004	hit
s[0][1].a[0]	0x10008	hit
s[0][1].a[1]	0x1000c	hit
s[0][2].a[0]	0x10010	miss

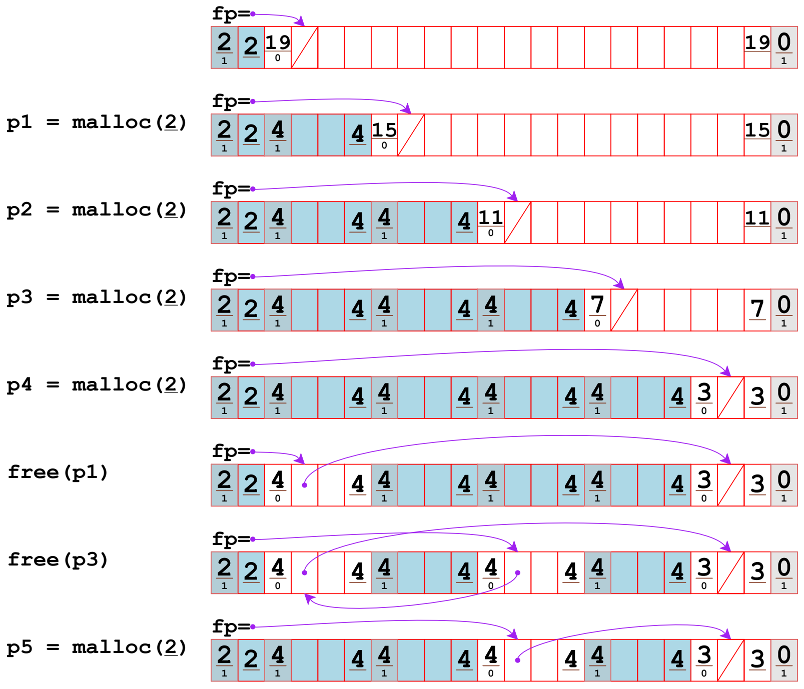
18. 25%

19. Six possible outputs: +ppcc-, +pcpc-, +ccpp-, +pccp-, +cpcp-, and +ccpp-. The outputs reflect that + must be first, - must be last, and the two cs and two ps can be in any order.A possible progress/thread graph is



It's also possible for the Vs and Ps to pair up in the other way, but that makes no difference to the output. The `V(&done_copy)` nodes reflect that the child processes perform a semaphore post on `done`, but it has no effect since the semaphore is not shared with the parent process.

20.



In the last step, allocating the middle free block would also be valid.

21.

$b = 4$
$c = 14$
$d = 10$
$e = 22$
$f = 6$
$g = 17$
$h = 16$
$i = 2$
$j = 3$
$k = 17$
$l = 19$
$m = 16$
$n = 15$
$o = 21$
$p = 20$