CS 4400 Fall 2016
Midterm Exam 2 – 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 a little shorter than this practice exam.

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

```
typedef struct {
  int a[3];
  short b, c;
} stuff;

stuff s[32][32];

int sum(int mode, stuff s[32][32]) {
  int i, j, a = 0;

  for (i = 0; i < 32; i++)
    for (j = 0; j < 32; j++) {
      a += s[i][j].a[mode];
      if (mode)
        a += s[i][j].c;
      else
        a += s[i][j].b;
    }

  return a;
}
```

For each question below, assume a 4kB direct-mapped cache that uses 8-byte blocks, the cache is initially empty, and local variables are in registers. Also assume that the array `s` is at the address `0xA0000` in memory.

1. For each of first six memory accesses via `s` in `sum(0, s)`, 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

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

3. For each of first six memory accesses via `s` in `sum(2, s)`, 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

4. What is the expected cache-miss rate of `sum(0, s)`?

The next two questions refer to a directory that contains the following files.

z.c

```
int ns[] = { 1, 2, 3 };
int y(int*);

int z() {
  return y(ns);
}
```

y.c

```
static int ns[] = { 4, 5, 6 };

int y(int* p) {
  return p[0] + ns[0];
}
```

x.c

```
short ns[] = { 1, 2, 3, 4, 5, 6 };

 int x() {
   return ns[0] + 1;
 }
```

w.c

```
extern short ns[6];

int w() {
  return ns[1] + ns[2];
}
```

main.c

```
#include <stdio.h>

int z();
int w();

int main () {
  printf("%d\n", z() + w());
  return 0;
}
```

Makefile

```
a.out: w.c x.c y.c z.c main.c
        gcc -c w.c
        gcc -c x.c
        gcc -c y.c
        gcc -c z.c
        ar -ruv l.a x.o w.o
        gcc main.c z.o l.a y.o
```

5. Cross out the files above that turn out to be irrelevant to the result of `make && ./a.out` (in the sense that erasing the file content would not change the output).

6. What does `make && ./a.out` print? In case you don't compute the output correctly, to improve opportunities for partial credit, list specific functions that are called and the value that each call returns.

7. The output further below is the partial result of using `readelf` on the shared library produced by `gcc -shared` on a source file. Cross out the files among the four shown below that could not have been the source file that lead to this output.

1.c

```
extern int a;
int w();

int q() {
  return a + w();
}
```

2.c

```
extern int a;
int q();

int w() {
  return a + q();
}
```

3.c

```
extern int q;
int a();

int w() {
  return q + a();
}
```

4.c

```
extern int w;
int a();

int q() {
  return w + a();
}
```

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000794 0x0000000000000794  R E    200000
  LOAD           0x0000000000000df0 0x0000000000200df0 0x0000000000200df0
                 0x0000000000000240 0x0000000000000248  RW     200000
  DYNAMIC        0x0000000000000e10 0x0000000000200e10 0x0000000000200e10
                 0x00000000000001c0 0x00000000000001c0  RW     8
  [....]

 Section to Segment mapping:
  [....]

Dynamic section at offset 0xe10 contains 24 entries:
  [....]

Relocation section '.rela.dyn' at offset 0x480 contains 9 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000200df0  000000000008 R_X86_64_RELATIVE                    6b0
000000200df8  000000000008 R_X86_64_RELATIVE                    670
000000200e08  000000000008 R_X86_64_RELATIVE                    200e08
000000200fd0  000200000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_deregisterTMClone + 0
000000200fd8  000300000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000200fe0  000400000006 R_X86_64_GLOB_DAT 0000000000000000 a + 0
000000200fe8  000600000006 R_X86_64_GLOB_DAT 0000000000000000 _Jv_RegisterClasses + 0
000000200ff0  000700000006 R_X86_64_GLOB_DAT 0000000000000000 _ITM_registerTMCloneTa + 0
000000200ff8  000800000006 R_X86_64_GLOB_DAT 0000000000000000 __cxa_finalize + 0

Relocation section '.rela.plt' at offset 0x558 contains 3 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000201018  000300000007 R_X86_64_JUMP_SLO 0000000000000000 __gmon_start__ + 0
000000201020  000500000007 R_X86_64_JUMP_SLO 0000000000000000 q + 0
000000201028  000800000007 R_X86_64_JUMP_SLO 0000000000000000 __cxa_finalize + 0

[....]
```

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

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 graphs.

```
#include "csapp.h"

int main() {
  pid_t pid1, pid2;
  int status;
  char buffer[1];

  pid1 = Fork();
  write(1, "F", 1);

  if (pid1 == 0) {
    exit(6);
  }

  pid2 = Fork();
  write(1, "S", 1);

  if (pid2 == 0) {
    exit(7);
  }

  Waitpid(pid2, &status, 0);
  buffer[0] = WEXITSTATUS(status) + '0';
  Write(1, buffer, 1);

  Waitpid(pid1, &status, 0);
  buffer[0] = WEXITSTATUS(status) + '0';
  Write(1, buffer, 1);

  return 0;
}
```

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

Again, to improve opportunities for partial credit, show how you arrived at your answer by sketching one or more process graphs.

```
#include "csapp.h"

int main() {
  int fds[2];
  char buffer[1];

  Pipe(fds);

  if (Fork() == 0) {
    if (Fork() == 0) {
      Write(1, "2", 1);
      Write(fds[1], "5", 1);
      return 0;
    }
    Write(1, "1", 1);
    Read(fds[0], buffer, 1);
    return 0;
  }

  Wait(NULL);

  Write(1, "3", 1);
  Write(fds[1], "4", 1);

  return 0;
}
```

10. Consider a memory system with 16 bit virtual addresses, 16 bit physical addresses with a page size of 256 bytes. In the page table below, some entries are not listed; assume that those entries are all marked as invalid. For each of the following virtual addresses, indicate its physical address or indicate that it is a page fault.

Virtual address	Physical address or *page fault*
0x3111	
0x4161	
0x00aa	
0x0880	
0x2198	

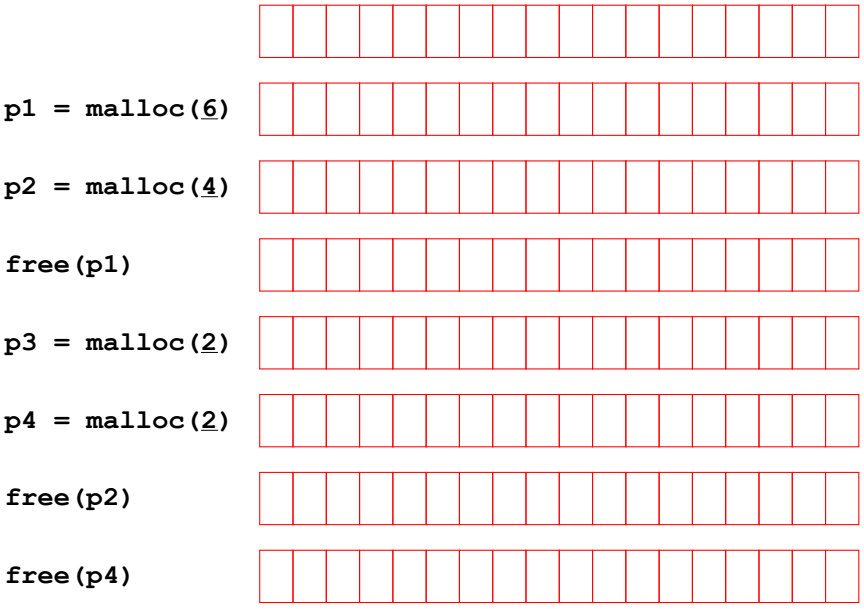
VPN	PPN/valid?	VPN	PPN/valid?	VPN	PPN/valid?	VPN	PPN/valid?
0x00	0x00 / 0	0x21	0x09 / 1	0x42	0x12 / 1	0x63	0x1b / 1
0x01	0x08 / 1	0x22	0x11 / 1	0x43	0x1a / 1	0xe0	0x07 / 0
0x02	0x10 / 1	0x23	0x19 / 0	0x44	0x22 / 1	0xe1	0x0f / 1
0x03	0x18 / 1	0x24	0x21 / 1	0x45	0x2a / 1	0xe2	0x17 / 1
0x04	0x20 / 1	0x25	0x29 / 1	0x46	0x32 / 0	0xe3	0x1f / 1
0x05	0x28 / 0	0x26	0x31 / 1	0x47	0x3a / 1	0xe4	0x27 / 1
0x06	0x30 / 1	0x27	0x39 / 1	0x48	0x42 / 1	0xe5	0x2f / 0
0x07	0x38 / 1	0x28	0x41 / 0	0x49	0x4a / 1	0xe6	0x37 / 1
0x08	0x40 / 1	0x29	0x49 / 1	0x4a	0x52 / 1	0xe7	0x3f / 1
0x09	0x48 / 1	0x2a	0x51 / 1	0x4b	0x5a / 0	0xe8	0x47 / 1
0x0a	0x50 / 0	0x2b	0x59 / 1	0x4c	0x62 / 1	0xe9	0x4f / 1
0x0b	0x58 / 1	0x2c	0x61 / 1	0x4d	0x6a / 1	0xea	0x57 / 0
0x0c	0x60 / 1	0x2d	0x69 / 0	0x4e	0x72 / 1	0xeb	0x5f / 1
0x0d	0x68 / 1	0x2e	0x71 / 1	0x4f	0x7a / 1	0xec	0x67 / 1
0x0e	0x70 / 1	0x2f	0x79 / 1	0x50	0x82 / 0	0xed	0x6f / 1
0x0f	0x78 / 0	0x30	0x81 / 1	0x51	0x8a / 1	0xee	0x77 / 1
0x10	0x80 / 1	0x31	0x89 / 1	0x52	0x92 / 1	0xef	0x7f / 0
0x11	0x88 / 1	0x32	0x91 / 0	0x53	0x9a / 1	0xf0	0x87 / 1
0x12	0x90 / 1	0x33	0x99 / 1	0x54	0xa2 / 1	0xf1	0x8f / 1
0x13	0x98 / 1	0x34	0xa1 / 1	0x55	0xaa / 0	0xf2	0x97 / 1
0x14	0xa0 / 0	0x35	0xa9 / 1	0x56	0xb2 / 1	0xf3	0x9f / 1
0x15	0xa8 / 1	0x36	0xb1 / 1	0x57	0xba / 1	0xf4	0xa7 / 0
0x16	0xb0 / 1	0x37	0xb9 / 0	0x58	0xc2 / 1	0xf5	0xaf / 1
0x17	0xb8 / 1	0x38	0xc1 / 1	0x59	0xca / 1	0xf6	0xb7 / 1
0x18	0xc0 / 1	0x39	0xc9 / 1	0x5a	0xd2 / 0	0xf7	0xbf / 1
0x19	0xc8 / 0	0x3a	0xd1 / 1	0x5b	0xda / 1	0xf8	0xc7 / 1
0x1a	0xd0 / 1	0x3b	0xd9 / 1	0x5c	0xe2 / 1	0xf9	0xcf / 0
0x1b	0xd8 / 1	0x3c	0xe1 / 0	0x5d	0xea / 1	0xfa	0xd7 / 1
0x1c	0xe0 / 1	0x3d	0xe9 / 1	0x5e	0xf2 / 1	0xfb	0xdf / 1
0x1d	0xe8 / 1	0x3e	0xf1 / 1	0x5f	0xfa / 0	0xfc	0xe7 / 1
0x1e	0xf0 / 0	0x3f	0xf9 / 1	0x60	0x03 / 1	0xfd	0xef / 1
0x1f	0xf8 / 1	0x40	0x02 / 1	0x61	0x0b / 1	0xfe	0xf7 / 0
0x20	0x01 / 1	0x41	0x0a / 0	0x62	0x13 / 1	0xff	0xff / 1

For the following two questions, 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.

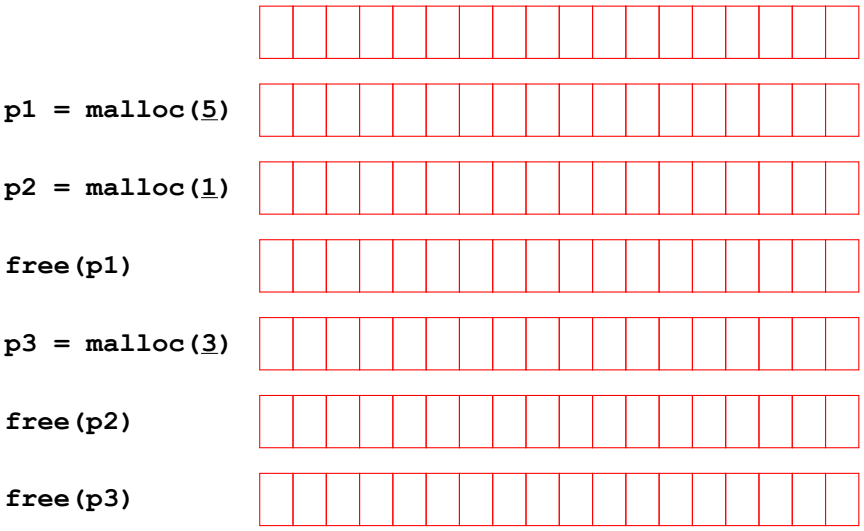
Assume that an allocator produces word-aligned payload pointers, uses a word-sized header and footer for each allocated block, uses a 2-word prolog block and a 1-word terminator block, coalesces unallocated blocks, and is confined to 18 words of memory that is initially filled with 0s. Show a header 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.

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 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.

11. Show the state of memory after each step for an allocator that uses a **first-fit** allocation strategy, where the allocator searches from the start of an *implicit* free list.



12. Show the state of memory after each step for an allocator that uses a **best-fit** allocation strategy, where the allocator finds the smallest unallocated block that matches the requested allocation size.



Answers

1.

Access expression	Access address	Hit or miss
`s[0][0].a[0]`	`0xa0000`	**miss**
`s[0][0].b`	`0xa000c`	**miss**
`s[0][1].a[0]`	`0xa0010`	**miss**
`s[0][1].b`	`0xa001c`	**miss**
`s[0][2].a[0]`	`0xa0020`	**miss**
`s[0][2].b`	`0xa002c`	**miss**

2. 100%

3.

Access expression	Access address	Hit or miss
`s[0][0].a[2]`	`0xa0008`	**miss**
`s[0][0].c`	`0xa000e`	**hit**
`s[0][1].a[2]`	`0xa0018`	**miss**
`s[0][1].c`	`0xa001e`	**hit**
`s[0][1].a[2]`	`0xa0028`	**miss**
`s[0][1].c`	`0xa002e`	**hit**

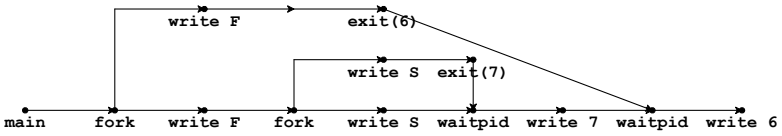
4. 50%

5. Cross out `x.c`

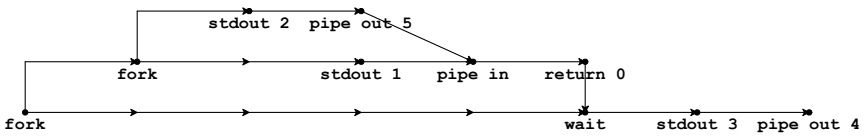
6. Output: 7; `y(ns)` returns 5, `z()` returns 5, `w()` returns 2

7. Cross out all except `2.c`

8. Four possible outputs: `FFSS76`, `FSFS76`, `FSSF76`, `FSS7F6`



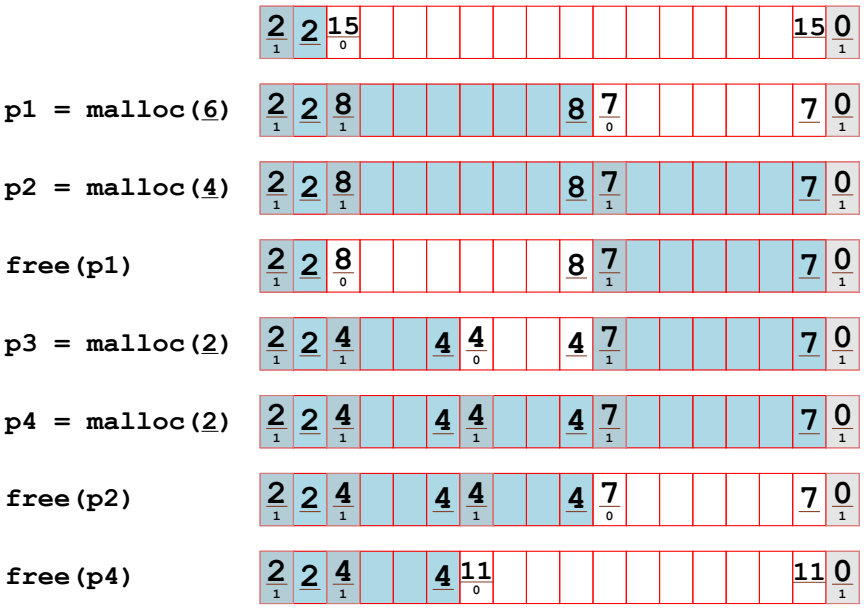
9. Two possible outputs: `123`, `213`



10.

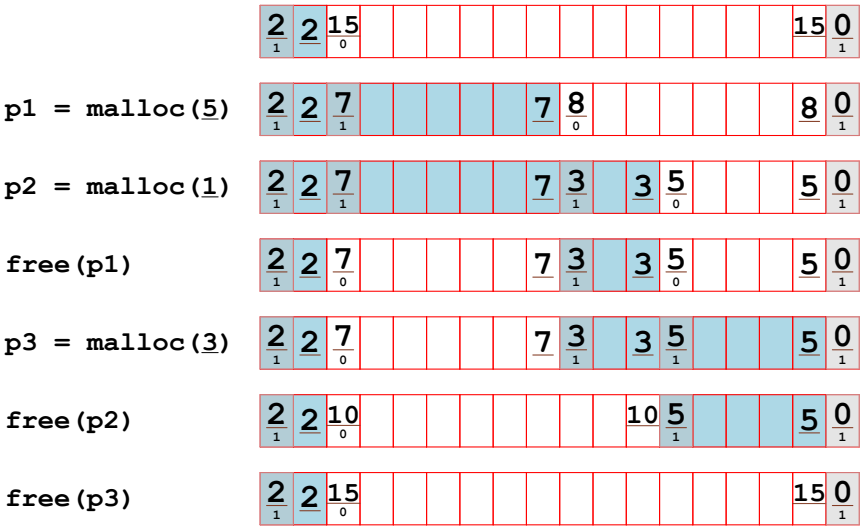
Virtual address	Physical address or *page fault*
`0x3111`	`0x8911`
`0x4161`	page fault
`0x00aa`	page fault
`0x0880`	`0x4080`
`0x2198`	`0x0998`

11.



Shading above is not required in an answer.

12.



Shading above is not required in an answer.