1. **(40 points) Quick Hits.** 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.

1.1. -sizeof(int) > 1 is true.

a. True
b. False

1.2. Suppose i is a double. After the statements

```
i = 7.7;
j = (int) i;
```

are executed, what is the value of i?

a. 7
b. 7.0
c. 7.7
d. 8.0

1.3. Which of the following statements regarding the selection of a data type for a variable is FALSE?

a. The operations that can be performed on a value are limited by its type.
b. The amount of memory necessary to store a value depends on its type.
c. How a value is stored in the memory of the computer depends on its type.
d. None of the above.

1.4. If a is of type (int) and b is of type (unsigned int), then (a < b) will perform

a. A signed comparison
b. An unsigned comparison
c. A segmentation fault
d. A compile error

1.5. Consider an int *a and an int n. If the value of %ecx is a and the value of %edx is n, which of the following assembly snippets best corresponds to the C statement return a[n]?

a. ret (%ecx,%edx,4)
b. leal (%ecx,%edx,4),%eax
   ret
c. mov (%ecx,%edx,4),%eax
   ret
d. mov (%ecx,%edx,1),%eax
   ret

1.6. What is the C equivalent of mov %eax,%ecx
   a. eax = ecx
   b. ecx = eax
   c. eax = *ecx
   d. ecx = *eax

1.7. Which of the following is the correct ordering (left-to-right) of a file's compilation cycle (a filename with no extension is an executable):
   a. foo ⇒ foo.s ⇒ foo.o ⇒ foo.c
   b. foo.c ⇒ foo.o ⇒ foo.s ⇒ foo
   c. foo.c ⇒ foo.s ⇒ foo.o ⇒ foo
   d. Foo.c ⇒ foo.s ⇒ foo ⇒ foo.o

1.8. Which types of locality are leveraged by virtual memory?
   a. Spatial locality
   b. Temporal locality
   c. Prime locality
   d. Spatial and temporal locality

1.9. Each process has its own page table.
   a. True
   b. False
   c. Depends on OS

1.10. Two processes can store different data at the same virtual address.
   a. True
   b. False
   c. Depends on OS

1.11. Memory blocks returned by malloc are initialized to 0.
   a. True
   b. False

1.12. After instructions
```
movl $0x8FFFFFFF, %eax
cmpl $0x7FFFFFFF, %eax
```
which of the following flag has the condition code set to 1?
   a. OF
   b. CF
   c. ZF
   d. SF

1.13. When a user types CTRL-C, which signal does the OS send?
   a. SIGKILL
   b. SIGTSTP
   c. SIGINT
   d. SIGQUIT

1.14. Which of the signals can we install a handler for?

a. SIGKILL
b. SIGTRAP
c. SIGSTOP
d. None of the above

1.15. If a parent process forks a child process, to which resources might they need to synchronize their access to prevent unexpected behavior?

a. File descriptors
b. malloc'ed memory
c. Stack
d. None of the above

1.16. What is the output of the following code? Assume that int is 32 bits, short is 16 bits, and the representation is two's complement.

```
unsigned int x = 0xDEADBEEF;
unsigned short y = 0xFFFF;
signed int z = -1;
if (x > (signed short) y)
   printf("Hello");
if (x > z)
   printf("World");
```

a. Prints nothing
b. Prints "Hello"
c. Prints "World"
d. Prints "HelloWorld"

1.17. In the following code, what order of loops exhibits the best locality?

```
// int a[X][Y][Z] is declared earlier
int i, j, k, sum = 0;
for (i = 0; i < Y; i++)
   for (j = 0; j < Z; j++)
      for (k = 0; k < X; k++)
         sum += a[k][i][j];
```

a. i on the outside, j in the middle, k on the inside (as is)
b. j on the outside, k in the middle, i on the inside
c. k on the outside, i in the middle, j on the inside
d. The order does not matter

1.18. Consider the C declaration

```
int array[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Suppose that the compiler has placed the variable array in the %ecx register. How do you move the value at array[3] into the %eax register? Assume that %ebx is 3.

a. leal (%ecx,%ebx,4),%eax
b. leal 4(%ecx,%ebx,1),%eax
c. movl (%ecx,%ebx,4),%eax
d. movl 8(%ecx,%ebx,2),%eax

1.19. What is the output of this program?

```
int main() {
  char buf[4] = {0x61, 0x62, 0x63, 0x64};
  printf("buf(int) = 0x%x\n", *(int*)buf);
}
```

a. buf(int) = 0x64636261
b. buf(int) = 0x61626364
c. buf(int) = 0x61616161
d. buf(int) = 0x64646464

1.20. From the code below, what is the output of the printf?

```
int x = 0x15213F10 >> 4;
char y = (char) x;
unsigned char z = (unsigned char) x;
printf("%d, %u", y, z);
```

a. -241, 15
b. -15, 241
c. -241, 241
d. -15, 15

(Bonus question; there is no right or wrong answer for this question. You get +2 points just for answering it honestly. We're interested in understanding the impact this course has on your future interest). After taking EE209, I'm more inclined to take courses offered by the computer division of EE (e.g., data structures, networking, architecture, operating systems, etc.).

a. True
b. False

2. (15 points) Assembly

2.1 (4 points) This is assembly code. Please convert this code into C in (1) within the function "func".

```
.section .rodata
printFormat:
  .asciz "ans=%d\n"

.section .text
func:
  pushl %ebp
  movl  %esp, %ebp
  movl  8(%ebp), %eax
  imull 12(%ebp), %eax
  movl  %eax, %edx
  movl  16(%ebp), %eax
  addl  %edx, %eax
  popl  %ebp
  ret

.globl main

main:
  pushl %ebp
  movl  %esp, %ebp
  pushl $4
  pushl $3
  pushl $2
  call  func
  addl  $12, %esp
  pushl %eax
  pushl $printFormat
  call  printf
  mov %ebp, %esp
  pop %ebp
  ret
```

```
int func(int a, int b, int c) {
 (1)
}
```

```
int main() {
  printf("%d\n", func(2, 3, 4));
}
```

Ans:

2.2. (8 points) This is C code for calculating a fibonacci number. Fill (1) ~ (4) to convert this code into the assembly.

```
int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

```
fib:
  pushl %ebp
  movl  %esp, %ebp
  pushl %ebx
  cmpl  $1, 8(%ebp)
  jg  loop
  (1)
  jmp end
loop:
  movl  8(%ebp), %eax
  (2)
  pushl %eax
  call  fib
  addl  $4, %esp
  movl  %eax, %ebx
  movl  8(%ebp), %eax
  (3)
  pushl %eax
  call  fib
  addl  $4, %esp
  (4)
end:
  pop %ebx
  mov %ebp, %esp
  pop %ebp
  ret
```

(1)

(2)
(3)
(4)

2.3 (3 points) In the above code, %ebx is stored into the stack because it is a callee-saved register. Then, why aren't other callee-saved registers (e.g. ,%edi, %esi) stored?

Ans:

3. (12 points) Process

3.1 (4 points) What is the output of this program?

```
int global = 3;

int main() {
  pid_t pid;
  int local = 1;
  pid = fork();

  if (pid == 0) {
    printf("%d %d ", --local, global++);
  }
  else {
    wait(NULL);
    printf("%d %d ", local++, --global);
  }
}
```

Ans:

3.2 (4 points) How many 'A's would be printed if we run this program?

```
int main() {
  fork();
  printf("A\n");
  fork();
  printf("AA\n");
  fork();
  printf("AAA\n");
}
```

Ans:

3.3 (4 points) How many ‘A’s would be printed if we run this program with argument "5" (i.e., ./prog 5)?

```
#define MAXLEN 256

int main(int argc, char** argv) {
  if (argc < 2)
    exit(1);

  int count = atoi(argv[1]);

  for (int i = 0; i < count; i++) {
    if (fork() == 0) {
      char buf[MAXLEN];
      sprintf(buf, "%d", i); // Change integer to string
      char* newArgv[] = {argv[0], buf, NULL};
      execvp(argv[0], newArgv);
    }
    else {
      wait(NULL);
      fprintf(stderr, "A\n");
    }
  }
}
```

Ans:

4. (10 points) Signal

4.1 (5 points) What are the all possible outputs of this code?

```
int counter = 0;
void handler1(int sig) {
  counter+=10;
}

void handler2(int sig) {
  counter += 100;
}

void handler3(int sig) {
  counter += 1000;
}

int main() {
```

```
  signal(SIGUSR1, handler1);
  signal(SIGUSR2, handler2);

  int parent = getpid();
  int child = fork();
  if (child == 0) {
    signal(SIGUSR1, handler3);
    kill(parent, SIGUSR1);
    exit(0);
  }

  raise(SIGUSR2);
  sleep(1);
  waitpid(child, NULL, 0);
  printf("%d\n", counter);
  return 0;
}
```

Ans:

4.2 (5 points) What are the all possible outputs of this code?

```
int counter = 1;

void handler1(int sig) {
  counter *= 2;
  raise(SIGUSR2);
}

void handler2(int sig) {
  counter += 3;
  if (counter < 10)
    raise(SIGUSR2);
  signal(SIGUSR2, SIG_IGN);
}

int main() {
  signal(SIGUSR1, handler1);
  signal(SIGUSR2, handler2);

  raise(SIGUSR1);
  sleep(1);
  printf("%d\n", counter);
  return 0;
}
```

Ans:

5. (8 points) IO

5.1 (4 points) Consider the following code. Assume all system calls succeed, and that calls to read() and write() are atomic with respect to each other.

The contents of foo.txt are “ABCDEFG”.

```
void read_and_print_one(int fd)
{
   char c;
   read(fd, &c, 1);
   printf("%c", c); fflush(stdout);
}

int main(int argc, char *argv[])
{
   int fd1 = open("foo.txt", O_RDONLY);
   int fd2 = open("foo.txt", O_RDONLY);
   read_and_print_one(fd1);
   read_and_print_one(fd2);

if(!fork()) {
   read_and_print_one(fd2);
   read_and_print_one(fd2);
   close(fd2);
   fd2 = dup(fd1);
   read_and_print_one(fd2);
}

else {
   wait(NULL);
   read_and_print_one(fd1);
   read_and_print_one(fd2);
   printf("\n");
}

close(fd1);
close(fd2);
return 0;
}
```

What is the output of this code?

5.2 (4 points) You are given a text file file.txt that contains exactly a single word “descriptors” without any whitespace or special characters. The content is shown in the table below.

File name	File contents
file.txt	descriptors

You are also given a program (headers omitted) that uses the simple and familiar UNIX system functions to perform file I/O operations. For the program below, what will be the output to stdout, based on the file contents as shown above? Assume that all system calls will succeed and the files are in the same directory as the program.

```
/* buf is initialized to be all zeroes */
char buf[20] = {0};

int main(int argc, char* argv[]) {
  pid_t pid;
  int fd1, fd2, fd3 = open("file.txt", O_RDONLY);

  read(fd3, buf, 1);
  fd1 = dup(fd3);

  if ((pid = fork()) > 0){
    waitpid(pid, NULL, 0);
    read(fd1, &buf[1], 2);
  }
  else {
    fd2 = open("file.txt", O_RDONLY);
    read(fd1, &buf[1], 1);
    read(fd2, &buf[2], 2);
  }

  printf("%s", buf);

  /* Don’t worry about file descriptors not being closed */
  return 0;
}
```

6. (9 points) Memory

For this question, let’s look at the 32-bit libc implementation of malloc.

- The libc implementation uses an 8 byte alignment of the payload areas.
- The libc implementation uses the following layout for free blocks:

Header (4 bytes)	Prev (4 bytes)	Next (4 bytes)	Payload (arbitrary size)	Footer (4 bytes)

where prev, next and footer are stored inside the space for the payload.

• The libc implementation uses the following layout for allocated blocks:

Header (4 bytes)	Payload (arbitrary size)

You are writing a linked list implementation of a dictionary. You are experiencing a strange bug where your dictionary works on everything except for 12 letter words, on which it generates a Segmentation Fault. After some debugging, you find that it also doesn't work on words of size 20 and 28 (you don't test any further).

Here is your addWordDict method:

```
typedef struct {
    linkedlist *wordList;
    unsigned long count;
  } dictionary;


int addWordDict(dictionary * dict, char * word){
   int result;
   char * wordCopy;

   if (dict == NULL){
      return ERR_NULL_DICT;
   }

   if(word == NULL){
      return WARN_INVALID_ARGUMENT;
   }

/*add the word */
/*We're going to make a copy of the word because the word buffer gets reused. This
wordCopy will get free'd when we remove the word from the dictionary. */

   wordCopy = (char *)malloc((strlen(word)) * sizeof(char));
   strcpy(wordCopy,word);
   result = addItemLL(((dict)->wordList),(void *) wordCopy);
   dict->count = ((dict)->wordList)->count; /*update the count */
   return result;
}
```

6.1 (3 points) What is wrong with your addWordDict method?

6.2 (3 points) Why does this code work on words of sizes less than 12? (Be as detailed as possible)

6.3 (3 points) Why doesn't this code work on words of sizes 12, 20, 28... ? (Be as detailed as possible)

7. (6 points) Linking

Consider the executable object file a.out, which is compiled and linked using the command

gcc -o a.out main.c foo.c

and where the files main.c and foo.c consist of the following code:

```
/* main.c */

#include <stdio.h>

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

int main()
{
   int c = 4;

   foo();
   printf("a=%d b=%d c=%d\n", a, b, c);
   return 0;
}
```

```
/* foo.c */

int a, b, c;

void foo()
{
   a = 5;
   b = 6;
   c = 7;
}
```

What is the output of a.out?

a=, b=, c=