CS221 C and Systems Programming – Study Questions

Instructor: Paul Haskell

1. What does the acronym "CPU" stand for?

   _____central processing unit_____

2. Fill in the following table. Assume all values are 16 bits long

	Binary	Octal	Hex
1	0000 0000 0000 0001	000001	0001
-1	1111 1111 1111 1111	177777	ffff
10	0000 0000 0000 1010	000012	000a
-10	1111 1111 1111 0110	177766	fff6
256	0000 0001 0000 0000	000400	0100
16384	0100 0000 0000 0000	400000	4000
-32768	1000 0000 0000 0000	100000	8000

3. What are the 4 "memory regions" in a running program, and what is stored in each?

   code: machine code for every method in the program

   data: global, static, and constant variables

   heap: data created via malloc() or "new"

   stack: a "stack frame" for each running or waiting method, and a "stack frame" contains each method's local variables, input variables, and return value

   _____

4. What is the value of w?

```
short w = 16384 << 1;
```

_____-32768_____

5. What is the `printf()` command to print a pointer variable?

_____printf("%p", thePointerVariable);_____

6. What is done by the C Preprocessor during compilation?

The preprocessor handles all "preprocessing directives" such as #define and #include, turning them into regular C (or C++ code).

7. What is wrong with the following code?

```
struct Dog_t {
    char name[64];
    double age;
}
```

_____semicolon is missing at the end of the struct declaration_____

8. Please write a typedef for the above `struct`.

____`typedef struct Dog_t Dog;`_____

9. Write a Makefile that builds **prog.exe** from **prog.c**

```
prog.exe: prog.c
  gcc -o prog.exe prog.c
```

_____

10. What is meant by "big-endian" and "little-endian"?

Different CPU architectures: big-endian CPUs store multi-byte data types (e.g. short, long) with the "big end" first in memory (i.e. at the lower memory address). Little-endian CPUs store multi-byte data types with the "small end" first in memory.

_____

11. Write the code to declare a pointer to `a1`

```
int main() {
    int a1 = 1, b2 = 2;
```

```
}
```

_____int* aPtr = &a1;_____

12. Write code that generates a segmentation fault.

```
char* constString = "sparky";
constString[0] = 'S';
```

_____

13. What is `triple`?

```
double tripleMe(int inp) { return 3.0 * inp; }
double (*triple)(int) = tripleMe;
```

a Function Pointer, to a function that takes a single int input and that returns a double

_____

14. Please write a simple C++ class.

```
class Simple {
public:
   char title[64];
};
```

---

15. Describe at least two multithreading "work flows".

parallel: multiple threads consume input from the same place and send output to the same destination

serial: multiple stages run at the same time on different data samples. Each stage passes its output on to the next stage's input

feedback: different stages pass data back and forth to each other

---

16. What socket operations are done by a socket "server"? By a "client"?

server: establish a socket, bind it to an address, listen for remote connections and accept them

client: establish a socket, connect it to a remote server

---

17. Write a signal handler

```
void hander(int signum) { printf("signum is %d\n", signum); exit(1); }
```

---

18. Please write the structs needed for a basic singly-linked list.

```
struct Node { int value; struct Node* next; };
struct SLL { struct Node* head; }
```

19. Write a method that determines the endianness of the current computer and that prints either "big" or "little".

```
void whichEndian() {
long testVal = 0x0a000000; char* ptr = (char*) &testVal;
printf("%sendian\n", ptr[0] == 0x0a ? "big" : "little");
```

20. Write a function that takes two pointers to integers and swaps their values.

```
void swap(int *a, int *b) {
int tmp = *a; *a = *b; *b = tmp; }
```

21. Implement a function that takes a pointer to a dynamically allocated array and its size, then doubles the array size while preserving its contents. The function should return a pointer to the newly allocated array and should "clean up" the old array.

```
int* resizeArray(int *arr, int size) {
  int* retval = malloc(sizeof(int) * size * 2);
  for (int j = 0; j < size; j++) { retval[j] = arr[j]; }
  free(arr);
  return retval;
}
```

22. Write a function that removes all occurrences of the integer 'value' from an array and returns the new number of elements in the array. The array should be modified in place using pointers.

```
int removeElement(int *arr, int size, int value) {
  for (int j = 0; j < size; ) {
    if (arr[j] == value) { arr[j] = arr[size-1]; --size; }
    else { j++; }
  }
  return size;
}
```

23. Use `fscanf()` and `fopen()` to open a file "**Input.txt**", read each word, and print out the length of each word. The program should end cleanly at the end of the input file.

```
#include <stdio.h>
int main() {
 FILE* fptr = fopen("Input.txt", "r"); if (!fptr) { exit(1); }
 char word[4096];
 fscanf("%s", word);
 while (!feof(fptr)) { printf("%d\n", strlen(word); fscanf("%s", word); }
 return 0;
}
```

24. Write the RECURSIVE code for `reverseHelper()`

```
void reverseString(char* input) {
    char* end = input;
    while (end[0] != 0) { end++; }
    reverseHelper(input, end);
}
```

```
void reverseHelper(char* input, char* end) {
  if (input >= end) { return; }
  char t = *input; *input = *(end-1); *(end-1) = t; // -1 for the \0 at the end
  reverseHelper(input+1, end-1);
}
```

25. Write `myStrdup()`, your own implementation of the `strdup()` method.

```
char* myStrdup(char* inp) {
  char* retval = malloc(strlen(inp)+1);
  for (int j = 0; j < strlen(inp); j++) { retval[j] = inp[j]; }
  retval[strlen(inp)] = 0;
  return retval;
}
```

26. How could you write a multithreaded program that gets stuck in a "deadlock" when each of two threads is stuck waiting for the other.

Have thread A hold semaphore A and wait for semaphore B.

And have thread B hold semaphore B and wait for semaphore A.

27. Write a method `getline()` that mimics the built-in `fgets(char*)` method, reading the keyboard input until it finds a NEWLINE and returning the resulting string.

```
void getline(char* retval) {
  int indx = 0;
  char ch = getchar();
  while (ch != EOF && ch != '\n') { retval[indx++] = ch; ch = getchar(); }
  retval[indx] = 0;
}
```

28. Explain the difference between a pointer variable and a regular variable in C

A pointer variable contains the address of some other variable.

29. Please use `malloc()` to allocate memory for an array of 20 `double` values

```
double* arr = (double*) malloc(sizeof(double)*20);
```

30. Describe how bitwise operations can be used to check whether a number is odd or even.

```
int isOdd(int inp) { return (inp & 1); } // check the '1' bit in 'inp'
```

31. Write a short explanation of what happens when you dereference a NULL pointer in C.

A Segmentation Violation occurs, and the program throws a SIGSEGV signal. If this signal is

not caught, the program will terminate and will generate a corefile, if corefile generation is allowed by the operating system. Otherwise, the signal handler will be run. If the signal handler does not terminate the program, then the program will then continue at the line of code following the line where the segmentation violation occurred.