Name: _____ CSCI 237

Practice Problems: Cache Memories

Structs and Memory

Consider the following C structure that contains multiple fields and is similar to one you may want to use in your Cache lab:

```
struct simulation {
   int s;
   int b;
   int E;
};
```

The `struct simulation` contains 3 fields, each corresponding to some parameter relevant to your cache simulation.

Declaring a local `struct simulation` variable allocates enough space for the structure on the stack frame of the current function. The local variable's fields can be accessed with dot notation, and the memory is reclaimed once the function returns.

Passing a local struct variable as an argument to a function creates a complete bitwise copy of the structure. This means that changes to the parameter's value inside the function do not affect the original struct.

Consider the following code:

```
void initialize_simulation(struct simulation sim) {
   sim.s = 3;
   sim.b = 3;
   sim.E = 4;
}

int main(int argc, char *argv[]) {
   struct simulation sim;
   initialize_simulation(sim);
   printf("s: %d, b: %d, E: %d\n", sim.s, sim.b, sim.E);
}
```

What is printed when the function `main()` is executed?

Instead, we would like this program to print the output:
`s: 3, b: 3, E: 4`

We have inserted whitespace where additions *may* be needed; in some cases we also deleted necessary notation that you will need to restore. Modify the code so that it correctly complies and prints the desired output.

```
void initialize_simulation(struct simulation   sim   ) {
   sim  s = 3;

   sim  b = 3;

   sim  E = 4;
}

int main(int argc, char *argv[]) {
   struct simulation   sim;

   initialize_simulation(   sim   );

   printf("s: %d, b: %d, E: %d\n",
          sim  s, sim  b, sim  E);
}
```

What changes were necessary and why?

File Reading and Parsing

There are several ways to interact with files using C. When we simply want to read form a file, it may be helpful to compare it to steps we take when reading from a book.
The first step is to open the book. Once the book is open, we can begin reading from the beginning. Each word that we read moves us forward, and we use a finger to track our current position. If we pause our reading, we pick up again from the current word that our finger is pointing to. When we are done for good, we close the book.

A file can be opened using the function `fopen()`, passing the file's pathname and the permissions as arguments (e.g., `'r'` for read-only). What is returned is a pointer to a data structure that keeps track of our current position within the open file (like our finger). Each time we read data, our position (finger) advances, and subsequent reads pick up from where we left off. Once we are done reading, we must close the file (like we close a book).

```
FILE *fp;
char str[60];

/* opening file for reading */
fp = fopen("file.txt" , "r");
if(fp == NULL) {
  perror("Error opening file");
  return(-1);
}
if(fgets(      ,        , fp) != NULL) {
  printf("%s", str);
}
fclose(fp);
```

The above code is not yet complete, yet it is close enough that you should be able to reason about it.

What do you think is the purpose of the variable `char str[60]`?

Based on your understanding of the documentation for `fgets()` (found by typing `man fgets` at the command line, or by searching for the "linux fgets manual page" online), what should be the first two arguments to `fgets()`?

We saw the function `sscans()` in the “bomb lab”. It lets us read formatted input from a string. Consider the incomplete C code below:

```
int main (int argc, char *argv[]) {
  int day, year;
  char weekday[20], month[20], dtm[100];
  strcpy(dtm, "Saturday March 25 1989");
  sscanf(dtm, "%s %s %d %d", weekday,     , &day,       );
  printf("%s %d, %d = %s\n", month,      , year,        );
  return 0;
}
```

The function `strcpy()` copies a null-terminated string from the address pointed to by its second argument into the memory at the address pointed to by its first argument.

The function `sscanf()` uses a format string to convert components of the string passed as its first argument and store them at locations specified by its last arguments.

The output printed by this program should be:

```
March 25 1989 = Saturday
```

Complete the program so that it achieves this output.

Cache Memories

Consider a system with sixteen bytes of memory, and a direct mapped cache (E=1) with 4 sets (s=2) and a block size of 2 (b=1). For each address below, underline the set bits, and note which set index it maps to.

Address	Set Index
0000	_____
0001	_____
0010	_____
0011	_____
0100	_____
0101	_____
0110	_____
0111	_____
1000	_____
1001	_____
1010	_____
1011	_____
1100	_____
1101	_____
1110	_____
1111	_____

To the right of the table above, sketch a picture that corresponds to the dimensions of the cache described. Use your drawing and your understanding of cache memories to indicate whether reads to the (hex) addresses below correspond to hits or misses in the cache.

Address	Hit (H)/Miss (M)
C	_____
3	_____
9	_____
0	_____
8	_____
3	_____
B	_____
F	_____
4	_____
5	_____
7	_____
D	_____
C	_____
6	_____
1	_____
A	_____
E	_____

Now consider a system with sixteen bytes of memory, and a two-way set-associative cache (E=2) with 2 sets (s=1) and a block size of 2 (b=1). For each address below, underline the set bits, and note which set index it maps to.

Address	Set Index
0000	_____
0001	_____
0010	_____
0011	_____
0100	_____
0101	_____
0110	_____
0111	_____
1000	_____
1001	_____
1010	_____
1011	_____
1100	_____
1101	_____
1110	_____
1111	_____

Once again, sketch a picture that corresponds to the dimensions of the cache described. Use your drawing and your understanding of cache memories to indicate whether reads to the (hex) addresses below correspond to hits or misses in the cache.

Address	Hit (H)/Miss (M)
C	_____
3	_____
9	_____
0	_____
8	_____
3	_____
B	_____
F	_____
4	_____
5	_____
7	_____
D	_____
C	_____
6	_____
1	_____
A	_____
E	_____