Homework 1 Handout

Due Date: 1/30 (11:59PM)

Pointers: Variables that point to a location in memory. Please see the Deep C Dive slides for more details.

- **Double Pointers:** Pointers themselves are located somewhere in memory. Thus, it is also possible to have a pointer to a pointer, which points to the virtual memory address of the original pointer. Consider the following lines of code:

  ```
  int x = 5;

  int* x_ptr = &x;

  int** x_ptr_ptr = &x_ptr
  ```

  Here, x_ptr_ptr is a double pointer—it is a pointer to another pointer. If the variable x happens to be located at (virtual) address 0x1234, then x_ptr has the value 0x1234. This value is located somewhere in memory, let's say at (virtual) address 0x5678—in that case, x_ptr_ptr would have the value 0x5678.

Structures: User-defined data type to group items into a single type. It is possible for structures to be nested, i.e. for a struct to be a member of another struct. Please see the Deep C Dive slides for more details.

The *typedef* keyword is used to define a new name or an alias for a data type. It is common to use it with structures so that we would not have to use the struct keyword every time. You can see the following link for details: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#The-typedef-Statement

Recall that, pointer to a structure points to a contiguous block of memory where the members of the structure are located. The type of the pointer indicates how to interpret the contiguous memory block of the structure. It is possible for have a pointer of a different struct type to point to a

structure as long as the structure can be correctly interpreted by the different struct pointer as well. Another way to interpret it is that a pointer to a structure is equivalent (after suitable conversion) to a pointer to its first member.
Consider the following example:

```
Struct B {
        int p;
        float q; }

struct A {
        struct B;
        int i; }

struct C {
        struct B;
        float f; }

Struct C c1;

// Initialization done here

someFunction((struct A *) &c1); // This is OK
someFunction((struct B *) &c1); // This is OK
```

Unions:

Unions are declared just like a structure but each of its members occupy the same memory location. This size of a union is equal to the size of its largest member. Writing a value to one member of a union overwrites the value of another member. Please see the following link for more details:
https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Unions

Enum: Data type used to refer to integer values by particular names. For example:

```
enum colors = {blue, pink, green, yellow}
```

Here, the name of the enum is colors. Each of the color names corresponds to an integer value-- by default, these are 0, 1, 2, 3 (in blue, pink, green, yellow order).

Please see the following link for more details: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#index-data-types_002c-enumeration

Dynamic memory: Located in the heap region of the virtual address space. Can be helpful if you are using a data structure that could change in size over time (i.e., an array, a linked list, etc.).

- malloc(size_t size): Allocates the desired amount of memory in the heap. Pass in the desired number of bytes; returns a void pointer to the start of a chunk of memory that is at least as large as the desired number of bytes.
    - The sizeof() function can be useful to determine how many bytes of memory to request. For example:

      ```
      int* ptr = (int*)malloc(sizeof(int));
      ```

      This line of code allocates 4 bytes of memory in the heap, since that is the size of an integer. You can also pass structure types into the sizeof function.

- realloc(void* ptr, size_t size): Resizes a chunk of memory that is allocated in the heap. Pass in a pointer that was previously returned from malloc/realloc and the desired size; returns a pointer to a (potentially) different region in the heap that is at least as large as the desired number of bytes.
- free(void* ptr): Frees memory that was previously allocated in the heap. Pass in a pointer that was previously returned from malloc/realloc; returns nothing. If memory that was previously malloced is not freed once not needed anymore, this is a **memory leak**.
- Please see the man page for more details: https://man7.org/linux/man-pages/man3/free.3.html

- Please see OSTEP Chapter 14 for more details: https://pages.cs.wisc.edu/~remzi/OSTEP/vm-api.pdf

Kobject:

- Kobjects and ksets are abstractions used to manage the device model and the sysfs interface. They are used to map the device hierarchies.
    - Kobjects have a name and a reference count, as well as a pointer to a parent.
    - Kset is a group of kobjects that is used to create a hierarchichal data structure.
    - Ktype refers to the type associated with a kobject. We are not using it in our homework.
    - When reference count to a kobject becomes zero, the object is released. We are not going to consider this scenario in our homework.
- Read more about kobjects and ksets here: https://docs.kernel.org/core-api/kobject.html
- Consider the given diagram in bus_topology.pdf. It shows the hierarchy of the objects from top to bottom as follows-
    - Bus A is parent to Device A, Device B and Bus A-B Bridge
    - Bus A-B Bridge is parent of Bus B
    - Bus B is parent of Device C
- Alternately, we can also think of it this way-
    - Bus A has three members- Device A, Device B and Bus A-B Bridge
    - Bus B has two members- Device C and Bus A-B Bridge

Miscellaneous Information:

- strncpy(char *dest, const char *src, size_t n) function copies at most n bytes of the string pointed to by src, to the buffer pointed to by dest. Please see the man page for more details.