1. Declare a pointer to a short int and a pointer to a float.

```
short int *ptr;
float *ptr_f;
```

2. Of what use is the sizeof() operator?

The sizeof() operator returns the size in bytes of the variable/type passed on to it as a parameter.

3. In a given operating system, a pointer to a short int is 32 bits wide. How wide is a pointer to a long int in this same system?

Having the same system would mean having the same size of the pointer variable. The pointer variable has the same size of 32 bits regardless of the data type.

4. Assume p is a pointer to a float. Further, assume, the value of p is 1000 (i.e., the address of the float it points to is 1000). The value of the float is 17.6. What value is p++? Define in words what *p and &p mean. Is there a way to determine the values of *p and &p given the info above?

Given that (in this example the values are assigned)

```
float *p, x;
x = 17.6;
p = &x; // where x is located in 1000
```

The value of p++ would be the address worth the size of a float data type after 1000. Assuming float is 4 bytes, then p will now be pointing to 1004. *p means taking the "value of" the address that the pointer p is holding on. This means that we can access x by using *p (before executing p++), to get the value of 17.6. For &p, this is taking the address of the pointer variable p. With this information, where p++ had been executed, we cannot determine *p since we have no information regarding the value of the address it holds as well as we do not know whether or not that memory space had been reserved prior. The result would be unpredictable and non-deterministic. Furthermore, we also cannot determine &p (address of the variable p) exactly but we have a clue which part of the memory the variable p is located based on the qualifiers involved upon its declaration. For this example, assuming p is declared in a block, it should be in the stack memory.

5. Given the initializations and memory map at the top, fill out the memory map on the bottom after the code has executed. Assume pointers are 32 bits wide.

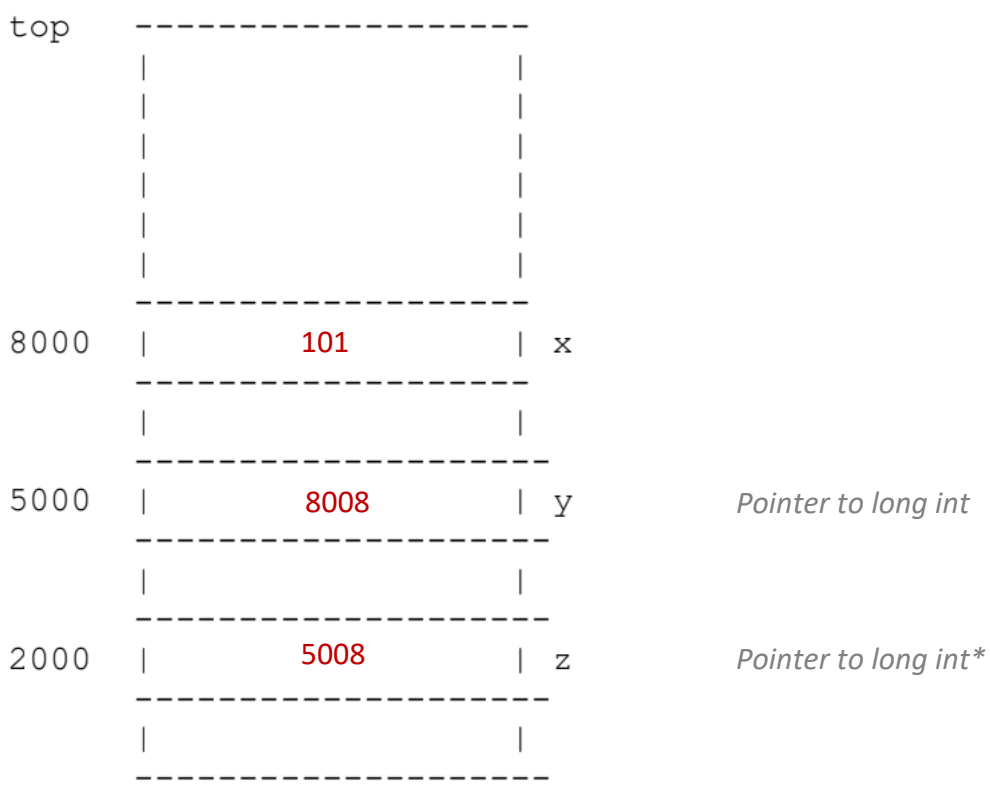
Pointers are 4 bytes (32bit)
Long int is 8 bytes

```
      long int x=100;
      long int *y;
      long int **z;

top   ------------------
      |                |
      |                |
      |                |
      |                |
      |                |
      |                |
      ------------------
8000  |     100        | x
      ------------------
      |                |
      --------------------
5000  |                | y
      --------------------
      |                |
      --------------------
2000  |                | z
      ------------------
      |                |
      --------------------
```

```
code

y=&x;
z=&y;
x++;
*y=*y++;
*z=*z++;
z++;
```




*Pointer to long int*

*Pointer to long int**