EL-GY 6483 **Name:** ______________________

1. Using the variable x, give definitions for the following:
   (a) An integer
   (b) A pointer to an integer
   (c) An array of 10 integers
   (d) An array of 10 pointers to integers

Solution:

```c
int x; // An integer
int *x; // A pointer to an integer
int x[10]; // An array of 10 integers
int *x[10]; // An array of 10 pointers to integers
```

2. What is the output of the following C program?

```c
#include <stdio.h>

int main ()
{
  int vals[5] = {4, 3, 2, 5, 1};
  int i;
  for (i=0; i<=5; i++) {
    printf("vals[%d]=%d\n", i, vals[i]);
  }
  return 0;
}
```

Solution:

```
vals[0]=4
vals[1]=3
vals[2]=2
vals[3]=5
vals[4]=1
vals[5]=??? (undefined)
```

The last value that is printed is outside the bounds of the array, and could be anything depending on what's in that address.

3. (a) What is the output of the following C program?

```
# include <stdio.h>
void fun(int y)
{
    y = 30;
}
int main()
{
  int y = 20;
  fun(y);
  printf("%d", y);
  return 0;
}
```

Solution: 20. `y` is a local variable in both functions; setting the value of `y` in `fun()` does not affect its value in `main()`.
If you missed this question, please read up on scope and related issues.

(b) In the program above, is the variable `y` in `main()` stored on the stack or on the heap?

Solution: Stack. Local variables are pushed onto the stack.

(c) What is the output of this C program?

```
# include <stdio.h>
void fun(int *y)
{
    *y = 30;
}
int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);
  return 0;
}
```

Solution: 30. The address of `y` in `main()` is passed to `fun()`, and a new value is stored in that address. When `main()` retrieves the value in that address, it gets the new value.
If you missed this question, read up on pointers and related issues.

(d) In the program above, is the variable `y` in `main()` stored on the stack or on the heap?

Solution: Stack. Local variables are pushed onto the stack.
Note that in C, when we pass arguments "by reference" as in this example, we are just passing a pointer as a local variable, just like any other local variable. More specifically, we are passing the *value* of the pointer to the function, just as in the previous example we passed the value of an `int` to the function.

(e) True or false: `&y` in `main()` and `y` in `fun()` have the same value.

Solution: True. If you're not convinced, run the code and convince yourself.