CSC209H Worksheet: malloc Basics

1. Each time a variable is declared or memory is otherwise allocated, it is important to understand how much memory is allocated, where it will be allocated and when it will be de-allocated. Complete the table below. (Note: some of the programs allocate more than one block of memory.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	sizeof(int)	stack frame for `main`	when program ends

`int fun() {`
`  float i;`
`}`
`int main() {`
`  fun();`
`}`			

`int fun(char i) {`
`  ...`
`}`
`int main() {`
`  fun('a');`
`}`			

`int main() {`
`  char i[10] = {'h','i'};`
`}`			

`int main() {`
`  char *i;`
`}`			

`int main() {`
`  int *i;`
`}`			

`int fun(int *i) {`
`  ...`
`}`
`int main() {`
`  int i[5] = {4,5,2,5,1};`
`  fun(i);`
`}`			

`int main() {`
`  int *i;`
`  i = malloc(sizeof(int));`
`}`			

`void fun(int **i) {`
`  *i = malloc(sizeof(int)*7);`
`}`
`int main() {`
`  int *i;`
`  fun(&i);`
`  free(i);`
`}`			

Good question to ask as you go around: Suppose that the last code fragment were changed so that the parameter to fun was j instead of i. Now should the call to free be free(j) or free(i)? How do you know? Answer: It should be i. Once fun returns j doesn't exist. (It has been deallocated.)

CSC209H Worksheet: malloc Basics

2. Trace the memory usage for the program below up to the point when `initialize` is about to return. We have set up both stack frames for you, and the location of the heap.

```
#include <stdio.h>
#include <stdlib.h>


// Initialize two parallel lists.
void initialize(int *a1, int *a2, int n) {
    for (int i = 0; i < n; i++) {
        a1[i] = i;
        a2[i] = i;
    }
}

int main() {
    int numbers1[3];
    int *numbers2 = malloc(sizeof(int) * 3);

    initialize(numbers1, numbers2, 3);

    for (int i = 0; i < 3; i++) {
        printf("%d %d\n",
               numbers1[i], numbers2[i]);
    }

    free(numbers2);
    return 0;
}
```

Section	Address	Value	Label
Heap	0x23c		
	0x240		
	0x244		
	0x248		
	⋮	⋮	
stack frame for initialize	0x454		
	0x458		
	0x45c		
	0x460		
	0x464		
	0x46c		
	0x470		
stack frame for main	0x474		
	0x478		
	0x47c		
	0x480		
	0x484		
	0x488		
	0x48c		

```
free(numbers2);
-> now memory is no longer reserved at 0x23c, but the value of numbers2 doesn't change.

Some takeway messages:
1) freeing doesn't clear memory
2) Statically allocation of array memory in main means there is no way to
reclaim or reuse that memory in this program.  Using malloc is an alternative
that allows you to free memory.
```