CSC209H Worksheet: Stacks and Heaps

1. Trace the memory usage for the program below

Code:

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

int *mkarray(int a, int b, int c) {
    int arr[3];
    arr[0] = a;
    arr[1] = b;
    arr[2] = c;
    int *p = arr;
    return p;
}

// Code for other_function() omitted.
int main() {
    int *ptr = mkarray(10, 20, 30);
    other_function();
    printf("%d %d %d\n", ptr[0], ptr[1], ptr[2]);
}
```

Memory Trace:

Section	Address	Value	Label
Heap	0x23c	...	...
	0x240	...	...
	0x244	...	...
Stack frame			
for mkarray	0x454	...	arr[0]
	0x458	...	arr[1]
	0x45c	...	arr[2]
	0x460	...	p

	...	...	...
Stack frame			
for main	0x480	...	ptr
	...	...	...

2. Why doesn't the program work?

The program does not work because the array `arr` is allocated on the stack in `mkarray`, and the memory is invalidated after the function returns. This causes `ptr` in `main` to point to invalid memory.

Fixed `mkarray` Function:

```c
int *mkarray(int a, int b, int c) {
    int *arr = malloc(3 * sizeof(int));
    arr[0] = a;
    arr[1] = b;
    arr[2] = c;
    return arr;
}
```

Updated Memory Trace:

Section	Address	Value	Label
Heap	0x23c	10	arr[0]
	0x240	20	arr[1]
	0x244	30	arr[2]
Stack frame			
for mkarray	0x454	...	arr (pointer)
	...	...	...
Stack frame			
for main	0x480	0x23c	ptr
	...	...	...

3. Deallocate the memory

The memory allocated on the heap should be deallocated to avoid memory leaks.

Updated `main` Function:

```c
int main() {
    int *ptr = mkarray(10, 20, 30);
    other_function();
    printf("%d %d %d\n", ptr[0], ptr[1], ptr[2]);
    free(ptr); // Deallocate memory
}
```

4. Trace the memory usage for the new program

Code:

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

int *multiples(int x) {
    int *a = malloc(sizeof(int) * x);
    for (int i = 0; i < x; i++) {
        a[i] = (i + 1) * x;
    }
    return a;
}

int main() {
    int *ptr;
    int size = 3;
    ptr = multiples(size);
    for (int i = 0; i < size; i++) {
        printf("%d\t", ptr[i]);
    }
    printf("\n");
    return 0;
}
```

Memory Trace:

Section	Address	Value	Label
Heap	0x224	3	a[0]
	0x228	6	a[1]
	0x22c	9	a[2]
Stack frame			
for multiples	0x46c	0x224	a (pointer)
	...	...	...
Stack frame			
for main	0x47c	0x224	ptr
	0x480	3	size
	...	...	...

5. Update `main` to handle multiple sizes and trace memory

Updated `main`:

```c
int main() {
    int *ptr;
    for (int size = 3; size <= 5; size++) {
        ptr = multiples(size);
        for (int i = 0; i < size; i++) {
            printf("%d\t", ptr[i]);
        }
        printf("\n");
        free(ptr); // Deallocate memory
    }
    return 0;
}
```

Memory Trace:

For size = 3:

Heap	0x224	3	a[0]
	0x228	6	a[1]
	0x22c	9	a[2]

```
For size = 4:
Heap            0x230     4          a[0]
                0x234     8          a[1]
                0x238     12         a[2]
                0x23c     16         a[3]
For size = 5:
Heap            0x240     5          a[0]
                0x244     10         a[1]
                0x248     15         a[2]
                0x24c     20         a[3]
                0x250     25         a[4]
```

6. Explanation of the problem

Without the `free(ptr)` call, each iteration of the loop allocates memory on the heap without releasing it, leading to a memory leak. Adding `free(ptr)` after each use resolves the issue.