# 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() {`<br>`    int i;`<br>`}` | sizeof(int) | stack frame for main | when program ends |
| `int fun() {`<br>`    float i;`<br>`}`<br><br>`int main() {`<br>`    fun();`<br>`}` | sizeof(float) == 4 byts | stack frame for fun | when fun returns |
| `int fun(char i) {`<br>`    ...`<br>`}`<br><br>`int main() {`<br>`    fun('a');`<br>`}` 3 | sizeof(char) == 1 byte | stack fun | when fun ends |
| `int main() {`<br>`    char i[10] = {'h','i'};`<br>`}` | → sizeof(i)<br>10 byts | stack main | when main ends |
| `int main() {`<br>`    char *i;`<br>`}` | 8 byts | 〃 | 〃 |
| `int main() {`<br>`    int *i;`<br>`}` | 8 byts | 〃 | 〃 |
| `int fun(int *i) {` →<br>`    ...`<br>`}`<br><br>`int main() {`<br>`    int i[5] = {4,5,2,5,1};` →<br>`    fun(i);`<br>`}` | → 8 byts<br><br>→ 20 byts | stack fun<br><br>stack main | when fun returns<br><br>when main ends |
| `int main() {`<br>`  int *i;` →<br><br>`  i = malloc(sizeof(int));` →<br>`}` | 8 bytes<br><br>4 byts | stack main<br><br>heap | when main ends<br><br>when prog. ends |
| `void fun(int **K) {` →<br>`  *K = malloc(sizeof(int)*7);` →<br>`}`<br><br>`int main() {`<br>`    int *i;` →<br>`    fun(&i);`<br>→ `free(i);`<br>`}` | 8 byts<br>28 byts<br><br>8 byts | stack fun<br>heap<br><br>stack main | when fun returns<br>when free is called<br><br>when main returns |

CSC209H Worksheet: `malloc` Basics

Addresses modified to match gdb addresses

You might get different values

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.

| Section | Address | Value | Label |
|---|---|---|---|
| Heap | ~~0x23c~~ 2a0 | 0 | |
| | 0x240 | 1 | |
| | 0x244 | 2 | |
| | 0x248 | | |
| | ⋮ | ⋮ | |
| stack frame for `initialize` | 0x454 7dc | 3 | n |
| | 0x458 7e0 | 0x2a0 | a2 |
| | 0x45c 7e4 | | |
| | 0x460 7e8 | 0x82c | a1 |
| | 0x464 7ec | | |
| | 0x46c 7f0 | | |
| | 0x470 7fc | ~~0~~ ~~1~~ ~~2~~ 3 | i |
| stack frame for `main` | 0x474 81c | | i |
| | 0x478 820 | 2a0 | numbers2 |
| | 0x47c 824 | | |
| | 0x480 828 | | |
| | 0x484 82c | 0 | numbers1 |
| | 0x488 830 | 1 | |
| | 0x48c 834 | 2 | |

```
#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;
}
```

example of a defined constant

#define MAXLINE 256