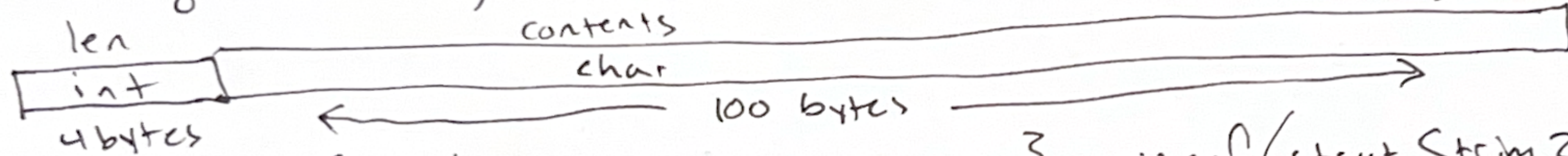
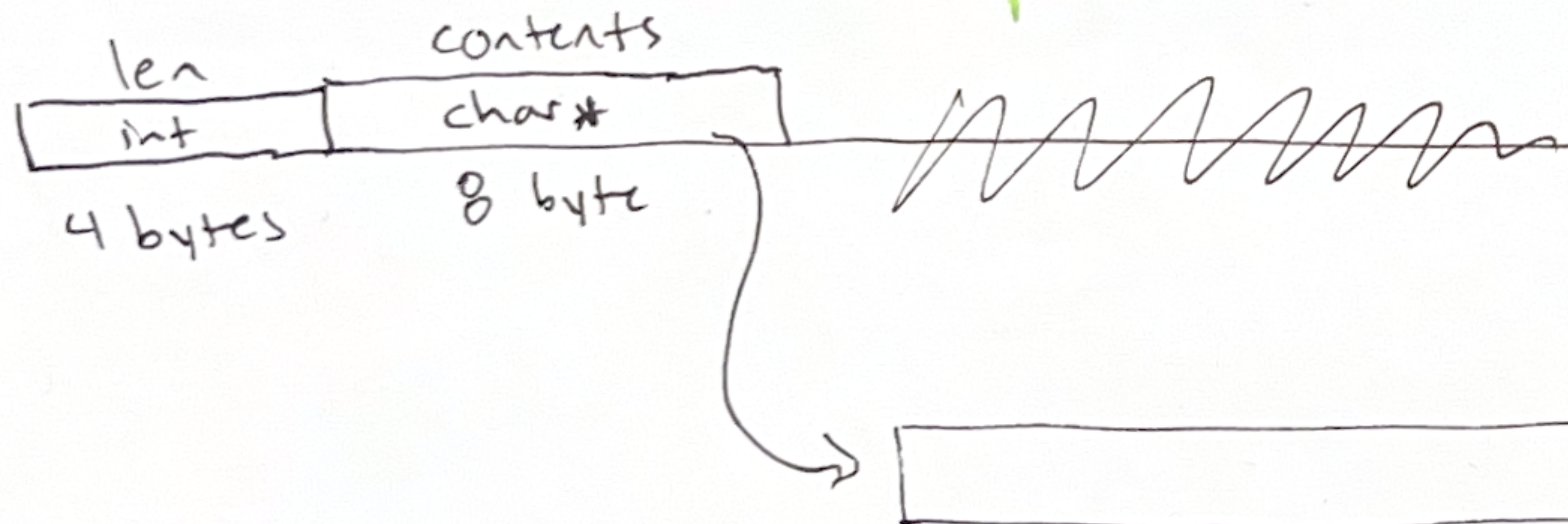


struct Point { int x; int y; } ^{field} sizeof(struct Point) = 8

struct String { int len; ^{array member} char contents[100]; } sizeof(struct String) = 104



struct String2 { int len; ^{pointer member} char* contents; } sizeof(struct String2) = 12



* maybe 16

$$\text{sizeof}(\text{ListOfDoubles}) = \frac{4 + 20 * \text{sizeof}(\text{double})}{8} = \frac{4 + 160}{8} = 164$$

```

1 #include <stdio.h>
2
3 #define MAX_LIST_SIZE 20
4
5 typedef struct ListOfDoubles {
6     int size;
7     double contents[MAX_LIST_SIZE];
8 } ListOfDoubles;
9
10 ListOfDoubles concat(ListOfDoubles d1, ListOfDoubles d2) {
11     ListOfDoubles result;
12     result.size = d1.size + d2.size;
13     for(int i = 0; i < d1.size; i += 1) {
14         result.contents[i] = d1.contents[i];
15     }
16     for(int i = 0; i < d2.size; i += 1) {
17         result.contents[i + d1.size] = d2.contents[i];
18     }
19     return result;
20 }
21
22 int main() {
23     ListOfDoubles lod = { 2, { 3.0, 4.0 } };
24     ListOfDoubles lod2 = { 3, { 6.0, 9.0, 10.0 } };
25     ListOfDoubles added = concat(lod, lod2);
26     for(int i = 0; i < added.size; i += 1) {
27         printf("%f ", added.contents[i]);
28     }
29     printf("\n");
30 }

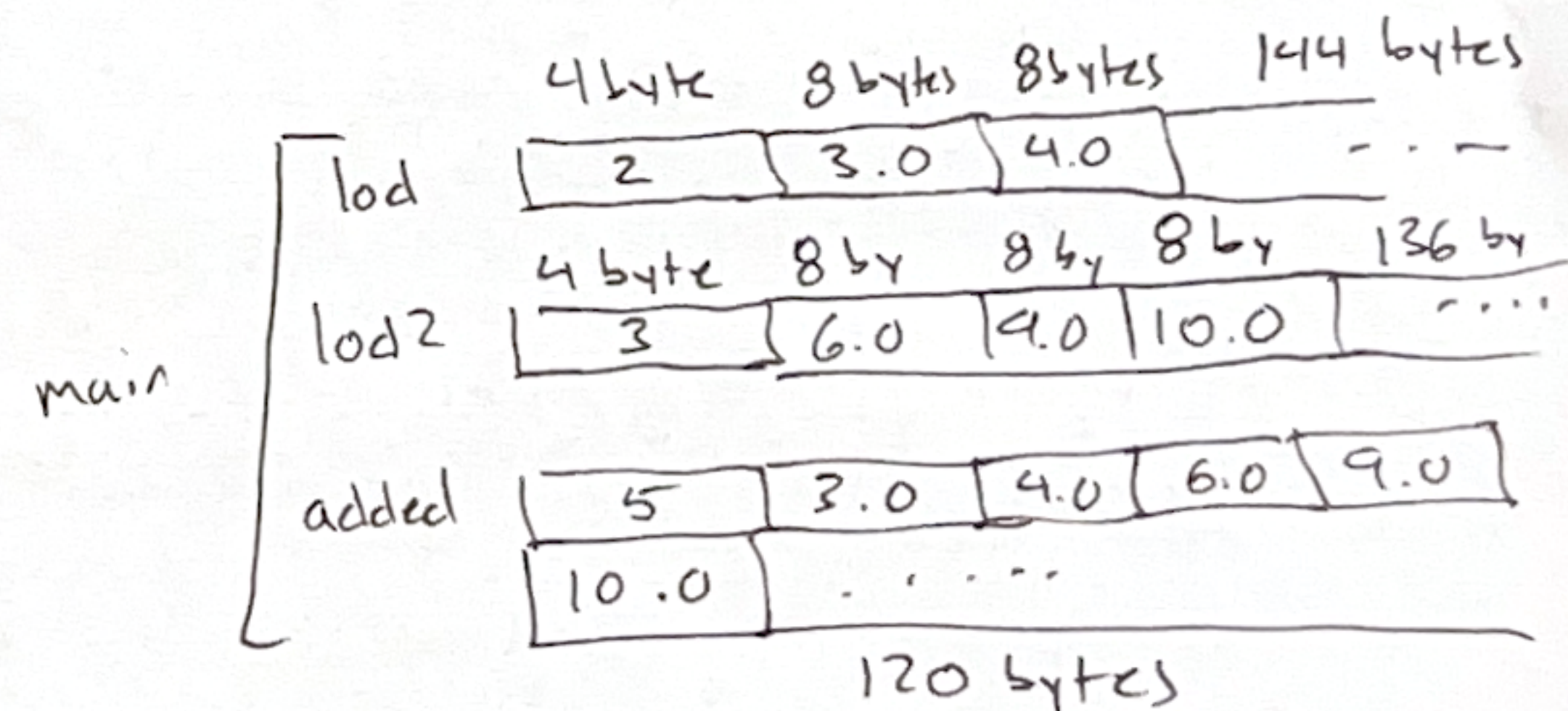
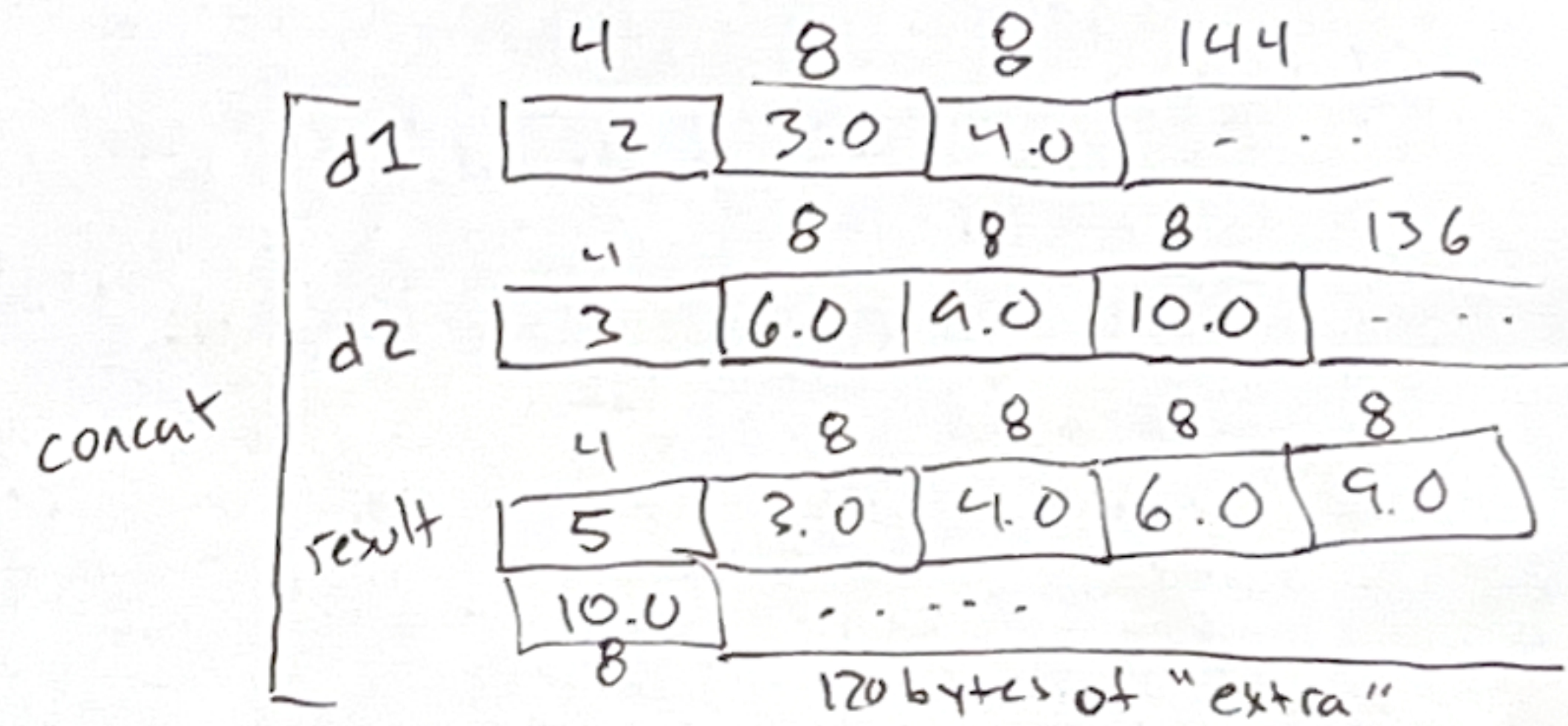
```

```

$ gcc -Wall list.c -o list
$ ./list
3.000000 4.000000 6.000000 9.000000 10.000000

```

copy the contents
of result to
the caller
so, added gets
a complete copy



Array members are great for fixed-max-size
list/array calculations

Bad (inefficient, wastespace) for large arrays
(large ~ > 10000 bytes)

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct ListOfDoubles {
5     int size;
6     double* contents;
7 } ListOfDoubles;
8
9 ListOfDoubles concat(ListOfDoubles d1, ListOfDoubles d2) {
10     ListOfDoubles result;
11     result.size = d1.size + d2.size;
12
13     result.contents = malloc(result.size * sizeof(double));
14
15     for(int i = 0; i < d1.size; i += 1) {
16         result.contents[i] = d1.contents[i];
17     }
18     for(int i = 0; i < d2.size; i += 1) {
19         result.contents[i + d1.size] = d2.contents[i];
20     }
21     return result;
22 }
23
24 int main() {
25     double init1[] = { 3.0, 4.0 };
26     ListOfDoubles lod = { 2, init1 };
27     double init2[] = { 6.0, 9.0, 10.0 };
28     ListOfDoubles lod2 = { 3, init2 };
29     ListOfDoubles added = concat(lod, lod2);
30     for(int i = 0; i < added.size; i += 1) {
31         printf("%f ", added.contents[i]);
32     }
33     printf("\n");
34 }

```

(in this example,
ask malloc for
40 bytes)

Need to create array
separately and pass in

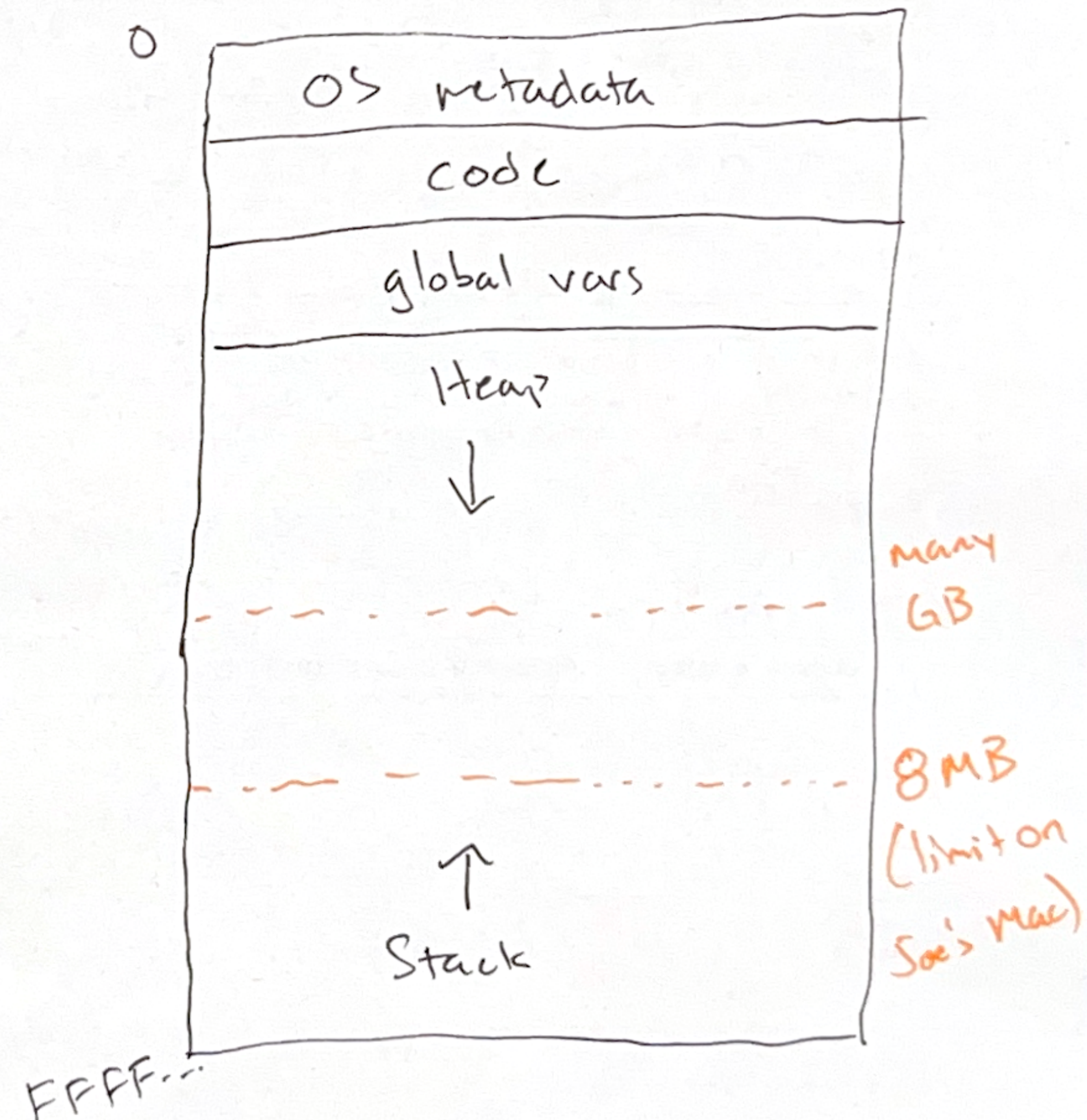
```

$ gcc -Wall list_heap.c -o list_heap
$ ./list_heap
3.000000 4.000000 6.000000 9.000000 10.000000

```

malloc(int size)
size_t

Returns a pointer to newly allocated
heap memory of size bytes
address space



```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct ListOfDoubles {
5     int size;
6     double* contents;
7 } ListOfDoubles;
8
9 ListOfDoubles concat(ListOfDoubles d1, ListOfDoubles d2) {
10     ListOfDoubles result;
11     result.size = d1.size + d2.size;
12
13     result.contents = malloc(result.size * sizeof(double));
14
15     for(int i = 0; i < d1.size; i += 1) {
16         result.contents[i] = d1.contents[i];
17     }
18     for(int i = 0; i < d2.size; i += 1) {
19         result.contents[i + d1.size] = d2.contents[i];
20     }
21     return result;
22 }
23
24 int main() {
25     double init1[] = { 3.0, 4.0 };
26     ListOfDoubles lod = { 2, init1 };
27     double init2[] = { 6.0, 9.0, 10.0 };
28     ListOfDoubles lod2 = { 3, init2 };
29     ListOfDoubles added = concat(lod, lod2);
30     for(int i = 0; i < added.size; i += 1) {
31         printf("%f ", added.contents[i]);
32     }
33     printf("\n");
34 }

```

```

$ gcc -Wall list_heap.c -o list_heap
$ ./list_heap
3.000000 4.000000 6.000000 9.000000 10.000000

```

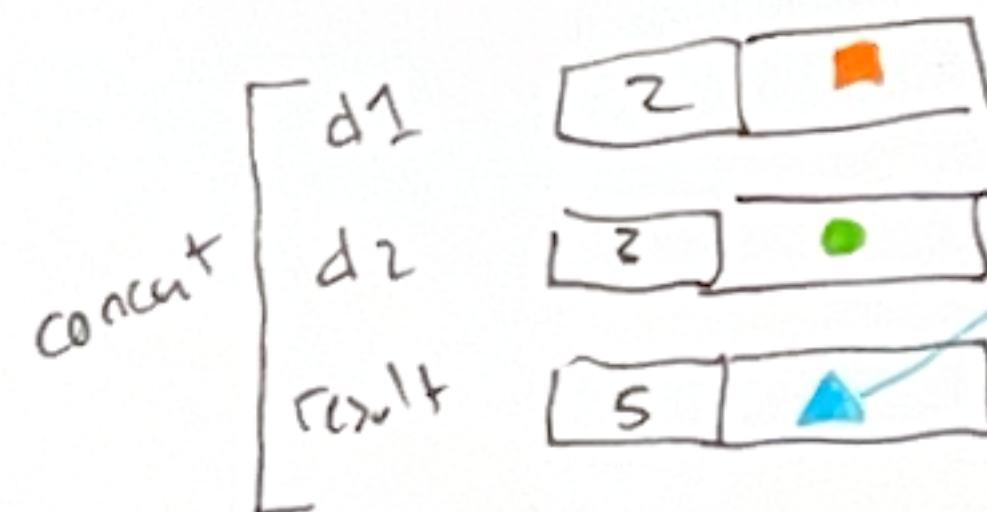
(in this example,
ask malloc for
40 bytes)

Need to create array
separately and pass in

malloc(int size)
size_t

Returns a pointer to newly allocated
heap memory of size bytes
address space

Heap memory
does not go away
when functions
return! Heap



Stack