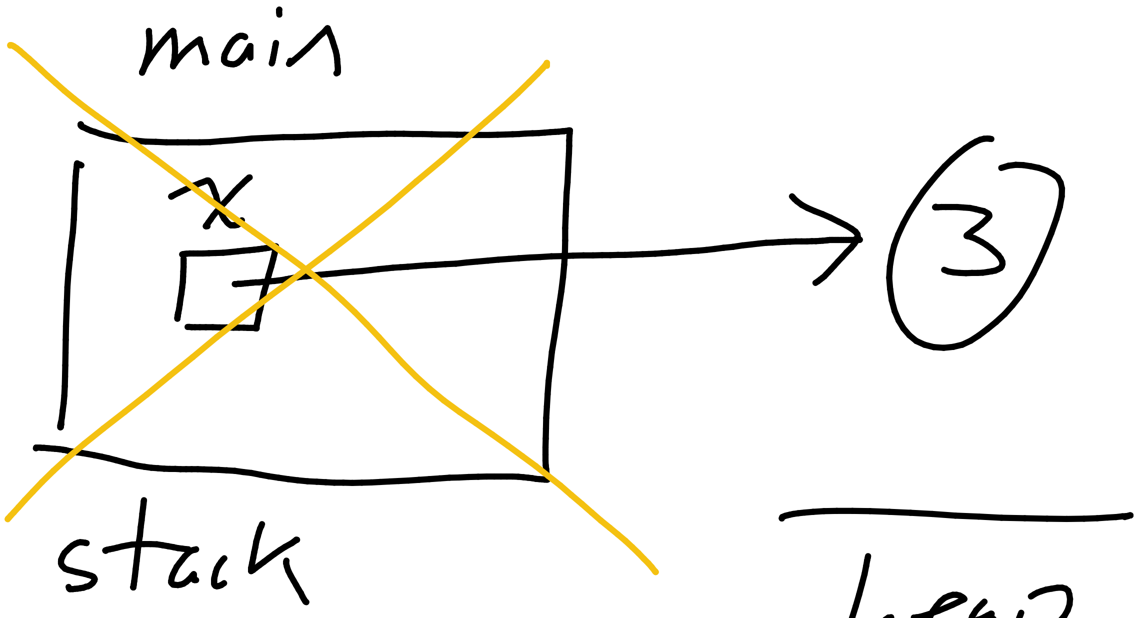
Today: lots more C
- pointers
- stack vs heap

Stack vs heap memory

Python

```
def main():
    x = 3

main()
```



local variables, global vars, params

objects go

stack memory is allocated and deallocated automatically based on structure of your code
- e.g. when main is over, x no longer exists, because all memory in main is gone

Heap memory (in Python) sticks around as long as something still refers to it

---

Why both capabilities?

stack memory allows for local variables and recursion

heap memory allows flexibility

- pick memory sizes yourself sometimes
- allows linked lists, for example (or any object that sticks around after the function that makes it is gone)

C examples

$\underbrace{\text{int } *}x;$ declares a variable named x, of type int* (pointer to an int)

(C allows int* x; but <u>bad</u> <u>idea</u>

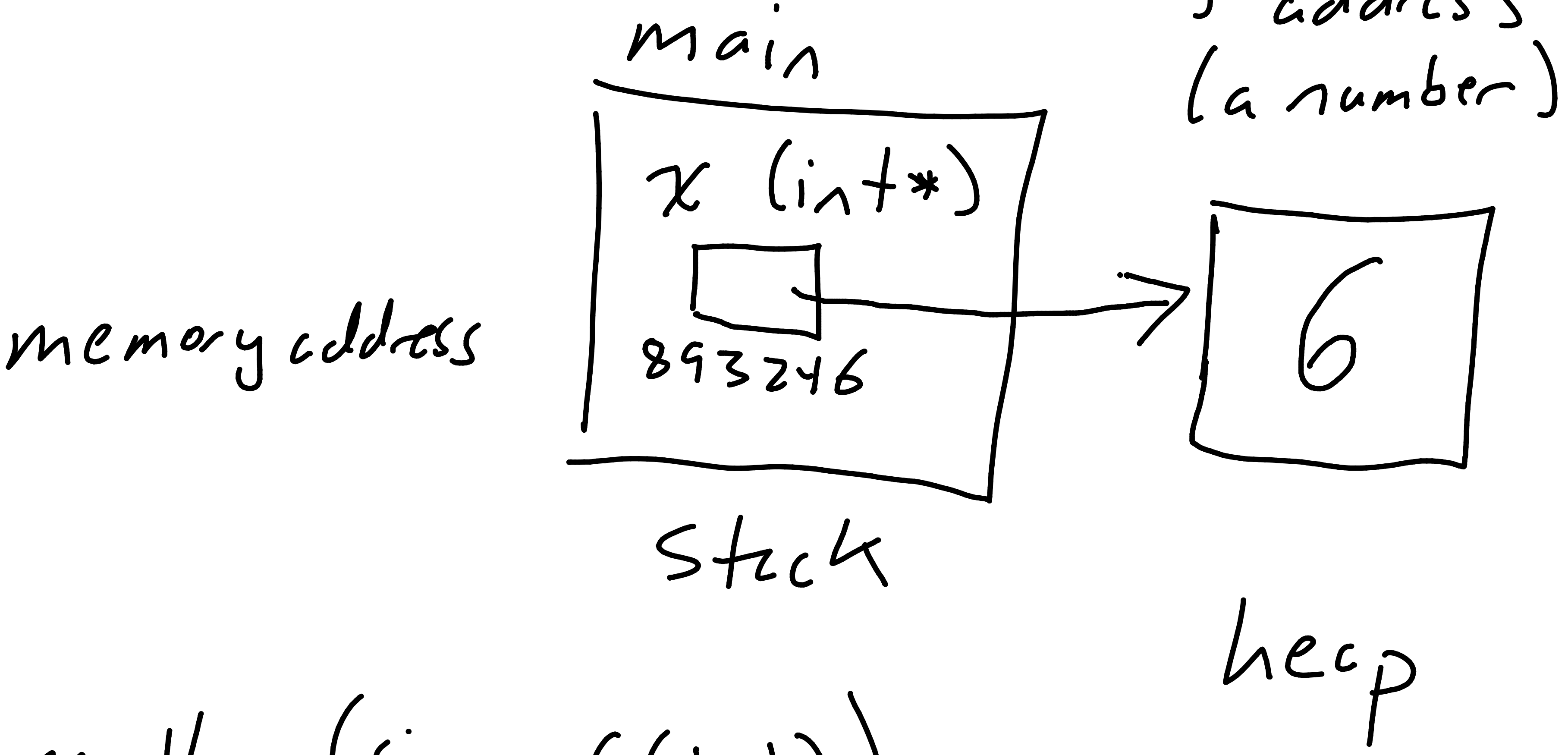
int* x, y, z; C does

C declares x of type int*

y } of type int
z)

main{

int *x; // a pointer in C is a memory address (a number)

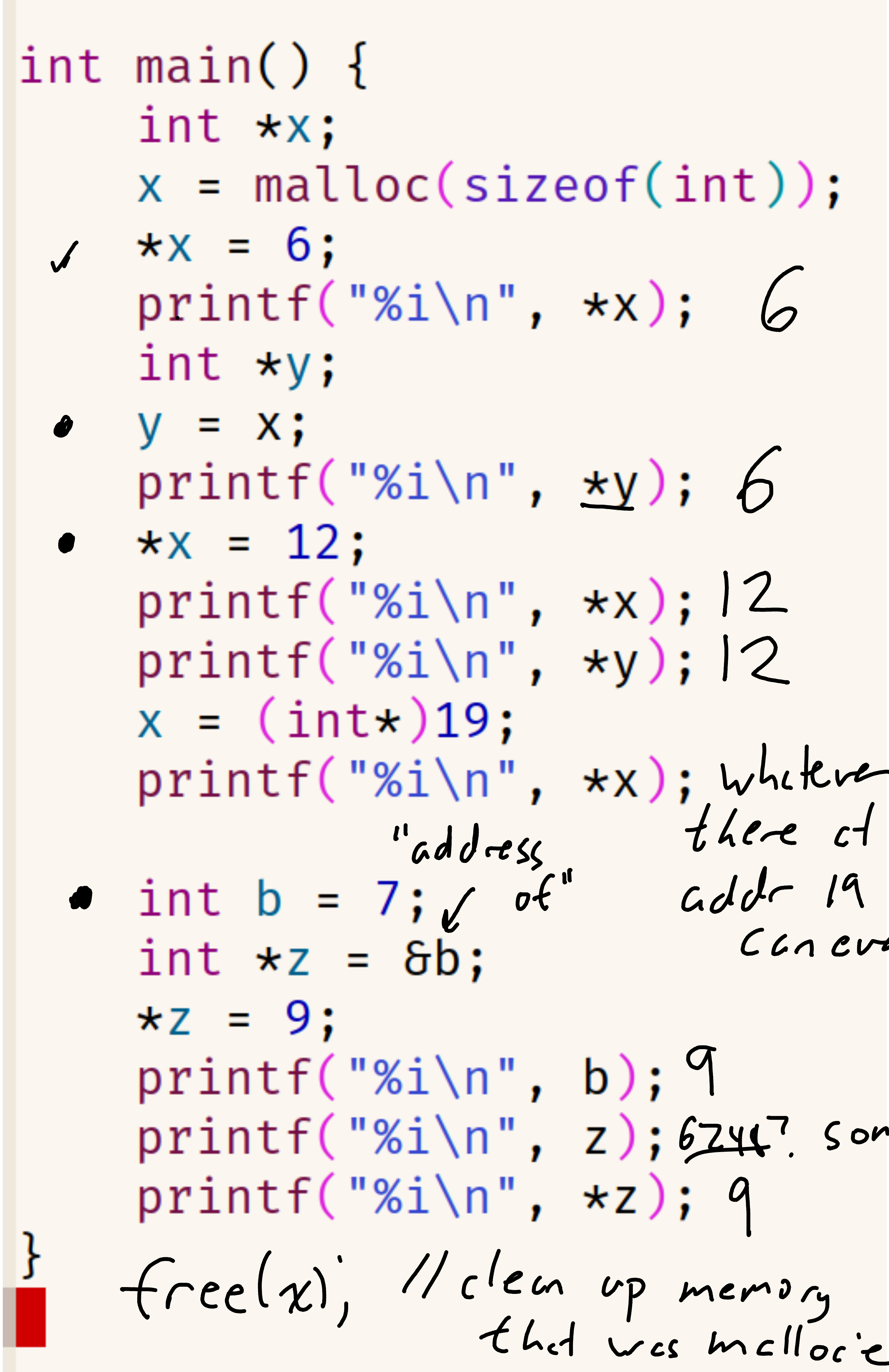


x = malloc(sizeof(int))

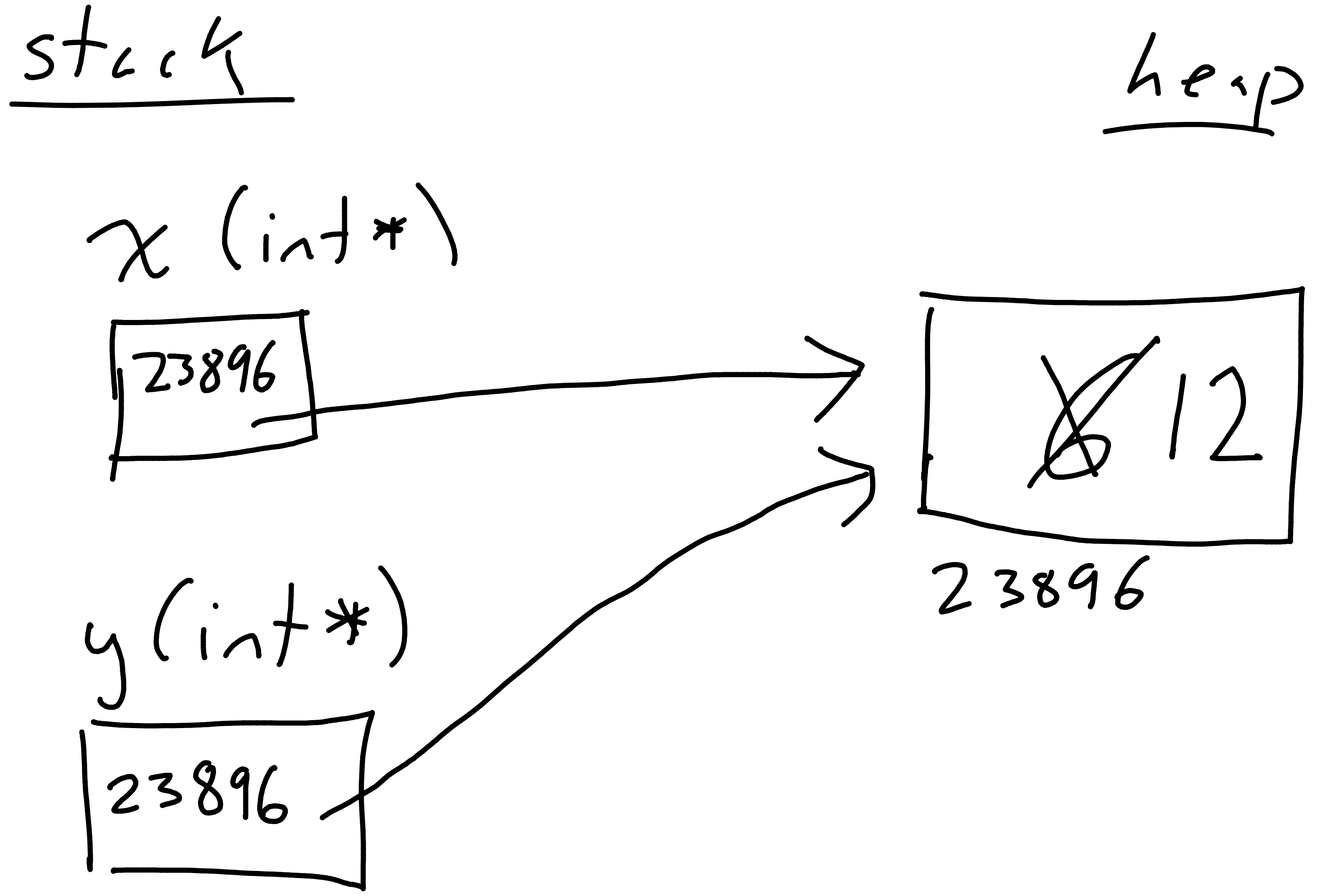
allocated in heap bytes as indicates

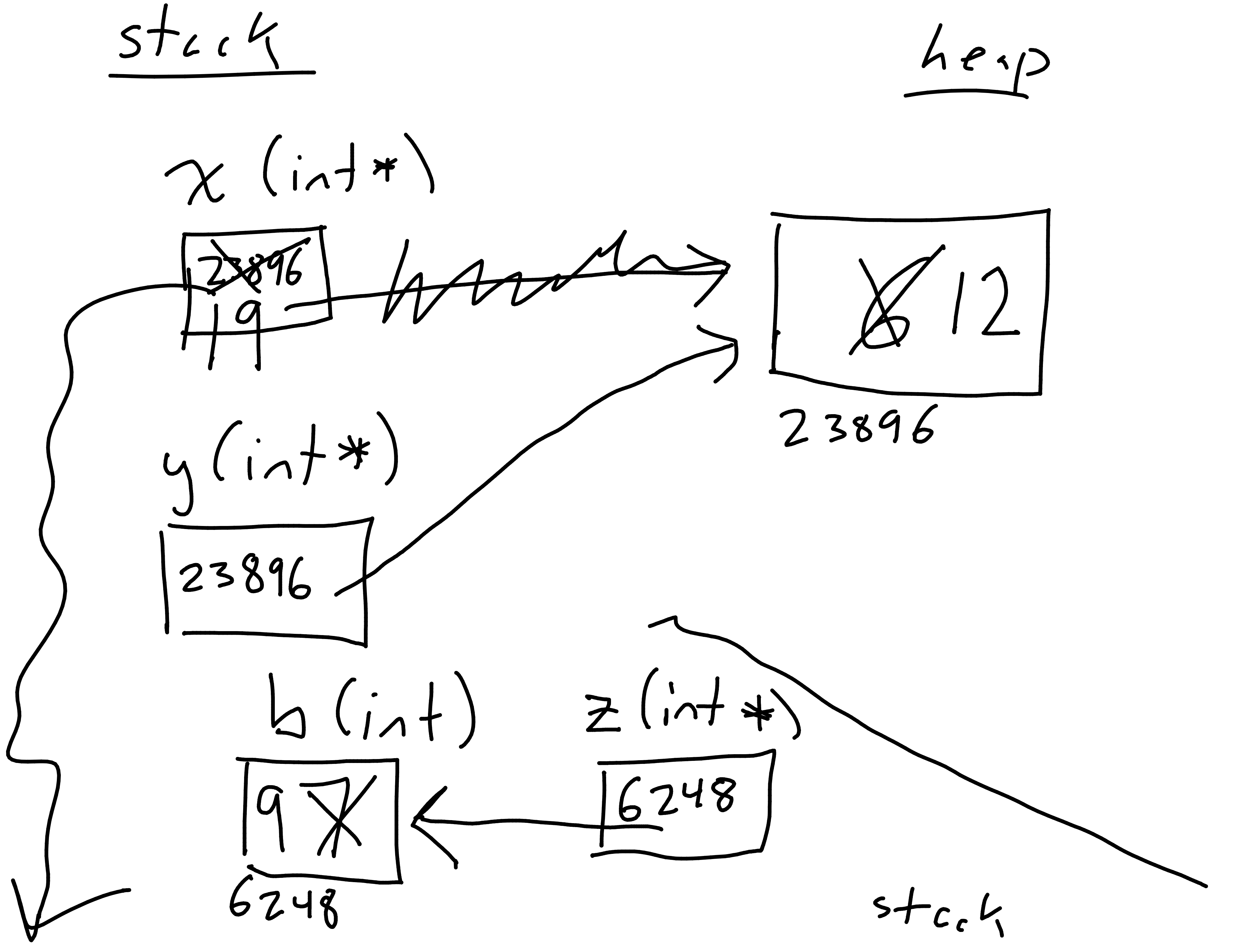
*x = 6

↑ "water slide operator"

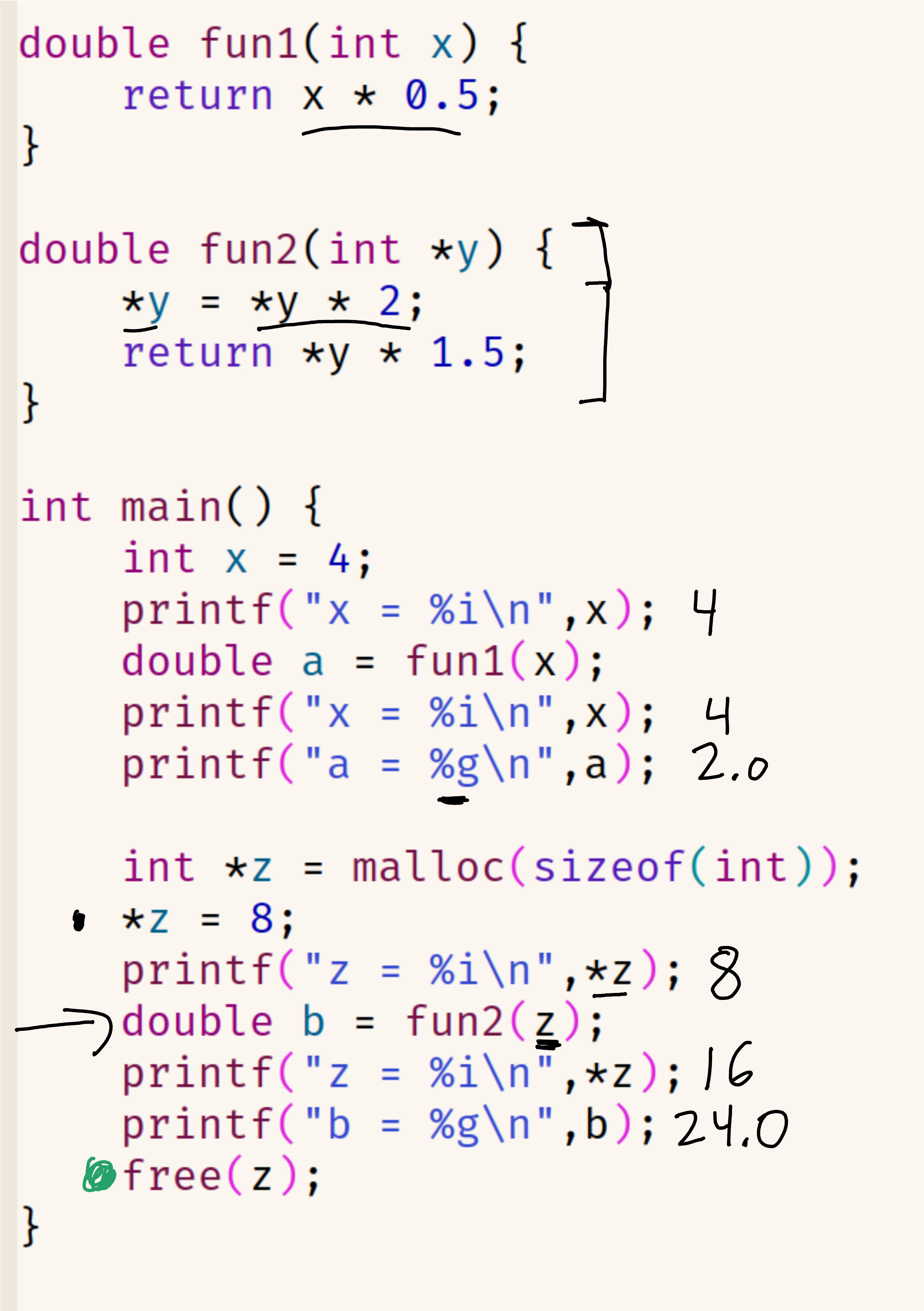


```
int main() {
      int *x;
      x = malloc(sizeof(int));
  ✓   *x = 6;
      printf("%i\n", *x);   6
      int *y;
  ●   y = x;
      printf("%i\n", *y);  6
  ●   *x = 12;
      printf("%i\n", *x); 12
      printf("%i\n", *y); 12
      x = (int*)19;
      printf("%i\n", *x); whatever is out
                          there at mem
                "address  addr 19 if I
  ●   int b = 7; ✓ of"      can even get to it
      int *z = &b;
      *z = 9;
      printf("%i\n", b); 9
      printf("%i\n", z); 6244?. something
      printf("%i\n", *z); 9
}
    free(x); // clean up memory
              that was malloc'ed
```

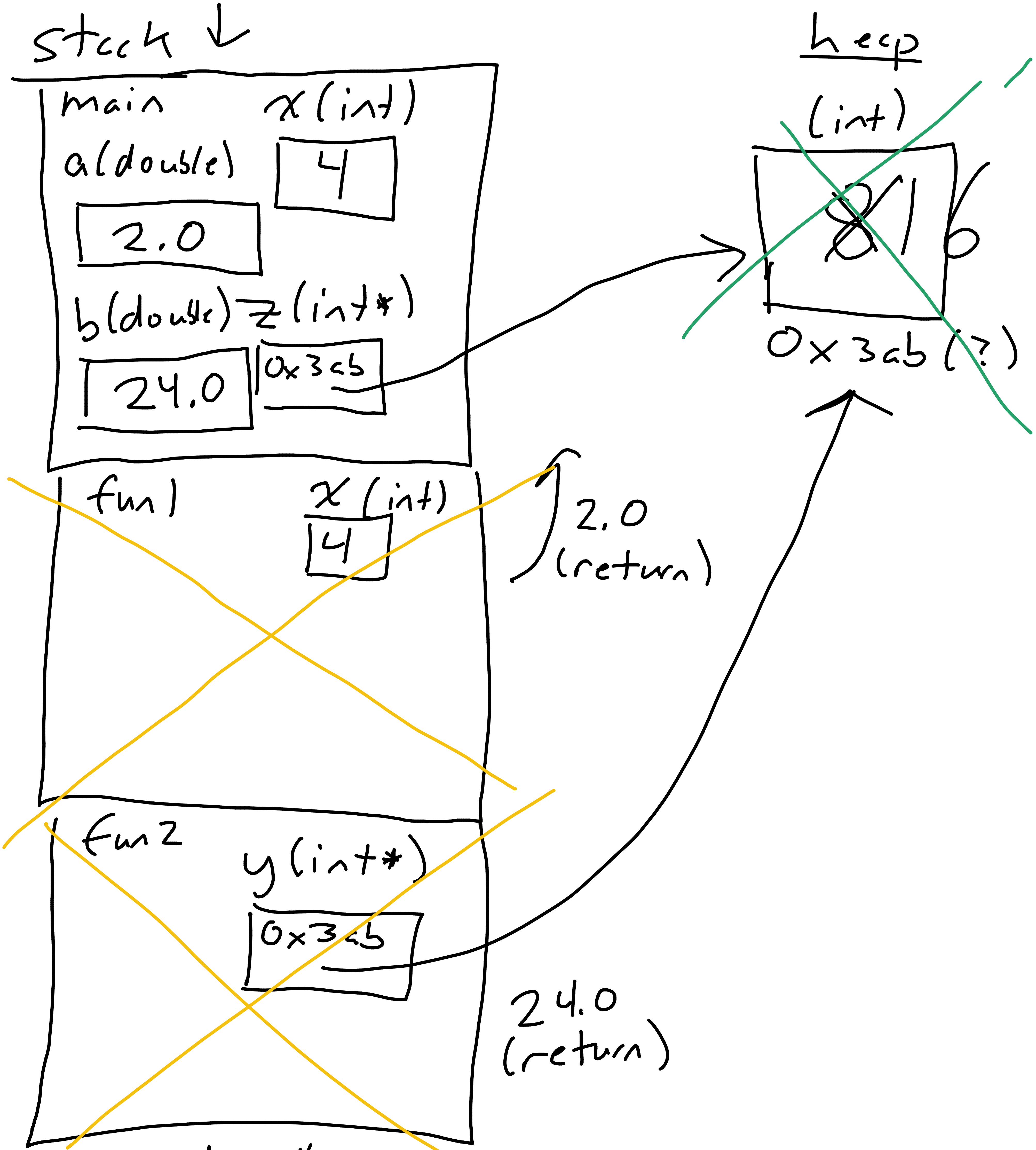

stack
heap
x (int*)
23896
y (int*)
23896
~~6~~ 12
23896


stack
heap
x (int*)
~~23896~~
9
~~6~~ 12
23896
y (int*)
23896
b (int)
9 ~~7~~
6248
z (int*)
6248
stack

stack vs heap.c program



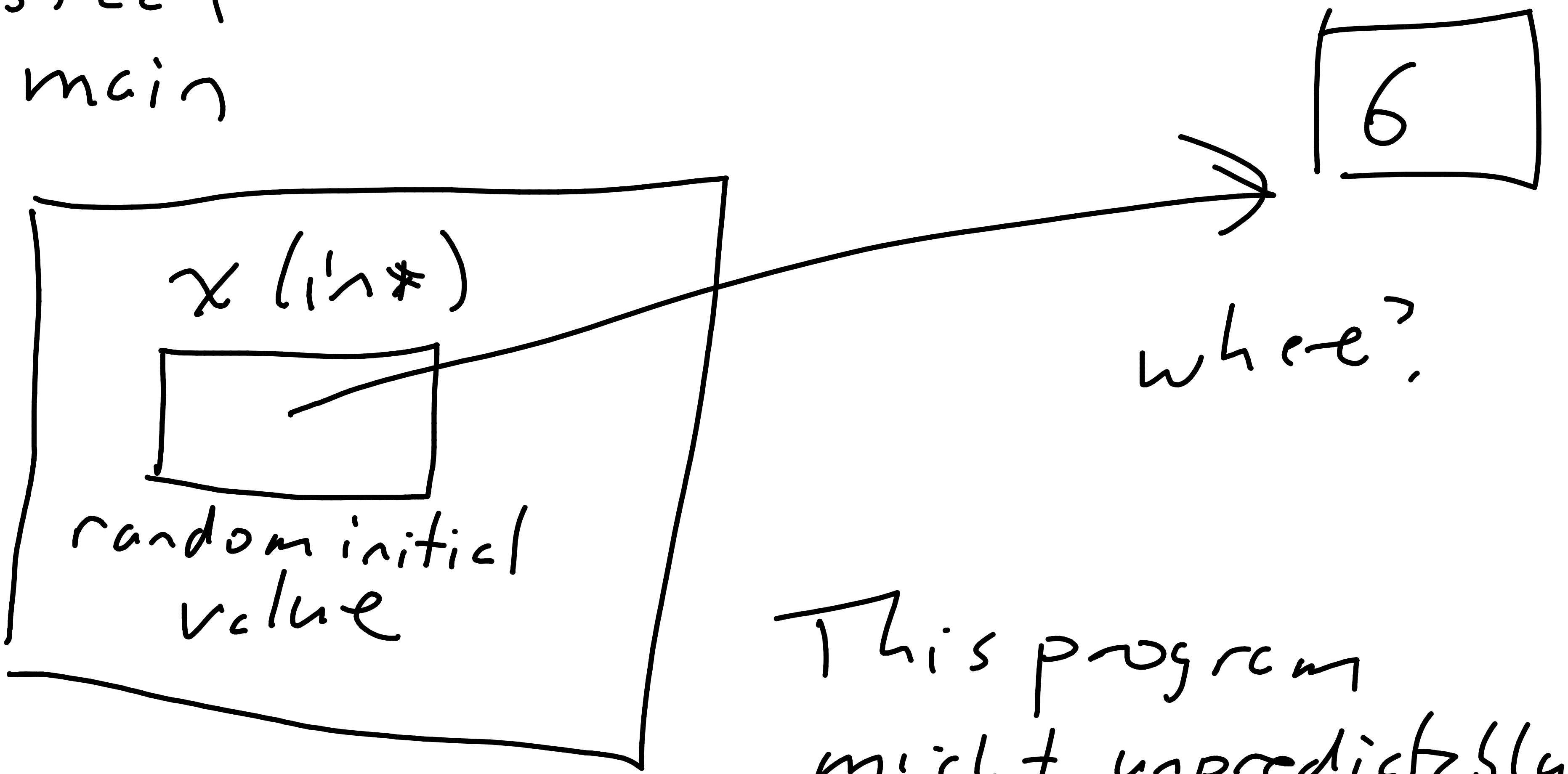
```
double fun1(int x) {
    return x * 0.5;
}

double fun2(int *y) {
    *y = *y * 2;
    return *y * 1.5;
}

int main() {
    int x = 4;
    printf("x = %i\n",x); 4
    double a = fun1(x);
    printf("x = %i\n",x); 4
    printf("a = %g\n",a); 2.0

    int *z = malloc(sizeof(int));
    *z = 8;
    printf("z = %i\n",*z); 8
    double b = fun2(z);
    printf("z = %i\n",*z); 16
    printf("b = %g\n",b); 24.0
    free(z);
}
```



(should really superimpose where fun1 was)

badstuff

stack
main



This program might unpredictably fail