# Recitation 09

Quiz 2 and Dynamic Memory

# Today's agenda

- We will discuss in recitation
  - Quiz 2
    - Grades are on NYU Classes
    - I have them if you want to look at them during office hours
      - You can't keep them
  - Dynamic Memory
    - I won't go into much depth today because I don't expect to have time
- For homework Tonight
  - R09
    - Fix a buggy program that tries to do dynamic memory but does it wrongly

# Dynamic Memory

For when static memory isn’t enough

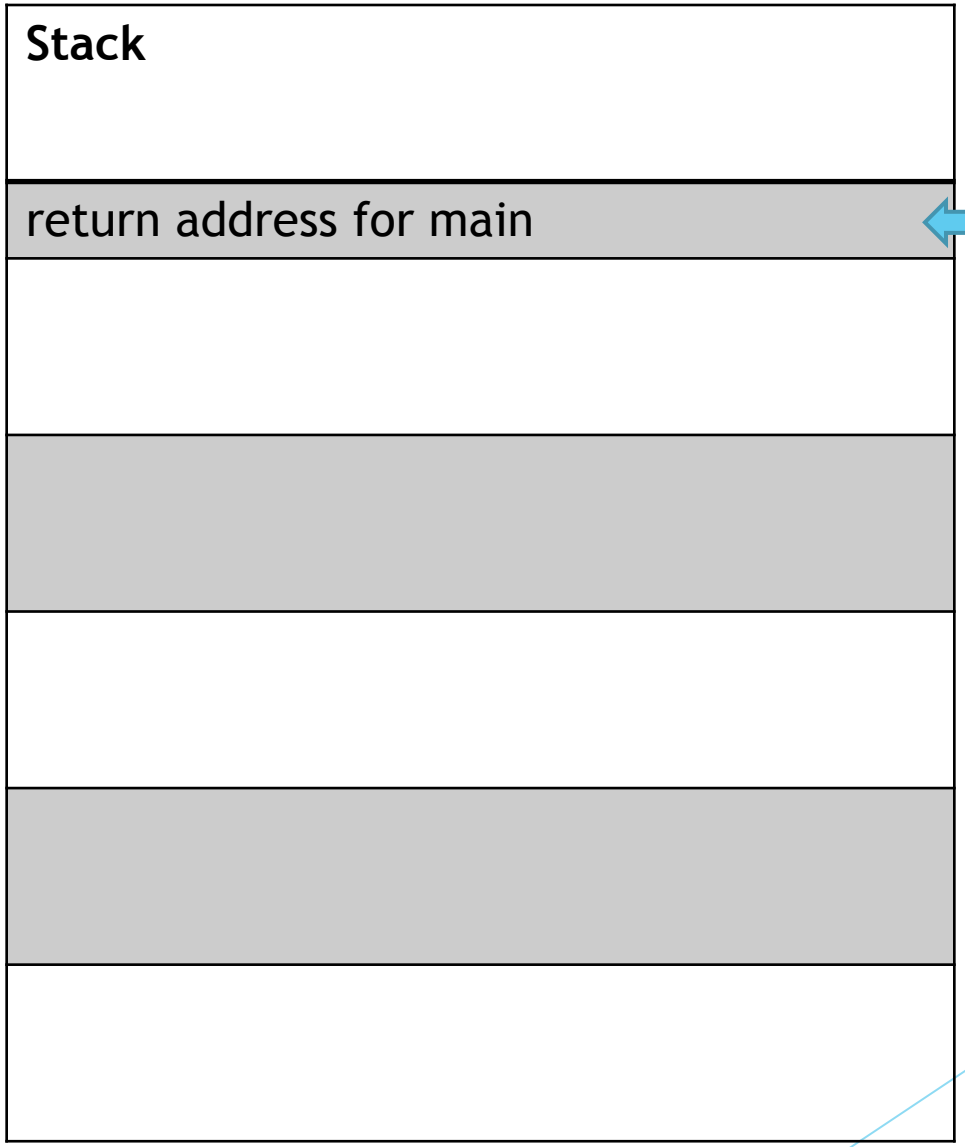
# Why Dynamic Memory?

- You don't always know how much memory you will need for your program
- What if you want to write a program that finds the average value in a column?
- If you did write such a program, how do you handle a user giving you a really big file, bigger than you expected?
- Even if you made sure you specified a really big global variable as a static buffer, people might still give you bigger files
  - And why go through that trouble anyway instead of just having dynamic memory?

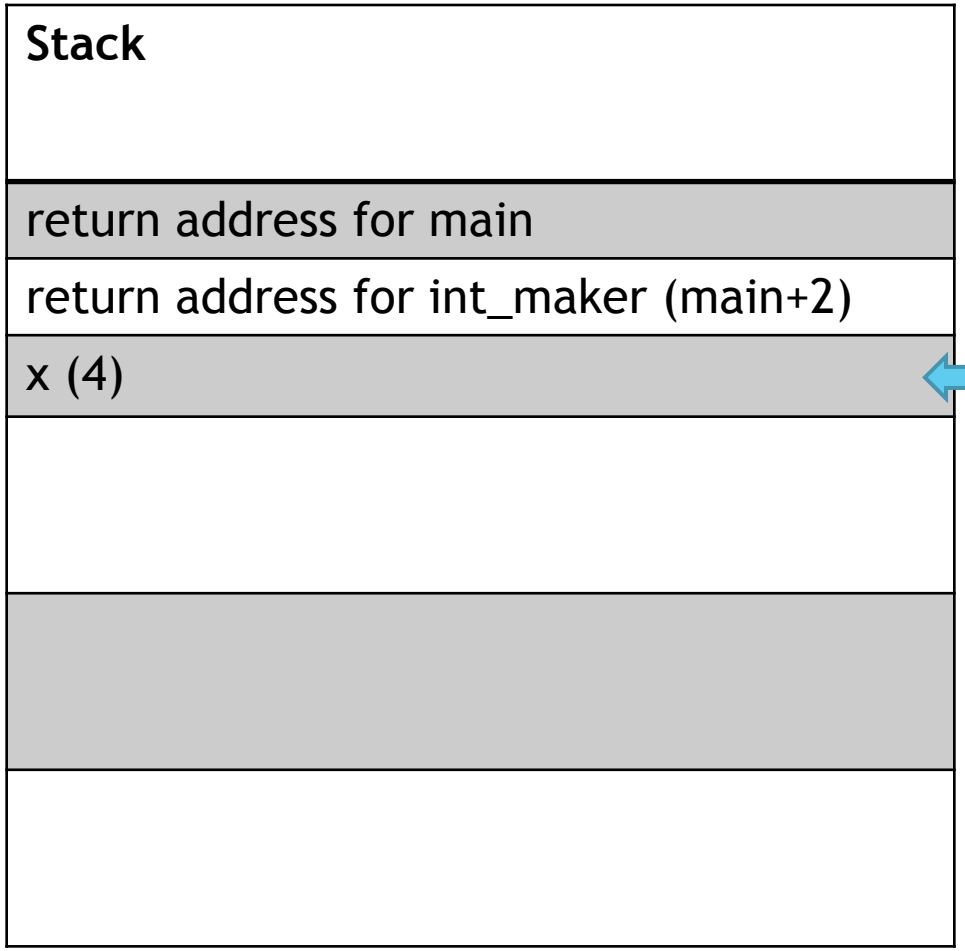
# Dynamic memory and the stack

- Does the stack give us dynamic memory?
  - In a sense, yes
  - However, it isn't always suitable, because the memory gets reused after we return from a function call
  - By default the stack is also only a few megabytes in size

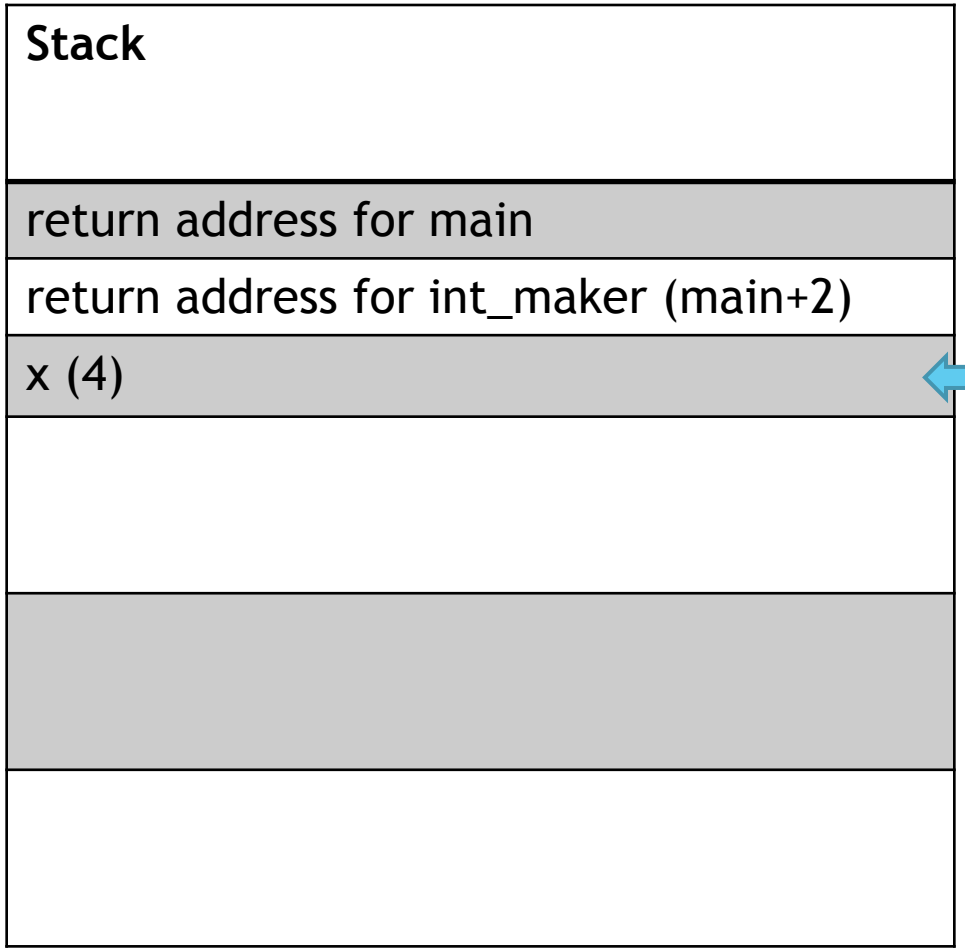
```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



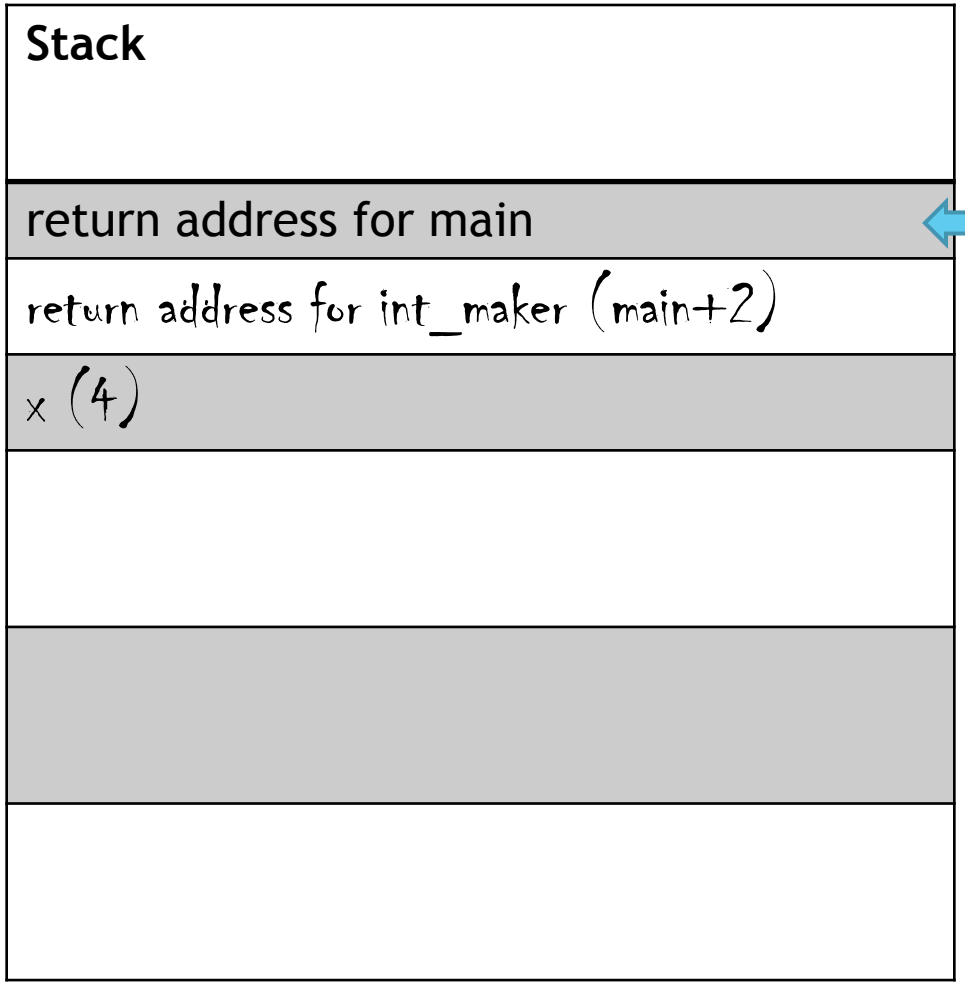
```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



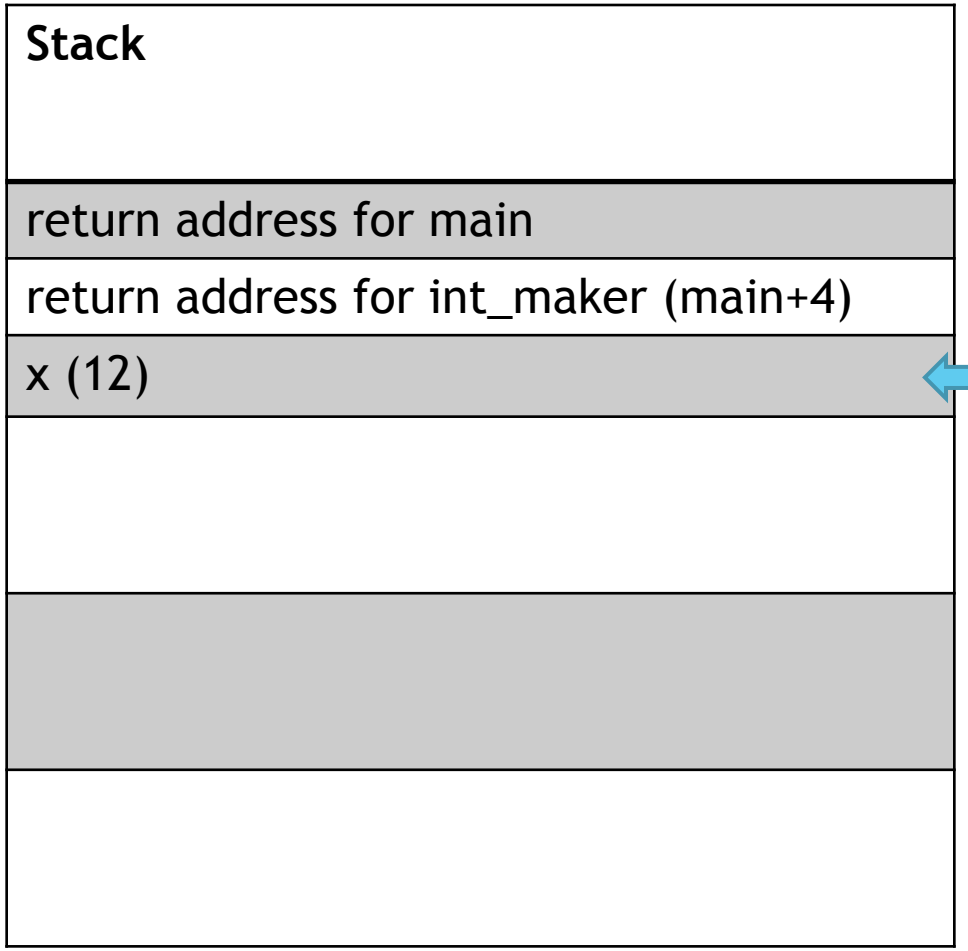

```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



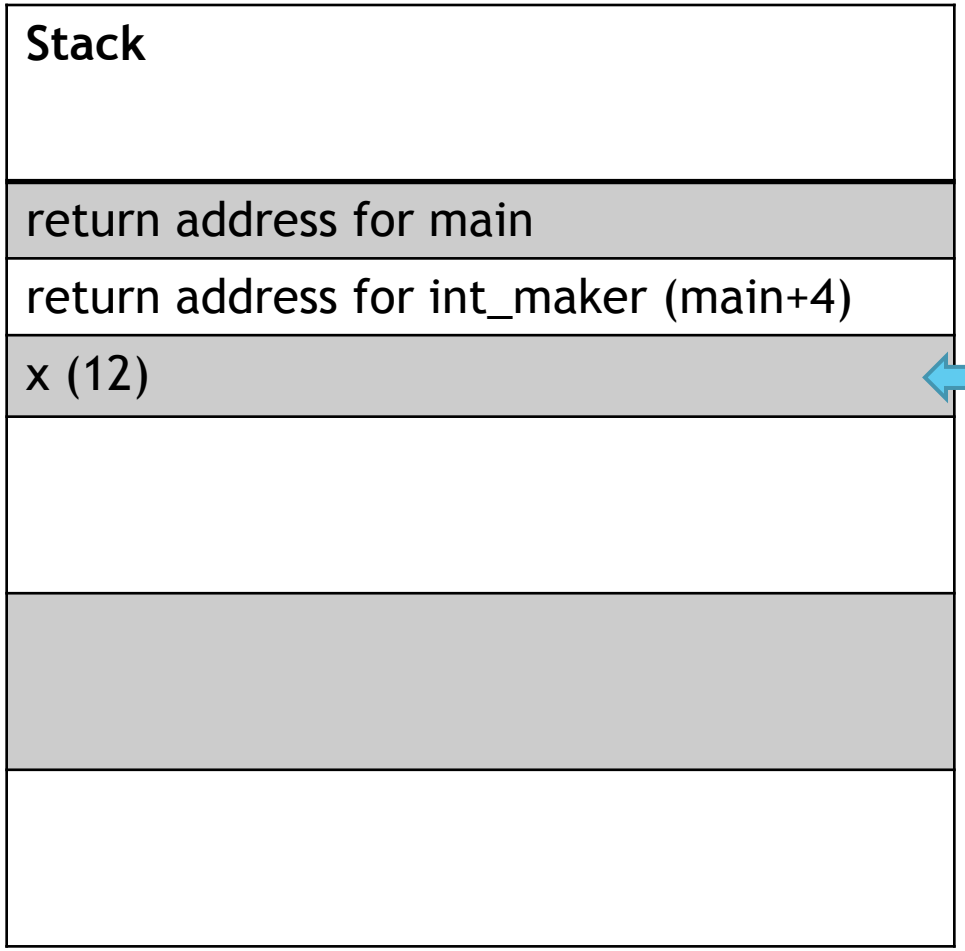
```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



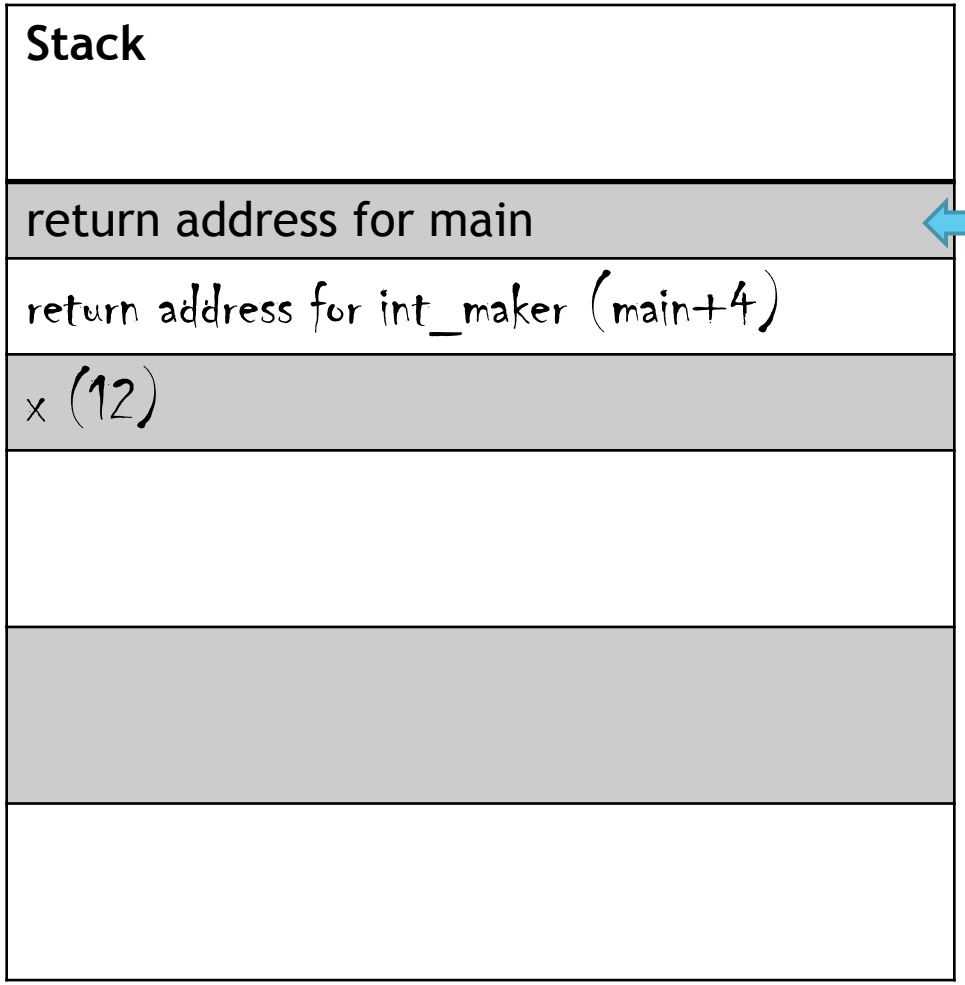

```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



```
int* int_maker(int i) {
        int x = i + 2;
        return &x;
}
int main() {
        int *p = int_maker(2);
        int *m = int_maker(10);
        printf("%d %d\n", *p, *m);
}
```



# Dynamic memory and the stack

- Where do p and m point to?
- What is at that location?
- Is that location still valid?

# Dynamic memory and the stack

- Where do p and m point to?
  - Undefined behavior!
- What is at that location?
  - Undefined behavior!
- Is that location still valid?
  - NO!
  - Well, it could be, but if you try to do this, you will write buggy code!
- So instead we use the heap

# Dynamic memory on the heap

- We can use the sbrk syscall to ask the operating system to give us more heap space
- We can also use it to give back to the operating system
- However, in the real world, programmers don't often do this themselves
  - Why?
- Instead, we usually use a library that handles things for us
  - Enter malloc and free

# Malloc and Free

- Malloc allocates us a contiguous section of memory
  - It returns a void*, which is just “pointer to anything”
  - So you cast the result of malloc to what you want, e.g. int*
  - Malloc can return NULL if there was an error
- Free gives the memory back to the allocator
  - DO NOT call free twice on the same section of memory
    - This is undefined behavior
  - What you call free on must be the result of malloc
    - (or calloc or realloc, but I won’t discuss those)

# Allocators

- Allocators can't move data around
  - Why?
- How do you track what parts of the heap are freed or malloced? How do you track their sizes?
  - The trick is to store metadata along with the data in the heap to create a linked list
  - You store the status of the chunk (free or allocated), and the size of the data (which effectively points to the next chunk)
- When someone asks for memory, what do you give them?
  - There are a number of different strategies