

This is CS50.

cs50.brianyu.me

Week 4

- Hexadecimal
- Pointers
- Dynamic Memory
- Memory Layout
- File I/O

What questions do you have?

Questions

Today

Pointers

File I/O

Dynamic Memory

PART ONE

Pointers

Pointers

Types

int

Integer

char

Character

Types

int * Address of an Integer

char * Address of a Character

Pointers

- **type *** is a pointer that stores the address of a **type**
- ***x** takes a pointer **x** and goes to that address
- **&x** takes **x** and gets its address


```
int a = 28;
```

```
int a = 28;
```

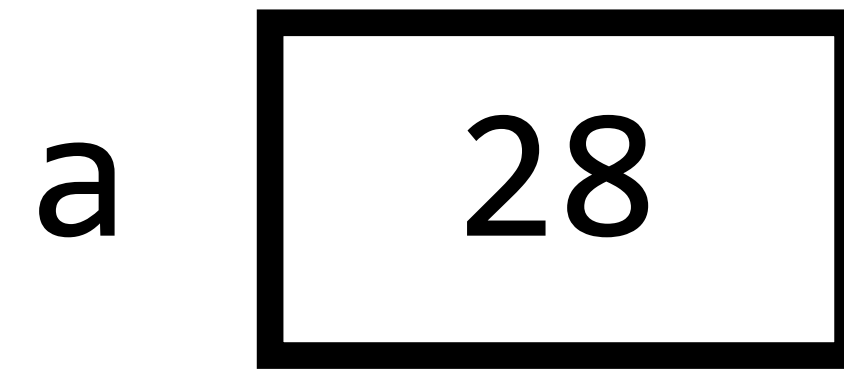


```
int a = 28;
```

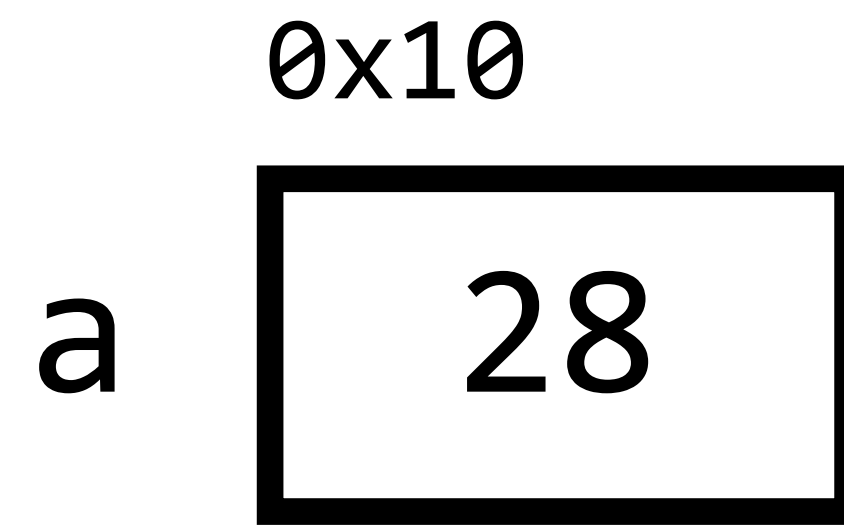
a



```
int a = 28;
```

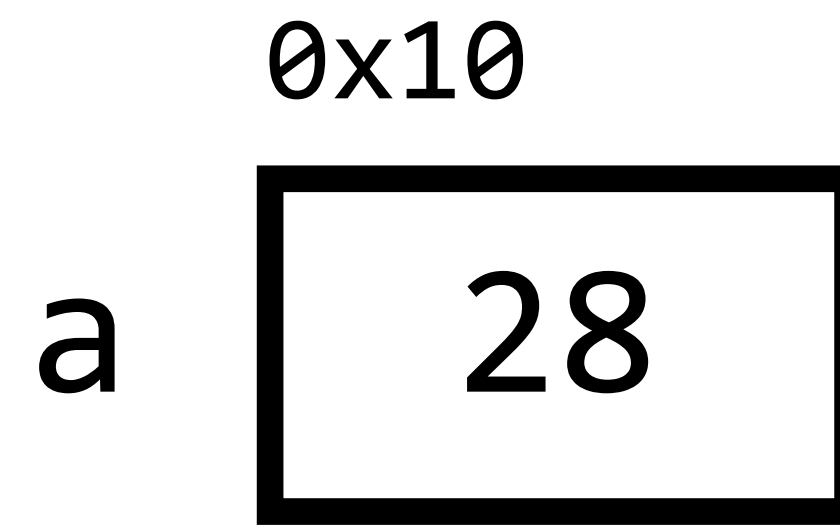


```
int a = 28;
```



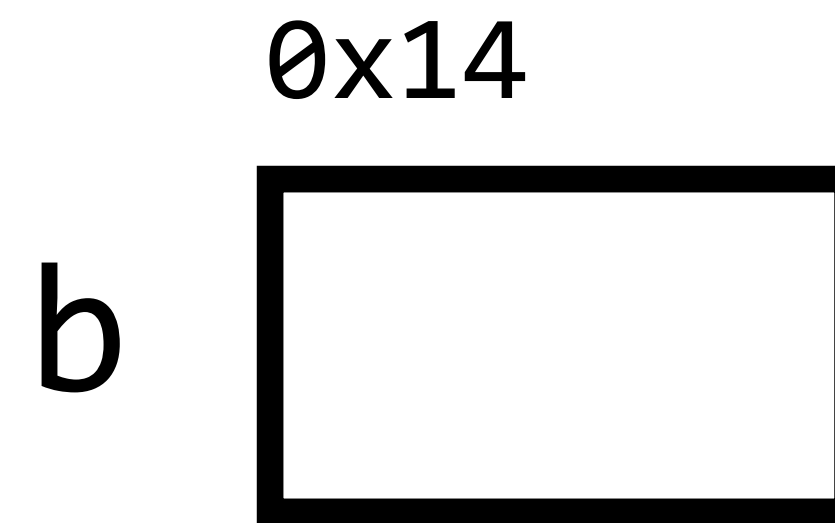
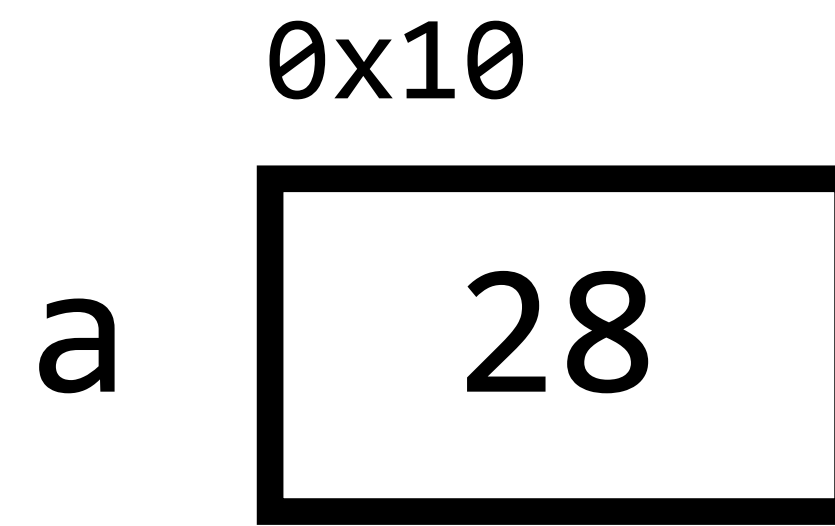
```
int a = 28;
```

```
int b = 50;
```



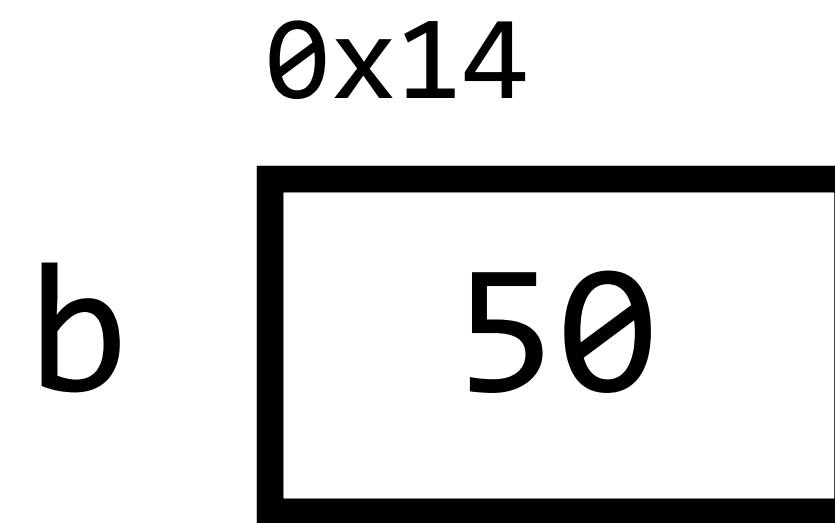
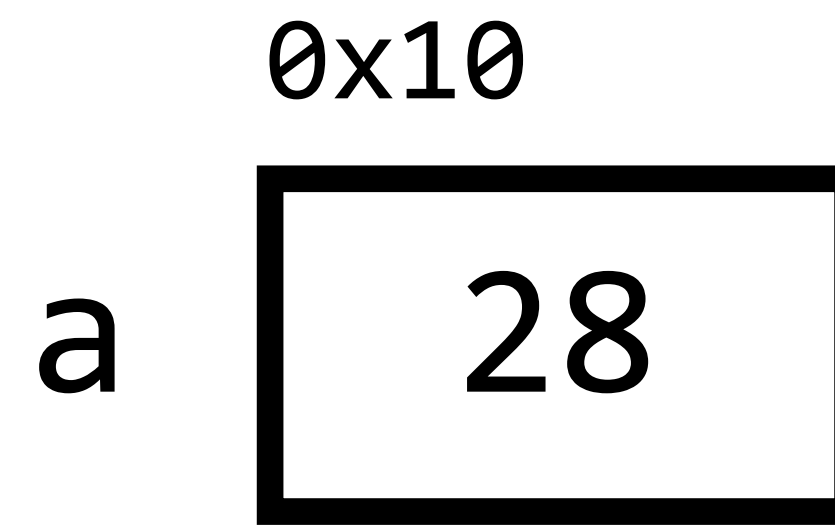
```
int a = 28;
```

```
int b = 50;
```



```
int a = 28;
```

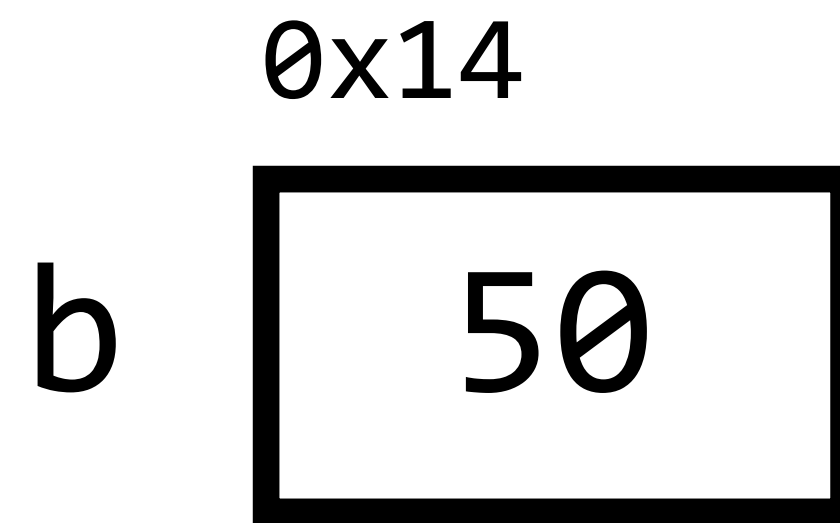
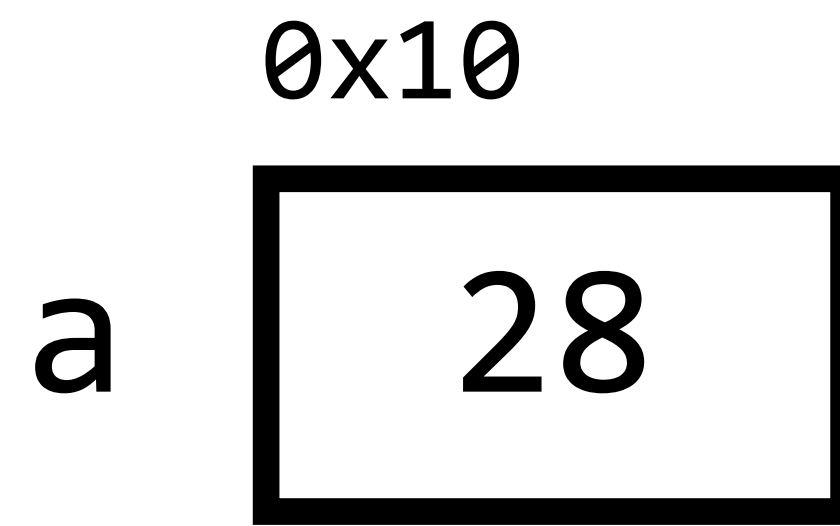
```
int b = 50;
```




```
int a = 28;
```

```
int b = 50;
```

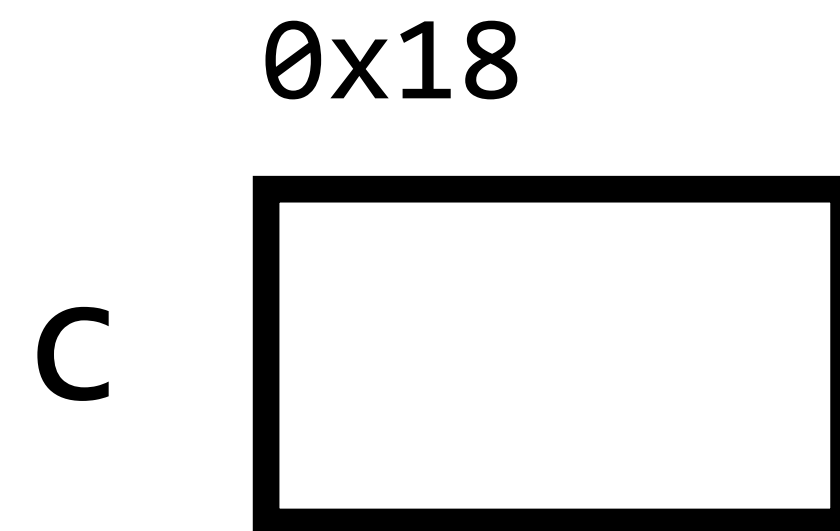
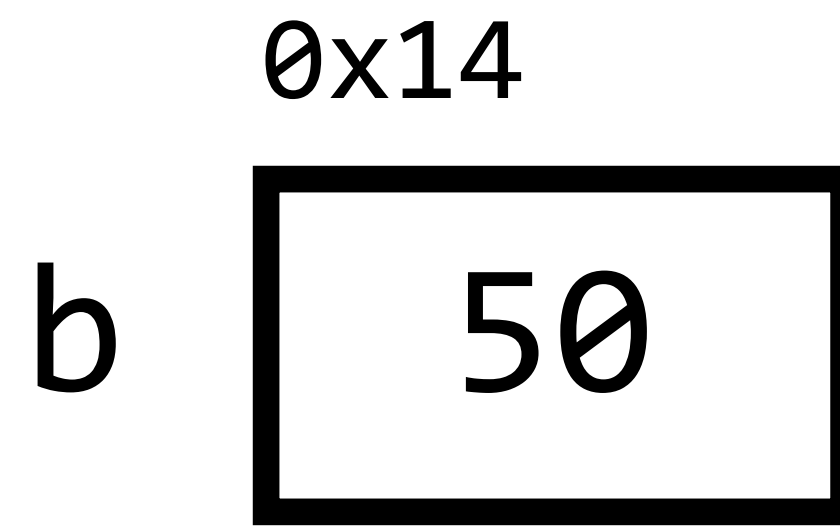
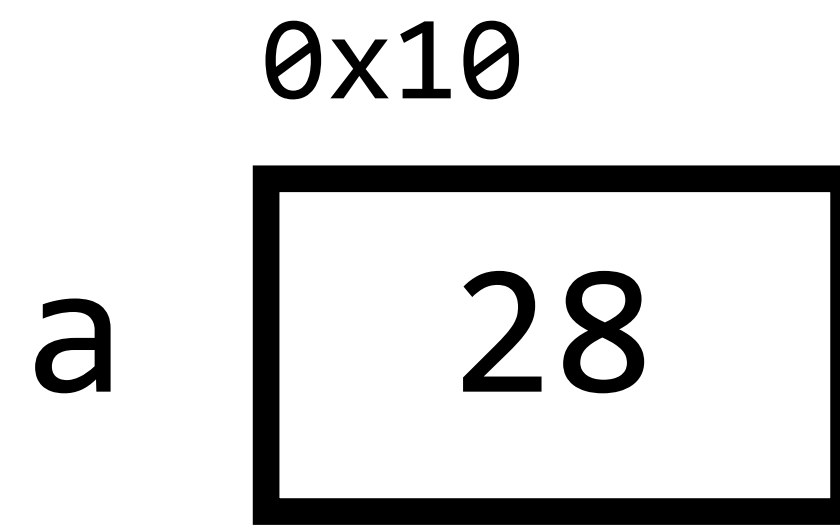
```
int *c = &a;
```



```
int a = 28;
```

```
int b = 50;
```

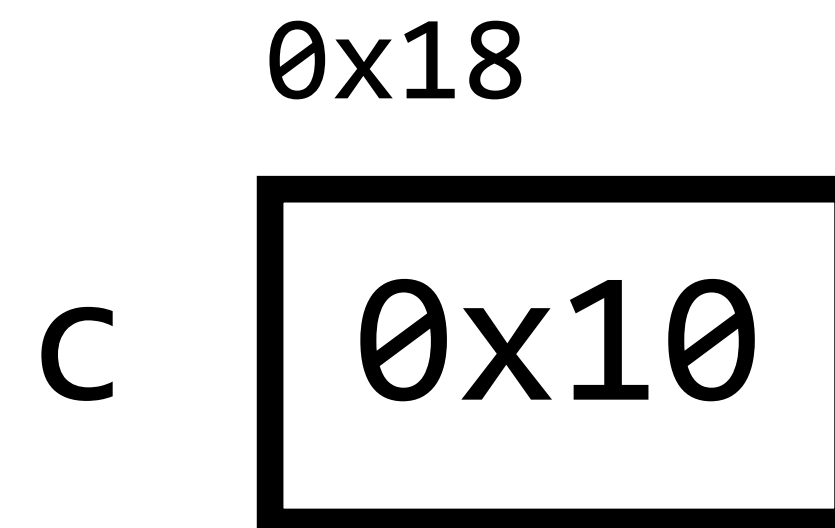
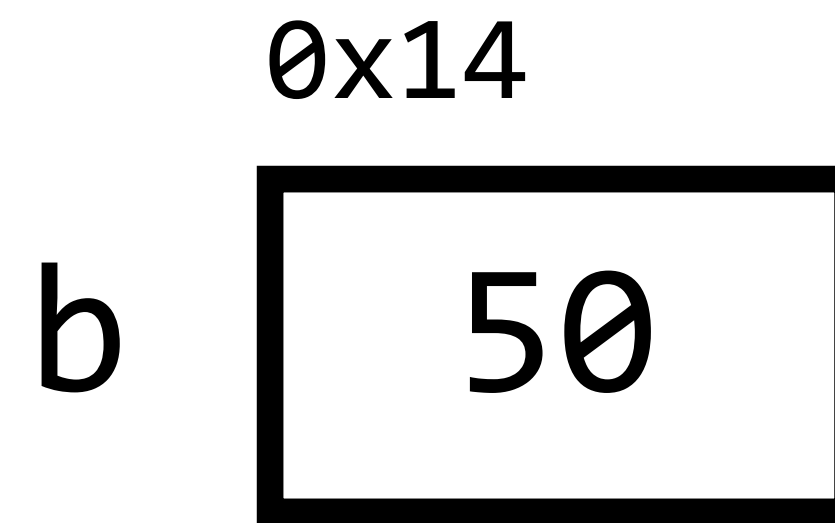
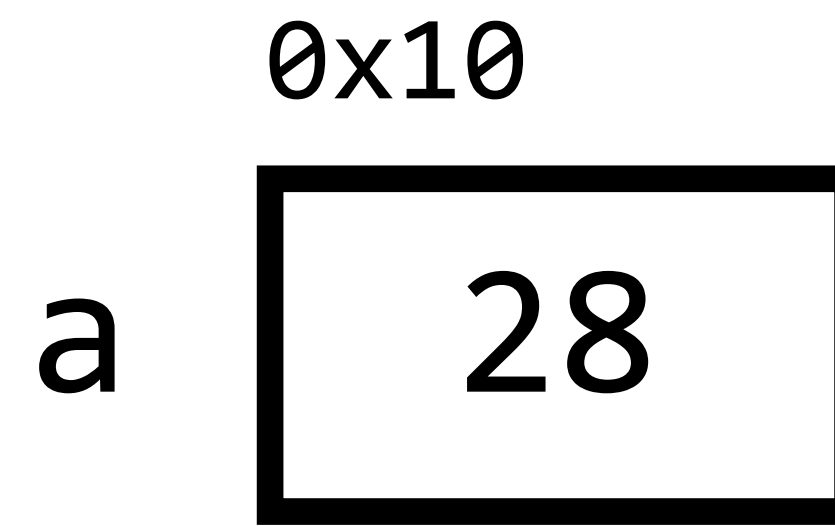
```
int *c = &a;
```



```
int a = 28;
```

```
int b = 50;
```

```
int *c = &a;
```

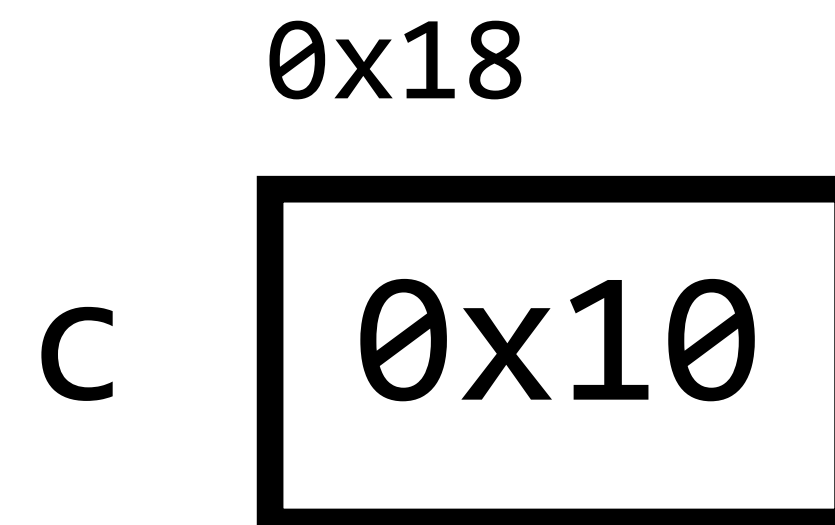
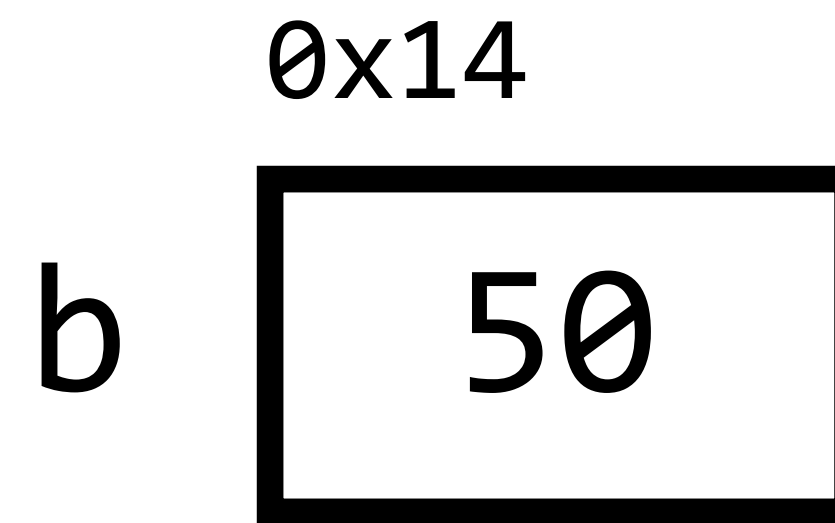
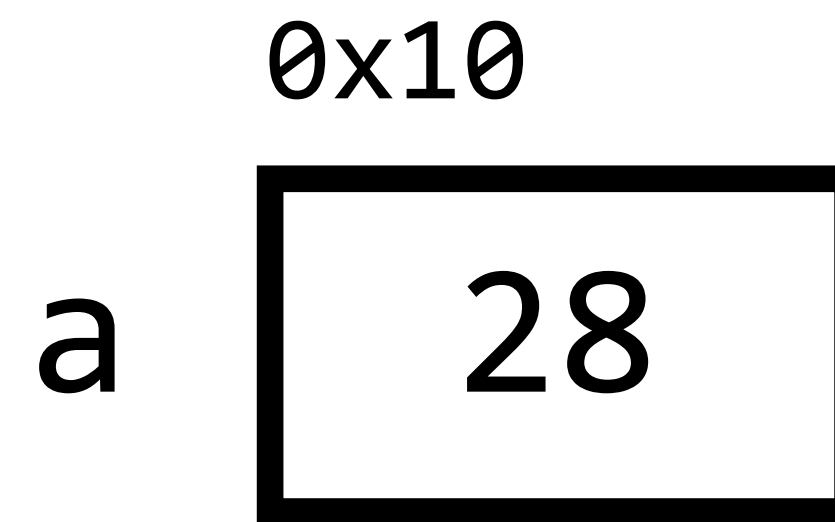


```
int a = 28;
```

```
int b = 50;
```

```
int *c = &a;
```

```
*c = 14;
```

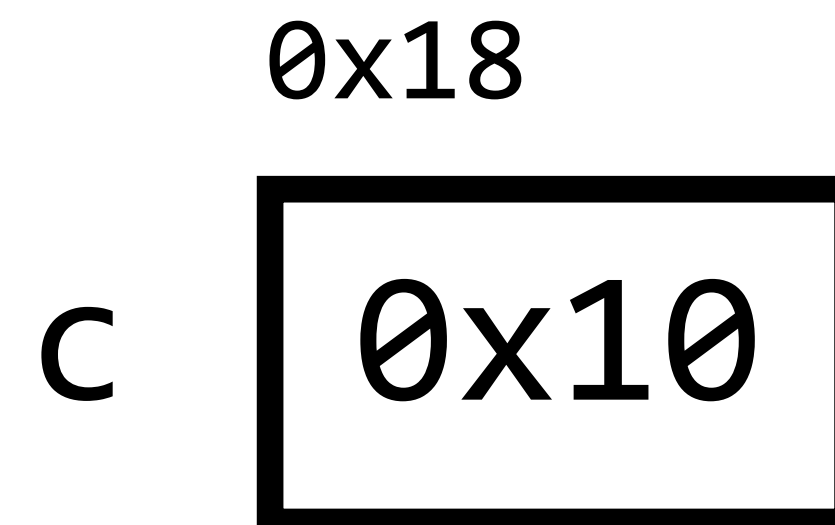
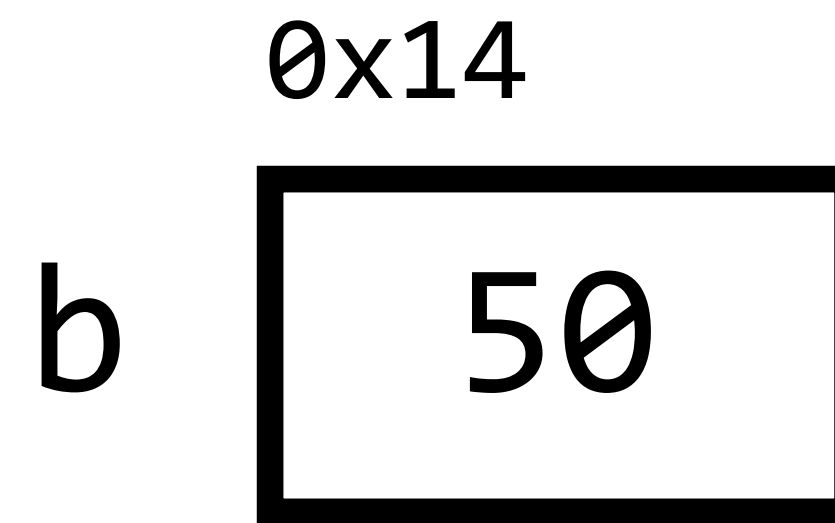
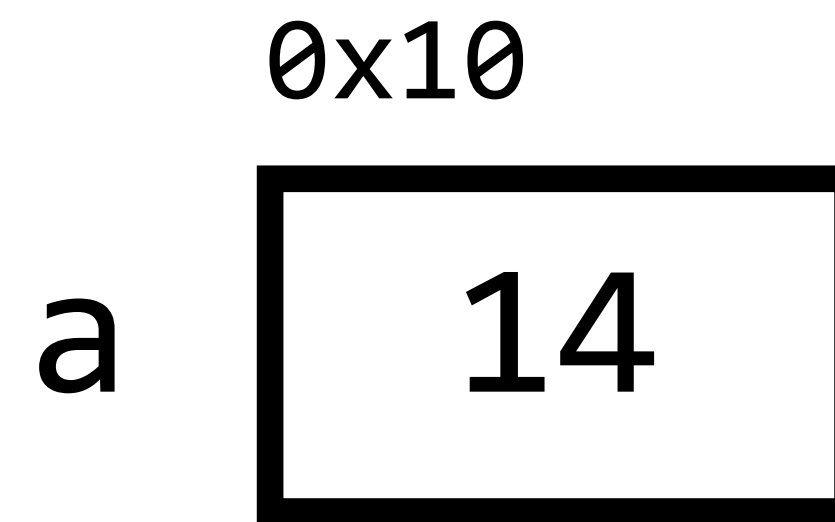


```
int a = 28;
```

```
int b = 50;
```

```
int *c = &a;
```

```
*c = 14;
```



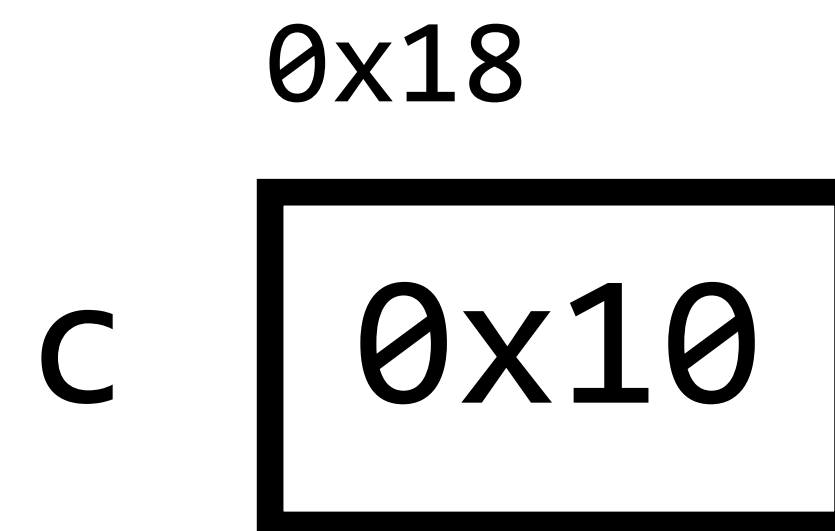
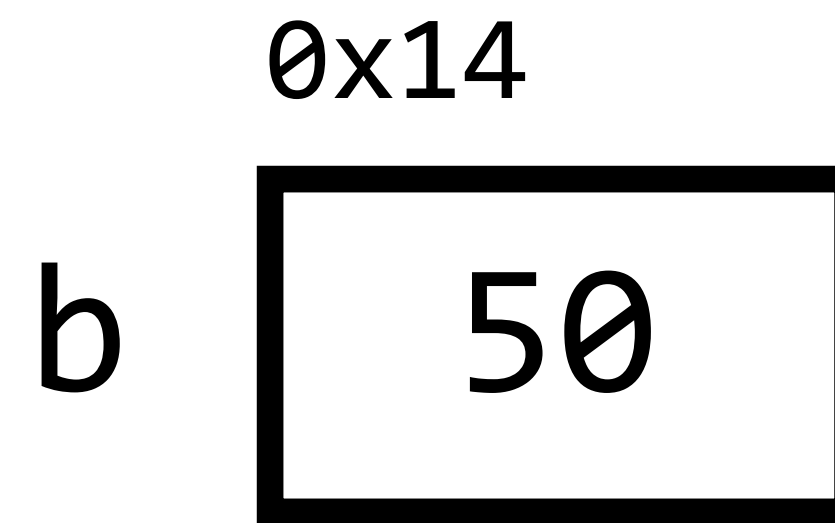
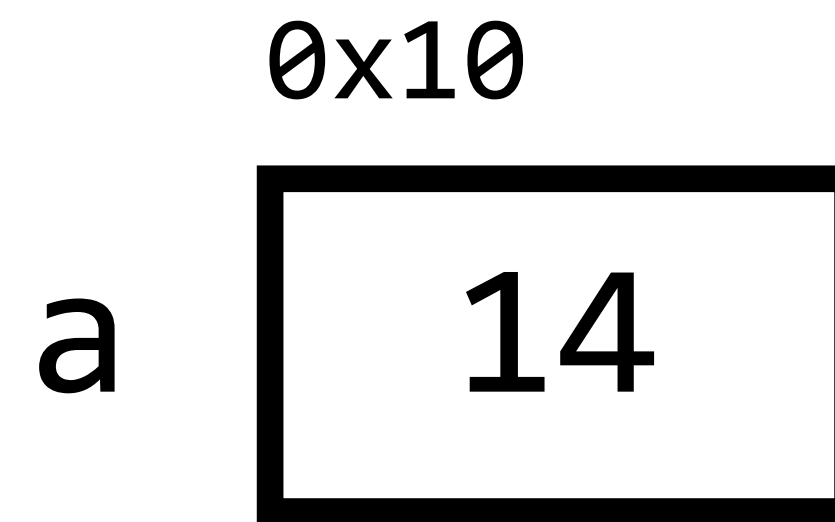
```
int a = 28;
```

```
int b = 50;
```

```
int *c = &a;
```

```
*c = 14;
```

```
c = &b;
```



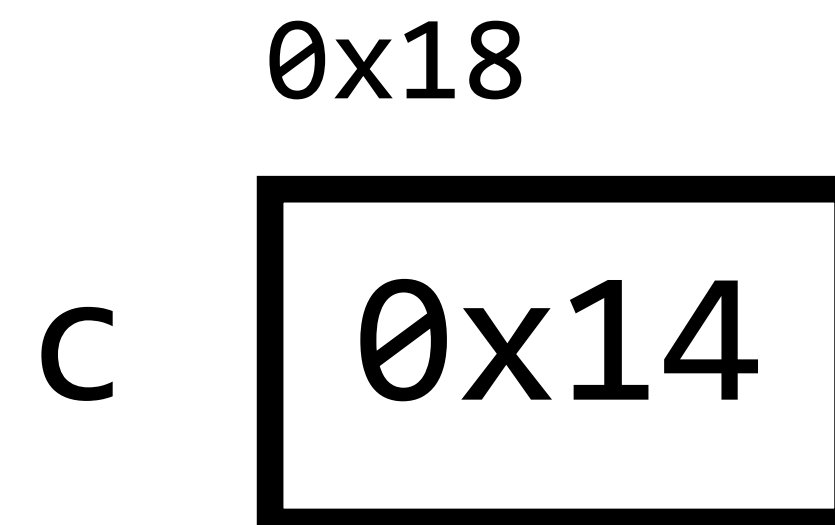
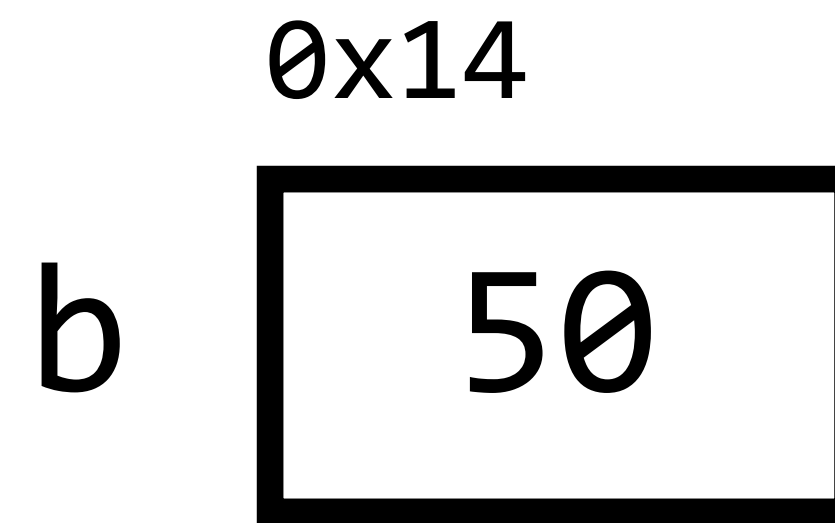
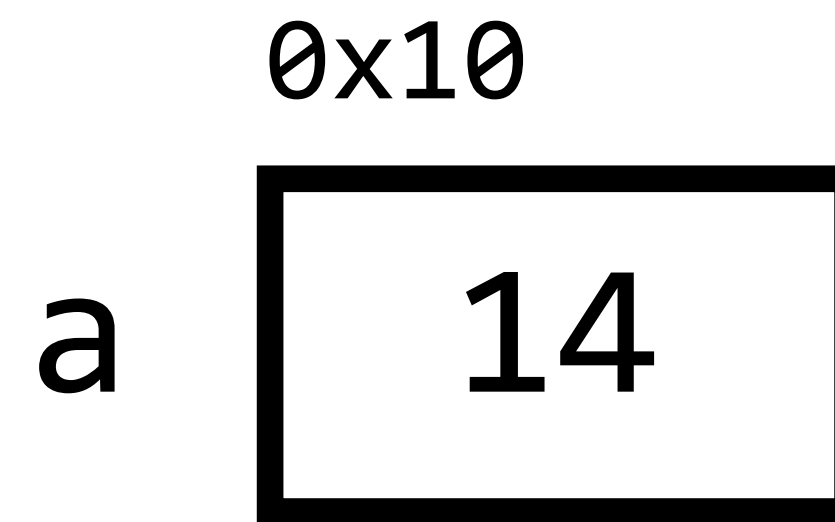
```
int a = 28;
```

```
int b = 50;
```

```
int *c = &a;
```

```
*c = 14;
```

```
c = &b;
```



```
int a = 28;
```

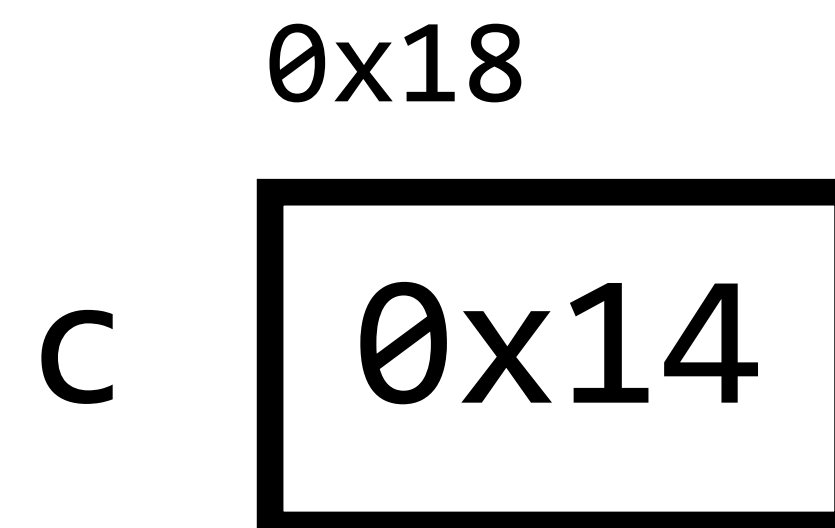
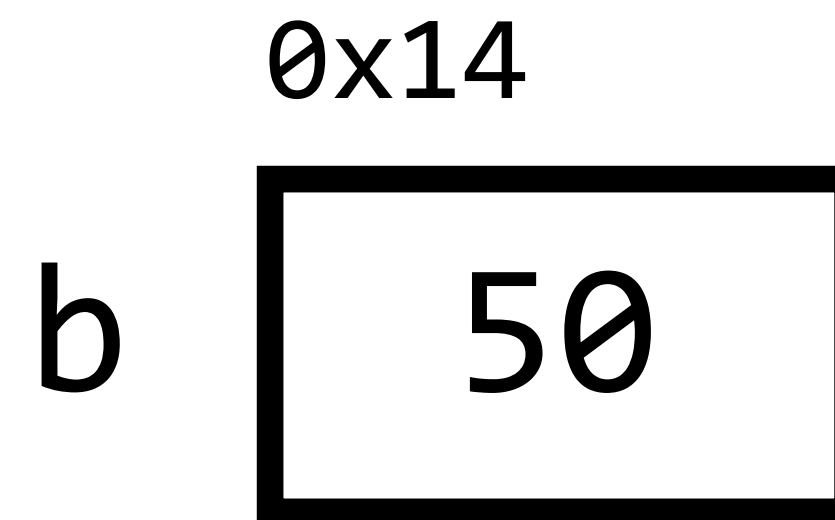
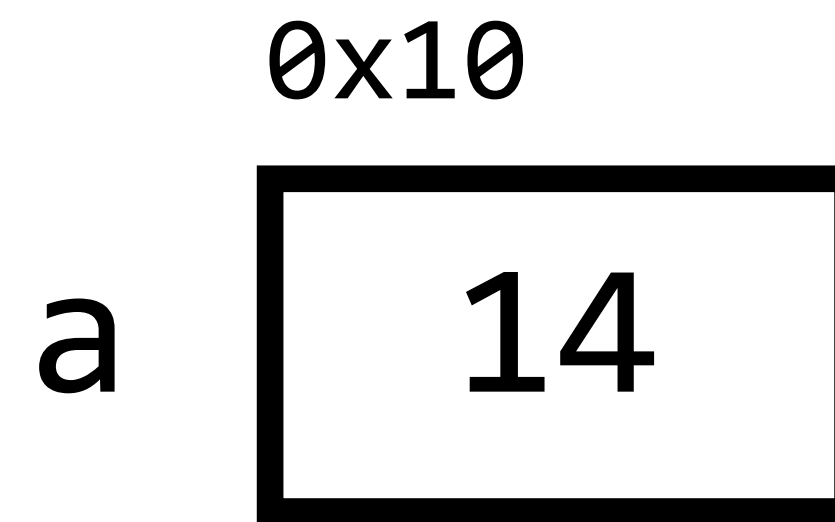
```
int b = 50;
```

```
int *c = &a;
```

```
*c = 14;
```

```
c = &b;
```

```
*c = 20;
```




```
int a = 28;
```

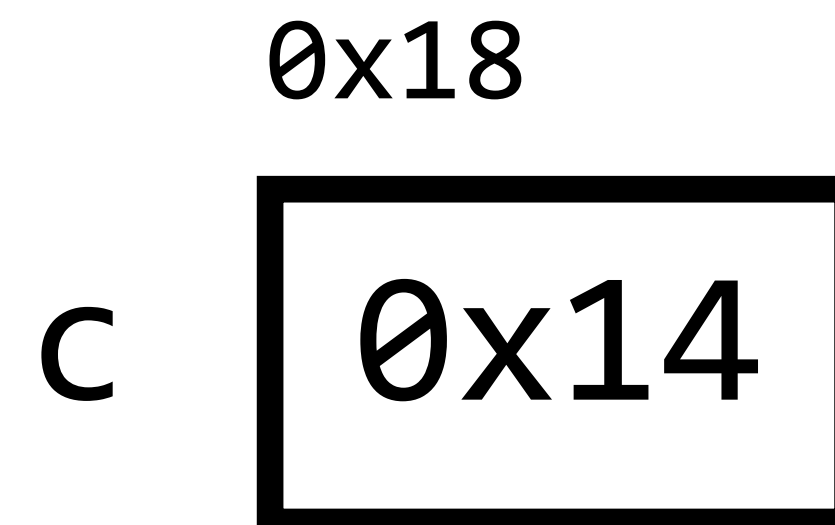
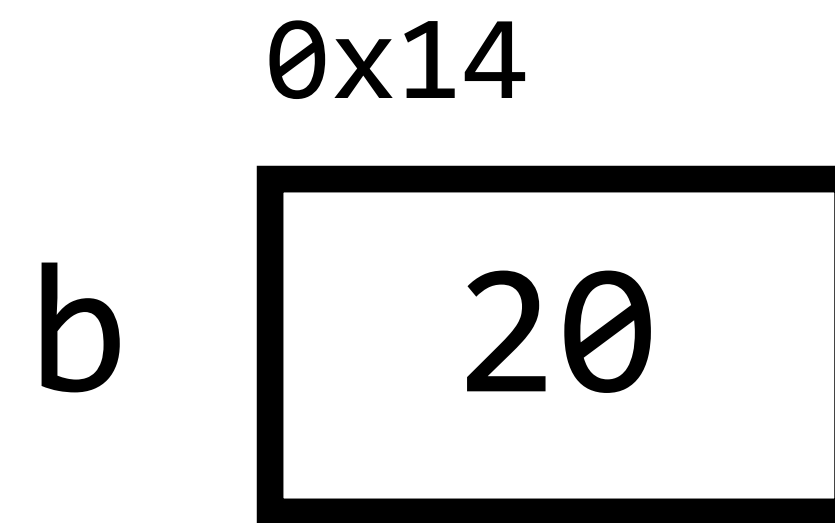
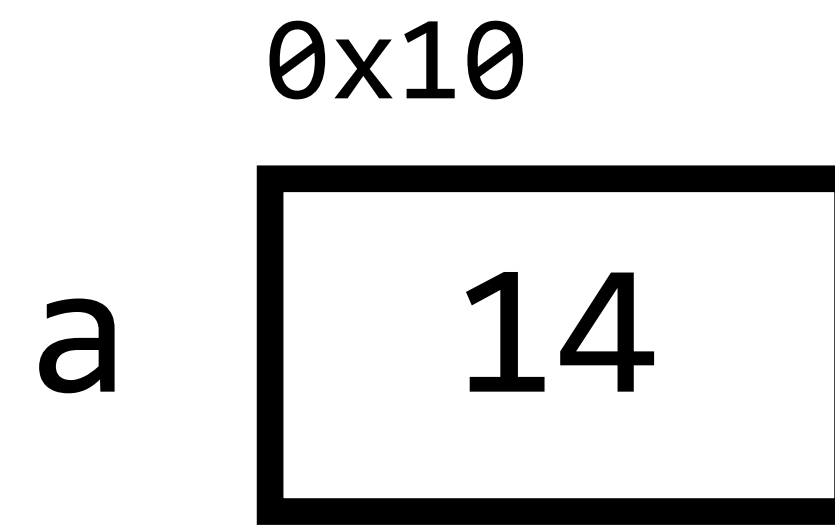
```
int b = 50;
```

```
int *c = &a;
```

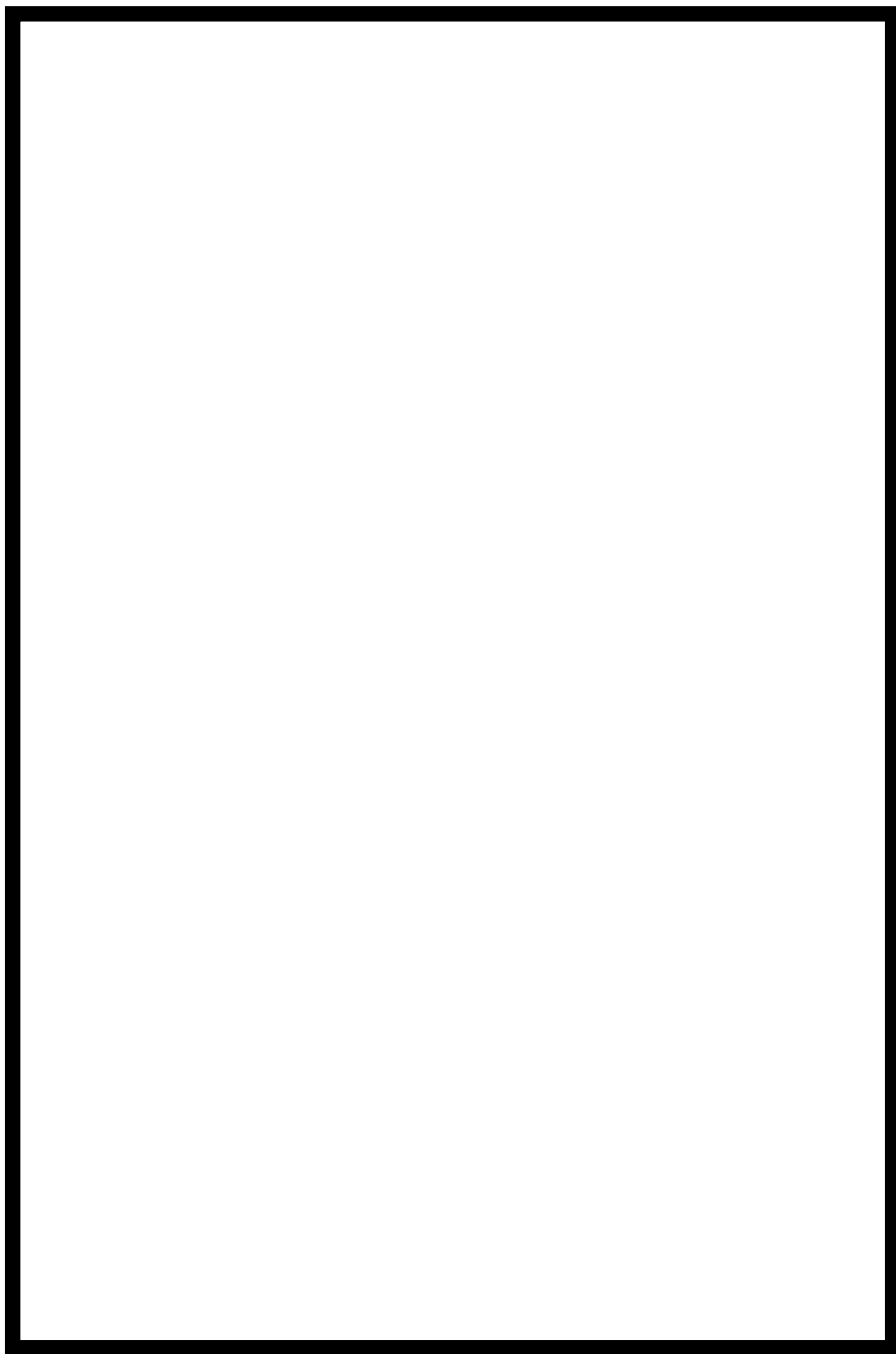
```
*c = 14;
```

```
c = &b;
```

```
*c = 20;
```

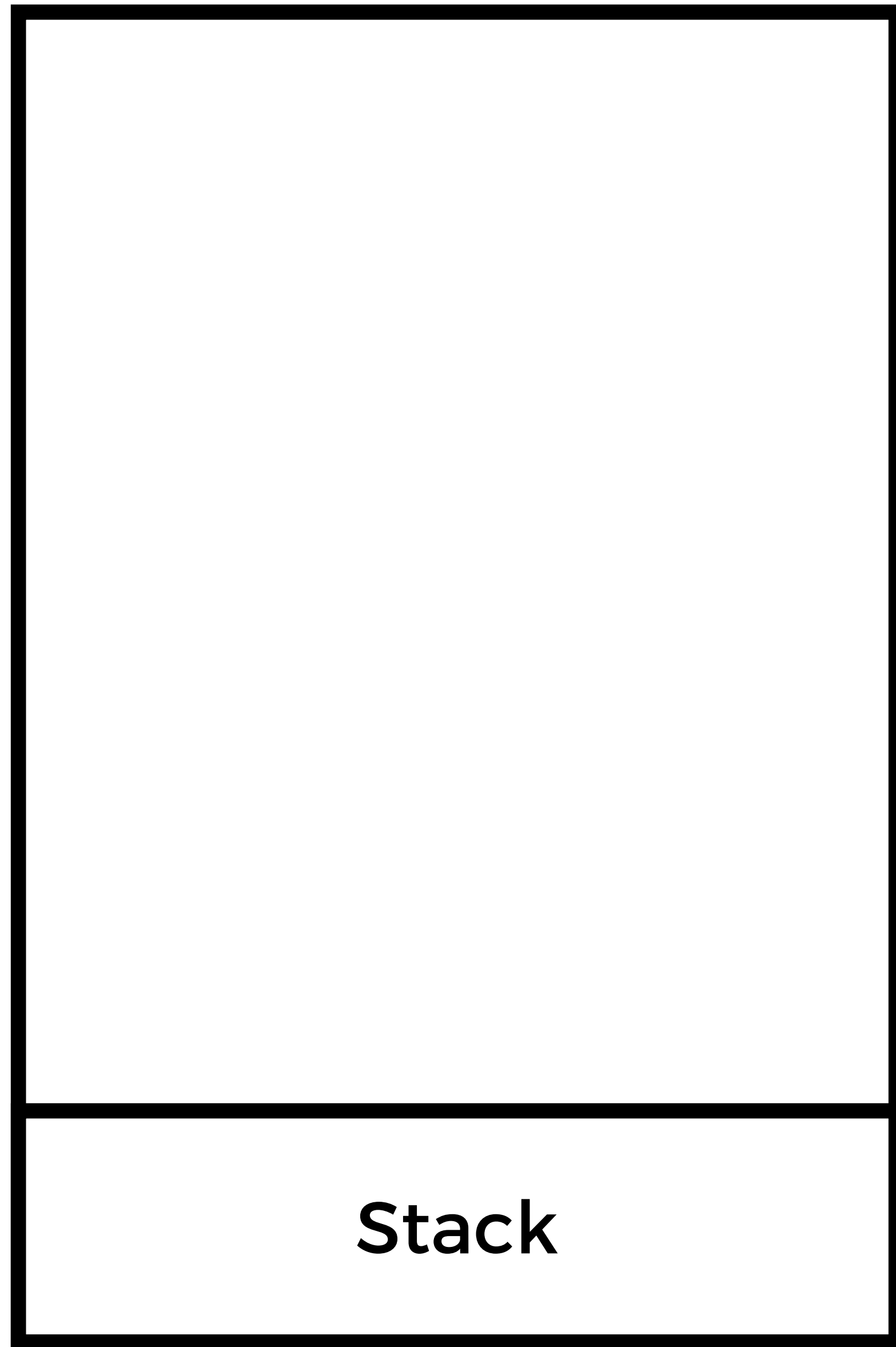


Memory Layout





Stack



local variables and functions



A vertical rectangle is divided into three horizontal sections by two thick black lines. The top section is labeled 'Heap', the middle section is empty, and the bottom section is labeled 'Stack'.

Heap

Stack

local variables and functions



A vertical rectangle is divided into three horizontal sections. The top section is labeled 'Heap', the middle section is empty, and the bottom section is labeled 'Stack'.

Heap

dynamically allocated memory

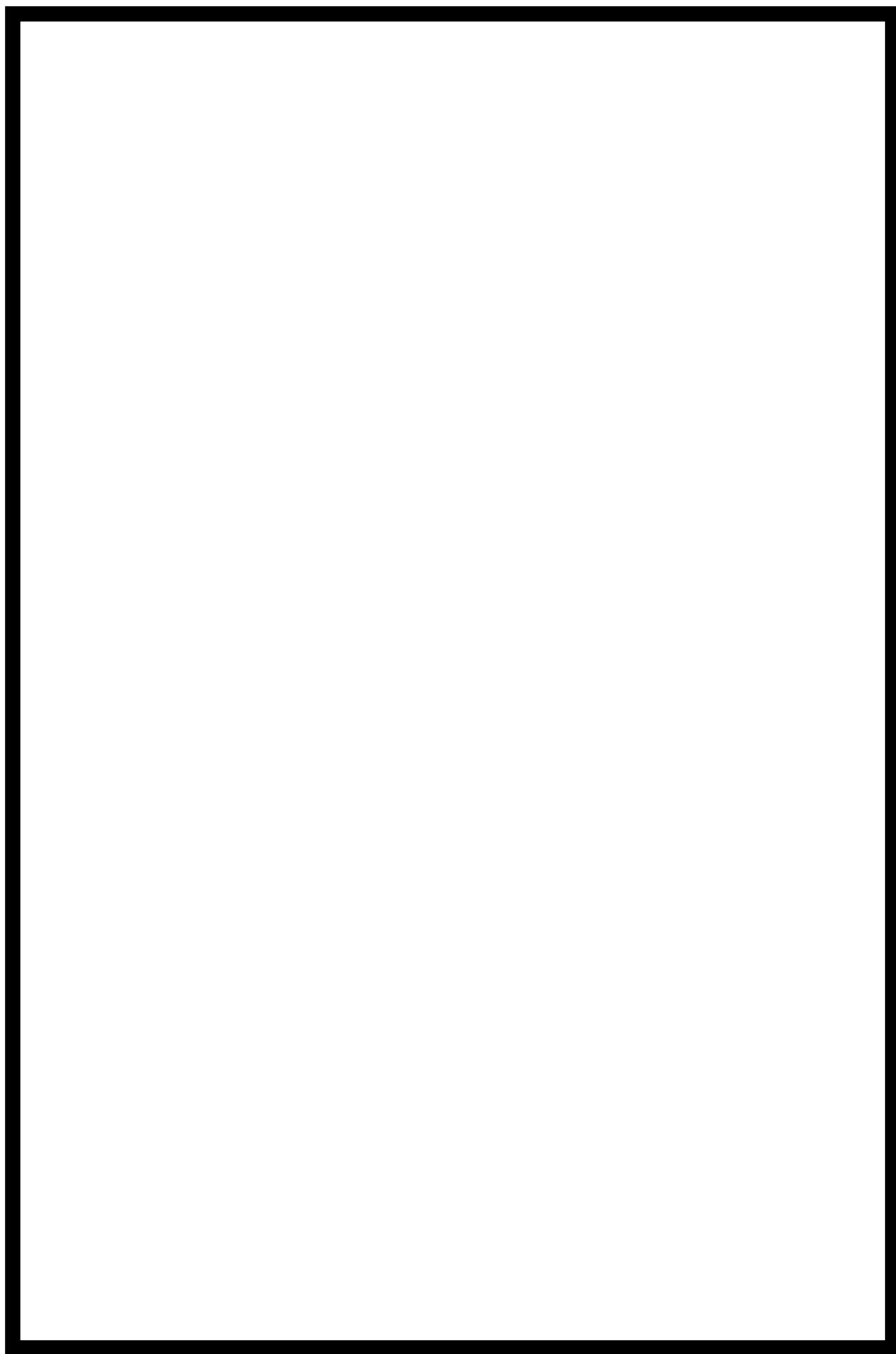
Stack

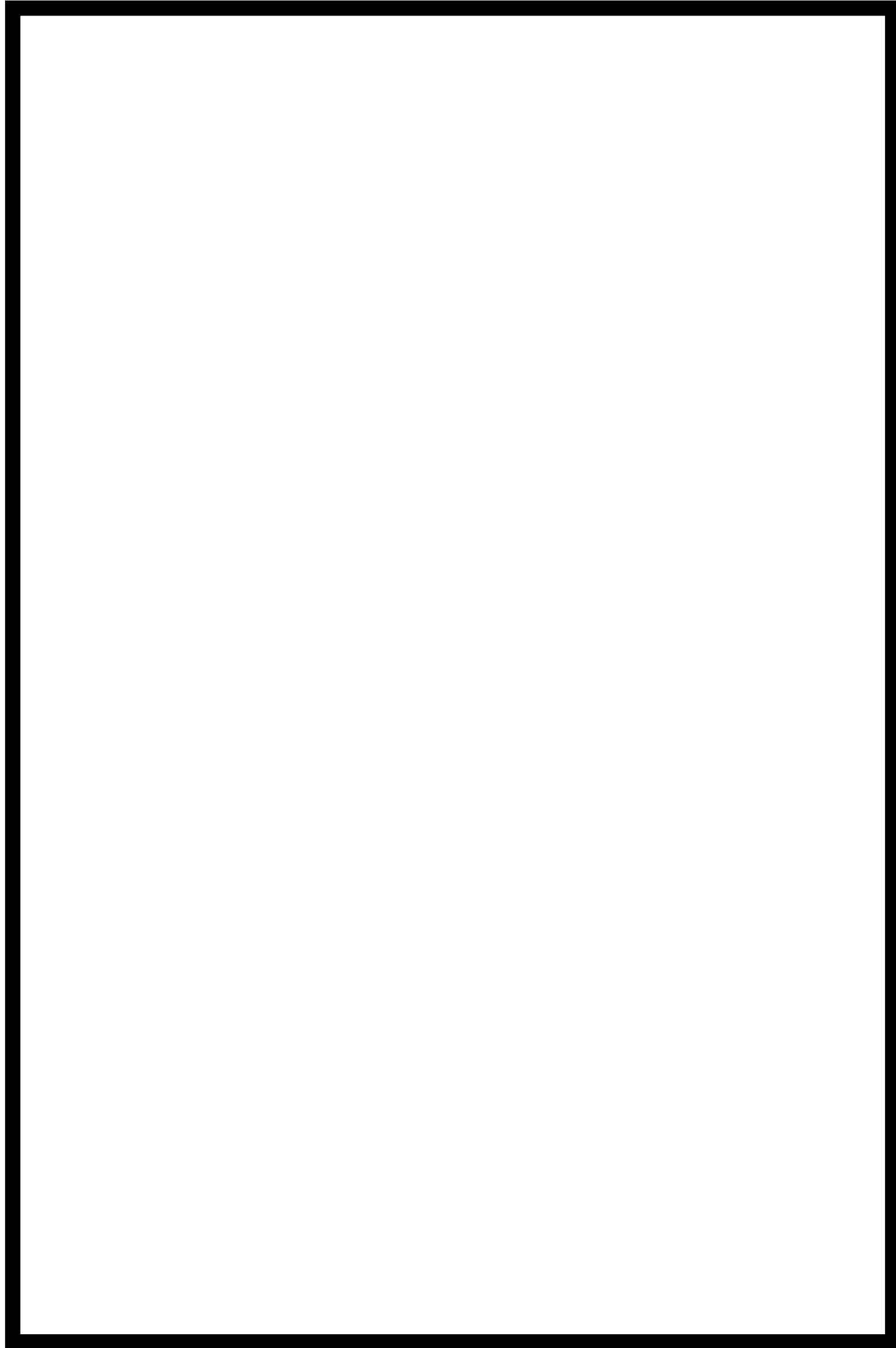
local variables and functions

Heap



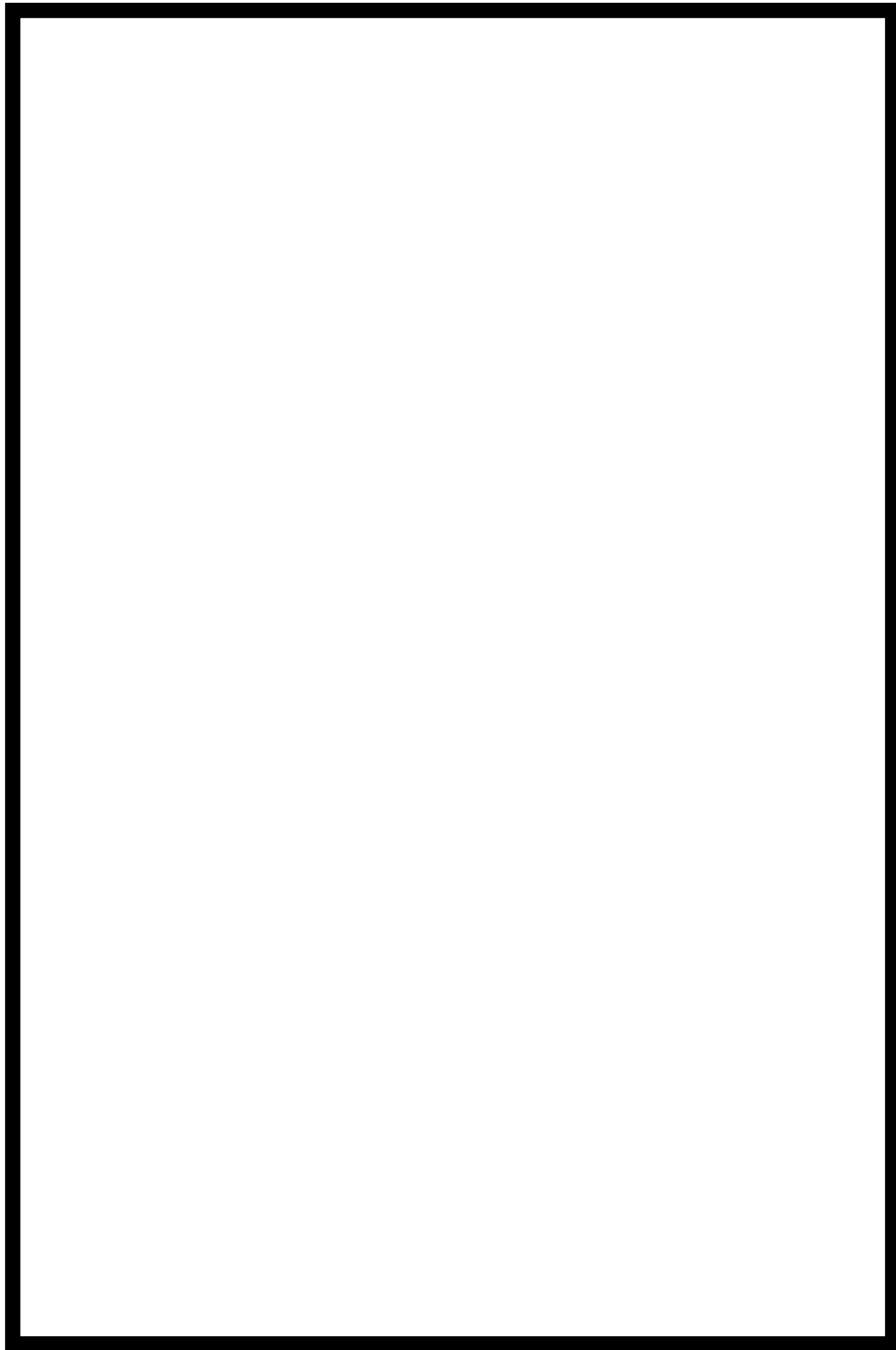
Stack





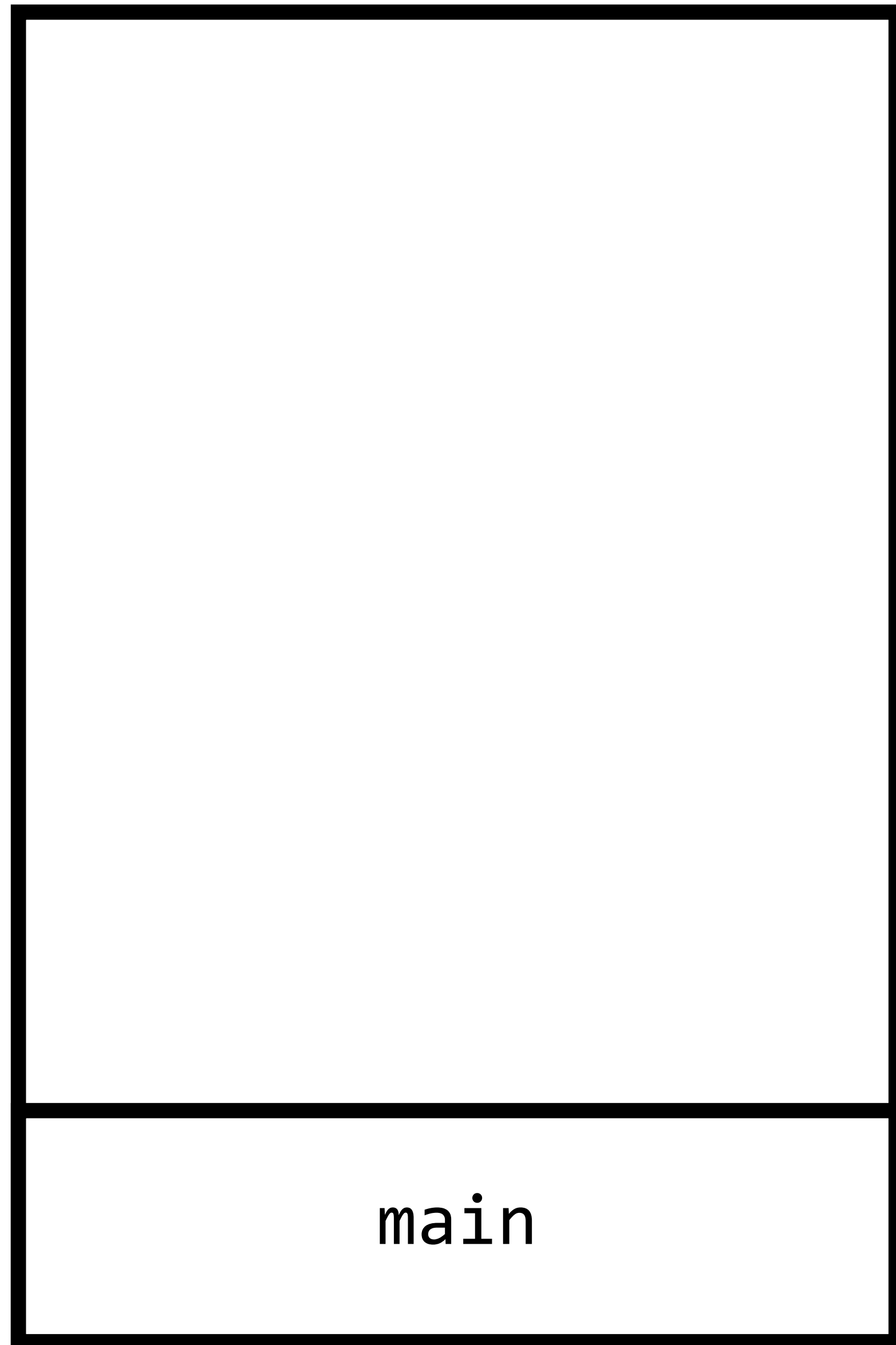
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



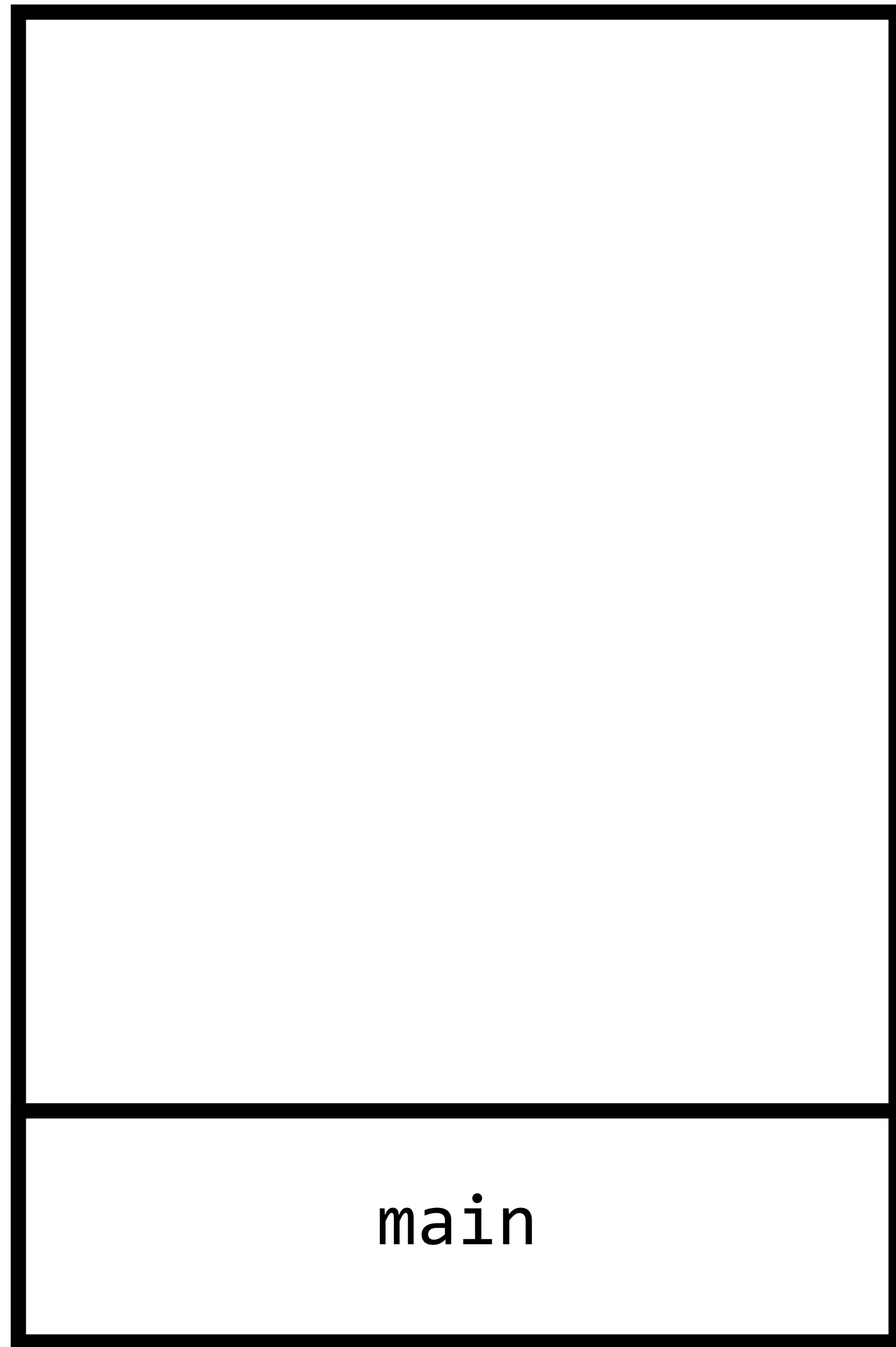
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



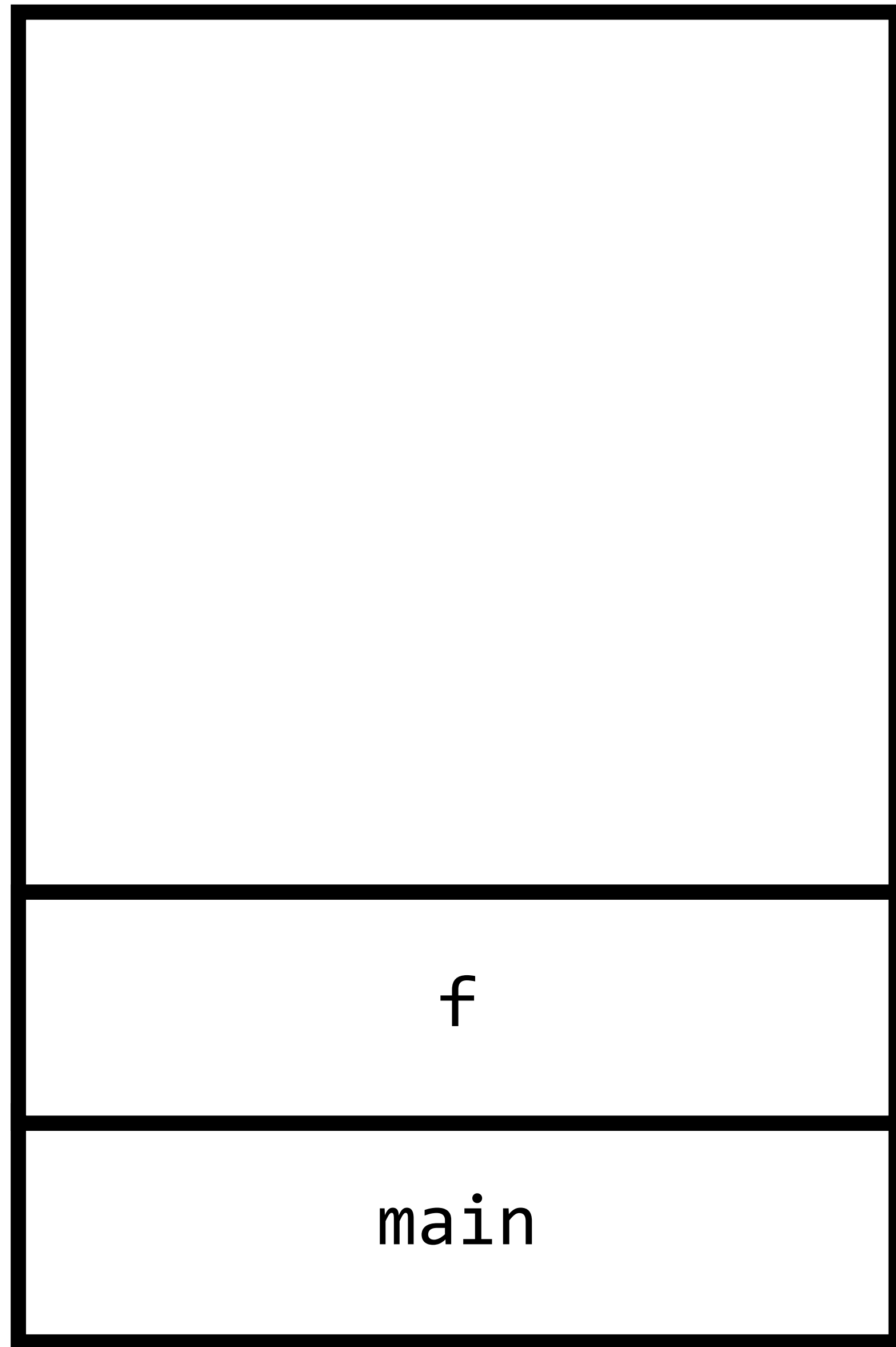
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



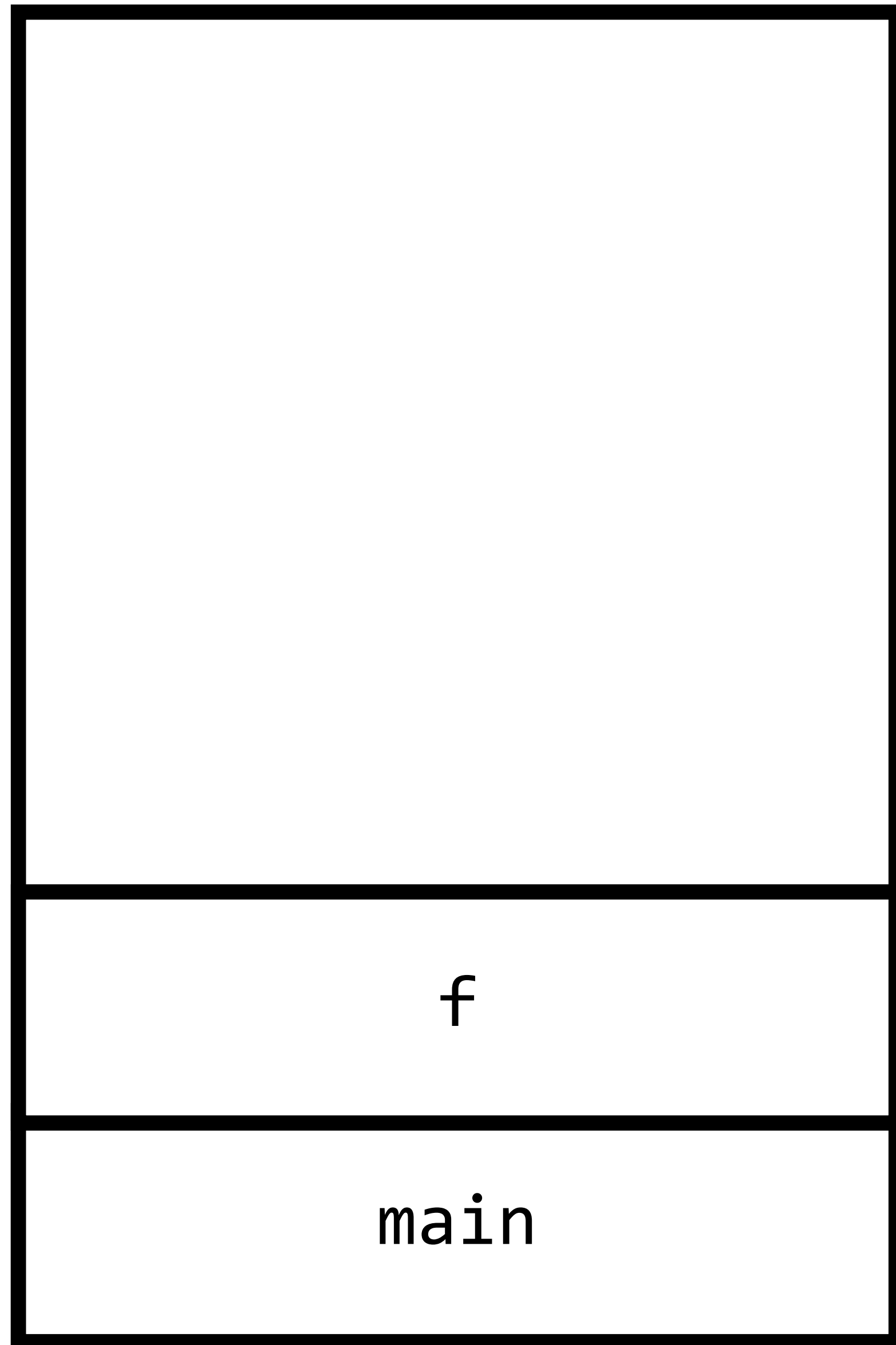
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



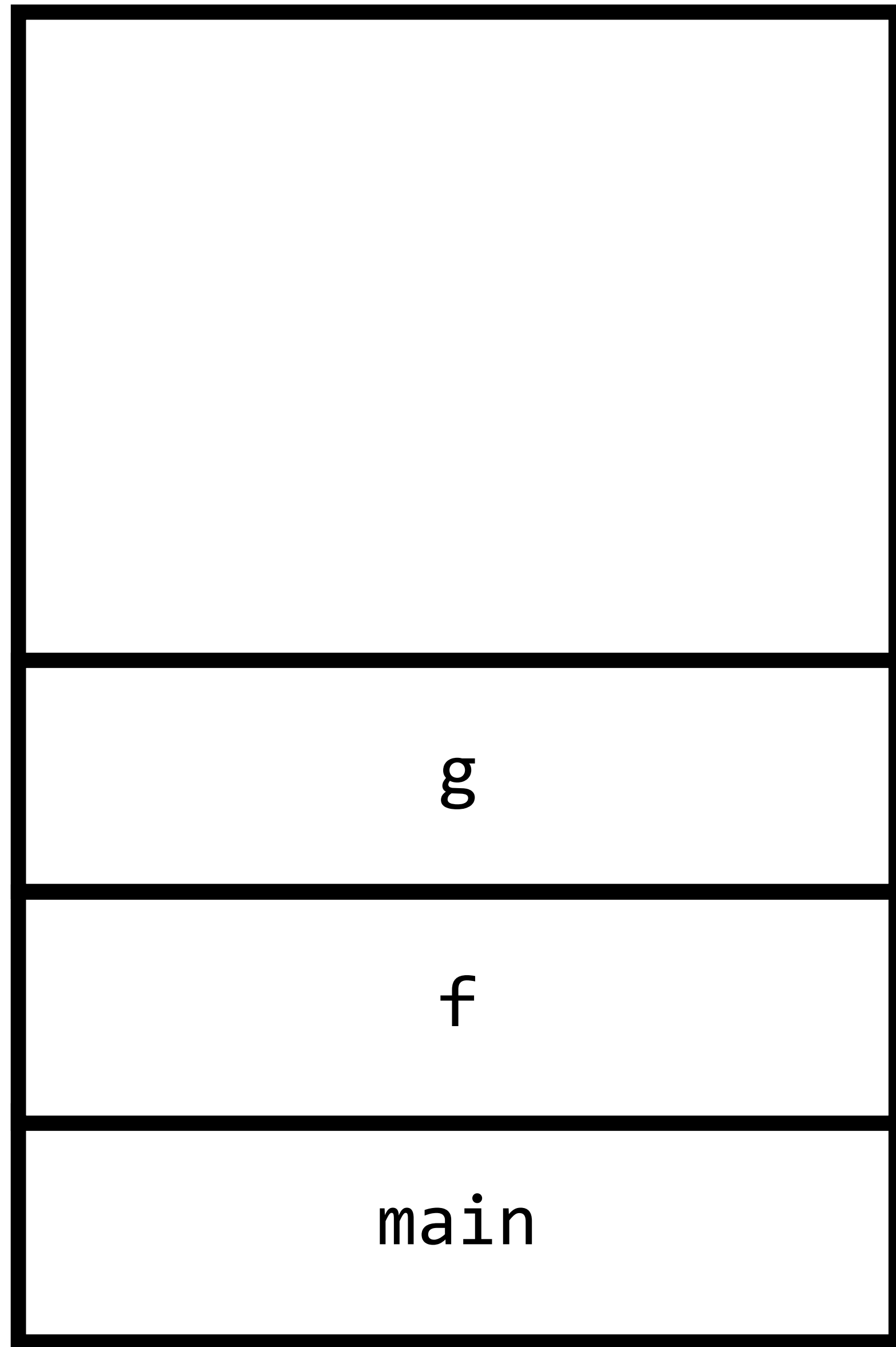
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



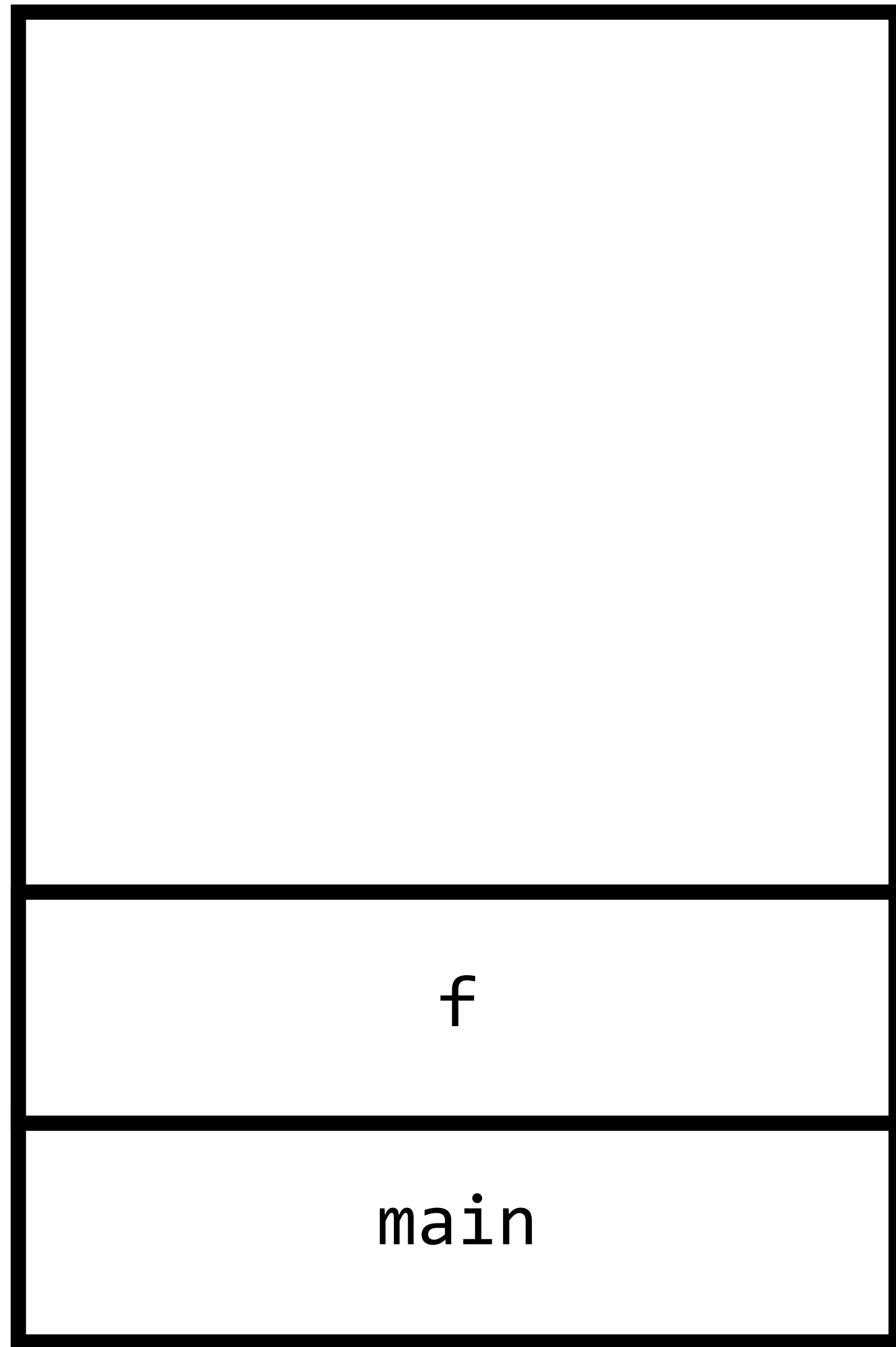
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



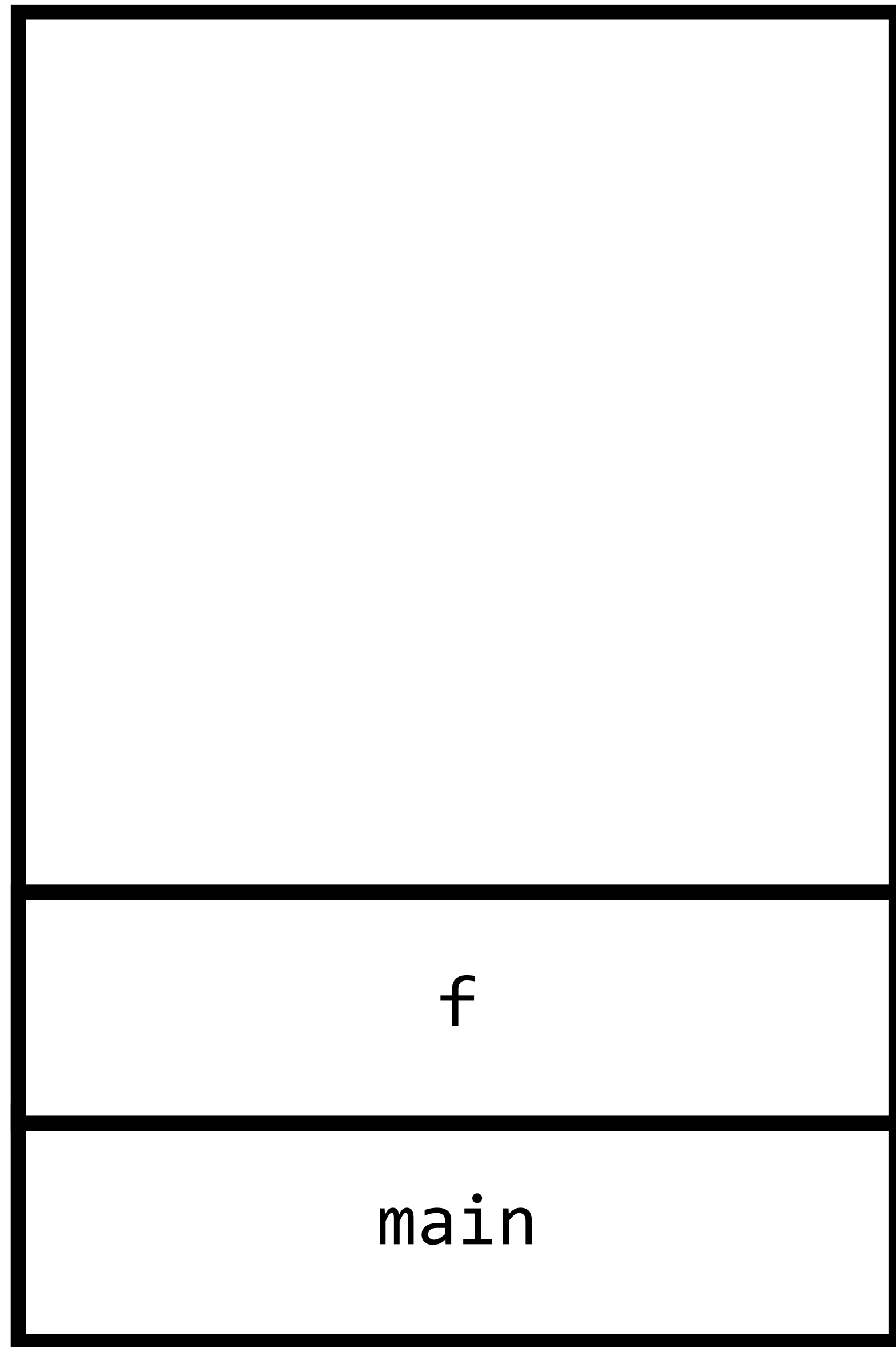
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```

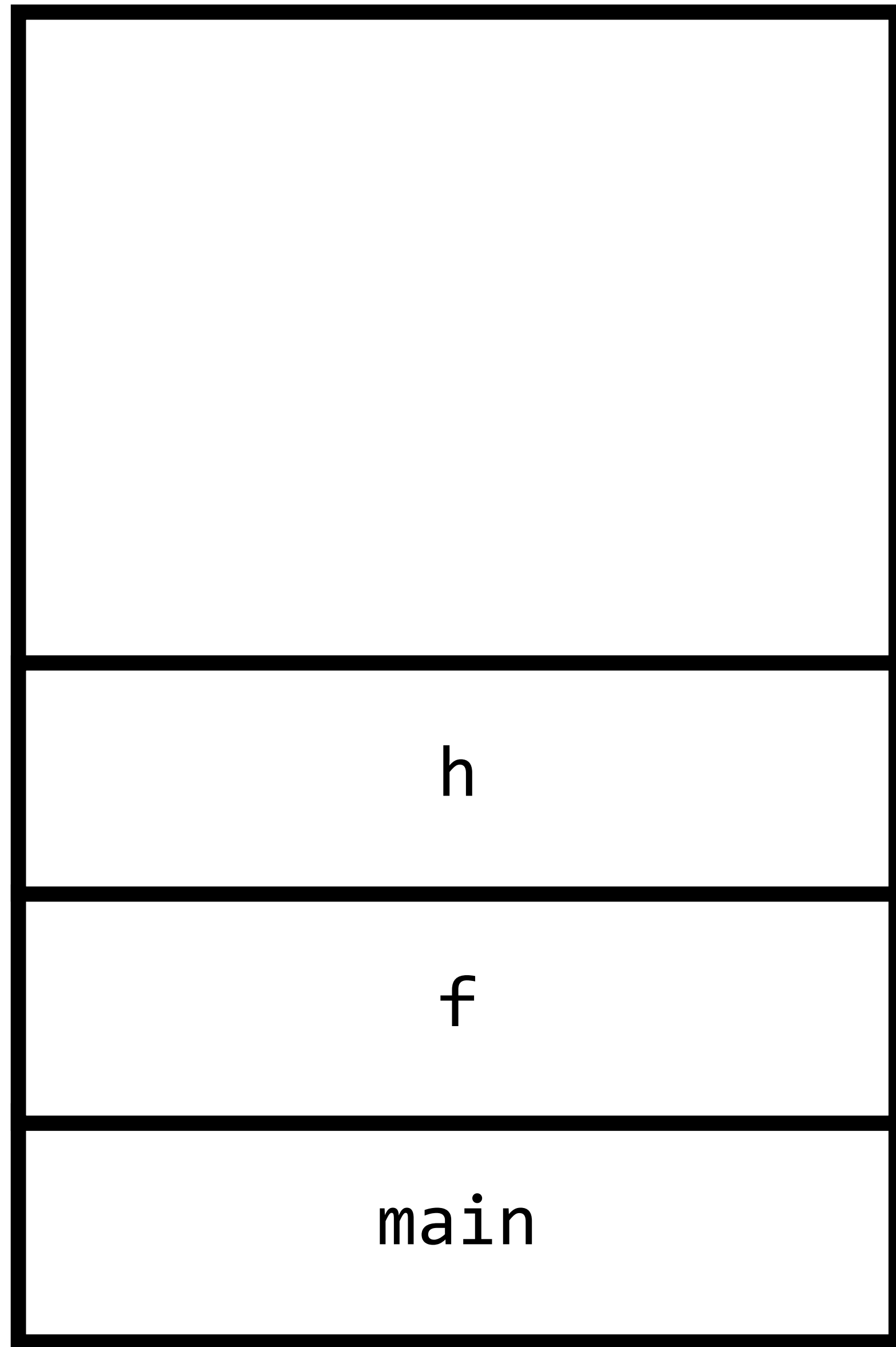
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



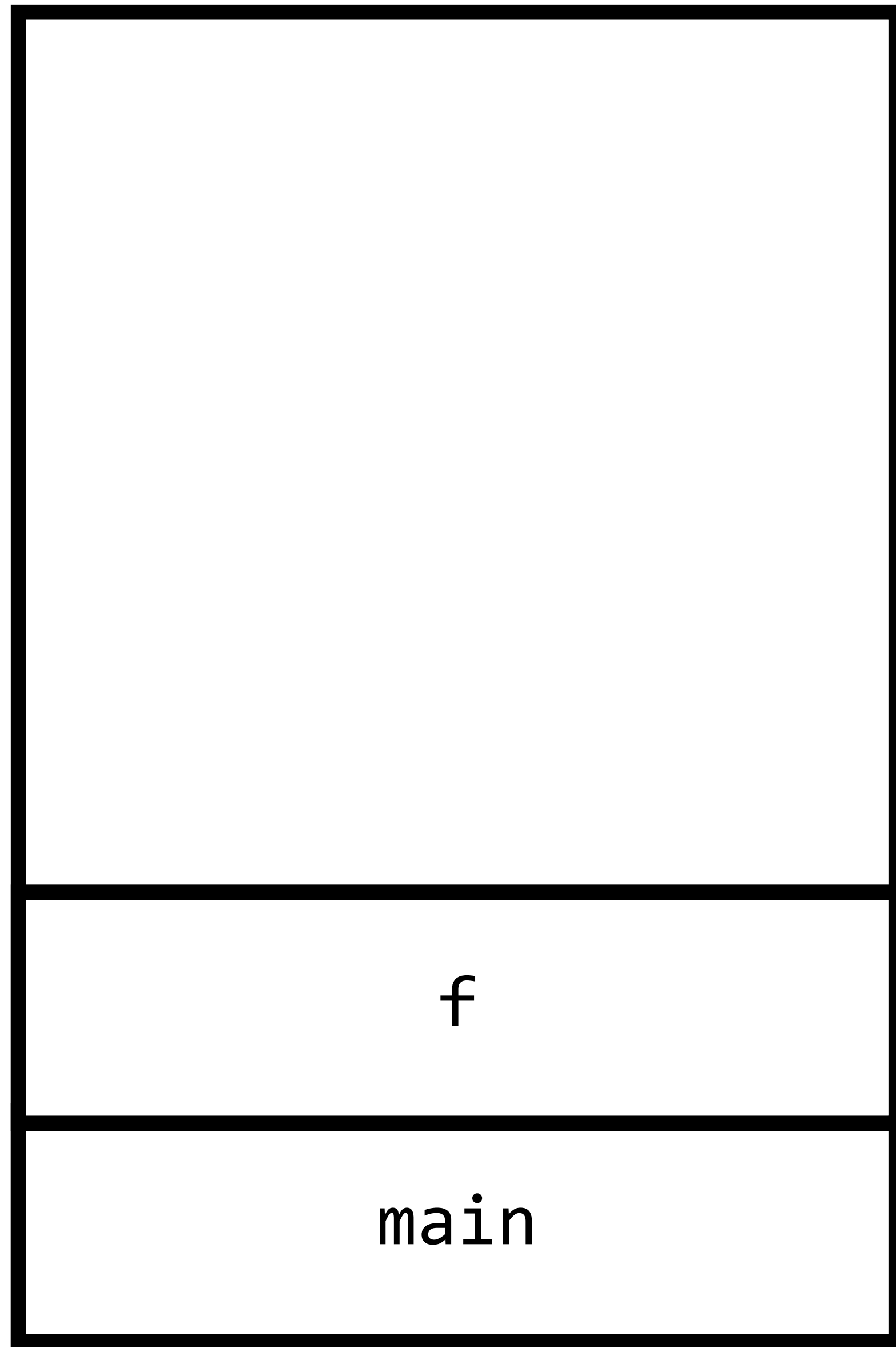
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



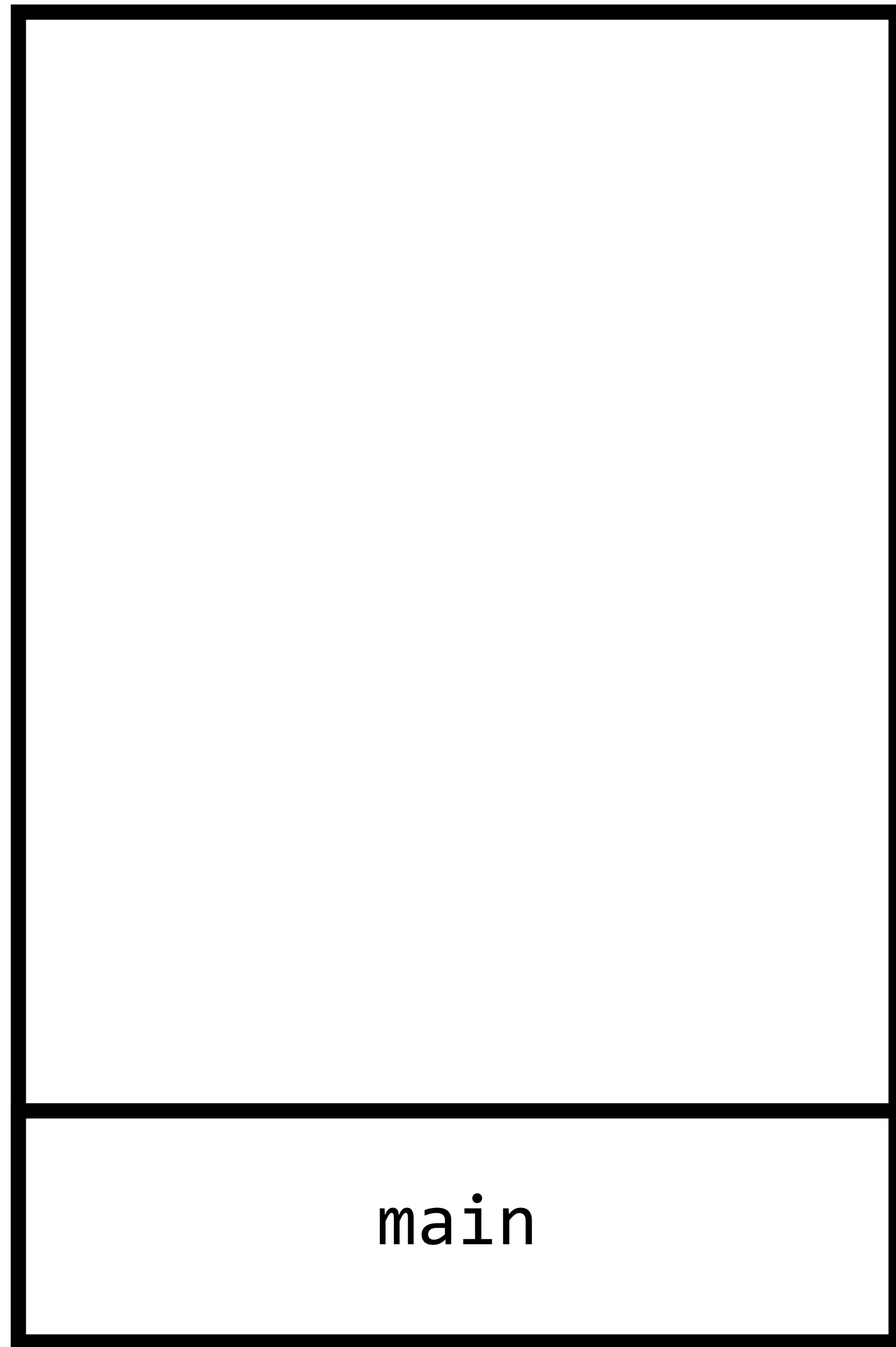
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



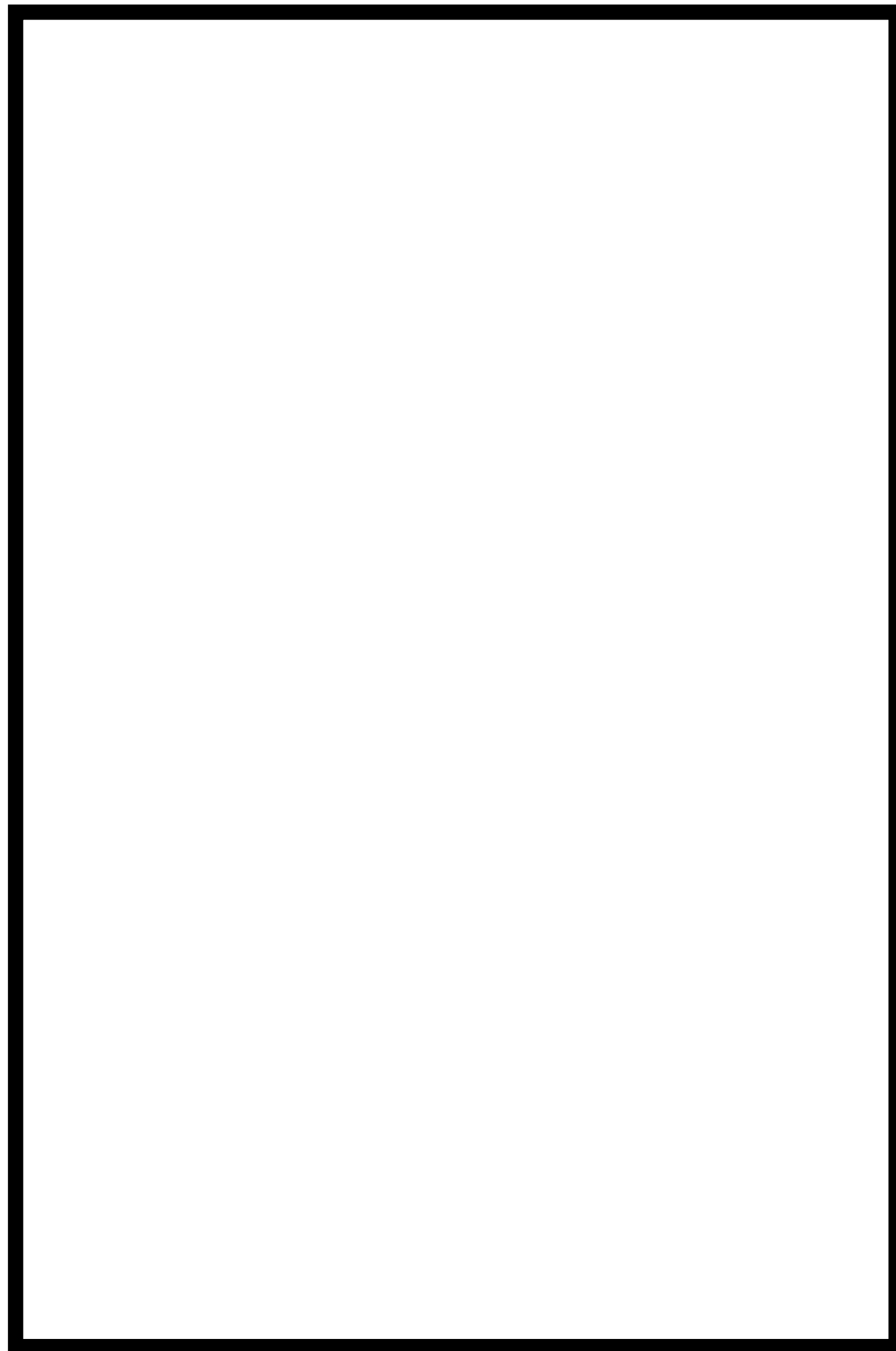
```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```



```
int main(void)
{
    f();
}

void f(void)
{
    g();
    h();
}
```

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

x 50

y 100

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



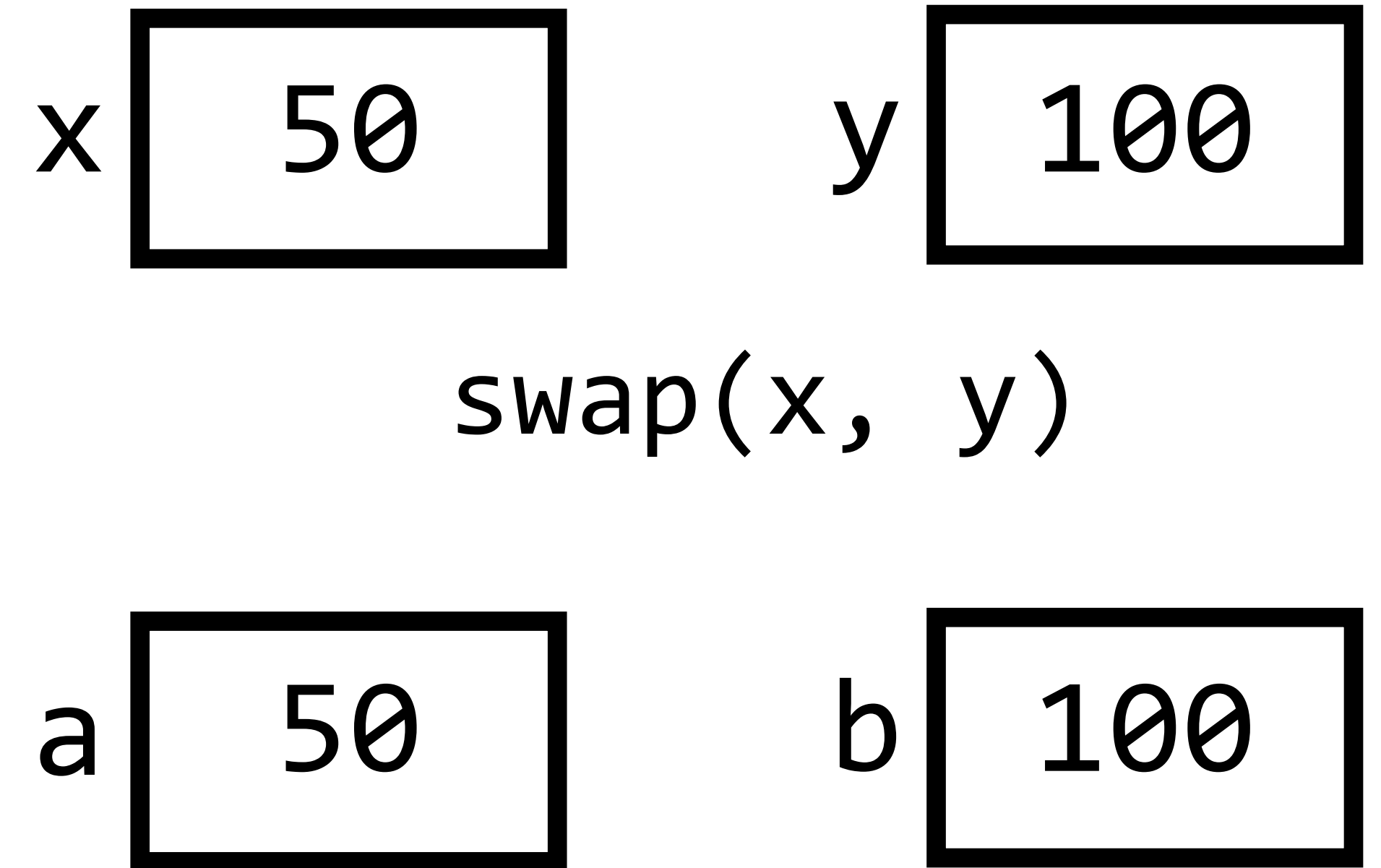
swap(x, y)

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

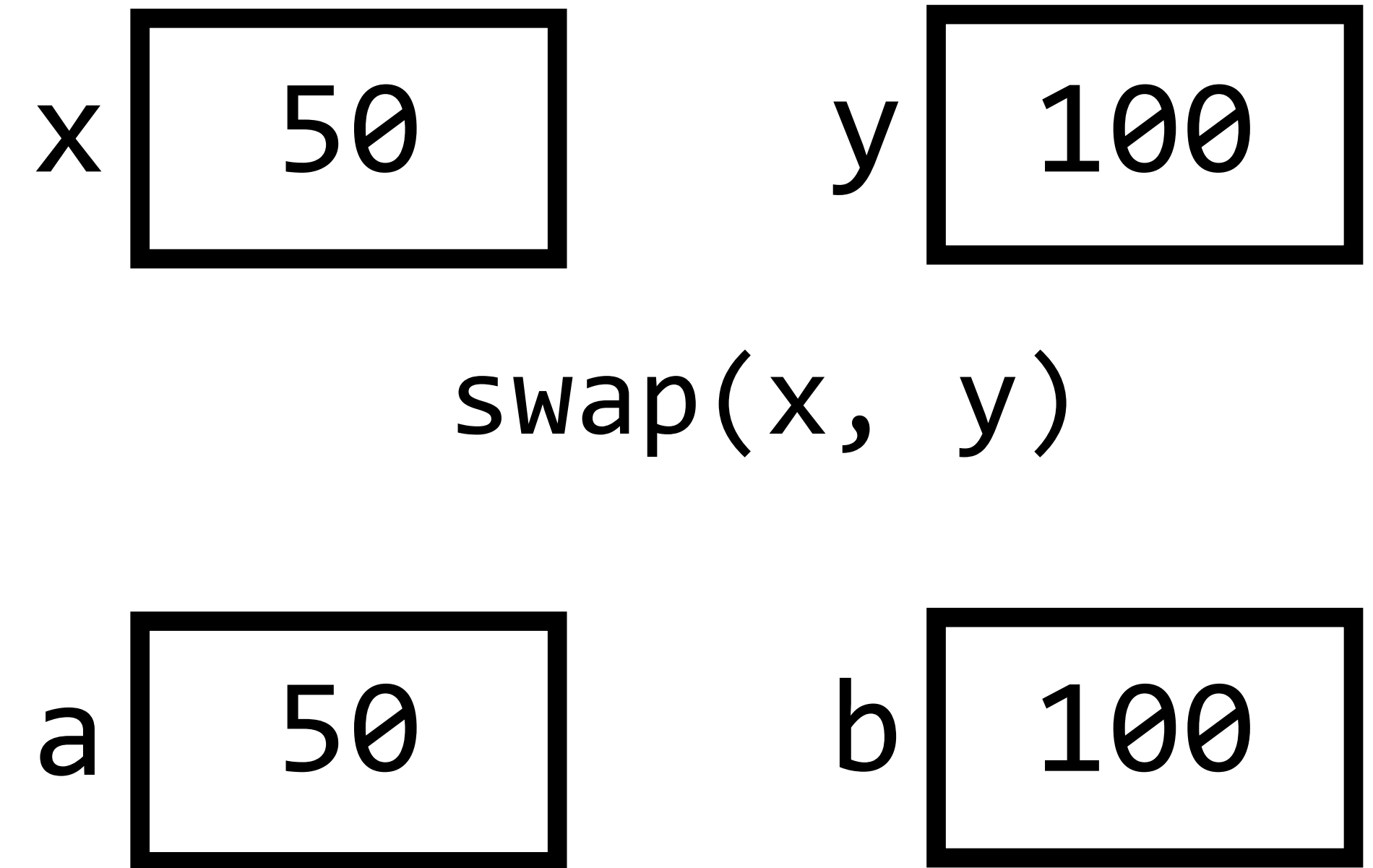


swap(x, y)

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



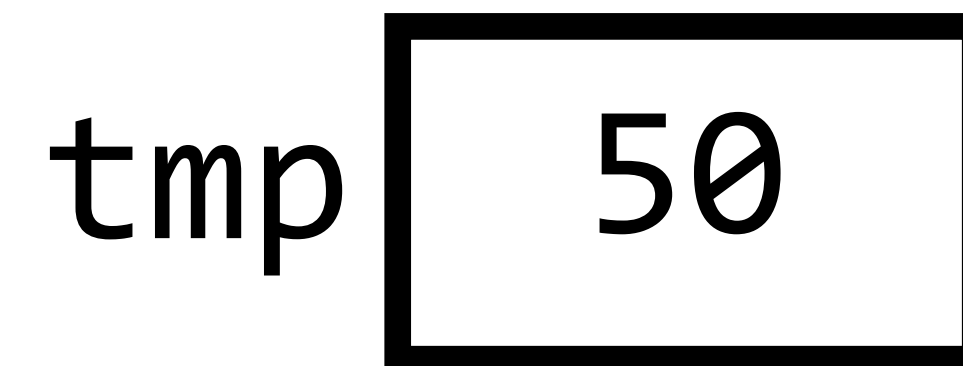
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



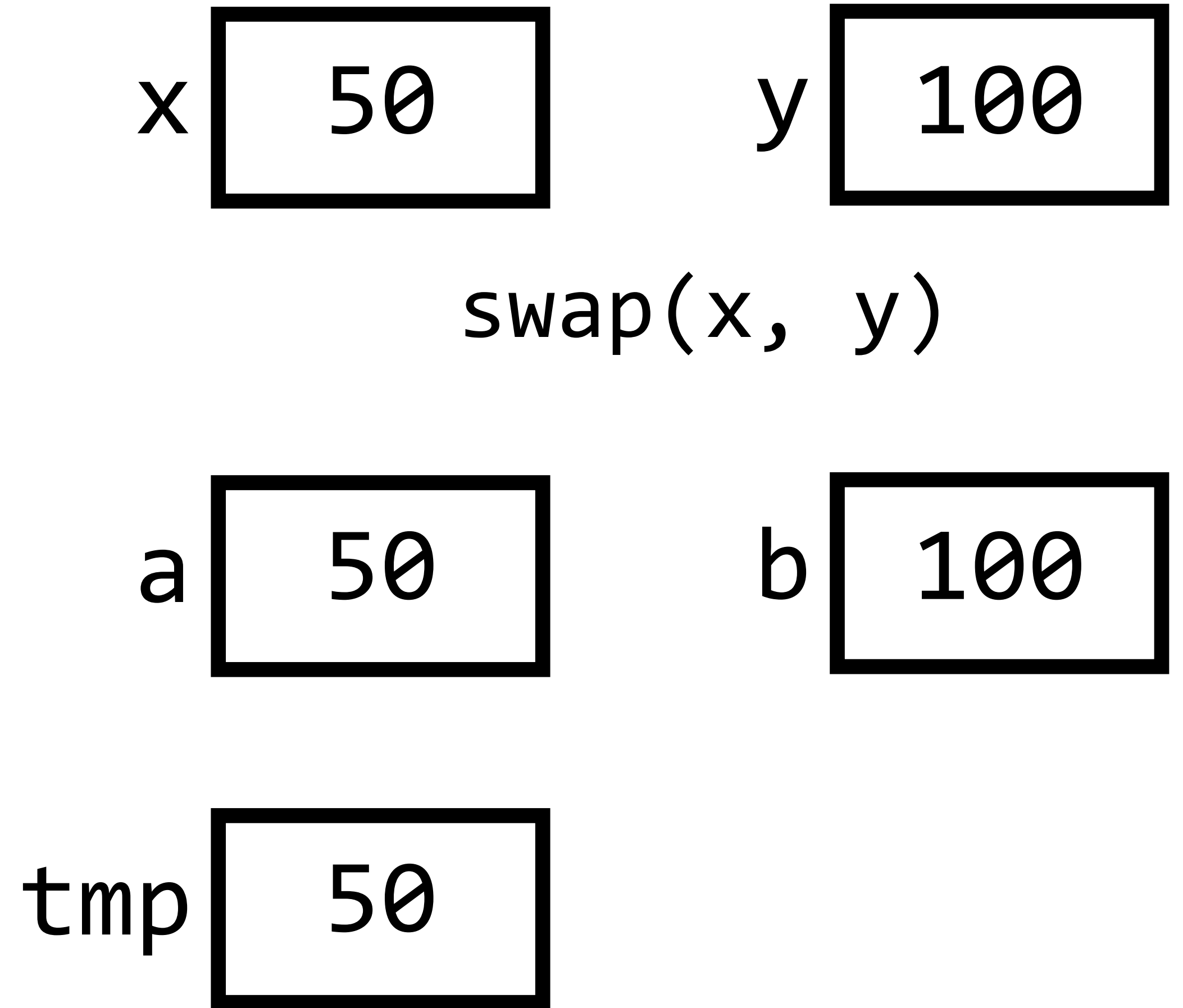
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



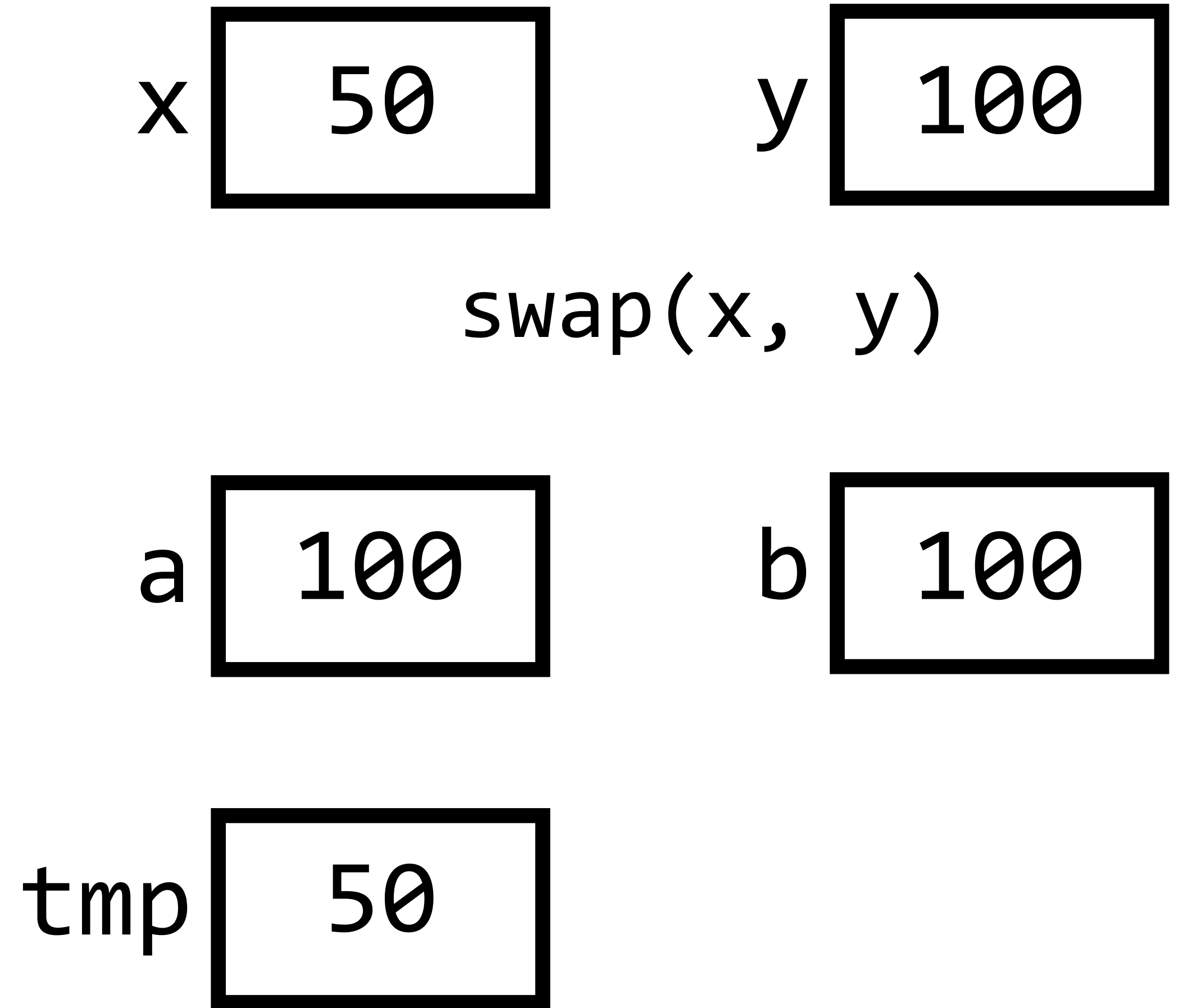
swap(x, y)



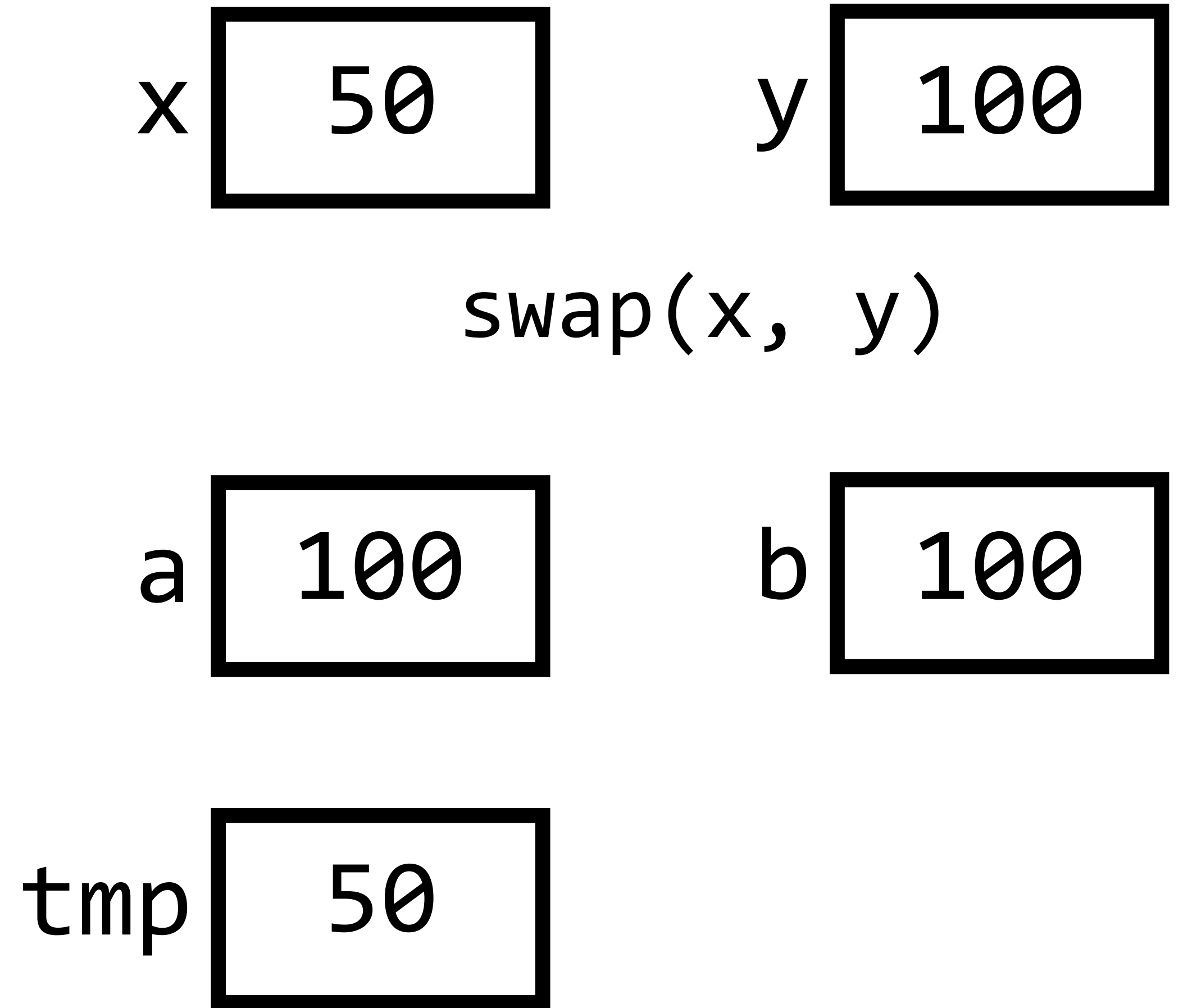
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



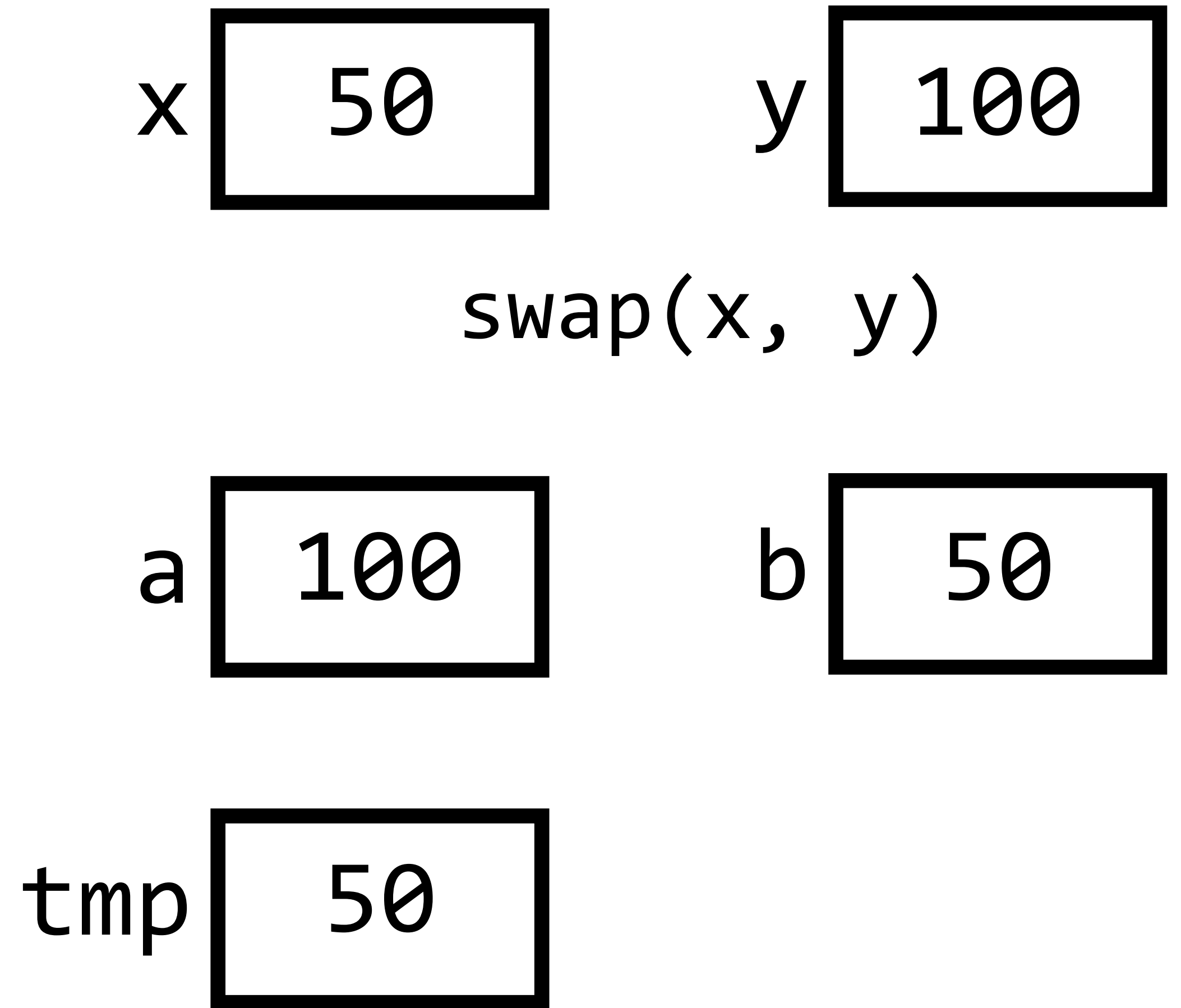

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



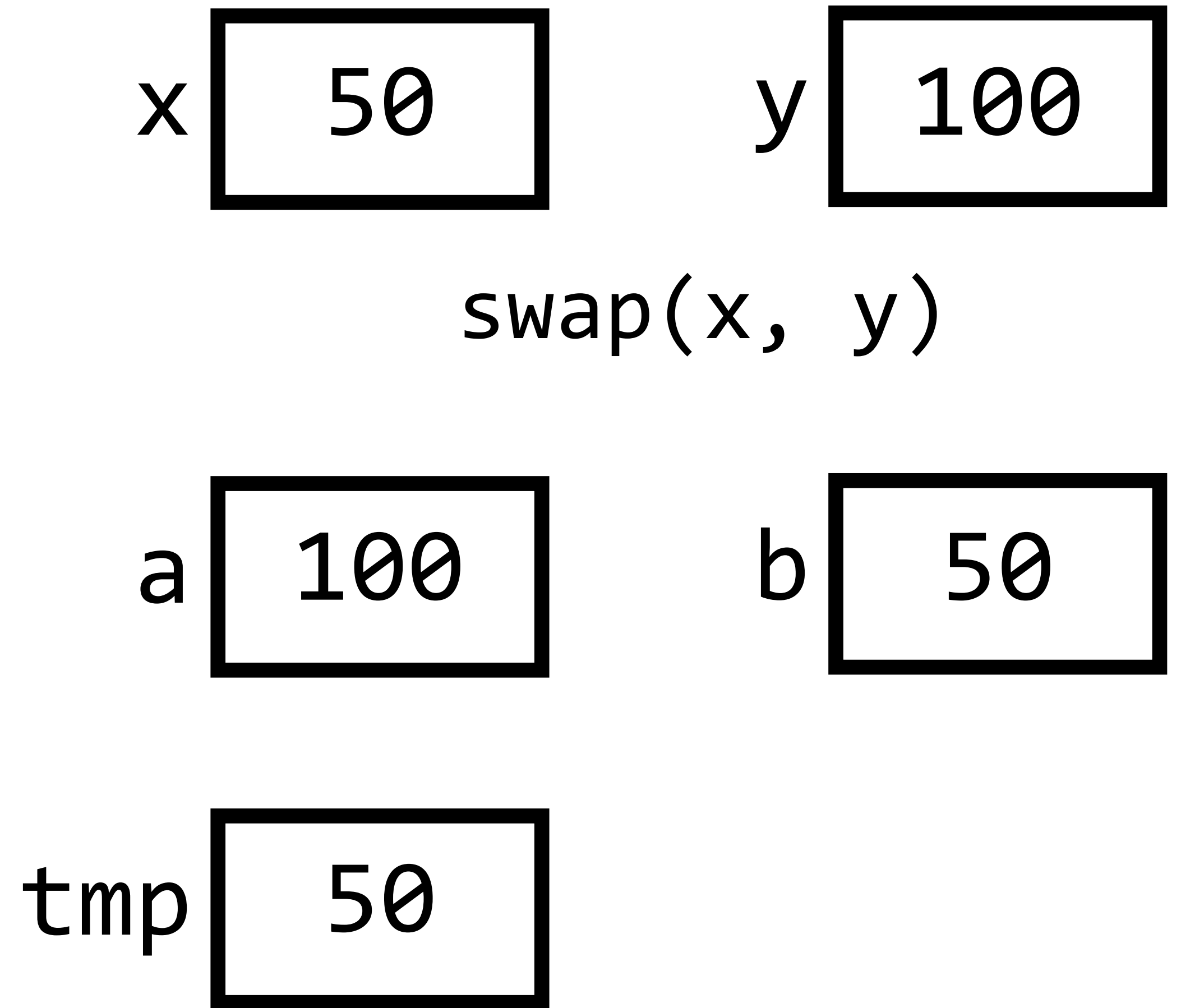
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```



x 50

y 100

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

x

50

y

100

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

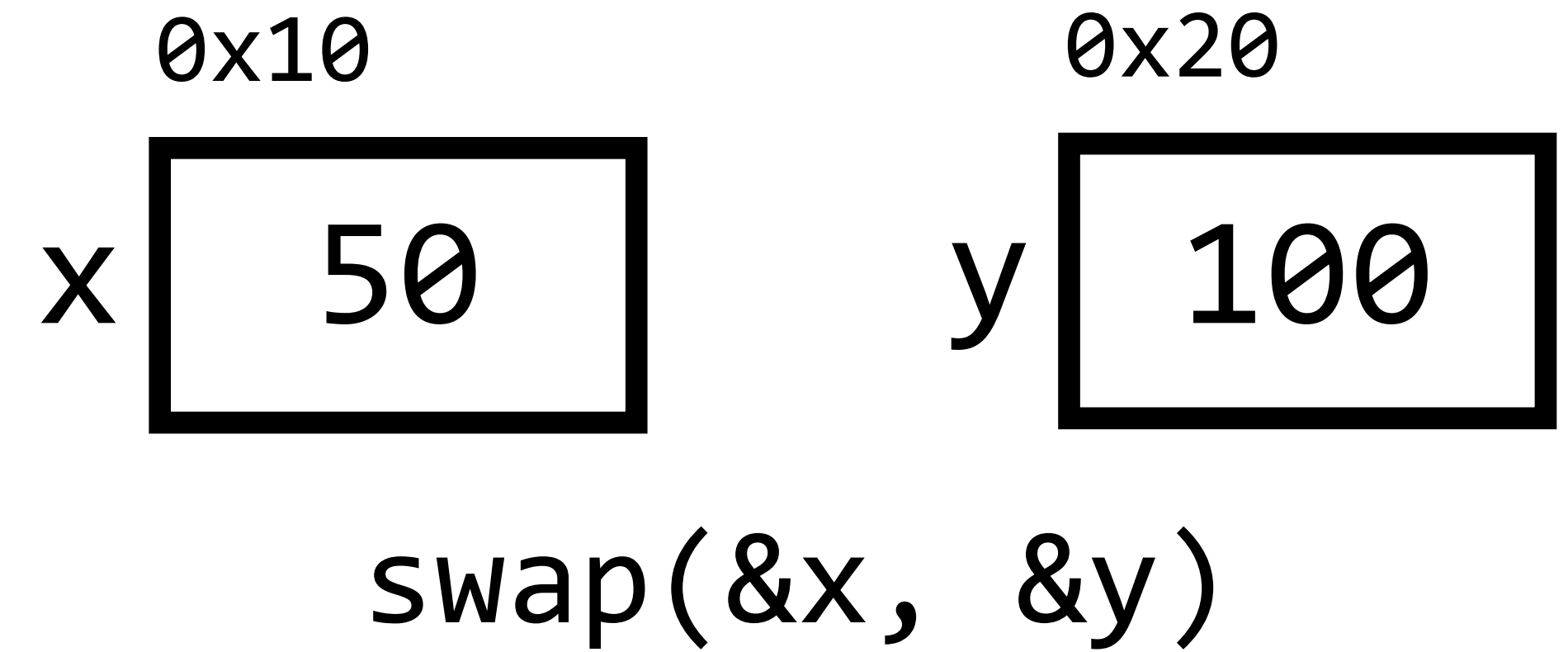
```
swap(&x, &y)
```



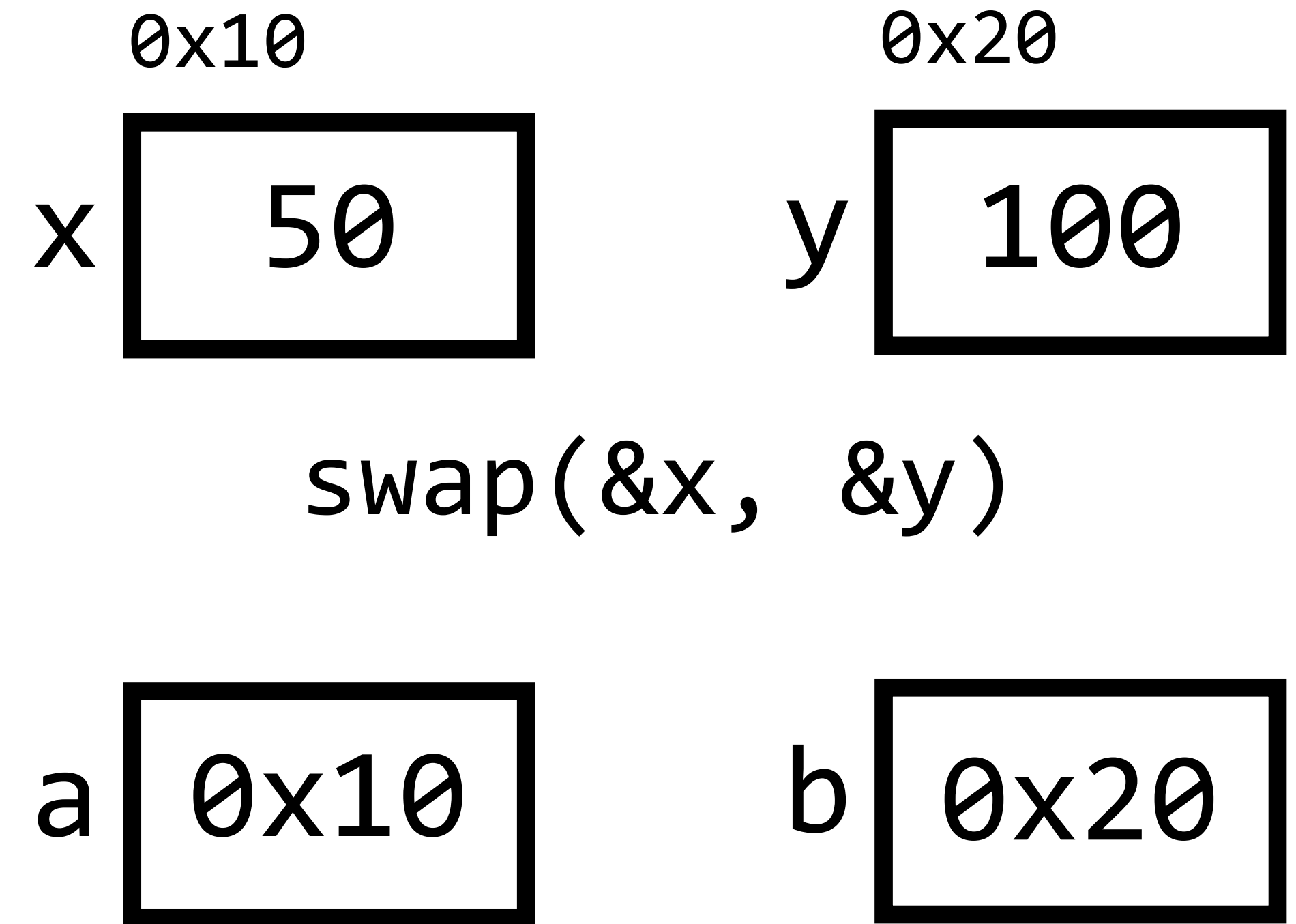

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

swap(&x, &y)

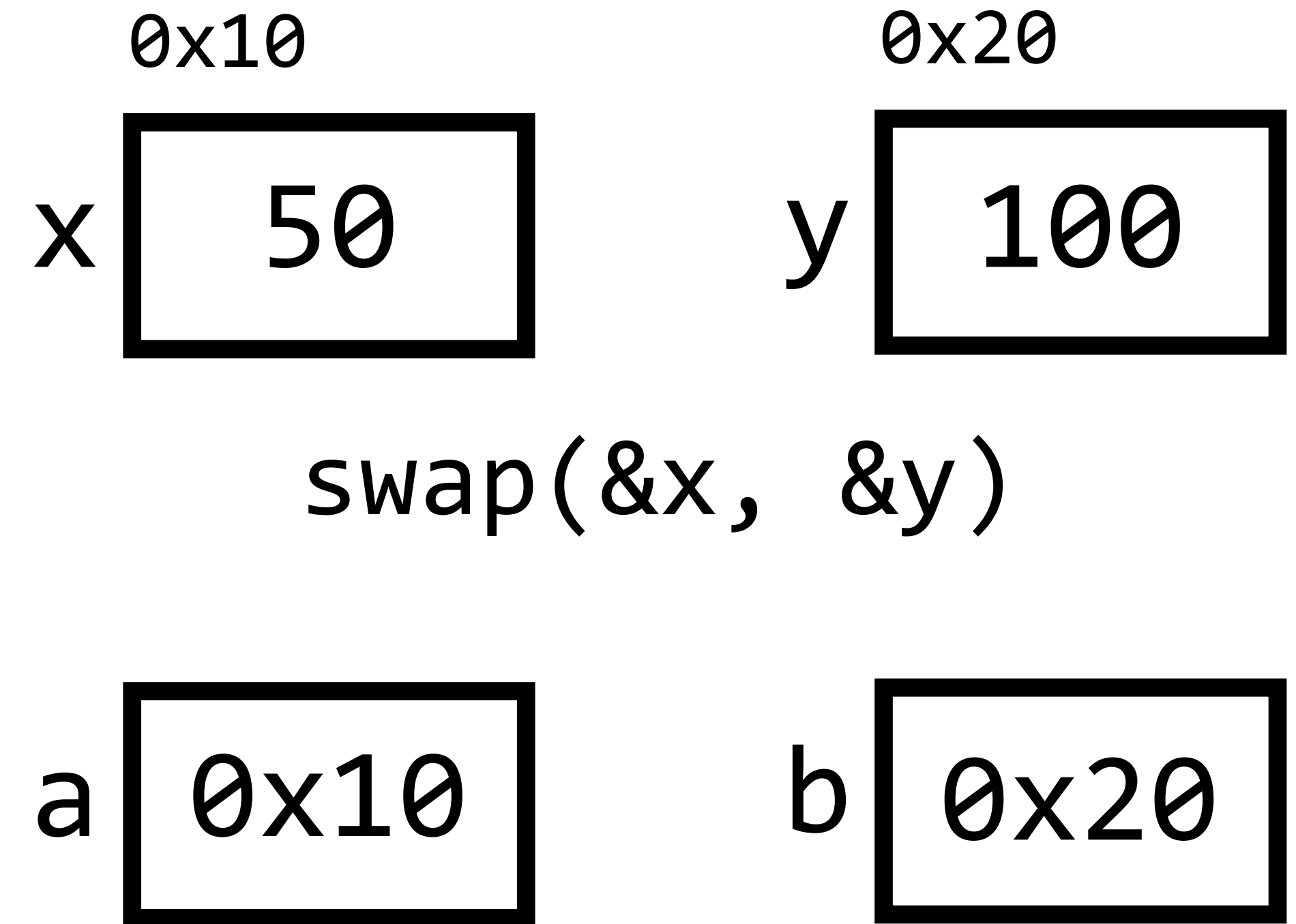
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



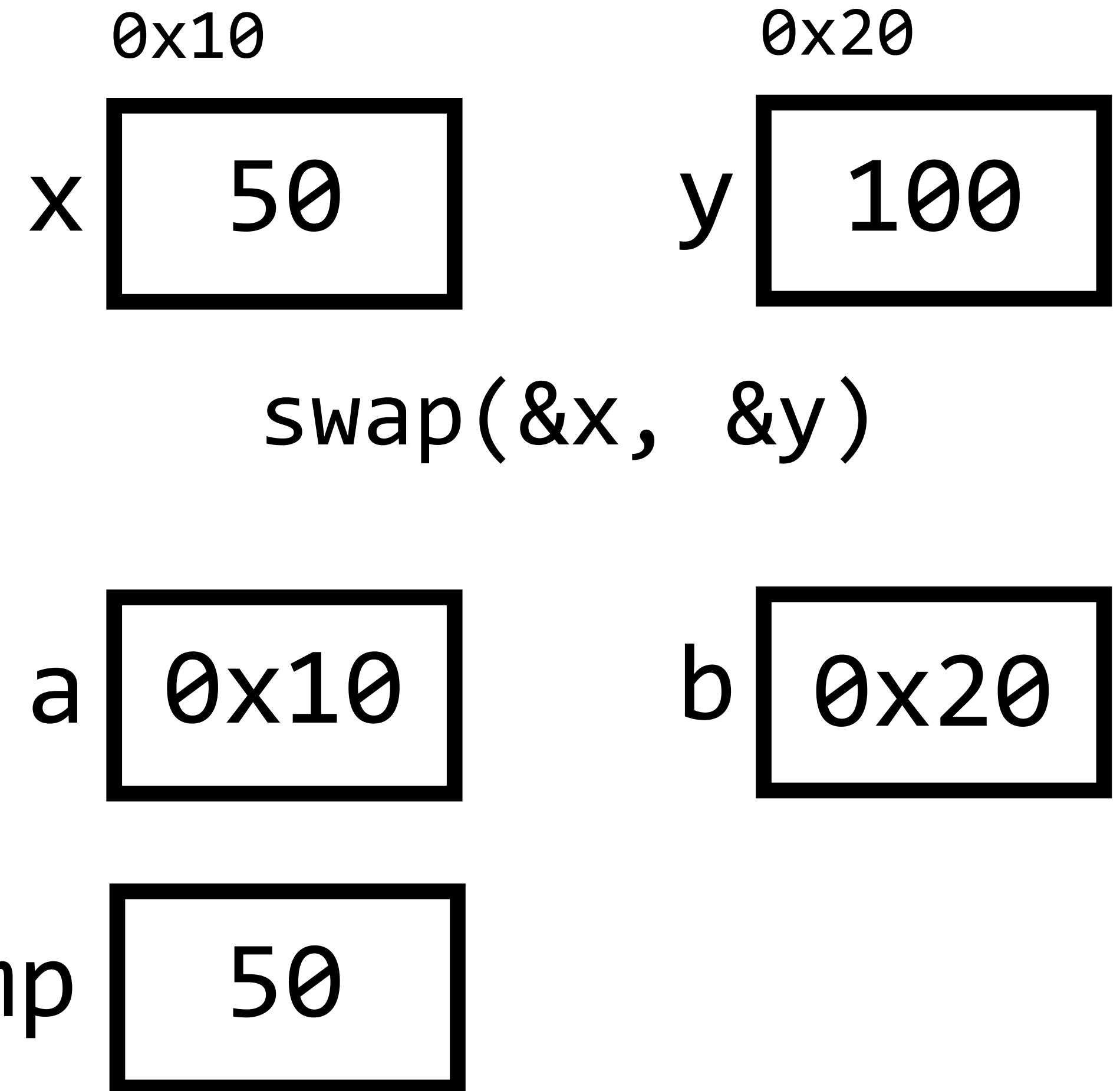
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



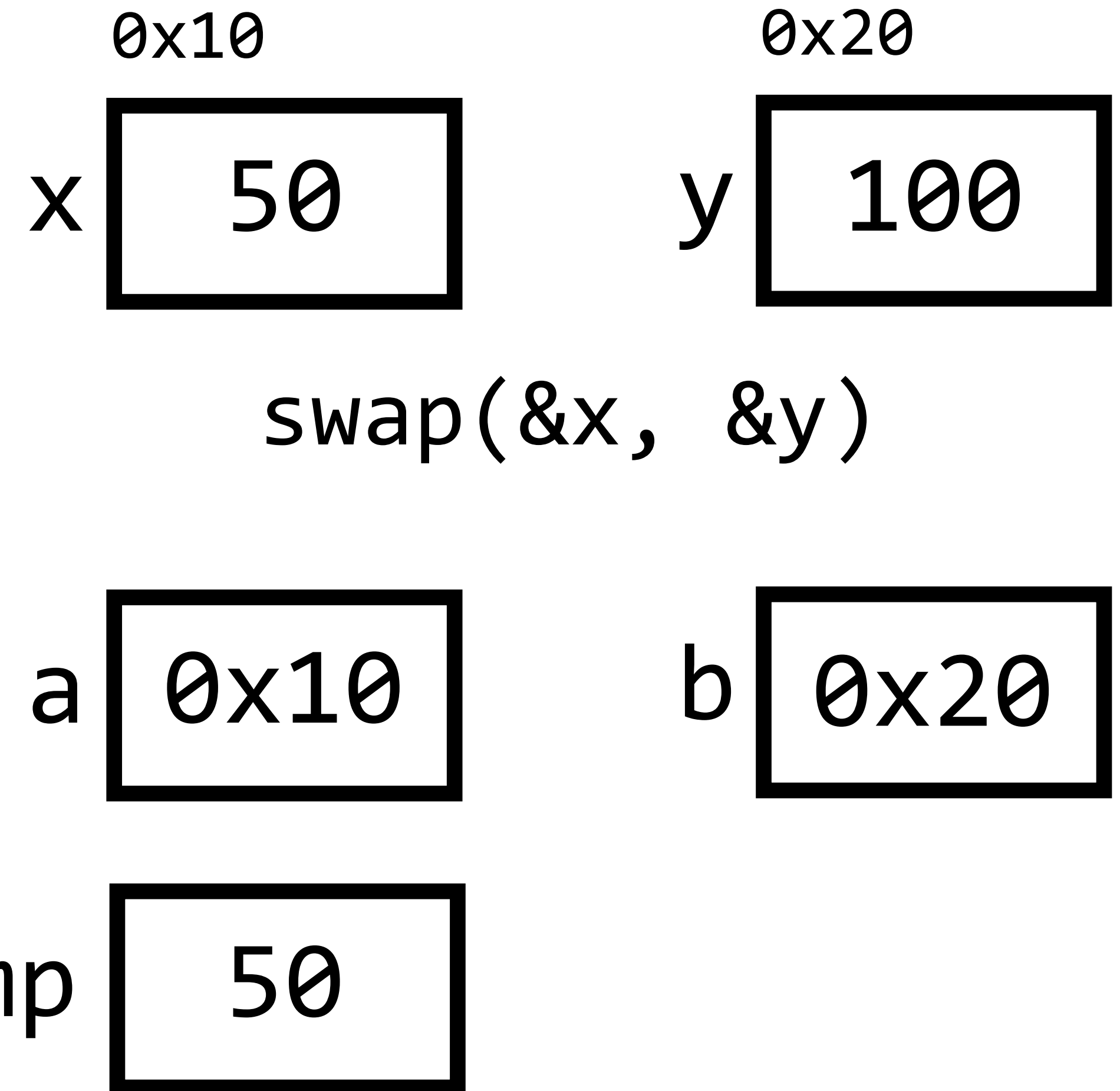
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



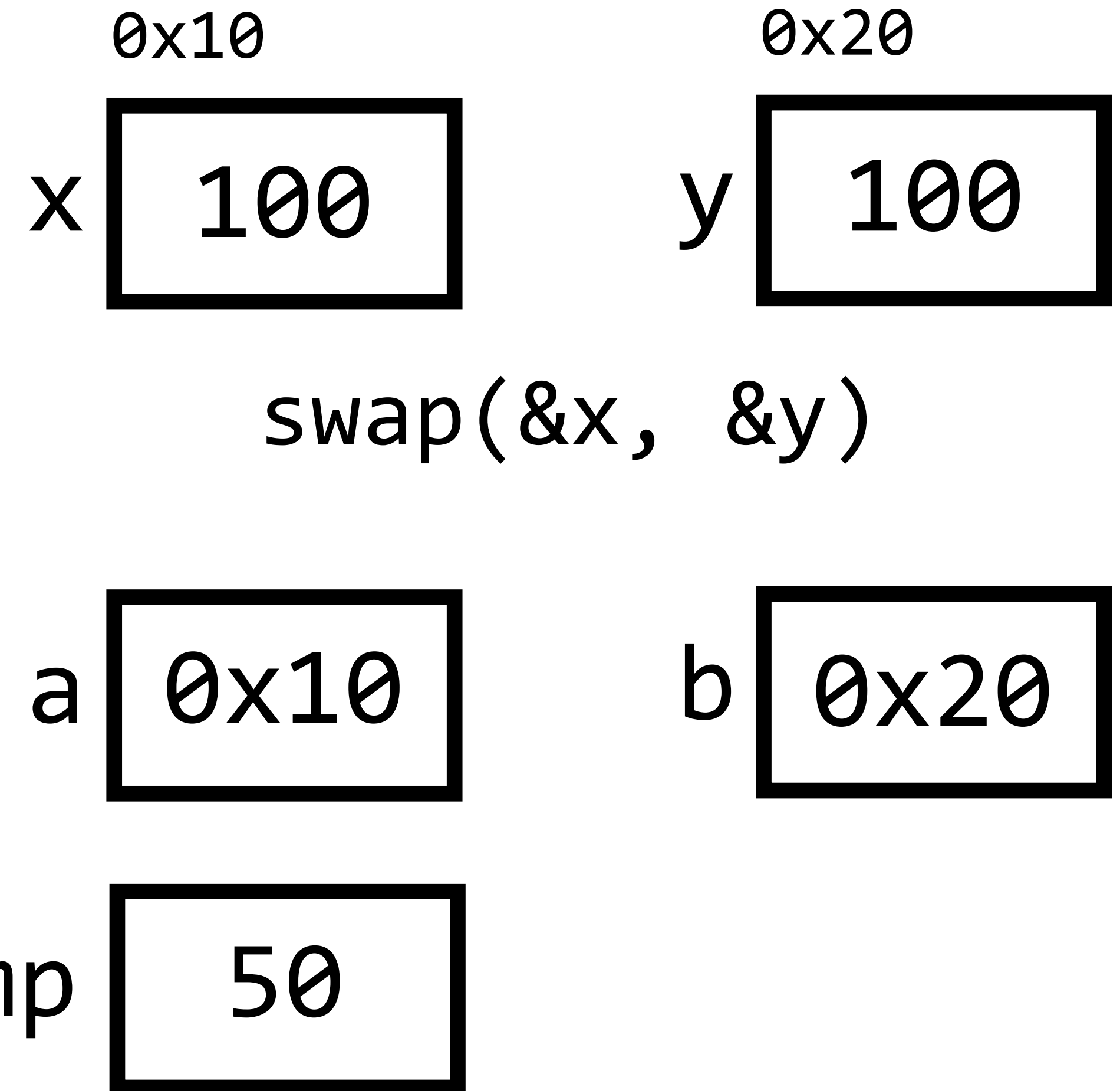
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



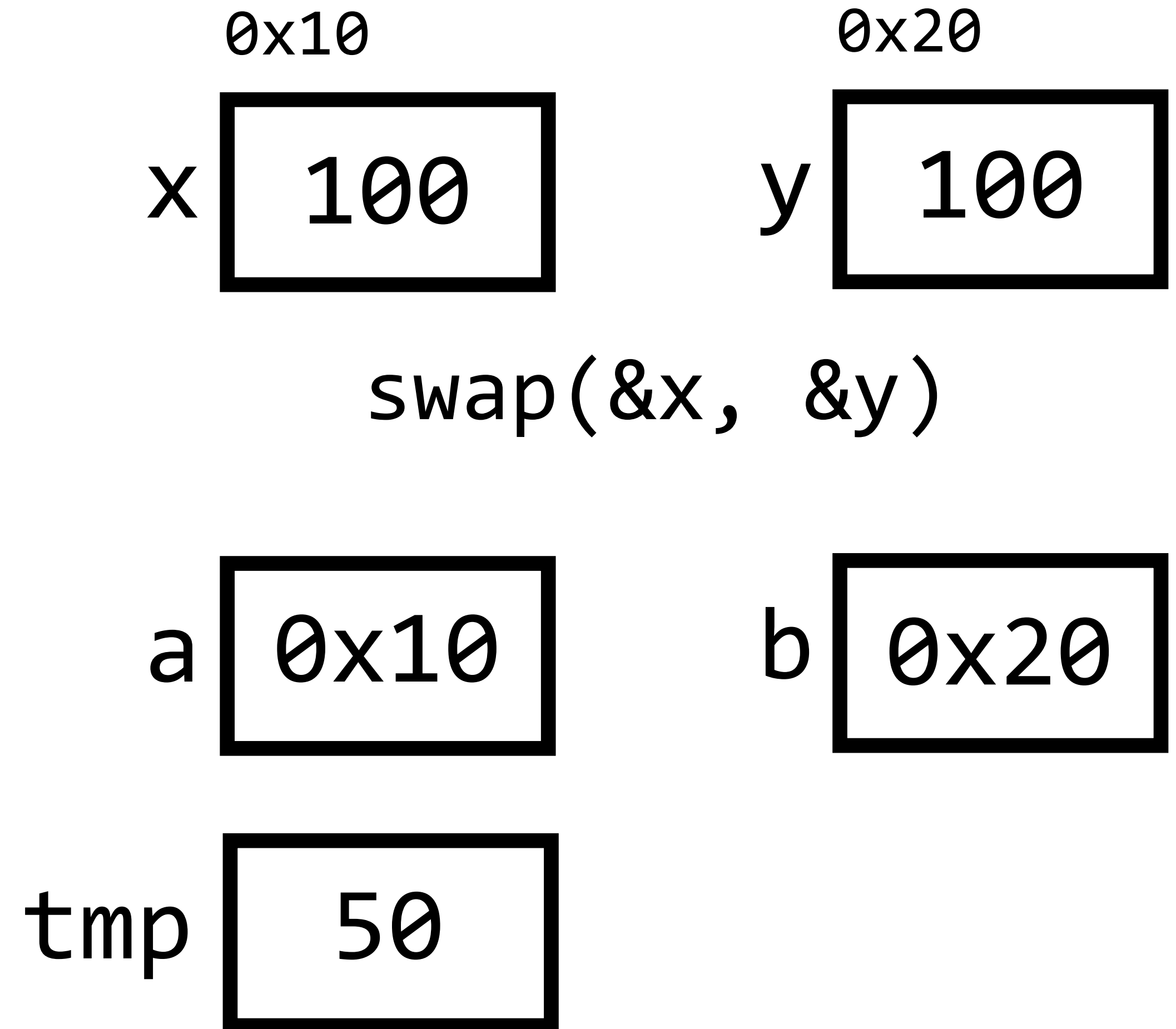
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



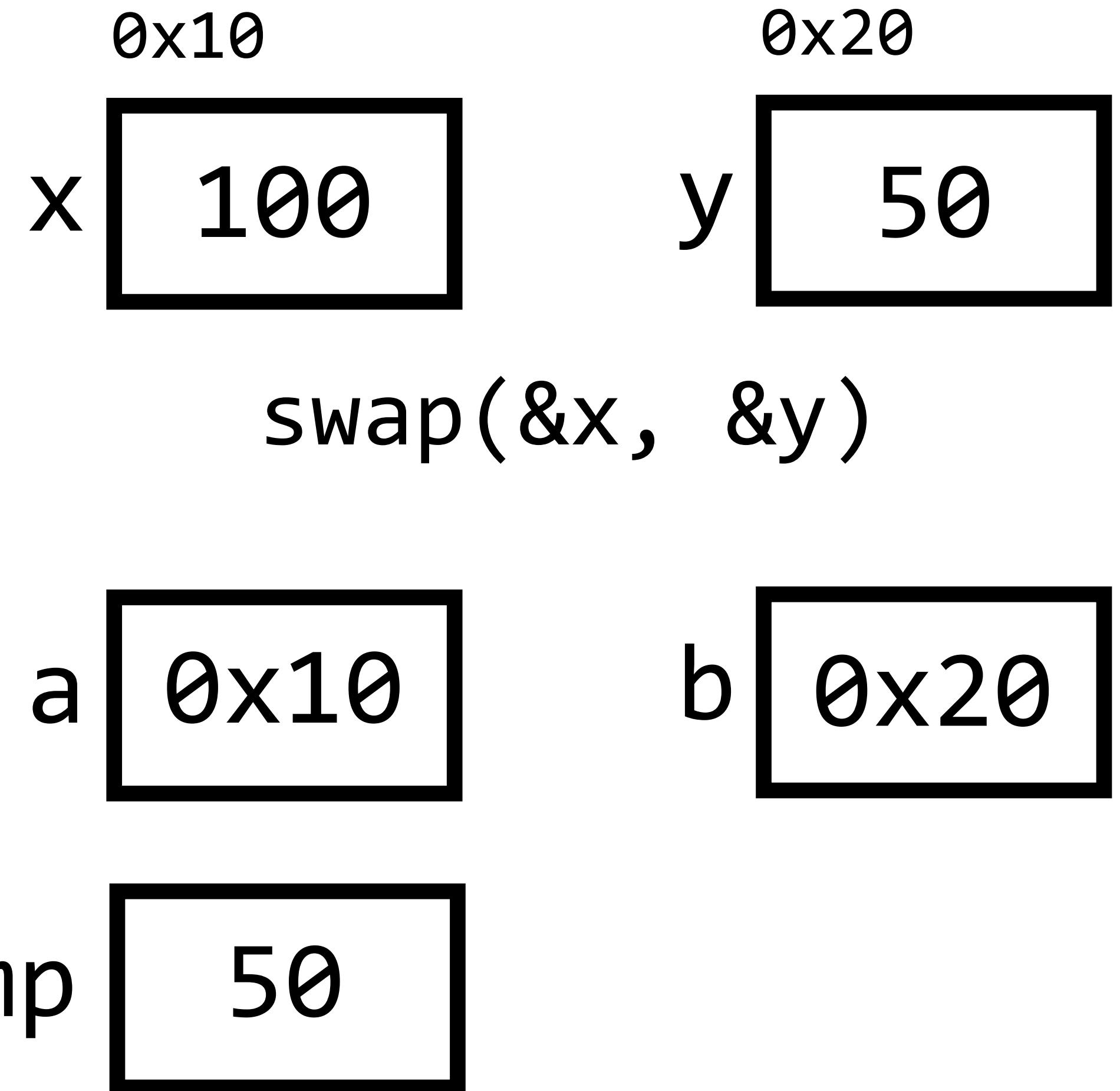
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



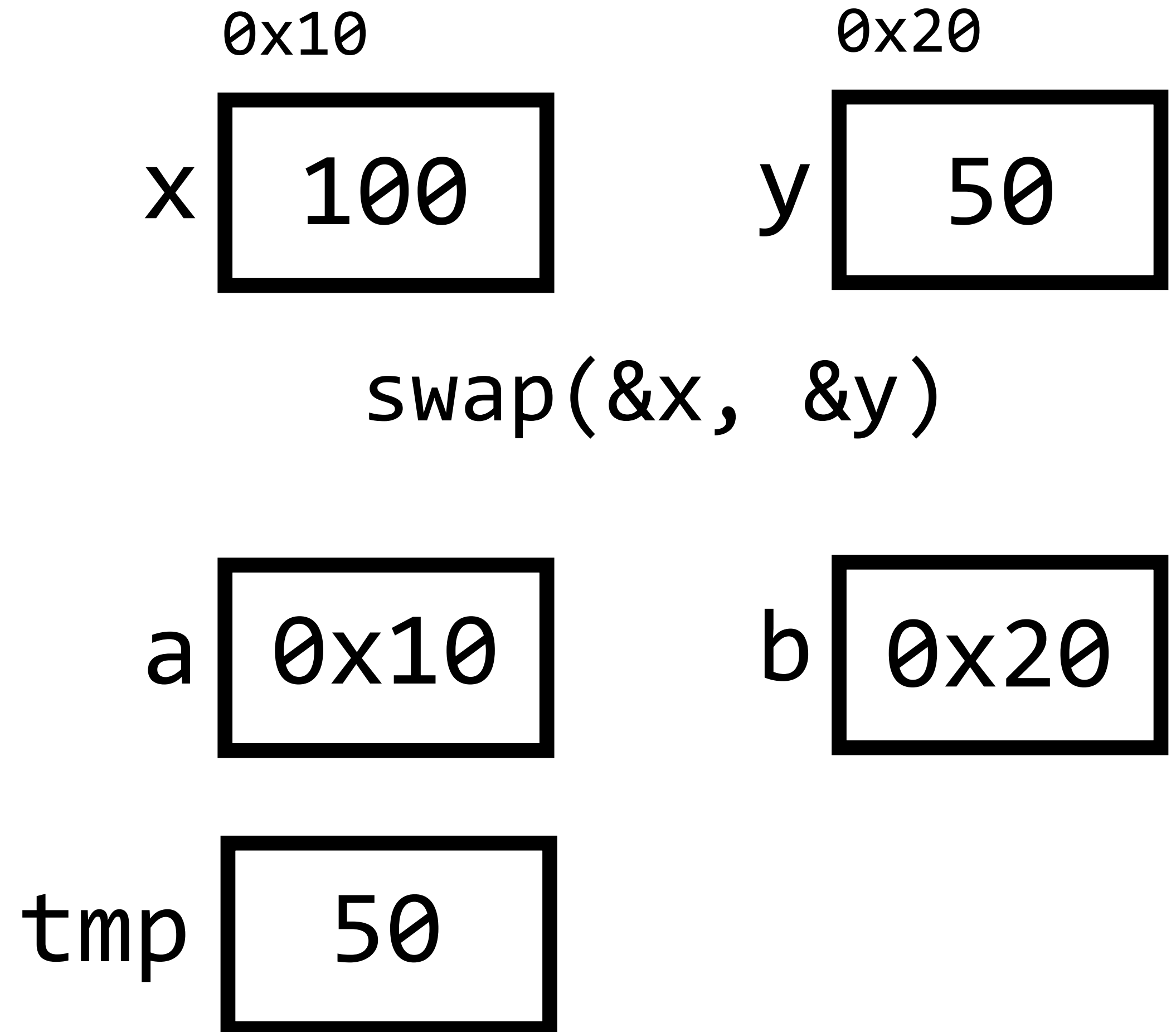
```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



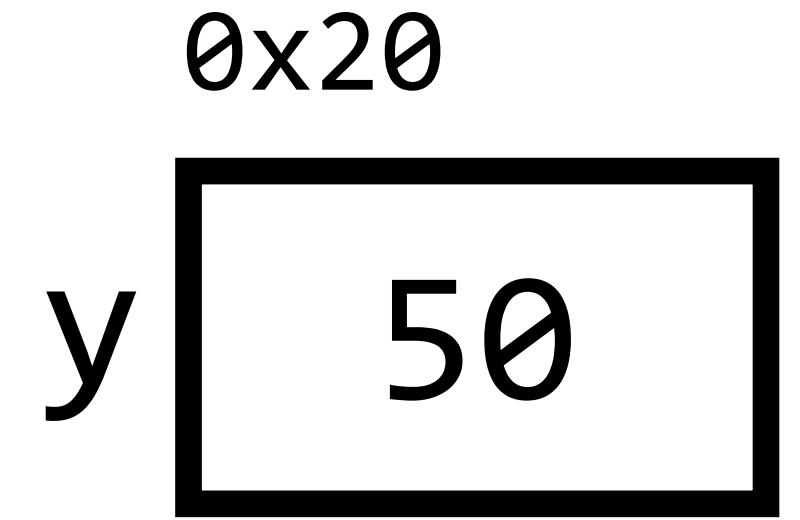
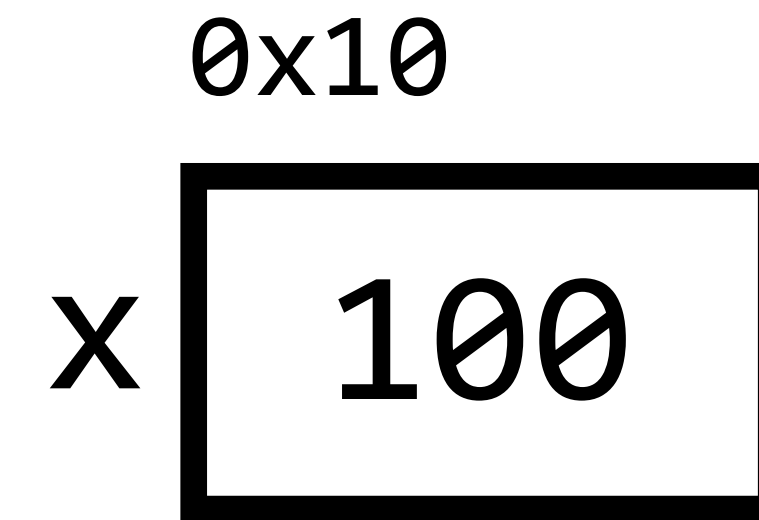

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```



PART TWO

File I/O

File I/O

Open a File

```
FILE *f = fopen("filename.txt", "r");
```

Open a File

```
FILE *f = fopen("filename.txt", "r");
```



Pointer
to file

Open a File

```
FILE *f = fopen("filename.txt", "r");
```

↑
Pointer
to file

↑
Function to
open file

Open a File

```
FILE *f = fopen("filename.txt", "r");
```

↑
Pointer
to file

↑
Function to
open file

↑
Name of file
to open

Open a File

```
FILE *f = fopen("filename.txt", "r");
```

↑
Pointer
to file

↑
Function to
open file

↑
Name of file
to open

↑
File Mode
(r, w, a)

Reading from a File

```
fread(ptr, size, blocks, fp)
```

Reading from a File

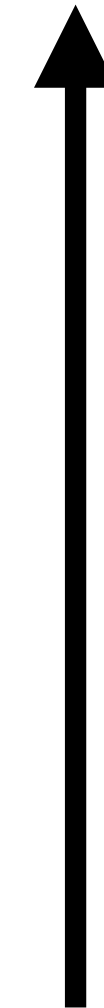
```
fread(ptr, size, blocks, fp)
```



Pointer to file to read from

Reading from a File

```
fread(ptr, size, blocks, fp)
```



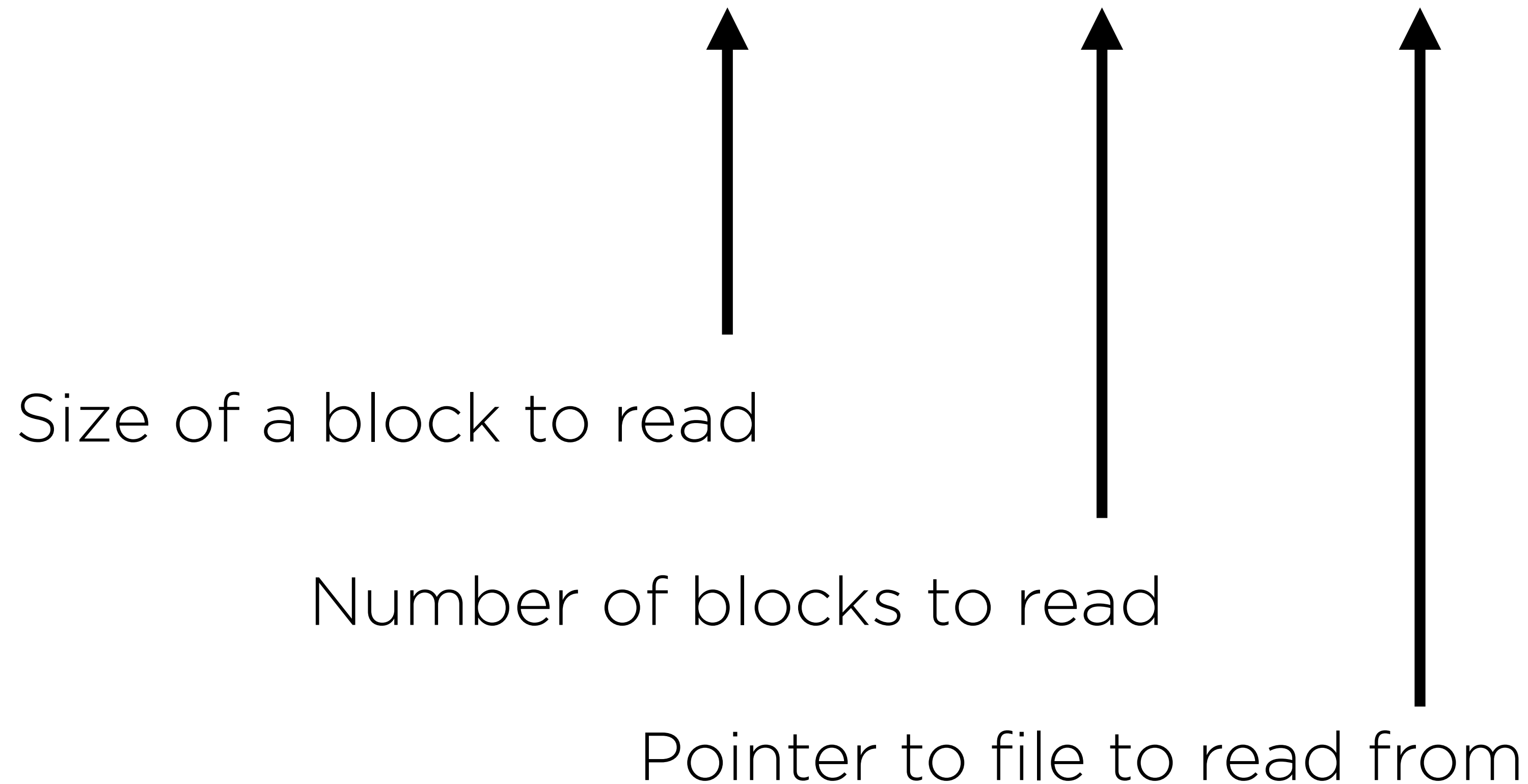
Number of blocks to read



Pointer to file to read from

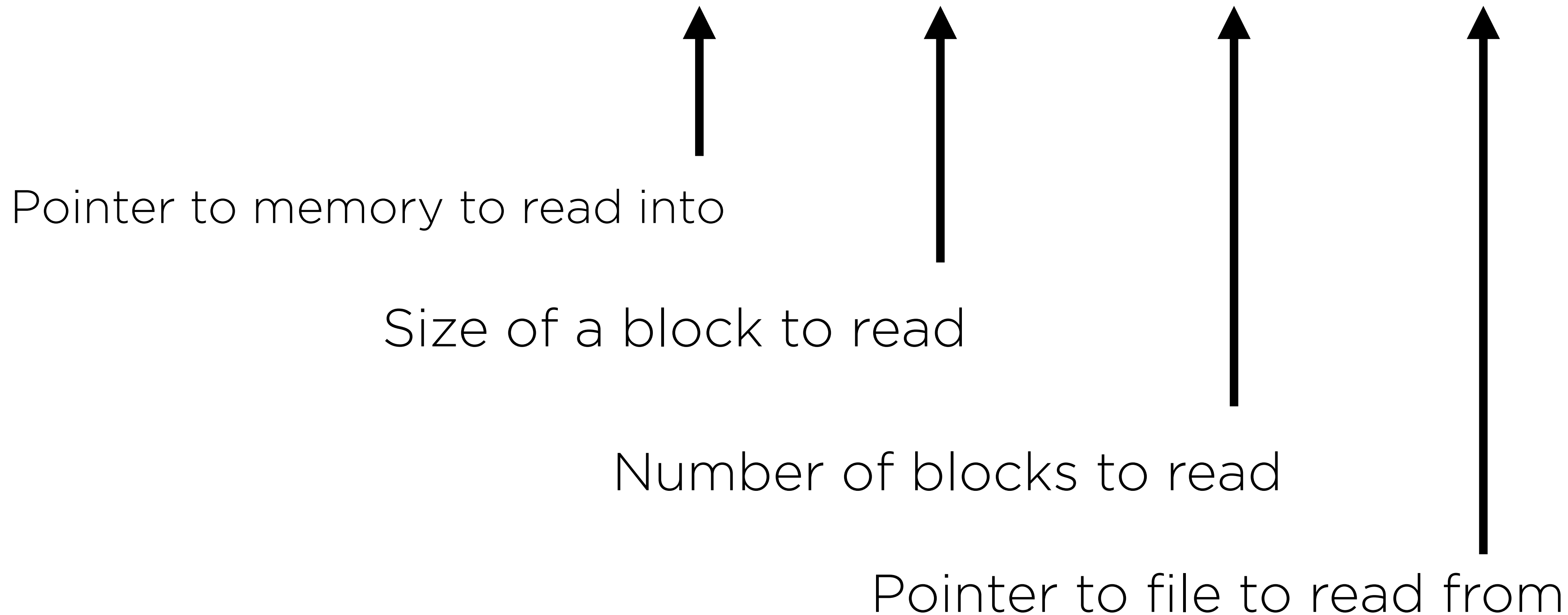
Reading from a File

`fread(ptr, size, blocks, fp)`



Reading from a File

`fread(ptr, size, blocks, fp)`



Writing to a File

```
fwrite(ptr, size, blocks, fp)
```


Writing to a File

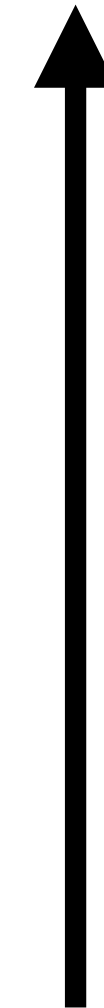
```
fwrite(ptr, size, blocks, fp)
```



Pointer to file to write to

Writing to a File

```
fwrite(ptr, size, blocks, fp)
```



Number of blocks to write



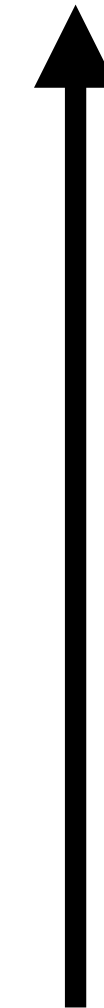
Pointer to file to write to

Writing to a File

```
fwrite(ptr, size, blocks, fp)
```



Size of a block to write



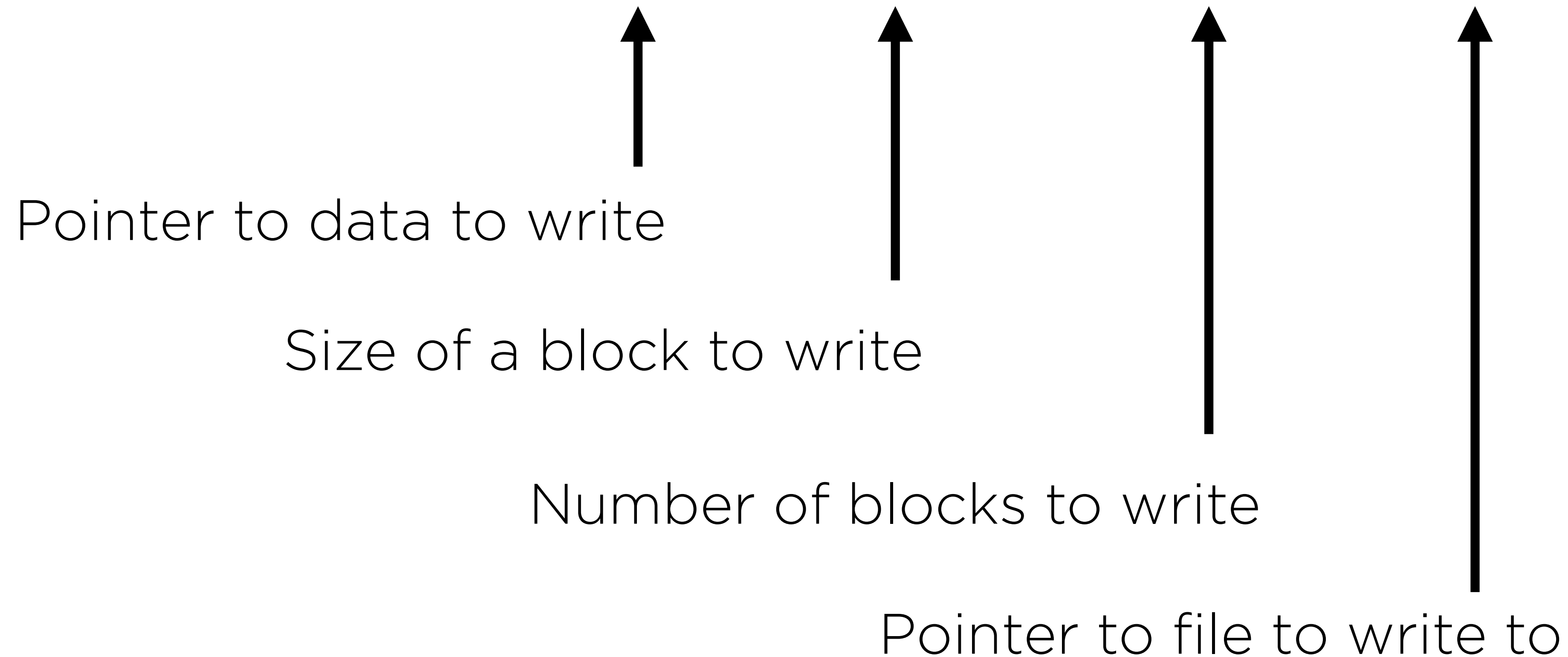
Number of blocks to write



Pointer to file to write to

Writing to a File

```
fwrite(ptr, size, blocks, fp)
```



Closing a File

```
fclose(fp)
```

Closing a File

`fclose(fp)`



Pointer to file to close

Other File/IO Functions

- fgetc
- fputc
- fprintf
- ...

Exercise

Write a program `copy.c` that copies a text file.

```
$ ./copy file1 file2
```


Exercise

Write a program **pdf.c** that checks if a file is a PDF.

Every PDF begins with the first five bytes: **0x25 0x50 0x44 0x46 0x2d**

```
$ ./pdf file.pdf
```

YES

```
$ ./pdf picture.jpg
```

NO

PART THREE

Dynamic Memory

Dynamic Memory

Dynamic Memory

- Memory that lives on the heap, outside of the stack
- Memory that can be requested at runtime

```
int *x = malloc(sizeof(int));
```

```
int *x = malloc(sizeof(int));
```



Pointer to data
on the heap

```
int *x = malloc(sizeof(int));
```

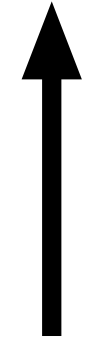


Pointer to data
on the heap

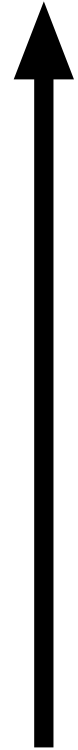


Request memory
from the heap

```
int *x = malloc(sizeof(int));
```



Pointer to data
on the heap



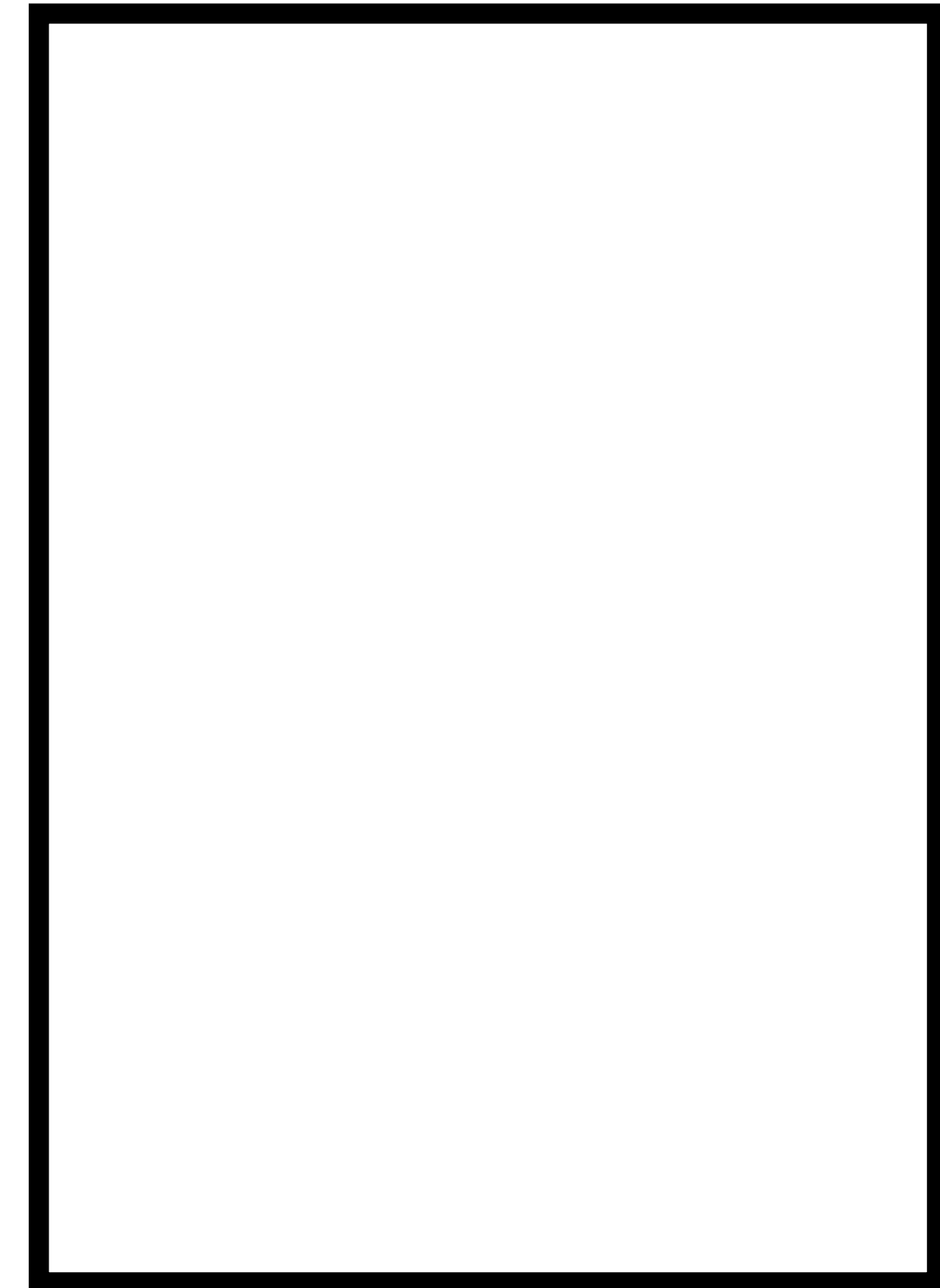
Request memory
from the heap



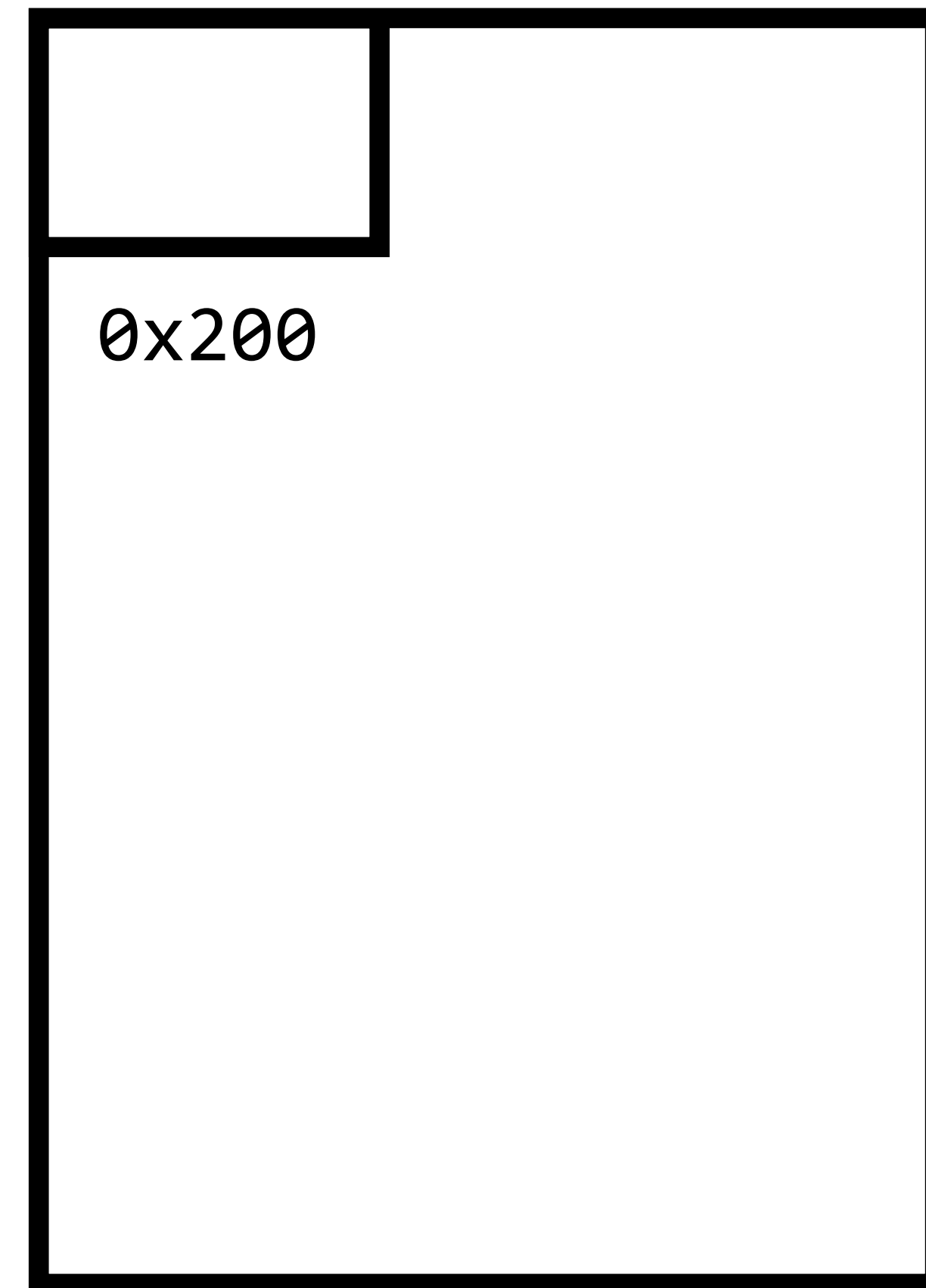
Number of bytes
to allocate


```
int *x = malloc(sizeof(int));
```

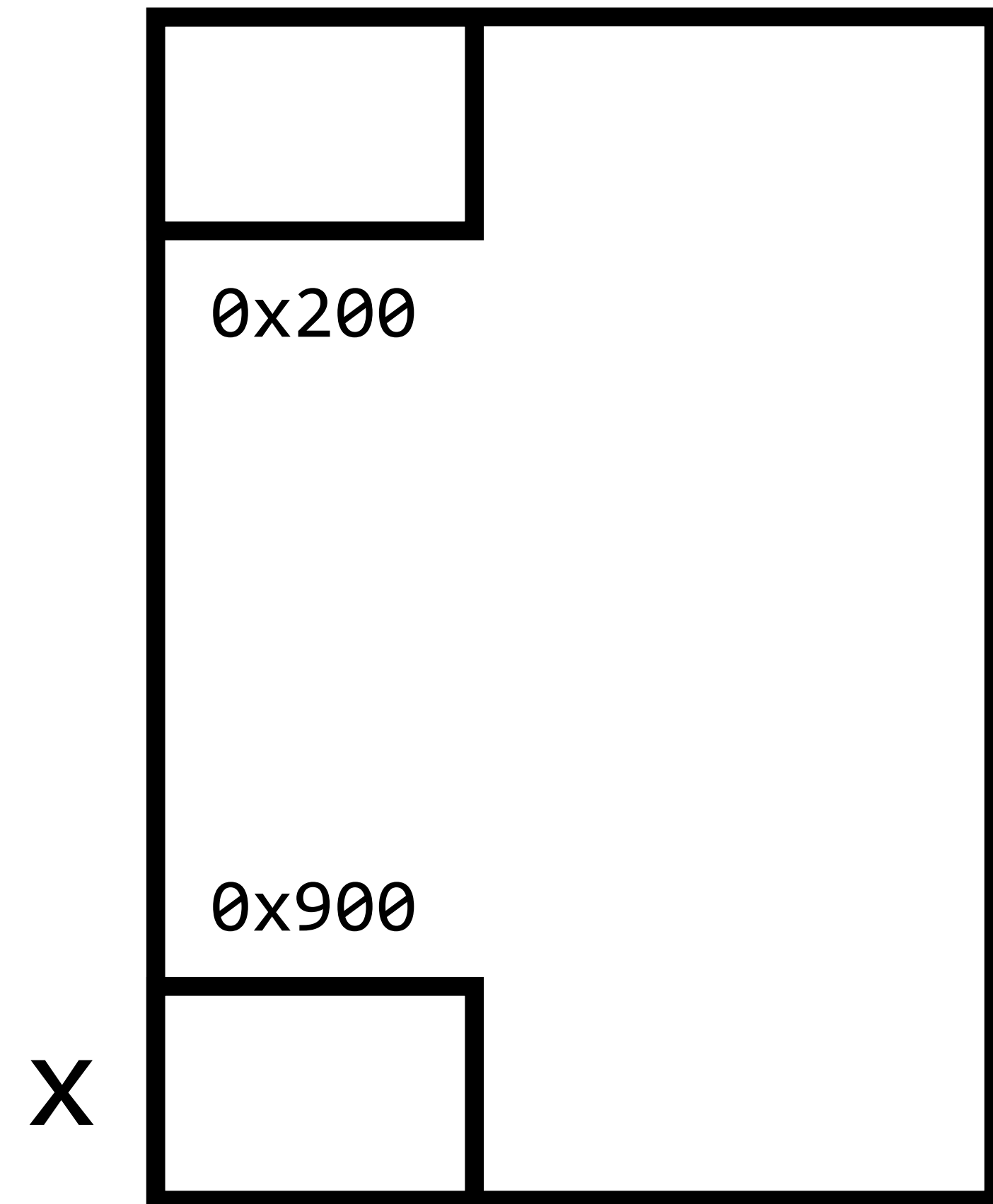
```
int *x = malloc(sizeof(int));
```



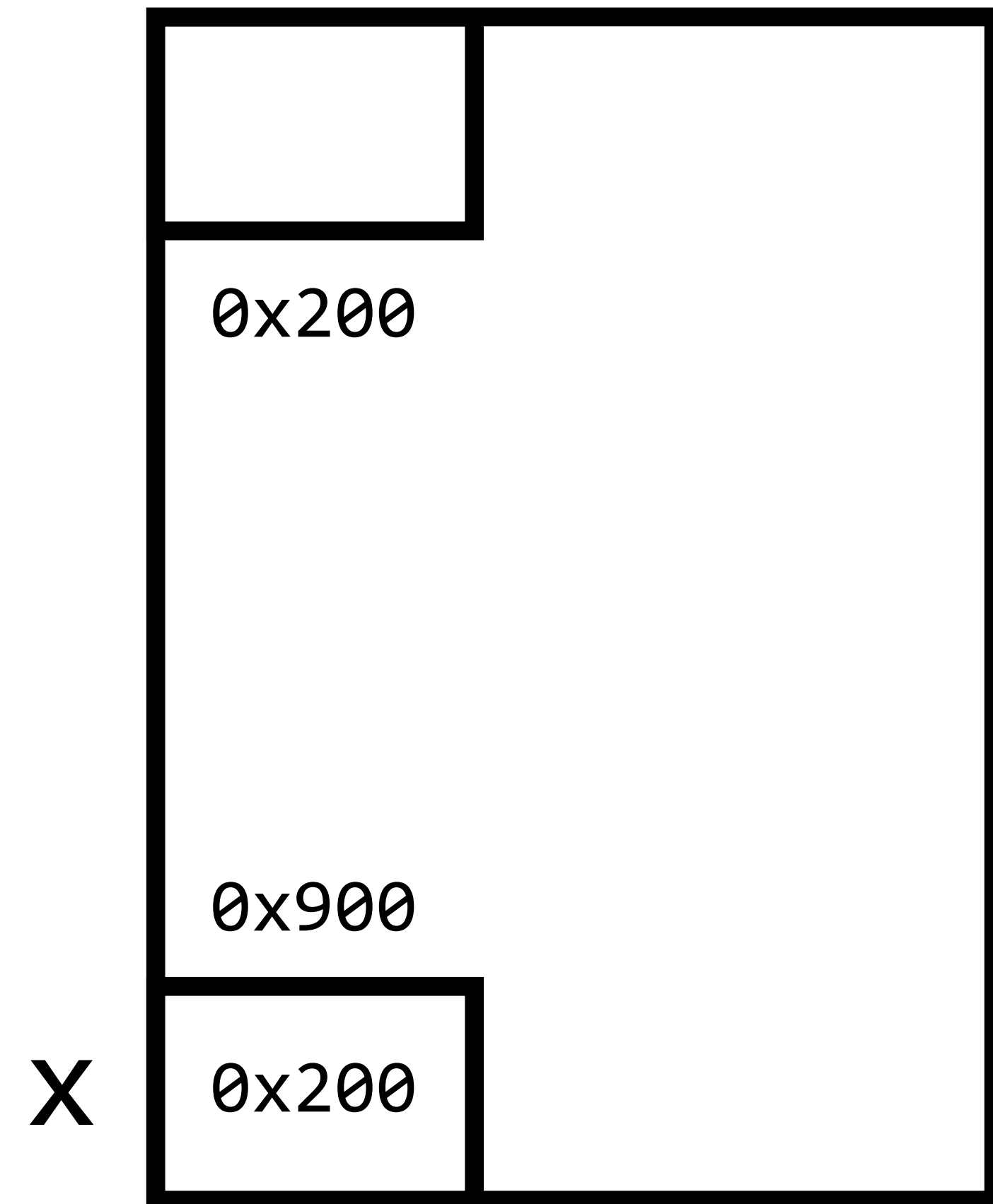
```
int *x = malloc(sizeof(int));
```



```
int *x = malloc(sizeof(int));
```

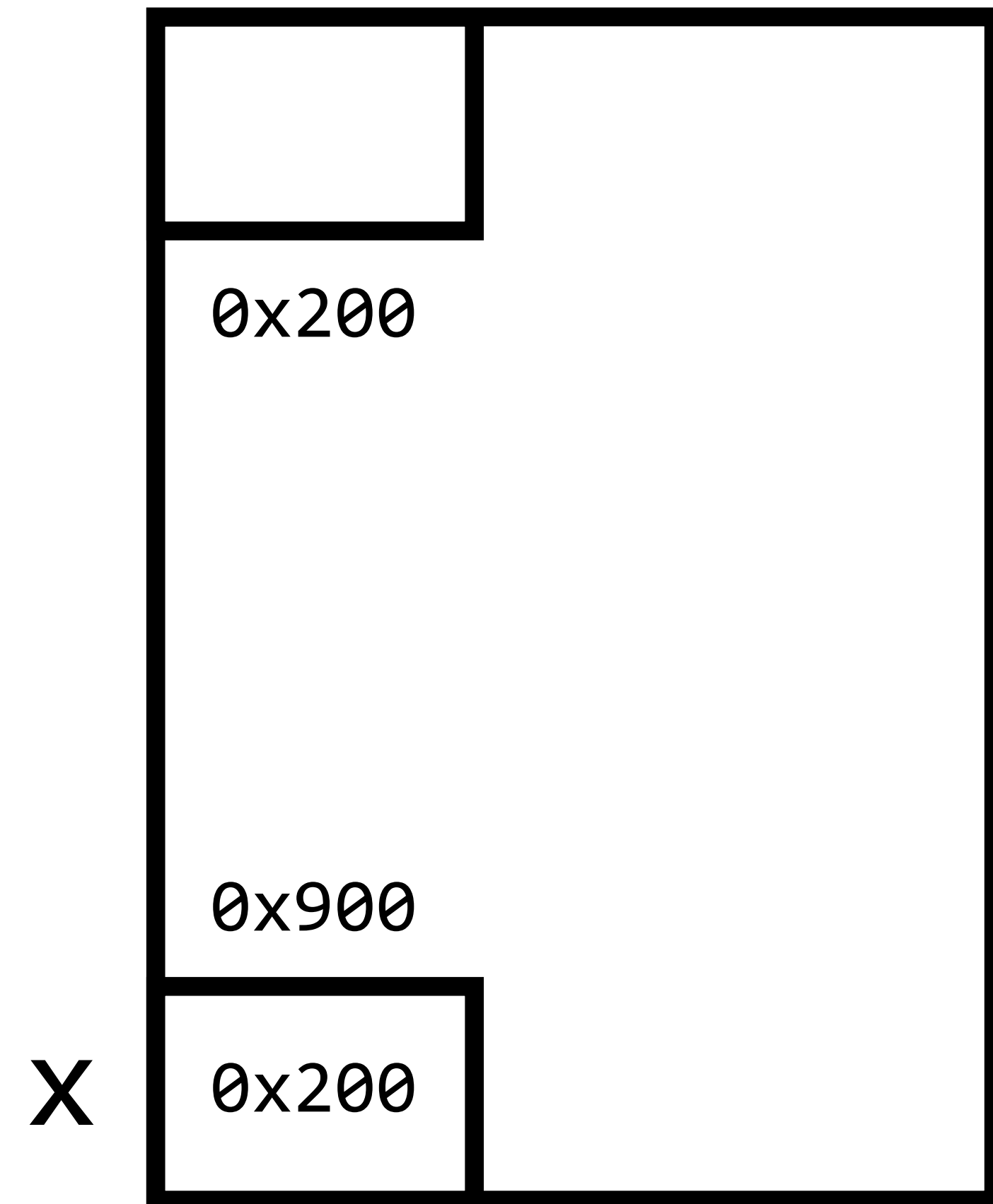


```
int *x = malloc(sizeof(int));
```



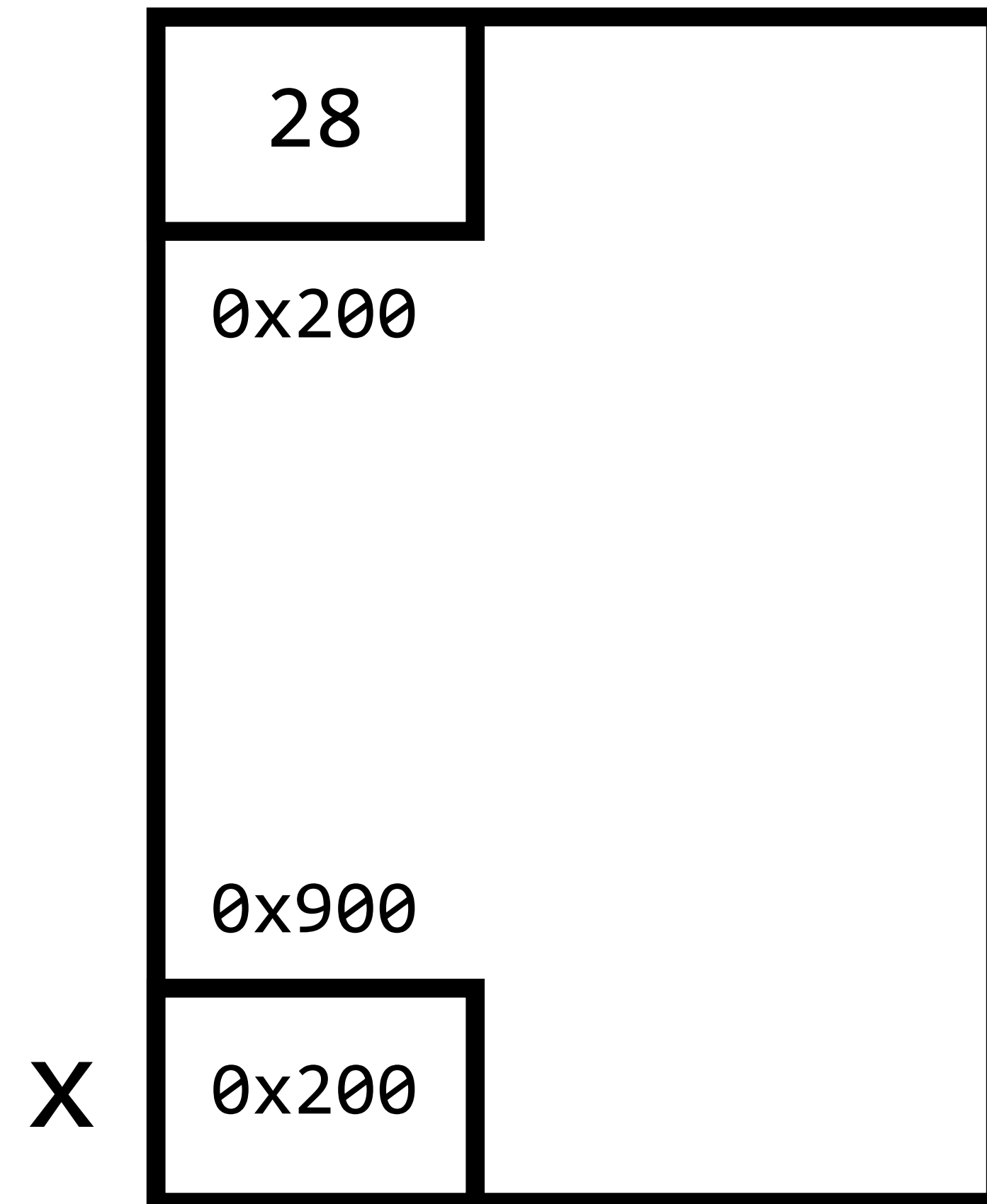
```
int *x = malloc(sizeof(int));
```

```
*x = 28;
```



```
int *x = malloc(sizeof(int));
```

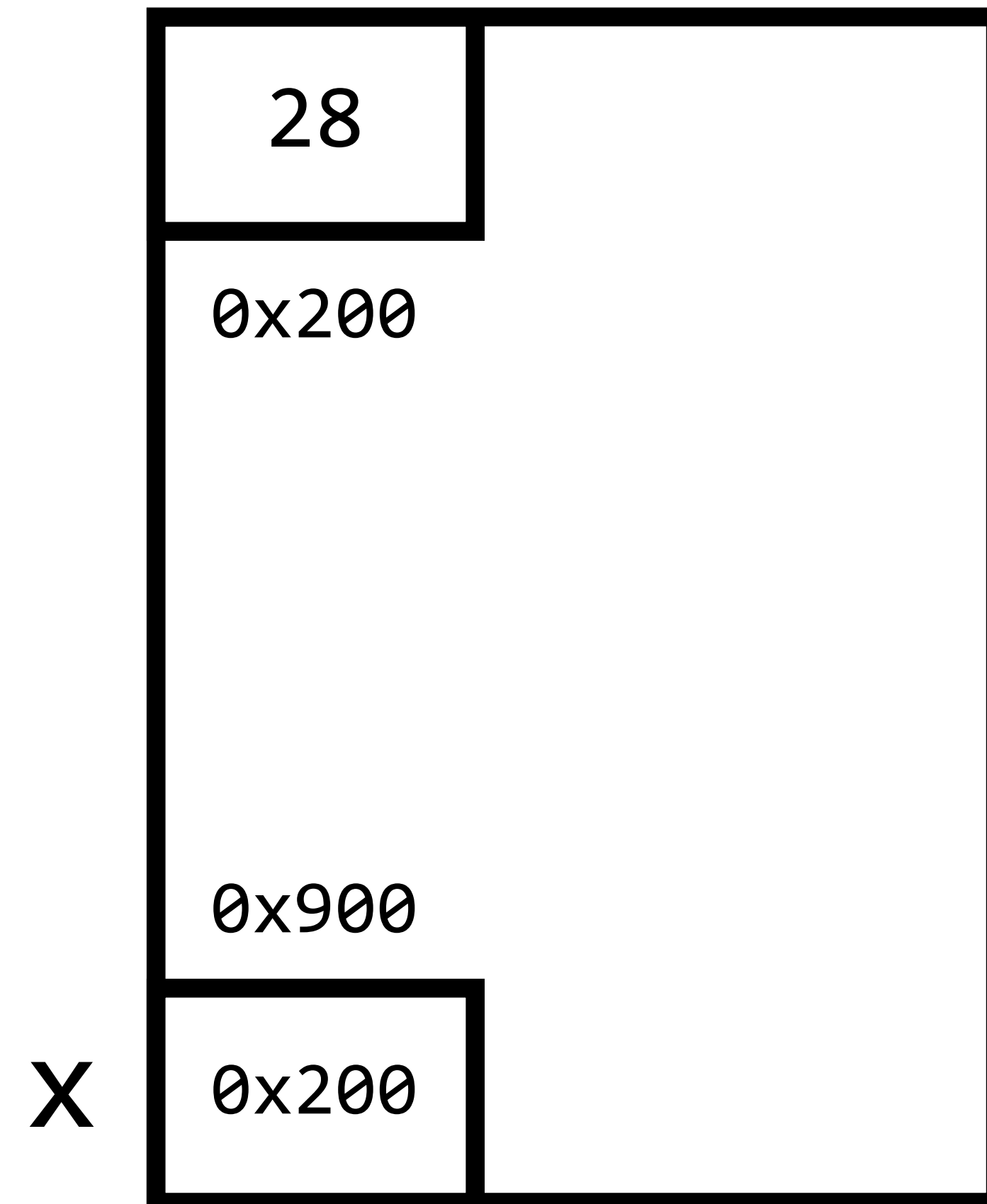
```
*x = 28;
```



```
int *x = malloc(sizeof(int));
```

```
*x = 28;
```

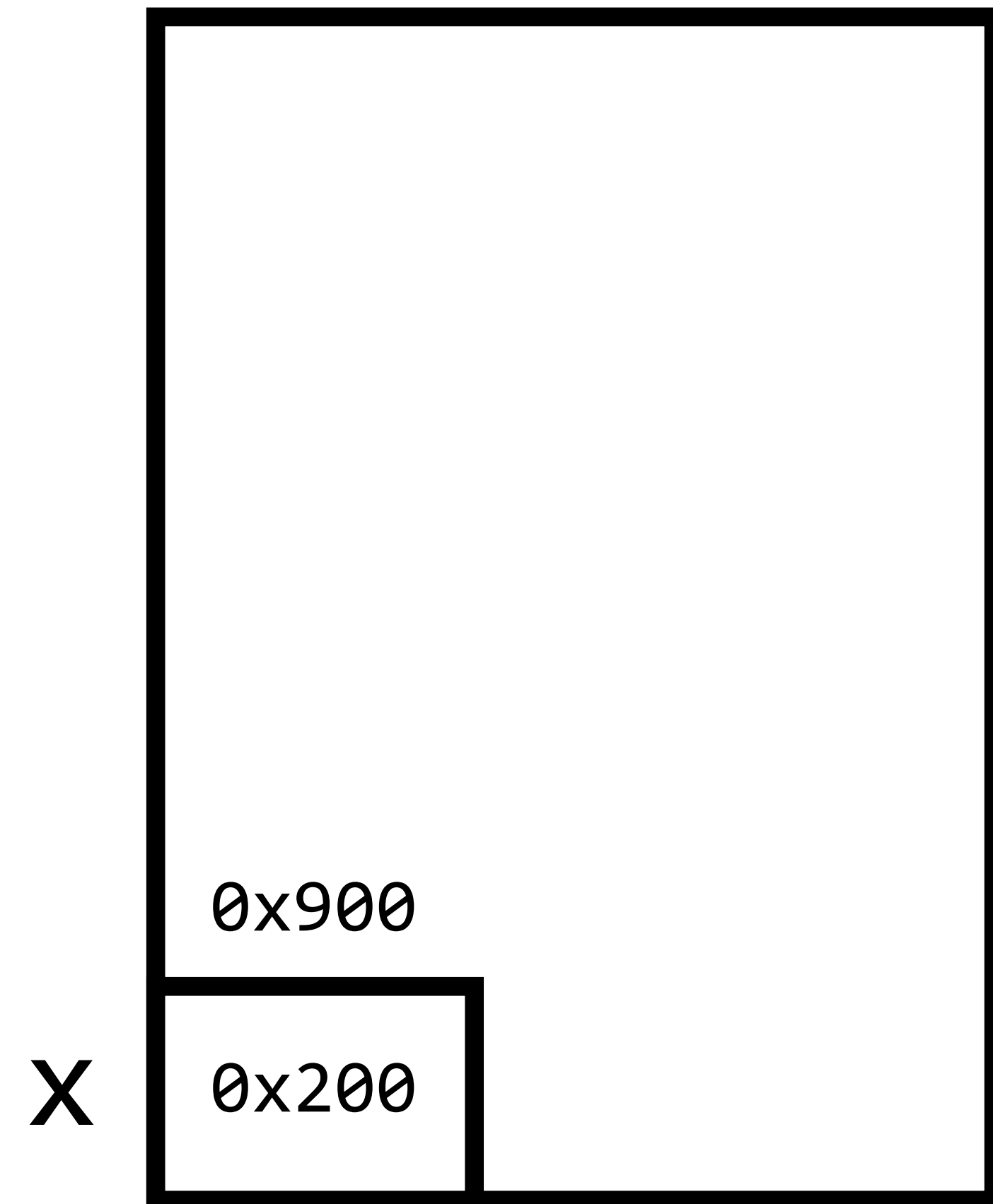
```
free(x);
```




```
int *x = malloc(sizeof(int));
```

```
*x = 28;
```

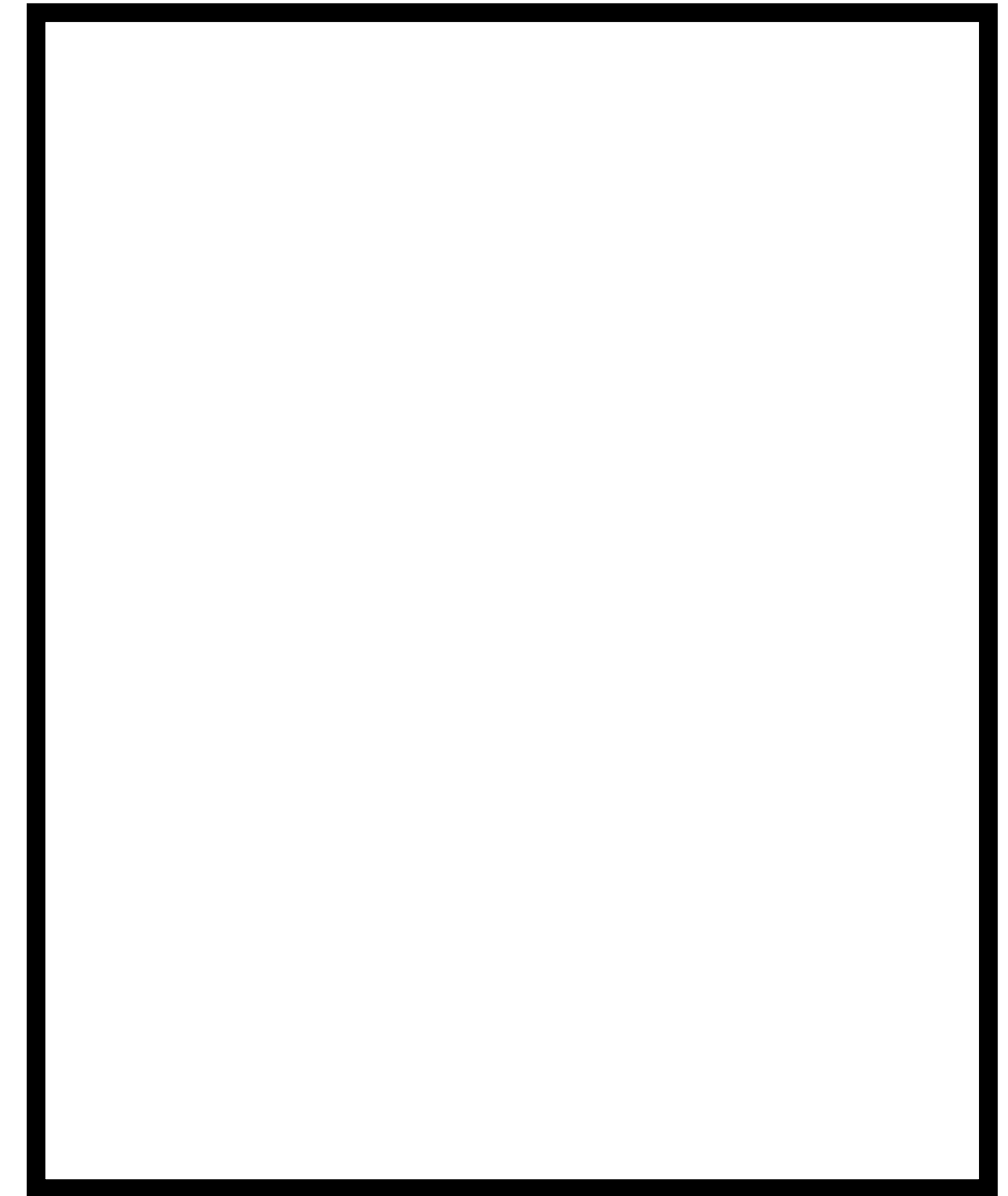
```
free(x);
```



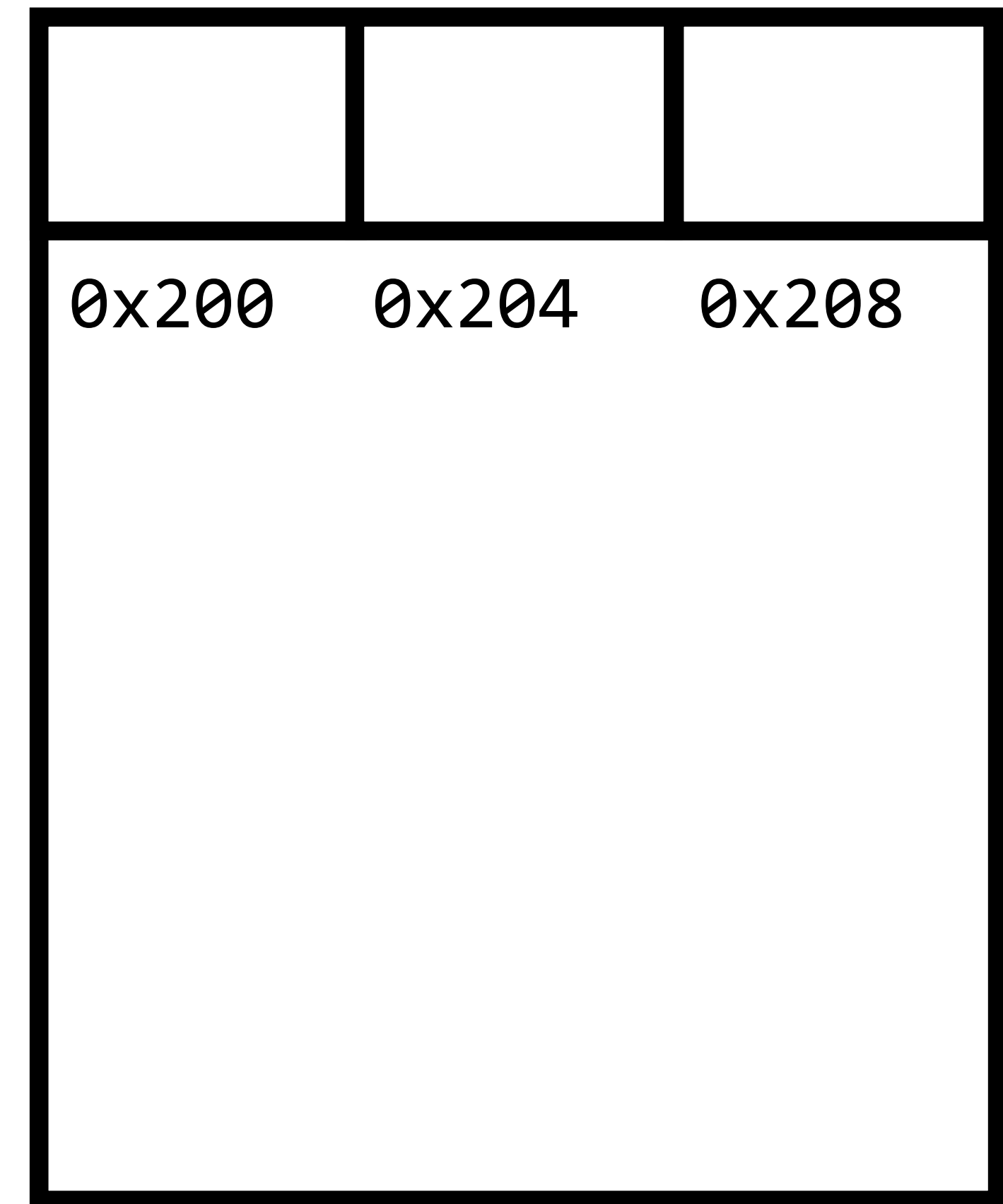
```
int *x = malloc(sizeof(int));
```

```
int *x = malloc(3 * sizeof(int));
```

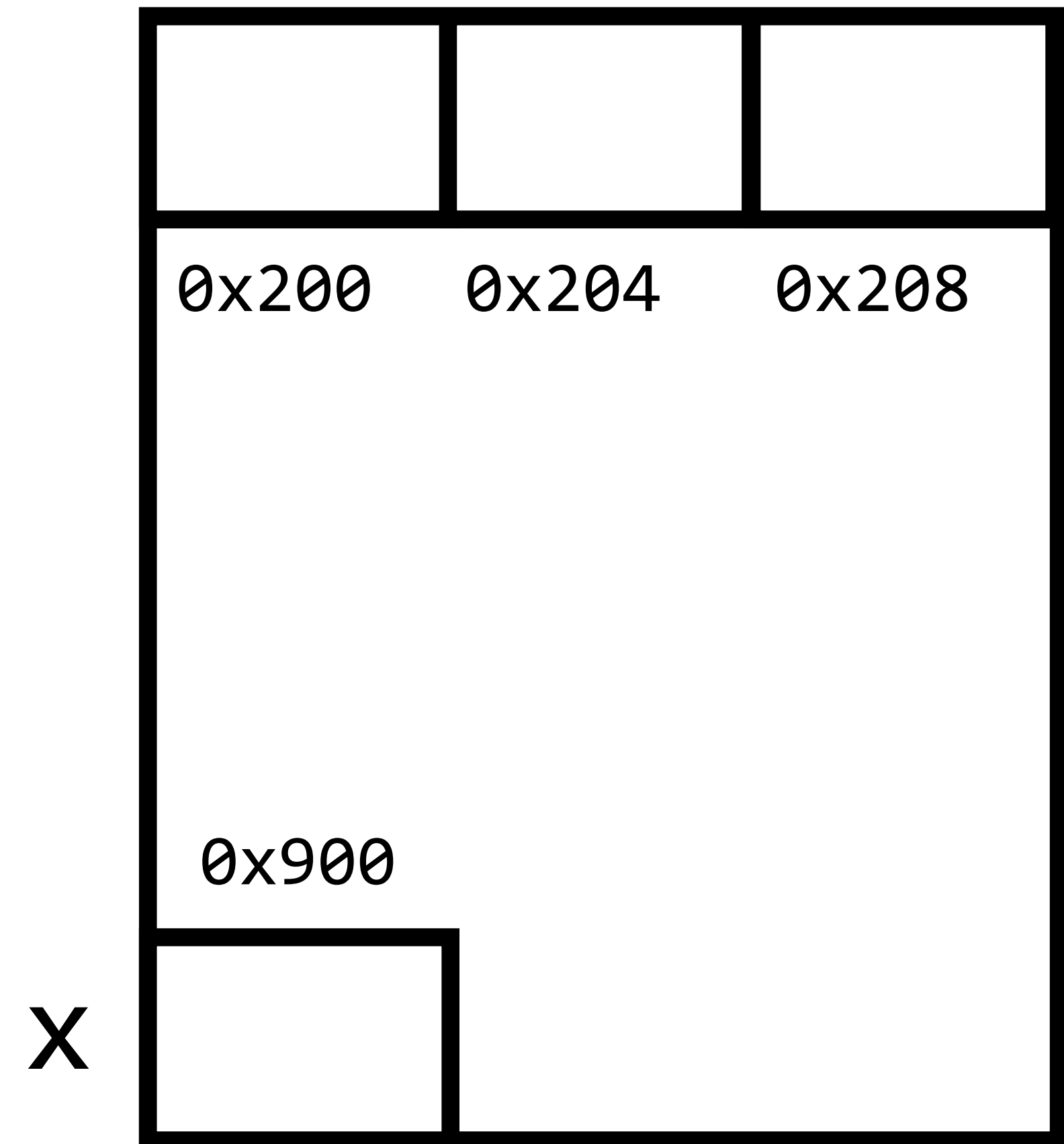
```
int *x = malloc(3 * sizeof(int));
```



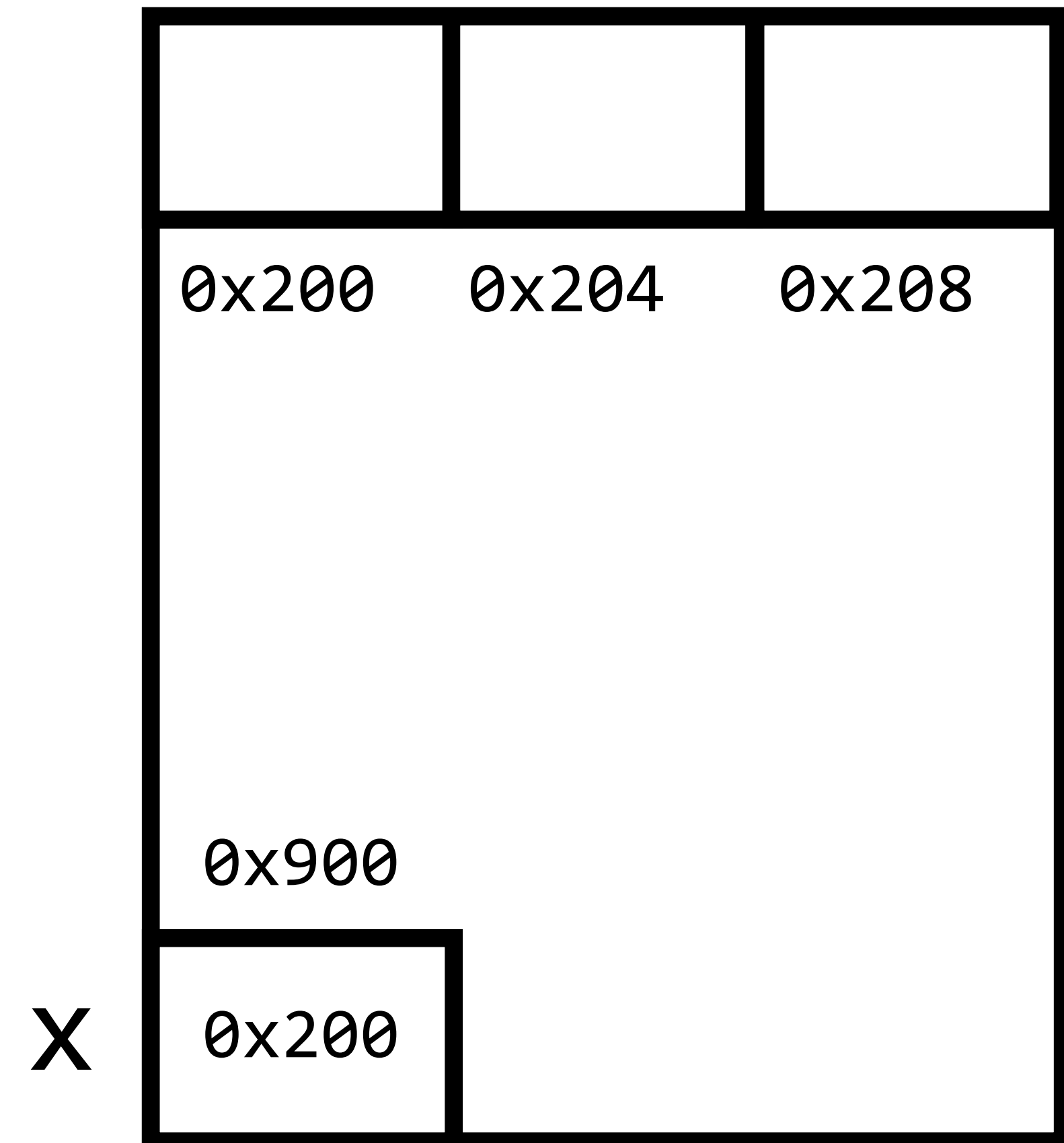
```
int *x = malloc(3 * sizeof(int));
```



```
int *x = malloc(3 * sizeof(int));
```

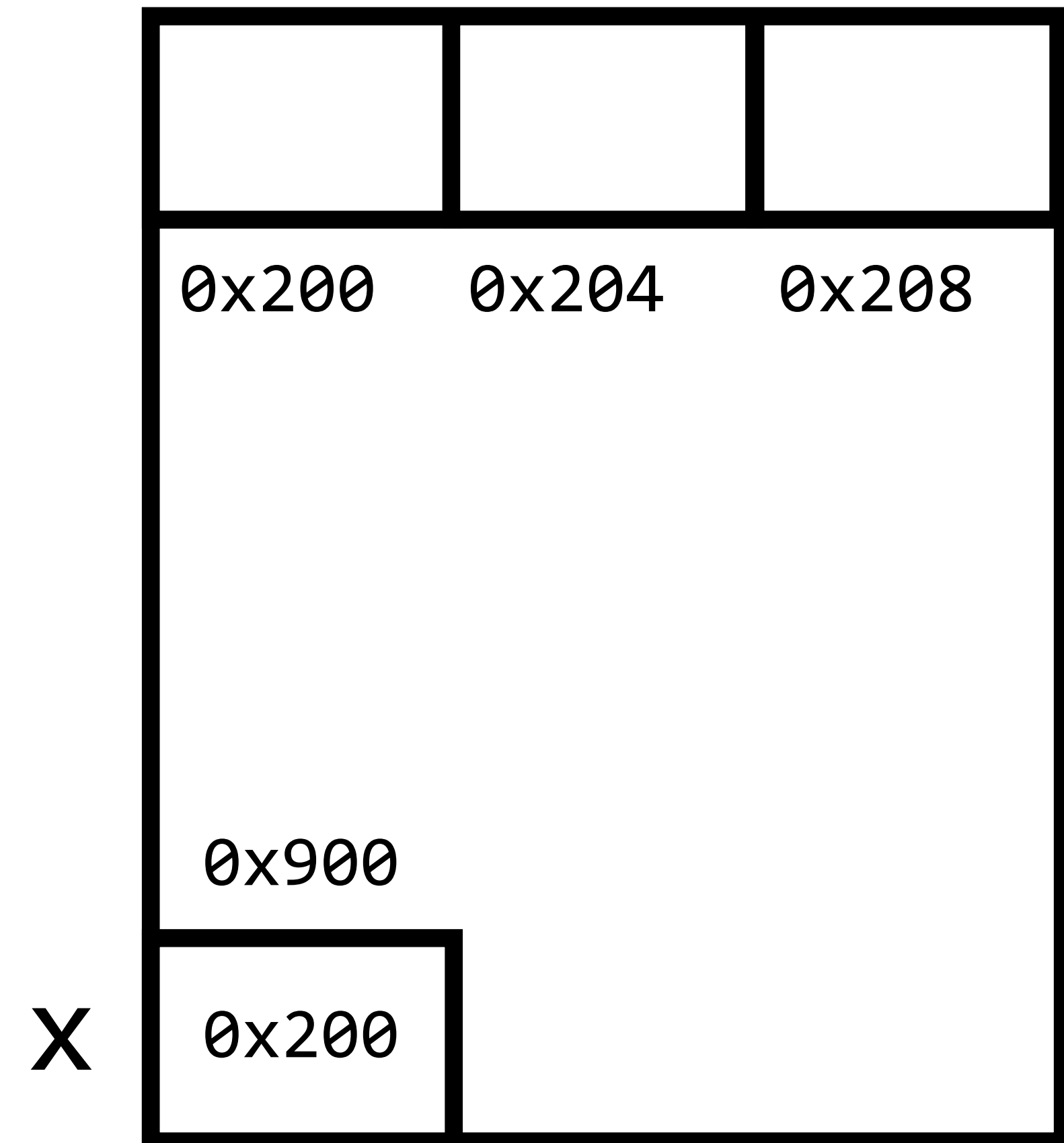


```
int *x = malloc(3 * sizeof(int));
```



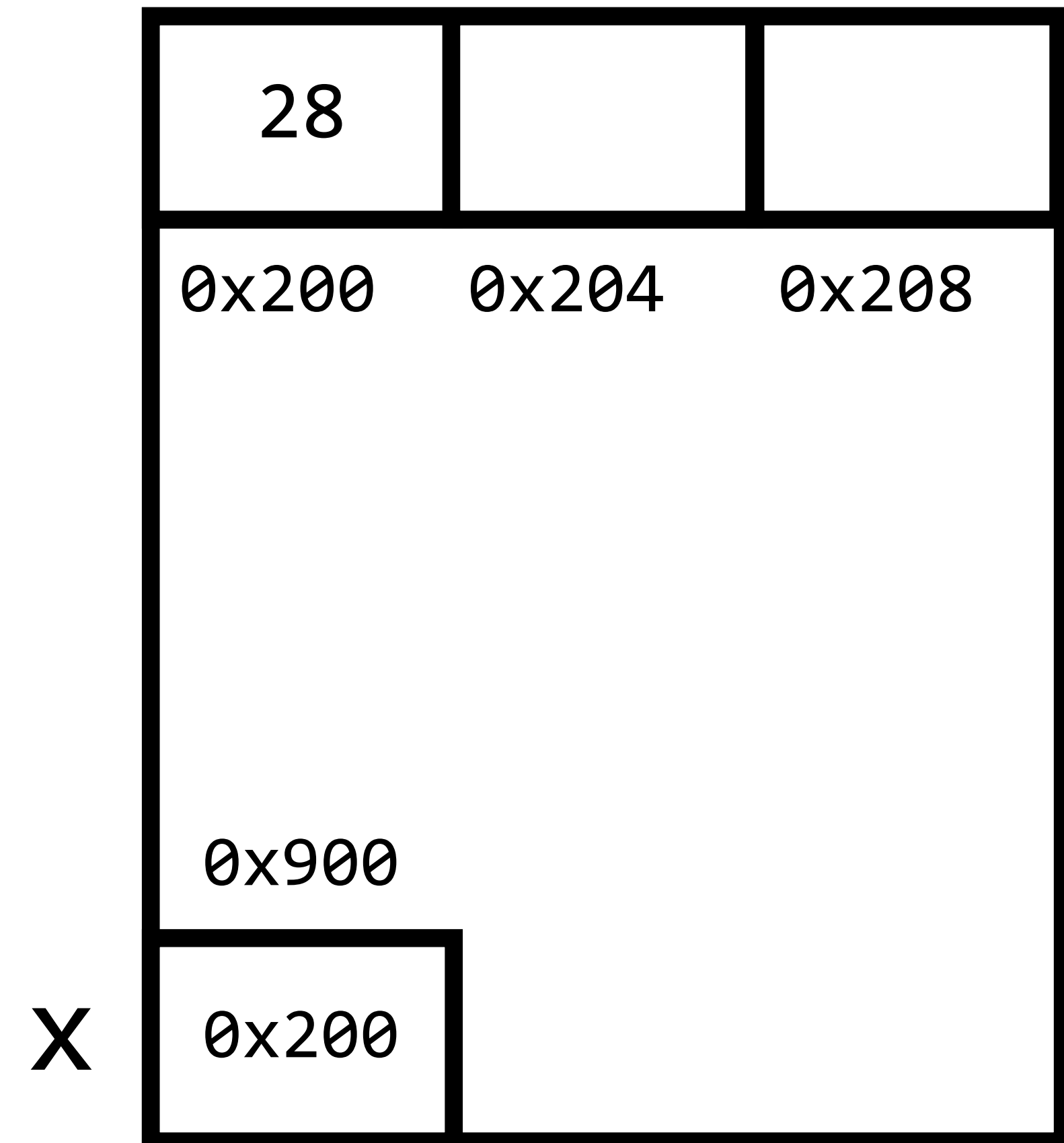
```
int *x = malloc(3 * sizeof(int));
```

```
*x = 28;
```




```
int *x = malloc(3 * sizeof(int));
```

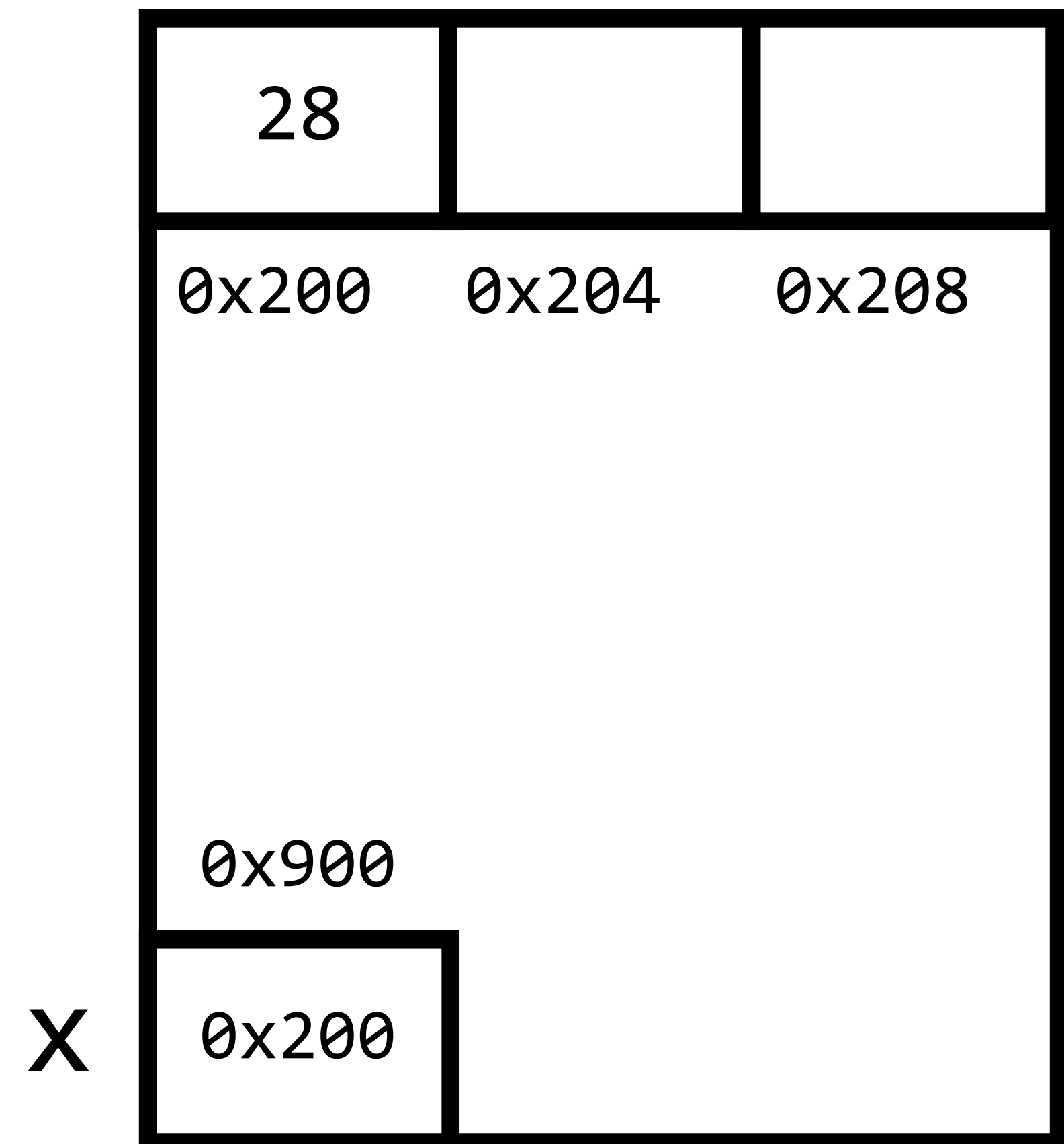
```
*x = 28;
```



```
int *x = malloc(3 * sizeof(int));
```

```
*x = 28;
```

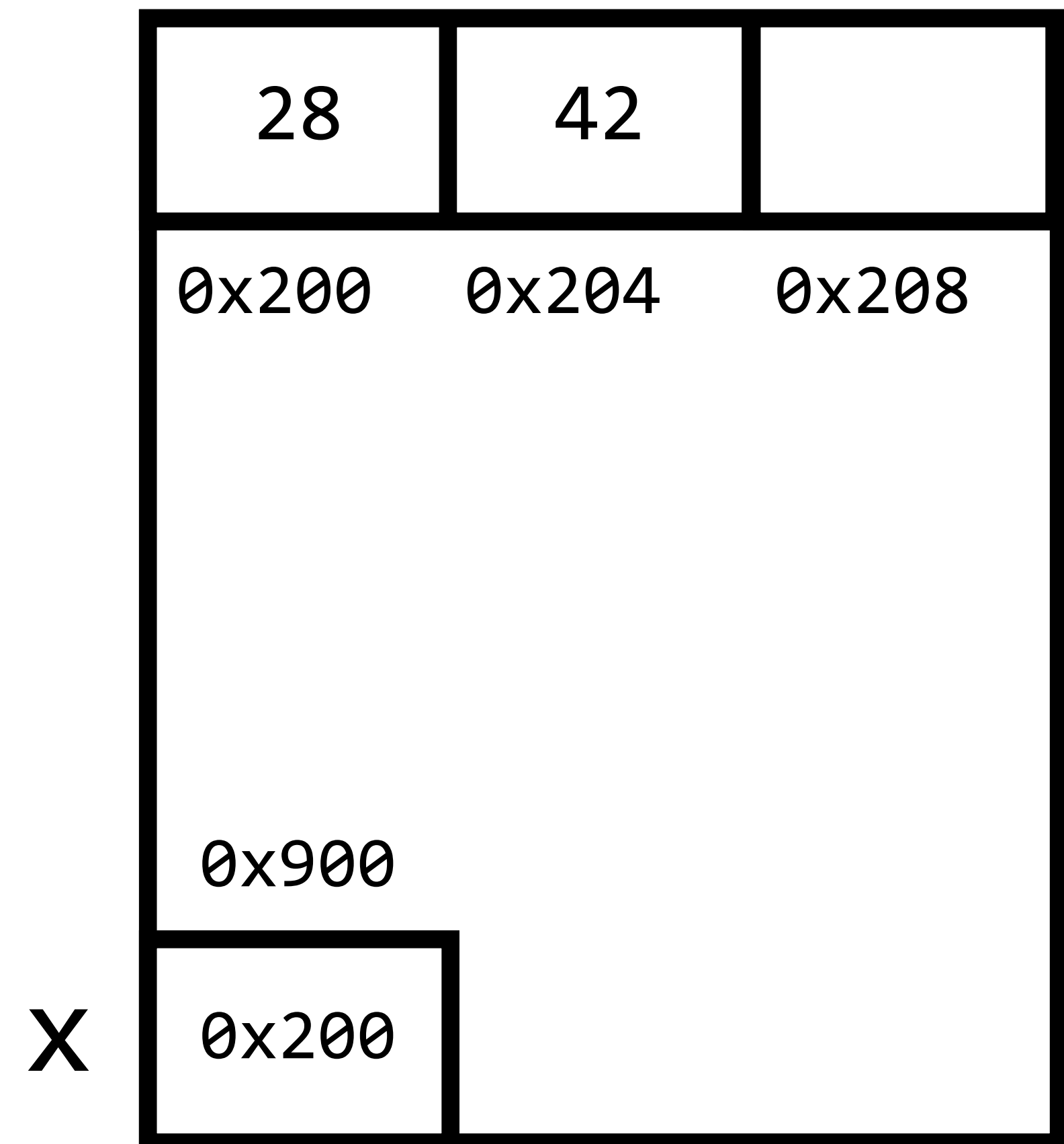
```
*(x + 1) = 42;
```



```
int *x = malloc(3 * sizeof(int));
```

```
*x = 28;
```

```
*(x + 1) = 42;
```

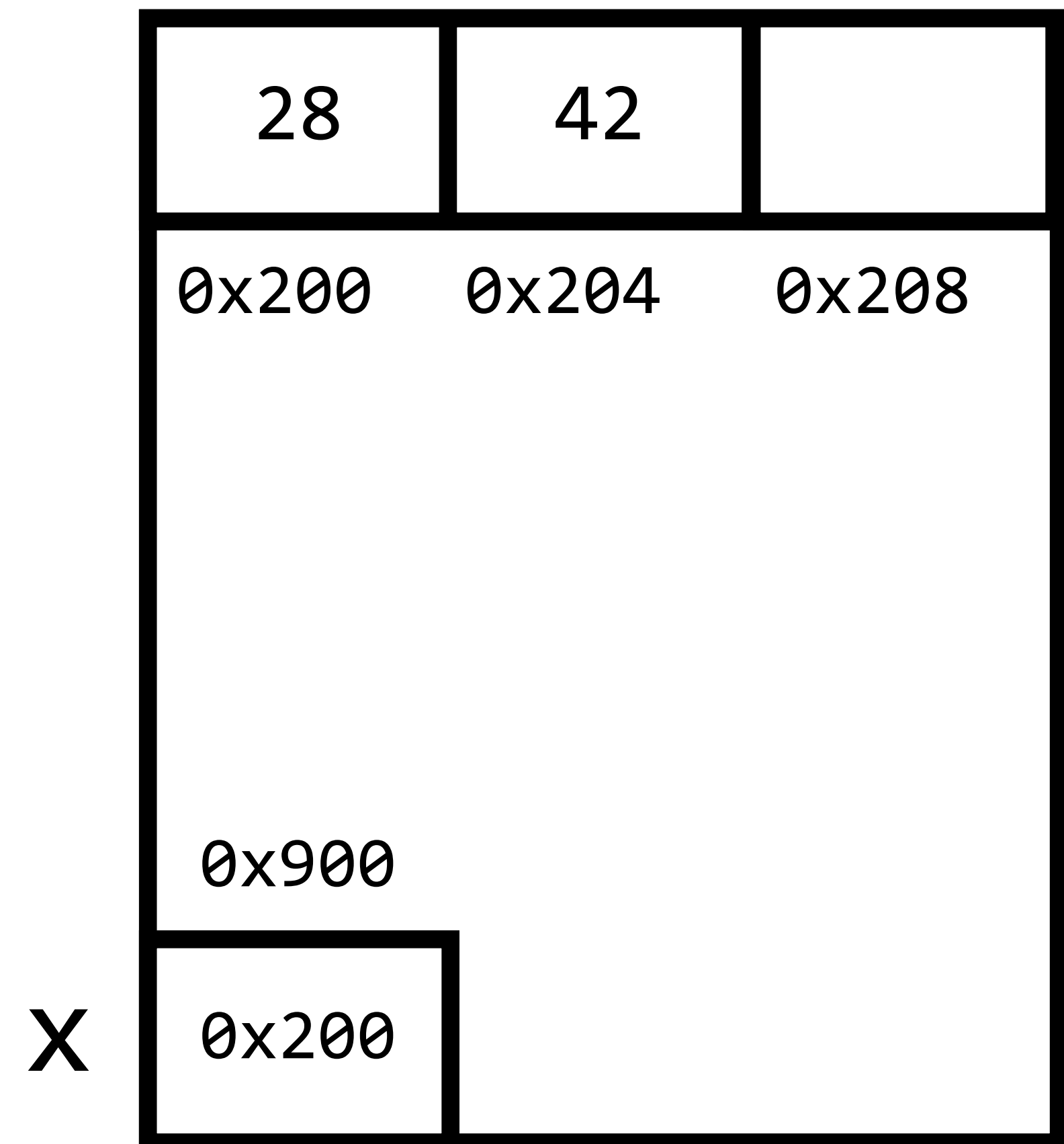


```
int *x = malloc(3 * sizeof(int));
```

```
*x = 28;
```

```
*(x + 1) = 42;
```

```
*(x + 2) = 50;
```

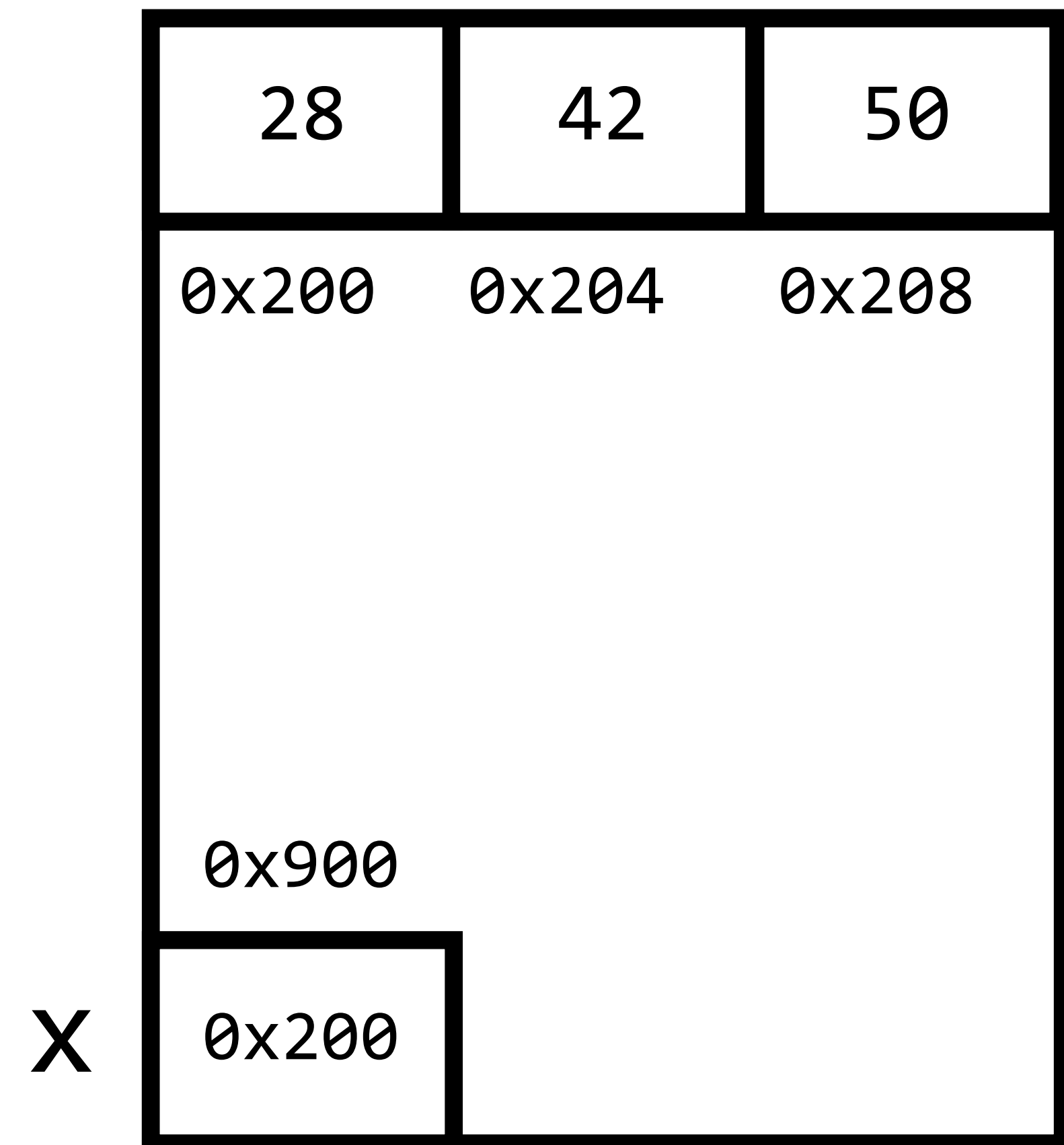


```
int *x = malloc(3 * sizeof(int));
```

```
*x = 28;
```

```
*(x + 1) = 42;
```

```
*(x + 2) = 50;
```

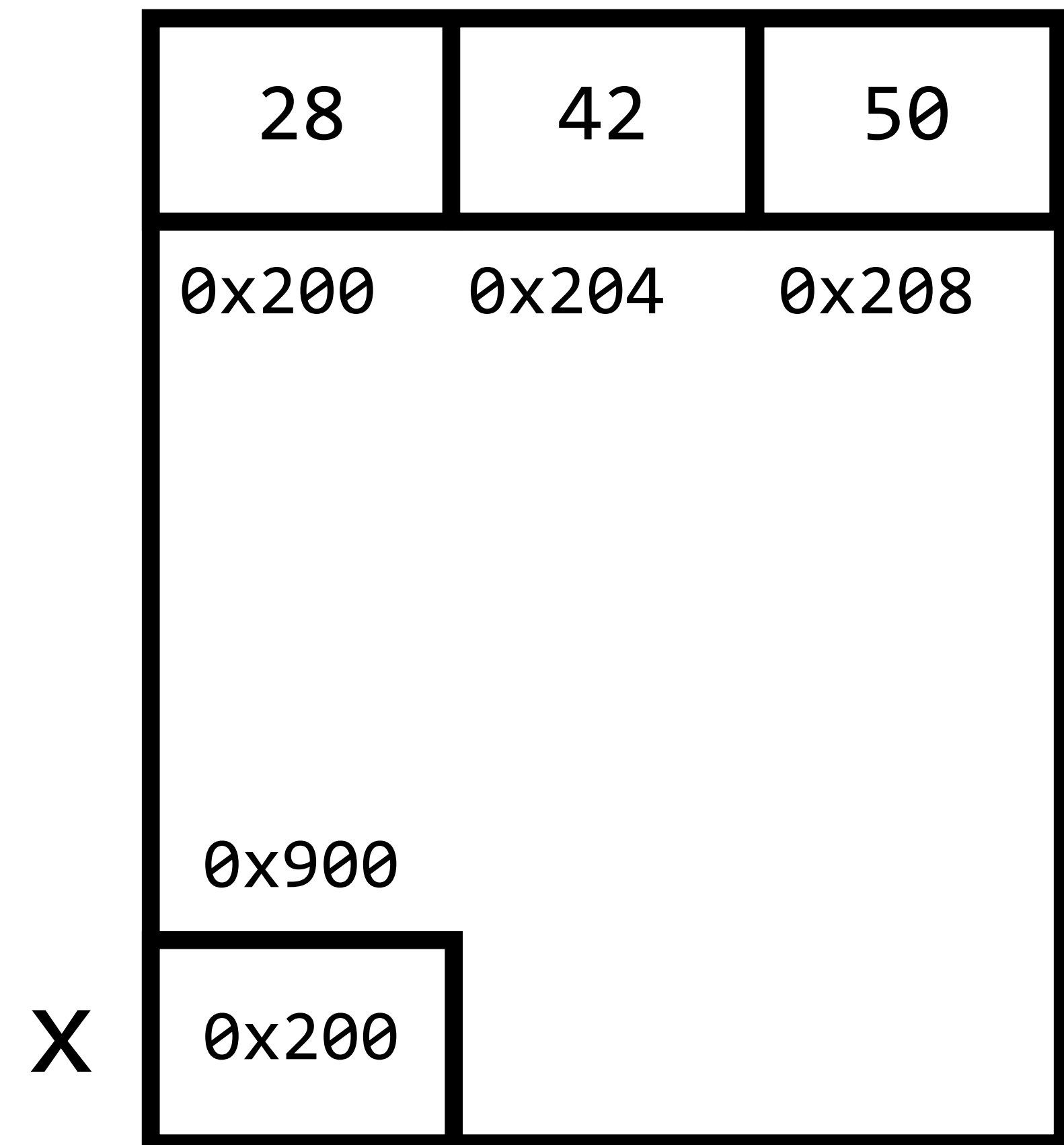


```
int *x = malloc(3 * sizeof(int));
```

```
x[0] = 28;
```

```
x[1] = 42;
```

```
x[2] = 50;
```

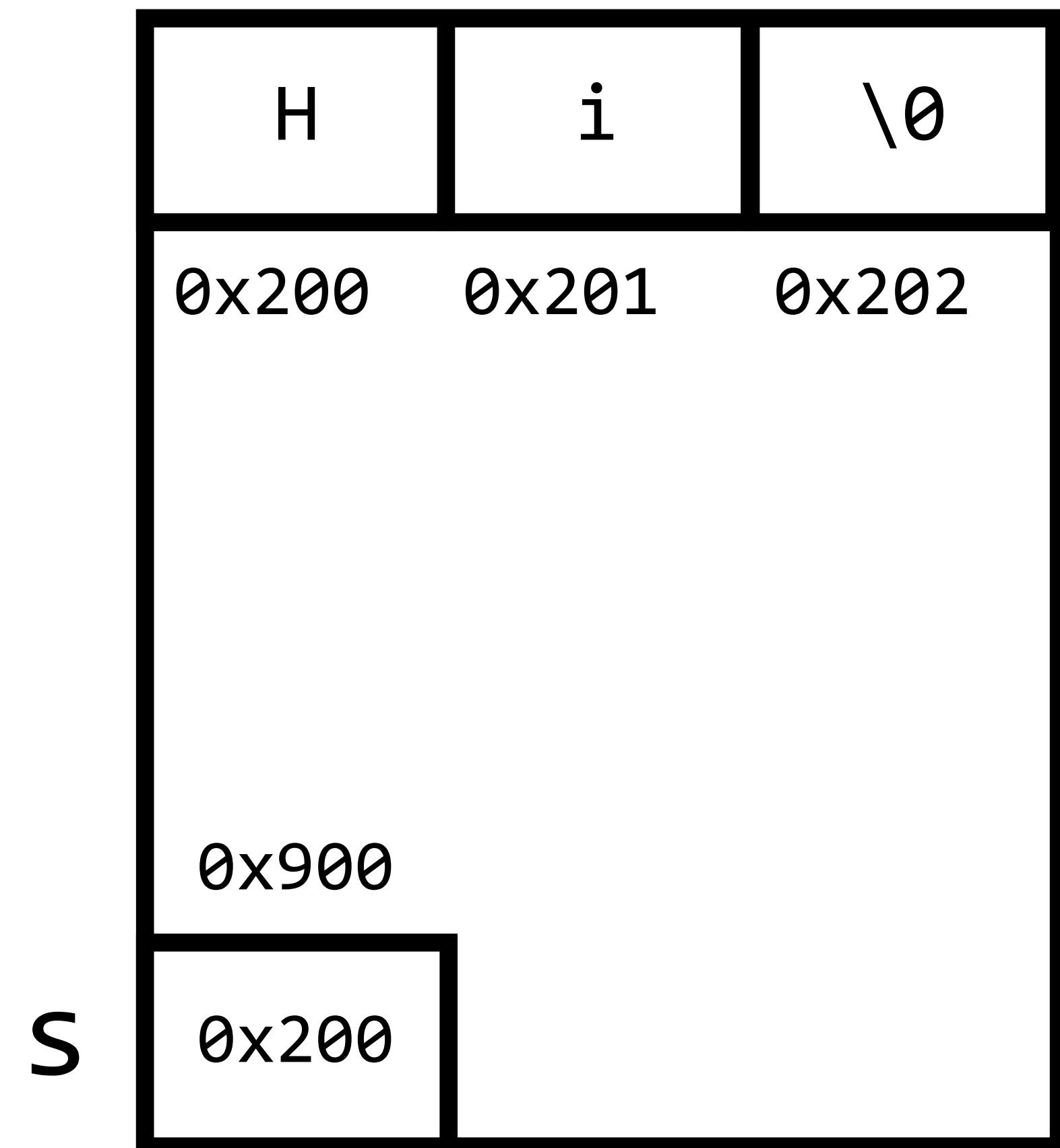


```
char *s = malloc(3 * sizeof(char));
```

```
s[0] = 'H';
```

```
s[1] = 'i';
```

```
s[2] = '\0';
```



Memory Errors

- Always **free** memory after done with it
- Use **valgrind** to check for memory errors

Problem Set 4

Problem Set 4

- Filter
 - Grayscale
 - Sepia (less comfortable)
 - Reflect
 - Blur
 - Edges (more comfortable)
- Recover



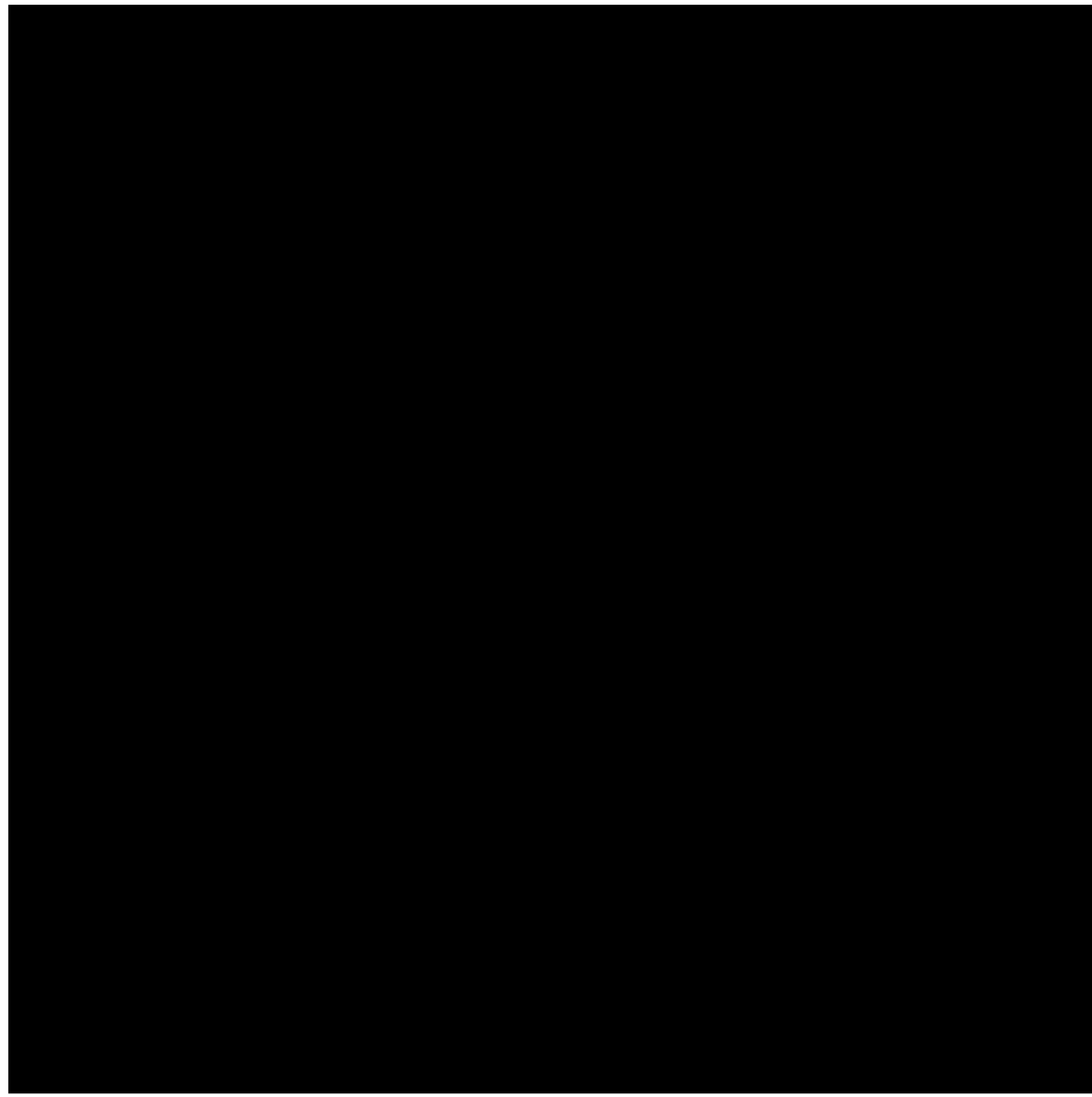






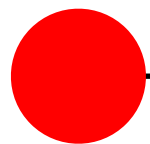






Red

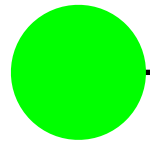
0



255

Green

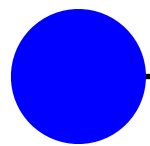
0



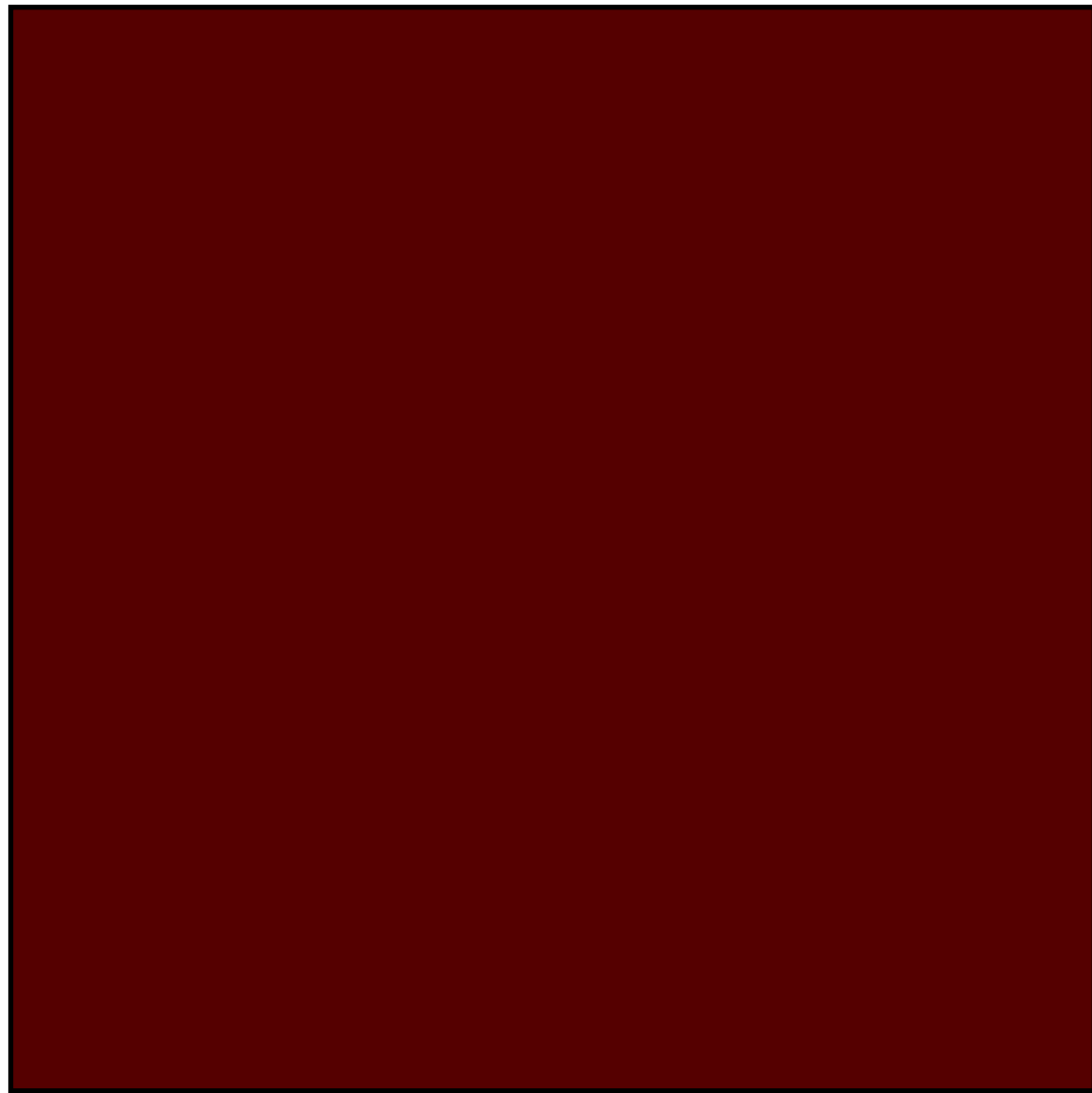
255

Blue

0

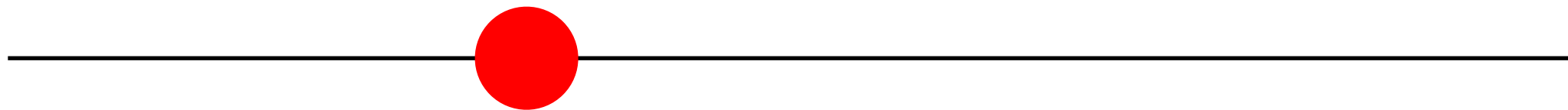


255



Red

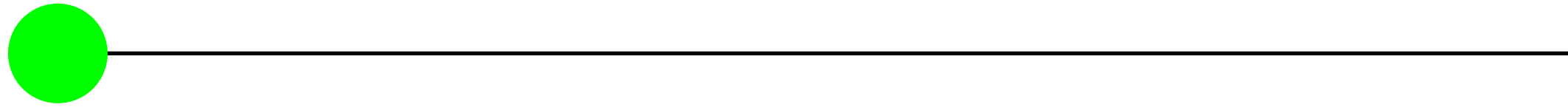
0



255

Green

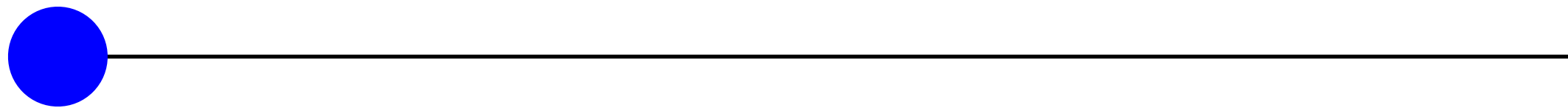
0



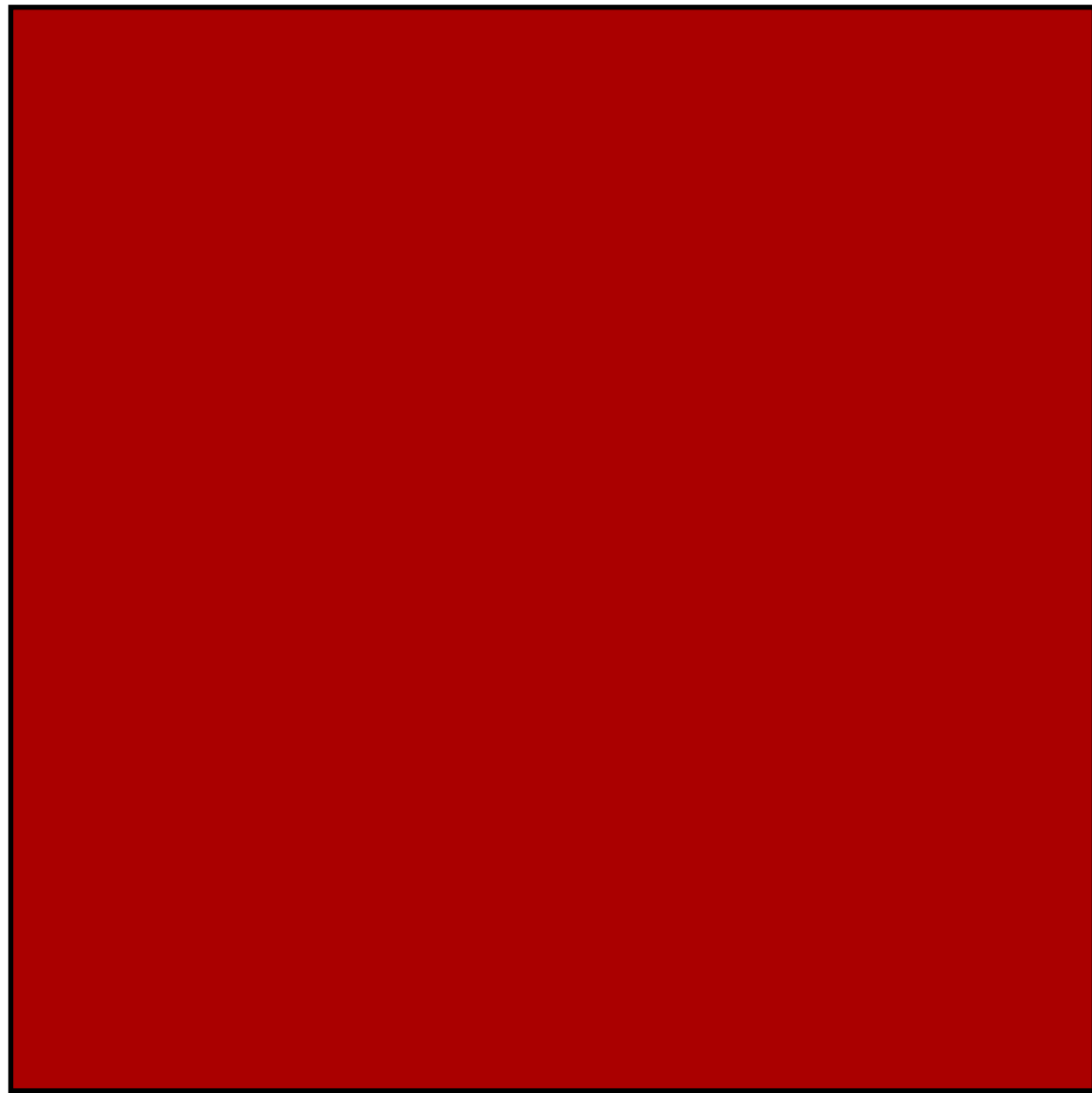
255

Blue

0

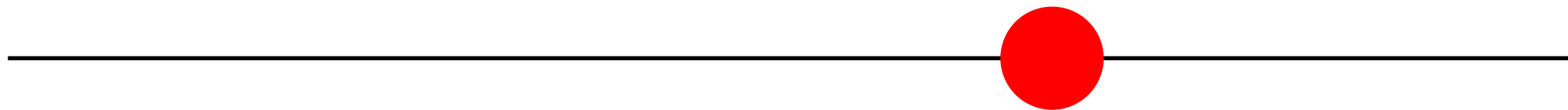


255



Red

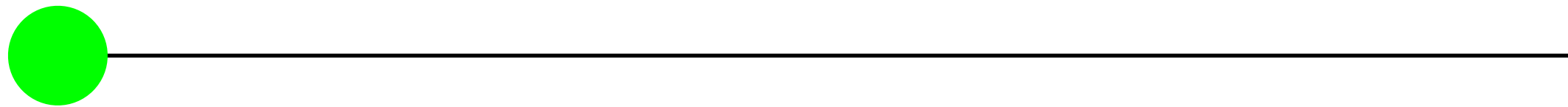
0



255

Green

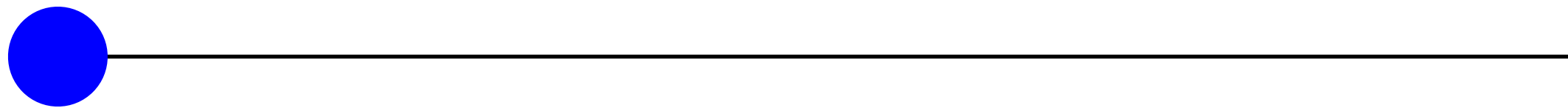
0



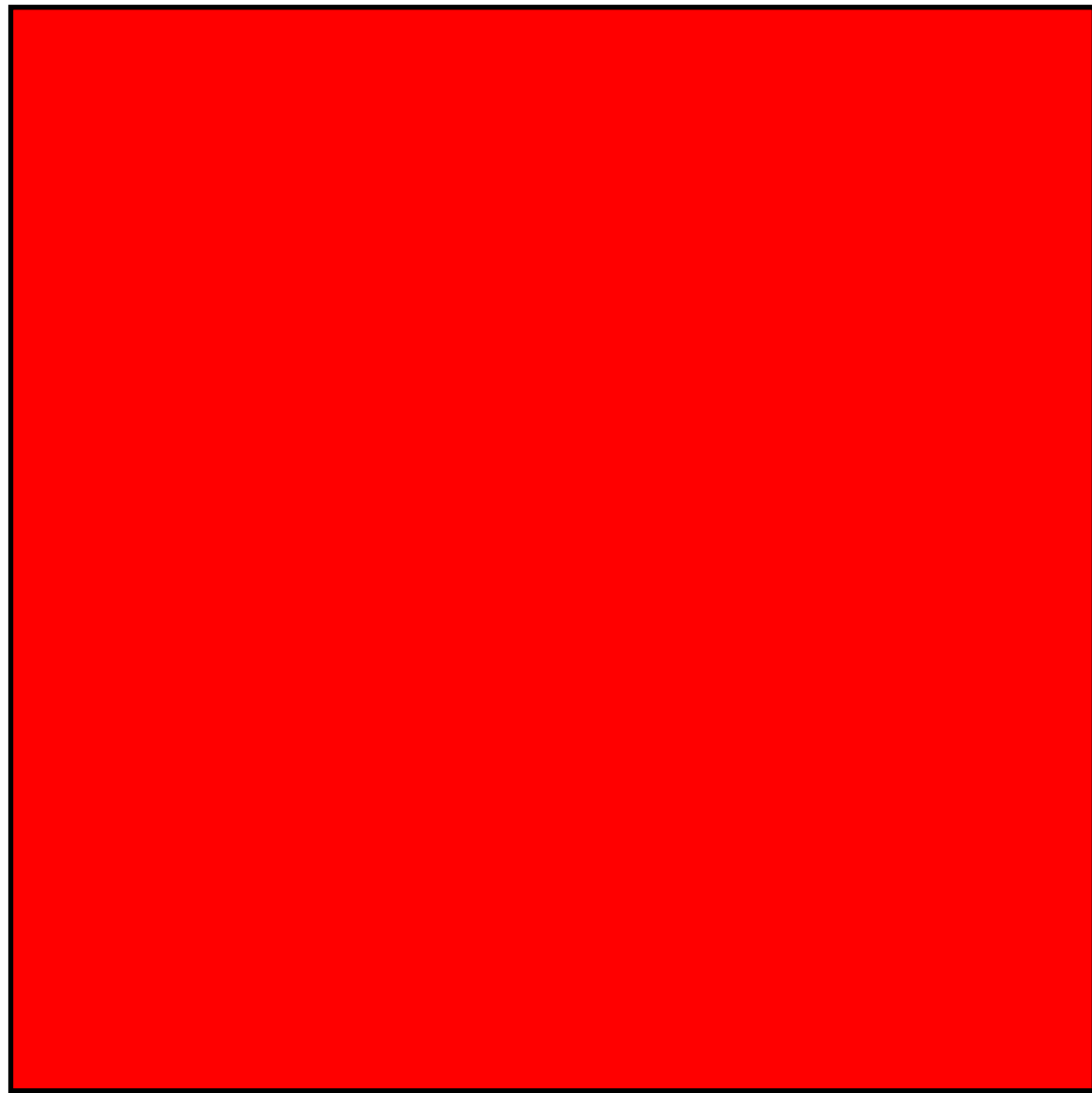
255

Blue

0

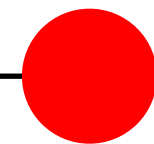


255



Red

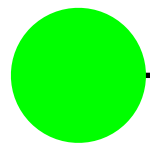
0



255

Green

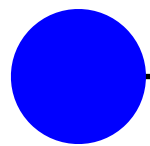
0



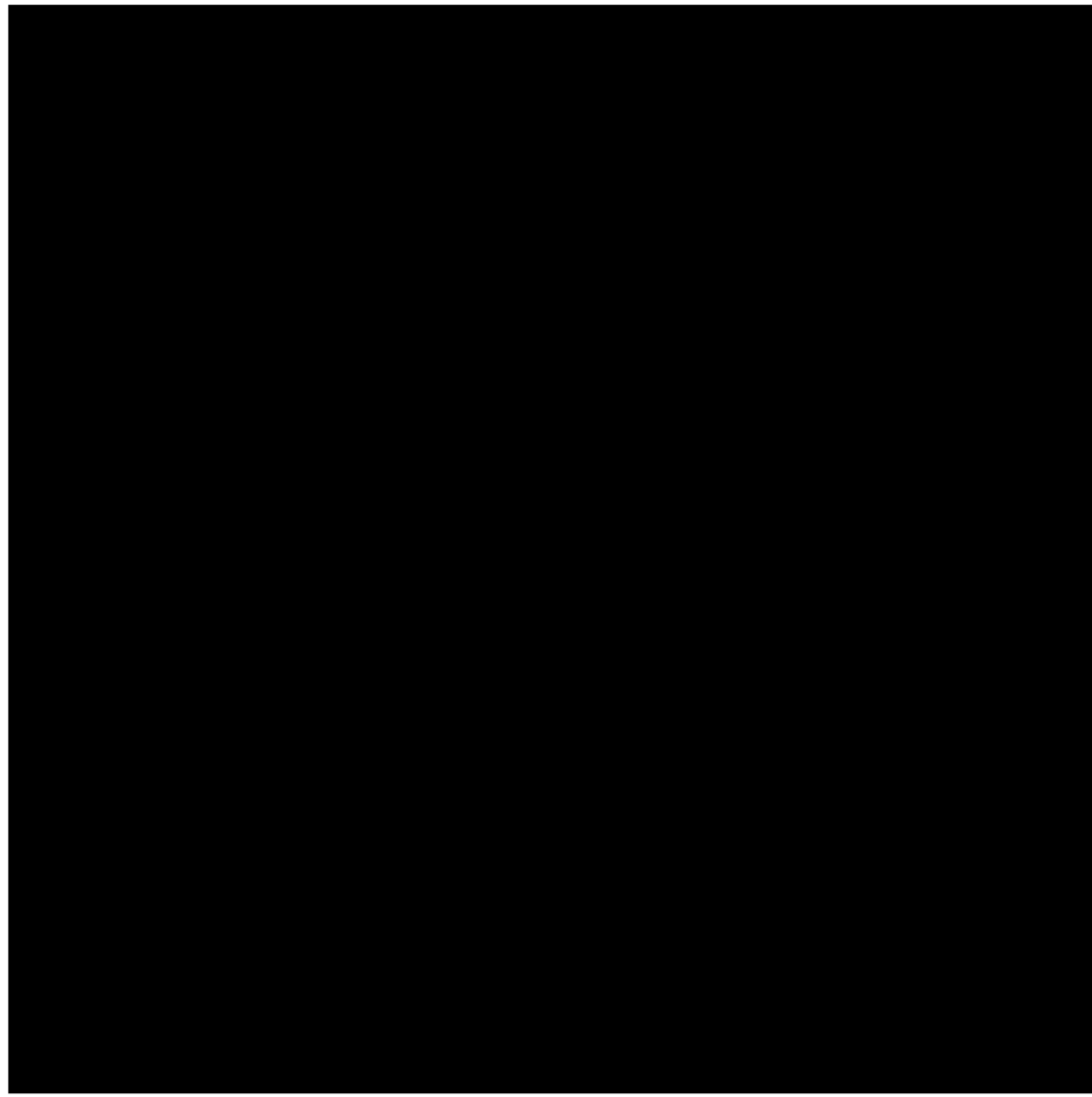
255

Blue

0

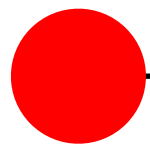


255



Red

0

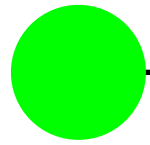


255



Green

0

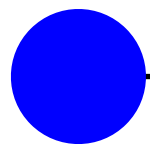


255



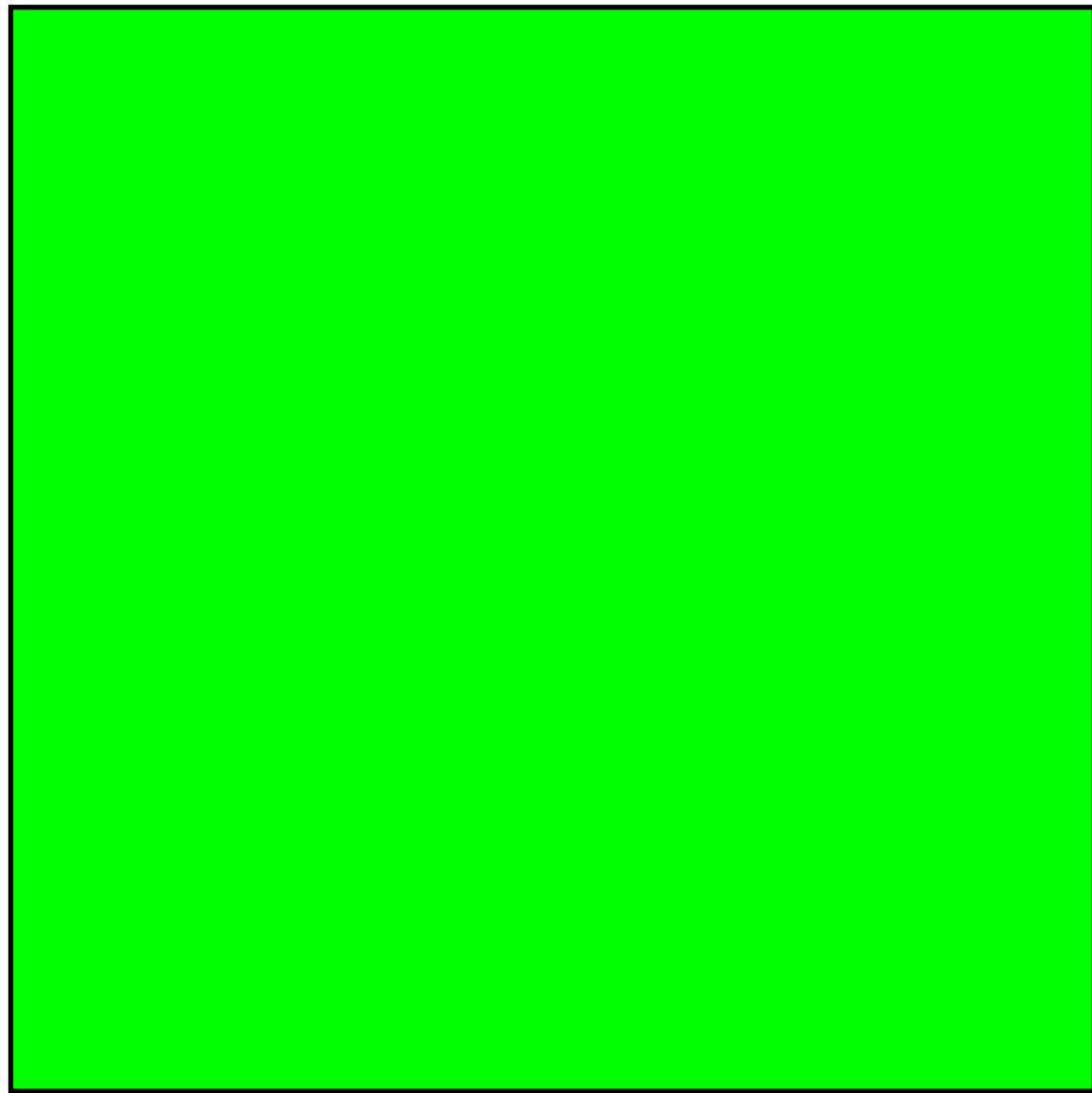
Blue

0



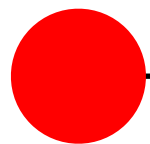
255





Red

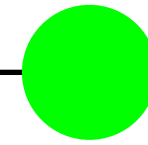
0



255

Green

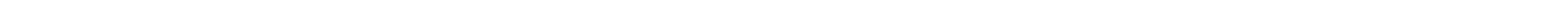
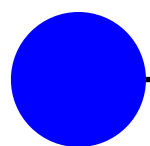
0



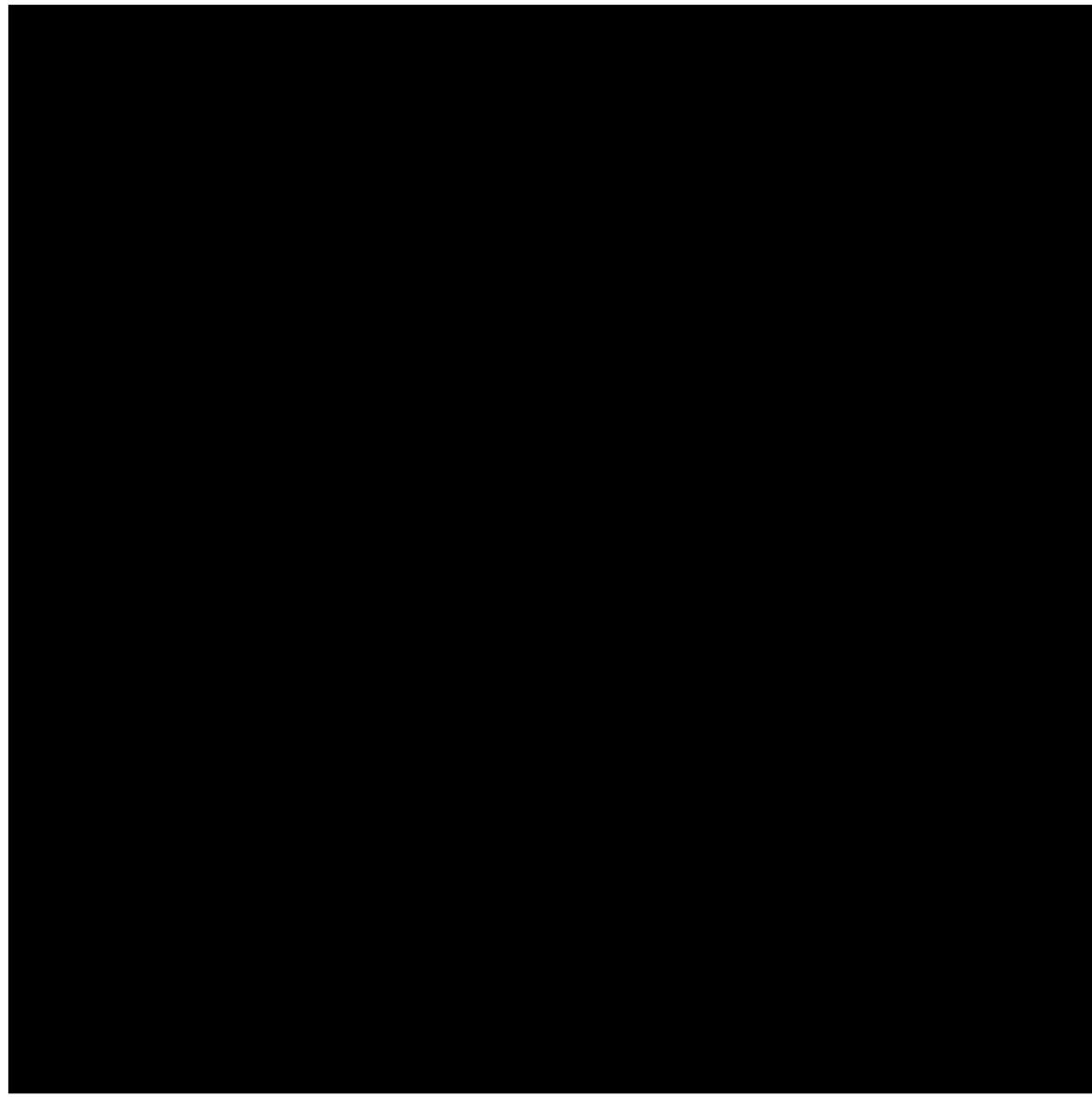
255

Blue

0

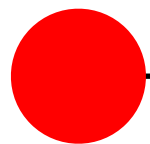


255



Red

0

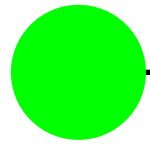


255



Green

0

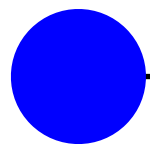


255



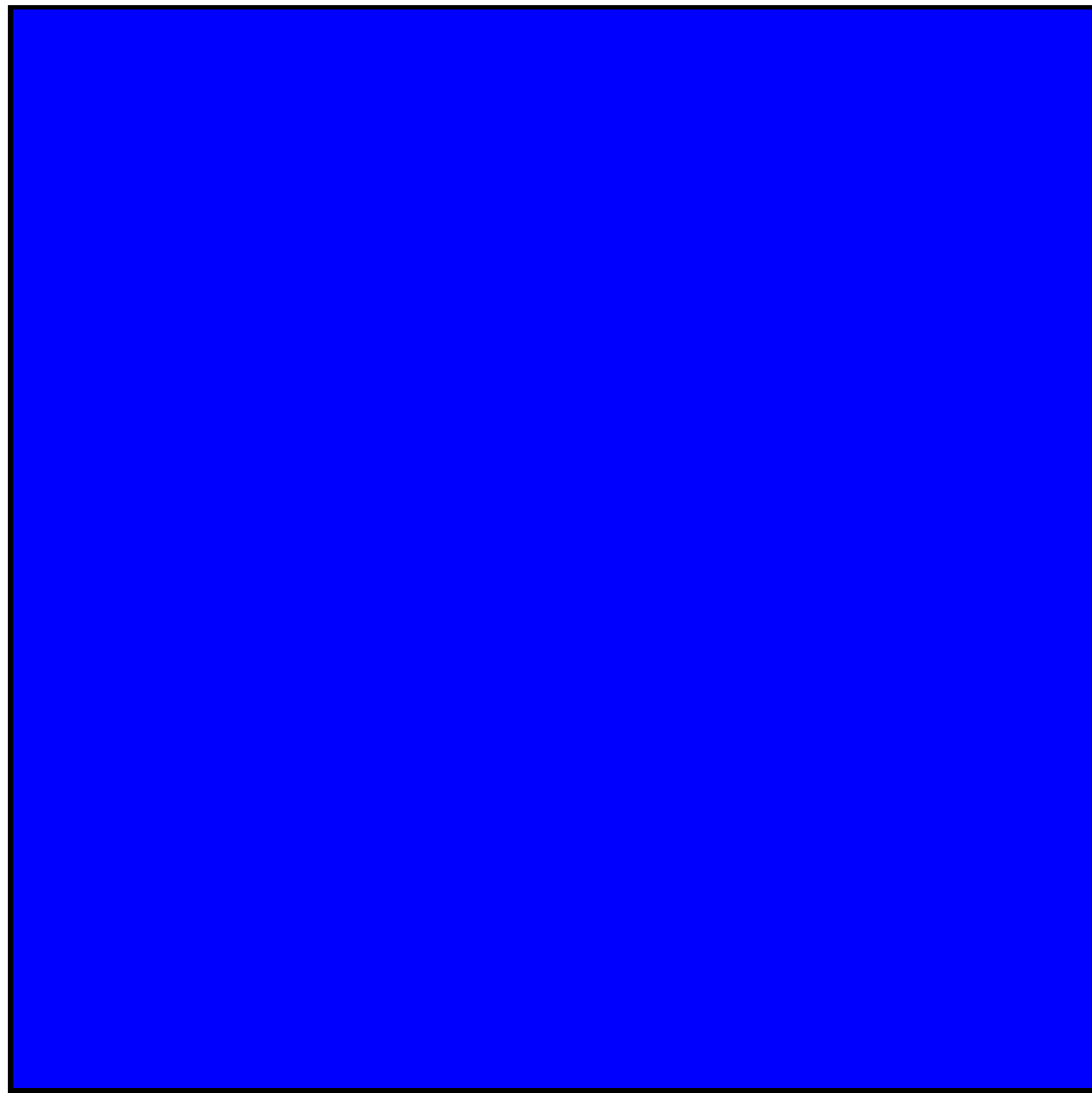
Blue

0



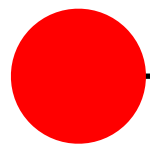
255





Red

0

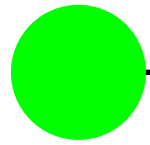


255



Green

0

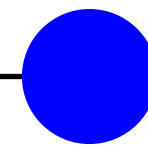


255



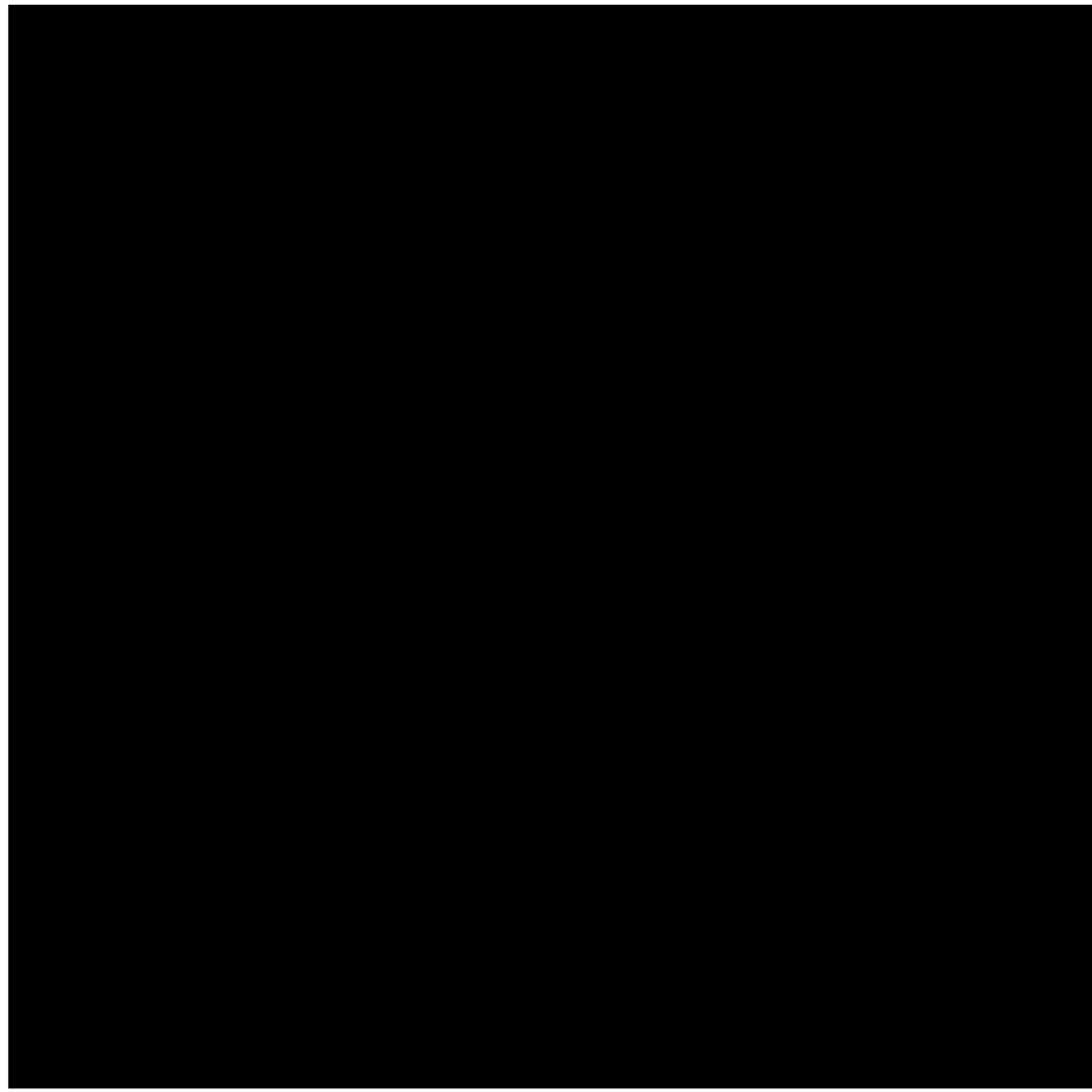
Blue

0



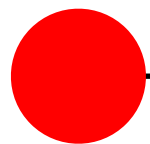
255





Red

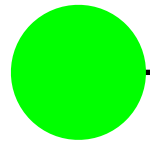
0



255

Green

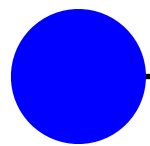
0



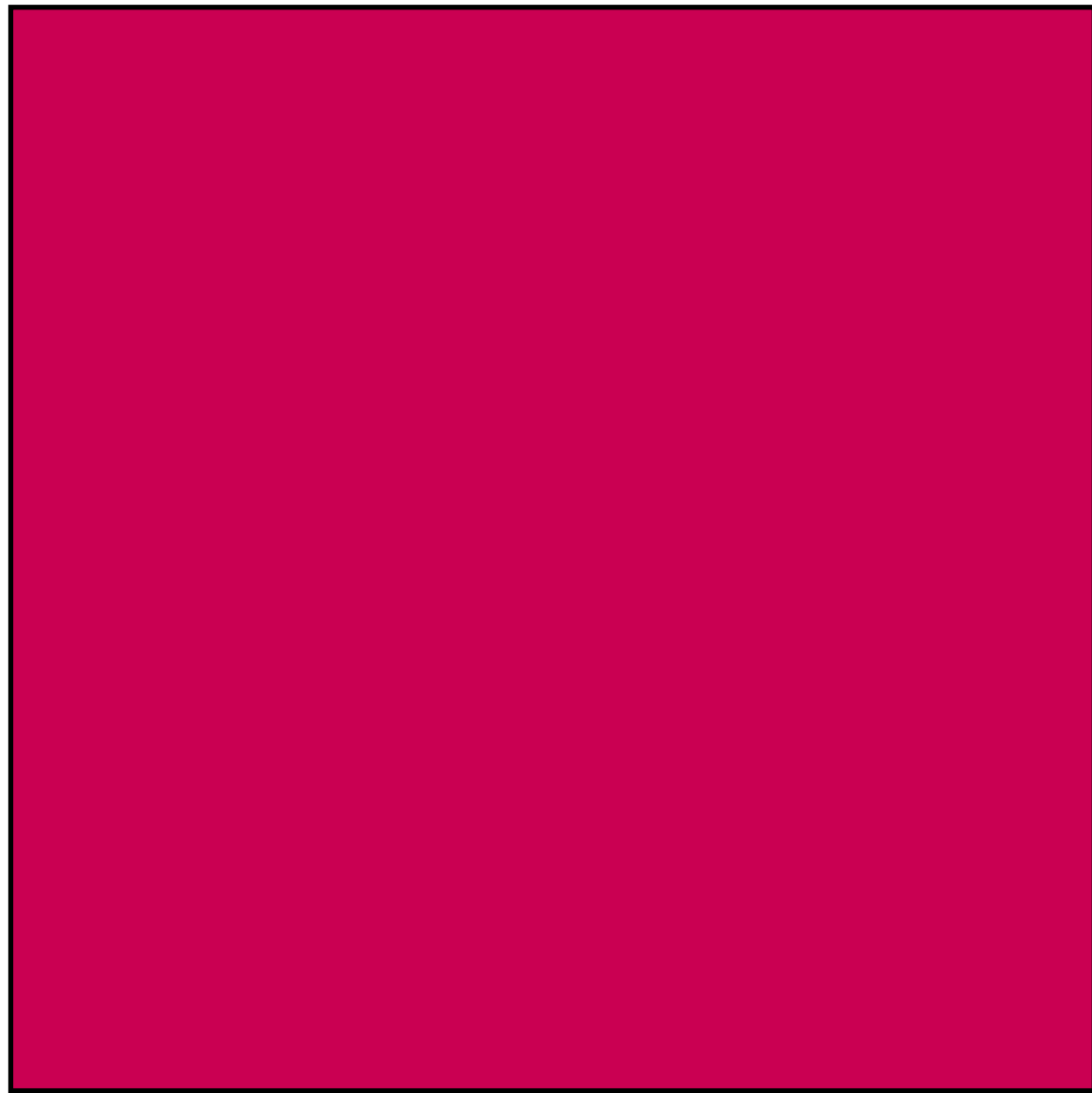
255

Blue

0

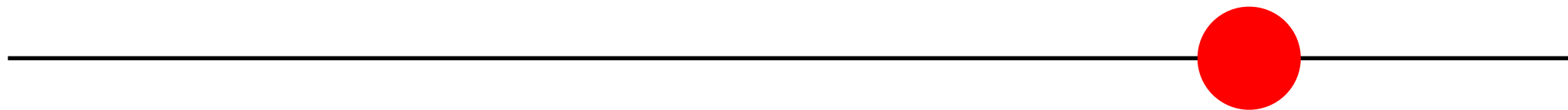


255



Red

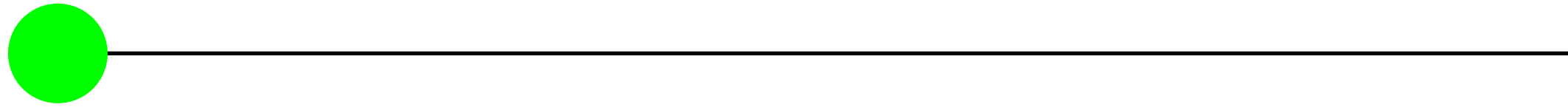
0



255

Green

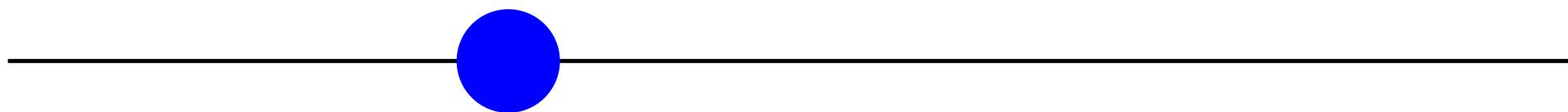
0



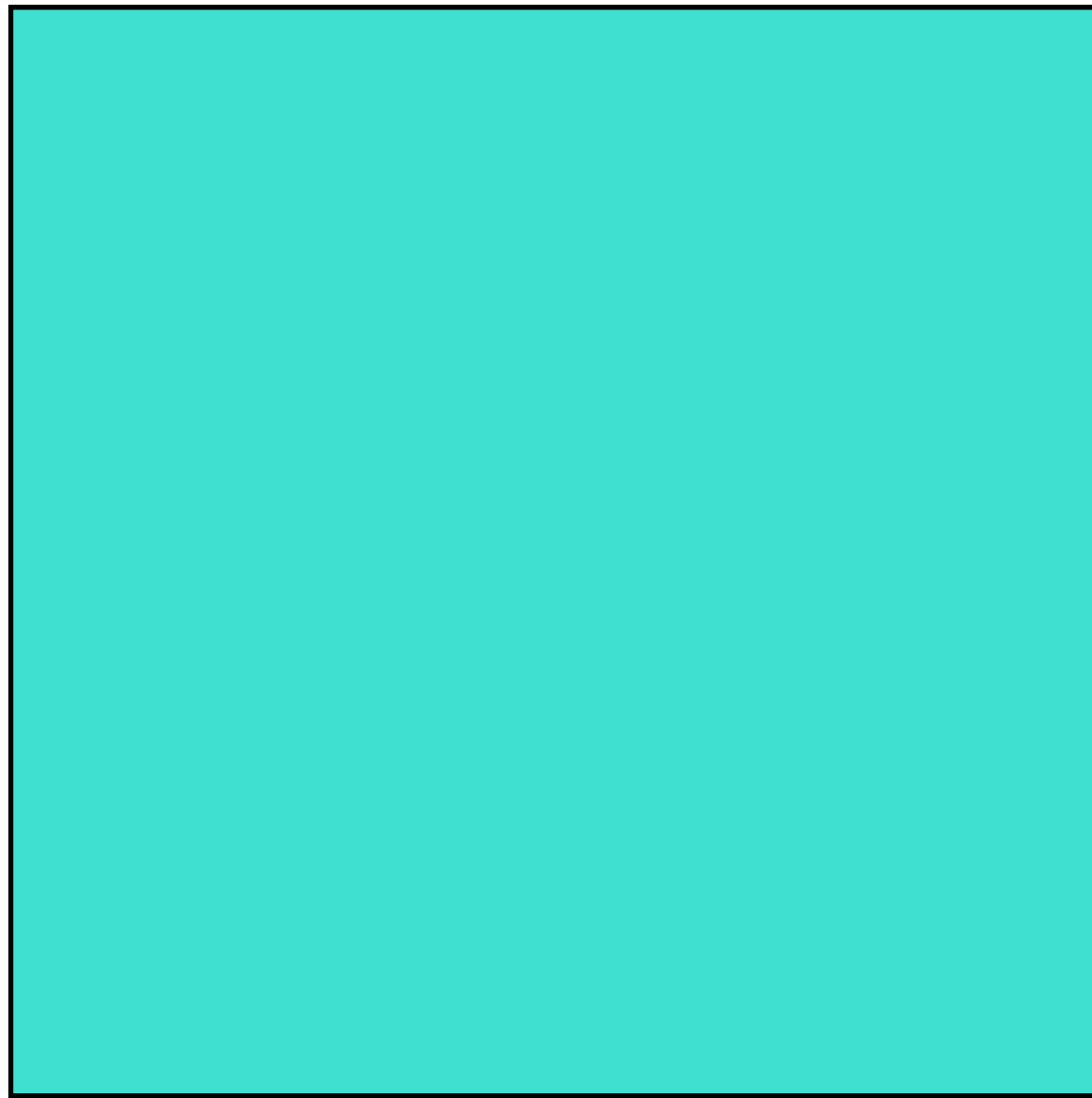
255

Blue

0

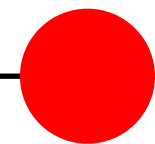


255



Red

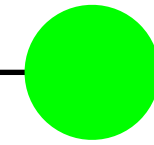
0



255

Green

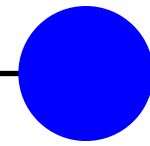
0



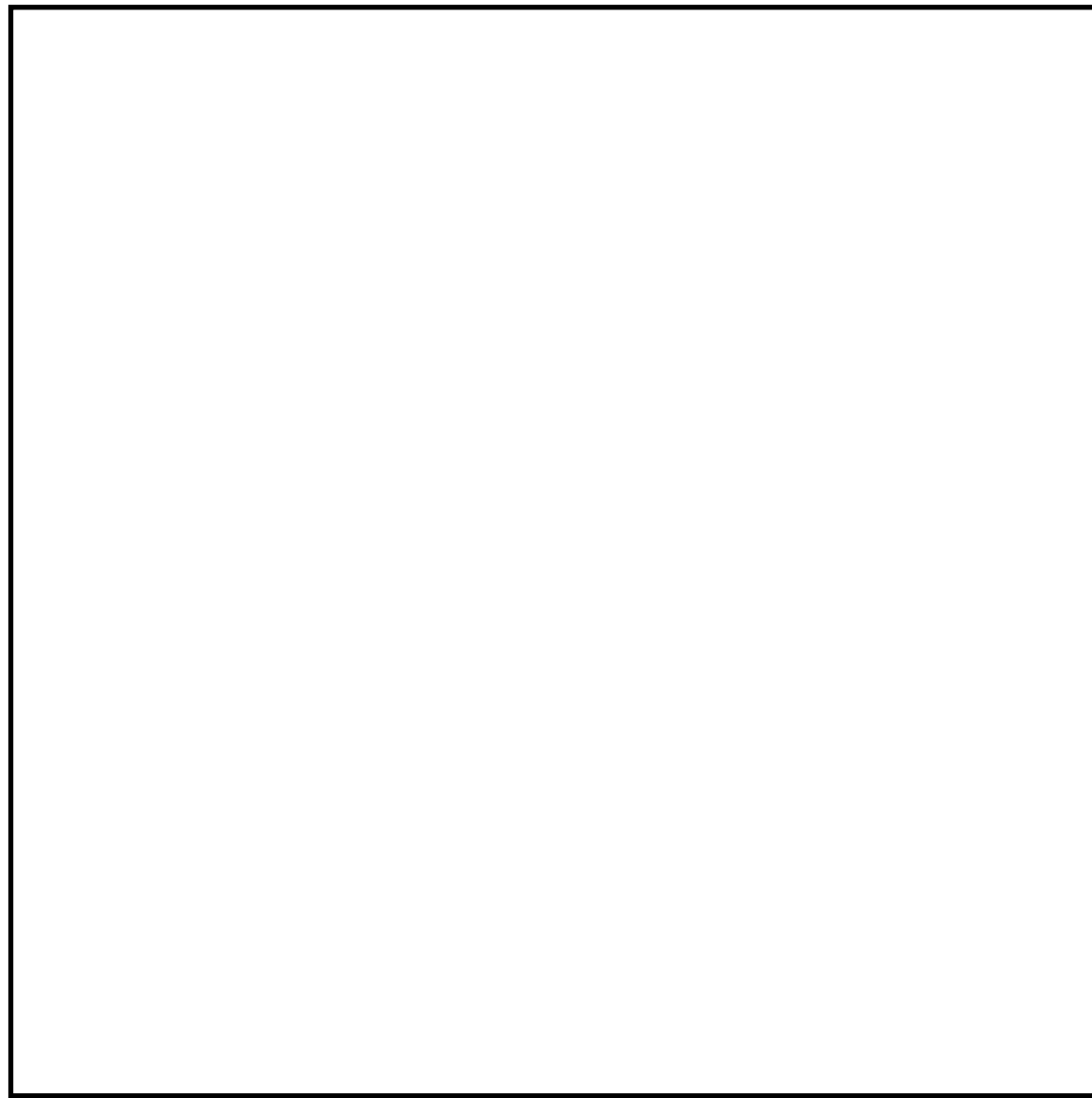
255

Blue

0

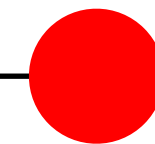


255



Red

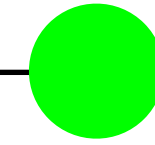
0



255

Green

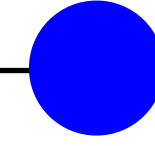
0



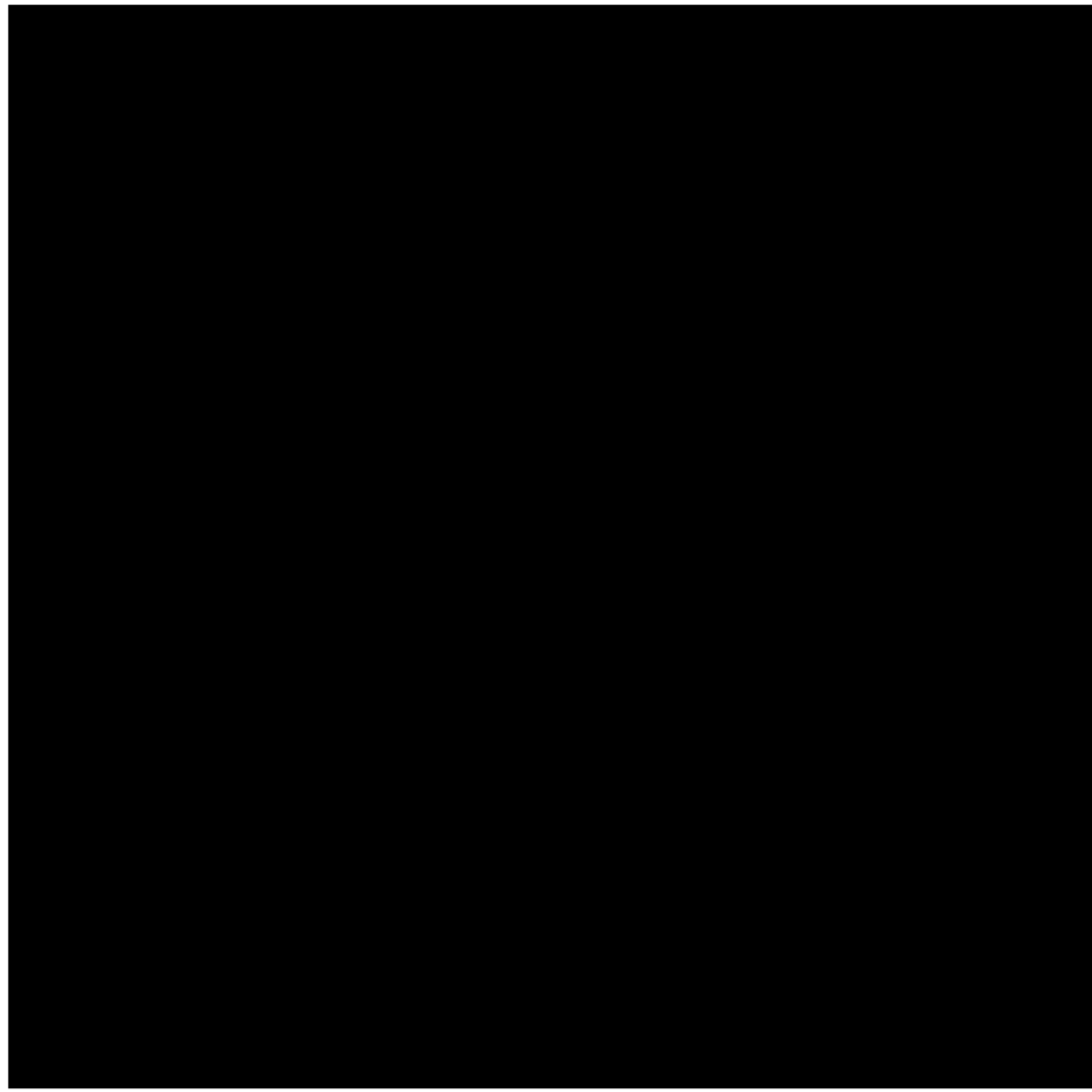
255

Blue

0

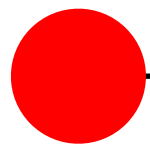


255



Red

0

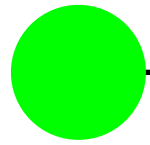


255



Green

0

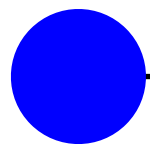


255



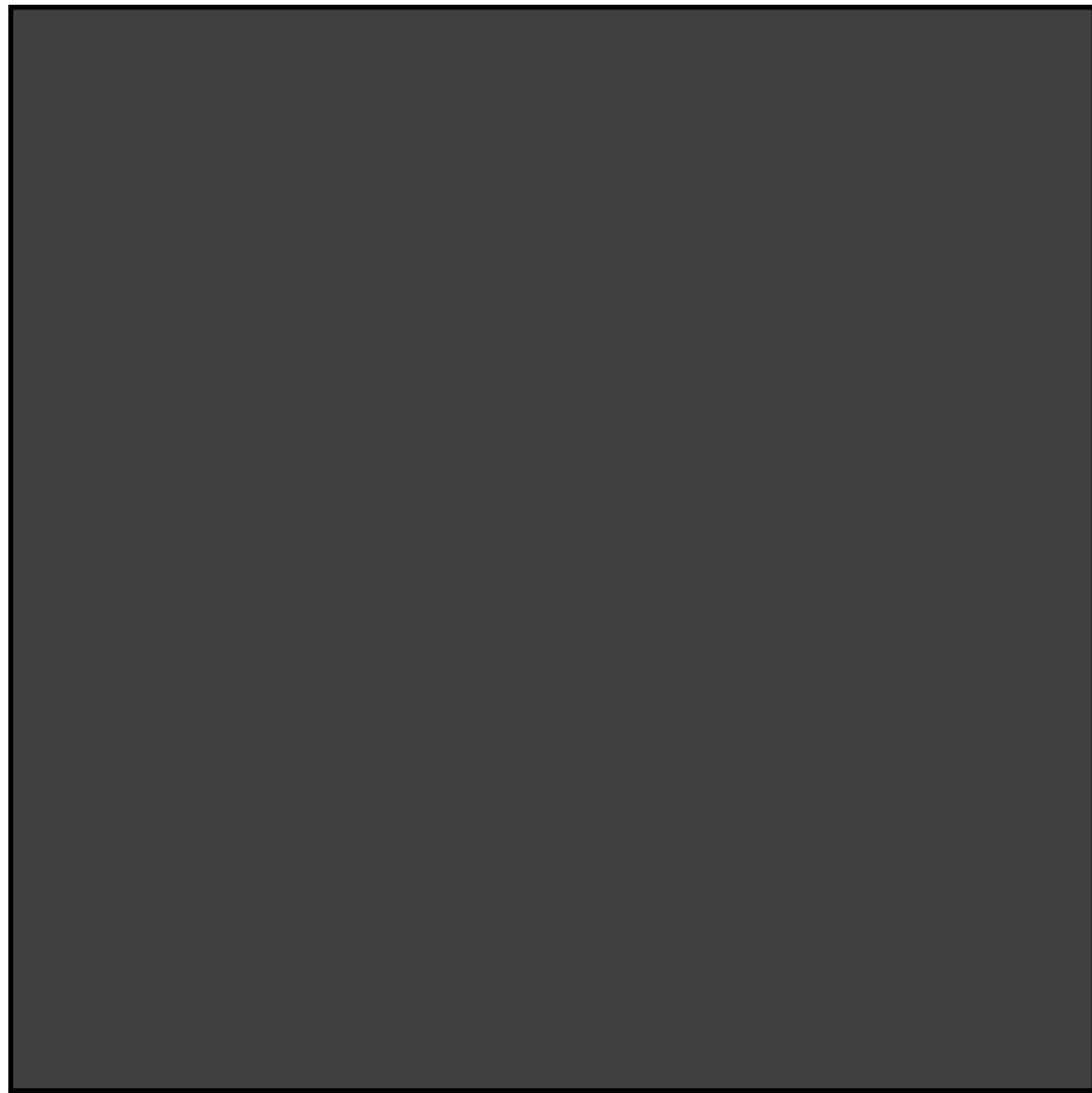
Blue

0



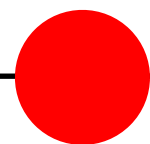
255





Red

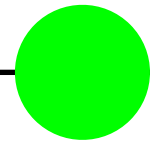
0



255

Green

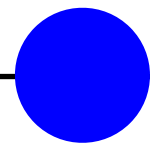
0



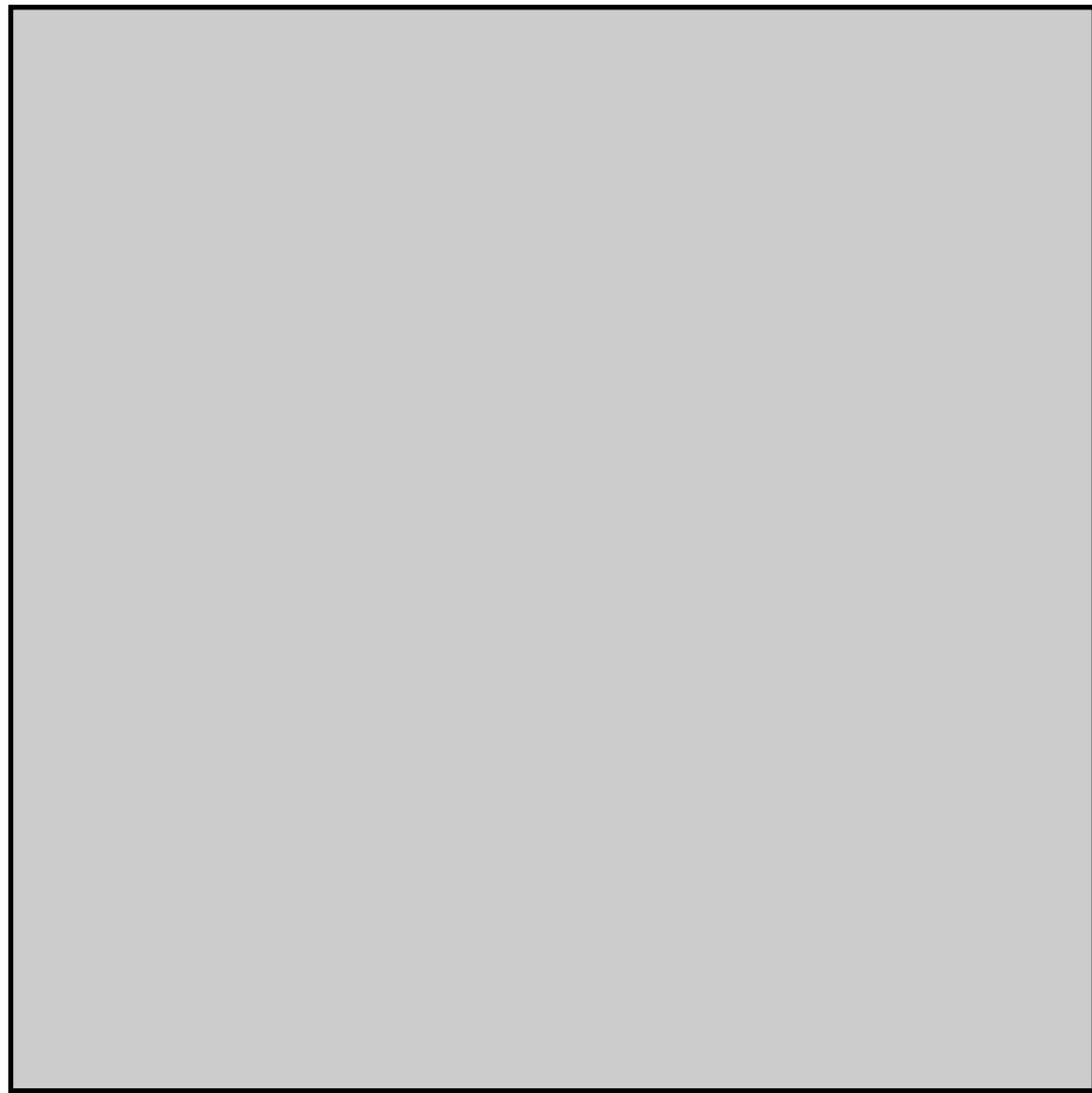
255

Blue

0

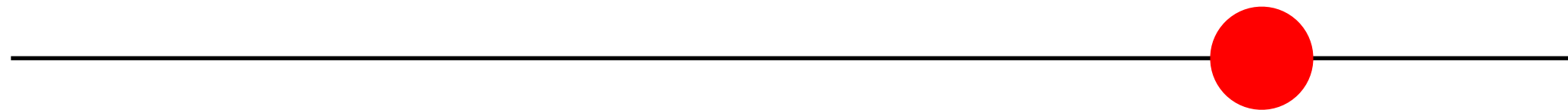


255



Red

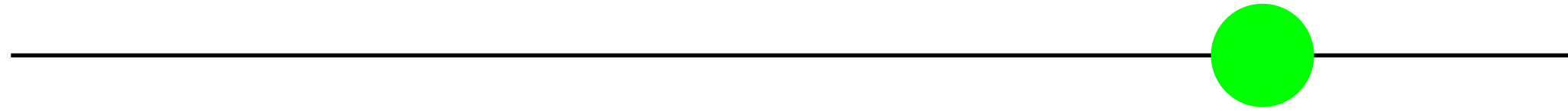
0



255

Green

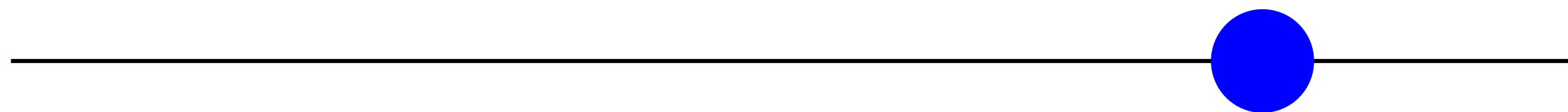
0



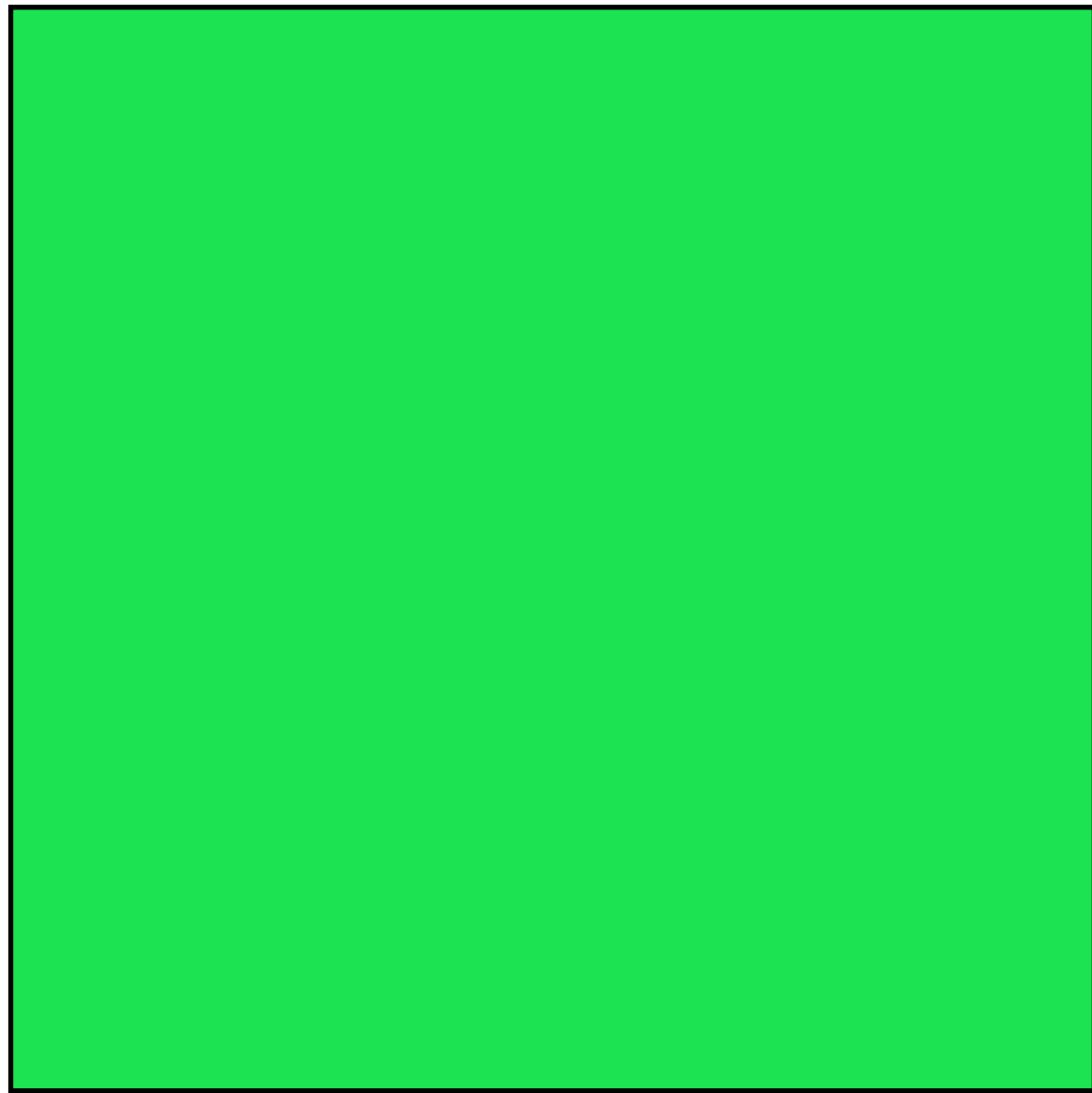
255

Blue

0

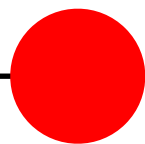


255



Red

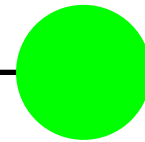
0



255

Green

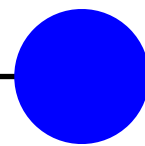
0



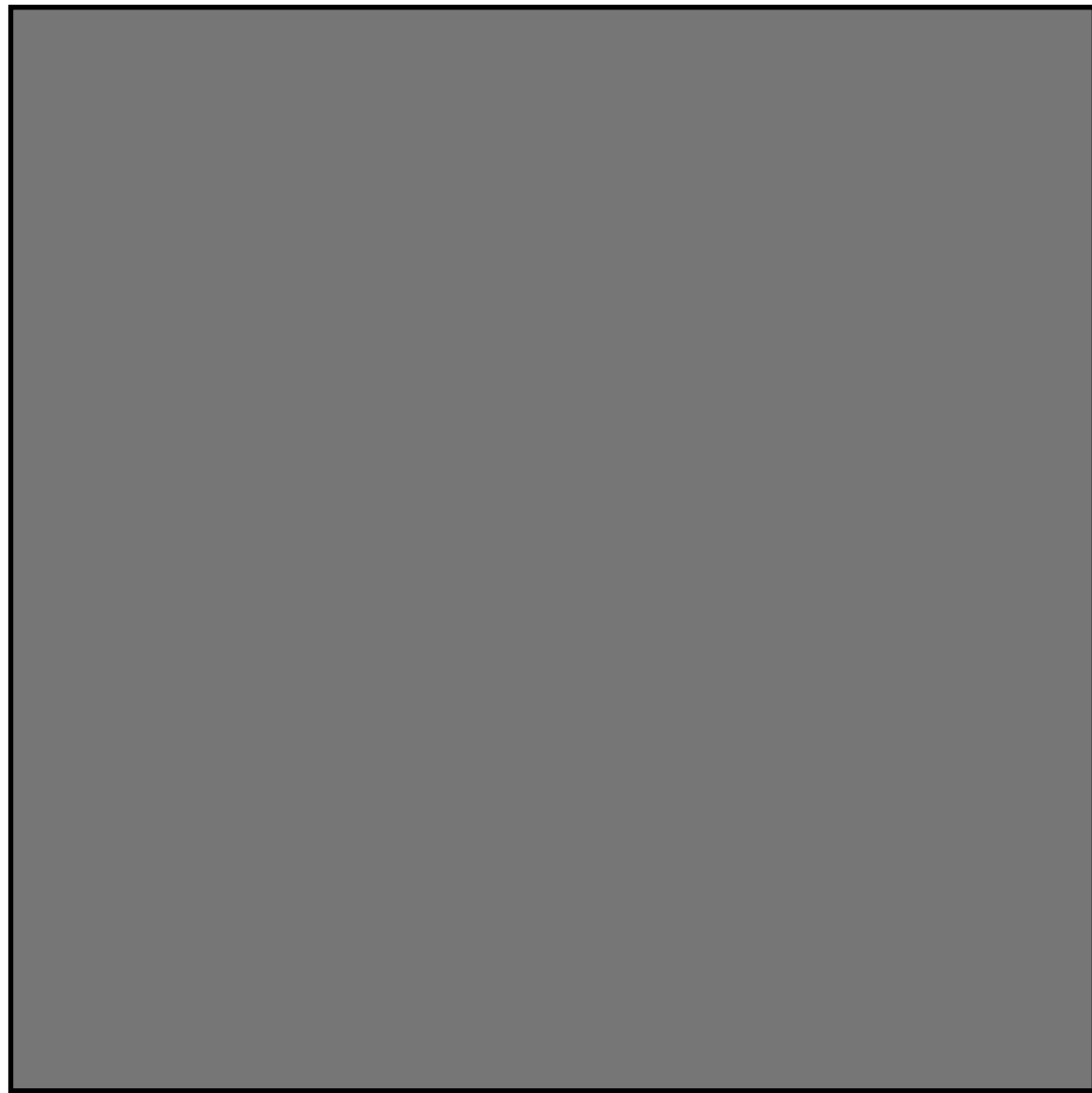
255

Blue

0

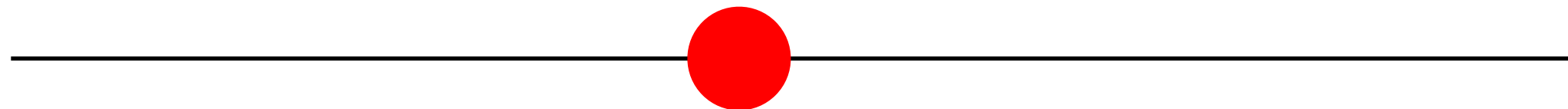


255



Red

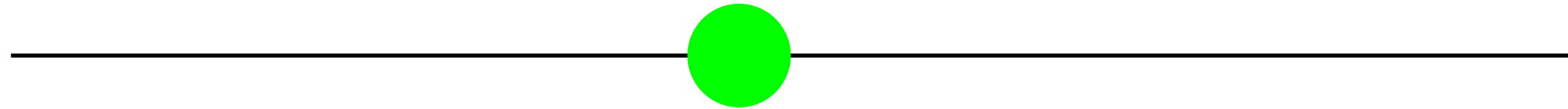
0



255

Green

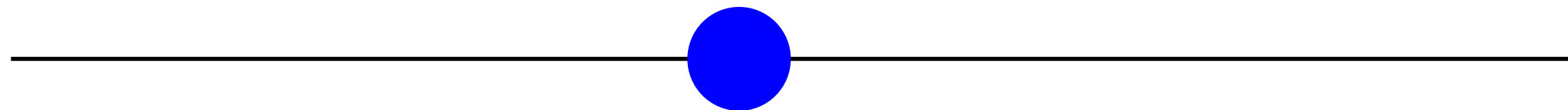
0



255

Blue

0



255



JPEG

0xff								

0xff	0xd8							

0xff	0xd8	0xff						

0xff	0xd8	0xff	0xe0					

0xff	0xd8	0xff	0xe1					

0xff	0xd8	0xff	0xe2					

0xff	0xd8	0xff	0xef					

0xff	0xd8	0xff	0xef					

			0xff 0xd8 0xff 0xe0 ...					

			0xff 0xd8 0xff 0xe0 ...					

			0xff 0xd8 0xff 0xe0 ...					

			0xff 0xd8 0xff 0xe0 ...					

			0xff 0xd8 0xff 0xe0 ...					

			0xff 0xd8 0xff 0xe0 ...				0xff 0xd8 0xff 0xe1 ...	

			0xff 0xd8 0xff 0xe0 ...				0xff 0xd8 0xff 0xe1 ...	

			0xff 0xd8 0xff 0xe0 ...				0xff 0xd8 0xff 0xe1 ...	

			0xff 0xd8 0xff 0xe0 ...				0xff 0xd8 0xff 0xe1 ...	
				0xff 0xd8 0xff 0xe5 ...				
0xff 0xd8 0xff 0xef ...					0xff 0xd8 0xff 0xec ...			
		0xff 0xd8 0xff 0xeb ...					0xff 0xd8 0xff 0xe4 ...	
					0xff 0xd8 0xff 0xe2 ...			

This is CS50.