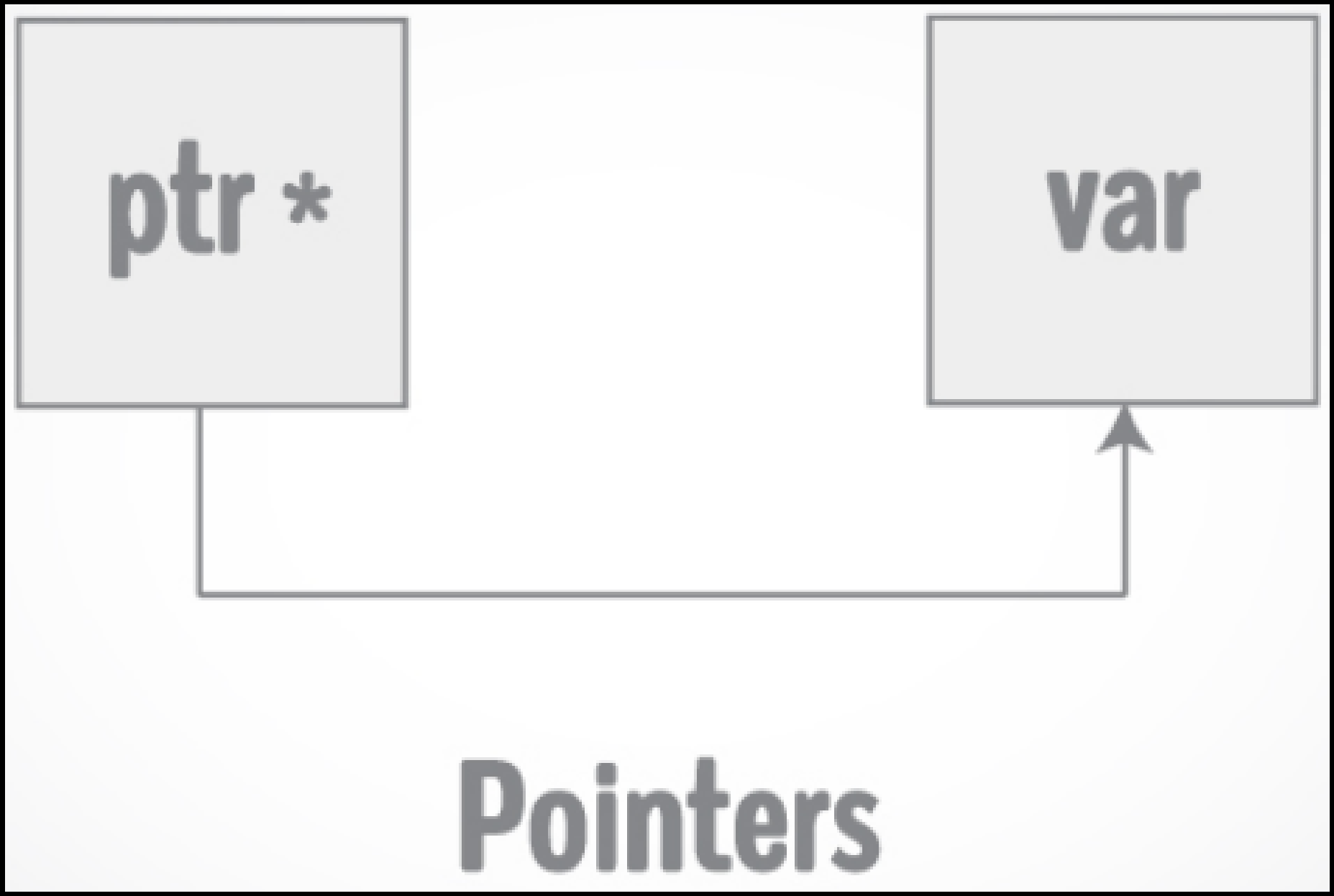
CSE102
Computer Programming

Credits: www.programiz.com

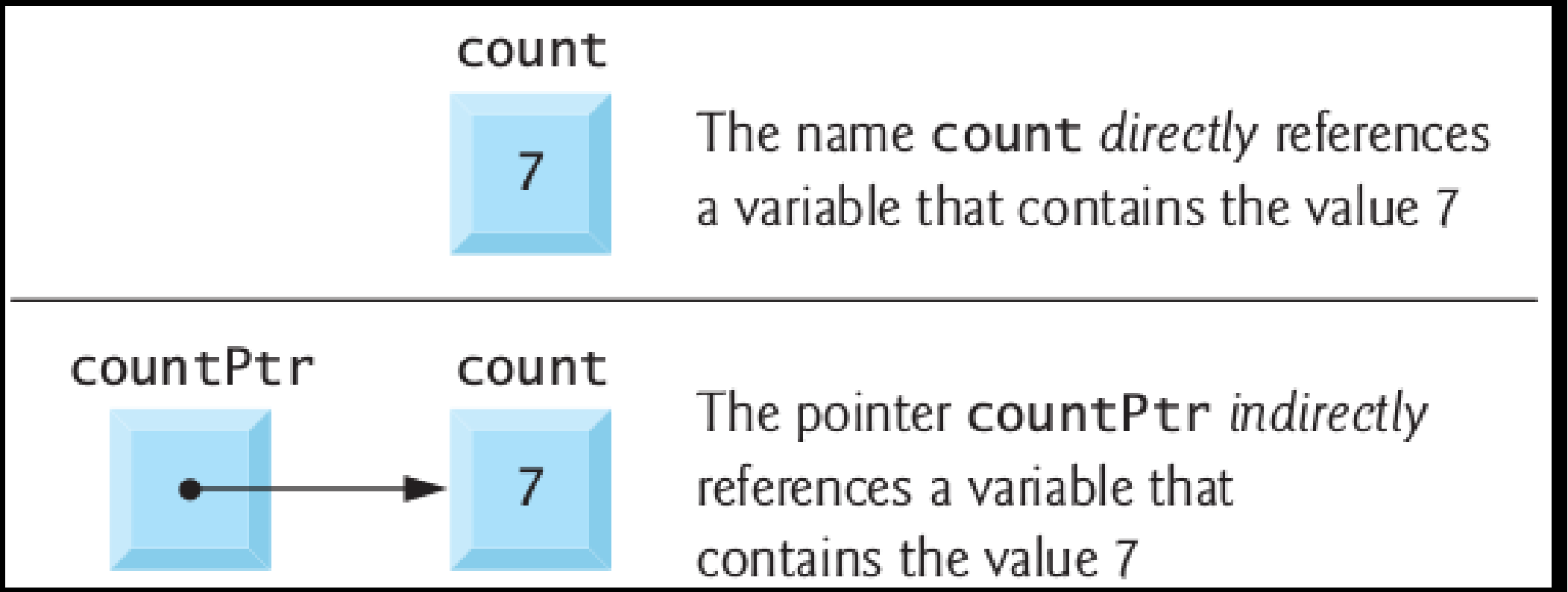
Pointers Point to Variables



Credits: www.programiz.com

Indirect & Direct Reference

```
int *countPtr, count;
```

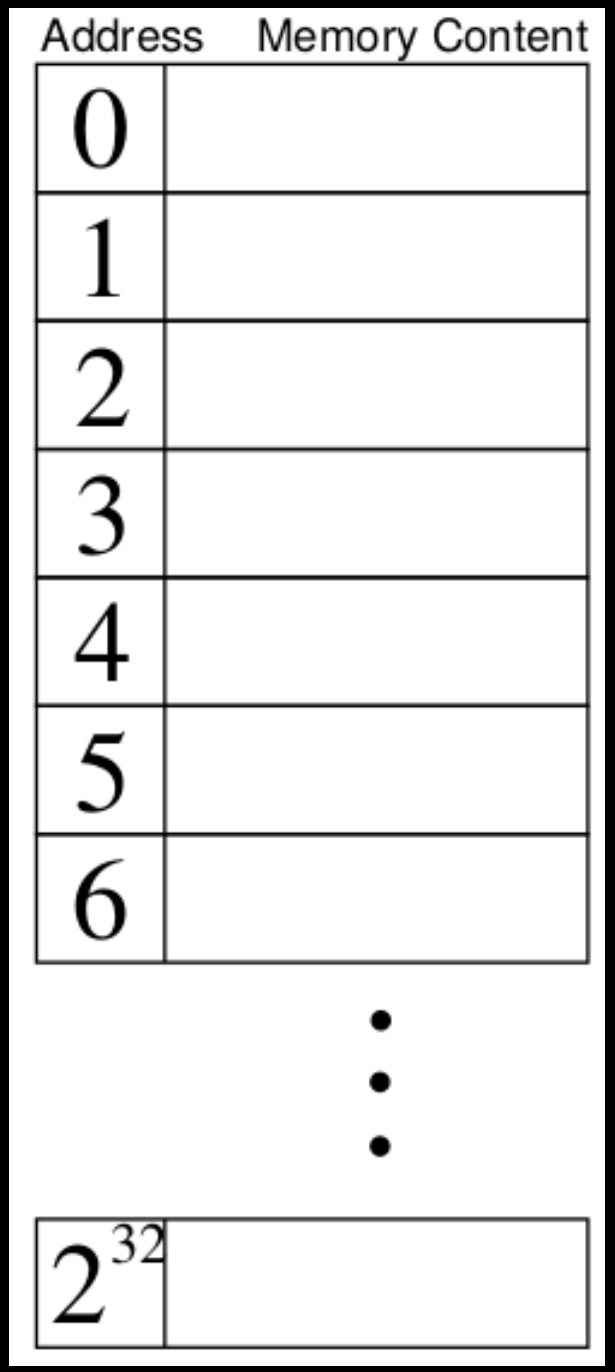


Credits: How to Program C

Declaring Pointers

```
int *countPtr;

// Did you notice the difference between
// declaring a variable and a pointer?

// * says countPtr is a pointer
// int says countPtr is pointing to an
// integer variable
```

Pointers Point Huh!?

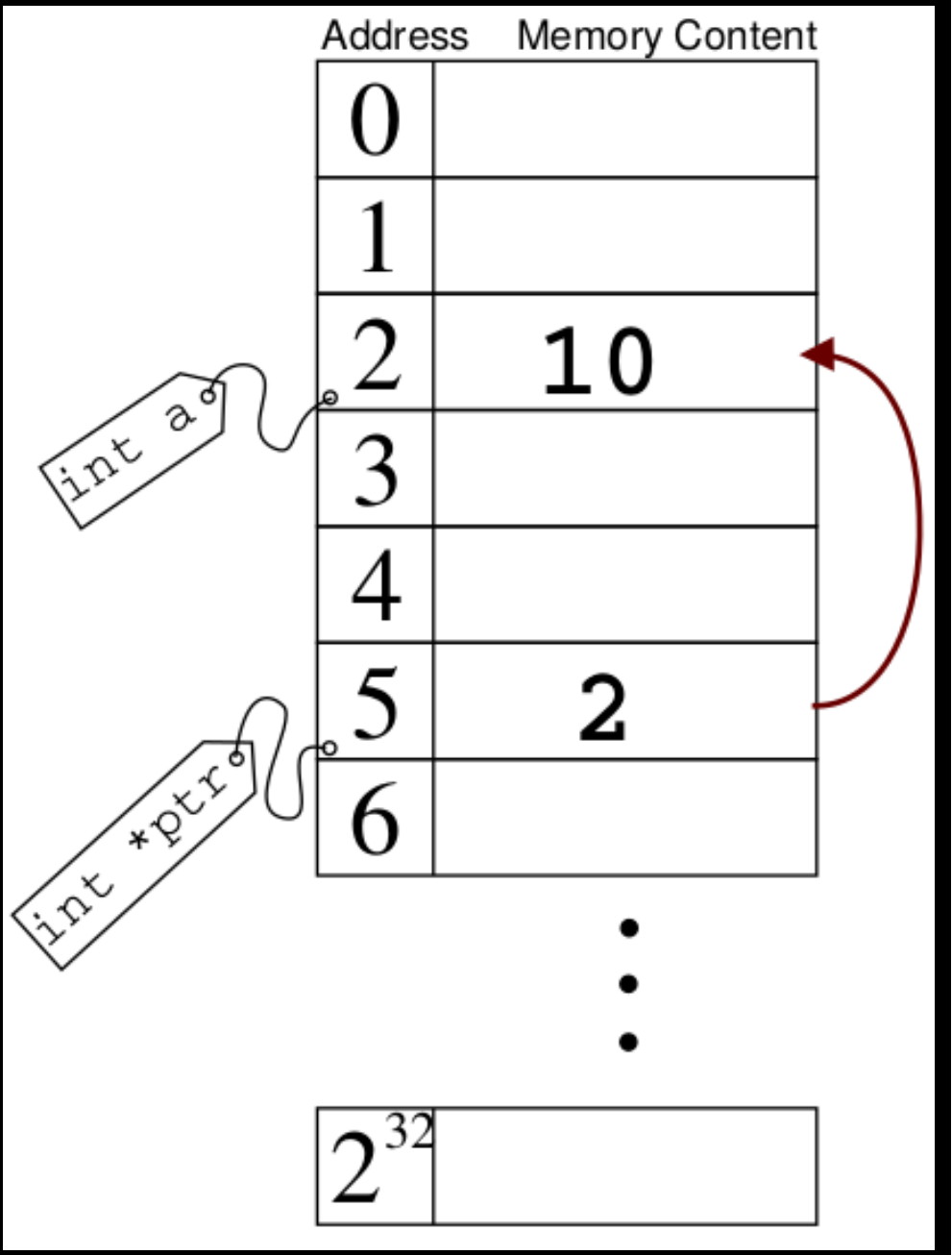


Memory organization in a system with a 32 bit addressing

Credits: Pointers Explained – John Tsiombikas

Pointers Point Huh!?

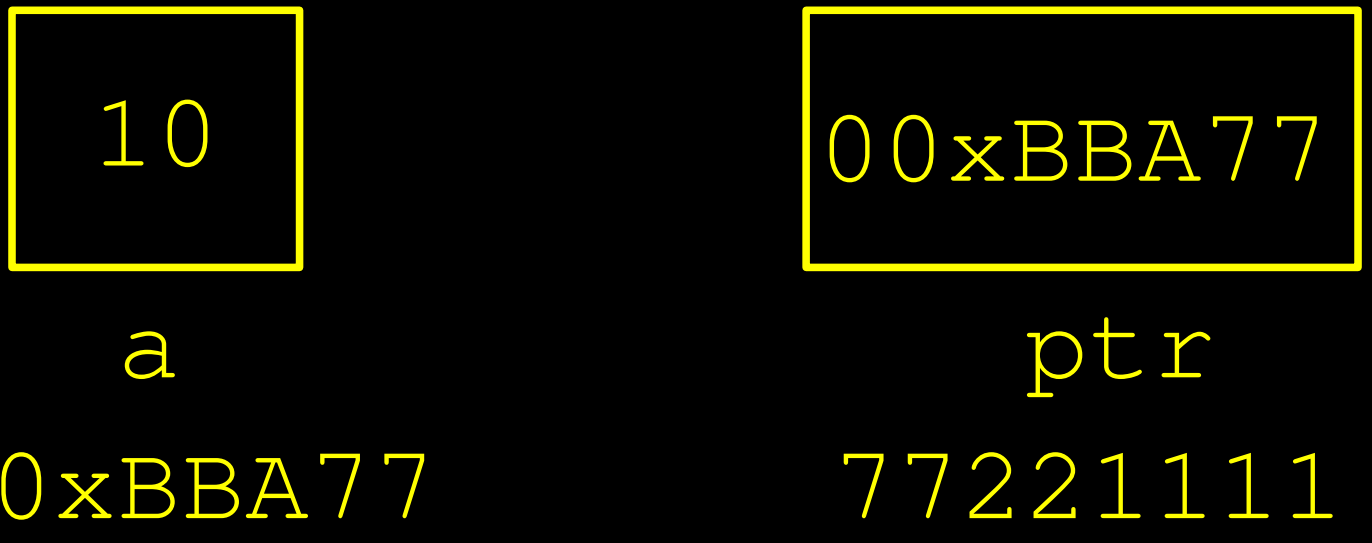
```
int a = 10;

int *ptr;

ptr = &a;
```



Credits: Pointers Explained – John Tsiombikas

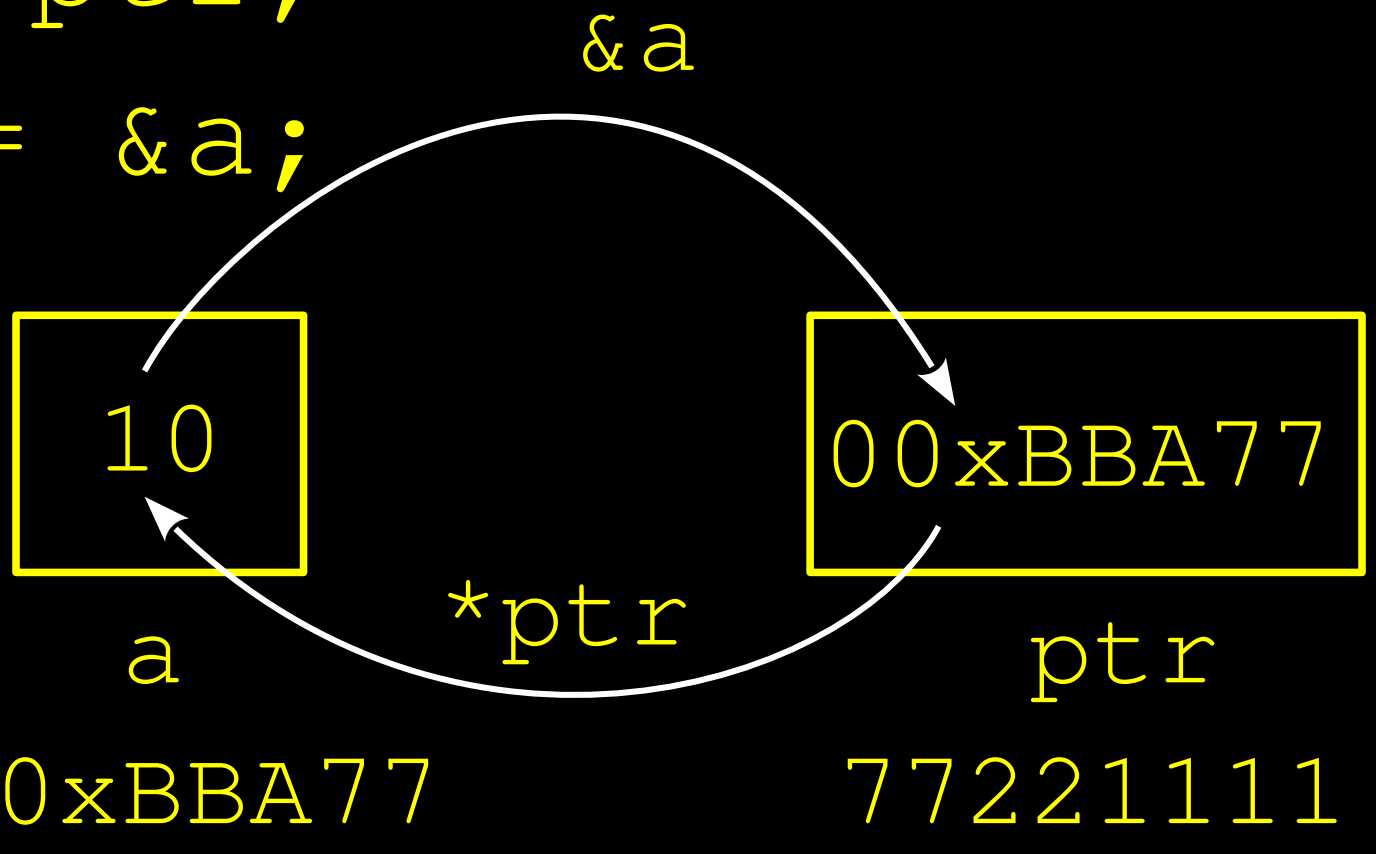
& and * Operators

```
int a = 10;
int *ptr;
ptr = &a;
```



& and * Operators

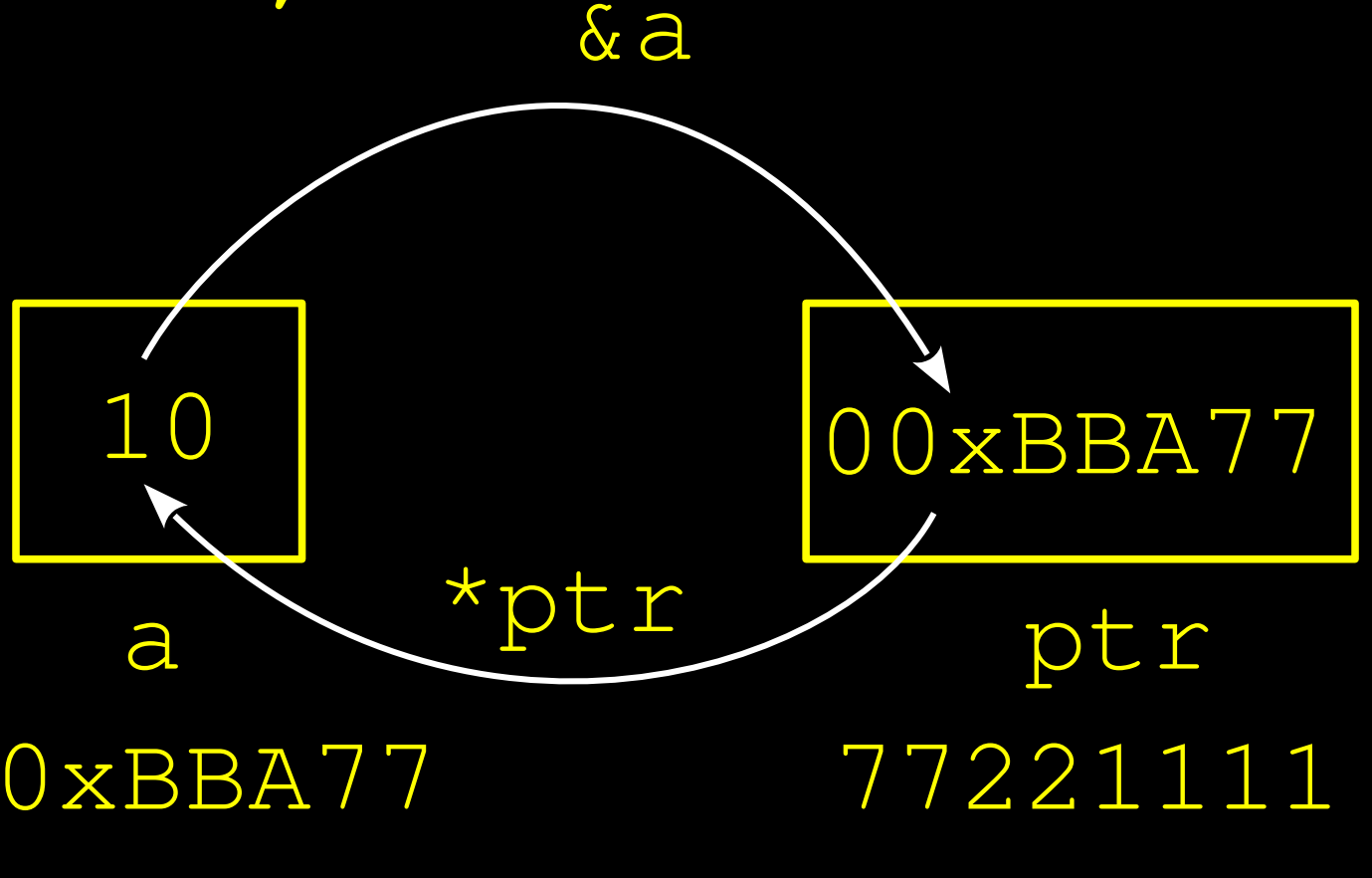
```
int a = 10;
int *ptr;
ptr = &a;
```



& and * Operators

```
int a = 10;
int *ptr;
ptr = &a;
```

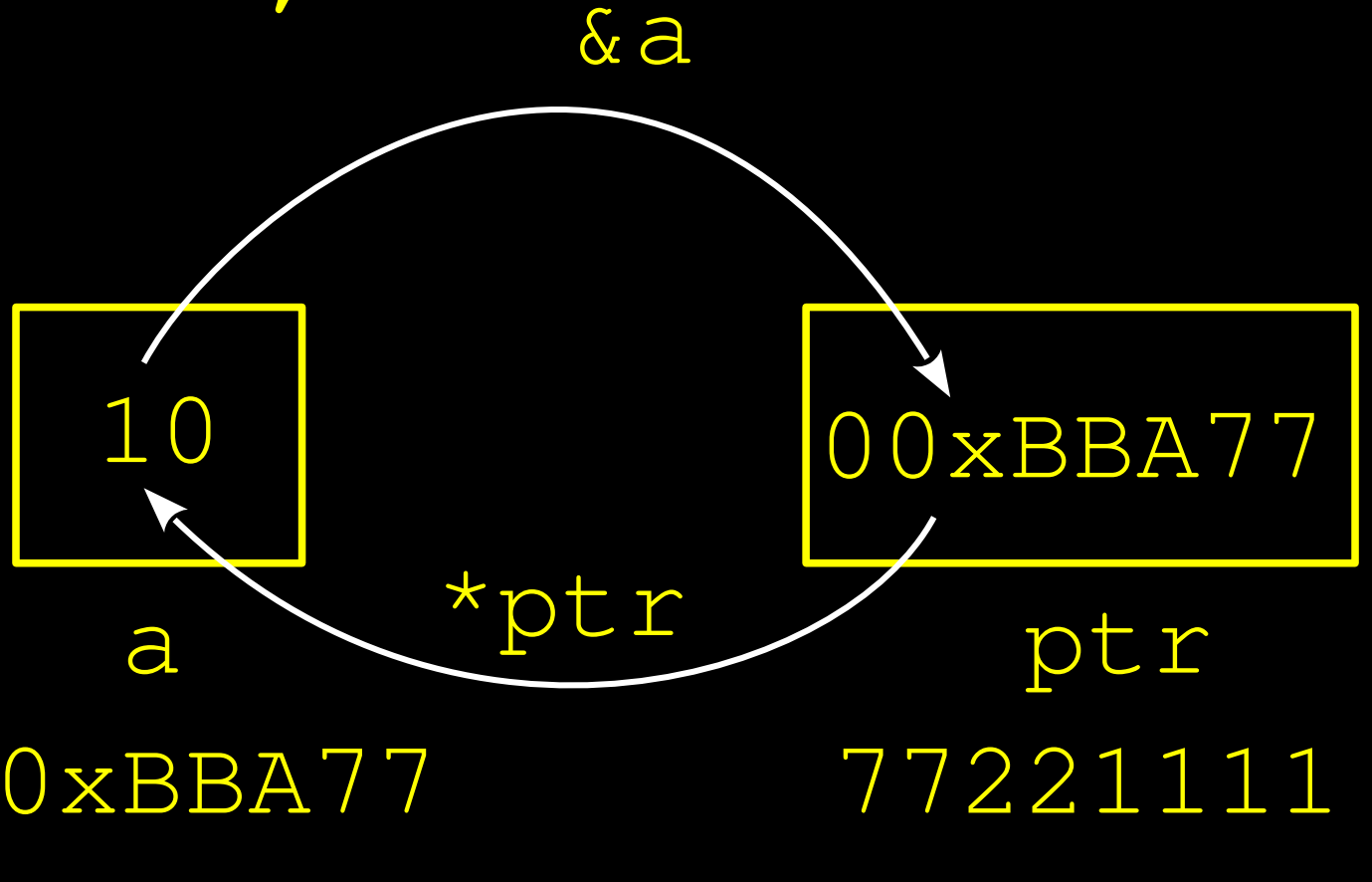
What will be the effect of **`*ptr = 3`**?



& and * Operators

```
int a = 10;
int *ptr;
ptr = &a;
```

The content of a becomes 3!!



Indirect & Direct Reference

```
// 'a' simply refers to the memory
// location alloted to a (direct ref.)

a    = 3;

*ptr = 3;

// *ptr also refers to the memory
// location alloted to a (in-direct ref.)
```

Remember

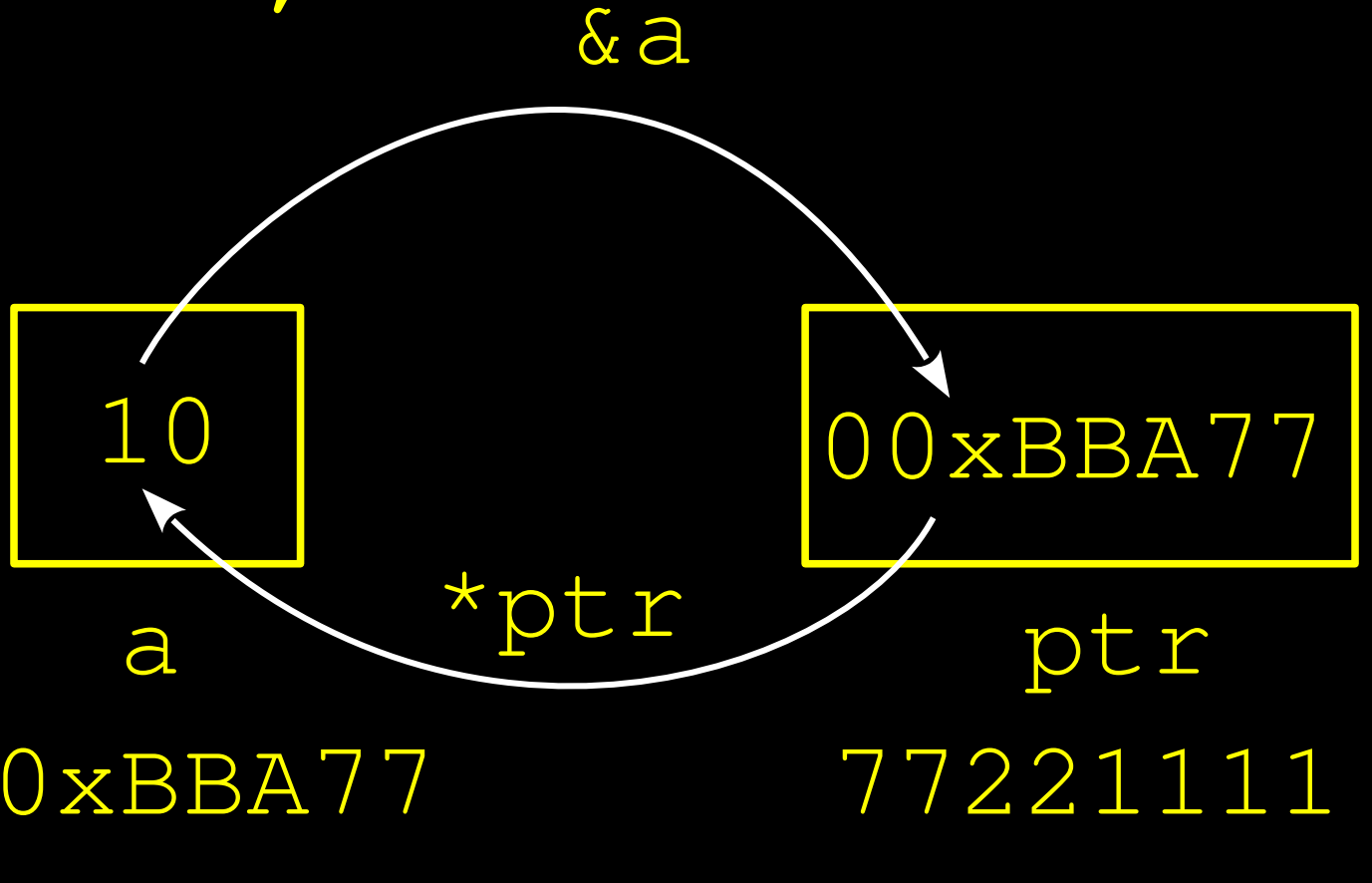
`ptr` means `&a`

`*ptr` means `a`

& and * Operators

```
int a = 10;
int *ptr;
ptr = &a;
```

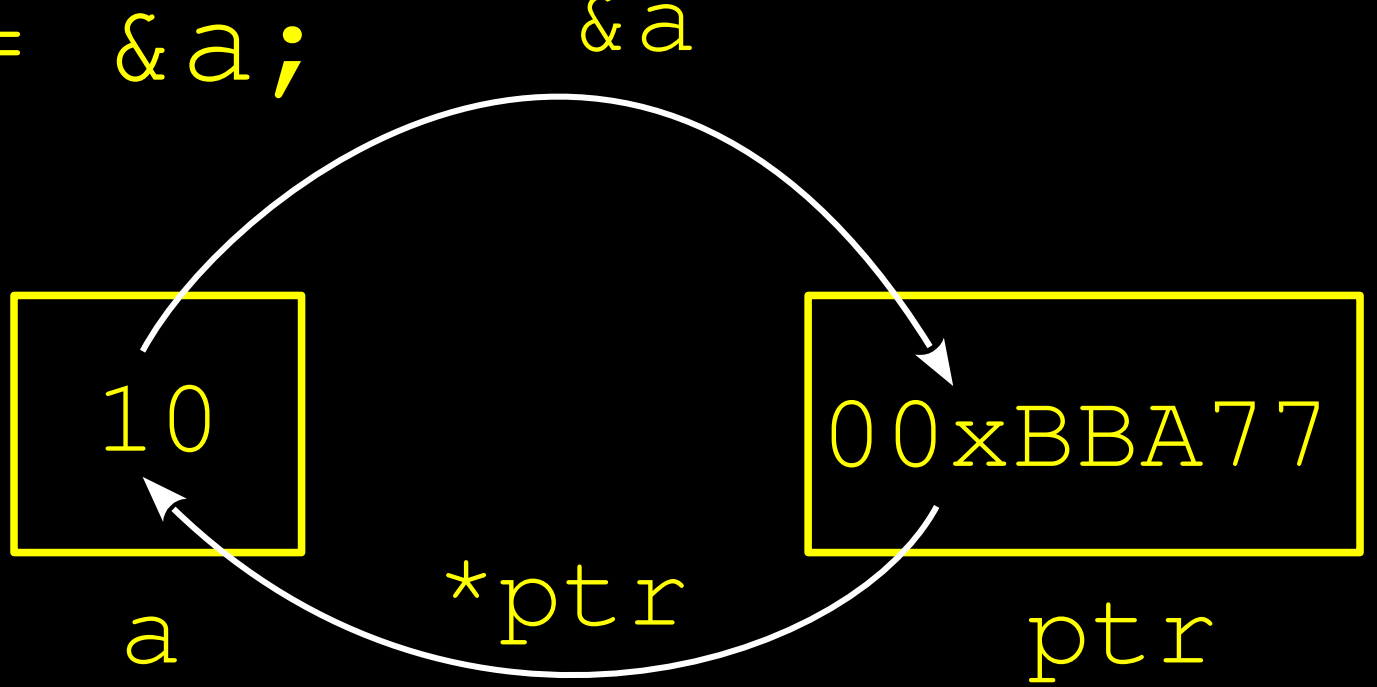
What will be the effect of **&ptr**?



& and * Operators

```
int a = 10;
int *ptr;
ptr = &a;
```

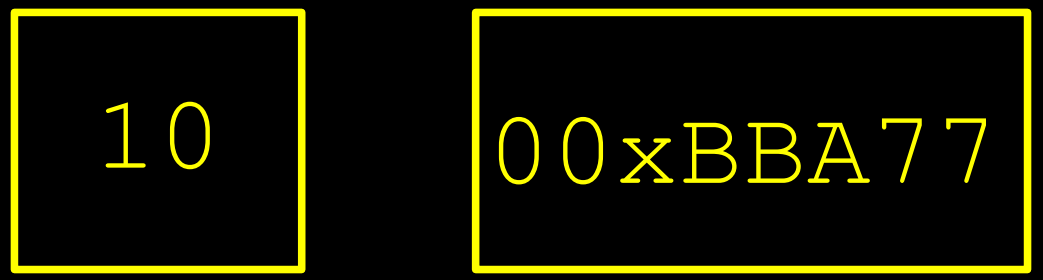
The address of ptr i.e. 77221111 is fetched.



// ptr is variable too and has an address

Multiple Indirection

```
int a = 10;
int *ptr = &a; // Don't confuse
```



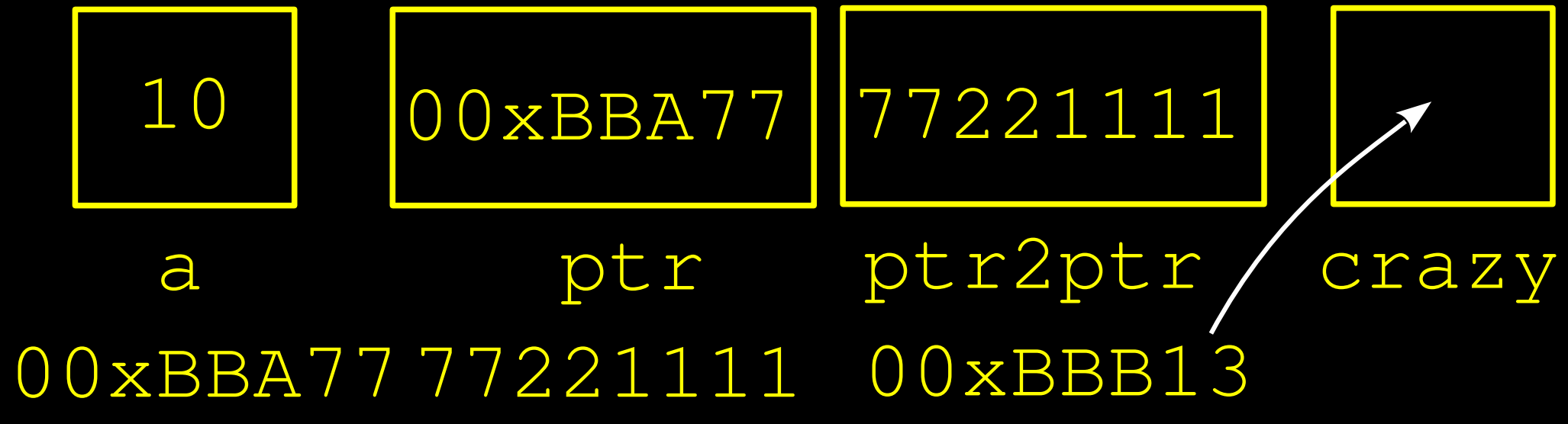
00xBBA77 77221111

Multiple Indirection

```
int a = 10;
int *ptr = &a;
int **ptr2ptr = &ptr;
```

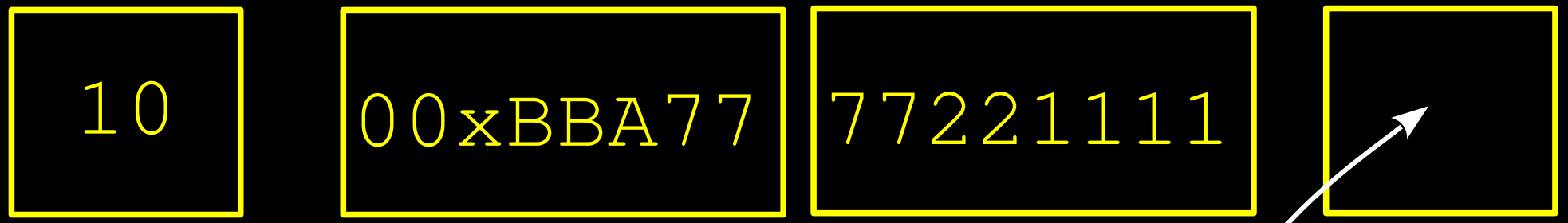
Multiple Indirection

```
int a = 10;
int *ptr = &a;
int **ptr2ptr = &ptr;
int ***crazy = &ptr2ptr
```



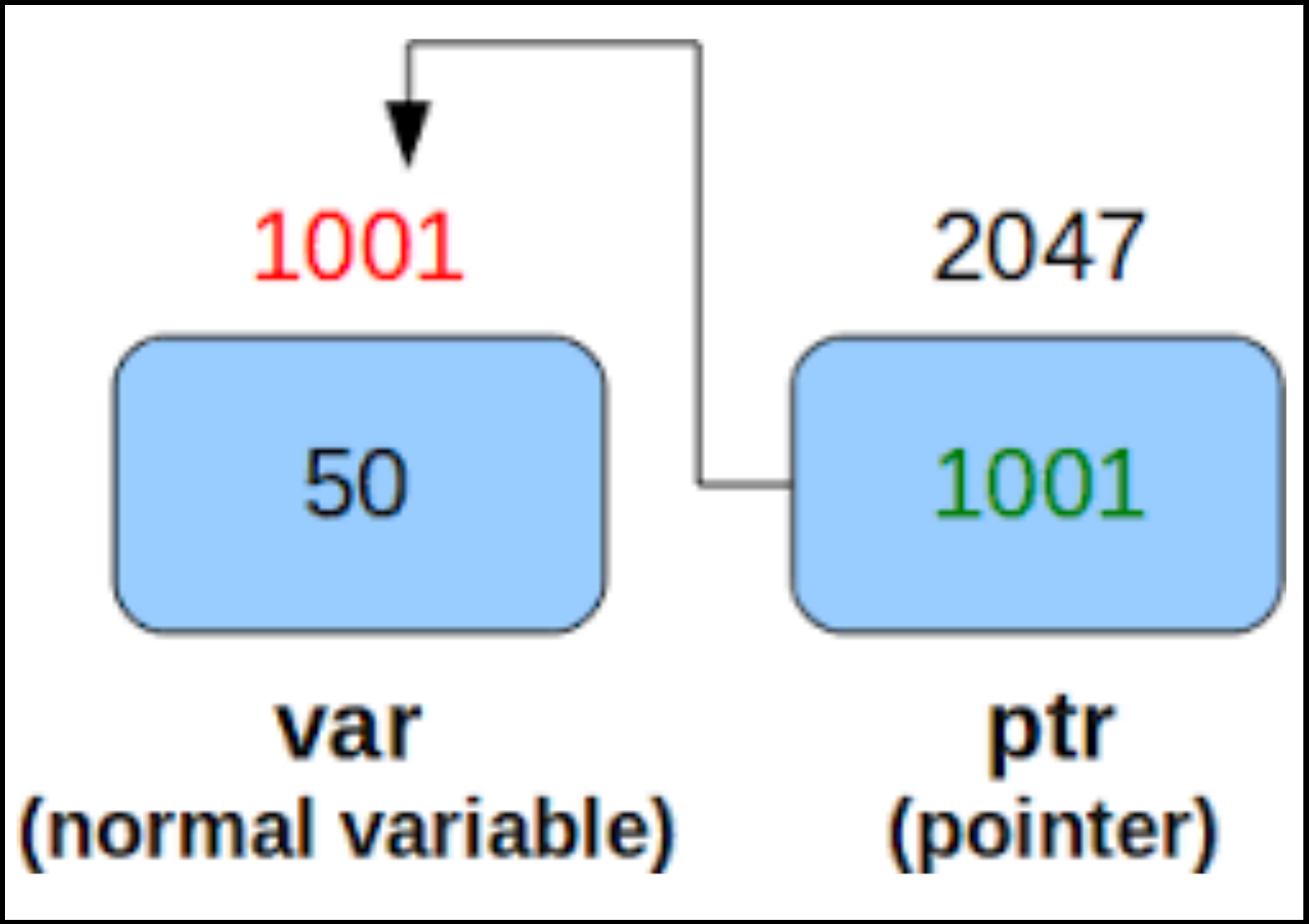
Multiple Indirection

```
int a = 10;
int *ptr = &a;
int **ptr2ptr = &ptr;
int ***crazy = &ptr2ptr
// Remember *crazy = ptr2ptr !!!!
// like *ptr = a !!
```



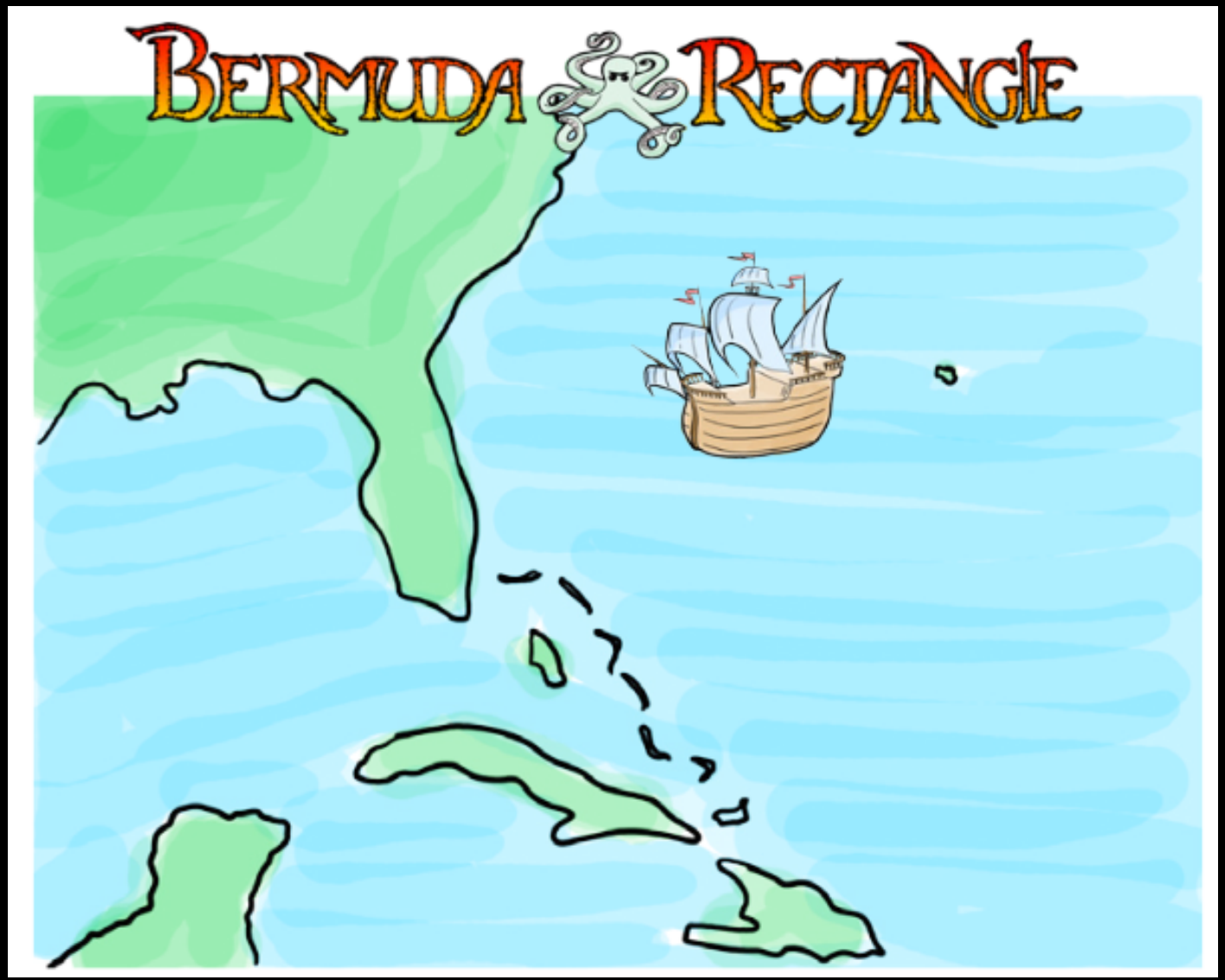
00xBBA77 77221111 00xBBB13

Pointers Point to Variables



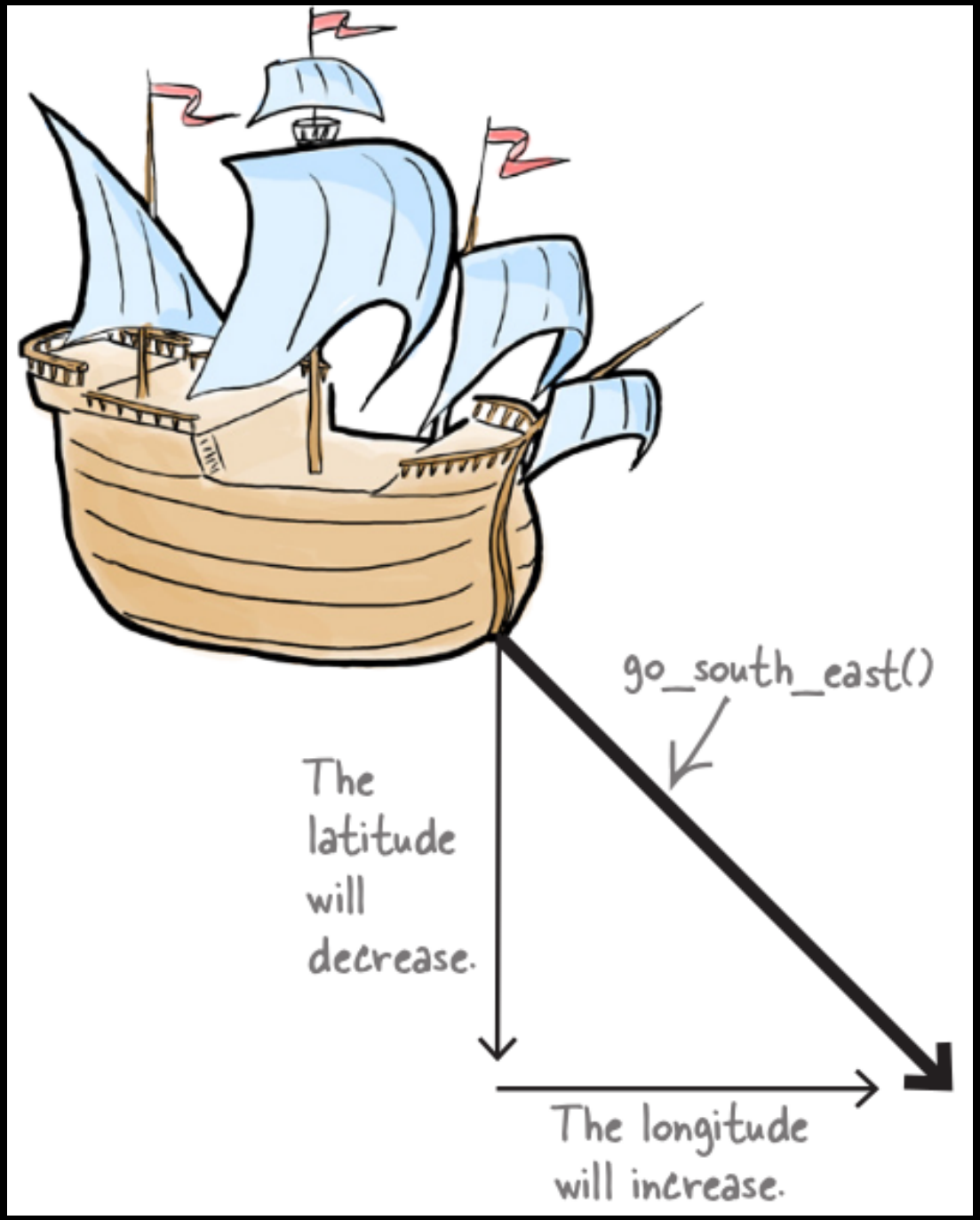
So What?

// Let's write a game for players to
// navigate around the



Credits: Head First C

Did I Say Write a Game!?!?



Nay!!
Let's write a function to set sail in south east!

Credits: Head First C

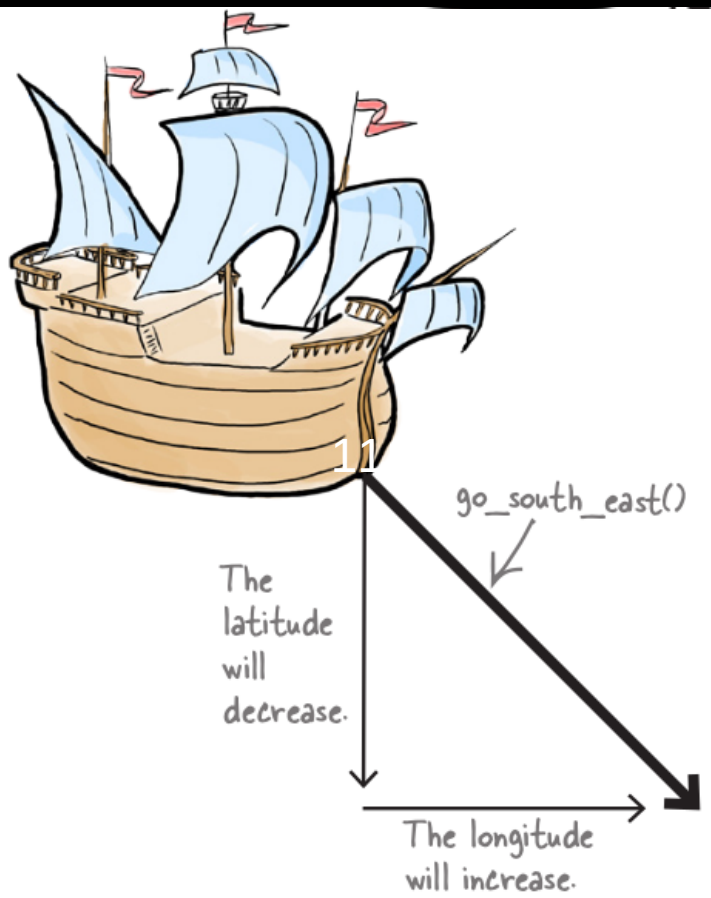
go_south_east

```
#include <stdio.h>


void go_south_east(int lat, int lon)
{
  lat = lat - 1;
  lon = lon + 1;
}

int main()
{
  int latitude = 32;
  int longitude = -64;
  go_south_east(latitude, longitude);
  printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
  return 0;
}
```

Pass in the latitude and longitude.

Decrease the latitude.

Increase the longitude.



Credits: Head First C

Oops!!

The code should move the ship southeast from [32, –64] to the new location at [31, –63]. But if you compile and run the program, this happens:

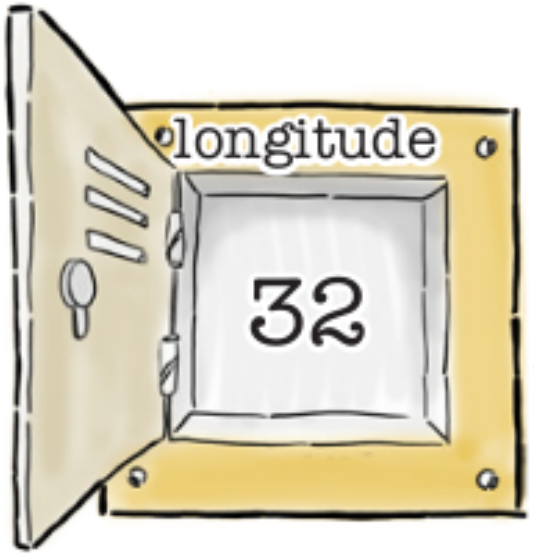
File Edit Window Help Savvy?

```
> gcc southeast.c -o southeast
> ./southeast
Avast! Now at: [32, -64]
>
```

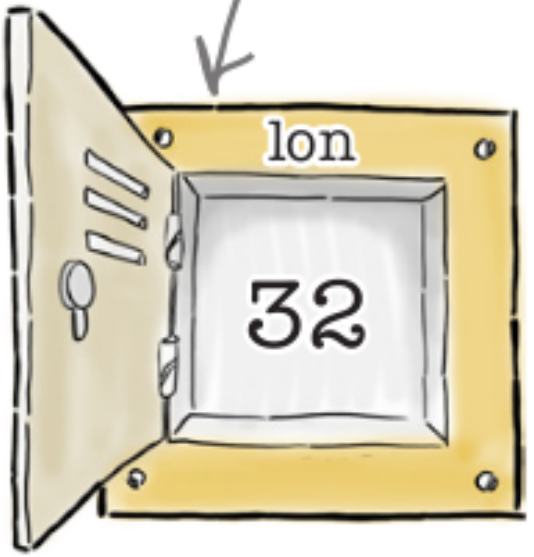
Credits: Head First C

What Went Wrong?

Remember call by value!!



This is a new variable containing a copy of the longitude value.

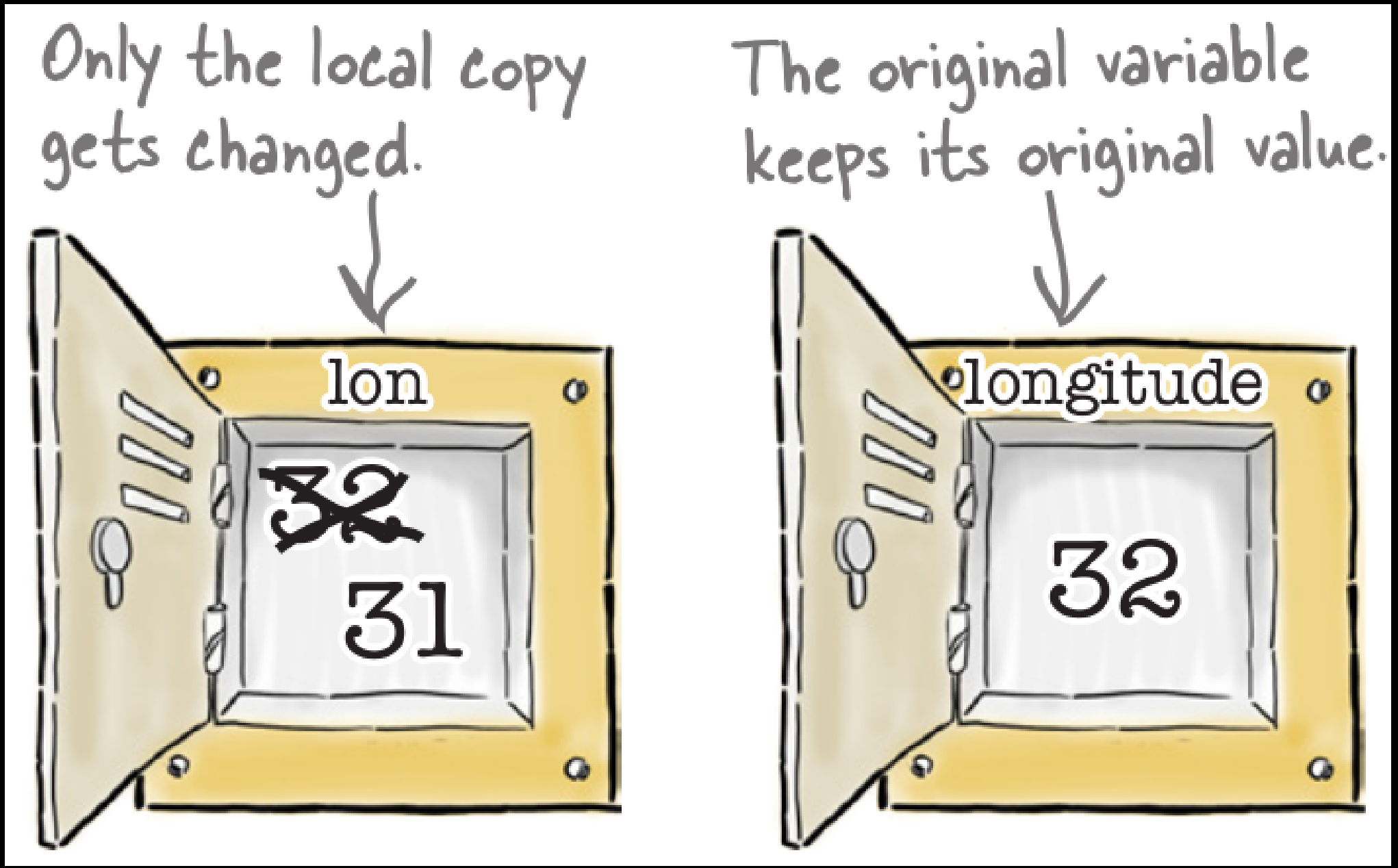


When you call a function you don't send the variable longitude to the function go_south_east() just its value 32 copied to lon

Credits: Head First C

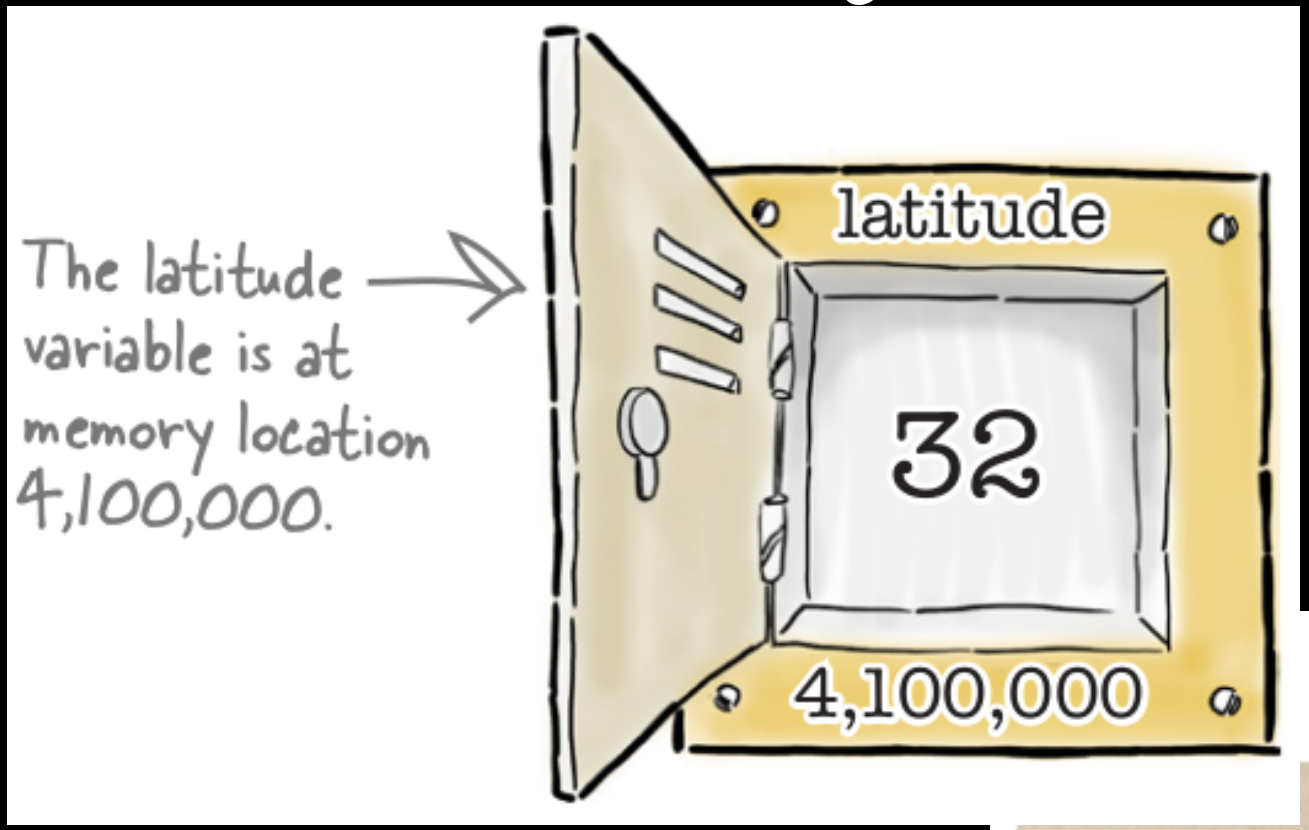
What Went Wrong?

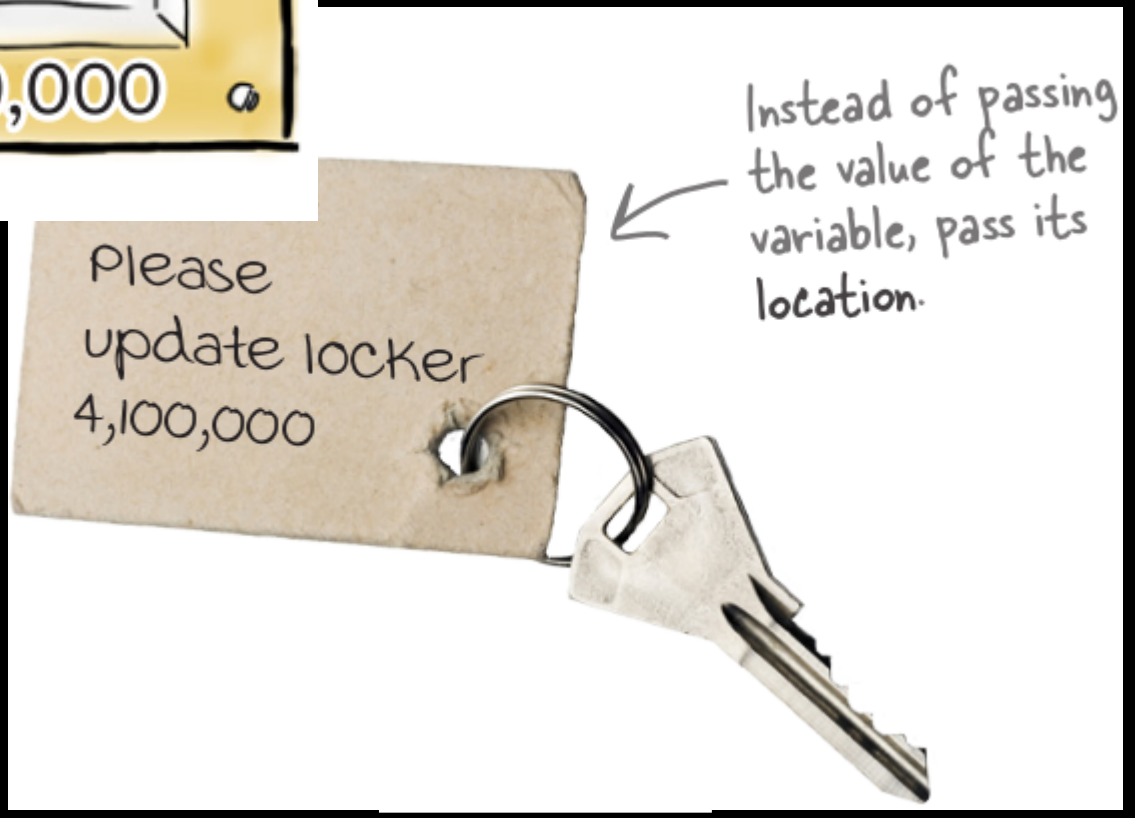
Remember call by value!!



Credits: Head First C

Call by Reference





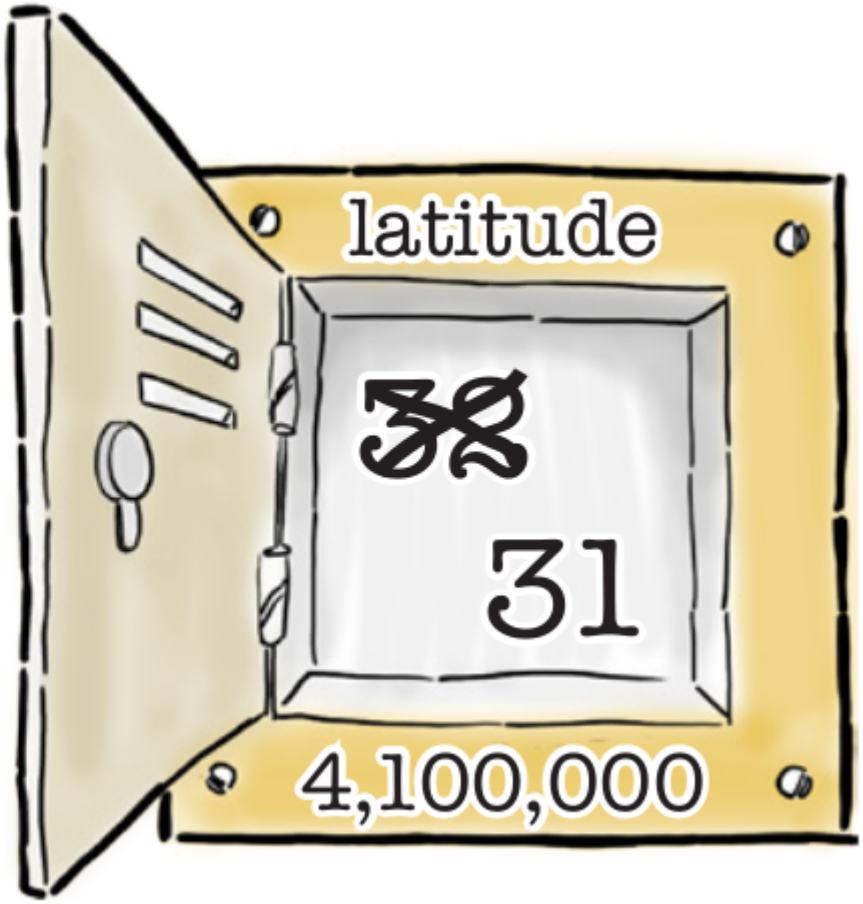
Credits: Head First C

Call by Reference

~~Read contents of memory 4,100,000~~

~~Subtract 1 from value~~

~~Store new value in memory 4,100,000~~



Credits: Head First C

Fixing go_south_east()

```
// The parameters are of pointer type
void go_south_east(int *lat,
                   int *lon){
  *lat = *lat - 1;
  *lon = *lon - 1;
}
```

*lat & *lon are de-referencing and can access values of latitude and longitude in main, in other words *(&a) is a itself!!

Fixing `main()`

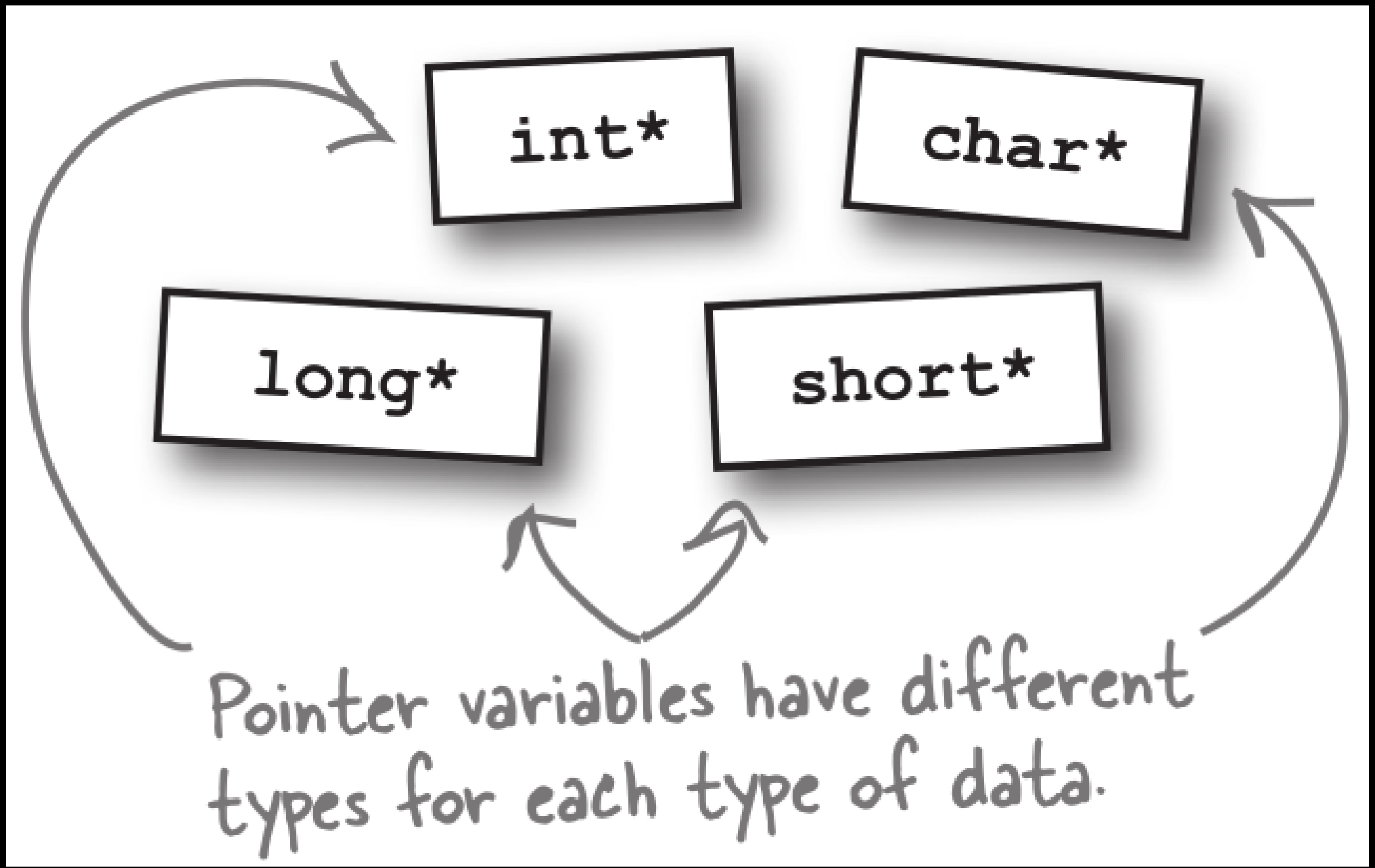
```
int main(){
  int latitude = 32;
  int longitude = -64;
  go_south_east(&latitude, &longitude);
  printf("Avast! Now at: [%i %i]\n",
         latitude, longitude);
  return 0;
}
```

```
File Edit Window Help Savvy?
> gcc southeast.c -o southeast
> ./southeast
Avast! Now at: [31, -63]
>
```

Share Memory

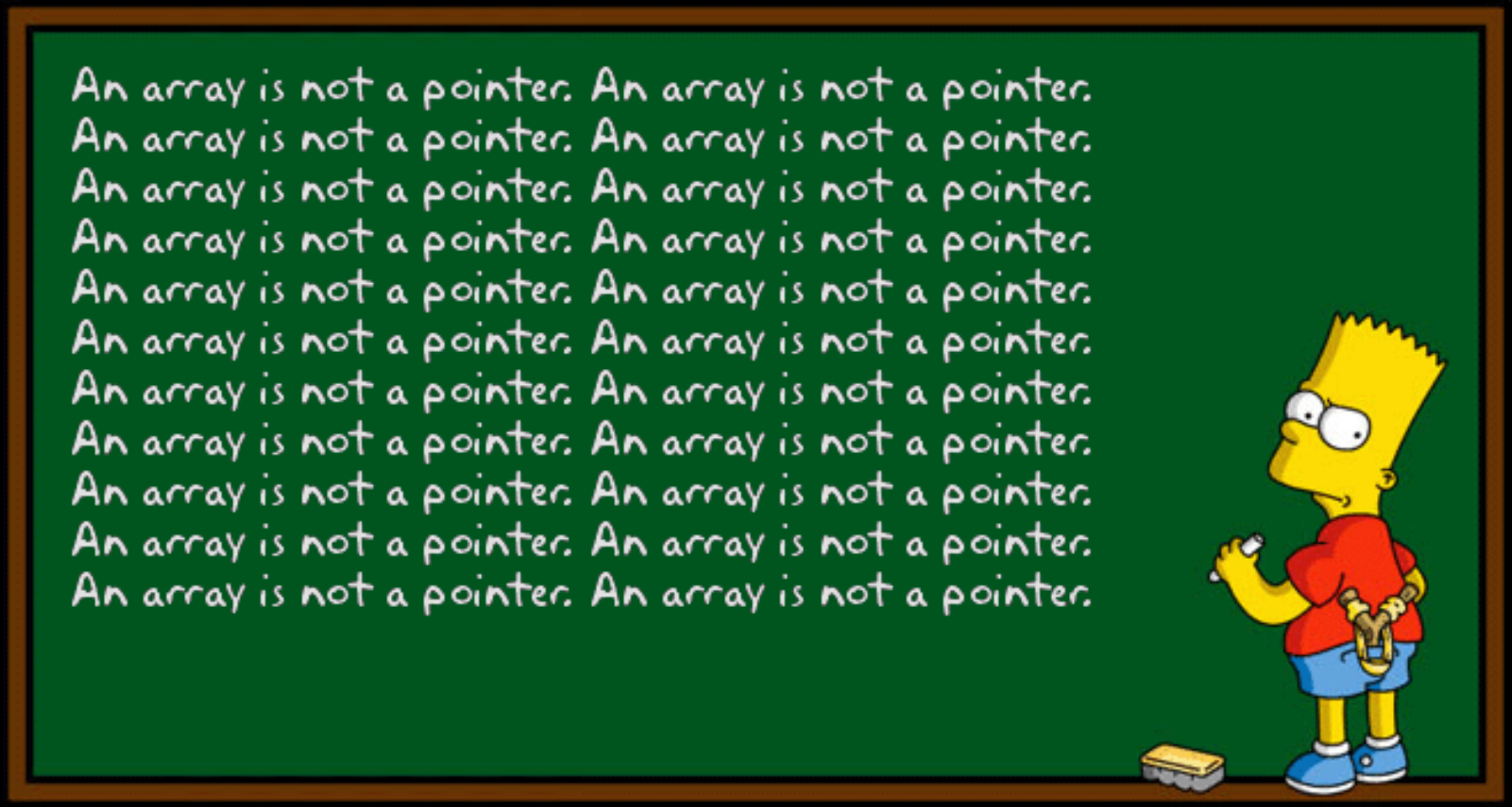
Pointers let functions share memory - data created by one function can be modified by another function, so long as it knows where to find it in memory

Pointers have Types



Credits: Head First C

CSE102
Computer Programming
(Next Topic)



Credits: i.imgur.com