

ED1021 - Introduction to computation and visualisation

L11 - Pointers in C (Very Very..... IMP)

Ramanathan Muthuganapathy (<https://ed.iitm.ac.in/~raman>)

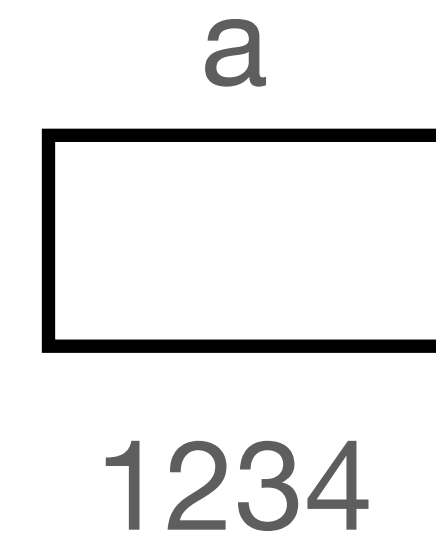
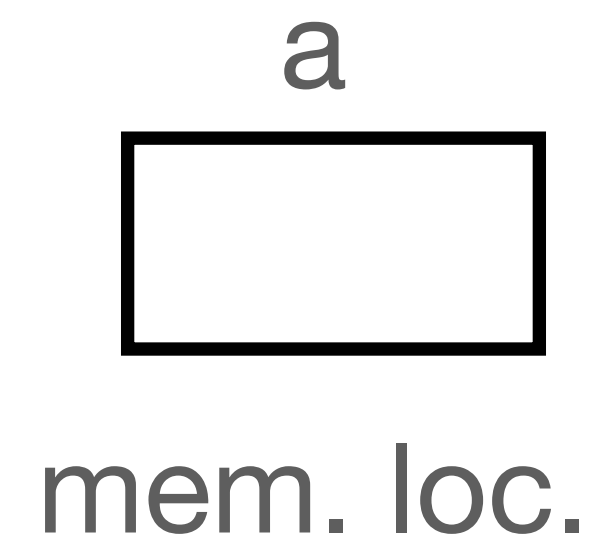
Course web page: <https://ed.iitm.ac.in/~raman/introcomp.html>

Moodle page: Available at <https://courses.iitm.ac.in/>

Declaring variables

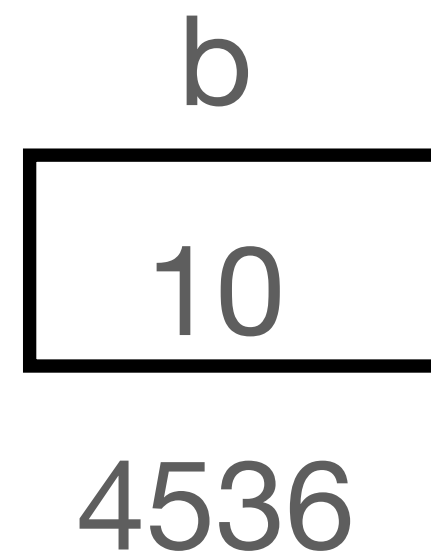
Understanding pointers in C by Kanetkar.

- `int a;`



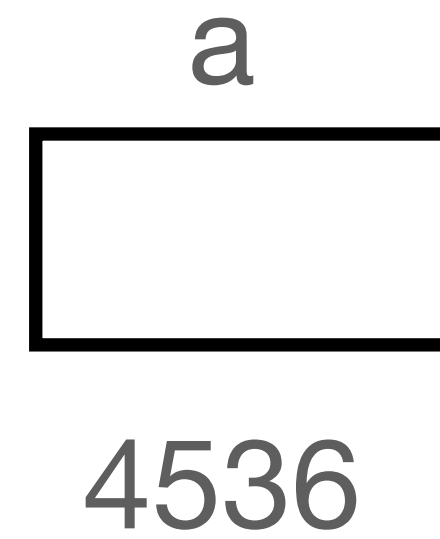
Defining variables

- `int b = 10;`

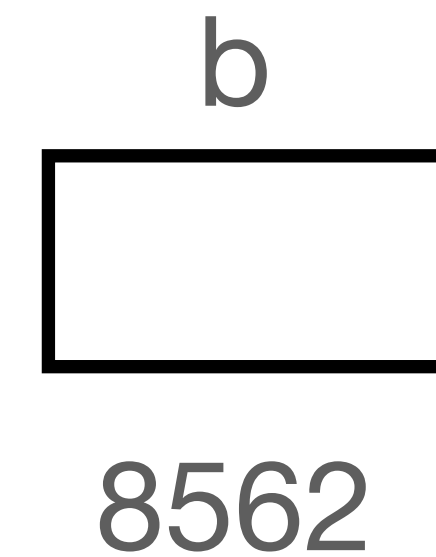


Declaring variables

int a;



float b;

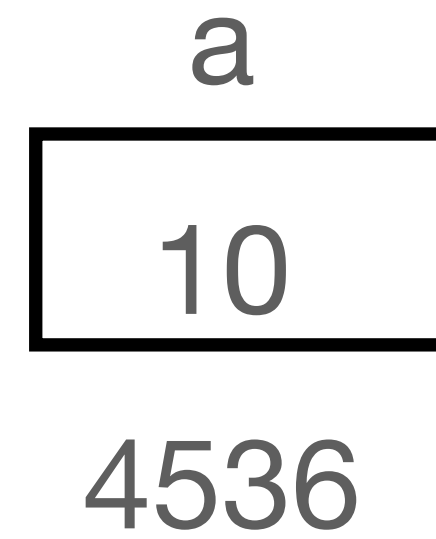


char ch;

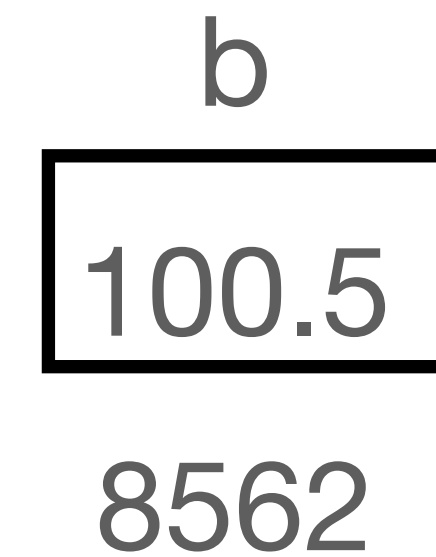


Defining variables

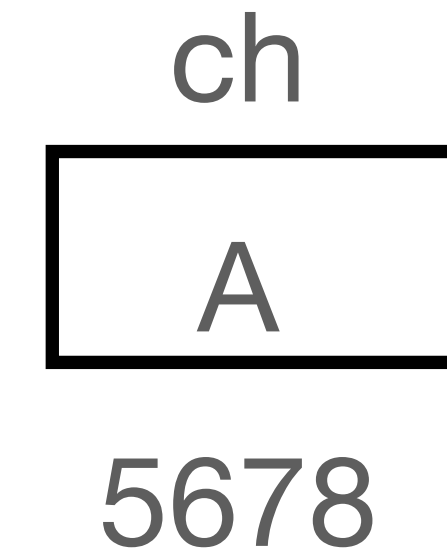
```
int a = 10;
```



```
float b = 100.5;
```

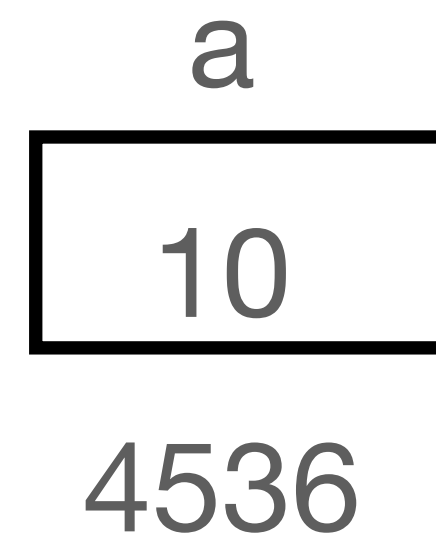


```
char ch = 'A';
```



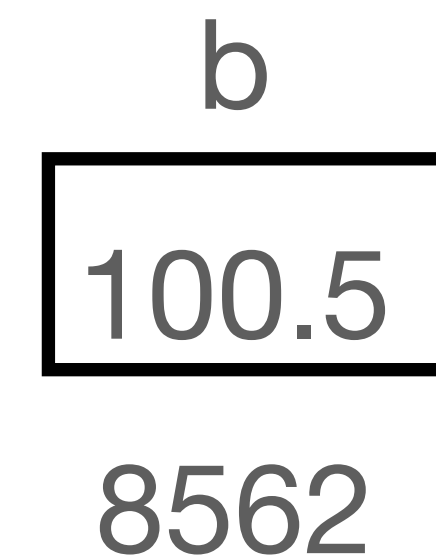
Key observation

`int a = 10;`

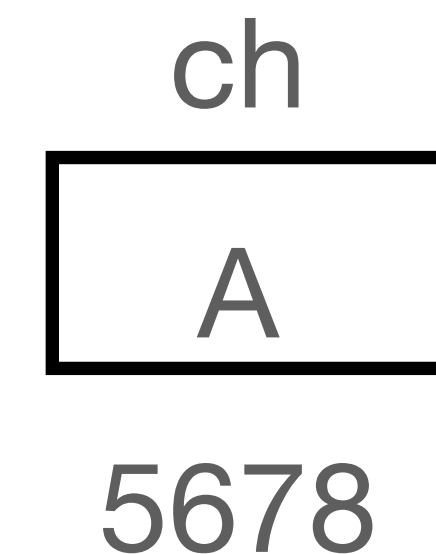


Irrespective of the datatype, the address or mem. location is always an integer value

`float b = 100.5;`



`char ch = 'A';`



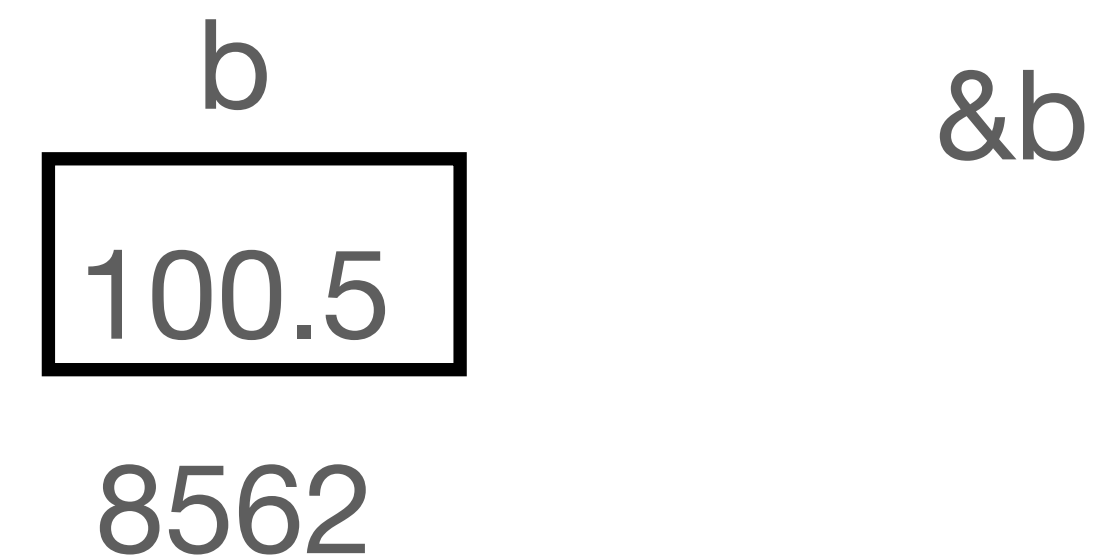
How to get this address of a variable?

using 'address of' operator , i.e. using '&'

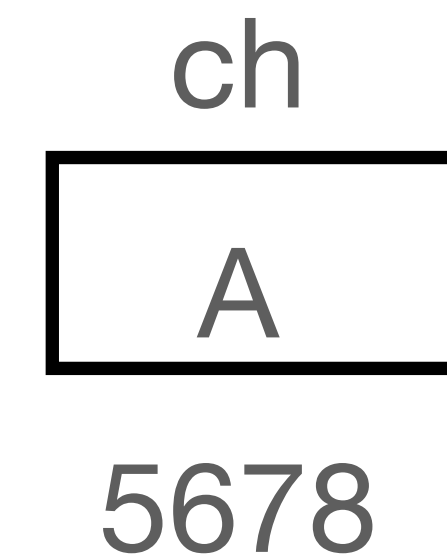
- `int a = 10;`



- `float b = 100.5;`



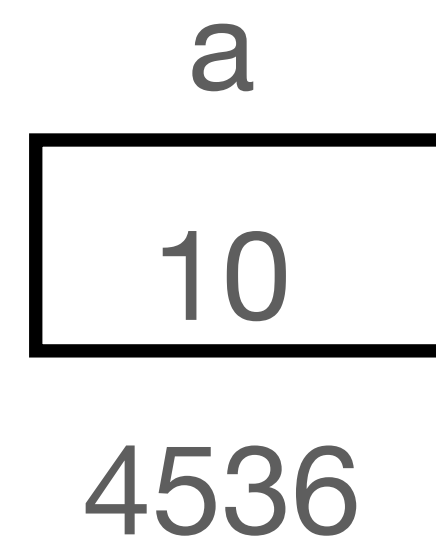
- `char ch = 'A';`



How to print this address of a variable?

using ‘&’ with format specifier

- `int a = 10;`
- `printf(“%u \n”, &a);`



`&a`

You can only ‘fetch’ or print the address of a variable.
The address is assigned by the system not by you.
You cannot assign an address
(at a very sophisticated level, this can be done);

CW: 1) Post your output in the chat box.
2) `printf` statement combining value for `a` and its address.

How to print this address of a variable?

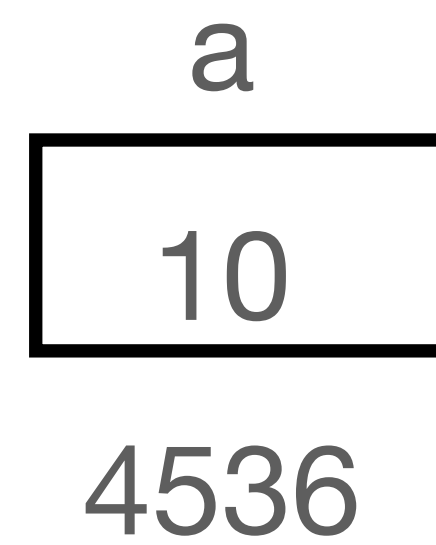
using ‘&’ with format specifier

```
int a = 10;
```

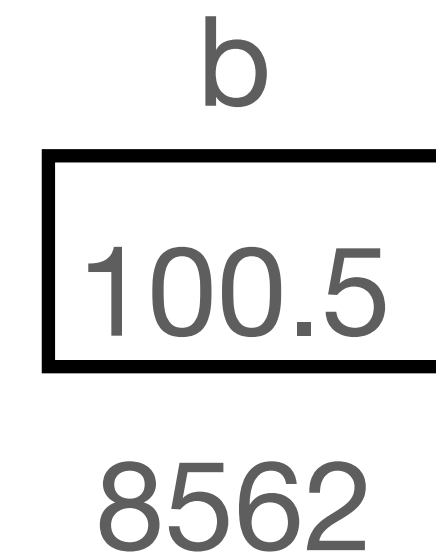
```
float b = 100.5;
```

```
char ch = 'A';
```

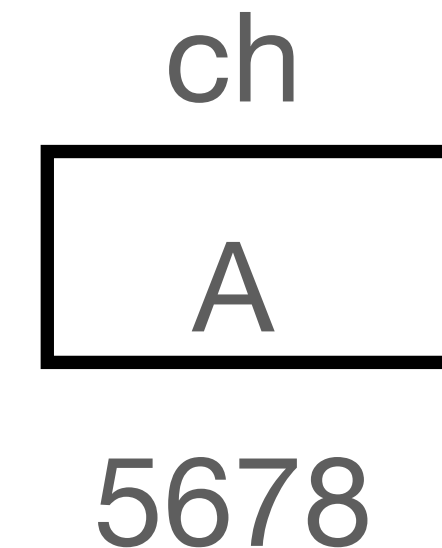
```
printf(“%u %u %u\n”, &a,  
&b, &ch);
```



&a



&b



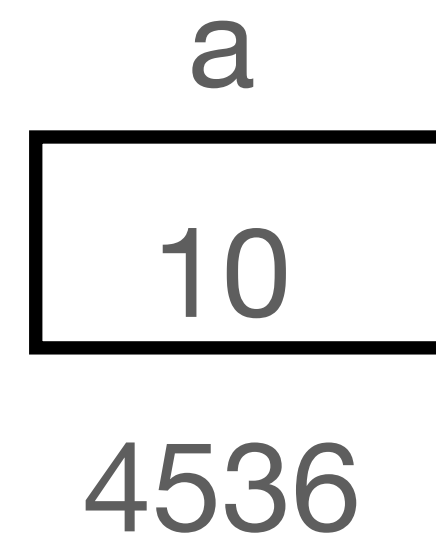
CW: Update your previous printf and post your code and output in the chat box.

How to store (or assign) this address of a variable?

Can a 'normal' scalar variable be used?

```
int a = 10, d;
```

```
d = &a;
```



&a

You can only 'fetch' or print the address of a variable.
The address is assigned by the system not by you.
You cannot assign an address
(at a very sophisticated level, this can be done);

CW: 1) Post your output in the chat box.
2) printf statement combining value for a and its address.

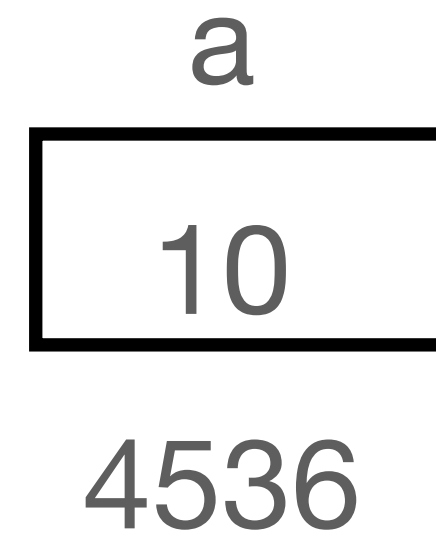
How to store (or assign) this address of a variable?

Pointer variable

```
int a = 10;
```

```
int *d;
```

```
d = &a;
```

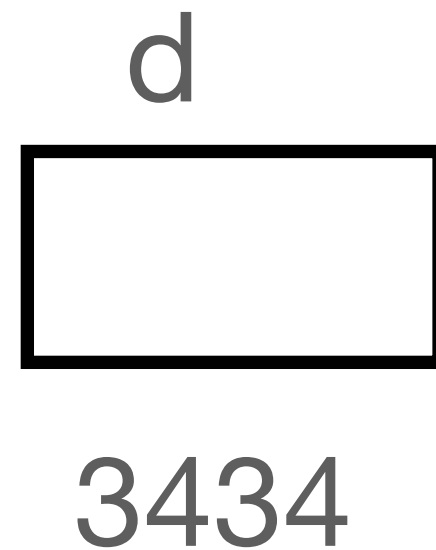


Declaring and its meaning of a pointer variable

You will see stars now!

```
int *d;
```

d is a variable that can store only the address of an integer variable.



CW: what is the difference between the following two declarations?

```
int d;
```

```
int *d;
```

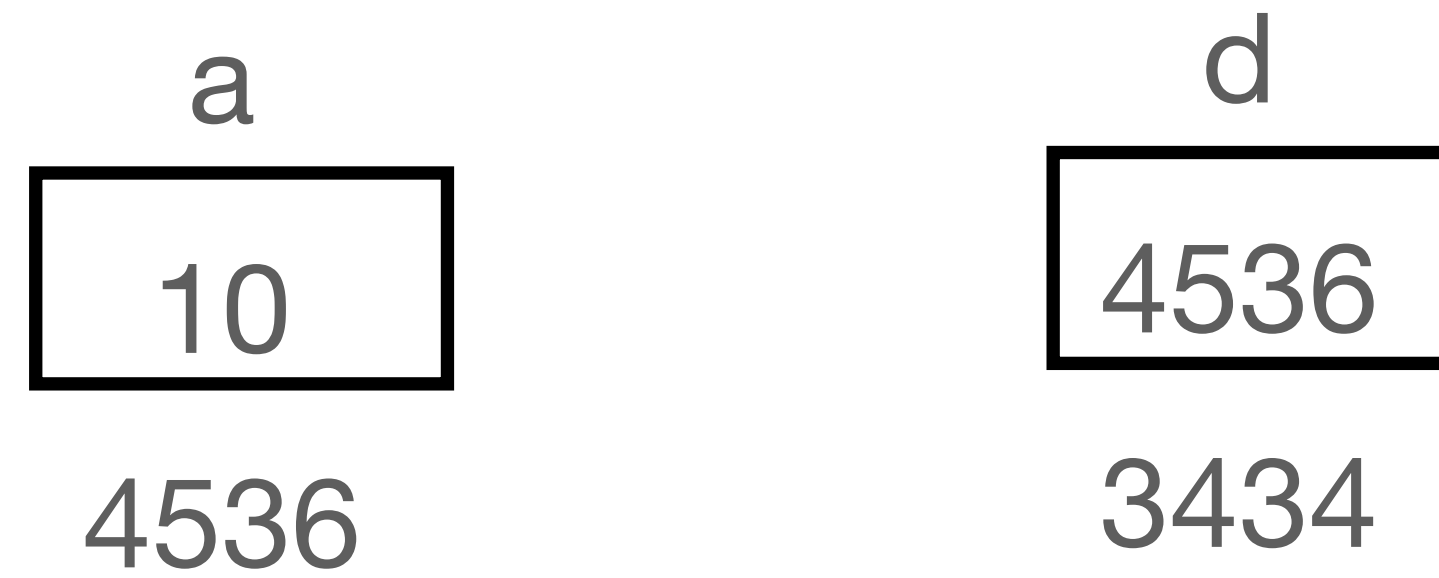
Meaning and picturization of a pointer variable

You will see stars now!

```
int a = 10;
```

```
int *d;
```

```
d = &a;
```



Meaning and picturization of a pointer variable

You will see stars now!

```
int a = 10;
```

```
int *d;
```

```
d = &a;
```

```
printf("\n", a, &a, d, &d);
```



CW: Fillin the format specifiers and the output.

How to store the address of other data types?

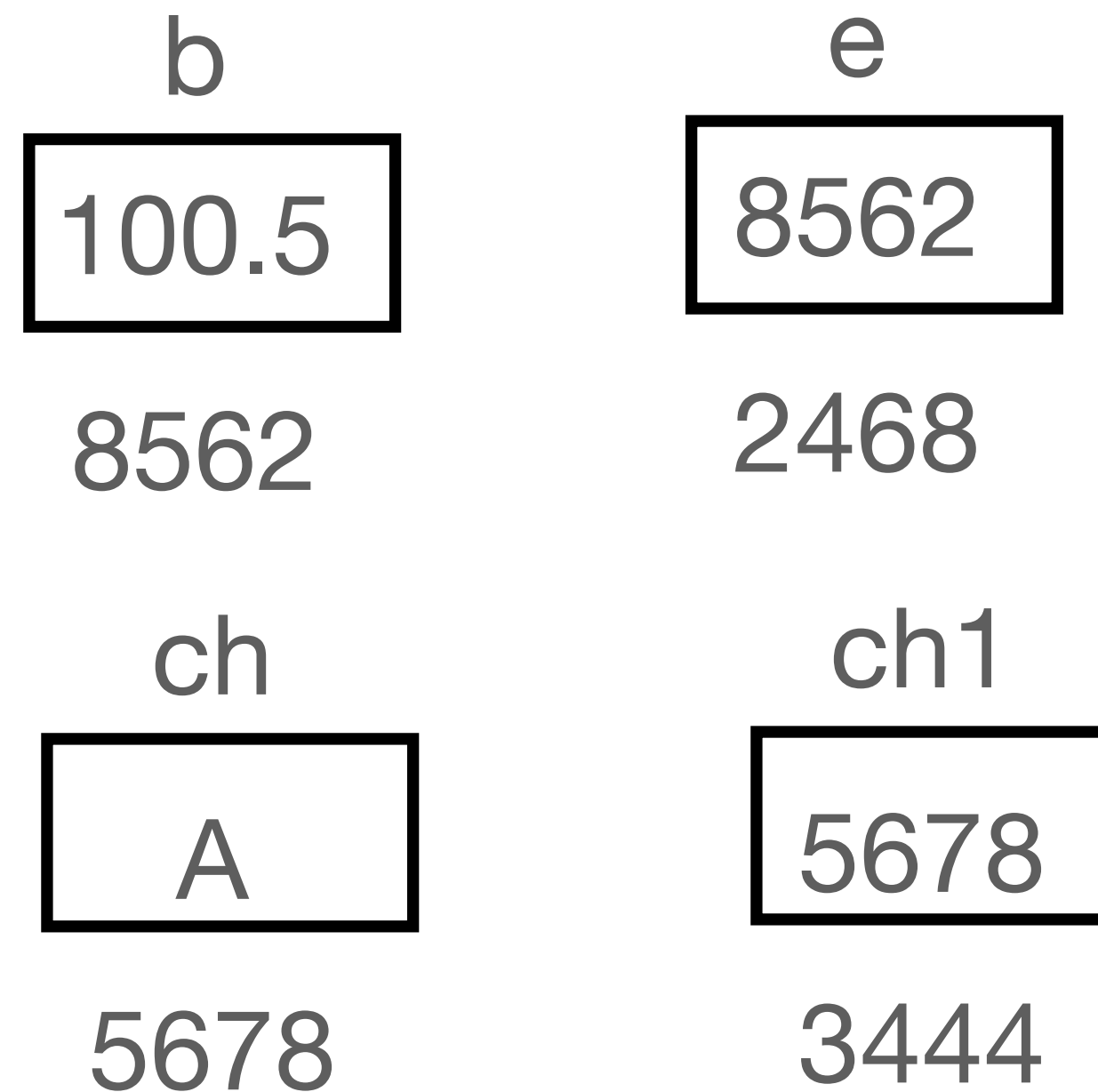
You will see more stars now!

```
float b = 100.5;
```

```
char ch = 'A'
```

```
float *e = &b;
```

```
char *ch1 = &ch;
```



CW: Fillin the format specifiers and the output.

Storing the address of different data types

You will see more stars now!

```
int a = 10;
```

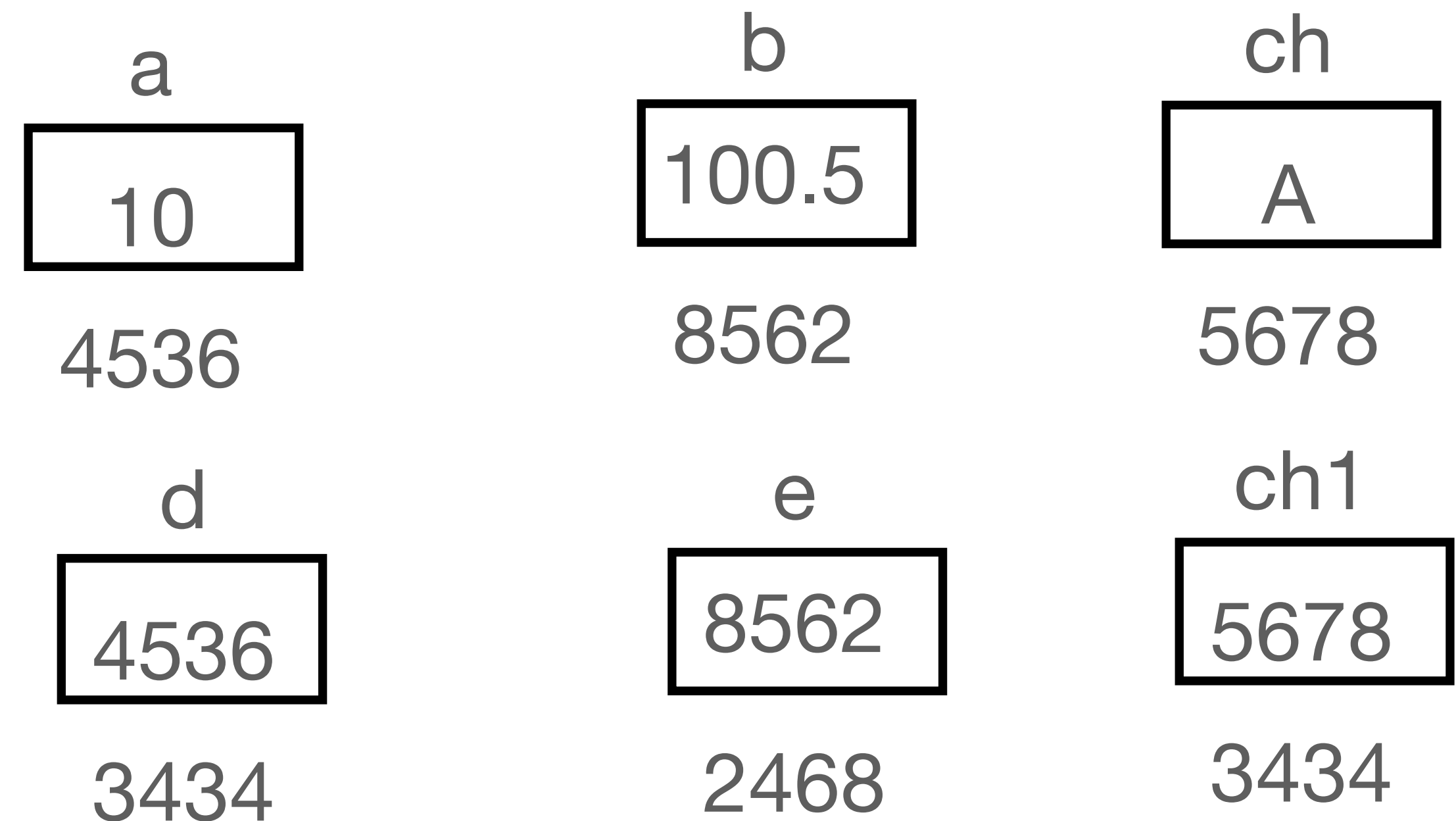
```
float b = 100.5;
```

```
char ch = 'A';
```

```
int *d = &a;
```

```
float *e = &b;
```

```
char *ch1 = &ch;
```



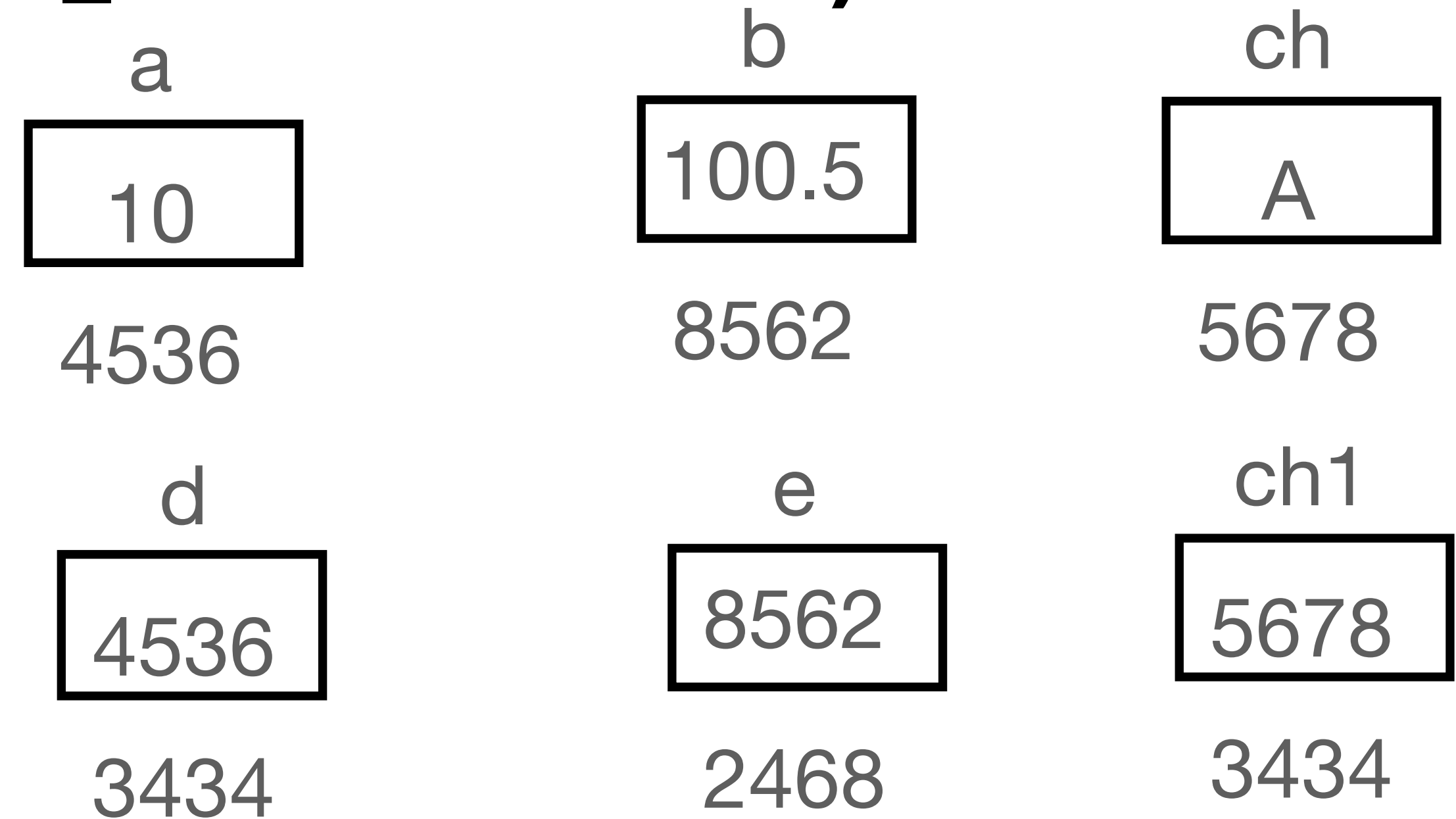
Storing the address of different data types

You will see more stars now! (L11_PointerBasics.c)

```
int a = 10, *d = &a;
```

```
float b = 100.5, *e = &b;
```

```
char ch = 'A', *ch1 = &ch;
```



```
printf("\n", a, d, b, e, ch, ch1);
```

CW: Fill in the format specifiers and the output.

Some terminologies

Jargons!

```
int *d;
```

d is a variable that store the address of an integer variable.

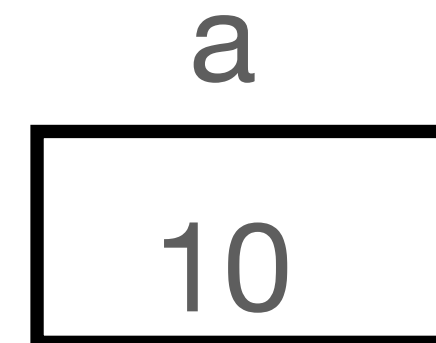
- — integer pointer
- — pointer to an integer
- — address variable
- — single pointer variable
- — stores memory location
- — stores the address

Back to integer pointer

stars but in a different form - watch out

```
int a = 10, *d;
```

```
d = &a;
```



4536



3434

```
printf("\n", a);
```

```
printf("\n", d);
```

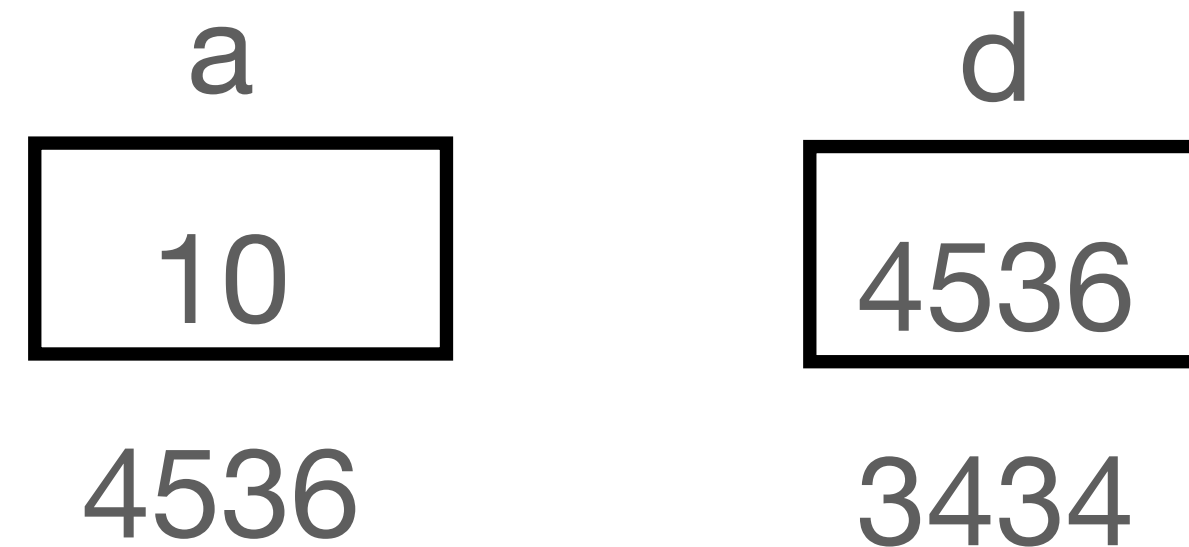
```
printf("\n", *d);
```

* operator (asterix)

Value at the address / indirection / dereferencing

int a = 10, *d;

d = &a;



Meaning of *d;

*d = *(4536) = value at the address 4536 = 10 = a

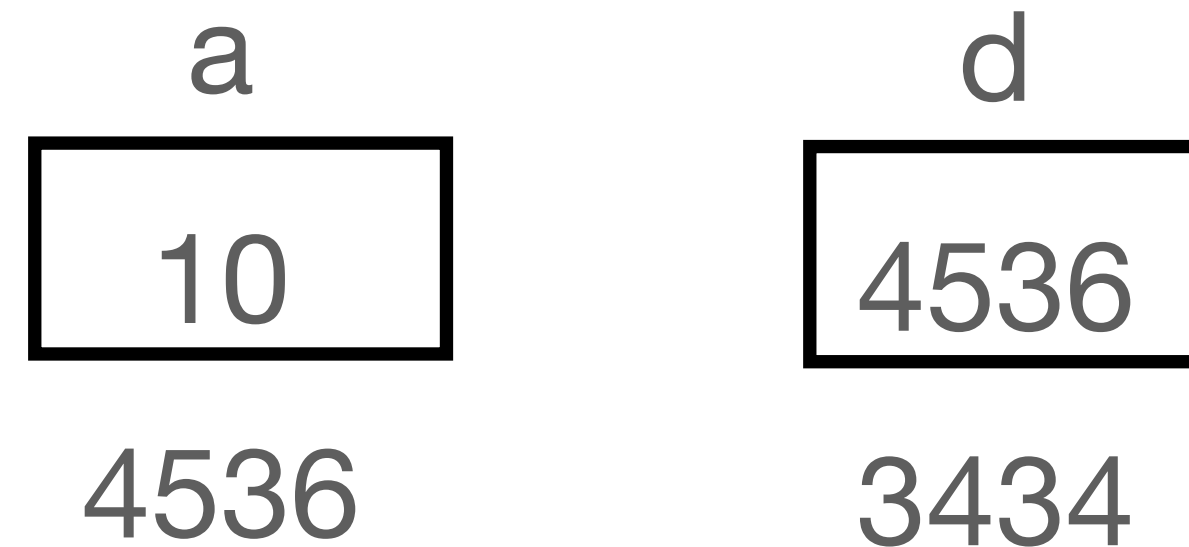
printf(" \n", a, *d);

* operator (asterix)

Value at the address / indirection / dereferencing

```
int a = 10, *d;
```

```
d = &a;
```



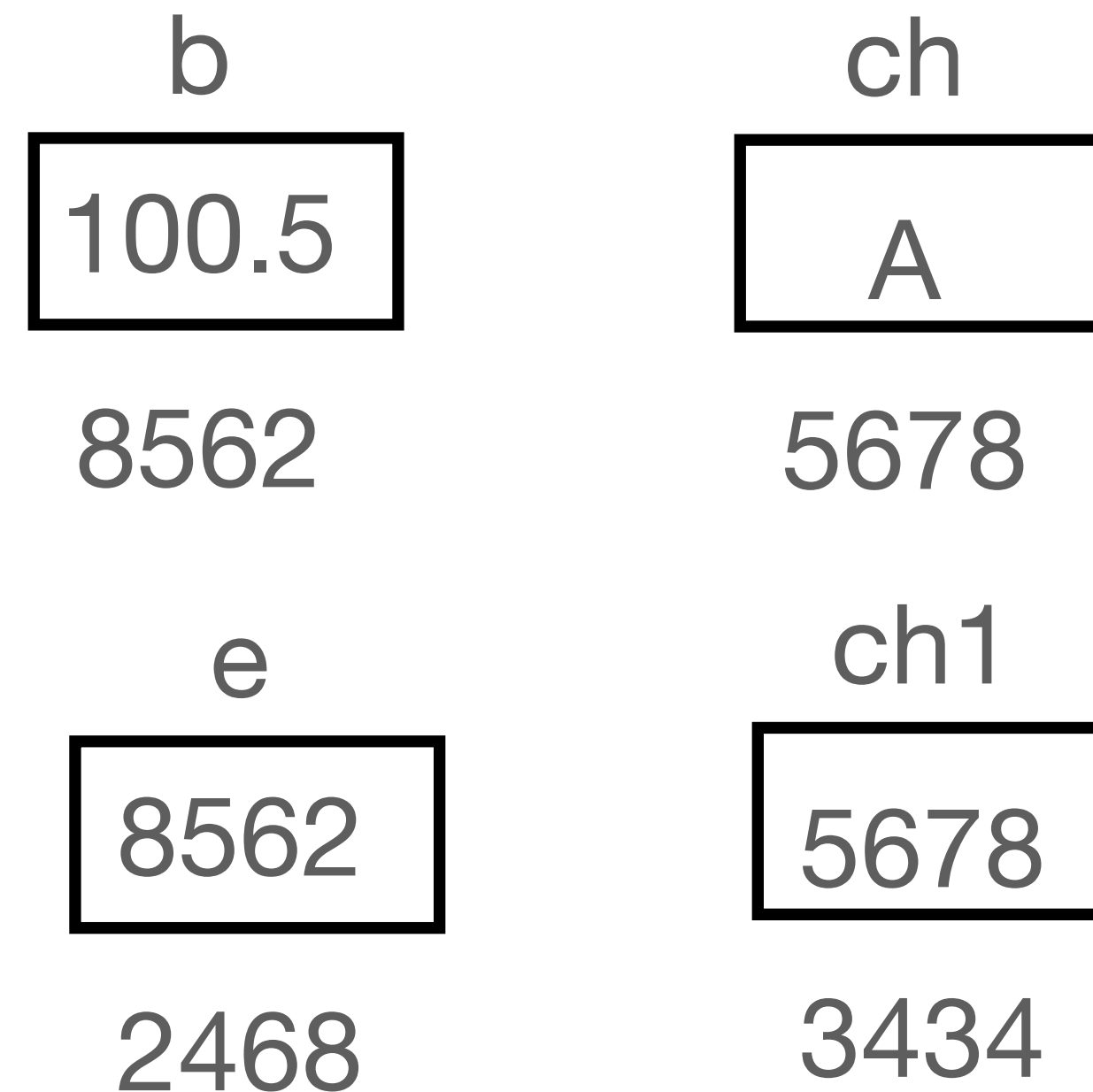
```
printf("\n", a, d, &a, &d, *d, *a, *&a, &*d, &&a);
```

Indirection for other data types

You will see more stars now!

```
float b = 100.5, *e = &b;
```

```
char ch = 'A', *ch1 = &ch;
```



```
printf("\n", e, *e, b, &b);
```

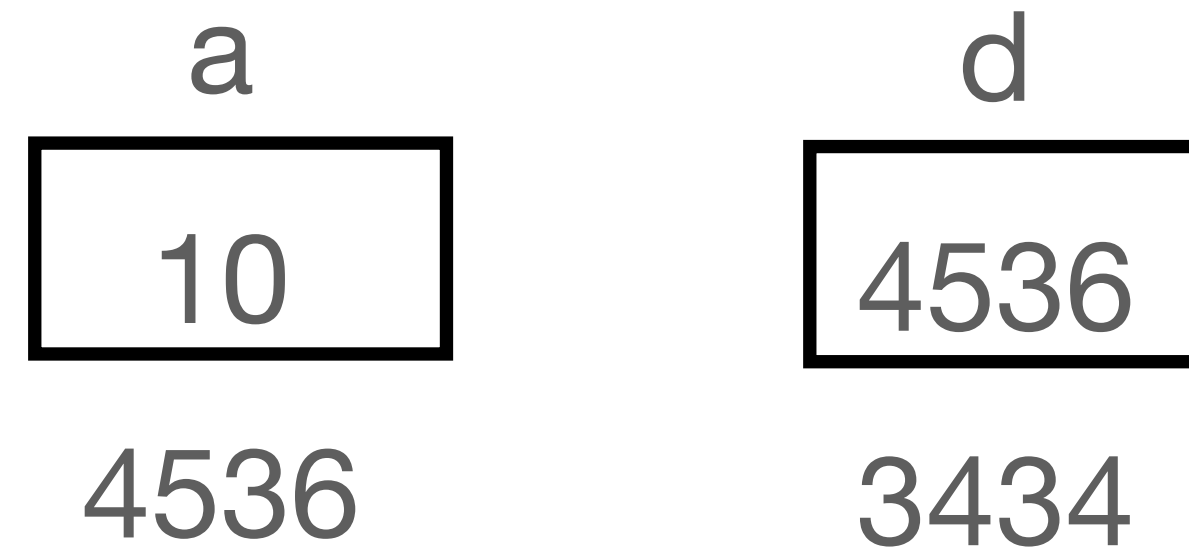
```
printf("\n", ch, *ch1, &ch, &ch1);
```

CW: Fill in the format specifiers and the output.

address of the address of more and more starts - double pointer

int a = 10, *d;

d = &a;



Suppose you want to store the address of d (i.e. &d)?

$\&d = \&(\text{address of } a) = (\text{address of})(\text{address of } a)$

address of address of more and more starts - double pointer

int a = 10, *d;
d = &a;
d = &a

a	d
10	4536
4536	3434

To store the 'address of' a, single pointer is needed.

&d = need to store the (address of)(address of a)

It needs a double pointer!!

i.e. int **p;

p = &d;

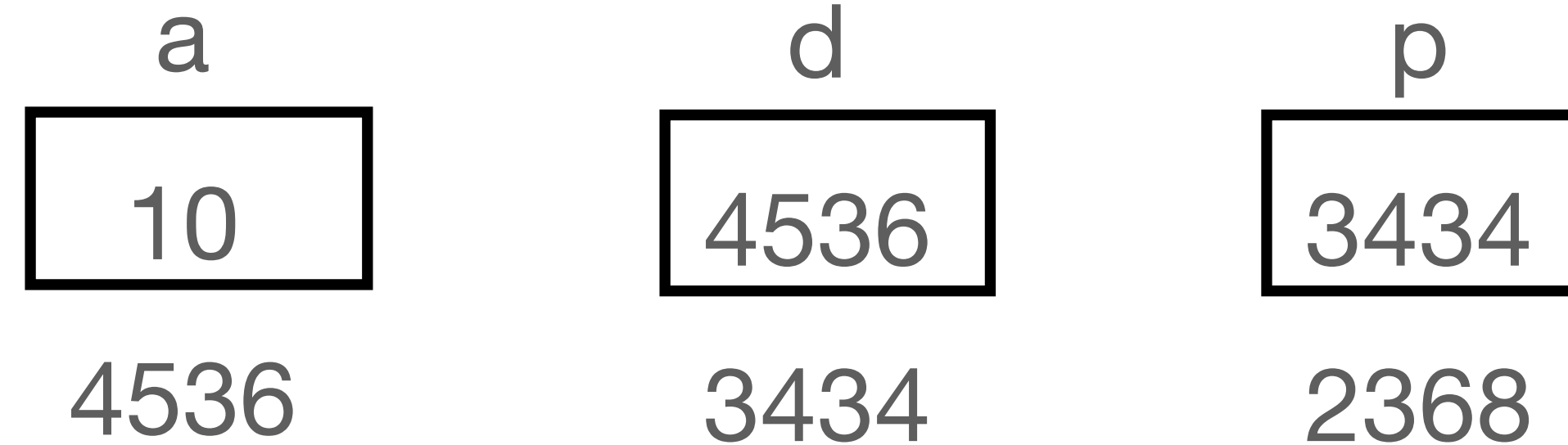
How does the memory map look?

more and more stars - double pointer

```
int a = 10, *d, **p;
```

```
d = &a;
```

```
p = &d;
```



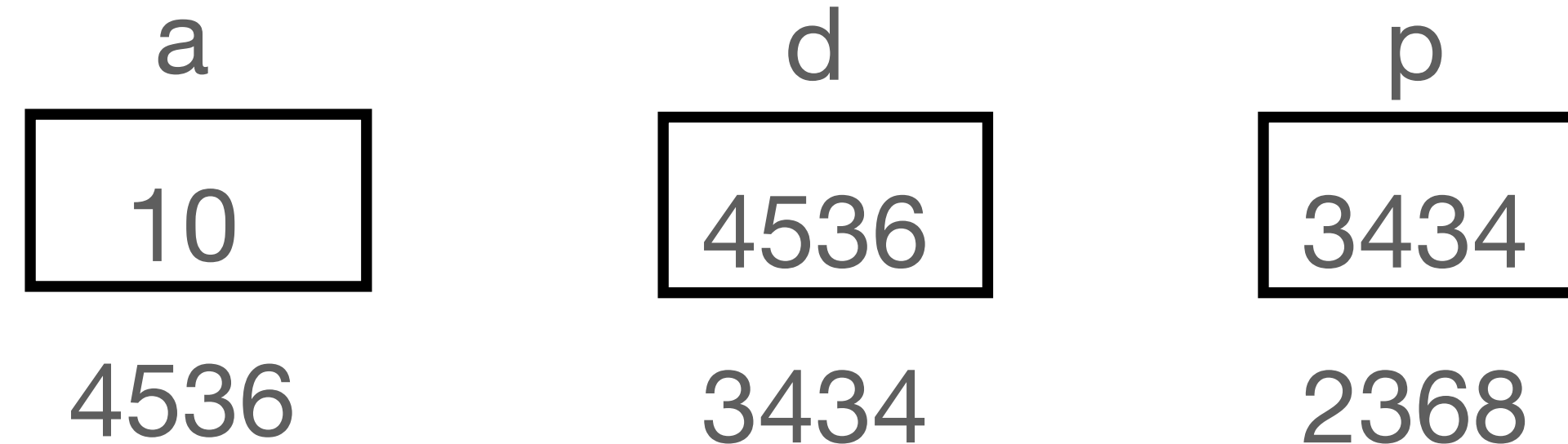
Indirection also works in a similar way

more and more stars

```
int a = 10, *d, **p;
```

```
d = &a;
```

```
p = &d;
```



Meaning of **p;

$**p = **(3434) = *(value\ at\ the\ address\ 3434) = *(4536) = value\ at\ the\ address\ 4536 = 10 = a$

```
printf("\n", a, *d, **p);
```

```
printf("\n", a, *d, **p, d, &d, p, &p, &a);
```

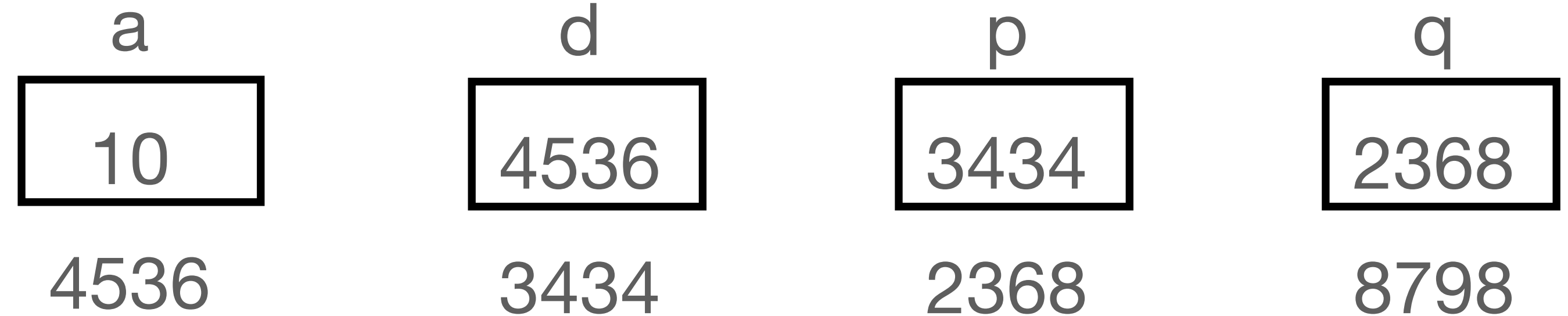
How will you store the address of p more and more stars (L11_PointerIndirection.c)

```
int a = 10, *d, **p, ***q;
```

```
d = &a;
```

```
p = &d;
```

```
q = &p;
```



HW: Do a similar exercise of single, double, triple pointer and their indirections for both floating point and character data types.

More deeper thoughts using indirection

```
int a = 10, *d, **p, ***q;
```

```
d = &a;
```

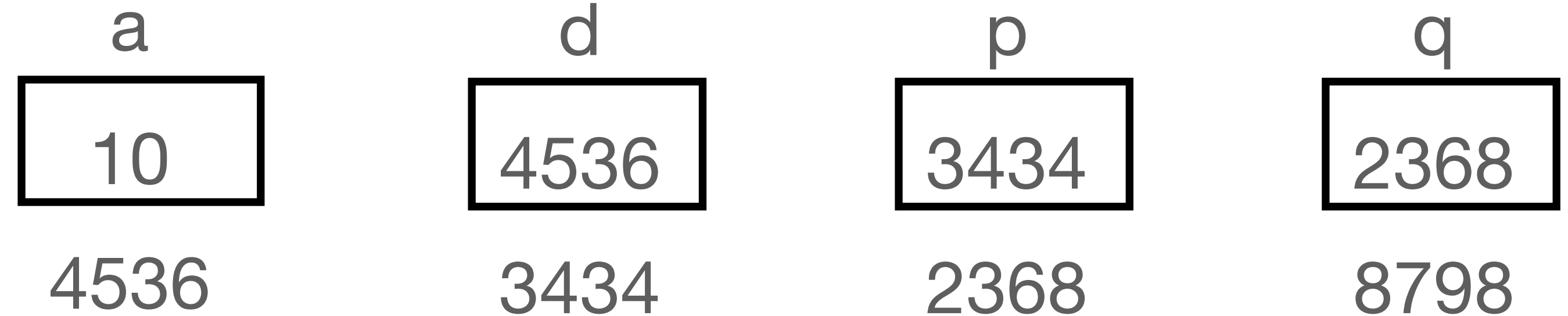
```
p = &d;
```

```
q = &p;
```

```
*d = 145;
```

```
**p = 111;
```

```
***q = 1234;
```



More deeper thoughts

using indirection (L11_PointerIndirection.c)

```
int a = 10, *d, **p, ***q;
```

```
d = &a;
```

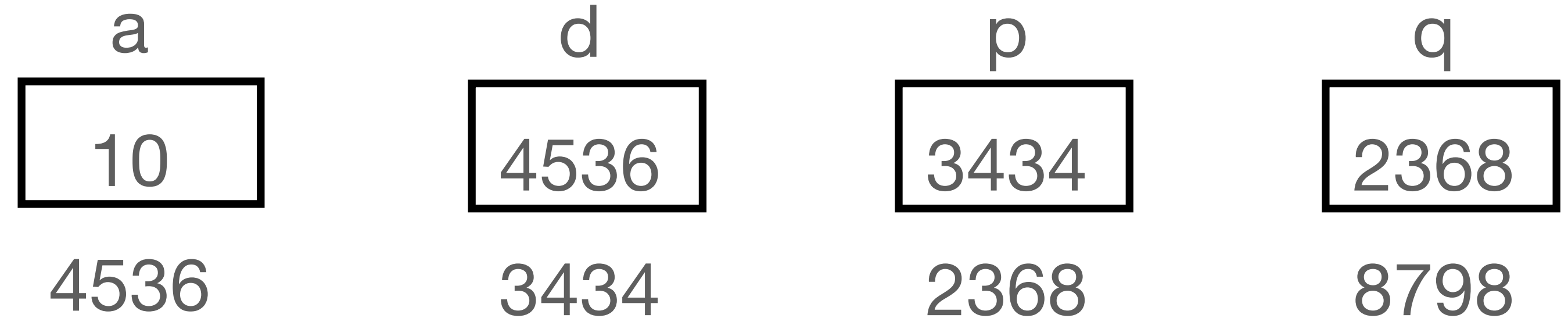
```
p = &d;
```

```
q = &p;
```

```
*d = 145;
```

```
**p = 111;
```

```
***q = 1234;
```



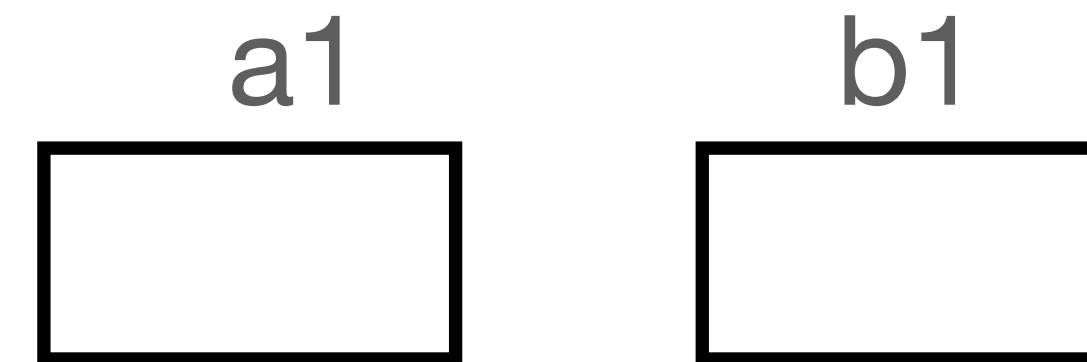
Changing the “value at the address of a variable” changes the “value of the variable”.

HW: Do a similar exercise of single, double, triple pointer and their indirections for both floating point and character data types.

Swapping using functions

swap - L10_Swapfunction.c

```
void swap(int a1, int b1);
```



```
void swap(int a1, int b1) { //a1, b1 – formal arguments
```

```
    int c;
```

```
    c = a1;
```

```
    a1 = b1;
```

```
    b1 = c;
```

```
    printf("Values of a and b after swapping a = %d, b = %d\n", a1, b1);
```

```
    return;
```

```
}
```

```
int main() {
```

```
    int a, b;
```

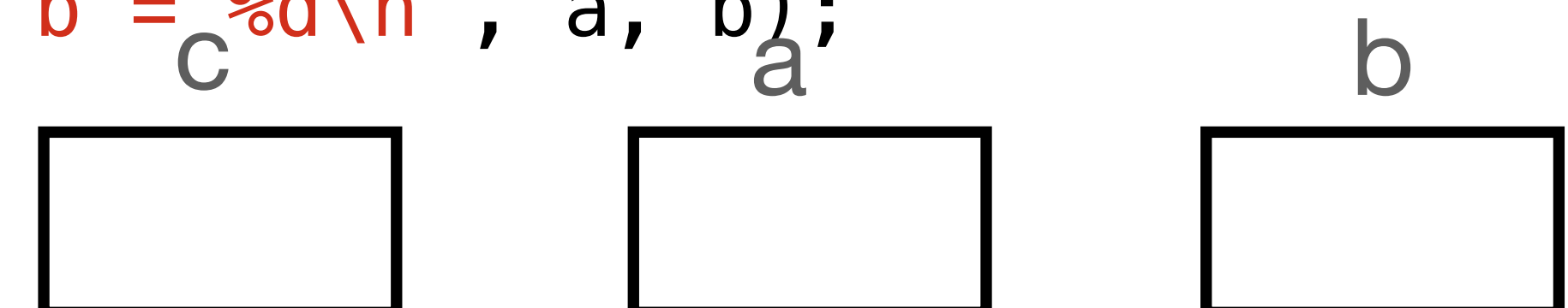
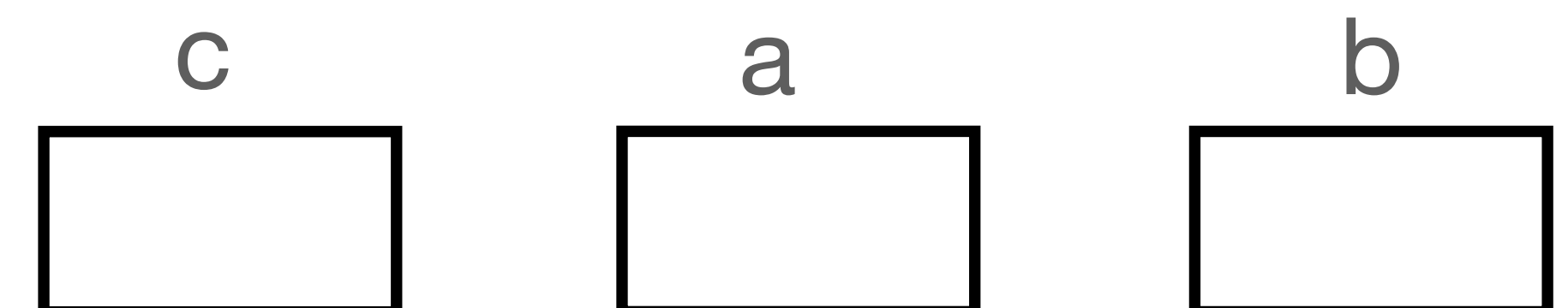
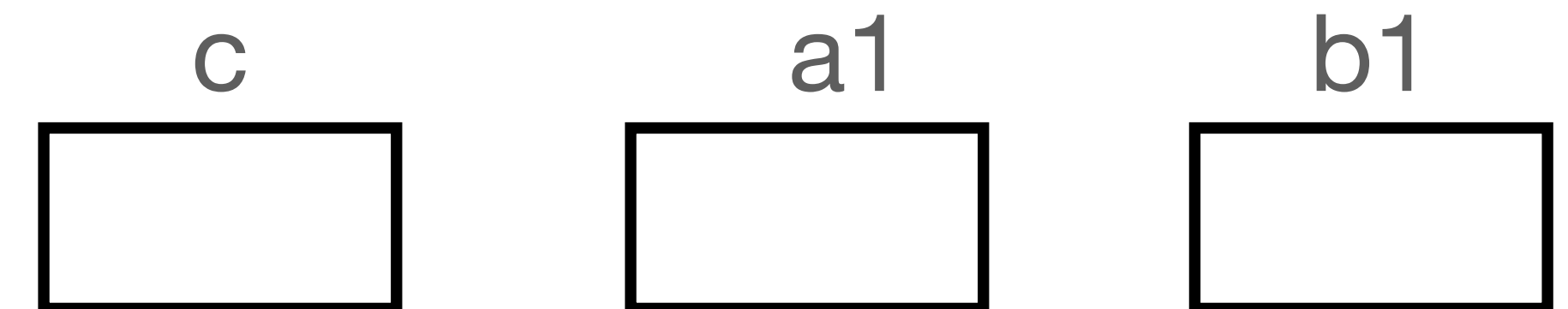
```
    a = 20; b = 45;
```

```
    printf("Values of a and b before swapping a = %d, b = %d\n", a, b);
```

```
    swap(a, b); // a, b – actual arguments
```

```
    printf("Values of a and b after calling a = %d, b = %d\n", a, b);
```

```
}
```



Swapped valued in main()

L10_GlobalSwap.c

- Global (CW)
 - variable is known to all functions.
 - Any change in inside any the functions will be reflected wherever the variable has been used.
- Combination of local / global (HW)
 - Hint: Pass one variable and use the other as global variable

More key points to be noted

- 'Call by value'
- We have been passing only scalar variables!

b
45
4536

Call by value

L11_CallV_CallR.c

```
void callbyvalue(int b)
{
    printf("value of b inside callbyvalue b = %d\n", b);
    b = 122;
    printf("after changing the value inside callbyvalue function, b = %d\n", b);
}

int main(void) {
    int b;
    b = 45;
    printf("b = %d\n", b);
    callbyvalue(b);
    printf("In main: callbyvalue function, b = %d\n", b);
    return 0;
}
```

Diagram illustrating the state of variable `b` during the execution of the `callbyvalue` function:

- Before the function call, `b` in `main` holds the value 45.
- When `callbyvalue` is called, a new local variable `b` is created, also holding the value 45.
- Inside `callbyvalue`, `b` is changed to 122. This change is local to the function and does not affect the `b` in `main`.
- After the function returns, `b` in `main` remains 45.

Call by address / reference

L11_CallV_CallR.c

```
void callbyadd(int *c)
{
    printf("value of b inside callbyvalue b = %d\n", *c);
    *c = 122;
    printf("after changing the value inside callbyvalue function, b = %d\n", *c);
}

int main(void) {
    int b;
    b = 45;
    printf("b = %d\n", b);
    callbyadd(&b);
    printf("In main: callbyvalue function, b = %d\n", b);
    return 0;
}
```

Diagram illustrating memory addresses and values:

- Variable **c** (address 7688) contains the value 1234.
- Variable **b** (address 1234) contains the value 45.

**HW: Do the swap of two variable
using both call by value and call
by reference approaches?
Explain your code with memory
maps.**