Computer Systems Principles

C Pointers

UMASSCS
SCHOOL OF COMPUTER SCIENCE

Learning Objectives

- Learn about typedef, enum, and union
- Learn and understand pointers
- Understand pointers and strings relationship
- Understand pointers and arrays relationship
- Understand stack allocation
- Learn about dynamic/heap allocation
- Learn about dynamic arrays
- Learn about header files and how to create them
- Understand two implementations for a stack

Activity!

- **strlen(char s[])**
  strncpy(char dest[], char src[], int n)
  - Take a moment to implement these functions!
  - Work with the people around you!
  - Write it down on a piece of paper!

What is in a name?

- **Names are useful**
  - Descriptive variable names are nice!
  - Descriptive function names are brilliant!
  - It is also great to name types!
- **C allows you to give a type an alias**
  - `typedef` is a keyword in C
  - Give a meaningful name to an existing type

Typedef Example

- **Syntax**

```
typedef existing-type new-name

typedef int color;
typedef char gender;
```

typedef.c example

- **Let us compile this example**
  - Convenient to give names to types.
  - C does not complain if you use the original type in place of the typedef!
  - Very convenient to remove the *struct* from the definition of a *structure*!

Enumerations in C

- **What are enumerations?**
  - A convenient construct for associating names with constant values that have a type.

- **Syntax:**

```
enum Color { RED, GREEN, BLUE };
enum Color color = RED;
```

enum.c example

- **Let us compile this example**
  - You can also use typedef to simplify the naming of enum types!
  - Note, that the C compiler will not check the type of an enum!
  - You need to wrap the enum in a structure if you want to have type checking!

C Unions

- **What is a union?**
  - Like structures, but every field occupies the same region in memory!
  - The largest type in the union defines the total size of that union.
- **Example:**

```
union value {
  float f;
  int i;
  char s;
};
```

```
union value v;
v.f = 45.7;
v.i = 12;
v.s = ‘X’;
```

iClicker question

```
union value { float f; int i; char s; };
struct value { float f; int I; char s; };
```

The sizes of the union and the struct are (on x86 with gcc):

A. union: 12 bytes, struct 12 bytes

B. union: 9 bytes, struct 12 bytes

C. union: 4 bytes, struct 9 bytes

D. union: 5 bytes, struct 12 bytes

E. union 4 bytes, struct 12 bytes

union.c example

- **Let us compile this example**
  - Compilers usually maintain information about variables, this example is the start of a data structure for doing this…
  - Note how the different types interpret the bits differently!
  - This example shows how character arrays and integers are interpreted differently!

animals.c

- **One last example!**
  - – Combines lots of the topics from today

Fun Exercise!

- **/etc/passwd**
  - A special file on Unix systems that define information about users.
- **Problem**
  - Write a program that will read in the characters in the /etc/passwd file and create an array of structs representing the information in the file.
  - You should define a struct that represents this file
  - Create an array of these structs (you can give your array a large enough size to hold them all)
  - Read in the file from standard input (hint: use a Unix command and pipe to help with this!)
  - Print out the information (next slide)

print-passwd output

```
username : <name>
password : <passwd>
userid   : <userid>
groupid  : <groupid>
userinfo : <userinfo>
home     : <home directory>
shell    : <shell>
…
```

Something to think about…

Binary Tree?

- What if we wanted to create a binary tree data structure in C?
- How would we do this using C structures?
- Is it even possible?
- Spend some time before next class thinking about how you might go about this.
- *Can you see why you can't?*

C Pointers

What is a pointer?

C Pointers

What is a pointer?

↓

A pointer is like a mailing address,
it tells you where something is **located**.

GEORGIA O'KEEFFE

I received your [illegible] card + dollar [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] to find [illegible] [illegible] [illegible] grandparents live in Pittsburgh. Thank you for making a little boy smile ☺ The [illegible]

The Placer
265 Four Seasons Rd
Boswell, PA 15531

C Pointers

What is a pointer?

A pointer is like a mailing address,
it tells you where something is **located**.

Every object (including simple data types)
in Java and C reside in the **memory**
of the machine.

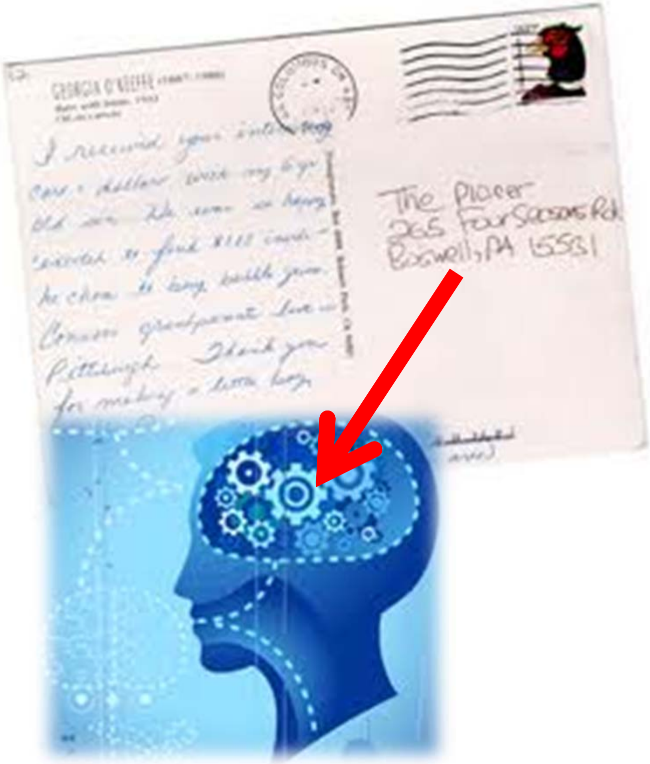
C Pointers

What is a pointer?

A pointer is like a mailing address,
it tells you where something is **located**.

Every object (including simple data types)
in Java and C reside in the **memory**
of the machine.

A **pointer** to an object is an “address” telling
You where the object is **located** in **memory**.



C Pointers

So why do I care about pointers?

C Pointers

So why do I care about pointers?

In Java, you do not have access to these pointers (or addresses).

C Pointers

So why do I care about pointers?

In Java, you do not have access to these pointers (or addresses).

In Java, you do **not** have **access to the address** of an object.

This provides **safety**!

C Pointers

So why do I care about pointers?

In Java, you do not have access to these pointers (or addresses).

In C, you do have **access to the address** of an object, which allows you to **manipulate that address** is a variety of ways.

C Pointers

```
#include <stdio.h>

int main() {
  int *ptr;
  int *ptr2;
}
```

A pointer is denoted by ‘*’ and has a type.

C Pointers

```
#include <stdio.h>

int main() {
  int *ptr;
  int *ptr2;
  int x = 2;
  int y = 5;
}
```

Here are a couple of regular integer declarations.

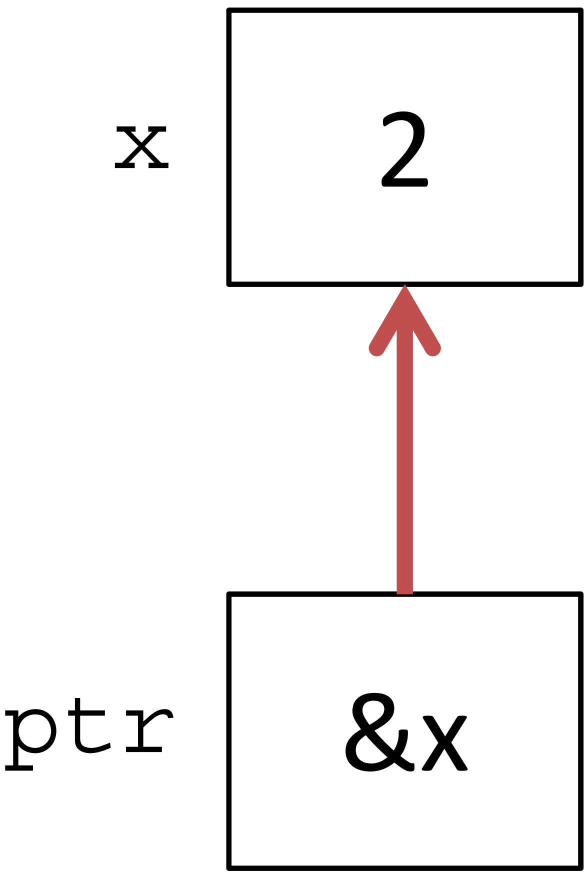
C Pointers

```
#include <stdio.h>

int main() {
  int *ptr;
  int *ptr2;
  int x = 2;
  int y = 5;
  ptr  = &x;
  ptr2 = &y;
}
```

You can assign an “address” to a pointer using the “address of” (&) operator.

A Visual…

```
int x = 2;


int ptr = &x;
```



C Pointers

```
#include <stdio.h>

int main() {
  int *ptr;
  int *ptr2;
  int x = 2;
  int y = 5;
  ptr  = &x;
  ptr2 = &y;
}
```

So, if `ptr` is a pointer that refers to a value in memory… How do we get the value?

C Pointers

```
#include <stdio.h>

int main() {
  int *ptr;
  int *ptr2;
  int x = 2;
  int y = 5;
  ptr  = &x;
  ptr2 = &y;
  printf(“Value  : *ptr = %d\n”, *ptr);
  printf(“Address:  ptr = %d\n”,  ptr);
}
```

You *dereference* (follow) the pointer!

pointers.c

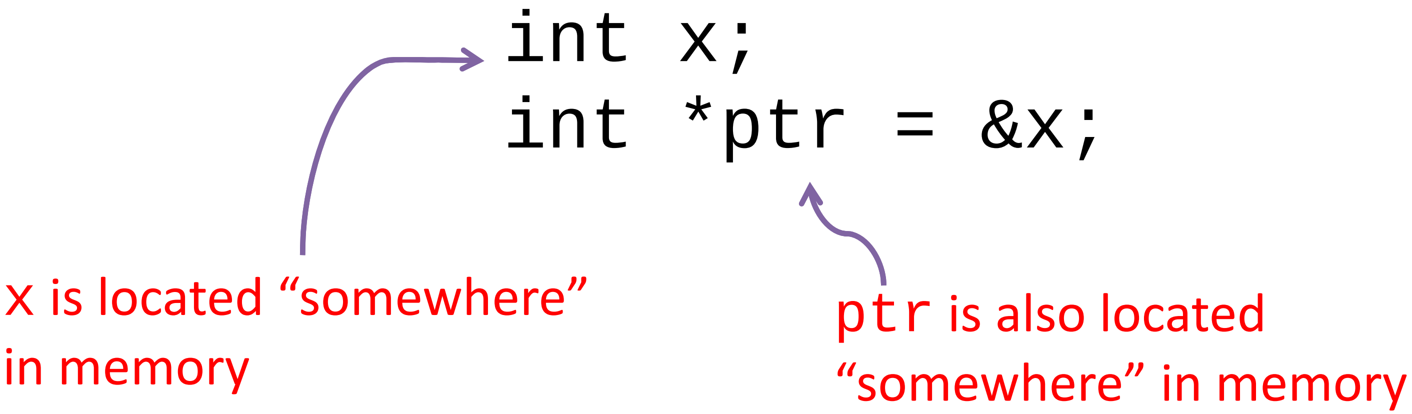
C Pointers

Imagine we have the following declarations…

```
int x;
int *ptr = &x;
```

C Pointers

Imagine we have the following declarations…



```
int x;
int *ptr = &x;
```

x is located “somewhere” in memory

ptr is also located “somewhere” in memory

C Pointers

Imagine we have the following declarations…

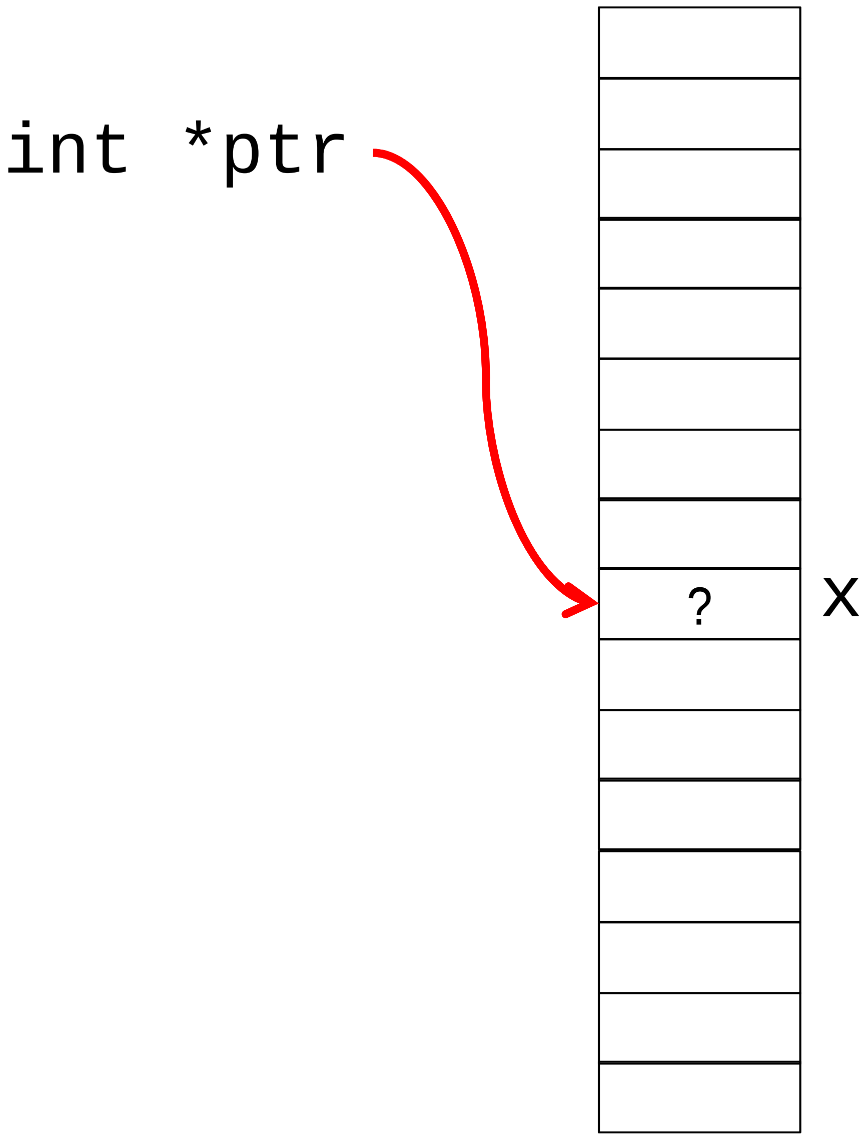
```
int x;
int *ptr = &x;
```

`x` is located “somewhere” in memory

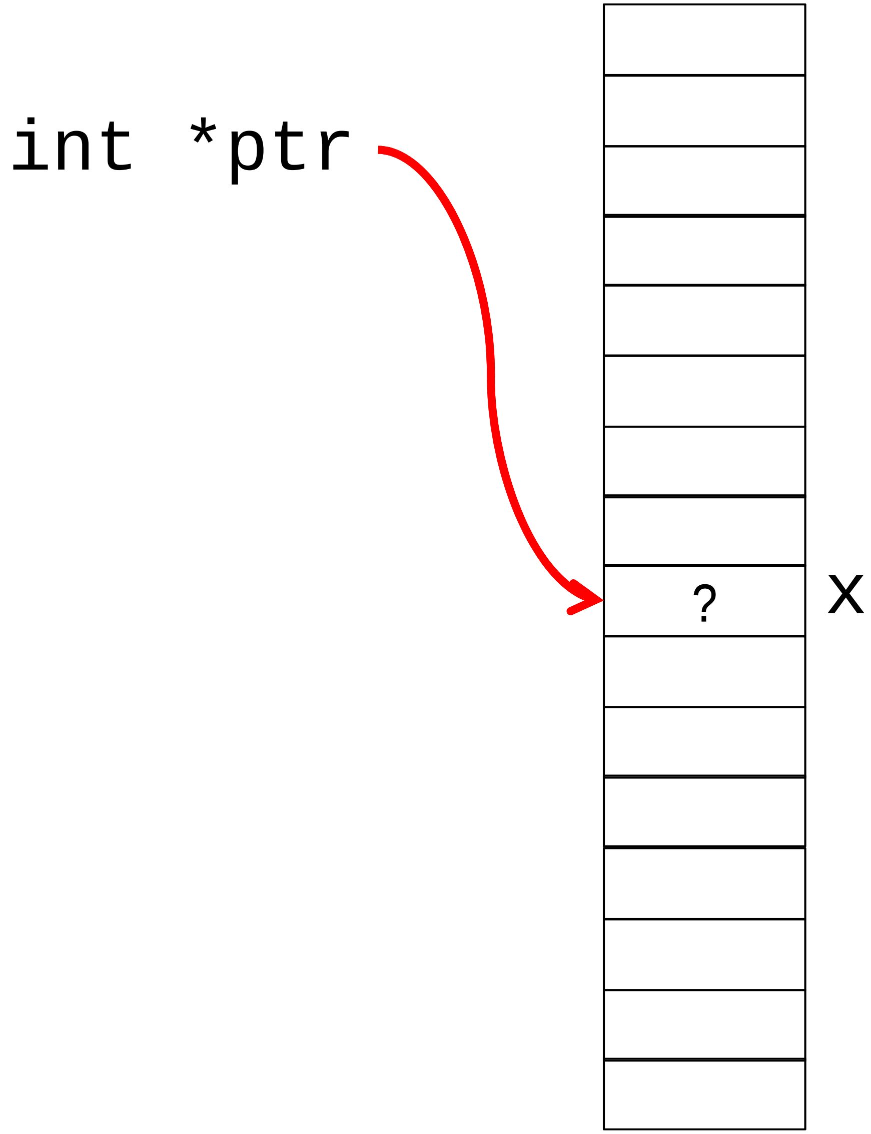
`ptr` is also located “somewhere” in memory

`ptr` “points” to the location representing `x`.

C Pointers



C Pointers



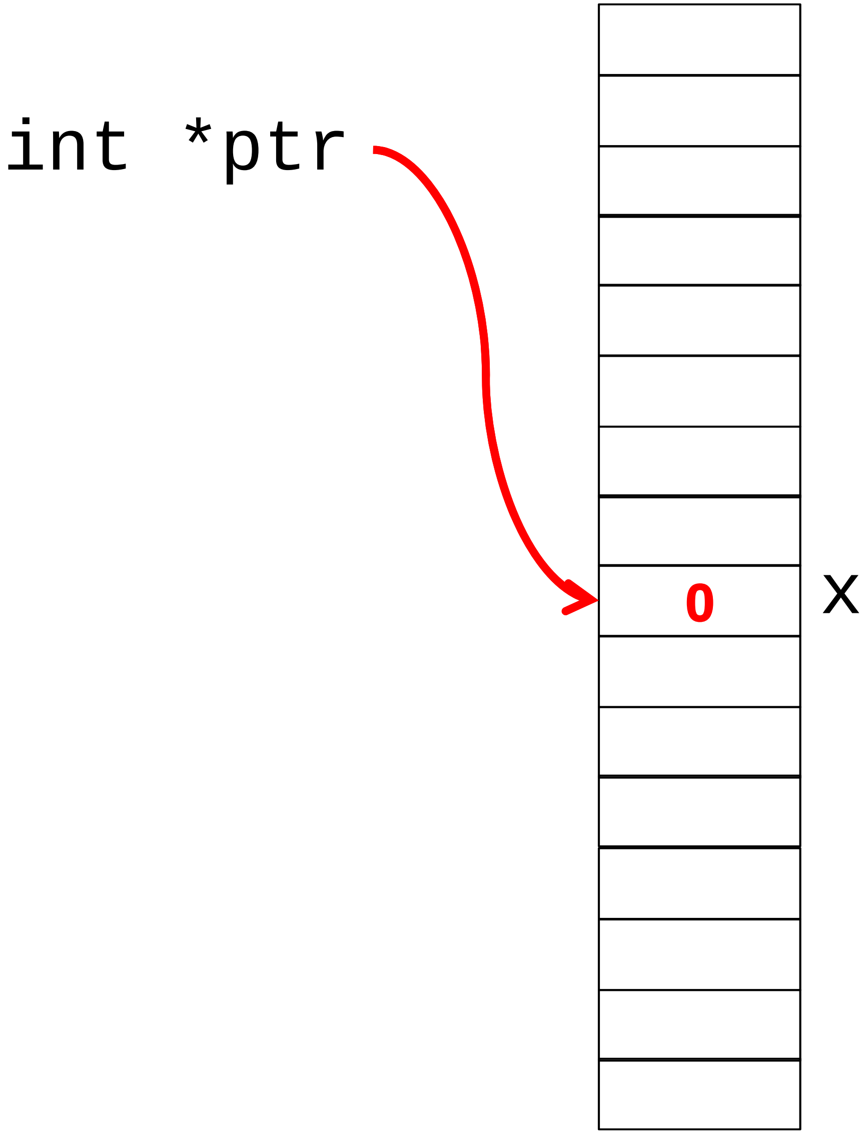
int *ptr

?

X

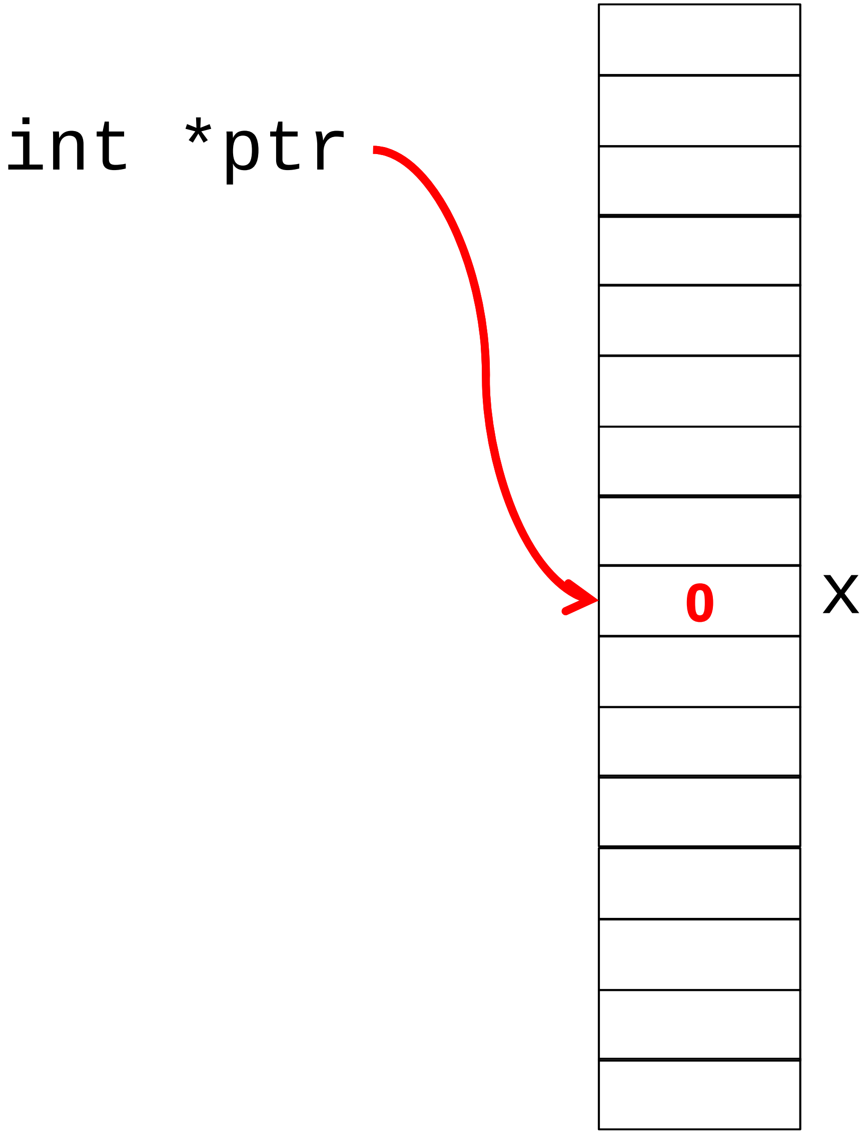
```
*ptr = 0;
```

C Pointers



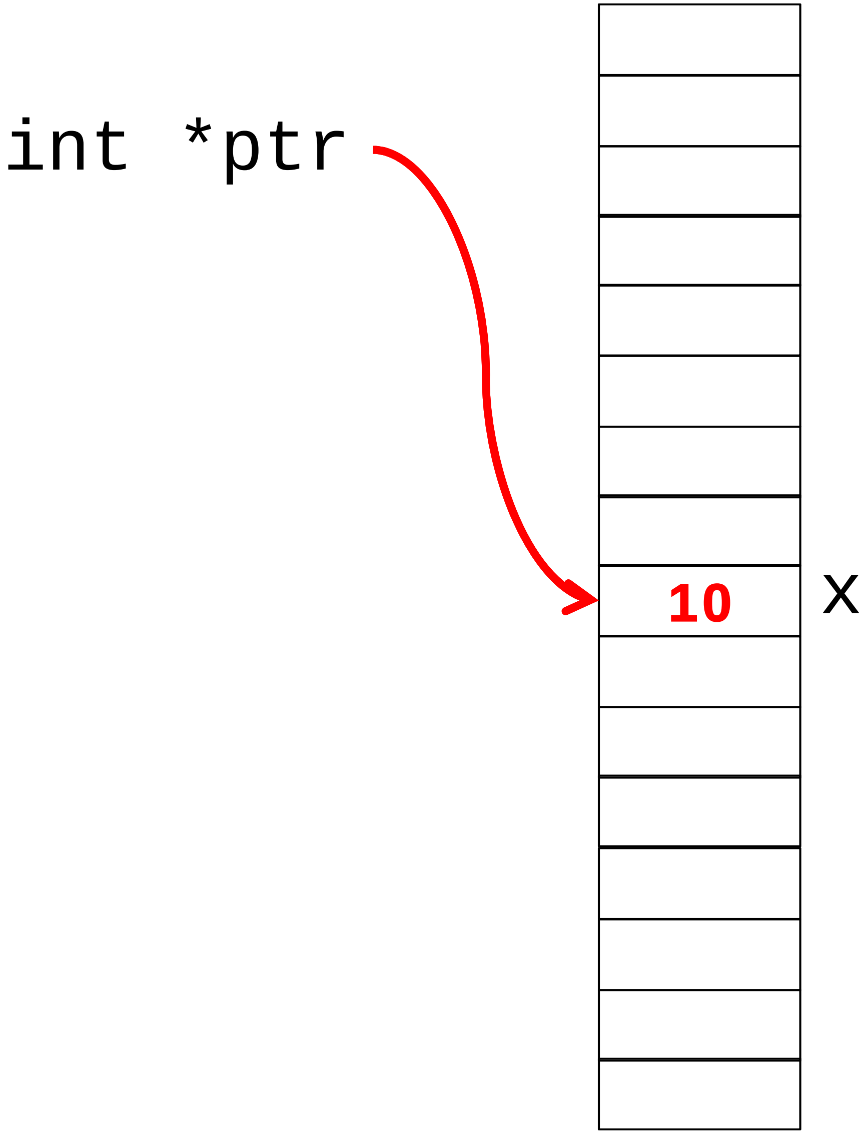
```
*ptr = 0;
```

C Pointers



```
*ptr = 0;
x = 10;
```

C Pointers



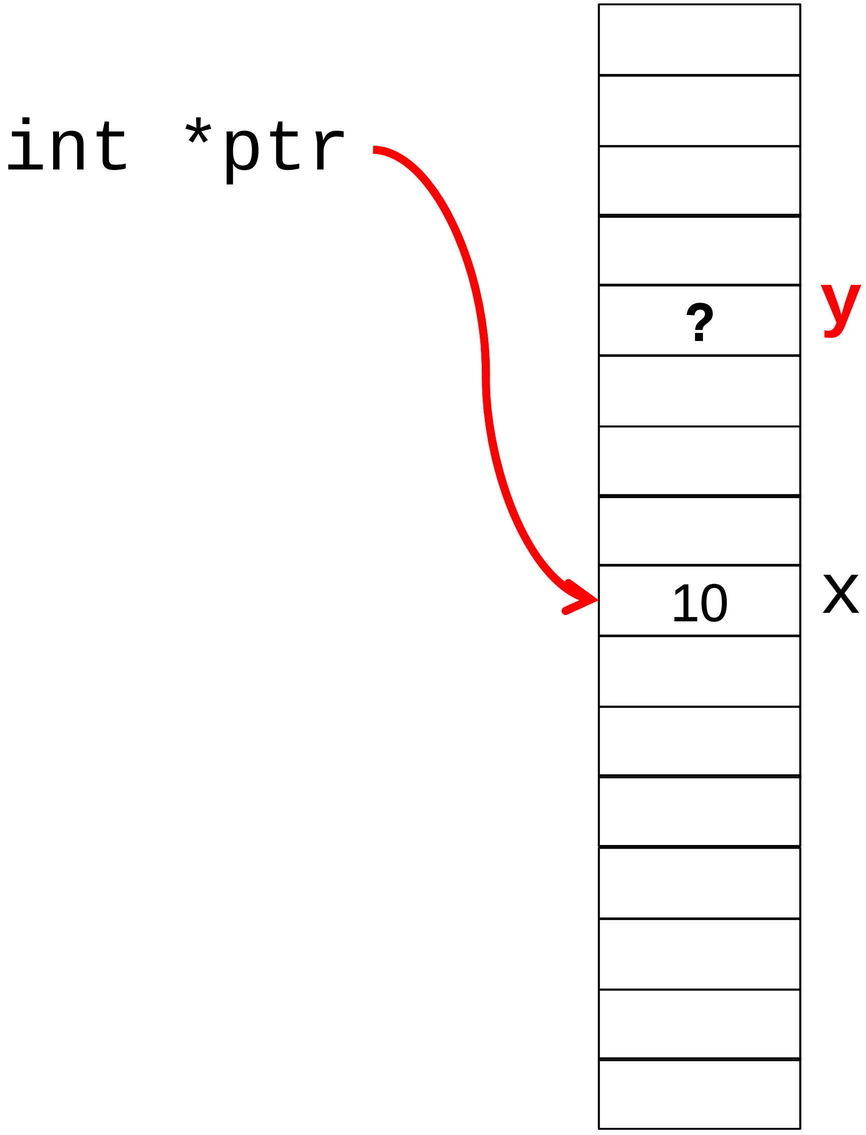
int *ptr

10

x

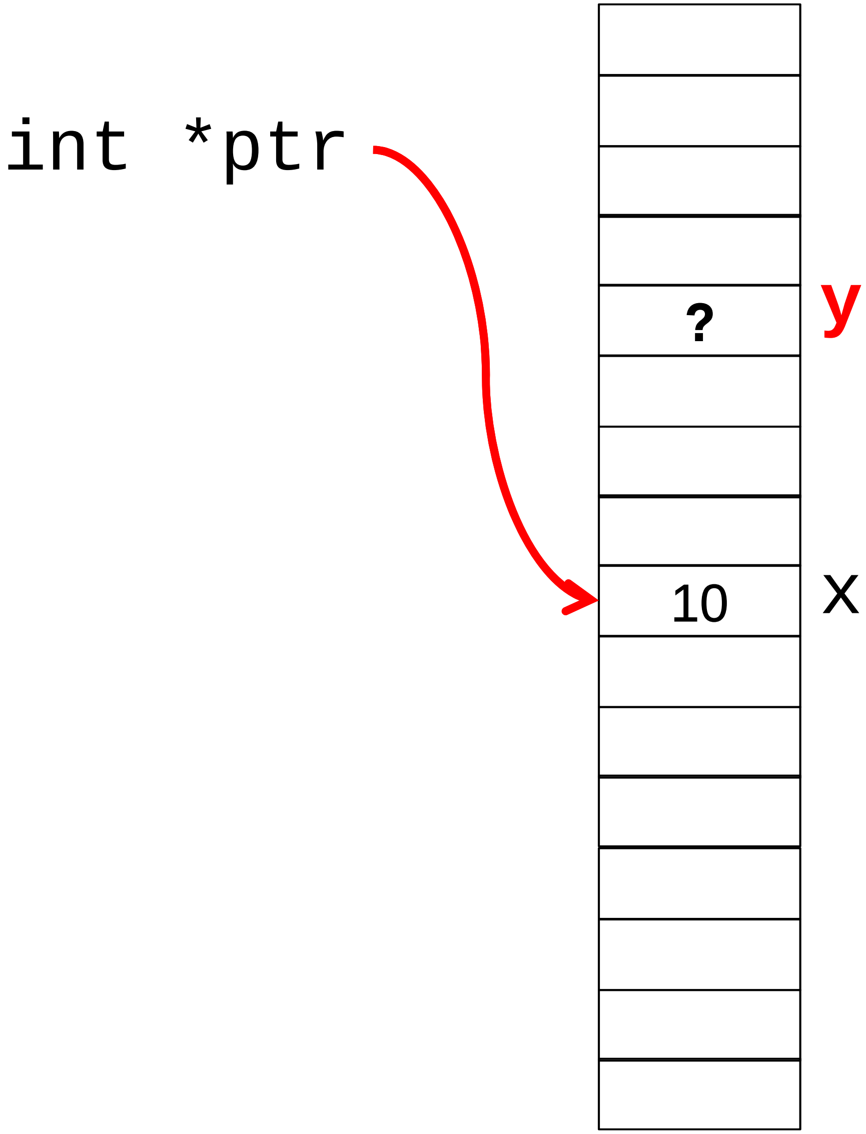
```
*ptr = 0;
x = 10;
```

C Pointers



```
*ptr = 0;
x = 10;
```

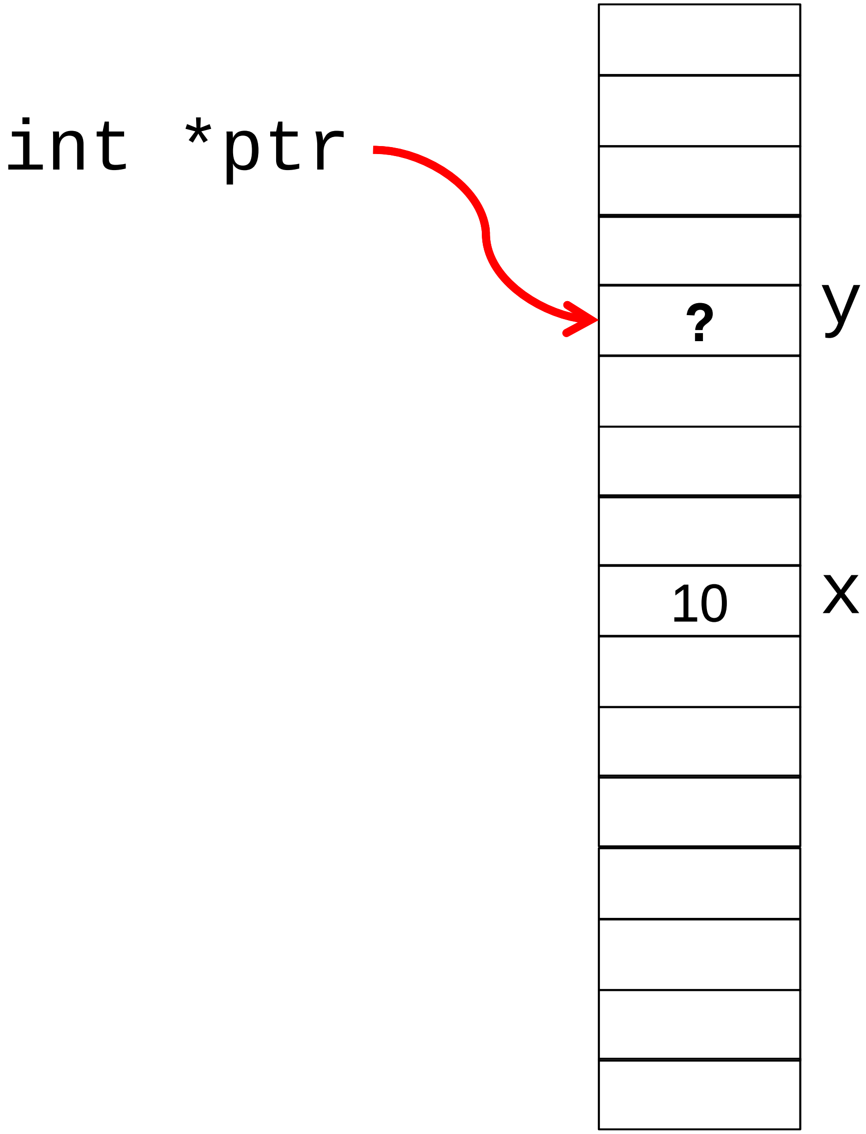
C Pointers



```
*ptr = 0;
x = 10;
ptr = &y;
```

What does this do?

C Pointers



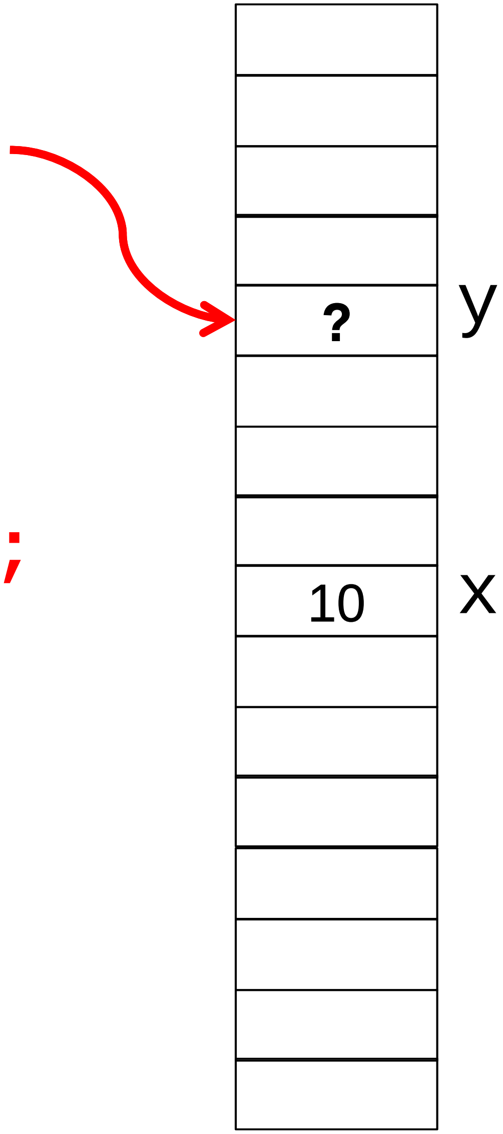
```
*ptr = 0;
x = 10;
ptr = &y;
```

What does this do?

The pointer (ptr) is assigned to a different address.

C Pointers

```
int *ptr
```

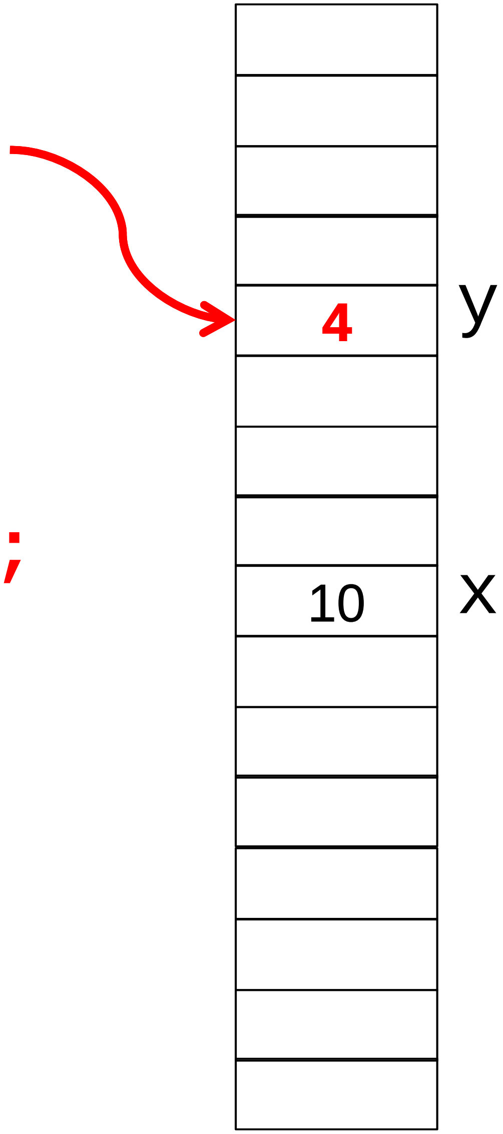


```
*ptr = 0;
x = 10;
ptr = &y;
```

```
*ptr = 4;
```

C Pointers

```
int *ptr
```

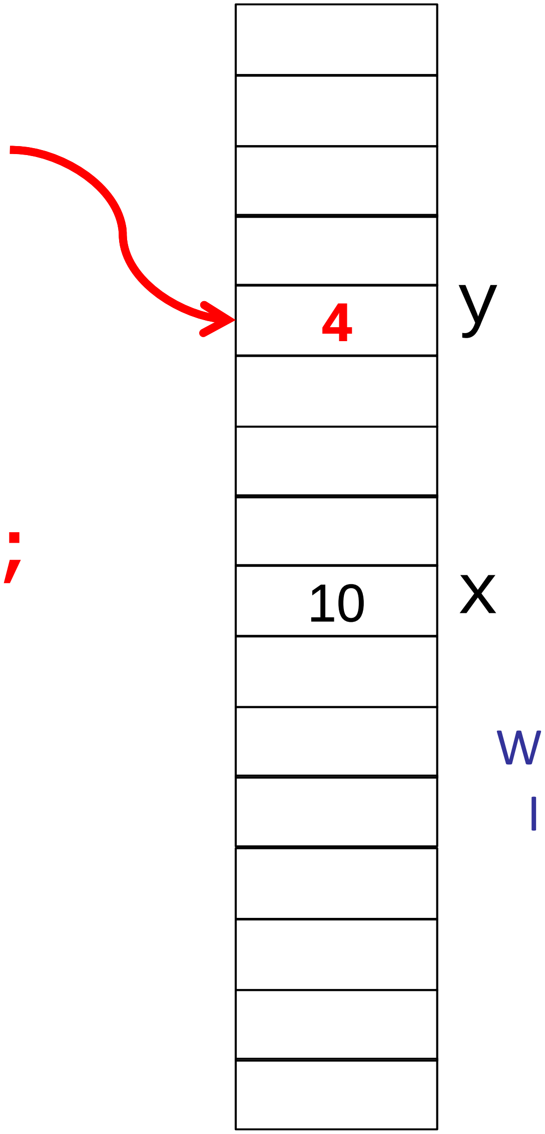


```
*ptr = 0;
x = 10;
ptr = &y;
```

```
*ptr = 4;
```

C Pointers

```
int *ptr
```



```
*ptr = 0;
x = 10;
ptr = &y;
```

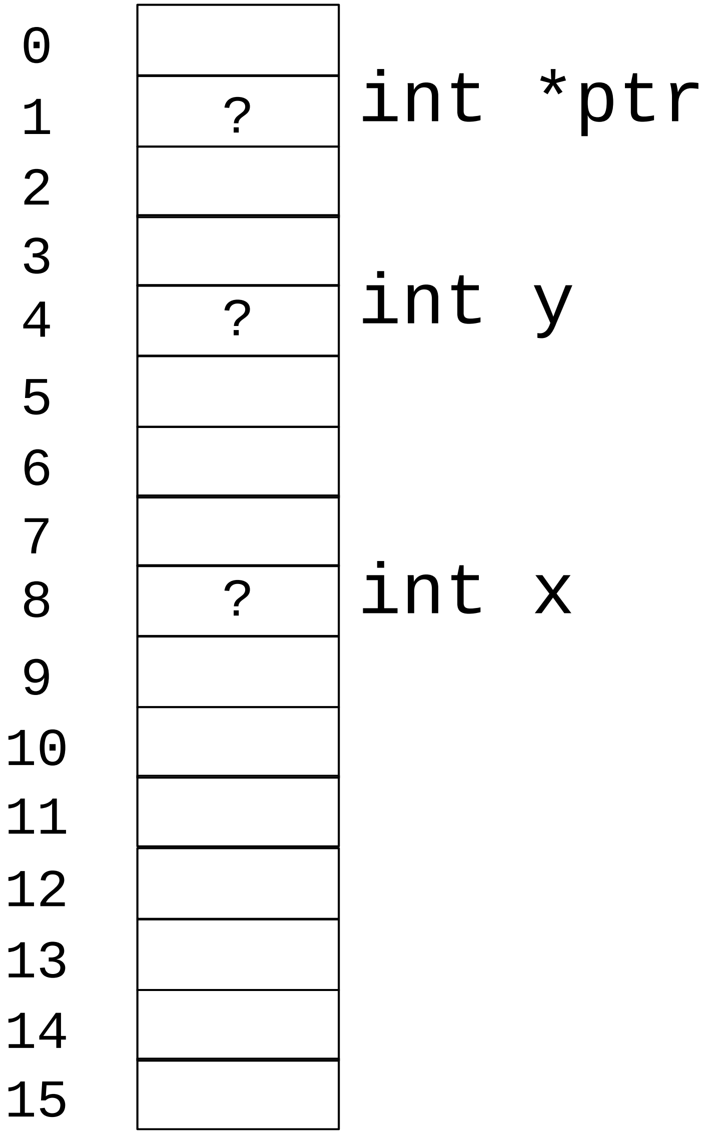
```
*ptr = 4;
```

Why is `ptr` floating off to the side?
I thought it was *also in memory*?

C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

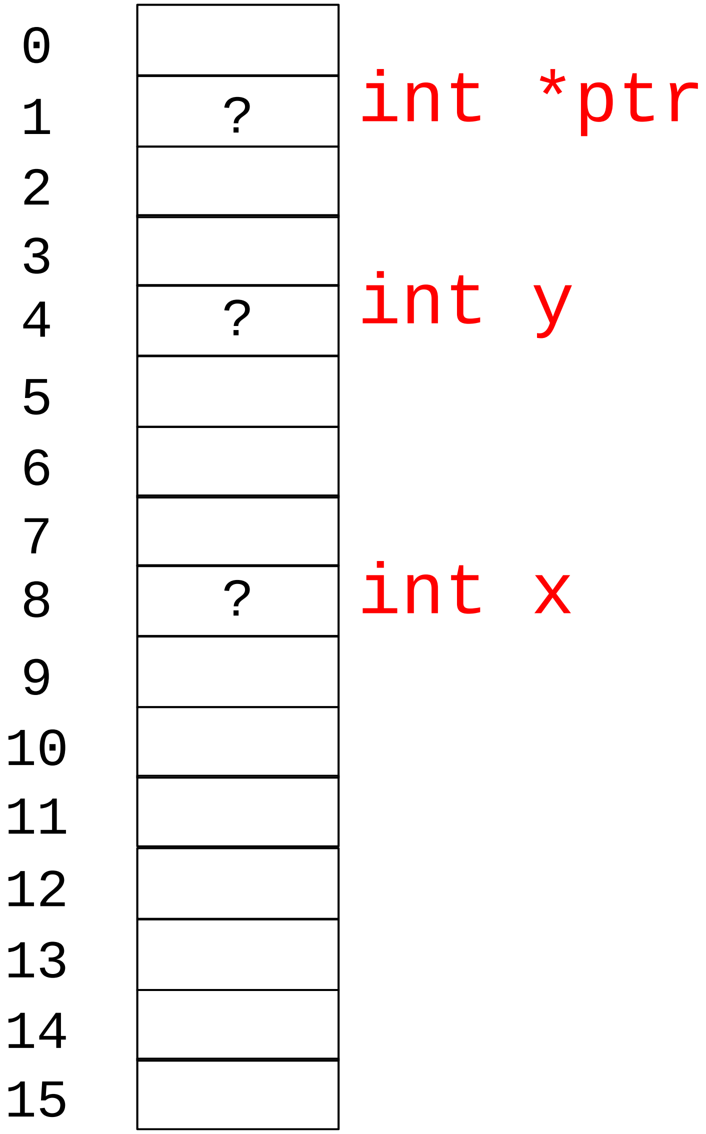


0		
1	?	int *ptr
2		
3		
4	?	int y
5		
6		
7		
8	?	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little
more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

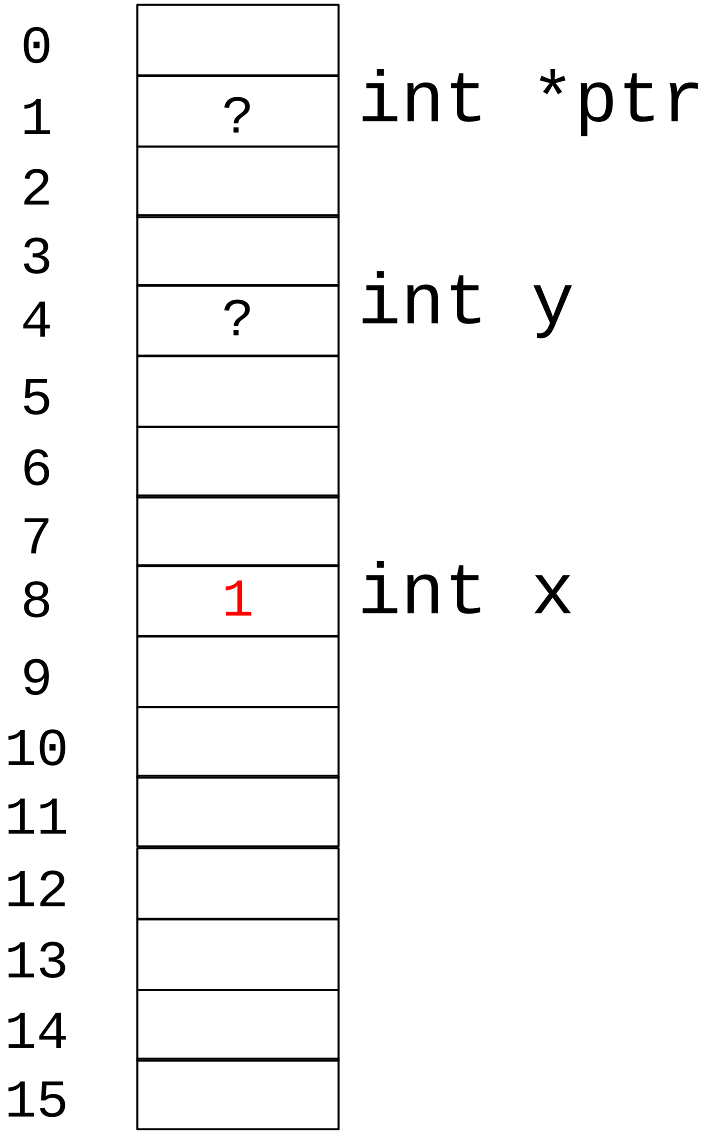


0		
1	?	int *ptr
2		
3		
4	?	int y
5		
6		
7		
8	?	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

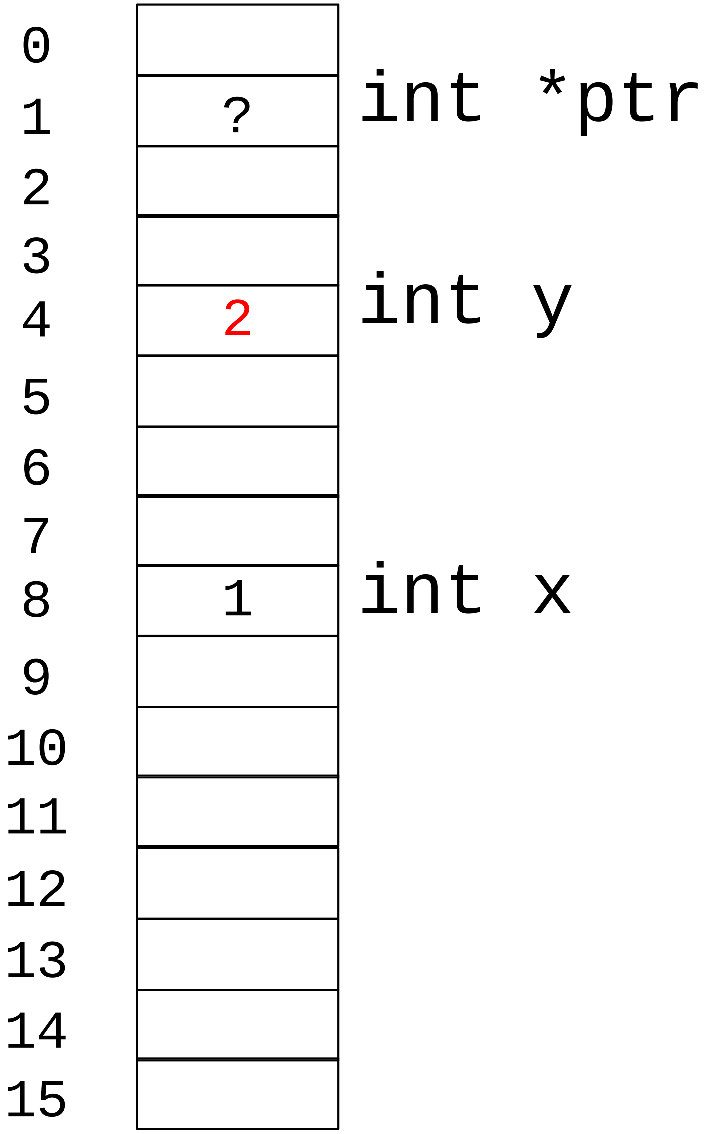


0		
1	?	int *ptr
2		
3		
4	?	int y
5		
6		
7		
8	1	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

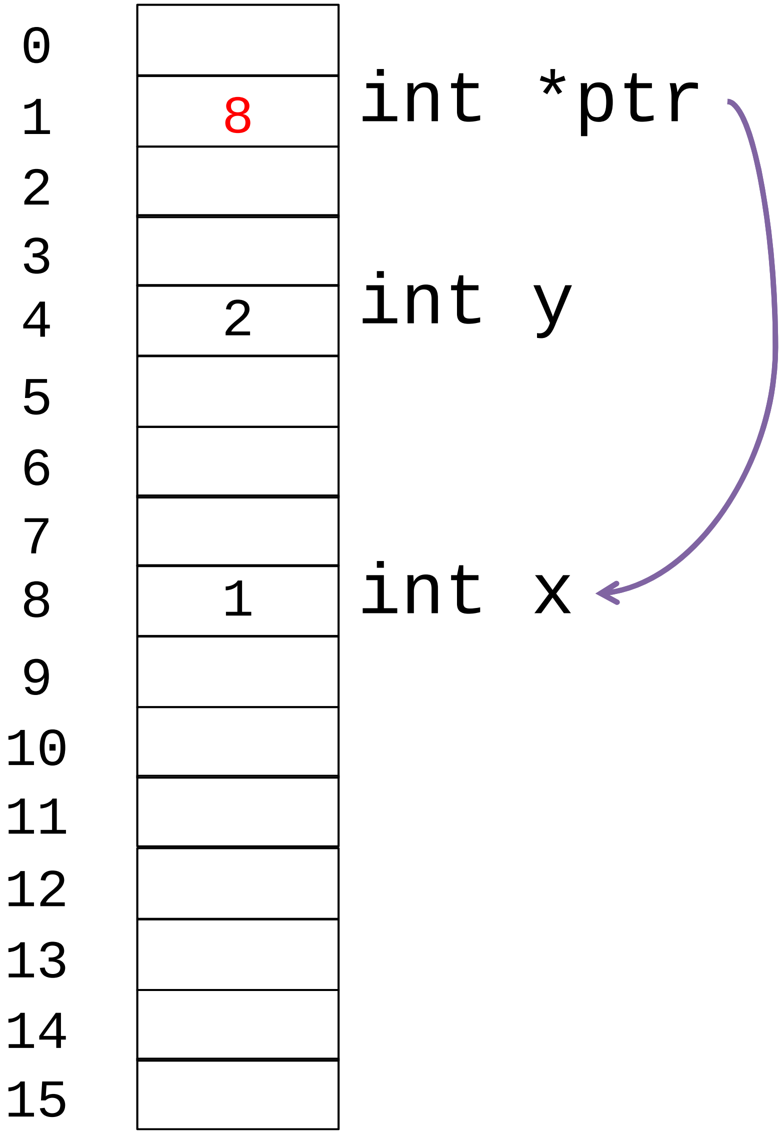


0		
1	?	int *ptr
2		
3		
4	2	int y
5		
6		
7		
8	1	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

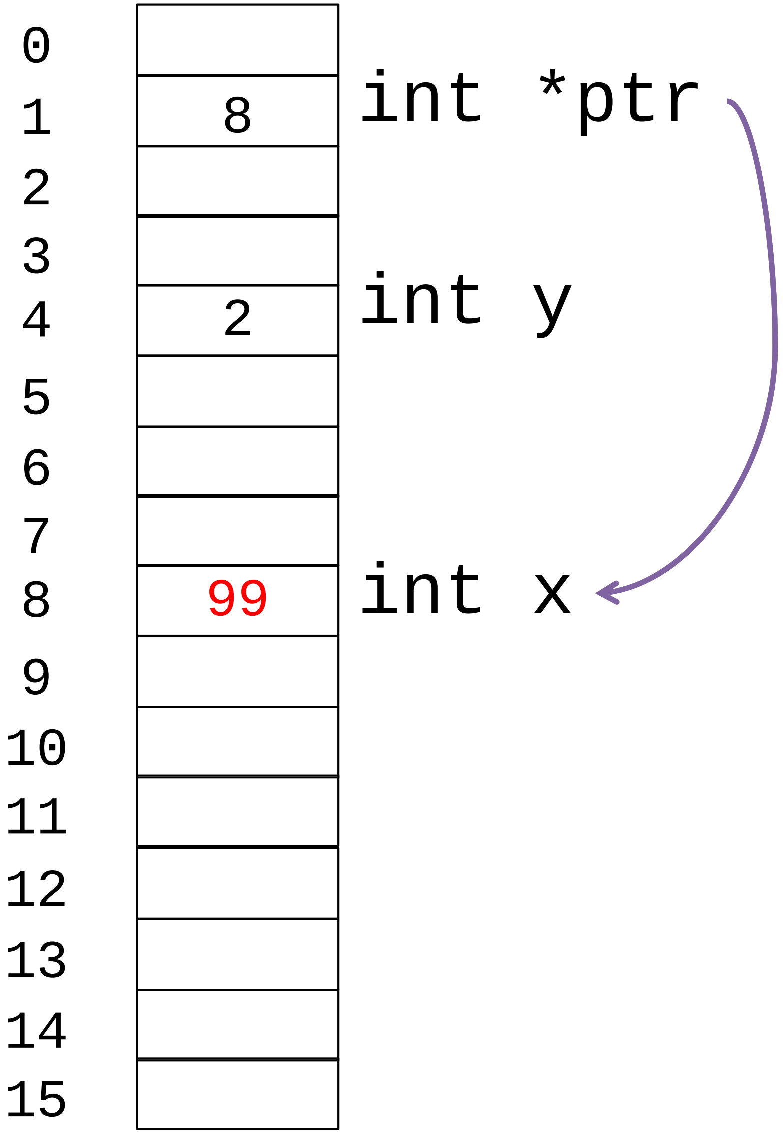
```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```



C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

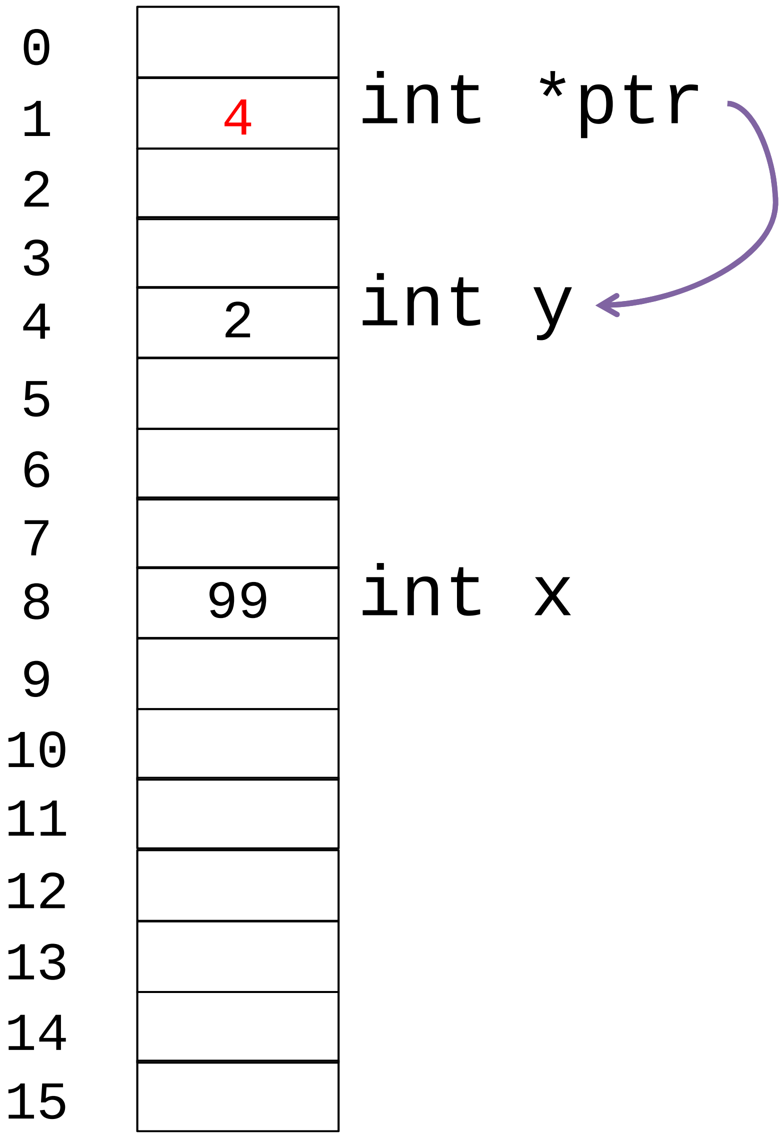


0		
1	8	int *ptr
2		
3		
4	2	int y
5		
6		
7		
8	99	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```

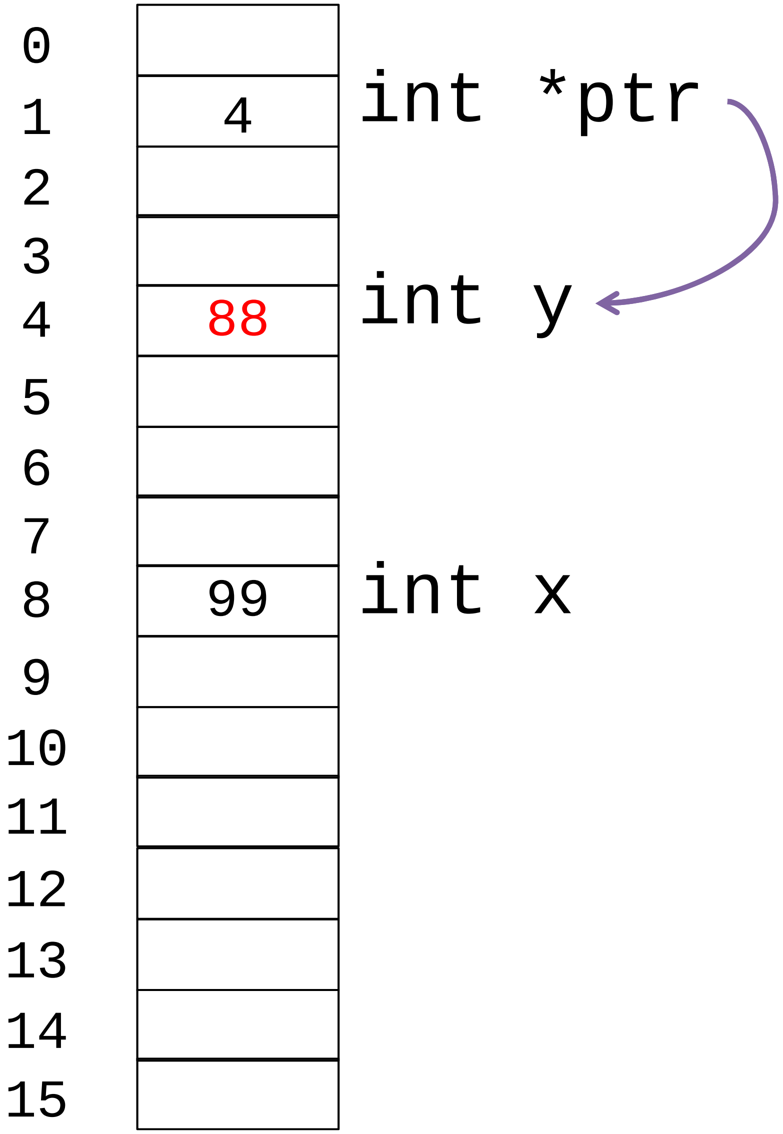


0		
1	4	int *ptr
2		
3		
4	2	int y
5		
6		
7		
8	99	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

```
int *ptr;
int x;
int y;
x = 1;
y = 2;
ptr = &x;
*ptr = 99;
ptr = &y;
*ptr = 88;
```



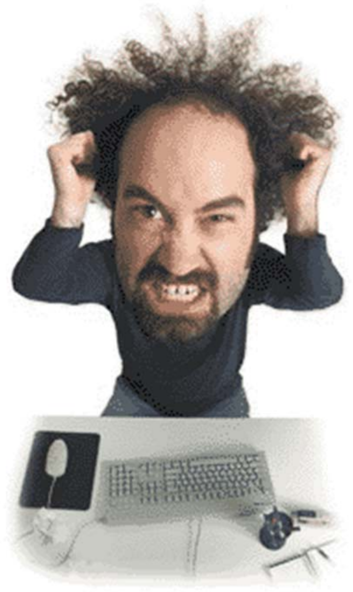
0		
1	4	int *ptr
2		
3		
4	88	int y
5		
6		
7		
8	99	int x
9		
10		
11		
12		
13		
14		
15		

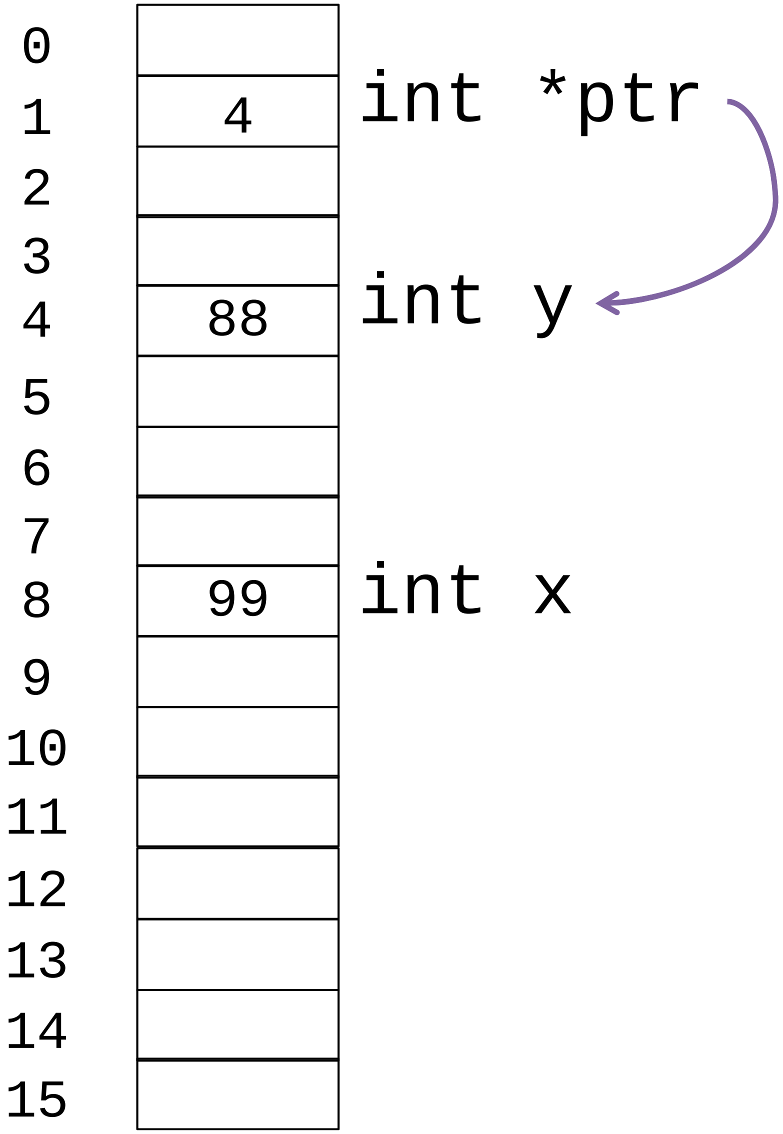
C Pointers

Let us look at this a little more carefully…

What if we do this?

```
int **dptr = &ptr;
```





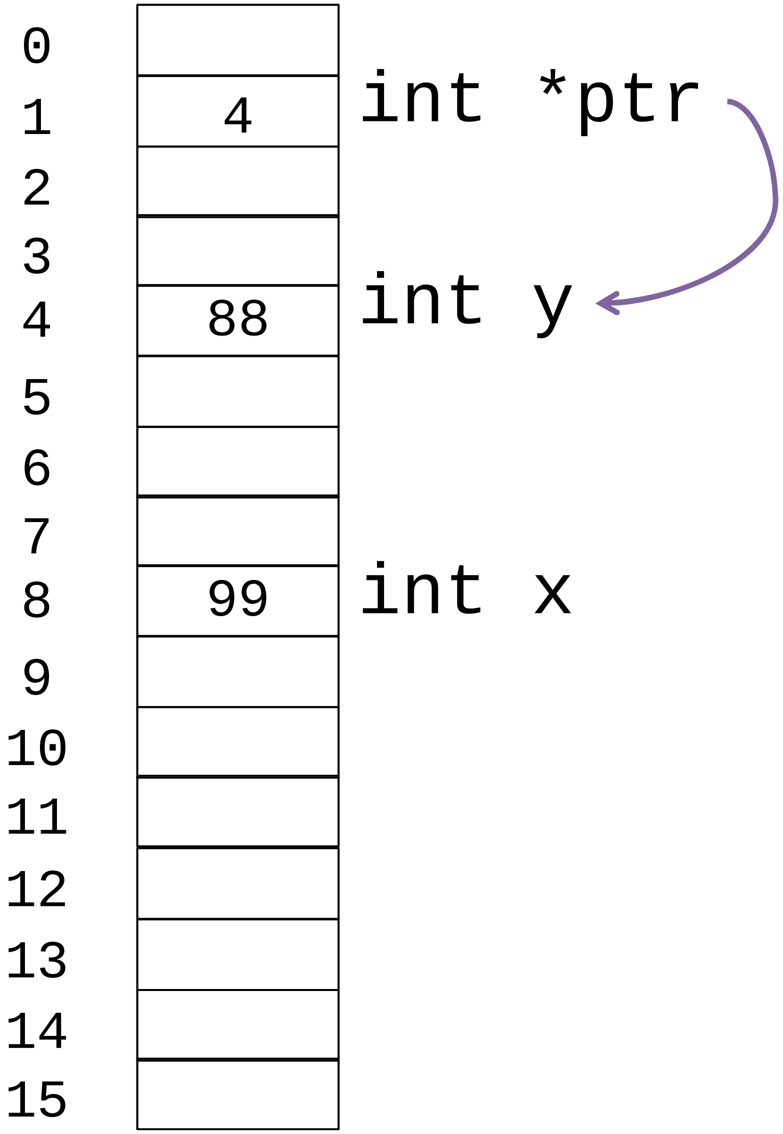
0		
1	4	int *ptr
2		
3		
4	88	int y
5		
6		
7		
8	99	int x
9		
10		
11		
12		
13		
14		
15		

C Pointers

Let us look at this a little more carefully…

What does this mean?

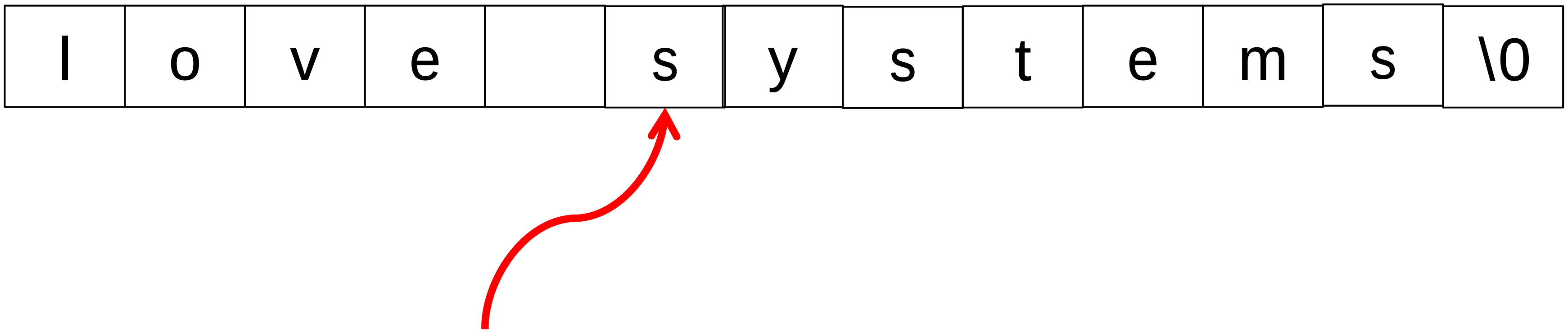
```
ptr = ptr + 1;
```



0		
1	4	int *ptr
2		
3		
4	88	int y
5		
6		
7		
8	99	int x
9		
10		
11		
12		
13		
14		
15		

C Strings Revisited

- **A C string is an array of characters**
  char str[] = “love systems”;



l	o	v	e		s	y	s	t	e	m	s	\0

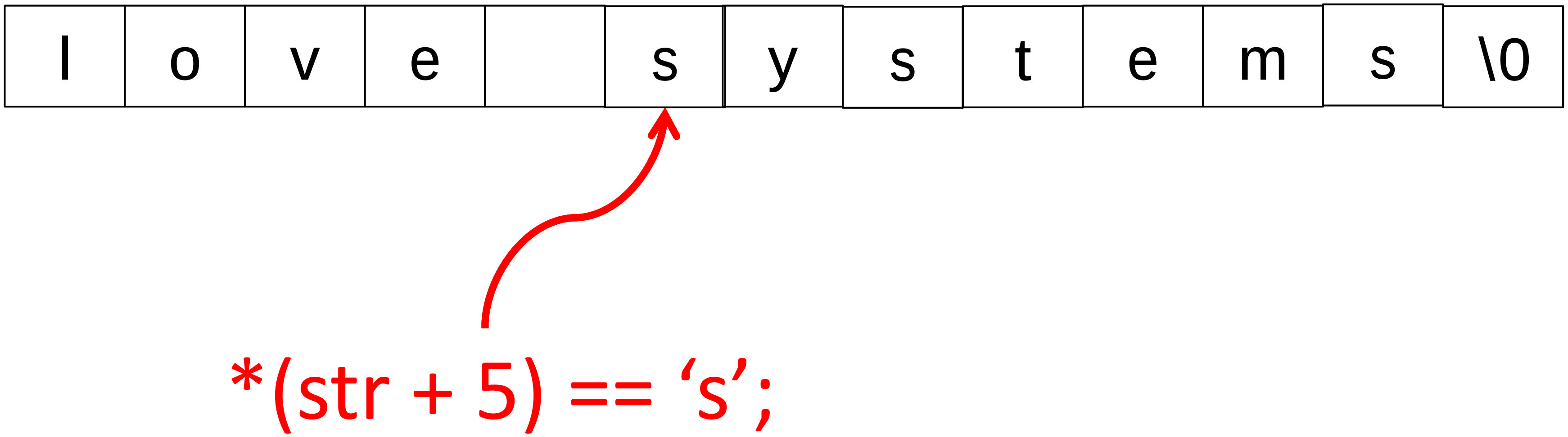
str[5] == ‘s’;

Remember this?

C Strings Revisited

- **A C string is an array of characters**

  char *str = “love systems”;



l	o	v	e		s	y	s	t	e	m	s	\0

*(str + 5) == ‘s’;

A C string can be defined as a pointer to char

Pointers and Arrays

- **Array Definitions**
  - char bytes[10];
  - int words[10];
- **Pointer Definitions**
  - char *bytes_p = bytes;
  - int *words_p = words;
- **Referencing Elements**
  - words_p[3] == *(words_p+3)
  - bytes_p[3] == *(bytes_p+3)

C Command Line Arguments

- **Program entry point is main**
- **main has two arguments:**
  - **argc:** the number of arguments
  - **argv:** the array of arguments
- **argv[0] is the program name**

```
int main(int argc, char *argv[]) {
  …
}
```

C Command Line Arguments

- **Program entry point is main**
- **main has two arguments:**
  - **argc:** the number of arguments
  - **argv:** the array of arguments
- **argv[0] is the program name**
- **What does char *argv[] mean?**

```
int main(int argc, char *argv[]) {
   …
}
```

C Parameter Passing

- **Pass-by-value**
  - Same as Java (all references/primitives)
  - The parameter is evaluated and bound to the corresponding variable in the function

```
void foo(int i) {
  i = 10; // Does not change i outside of function
}

int main() {
  int x = 5;
  foo(x);
}
```

C Parameter Passing

- **Pass-by-value (pointer)**
  - The parameter is a pointer
  - The referenced object can be manipulated

```
void bar(int *i) {
  *i = 20; // Does change *i outside of function
}

int main() {
  int x = 5;
  bar(&x);  // will change x
}
```