CSE 31
Computer Organization

Lecture 3 – C Pointers

Announcements

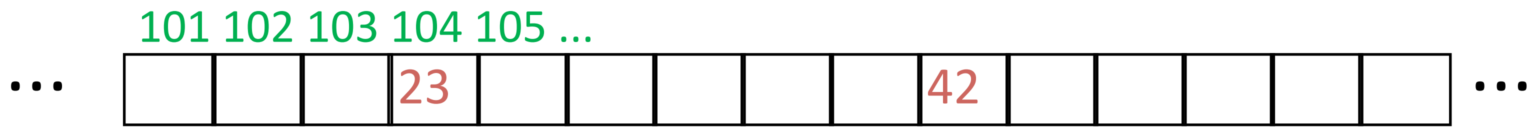
- Labs
  - Lab 1 (Introduction to C)
    - Due in **next** week (**no grace period**) due to error in posting
    - Make sure to demo your work to your TA (or me) before Assignment goes offline
    - Demo is REQUIRED to receive full credit
- Reading assignment
  - Chapter 4-6 of K&R (C book) to review C/C++ programming
  - Reading 01 (zyBooks 1.1 – 1.5) due 20-SEP
    - Complete *Participation Activities* in each section to receive grade towards Participation
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcement

- Homework assignment
  - Homework 01 (zyBooks 1.1 – 1.5) due 27-SEP
    - Complete *Challenge Activities* in each section to receive grade towards Homework
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

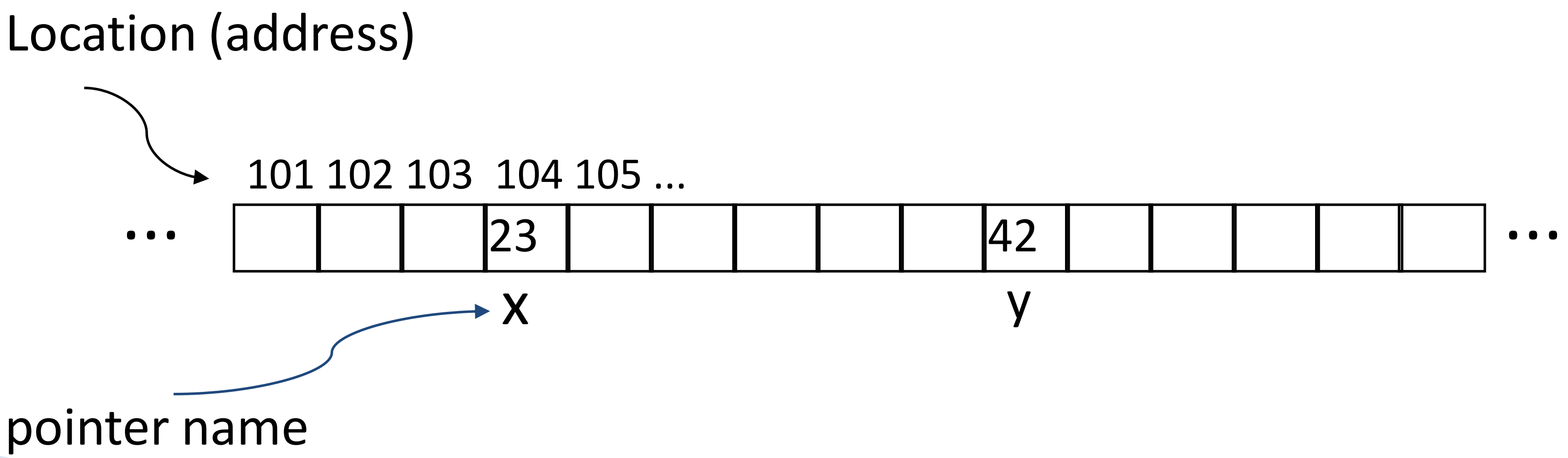
Address vs. Value

- Consider memory to be a single huge array:
  - Each cell of the array has an address associated with it.
  - Each cell also stores some value.
  - Do you think addresses use signed or unsigned numbers?
    - Negative address?!
- Don't confuse the address referring to a memory location with the value stored in that location.



Pointers

- An address refers to a particular memory location. In other words, it points to a memory location.
- Pointer: A variable that contains the address of a variable.



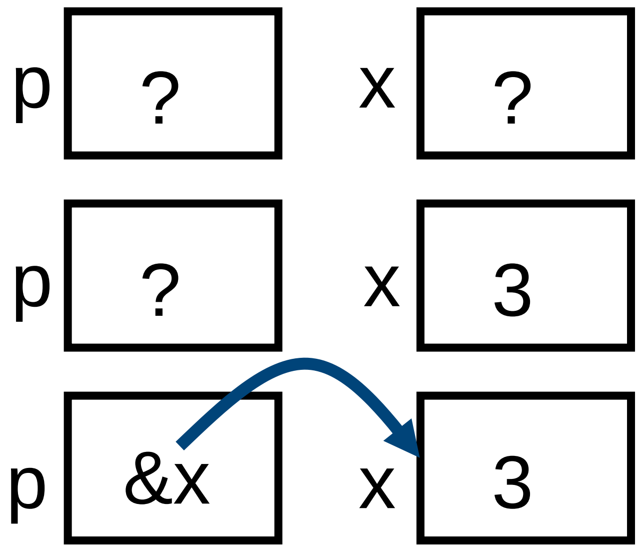
Pointers

- How to create a pointer:
  & operator: get address of a variable

```
int *p, x;

x = 3;

p =&x;
```



Note the "*" gets used 2 different ways in this example. In the declaration to indicate that p is going to be a pointer, and in the printf to get the value pointed to by p.

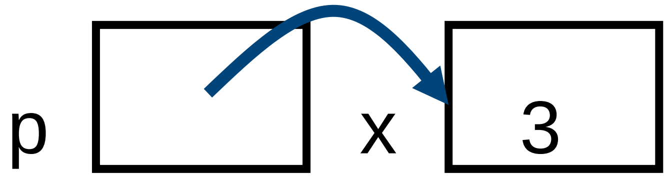
- How to get a value pointed to?

  * "dereference operator": get value pointed to

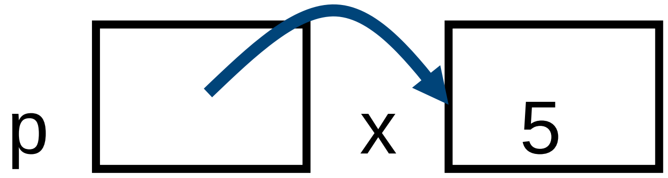
```
printf("p points to %d\n",*p);
```

Pointers

- How to change a variable pointed to?
  - Use dereference * operator on left of =



```
*p = 5;
```



Pointers and Parameter Passing

- Java and C pass parameters "by value"
  - procedure/function/method gets a copy of the parameter, so changing the copy cannot change the original

```
void addOne (int x) {
    x = x + 1;
}
int y = 3;
addOne(y);
```

`y` is still = 3 after the program ends!

Pointers and Parameter Passing

- How to get a function to change a value?

```
void addOne (int *p) {
   *p = *p + 1;
}
int y = 3;

addOne(&y);
```

Passing the reference of y

`y` is now = 4 after the program ends.

Pointers

- Pointers are used to point to any data type (`int`, `char`, a `struct`, etc.).
- Normally a pointer can only point to one type (`int`, `char`, a `struct`, etc.).
  - `void *` is a type that can point to anything (generic pointer)
  - Use sparingly to help avoid program bugs… and security issues… and a lot of other bad things!

Pointers & Allocation (1/2)

- After declaring a pointer:

  `int *ptr;`

  `ptr` doesn’t actually point to anything yet (it actually points somewhere - but we don’t know where!).
- We can either:
  - make it point to something that already exists, or
  - allocate room in memory for something new that it will point to… (we will talk about it later)

C Pointer Dangers

- Unlike Java, C lets you cast a value of any type to any other type without performing any checking.

```
    int x = 1000;
    int *p = x;              /* invalid */
    int *q = (int *) x;      /* valid */
```

- The first pointer declaration is invalid since the types do not match. (unsigned vs signed)
- The second declaration is valid in C but is almost certainly wrong
  - Is it ever correct?

More C Pointer Dangers

- Declaring a pointer just allocates space to hold the pointer – it does not allocate anything to be pointed to!
- Local variables in C are not initialized, they may contain anything.
- What does the following code do?

```
void f() {
    int *ptr;
    *ptr = 5;
}
```

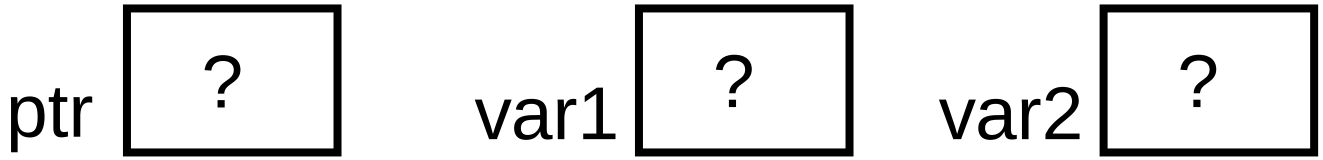
Where does it store the “5”?

Pointers & Allocation (2/2)

- Pointing to something that already exists:

```
int *ptr, var1, var2;
```

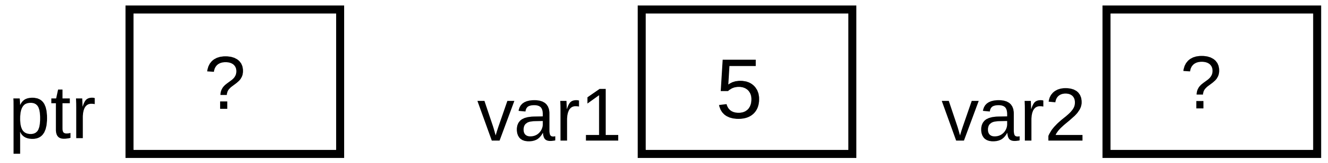
- `var1` and `var2` have room implicitly allocated for them.



Pointers & Allocation (2/2)

- Pointing to something that already exists:

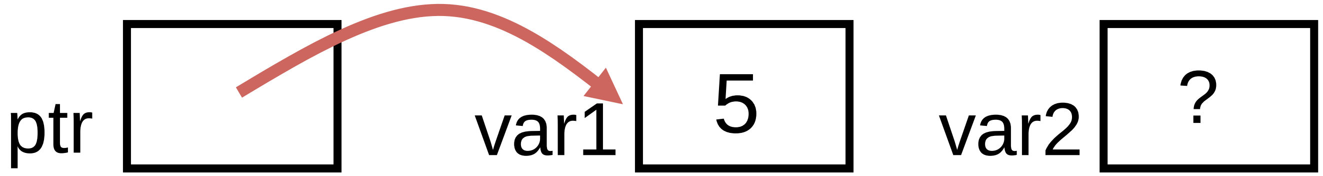
```
int *ptr, var1, var2;
var1 = 5;
```



Pointers & Allocation (2/2)

- Pointing to something that already exists:

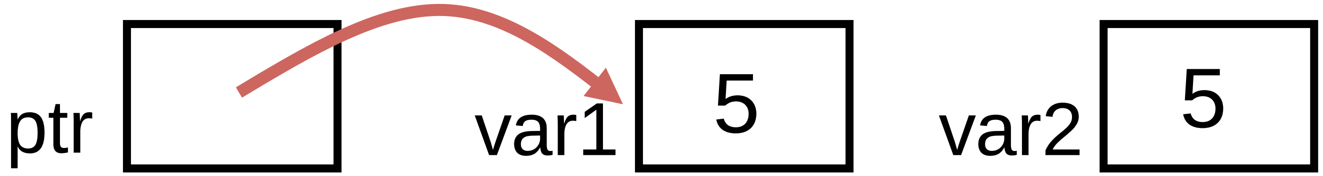
```
int *ptr, var1, var2;
var1 = 5;
ptr  = &var1;
```



Pointers & Allocation (2/2)

- Pointing to something that already exists:

```
int *ptr, var1, var2;
var1 = 5;
ptr  = &var1;
var2 = *ptr;
```


ptr
var1
5
var2
5