# Pointers (l4.pdf)

(or expression...)

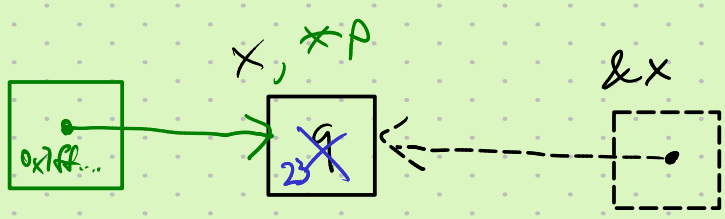
Idea: a pointer is just a variable that stores a memory address.

Example w/ pictures:

```
int x = 9;
int* p = &x;
```

`int*` — datatype of p

`&x` — "address of" operator

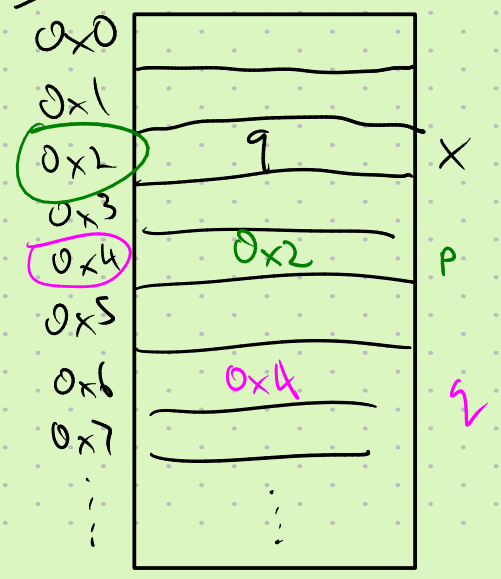



Note: assignment of pointers: makes LHS pointer point to the same place as RHS pointer.

After p = &x,
*p IS x.

```
// How to read from a pointer:
cout << *p; // prints 9
*p = 23;
cout << x; // prints 23
```

Alternate picture:





```
int** q = &p;
```

Note: int* p ≡ int *p ≡ int * p

≡ int*p.

Think: p has type int* ⟺ *p has type int.

(int* p;) (int *p;)

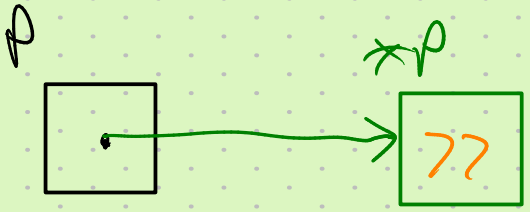
Note: for any datatype T, T* is the datatype for "pointer to thing of type T".

---

Why??

Dynamic memory allocation.

How to allocate memory as your program runs?
(How do vectors work (v.push_back(...))?)




int* p;

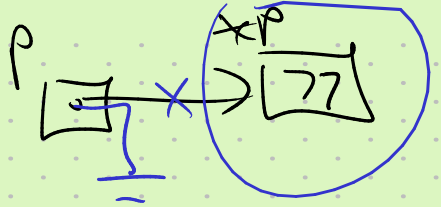
p = new int;

type int*

new: ① allocate memory
② tell you where to find it.

*p = 77;

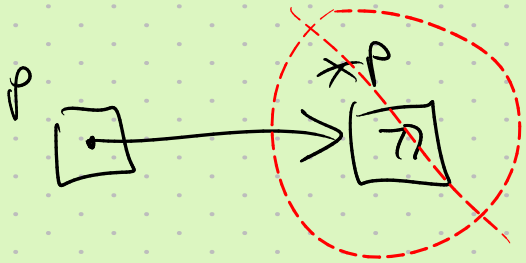
p = 0;




if we do this, the integer is lost!
No way to refer to it, no way to deallocate it...

Deallocation:

delete p;

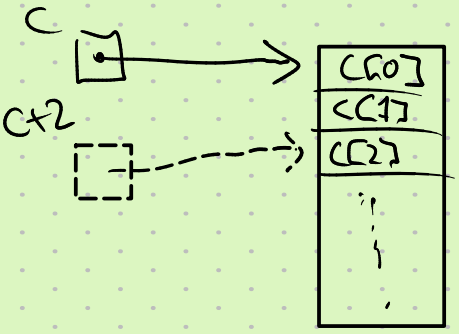



(memory marked as "free")

⊗ Important to deallocate for long running programs (e.g. a web server)

## Allocating larger chunks:

```
int* C = new int[100];
// allocates space for 100
// integers...
```




```
C[0] = 7;
```

Note: C[i] ≡ *(C+i)

Trivia: $c[i] \equiv *(c+i) \equiv *(i+c)$

$\equiv i[c]$