Today's Plan

Upcoming:

- A1 ongoing
- Q1 next week

Last time:

- Testing

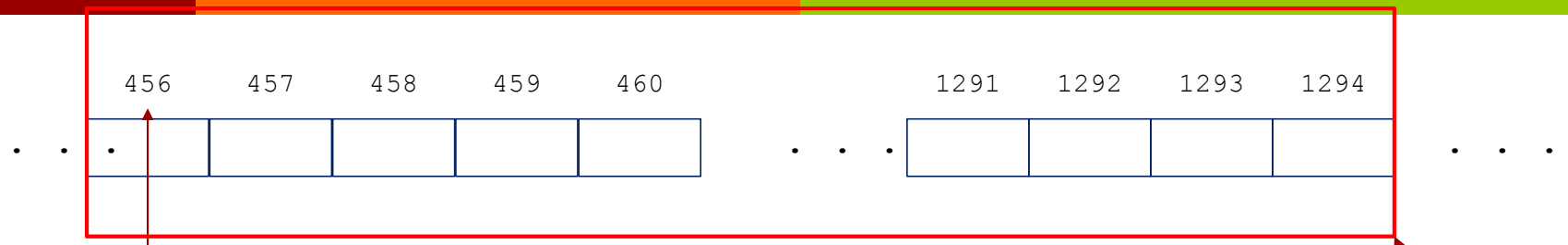
Today's topics:

- From last time:
  - Property & LLM Testing
- Pointers & Memory Management
- Memory of a Running C++ Program

Pointers and Memory Management

- Managing memory is a major topic in C++
- While **call-stack memory** is automatically managed, **free store memory** is manually managed by the programmer
  - Many other programming languages automatically manage memory using a special program called a **garbage collector** that runs while your program runs
  - The garbage collector automatically de-allocates any unused memory
  - Garbage collectors can result in slightly slower programs, or programs with short pauses, which might not be acceptable in real-time applications
- In C++, to use free store memory you must use pointers …

Memory of a running C++ program



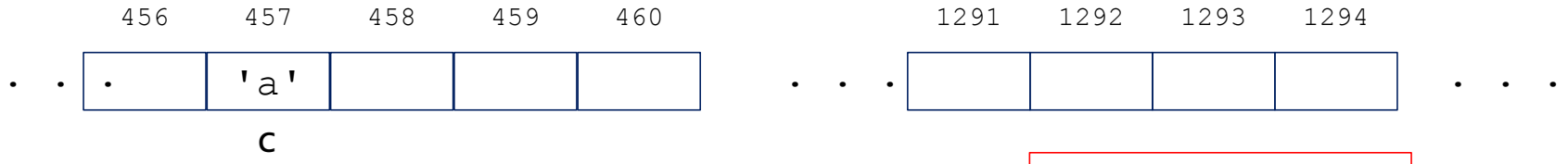
This starting address is probably **different every time you run the program**. 456 is an arbitrary value

You can think of the memory of a running C++ program as a long **array of bytes**.

But this array probably **doesn't start at address 0**: it is placed in a free region of memory called **the heap** by the operating system.

When you run a program, the operating system (OS) gives your program a chunk of memory it can use. Accessing memory out of these bounds usually makes the OS kill your program

Memory of a running C++ program




```
char c = 'a';

cout << c;  // prints value of c:

cout << &c; // prints address of c:
```

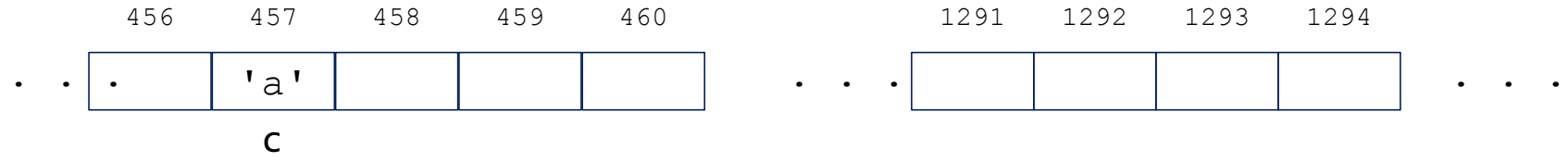
& is the **address-of operator**: &x returns the address of variable x

The 0x at the start means **hexadecimal** (**base 16**). Each digit/letter is 4 bits.

```
  value of c: a
address of c:
0x7ffdf3f9a7d7

  value of c: a
address of c:
0x7fff62a536c7

  value of c: a
address of c:
0x7fff6ba0fa47
```

3 separate runs of the code; different address for **c** each time

Memory of a running C++ program

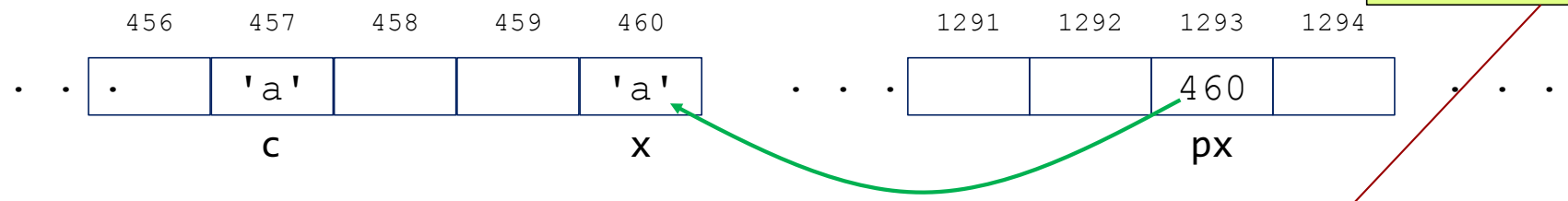



```
char c = 'a';

cout << c; // prints value of c: 'a'

cout << &c; // prints address of c: 457

char x = c;  // put a copy of c's value in x

        px = &x; // store the address of x in px
```

Question: what is the type of an address?

Memory of a running C++ program

*px is called a **pointer de-reference**: it is the value in the address px stores

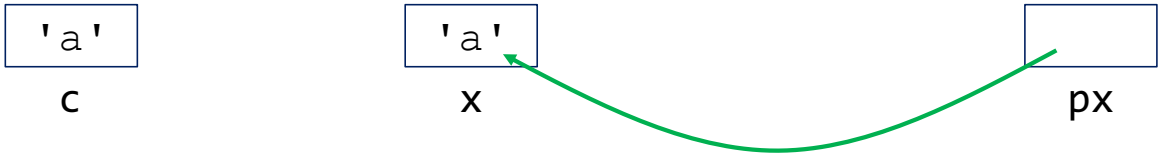



```
char c = 'a';

cout << c; // prints value of c: 'a'

cout << &c; // prints address of c: 457

char x = c;  // put a copy of c's value in x

char* px = &x; // store address of x in px
```

```
cout << px;  // print value of px:

cout << *px; // print value at
             // address contained
             // in px:
```

Address 460 now has two names: **x** and ***px**

Memory of a running C++ program



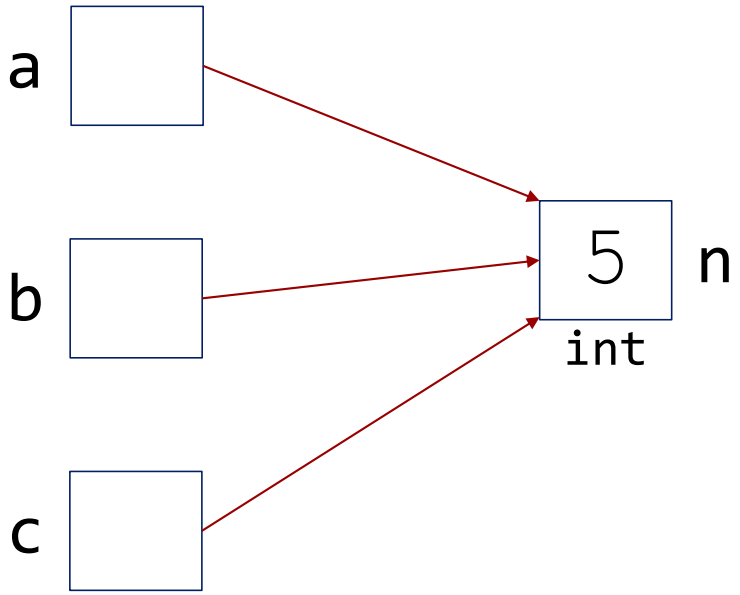
Usually we just draw the variables we care about with a specific address …

```
char c = 'a';

cout << c; // prints value of c: 'a'

cout << &c; // prints address of c: 457

char x = c;  // put a copy of c's value in x

char* px = &x; // store address of x in px
```

```
cout << px; // print value of px: 460

cout << *px; // print value at
             // address contained
             // in px: 'a'
```

Memory of a running C++ program

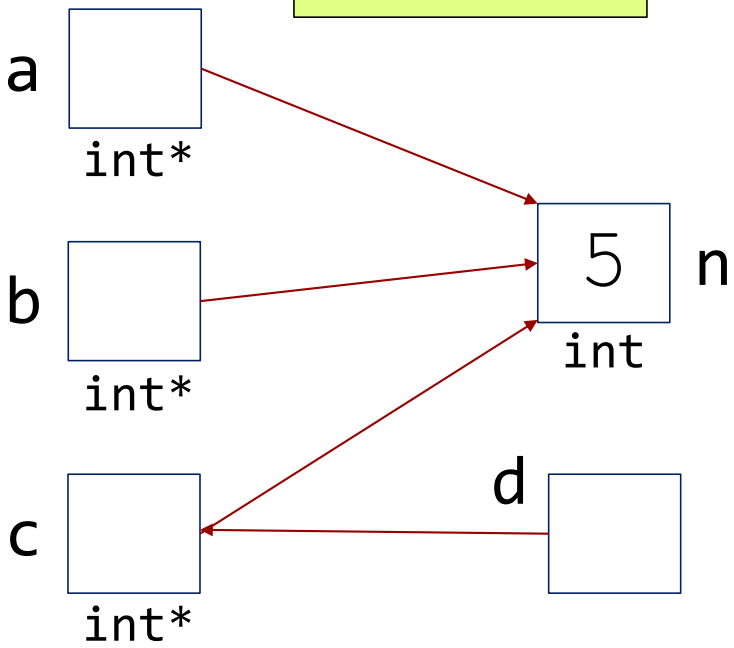
Challenge
Write a program that makes memory look like this.



Memory of a running C++ program

Challenge
Change it to look like this

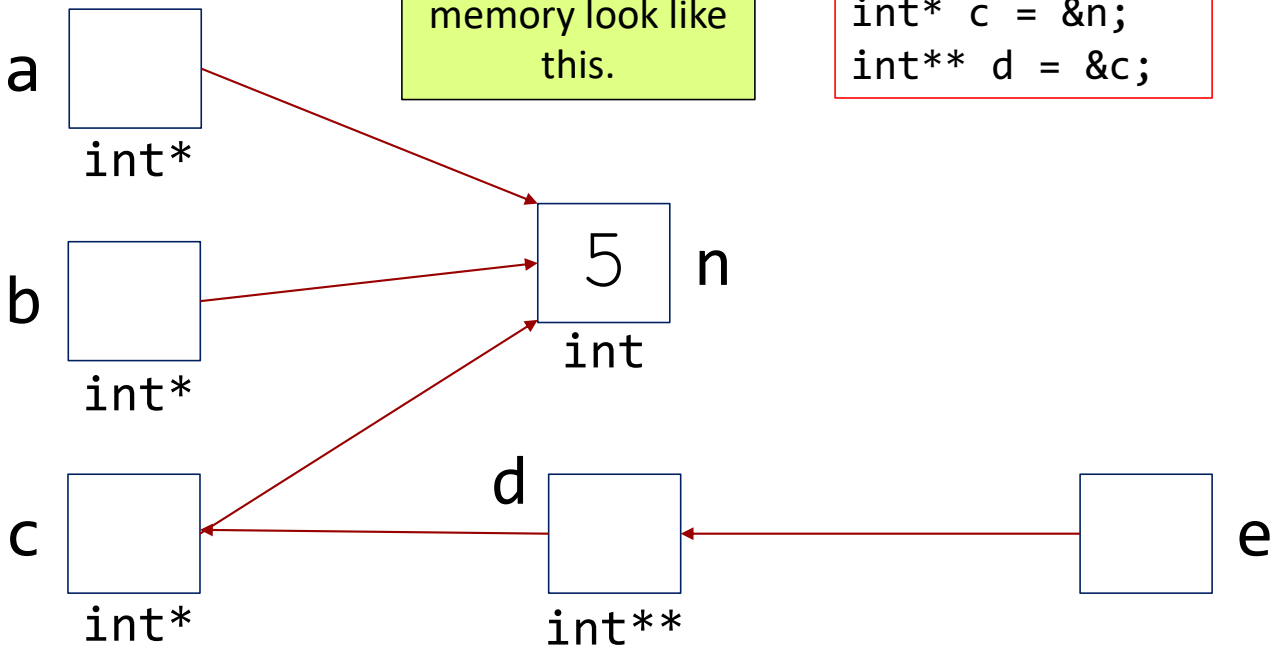
```
int n = 5;
int* a = &n;
int* b = &n;
int* c = &n;
```



Memory of a running C++ program

Challenge
Modify the program to make memory look like this.

```
int n = 5;
int* a = &n;
int* b = &n;
int* c = &n;
int** d = &c;
```

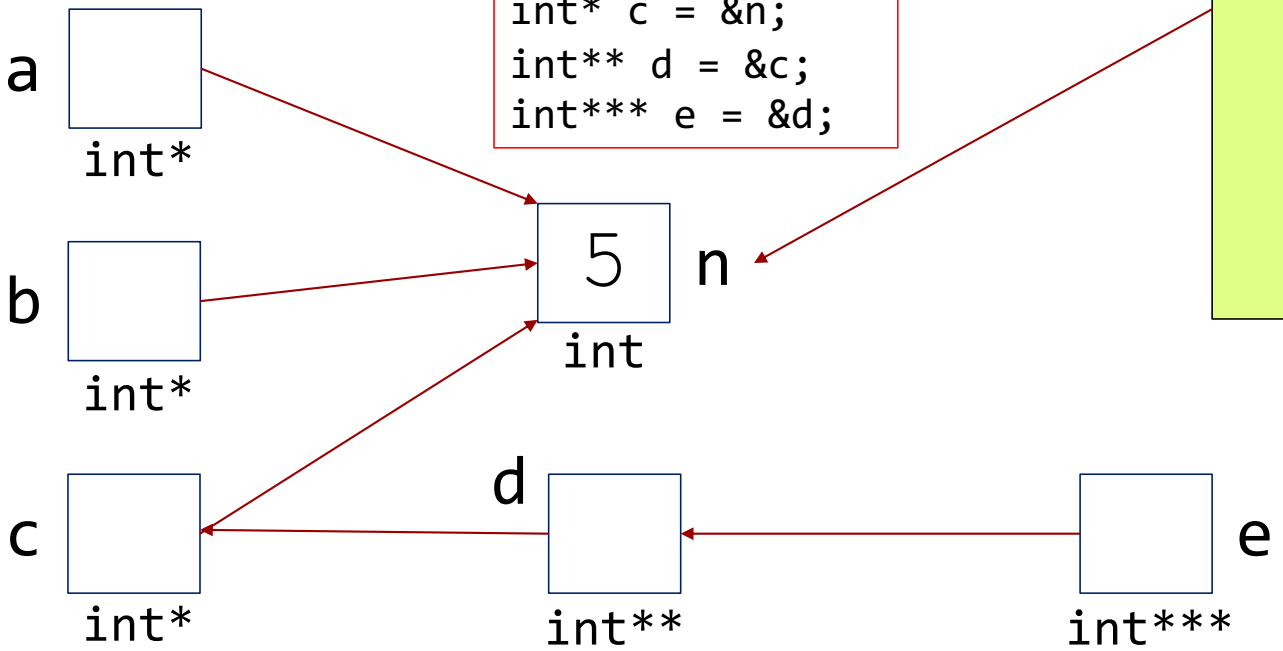


Memory of a running C++ program

```
int n = 5;
int* a = &n;
int* b = &n;
int* c = &n;
int** d = &c;
int*** e = &d;
```

Different names (**aliases**) for the int that n refers to:

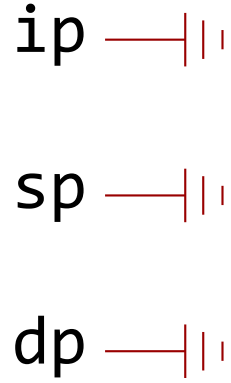
n
*a
*b
*c
**d
***e



The null Pointer

- **nullptr** is a special pointer value that means the pointer is **not** pointing to a valid memory location
- In earlier versions of C++, 0 is used instead of `nullptr`. But use `nullptr` instead: it's clearer.
- De-referencing a `nullptr` is **always** an error, e.g. `*ip` is always an invalid expression
  - Must always check the pointer before de-referencing it!

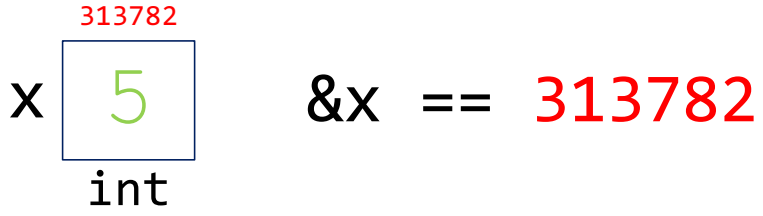
```
int*    ip = nullptr;
string* sp = nullptr;
double* dp = nullptr;
```



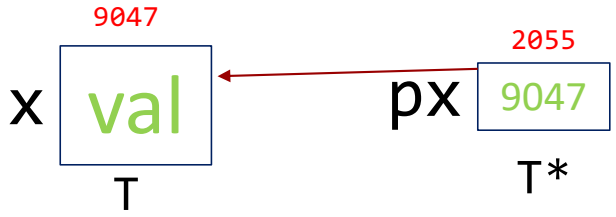
The "ground" symbol from electronics represents a null pointer.

Some general rules

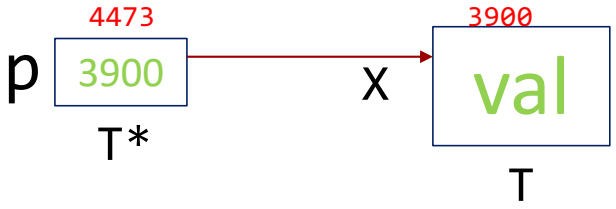
If x is a variable, then &x is the address where the value of x is stored



If T is a C++ type, then T* is the type of a pointer to a value of type T



If p points to a value, then *p evaluates to the value being pointed to



If `p == nullptr`, then evaluating `*p` is always an error