Designing Abstractions

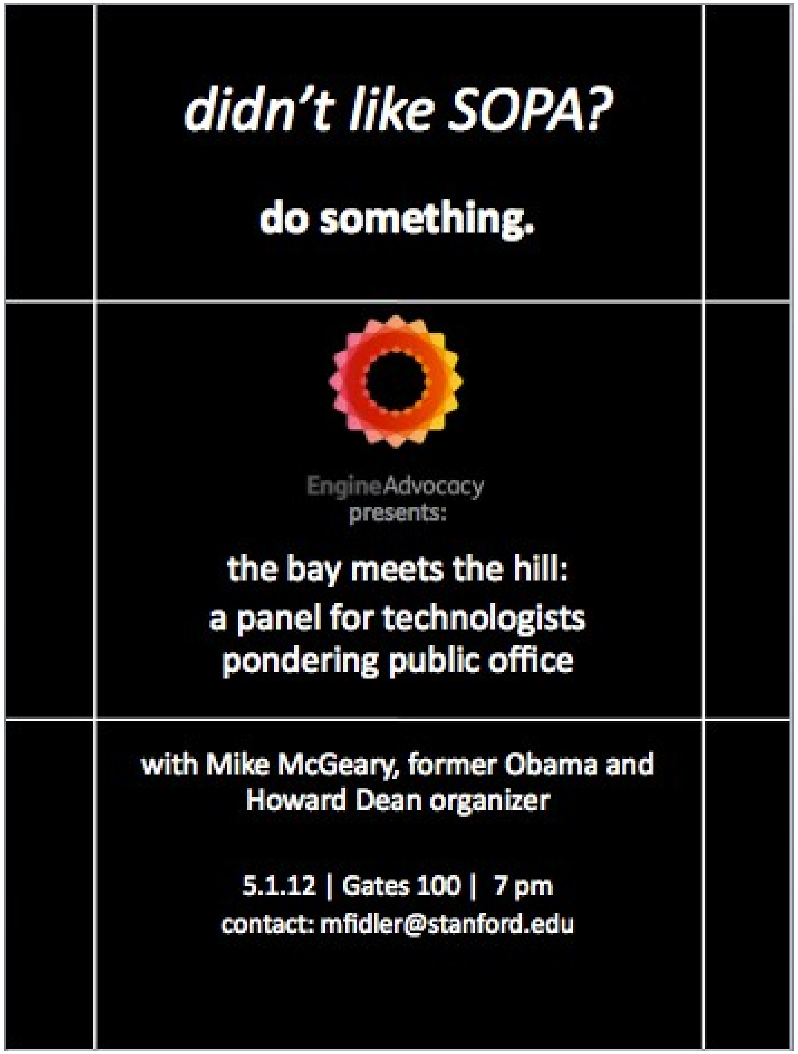
Announcements

- Assignment 3 due right now.
- Assignment 4: **Boggle** out, due next Friday, May 11.
  - Play around with a really cool application of recursion.
  - Write a computer program that can trounce you at a word game!

Announcements

- Casual dinner for women studying computer science tomorrow at 6:15PM in Gates 219.
  - Good food, good company.
  - Everyone is welcome!

Announcements



didn't like SOPA?

do something.

EngineAdvocacy
presents:

**the bay meets the hill:
a panel for technologists
pondering public office**

with Mike McGeary, former Obama and
Howard Dean organizer

5.1.12 | Gates 100 | 7 pm
contact: mfidler@stanford.edu

- Panel tomorrow night about technology and public office.
- 7PM tonight in Gates 100.

Announcements

- Midterm Exam #1 this Thursday, May 3 from 7:00PM – 9:00PM.
- Location by last name:
    - A – J: Go to Braun Auditorium
    - K – R: Go to Hewlett 201
    - S – Z: Go to Braun Lecture Hall
- Open-book, open-note, but **closed-computer**.
- Covers material up to and including last Friday's lecture on big-O and sorting.
- Alternate exams: We'll email out dates/times later today.

Fundamental Question #1

How do our tools work?

Fundamental Question #2

How do we build new tools?

Fundamental Question #3

How do we analyze our tools?

Classes

- `Vector`, `Stack`, `Queue`, `Map`, etc. are **classes** in C++.
- Classes contain
  - An **interface** specifying what operations can be performed on instances of the class.
  - An **implementation** specifying how those operations are to be performed.
- To define our own classes, we must define both the interface and the implementation.

Random Bags

- A **random bag** is a data structure similar to a stack or queue.
- Supports two operations:
  - **Add**, which adds an element to the random bag, and
  - **Remove random**, which returns and removes a random element from the bag.
- Has several applications:
  - Random maze generation
  - Shuffling decks of cards.

Let's Code it Up!

Defining Classes in C++

- First, create a **header file** containing the interface of your class.
- Then, create a **source file** containing the implementation of your class.
- Lots of details; in interest of space, consult the course reader for details.

Language Philosophy

- Every programming language exports some set of **primitives**:
  - Primitive data types (**`int`**, **`char`**, etc.)
  - Functions
  - Classes
  - etc.
- We can use those primitives to construct a larger set of primitives:
  - **`Vector`**, **`RandomBag`**, etc.

Where Does it Stop?

- The ADTs we've been using are not primitives in C++; they are defined in terms of other language features.
- Understanding those features will let us analyze their efficiency.
- Understanding those features will let us build other interesting abstractions.

A Quick Aside: Pages and URLs

- To visit webpages, you can just provide a URL that indicates what page you want to look up.
- Every page contains content, but also has a URL by which it can be referred to.
- There is a distinction between the page itself (the actual content) and the link to the page (a way of referring to the page).

A Quick Aside: Phone Numbers

- To talk to one of your friends, you can call their phone given their phone number.
- Your friends are all wonderful people, and they probably have phone numbers that can be used to refer to them.
- There is a distinction between your friends and their phone numbers.

A Quick Aside: Files and Filenames

- To read or write data on your computer, you can open a file with a given name.
- Most files have names that refer to them, and some files can contain the names of other files.
- There is a distinction between a file and a filename.

So What?

- These systems all have a distinction between **objects** and **names for objects**.
- We can look up the object given the name.
- This leads to key pieces of C++ design.

Memory Addresses

- Every object in C++ is physically located somewhere in memory.
- The location is called its **address**.
- Intuitively, think of the address as a link to the object, or a phone number for the object, or a name for the object.
- Given a variable, you can obtain its address by using the **address-of operator** (**&**):

```
cout << &myVariable << endl;
```

Pointers

- A **pointer** is a C++ variable that stores the address of an object.
- Given a pointer to an object, we can get back the original object.
  - Can then read the object's value.
  - Can then write the object's value.
- Think of a pointer as a URL for the object.

Pointers

- Setting up a pointer requires two steps:
    - Declare the pointer variable.
    - Initialize the pointer variable to refer to some object.
- These are all separate steps, and forgetting any one can result in hard-to-find bugs.
- Once the pointer is set up, we can use it to read and write the object it refers to.

Pointers

Setting up a pointer requires two steps:

- Declare the pointer variable.
- Initialize the pointer variable to refer to some object.

These are all separate steps, and forgetting any one can result in hard-to-find bugs.

Once the pointer is set up, we can use it to read and write the object it refers to.

Declaring a Pointer Variable

- In C++, pointers encode two pieces of information:
  - What object is being referred to?
  - What type of object is that?
- To declare a pointer that refers to an object of type ***T***, declare a variable of type ***T****:

```
T* variableName;
```

Pointers

- Setting up a pointer requires two steps:
  - Declare the pointer variable.
  - Initialize the pointer variable to refer to some object.
- These are all separate steps, and forgetting any one can result in hard-to-find bugs.
- Once the pointer is set up, we can use it to read and write the object it refers to.

Pointers

Setting up a pointer requires two steps:

- Declare the pointer variable.
- Initialize the pointer variable to refer to some object.

These are all separate steps, and forgetting any one can result in hard-to-find bugs.

Once the pointer is set up, we can use it to read and write the object it refers to.

Choosing What to Point To

- Now that we have a pointer, we should set it to point to some object!
- Pointers store addresses, so if we want our pointer to point at an object, we can assign the pointer the address of that object.
- For example:

```
int* myPtr = &myVariable;
```

- The object being pointed at is called the **pointee**.

Pointers

- Setting up a pointer requires two steps:
    - Declare the pointer variable.
    - Initialize the pointer variable to refer to some object.
- These are all separate steps, and forgetting any one can result in hard-to-find bugs.
- Once the pointer is set up, we can use it to read and write the object it refers to.

Pointers

- Setting up a pointer requires two steps:
  - Declare the pointer variable.
  - Initialize the pointer variable to refer to some object.
- These are all separate steps, and forgetting any one can result in hard-to-find bugs.
- Once the pointer is set up, we can use it to read and write the object it refers to.

Using a Pointer

- Once we have a pointer that points at some object, we can **dereference** the pointer to read and write that object.
- To dereference a pointer, prefix it with a **`*`**, as shown here:

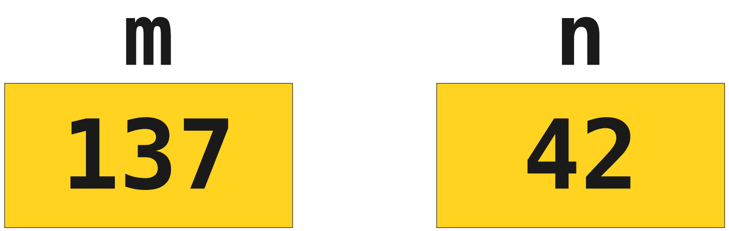
```
*ptr = 137;
cout << *ptr << endl;
```

Pointers, Visually

```
int m = 137;
int n = 42;
```

Pointers, Visually

```
int m = 137;
int n = 42;
```



m

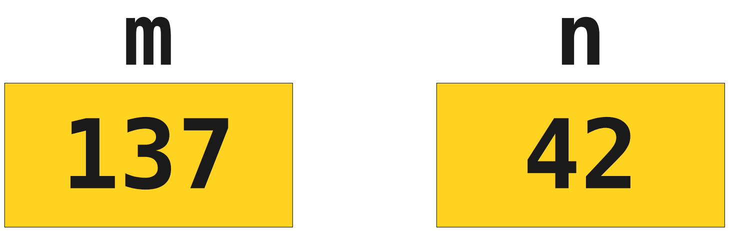
137

n

42

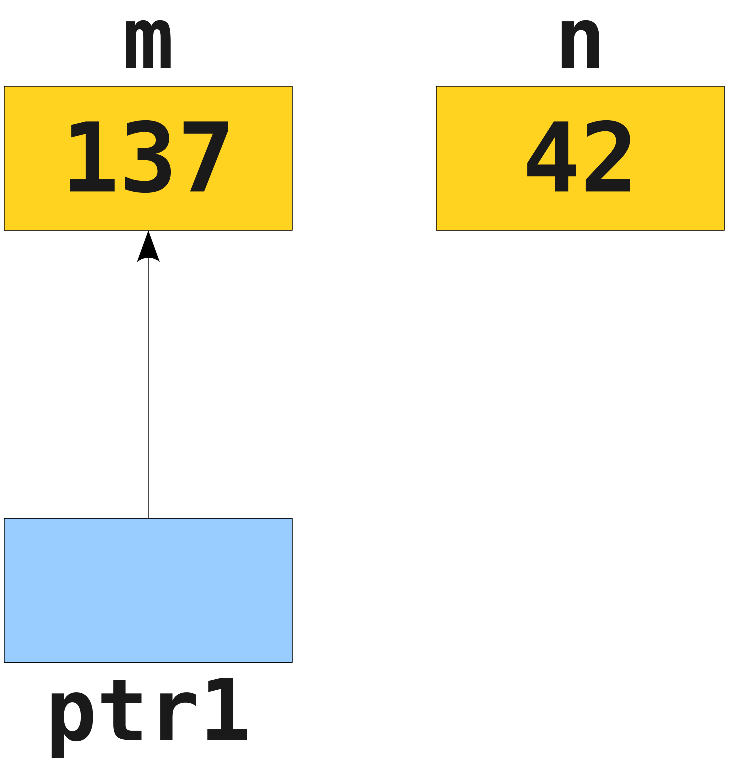
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
```



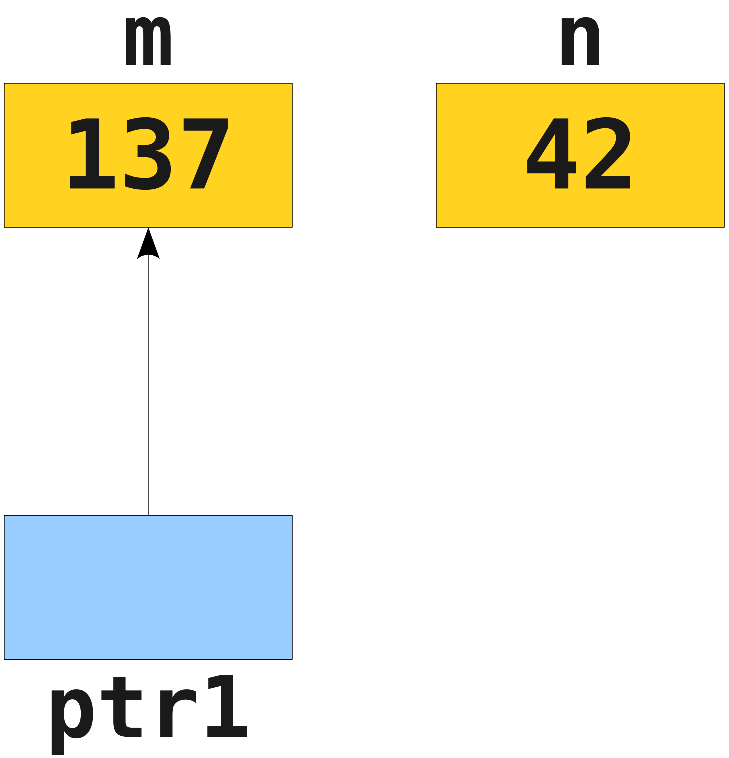
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
```



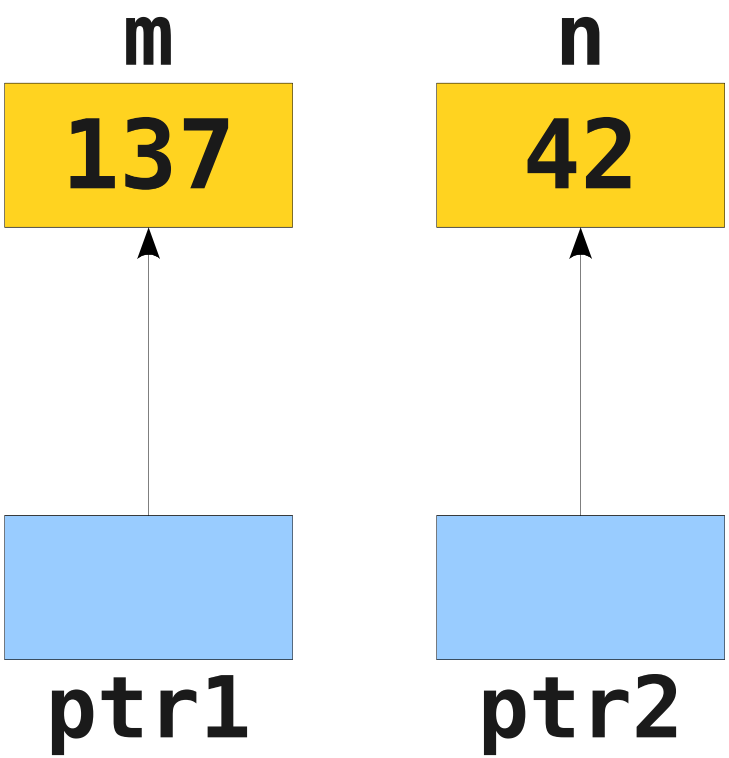
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;
```



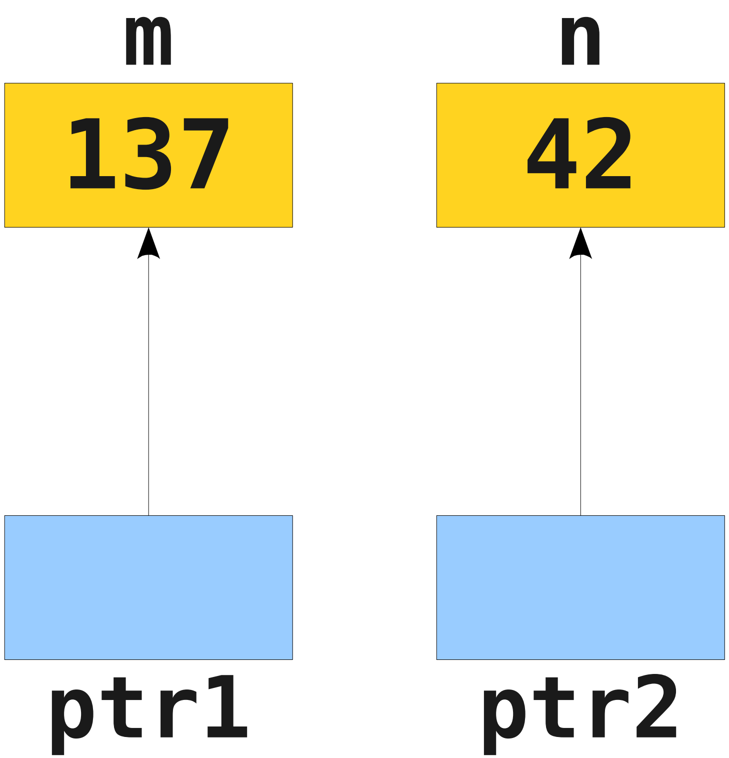
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;
```



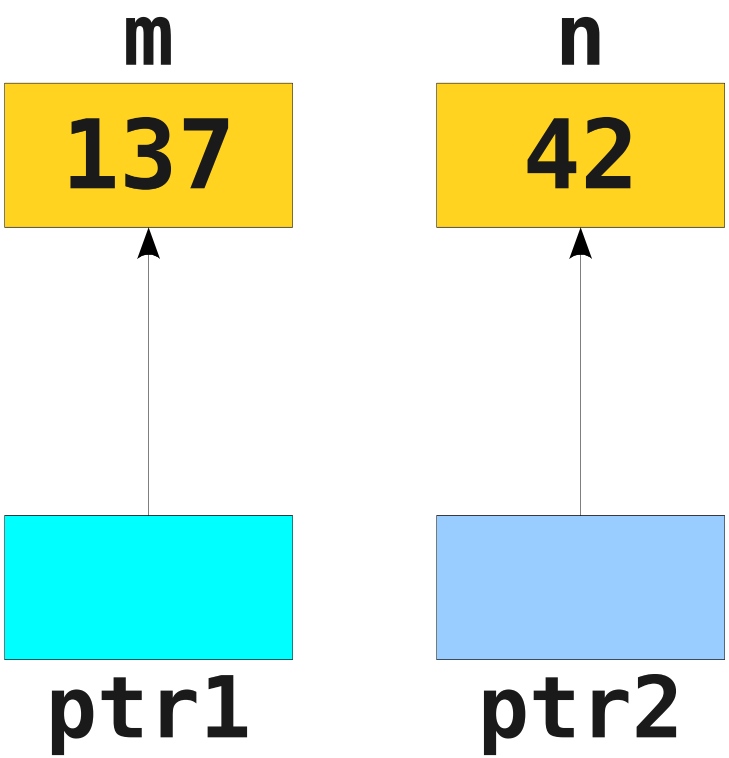
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```



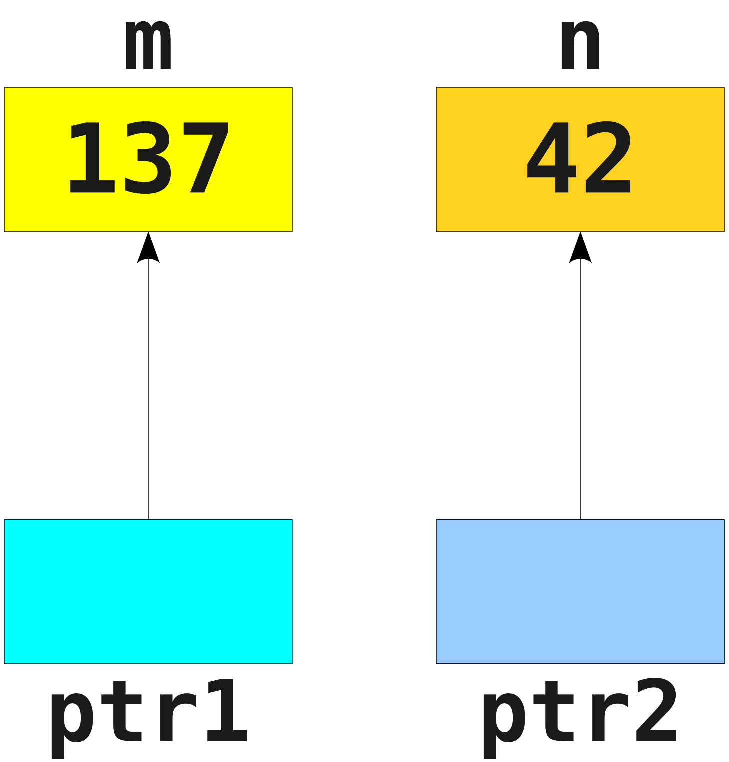
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```



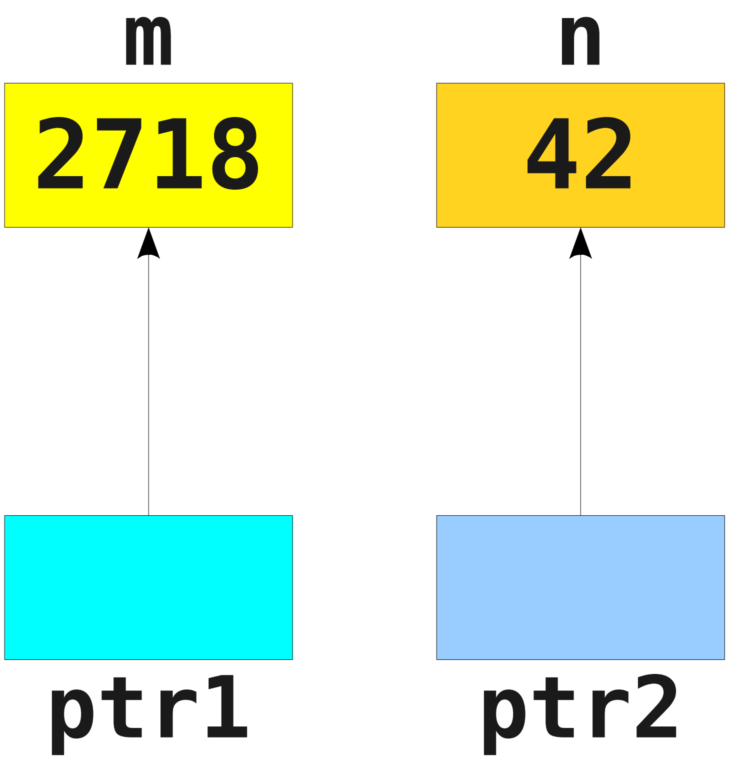
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```



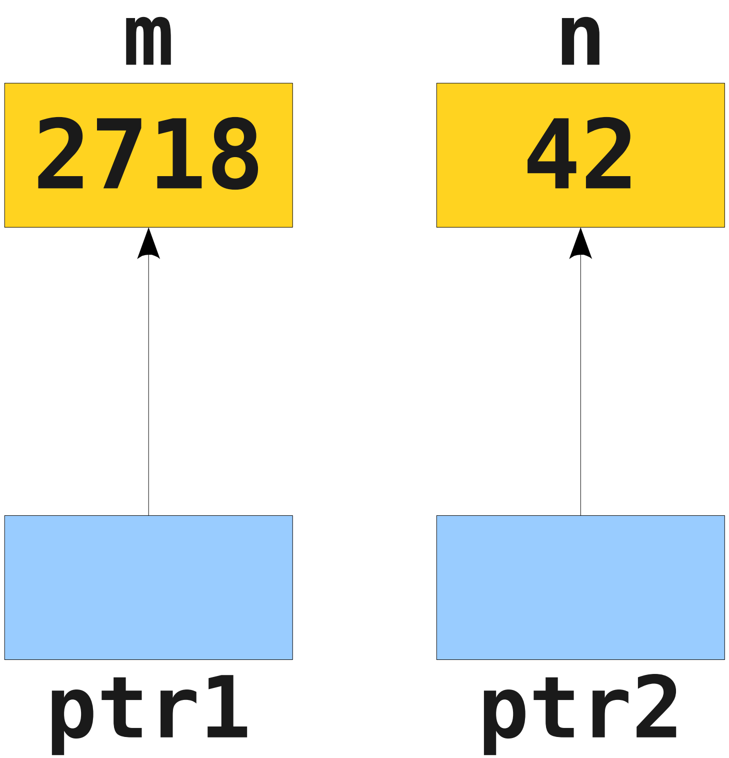
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```



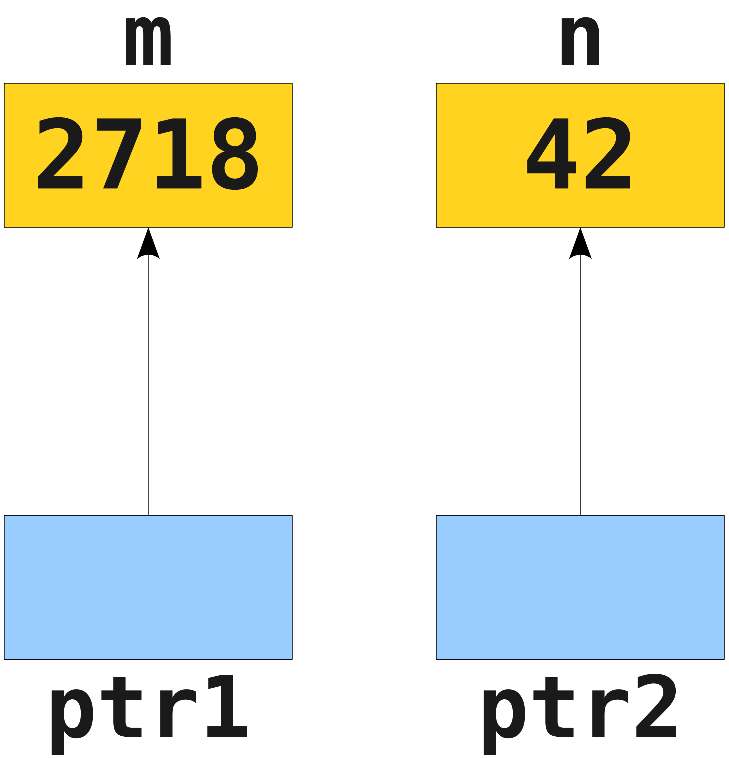
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
```



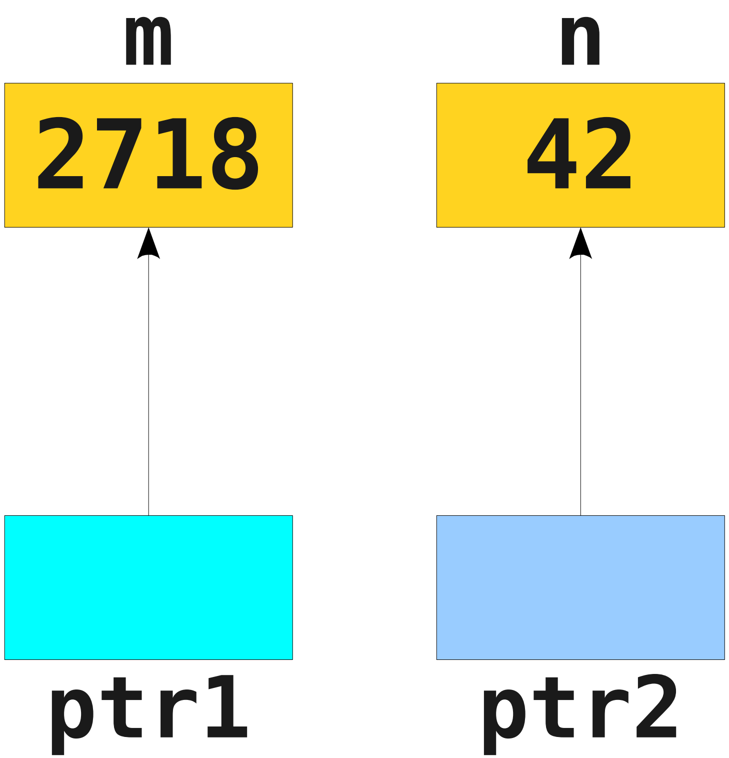
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



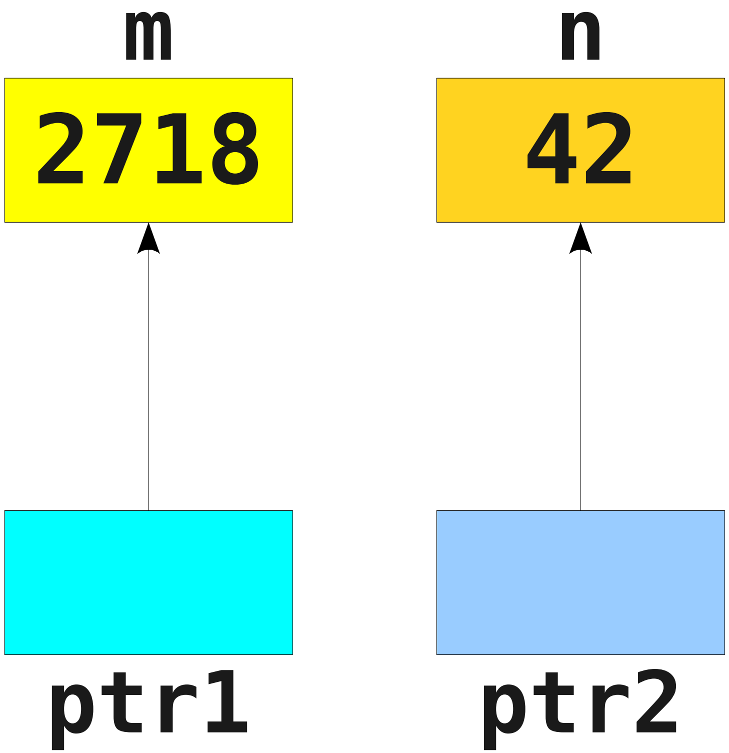
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



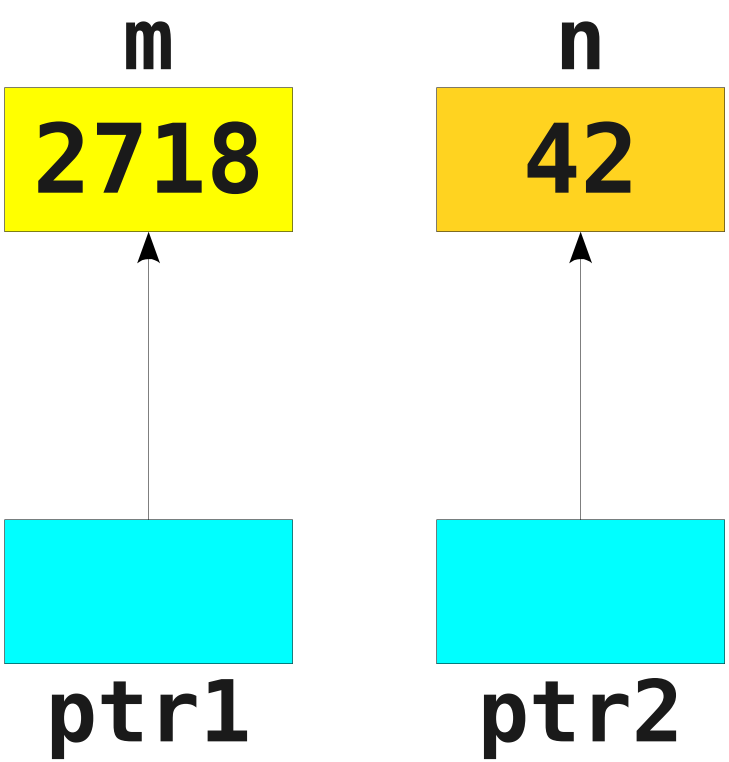
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



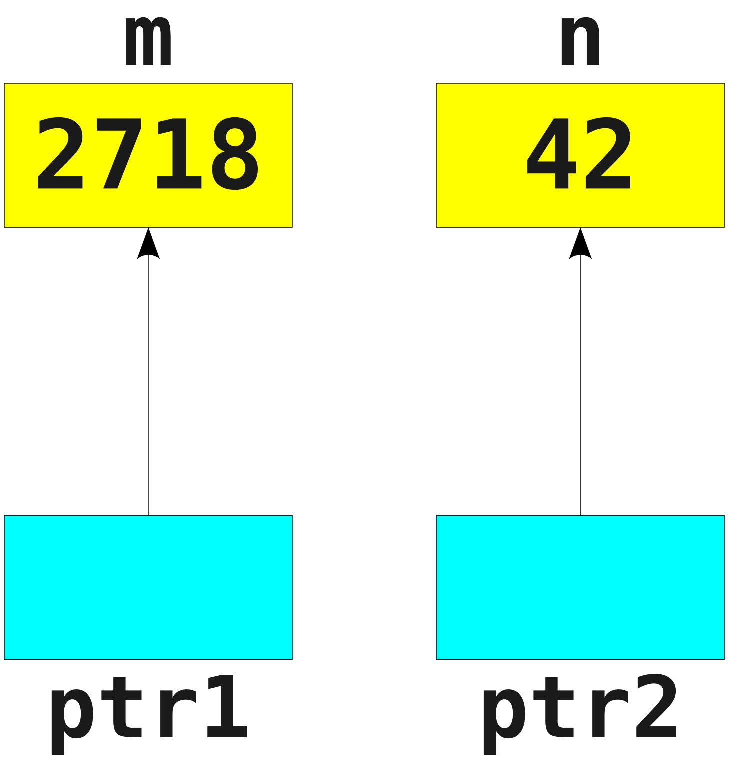
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



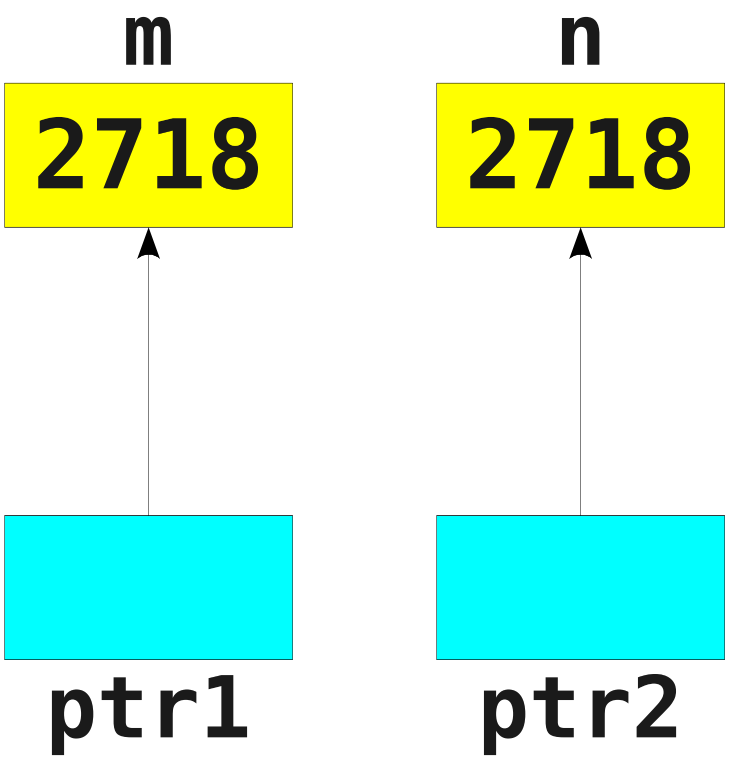
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



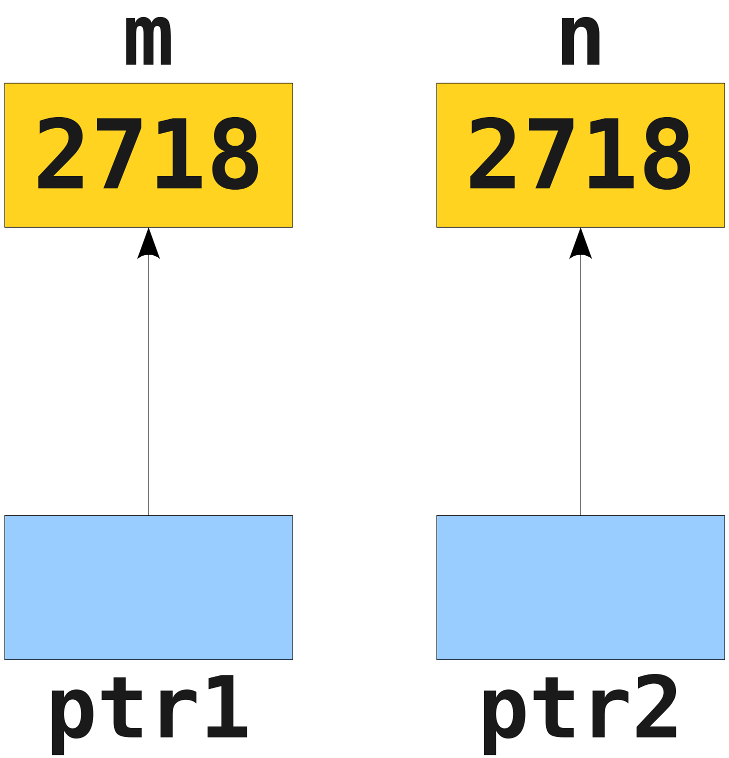
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



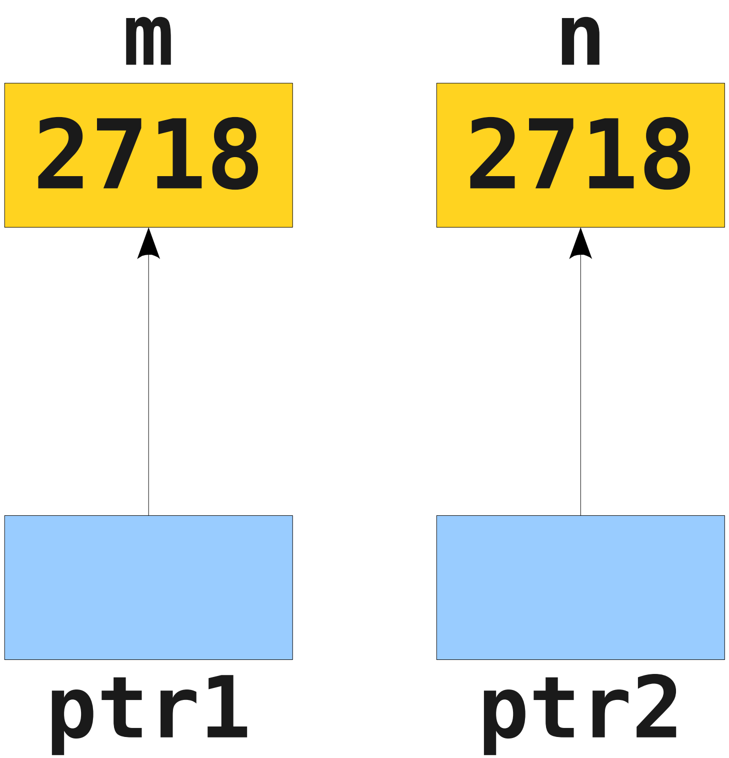
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
```



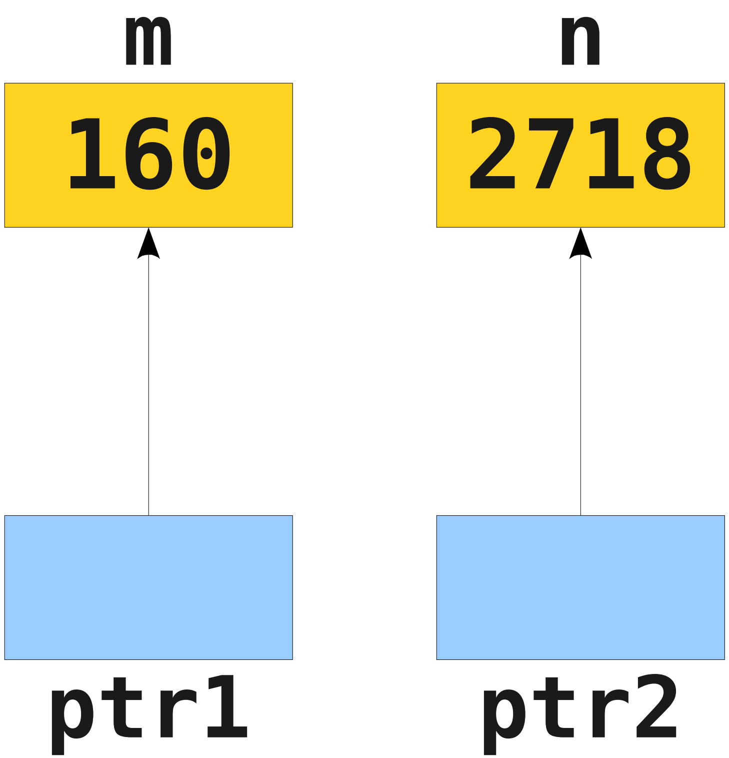
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;
```



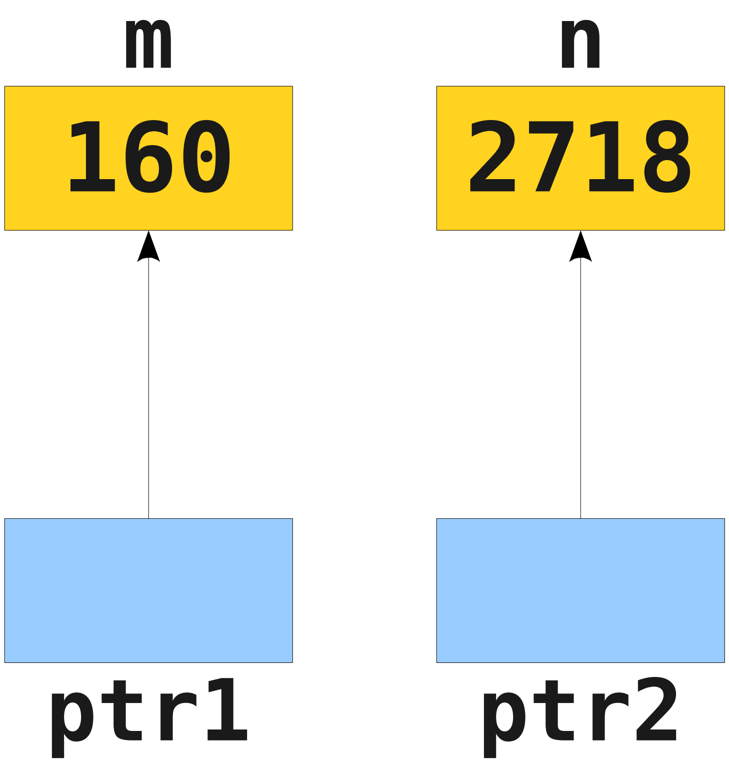
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;
```



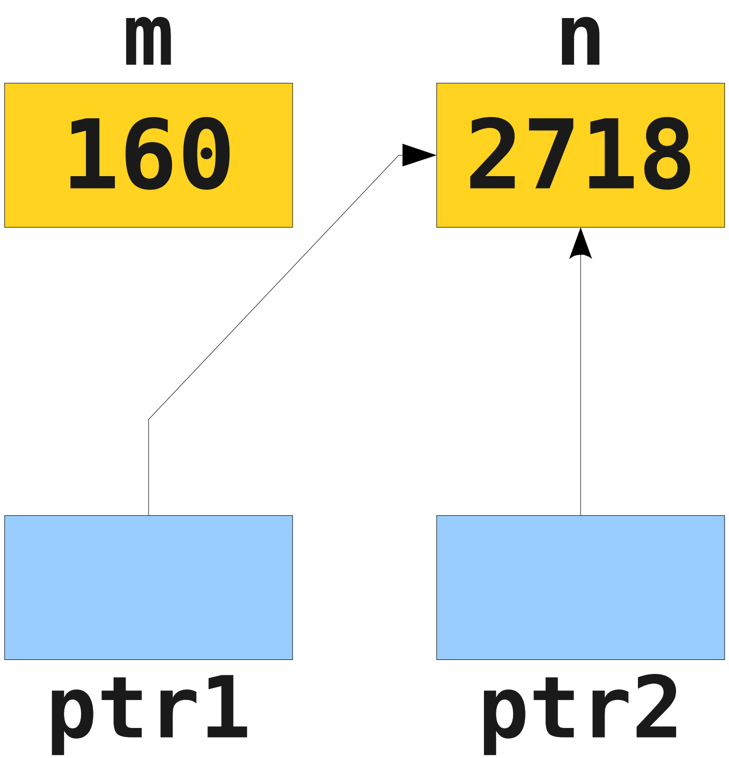
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```



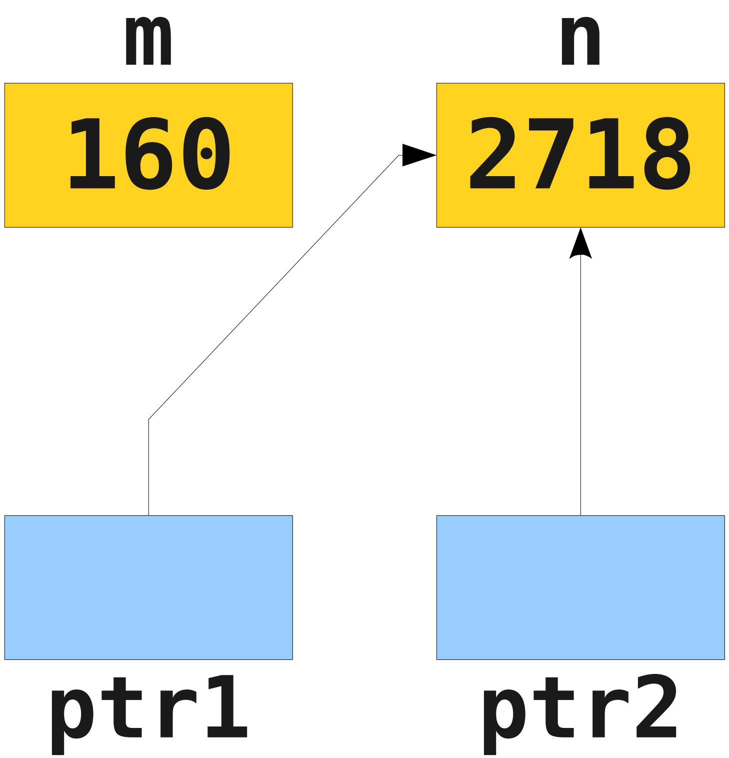
Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```



Pointers, Visually

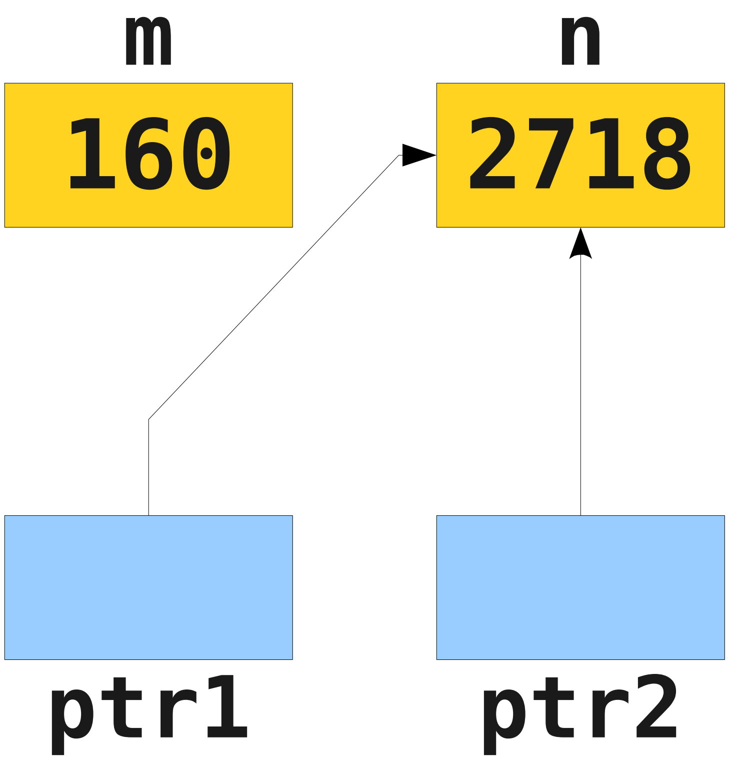
```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```



Assigning one pointer to another changes which object is being pointed at. It does not change the value of the pointee.

Pointers, Visually

```
int m = 137;
int n = 42;

int* ptr1 = &m;
int* ptr2 = &n;

*ptr1 = 2718;
*ptr2 = *ptr1;
m = 160;

ptr1 = ptr2;
```


m
n
160
2718
ptr1
ptr2