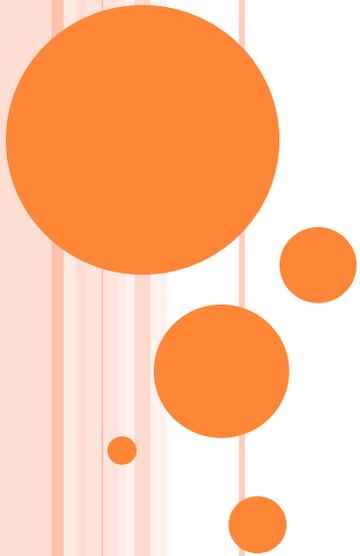


# RECORDS



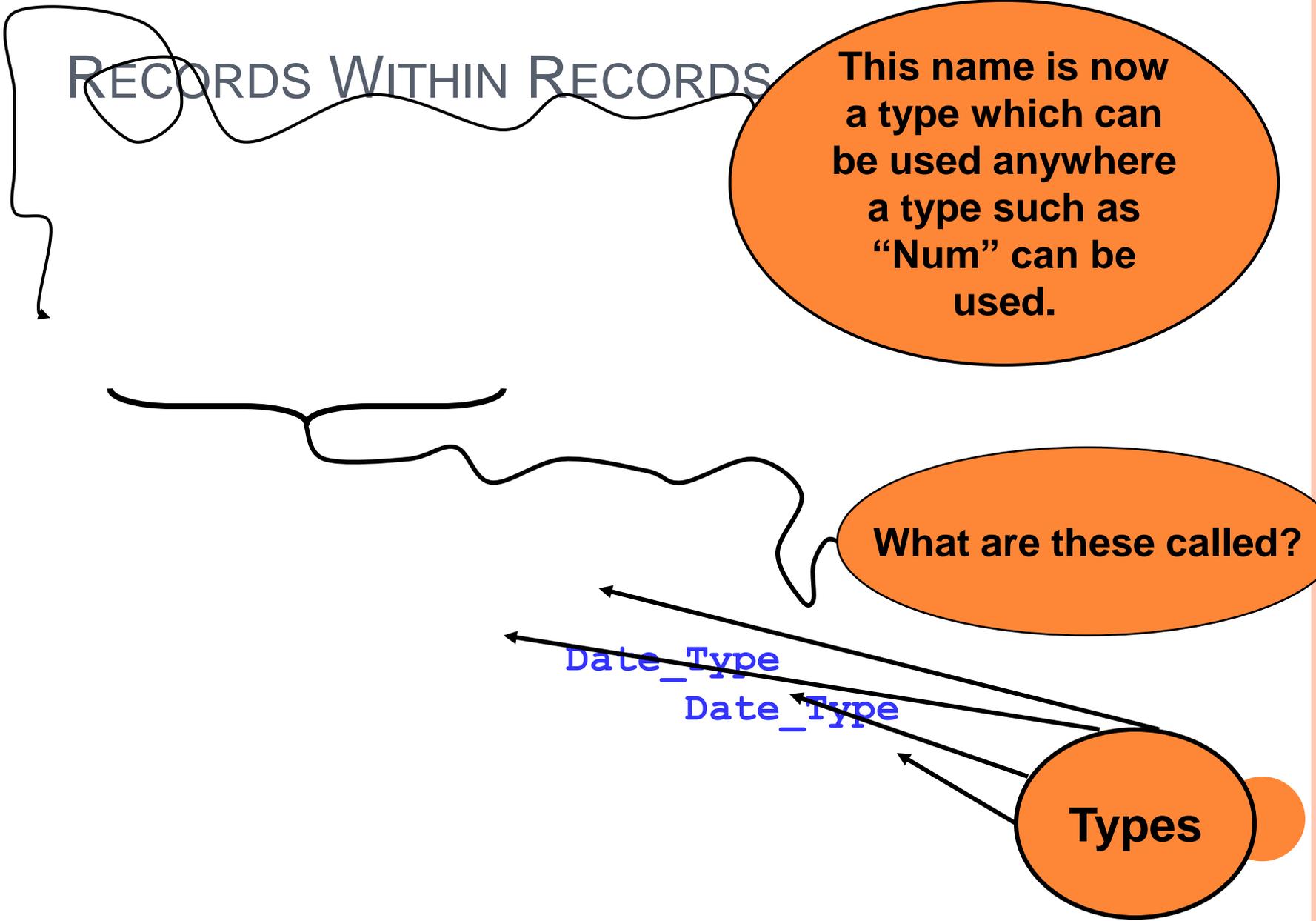
# RECORDS WITHIN RECORDS

This name is now a type which can be used anywhere a type such as "Num" can be used.

What are these called?

Date\_Type  
Date\_Type

Types



## RECORD WITHIN RECORDS

**Date\_Type:**

<b>day</b>	<b>month</b>	<b>year</b>
------------	--------------	-------------

**Student\_Type:**

	<b>name</b>	<b>gpa</b>	
<b>birth_day</b>	<b>day</b>	<b>month</b>	<b>year</b>
<b>graduation_day</b>	<b>day</b>	<b>month</b>	<b>year</b>

```
bob isoftype Student_Type
bob.birth_day.month <- 6
```



# TYPES VS. VARIABLES

## ○ TYPE Definitions

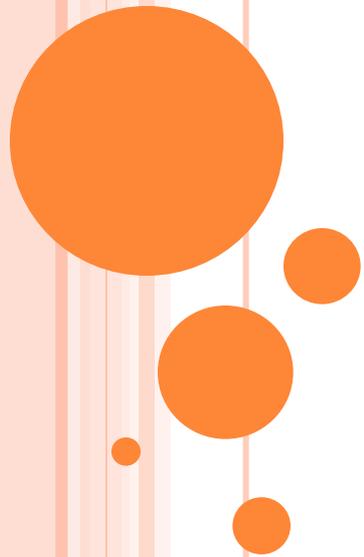
- Create **templates** for new kinds of variables
- Do not create a variable – no storage space is allocated
- Have **unlimited scope**

## ○ VARIABLE Declarations

- Actually create storage space
- Have **limited scope** - only module containing the variable can “see” it
- Must be based on an **existing** data type



# DYNAMIC MEMORY AND POINTERS



# DYNAMIC VS. STATIC

## Static (fixed in size)

- Sometimes we create data structures that are **“fixed”** and don't need to grow or shrink.

## Dynamic (change in size)

- Other times, we want the ability to **increase and decrease** the size of our data structures to accommodate changing needs.



# STATIC DATA

- **Static data**

- **“lives” for as long as that module is active.**



# DYNAMIC DATA

- **Dynamic data** refers to data structures which can grow and shrink to fit changing data requirements.
- We can **allocate** (create) additional dynamic variables whenever we need them.
- We can **de-allocate** (kill) dynamic variables whenever we are done with them.
- A key advantage of dynamic data is that we can always have a **exactly** the number of variables required - no more, no less.
- For example, with **pointer** variables to connect them, we can use dynamic data structures to create a **chain of data structures** called a **linked list**.



## NOTE

- **Dynamic data gives us more flexibility**
- **Memory is still limited**
- **But now we can use it where we need it**
- **And we can determine that while the program is running**

**Examples?**

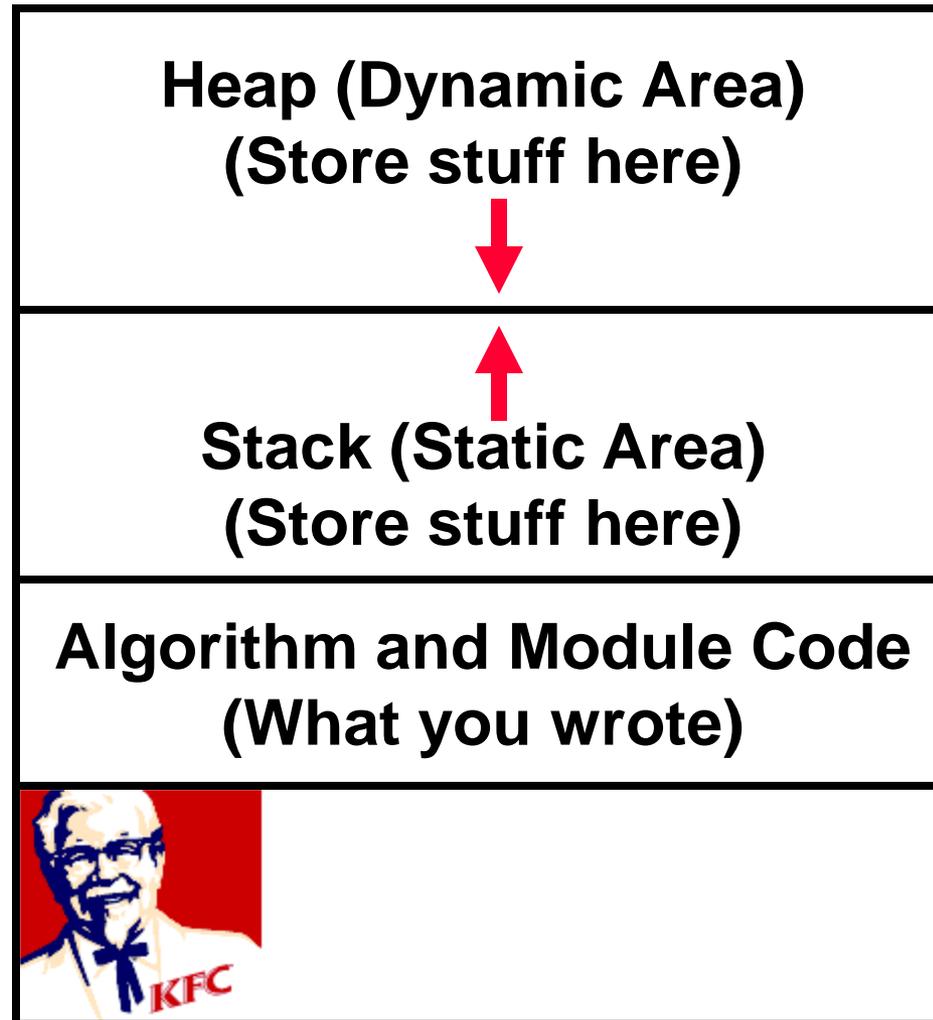
**Printer Queues**

**Airliners**

**uh, everything?**



# A VIEW OF MEMORY



## A LIST EXAMPLE

- We must maintain a list of data
- Sometimes we want to use only a little memory:



- Sometimes we need to use more memory



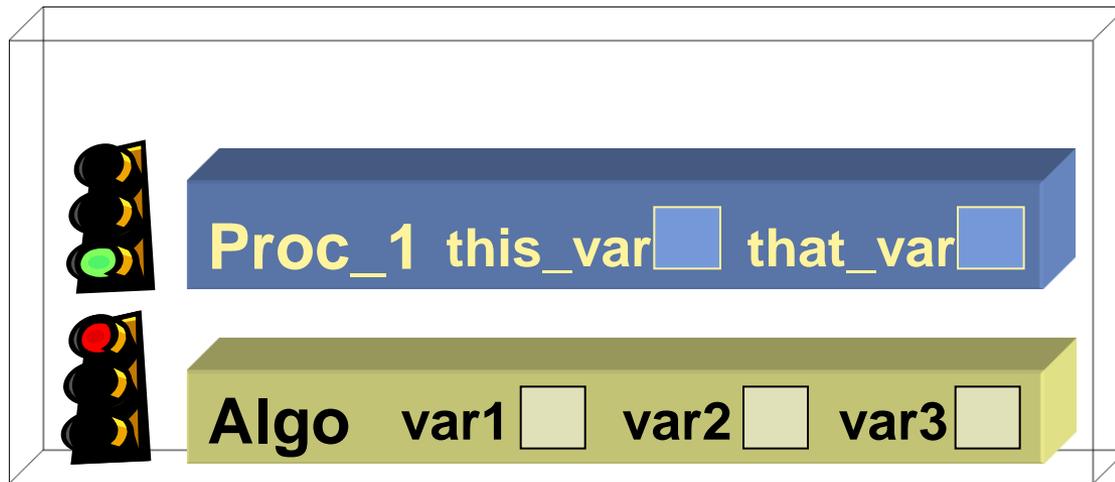
- Declaring variables in the standard way won't work here because we **don't know how many** variables to declare
- We need a way to **allocate** and **de-allocate** data **dynamically** (i.e., **on the fly**)



# THE STACK

- **Recall the activation stack**

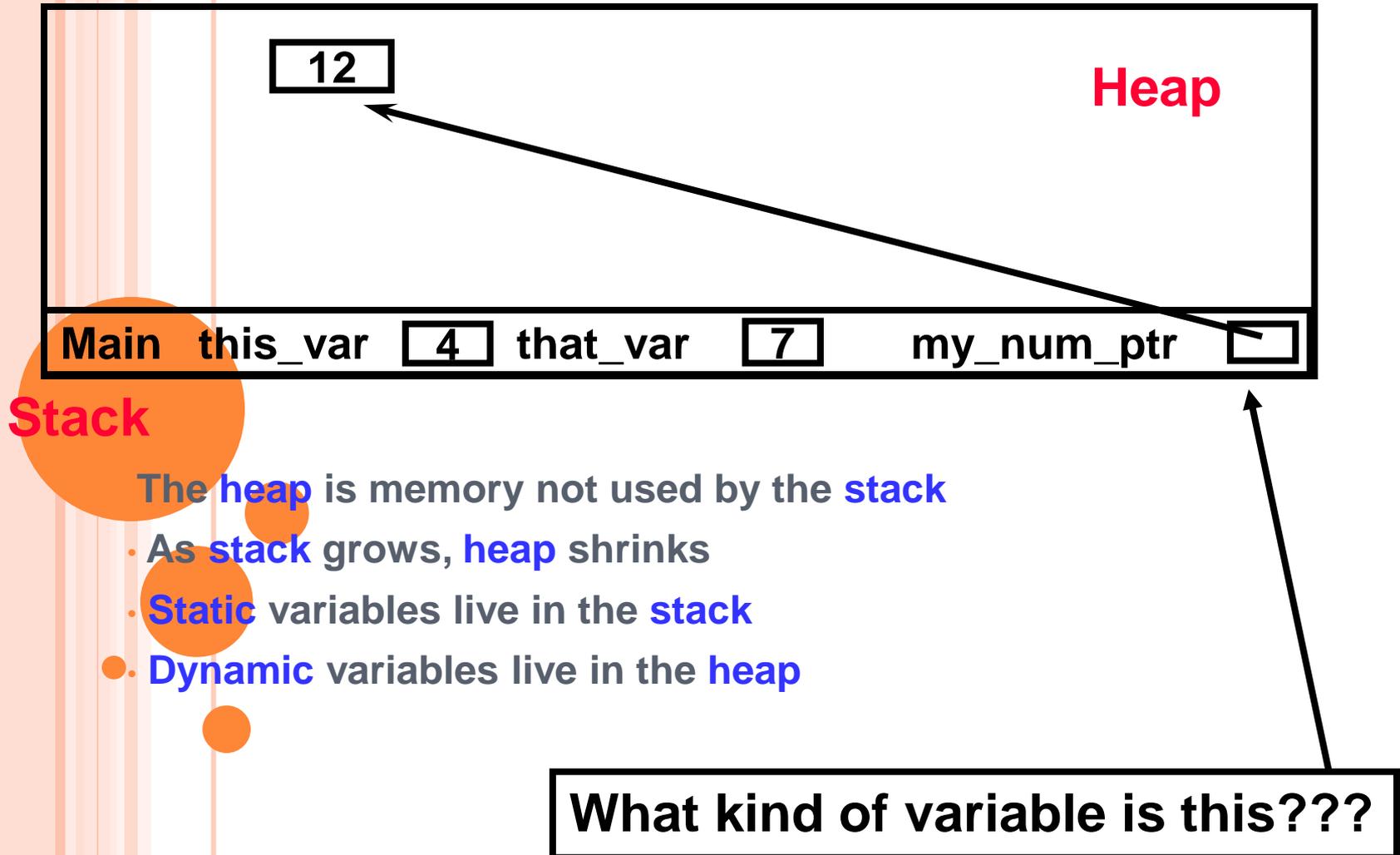
- **The stack can expand, but as for the data...**
- **Each frame contains static (fixed size) data**



**The number of variables needed come from the “isotype” statements.**



# THE STACK AND HEAP



**Stack**

The **heap** is memory not used by the **stack**

- As **stack** grows, **heap** shrinks
- **Static** variables live in the **stack**
- **Dynamic** variables live in the **heap**

What kind of variable is this???

# WHAT?

- We know (sort of) how to get a pointer variable

```
my_num_ptr isoftype Ptr toa Num
```

- But how do we get it to point at something?



## THE BUILT-IN FUNCTION NEW()

- Takes a type as a parameter
- Allocates memory in the heap for the type
- Returns a pointer to that memory

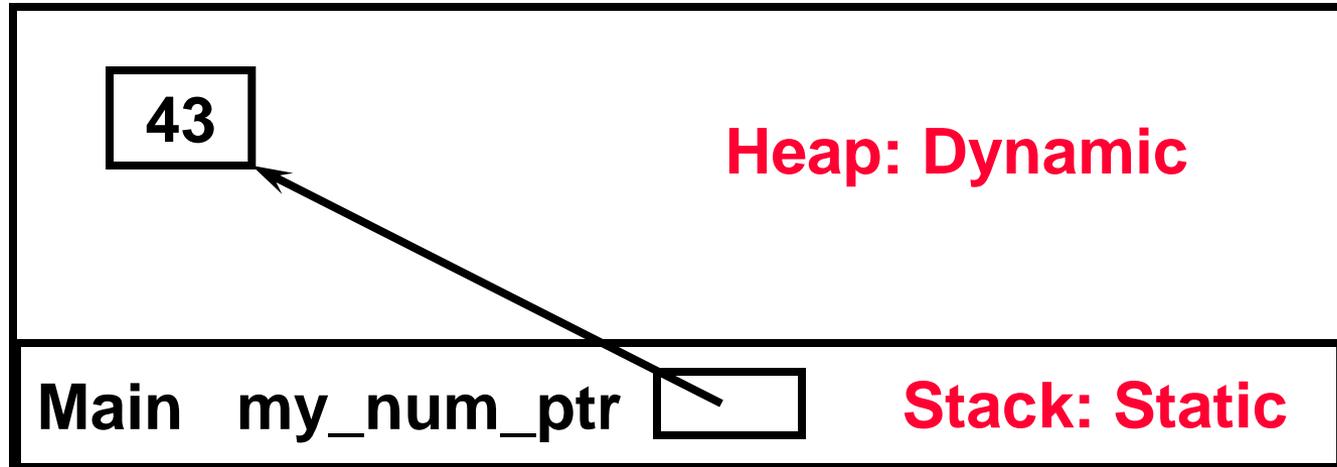
```
my_num_ptr <- new(Num)
```

```
dynamic_string <- new(String)
```

```
list_head <- new(Node)
```

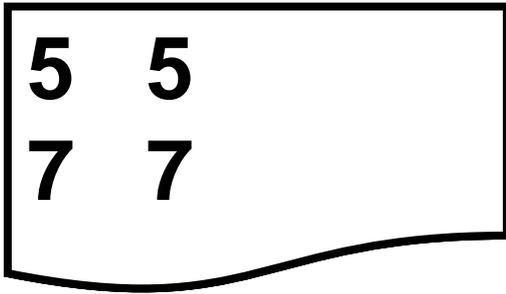


# ACCESSING DYNAMIC DATA VIA POINTERS

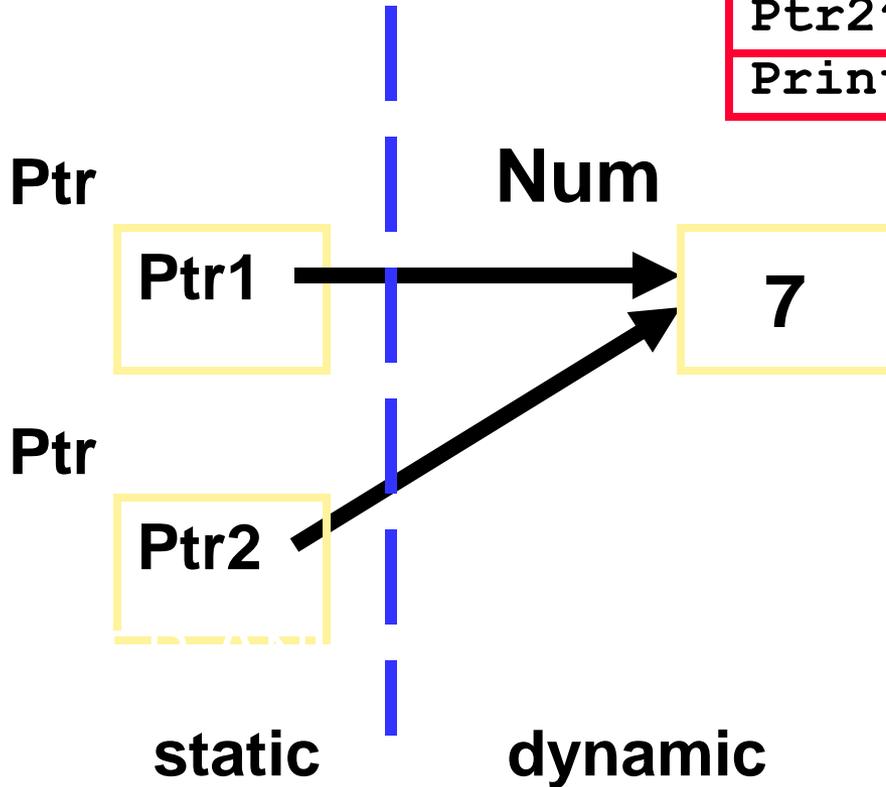


- When we “follow a pointer”, we say that we **dereference** that pointer
- The carat (^) means “dereference the pointer”
- `my_num_ptr^` means “**follow my\_num\_ptr to wherever it points**”
- `My_num_ptr^ <- 43` is valid



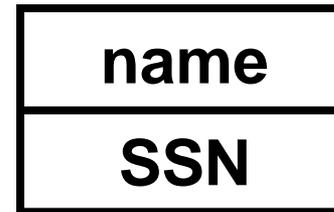


```
Ptr1 isoftype Ptr toa Num
Ptr2 isoftype Ptr toa Num
Ptr1 <- new(Num)
Ptr1^ <- 5
Ptr2 <- Ptr1
Print(Ptr1^, Ptr2^)
Ptr2^ <- 7
Print(Ptr1^, Ptr2^)
```



## A record to hold two items of data - a name and a SSN:

```
Student defines a record
  name isoftype String
  SSN isoftype num
endrecord
```



## And a pointer to a Student record:

```
current isoftype ptr to a Student
current <- new(Student)
```

