# CPSC 131
# Data Structures Concepts

Dr. Anand Panangadan
apanangadan@fullerton.edu

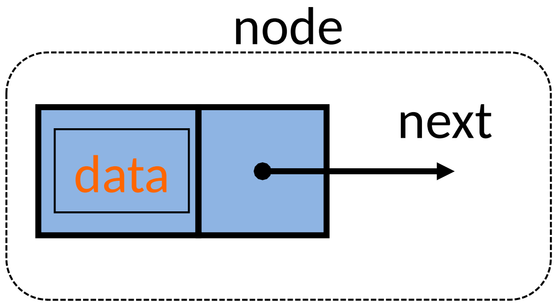
# Problem with FixedVector?

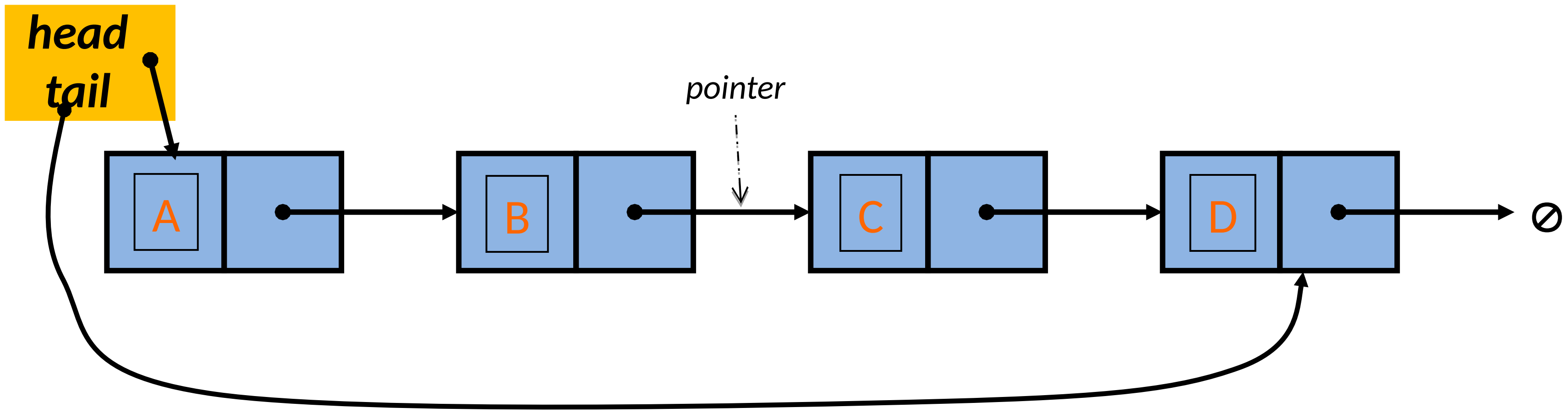
- Size has to be fixed when the data structure is created
  - FixedVector<Student> students(100000);
- Slow to insert values into the middle of a vector
  - Must move all values one at a time to create space
  - O(n) operation

## SINGLY LINKED LISTS

# Singly Linked Lists

- A singly linked list is a data structure consisting of a sequence of nodes
- Each node stores
  - data
  - pointer to the next node





# List ADT

| Operation | Description | Example starting with mylist: 99, 77 |
|---|---|---|
| List::Append(x) | Inserts x at end of list | mylist.Append(44), list: 99, 77, 44 |
| List::Prepend(x) | Inserts x at start of list | mylist.Prepend(44), list: 44, 99, 77 |
| List::InsertAfter(w, x) | Inserts x after w | mylist.InsertAfter(99, 44), list: 99, 44, 77 |
| List::Remove(x) | Removes x | mylist.Remove(77), list: 99 |
| List::IsEmpty(list) | Returns true if list has no items | mylist.IsEmpty() returns false |
| List::GetLength(list) | Returns the number of items in the list | mylist.GetLength() returns 2 |

## Two C++ implementations

- std::forward_list
  - Part of the C++ Standard Library
  - #include <forward_list>
- CPSC 131 implementation
  - class SinglyLinkedList
  - Based on zyBooks pseudocode

## CPSC 131 Singly Linked List Implementation

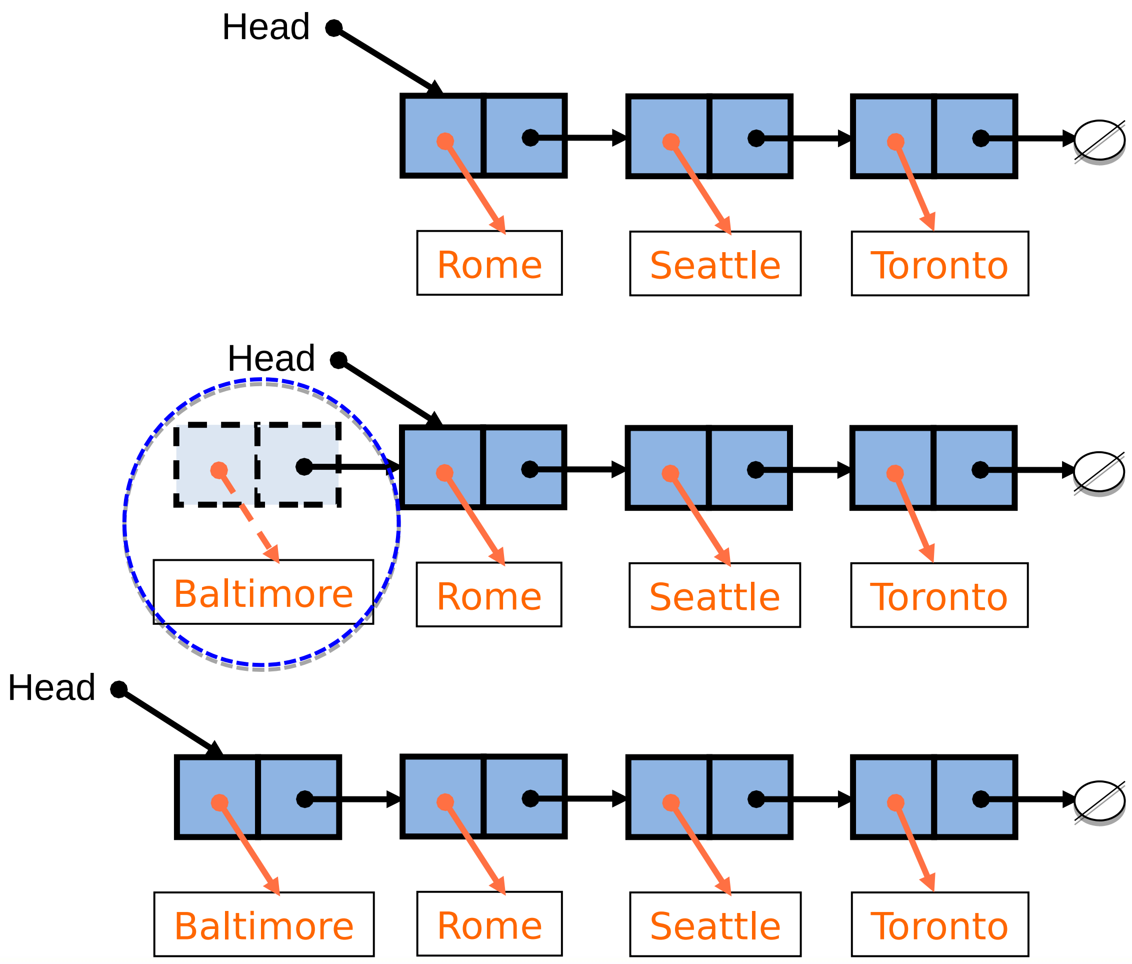
- https://github.com/CSUF-CPSC-131-Fall2019/Data-Structures-Code
- Based on zyBooks pseudocode
- SinglyLinkedList.hpp
- SinglyLinkedList_main.cpp

## Singly Linked List public methods

```
template <typename T>
class SinglyLinkedList {                              // a singly linked list
public:
        SinglyLinkedList();                      // empty list constructor
       ~SinglyLinkedList();                      // destructor
       bool empty() const;                       // is list empty?
       T& front();                          // return front element
       void prepend(const T& e);                  // add to front of list
       void append(const T& e);                   // add to back of list
       void pop_front();                         // remove front item
       int size() const;                         // list size
};
```

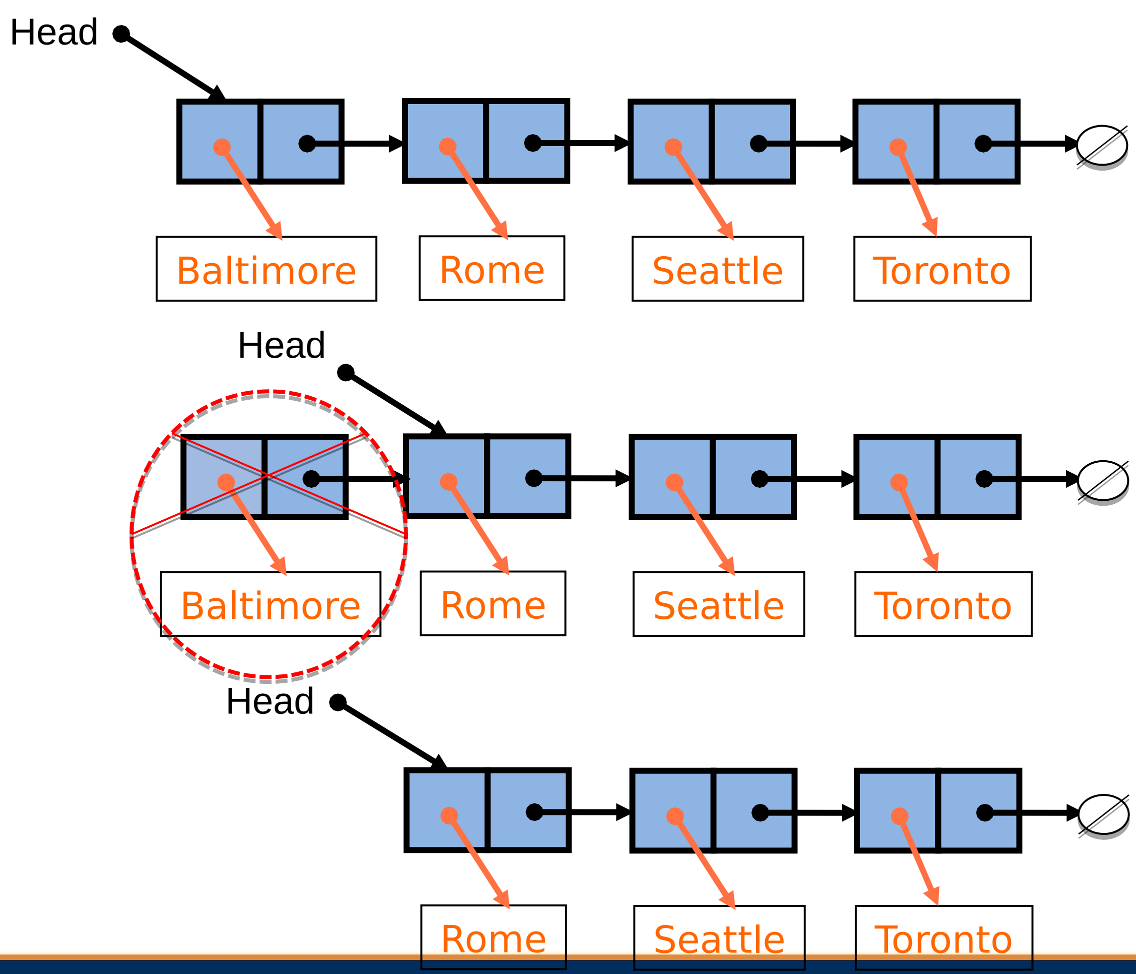
# Inserting at the Head (append)

1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node



# Deleting at the Head

1. Update head to point to next node in the list
2. Delete the former first node



## Draw data structure for this code

```
SinglyLinkedList<string> ds;
cout << ds.size();
ds.prepend(“road”);
ds. prepend(“winding”);
cout << ds.front();
ds. prepend(“and”);
ds.pop_front();
ds. prepend(“long”);
cout << ds.front();
```

## Nodes

To create a linked list using dynamic variables, we need a class which has two data members:

one to hold information

one to point to another object of the *same* class

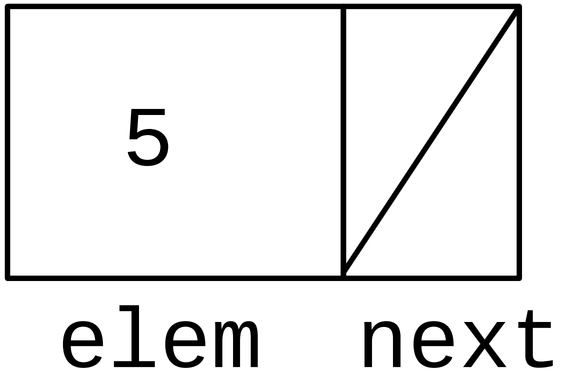
## Node

```
template <typename T>
class SNode
{
   public:
   T data;
   Snode<T> *next;
};
```

where ELT  will be the data type of whatever information you want stored.

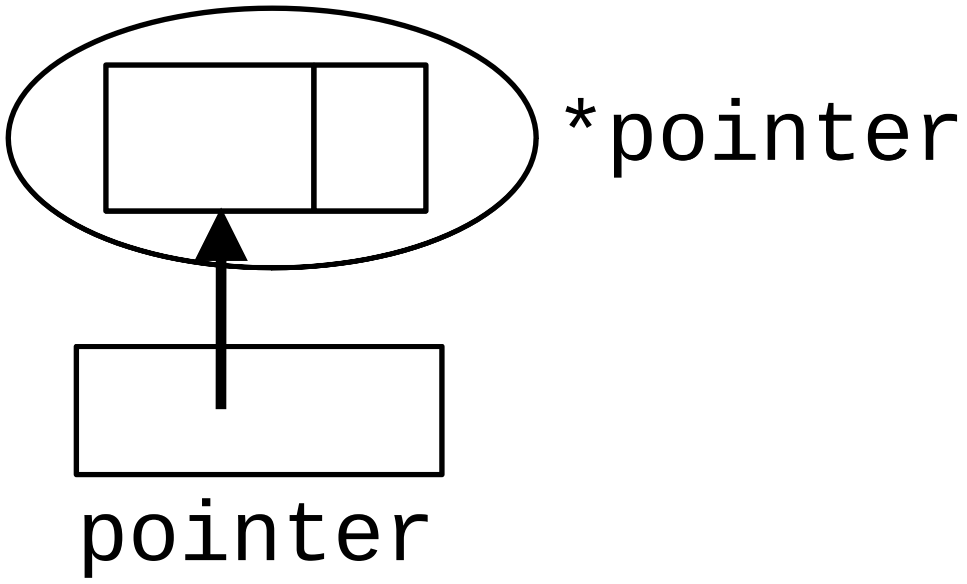
## Picture of a Node

```
Snode<int> Node;
Node.elem = 5;
Node.next = nullptr;
```



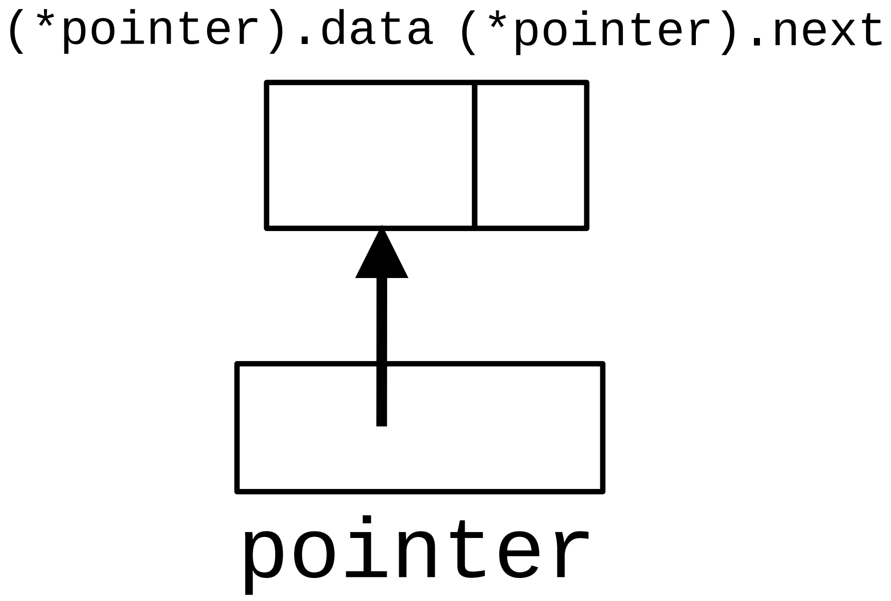
## Creating a node as a dynamic variable

```
Snode<int> *pointer;
pointer = new Snode<int>;
```



## Accessing the fields of the node

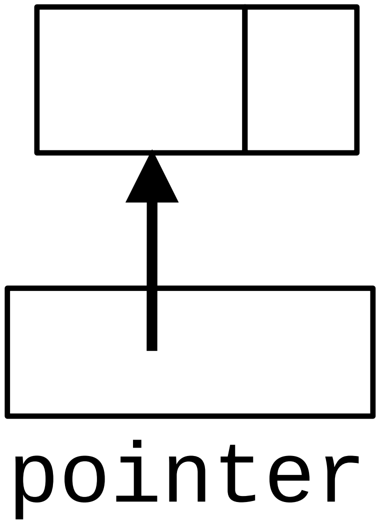
```
(*pointer).data
(*pointer).next
```



## Accessing the fields of the node

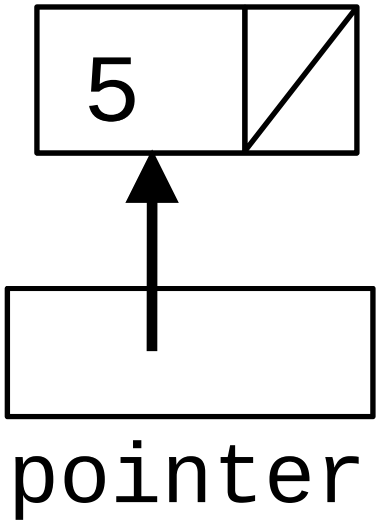
`(*pointer).data` can also be written as `pointer->data`

`(*pointer).next` can be `pointer->next`



## Accessing the fields of the node

```
pointer->data = 5;
pointer->next = NULL;
```

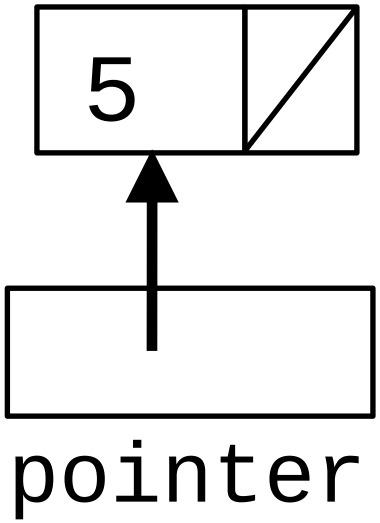


## Review: delete

What does

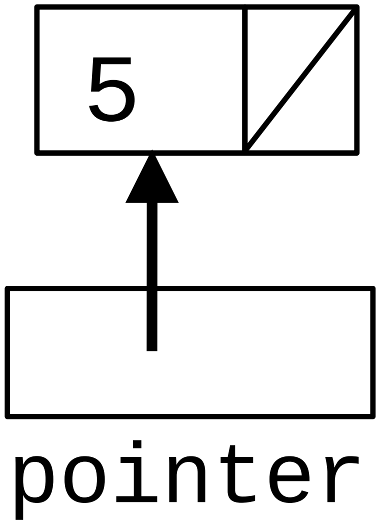
```
delete pointer;
```

do?



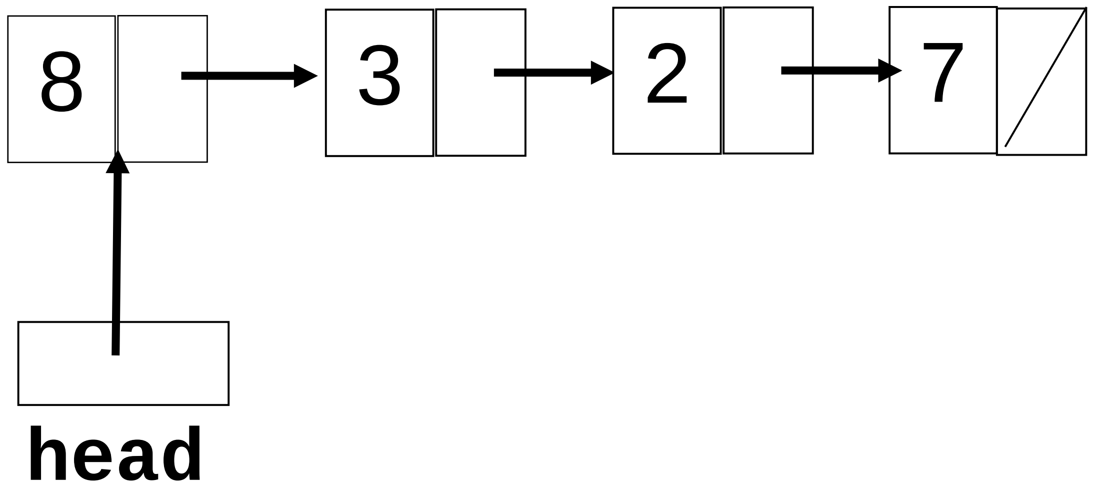
## Answer

It deletes the node, but leaves the pointer.



## What about a linked list?

Since the `next` field can point to another node, we can link nodes together like this:



# Creating a linked list

Start out with two NULL pointers to NodeType.

Code for this??

```
SNode<int> *pointer = NULL;
SNode<int> *head = NULL;
```

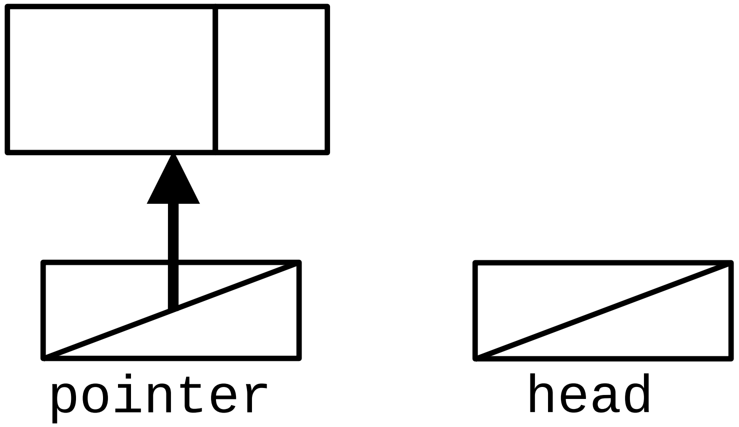
pointer
head

## Creating a linked list

Now create a new SinglyLinkedNode using `pointer`.

Code for this??

```
pointer = new Snode<int>;
```

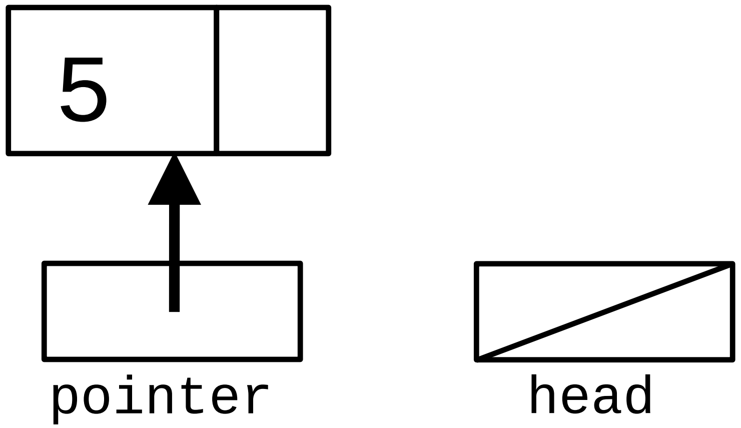


## Creating a linked list

Now insert a 5 in the info field.

Code for this??

```
pointer->data = 5;
```

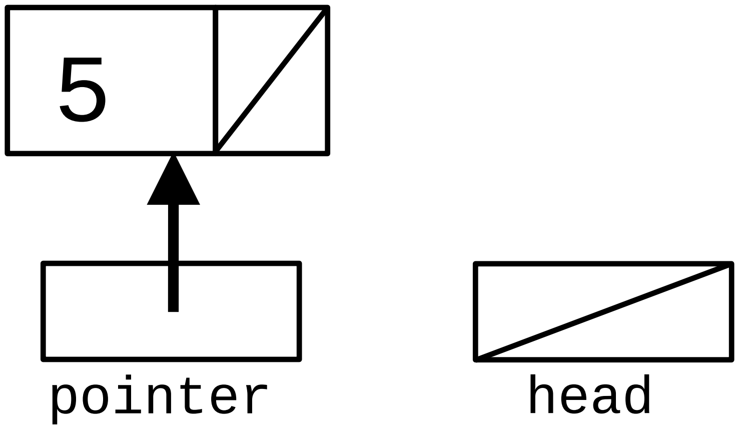


# Creating a linked list

Now what happens if we do

```
pointer->next = head;
```

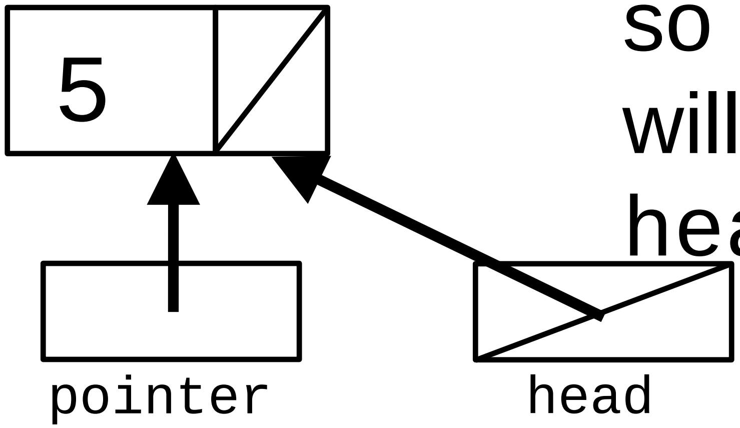
`head` contains `NULL`, which gets copied to `pointer->next`.



## Creating a linked list

Now we want head to point to the new node, i.e. head should contain the address of the new node.

What already has that address? **`pointer`**



so `head = pointer` will copy the address to `head`.

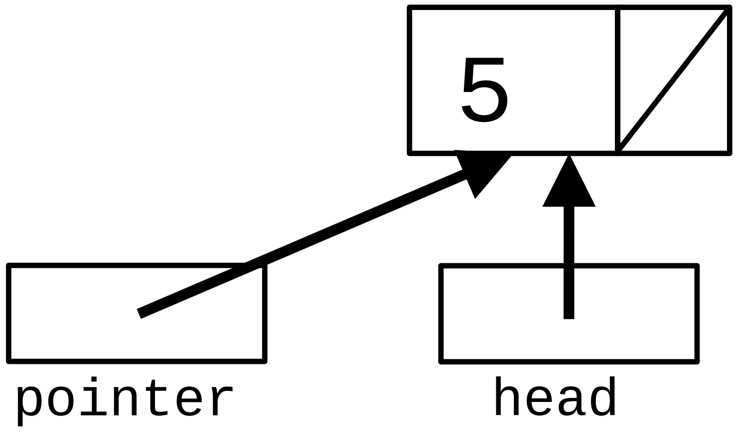
## Creating a linked list

Putting the code together:

```
pointer = new Snode<int>;
pointer->data = 5;
pointer->next = head;
head = pointer;
```

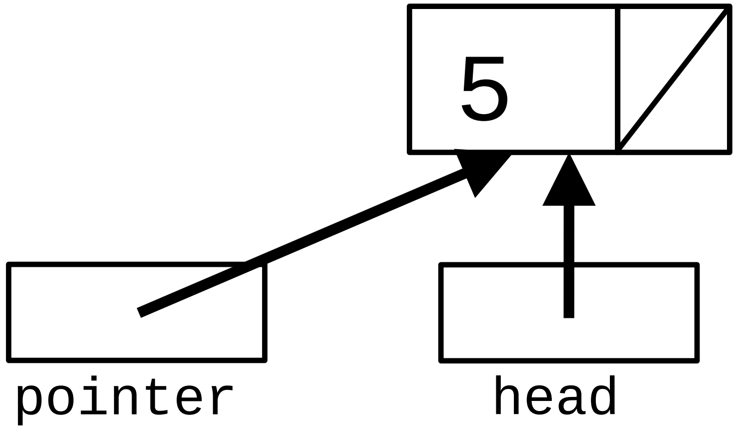
## Going from code to picture

Now try seeing what happens if we repeat the same code with `data` now set to 8, starting with this picture.



# Going from code to picture

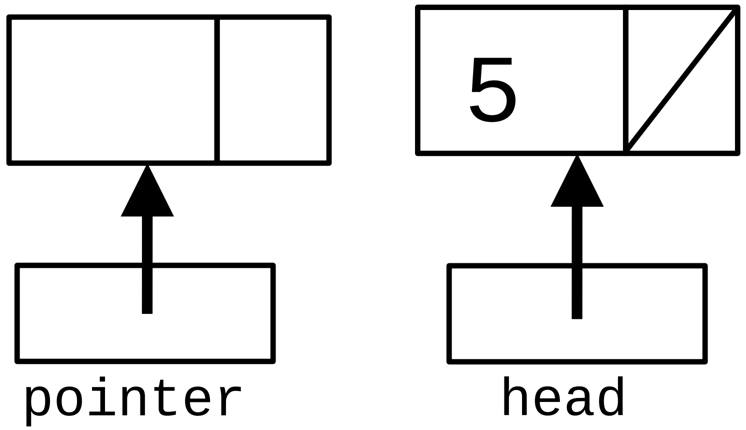
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



This does what?

## Going from code to picture

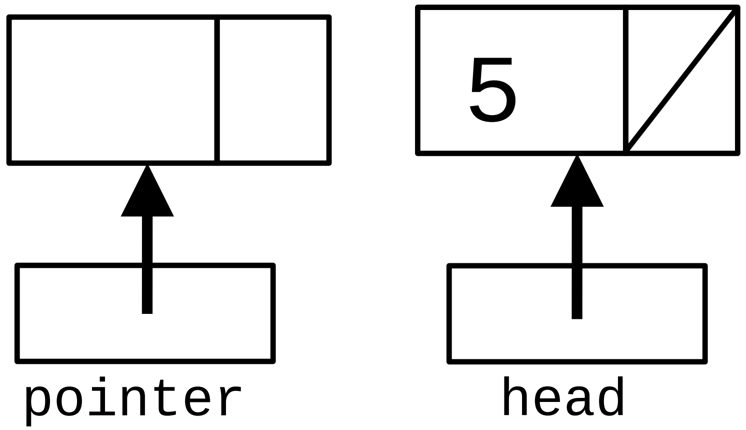
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



## Going from code to picture

```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```

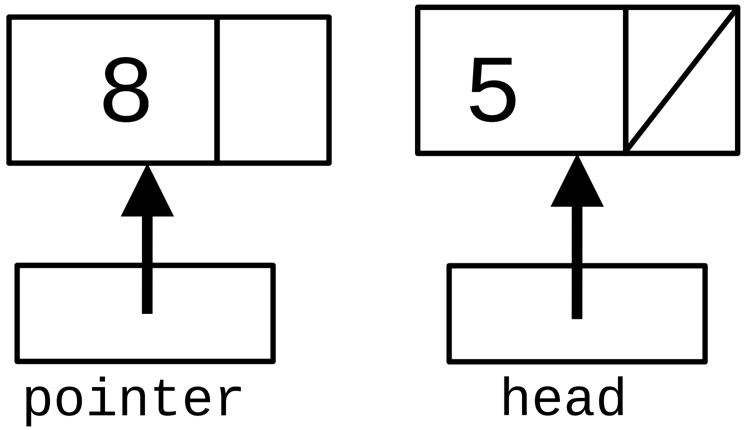
(The second line, `pointer->data = 8;`, is printed in bold.)



This does what?

# Going from code to picture

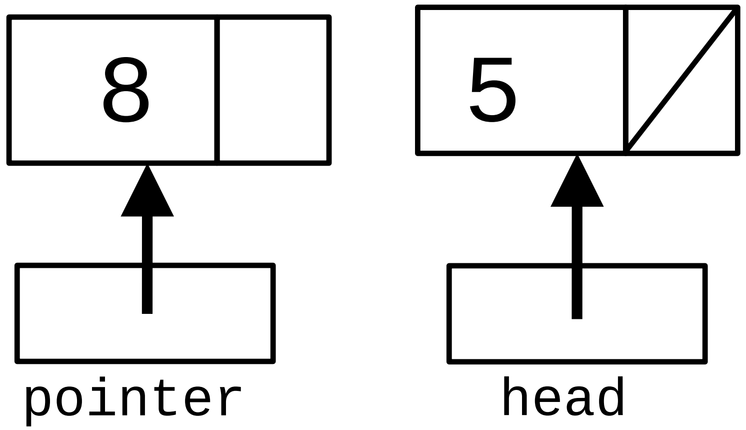
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



## Going from code to picture

```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```

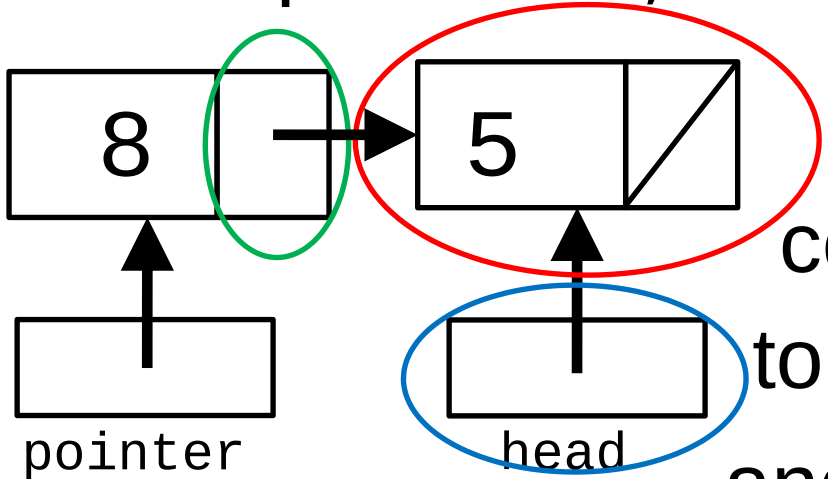
(The line `pointer->next = head;` is printed in bold.)



This does what?

# Going from code to picture

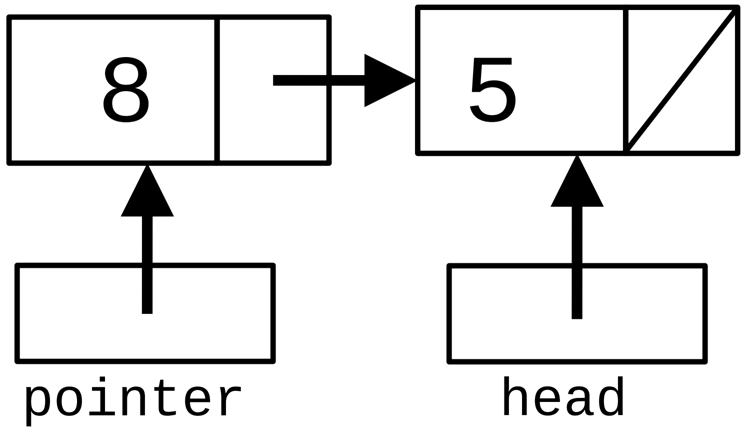
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



**head** contains the address of this so copy content of **head** to **pointer->next** and it will point to this.

## Going from code to picture

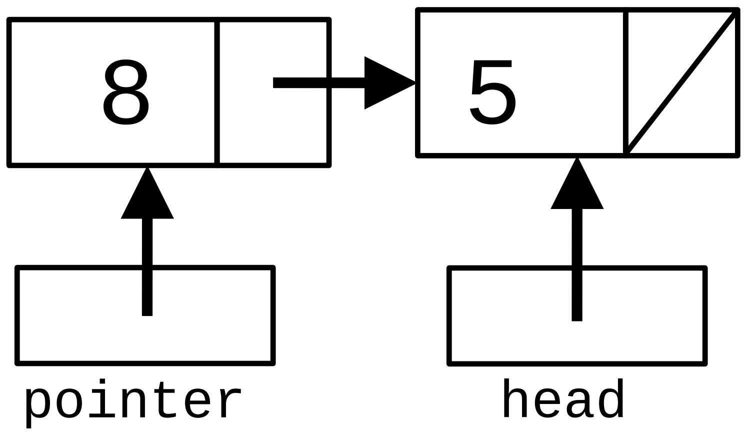
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
```
**`head = pointer;`**



This does what?

# Going from code to picture

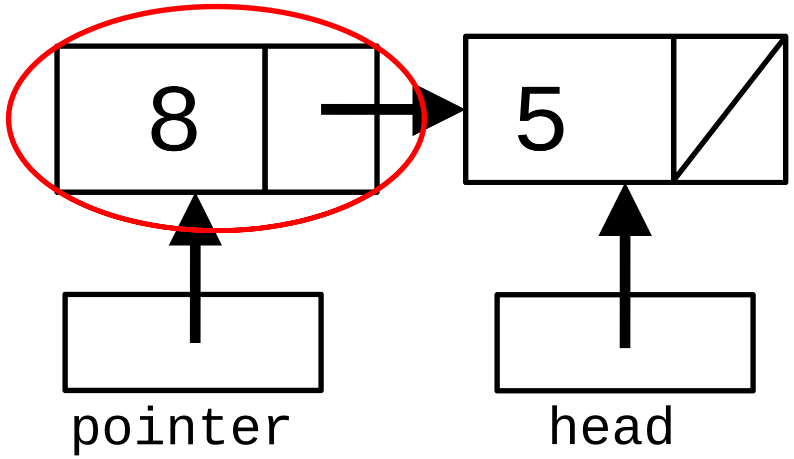
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
```
**`head = pointer;`**



**`pointer`** stores the address of (points to) what?

# Going from code to picture

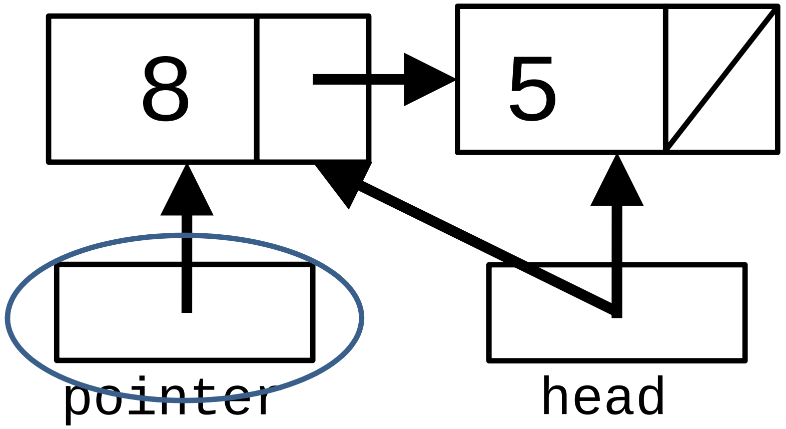
```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



**`pointer`** stores the address of (points to) what?

## Going from code to picture

```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
```
**`head = pointer;`**

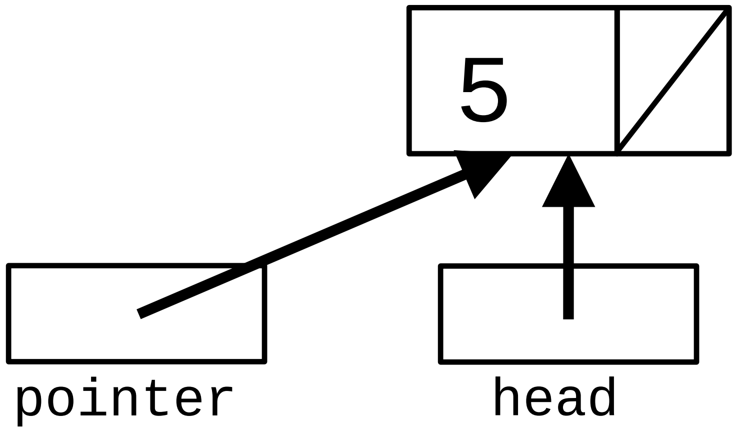


So if we copy the **`pointer`** to **`head,`** what will happen?

**head** will now point to the same node

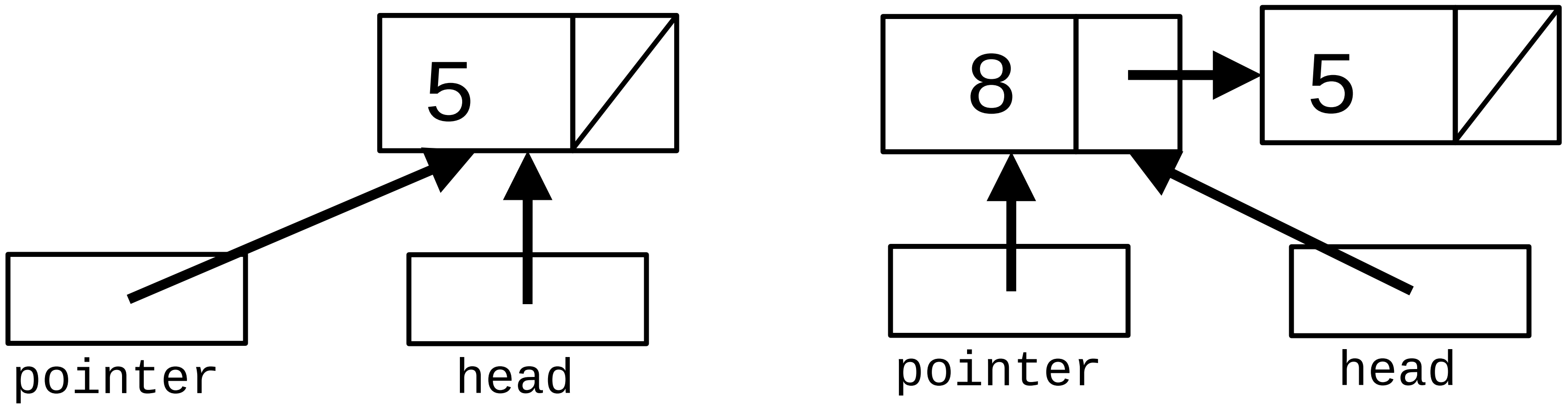
## So the following code will go from

```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```



## to this, i.e. insert a node at the front

```
pointer = new Snode<int>;
pointer->data = 8;
pointer->next = head;
head = pointer;
```


5
pointer
head
8
5
pointer
head