DYNAMIC DATASTRUCTURES

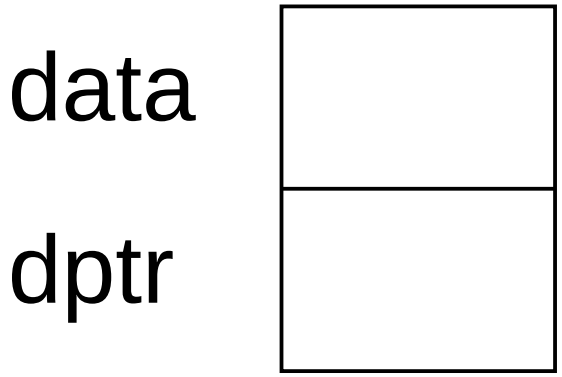
Today

- Today we will look at an extended example of using dynamic memory.
- We will build a small stack datastructure.
- All the code for this is in `exception2.cpp`.
- If you are happy reading and understanding the code, you don't need to bother with these notes.

- We start by defining a simple building block from which we can construct the stack.
- For a change we will use a `struct`.
- Remember that a `struct` is rather like a `class`, except that its members are `public` by default.
- We will exploit that here to avoid having to write access functions.
- However, this is *not* good programming practice.
- Our definition, then is:

```
struct dataElement {
public:
  int data;
  dataElement* dptr;
};
```

- We can think of this as defining a box with two parts.

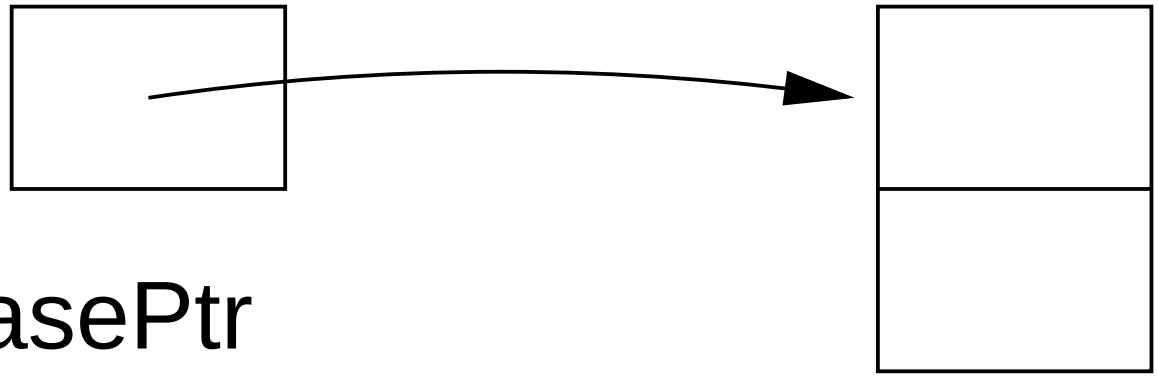


- In one part we can store data.
- In the other we have a pointer for linking boxes together.

- We will start by creating an object of this type.
- To make sure we don't lose it, we need a pointer

```
dataElement* basePtr;
basePtr = new dataElement;
```

- This sets up a situation that we can depict as:



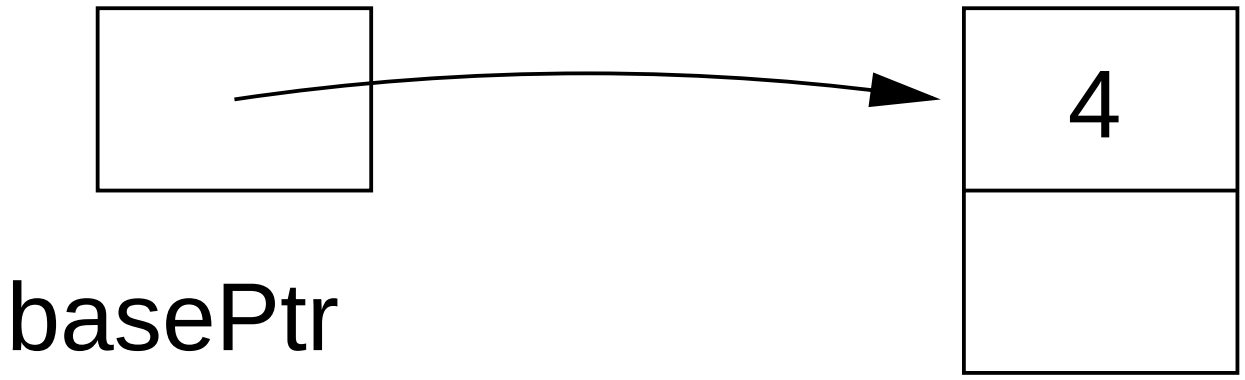
- We can add data into the datastructure, deferencing the pointer using either of

```
(*basePtr).data = 4;
```

or

```
basePtr->data   = 4;
```

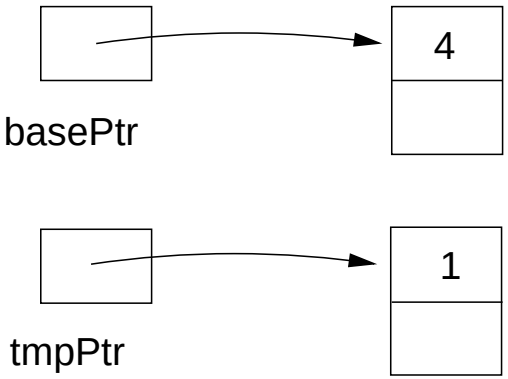
giving us:



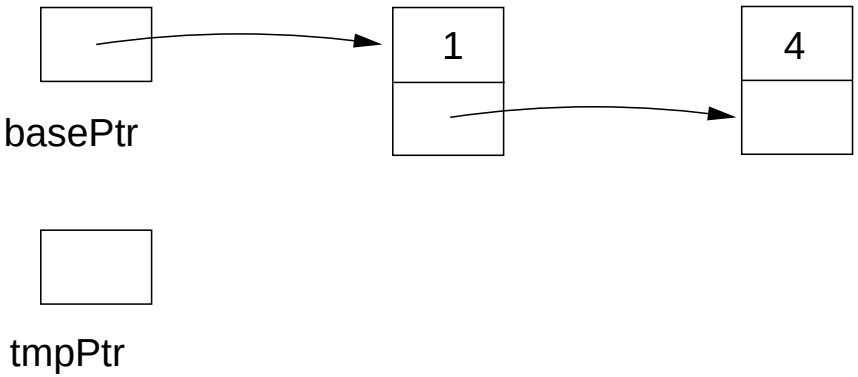
- Lets add another element to the stack.
- First we need a new element, and that requires another pointer:

```
dataElement* tmpPtr;
tmpPtr = new dataElement;
tmpPtr->data = 1;
```

which gives us:



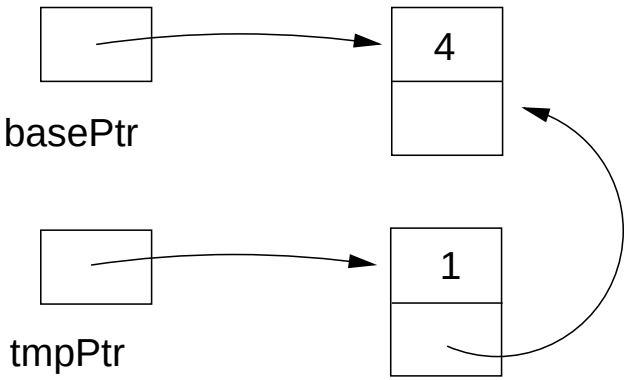
- We want to make these two elements into a stack.
- That is, a datastructure where the most recently created element is the one we have a link to, and that first element tells us where the next one is:



- We can achieve that using the following steps.

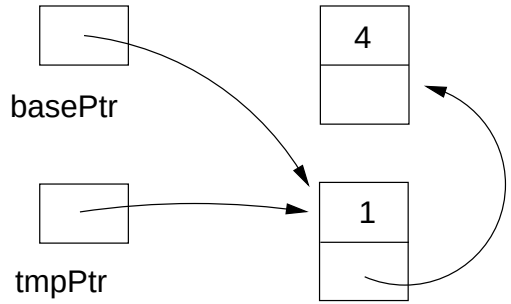
- First we make the new stack element point to the current top of the stack.

```
tmpPtr->dptr = basePtr;
```

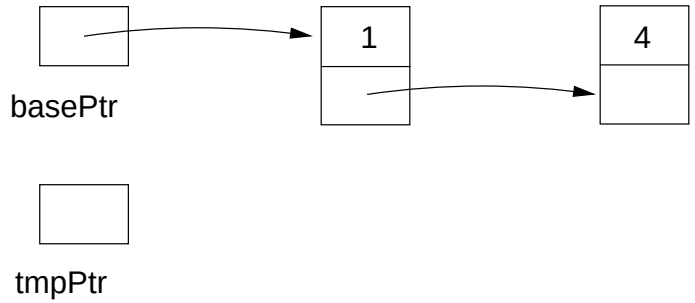


- Then we make `basePtr` point to this new element

```
basePtr = tmpPtr;
```

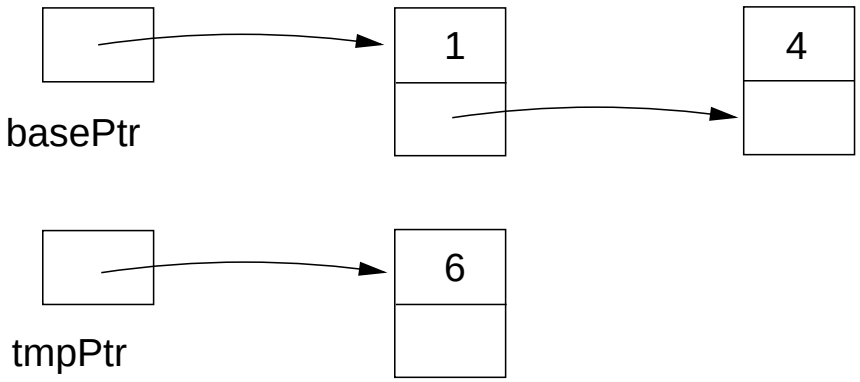


- Which is basically what we want, redrawn slightly (but not changing any significant values:



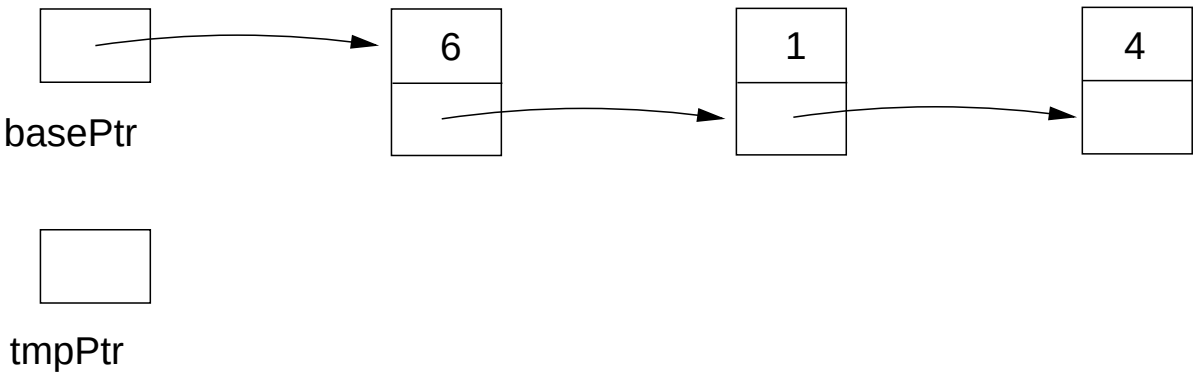
- We can add another new element.
- First we create the new element:

```
tmpPtr = new dataElement;
tmpPtr->data = 6;
```



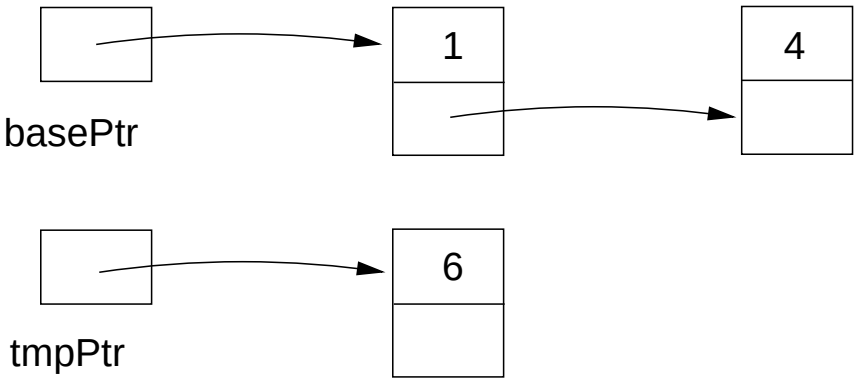
- Then by shuffling pointer values around, we attach it to the stack:

```
tmpPtr->dptr = basePtr;
basePtr = tmpPtr;
```



- To remove an element from the stack we just do:

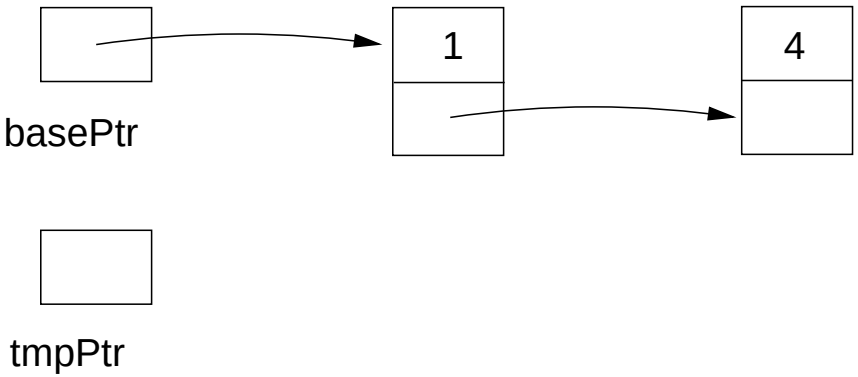
```
tmpPtr = basePtr;
basePtr = basePtr->dptr;
```



- Remembering to *delete* the memory we no longer need:

```
delete tmpPtr;
```

takes us back to:



Summary

- This lecture has been an extended illustration of the use of pointers to create dynamic datastructures.
- Now we will stop making such intensive use of pointers, though they are going to keep cropping up throughout the rest of the course.