Memory Layout of Data Members

- Every type has a size and an alignment requirement (see last lecture)
- To be compatible between different compilers and programming languages (mainly C), the memory layout of objects of class type is fixed
- Non-static data members appear in memory by the order of their declarations
- Size and alignment of each data-member is accounted for $\rightarrow$ leads to “gaps” in the object, called *padding bytes*
- Alignment of a class type is equal to the largest alignment of all non-static data members
- Size of a class type is at least the sum of all sizes of all non-static data members and at least 1
- static data members are stored separately

Size, Alignment and Padding

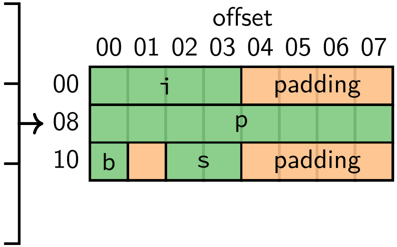
```
struct C {
      int i;
      int* p;
      char b;
      short s;
};
```

```
sizeof(i) == 4
alignof(i) == 4

sizeof(p) == 8
alignof(p) == 8

sizeof(b) == 1
alignof(b) == 1

sizeof(s) == 2
alignof(s) == 2
```



```
sizeof(C) == 24
alignof(C) == 8
```

Reordering the member variables in the order p, i, s, b would lead to `sizeof(C) == 16`!
In general: Order member variables by decreasing alignment to get the fewest padding bytes.