Seminar 01

Object-Oriented Programming, Structs, Separate Compilation

1. What is OOP?

- Looking at everything as an object, what is it, what does it have/do.
- **Inheritance** and **polymorphism**. How are objects connected to each other?
- **Encapsulation.** Creating and maintaining individual objects.
- **Abstraction** and reusability.

2. Structs.

- Why do we need them?

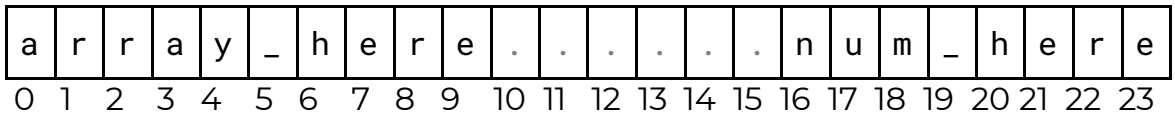
- How do we express them in code?

```
struct Point {
    double x;
    double y;
} p1, p2, p3, … ;
```

- Members' memory alignment.
  - Each member aligns to an address that's a multiple of its type size.

  Example:

```
struct Something {
    char str[10]; // 10 bytes
    double num;   // 8 bytes
} obj1, obj2;
sizeof(obj1) == sizeof(obj2) == sizeof(Something) == 24
```



str[10] *occupies* ***[0, 9]****, num occupies* ***[16, 23]****, spaces* ***[10, 15]*** *are unused.*

- Creating and accessing a variable of type struct.

```
Point p1;
p1.x = 3;
p1.y = 2;
```

```
Point p2 = { 3, 2 };
```

- Copying a simple struct

```
Point p1 = { 4, 2 };
Point p2 = p1; // p2 is a copy of p1
```

- Accessing data members of structs pointed by a pointer.

```
Point* ptrPoint = &p1;
(*ptrPoint).x = 42;    ⇔    ptrPoint->x = 42;
```

What's the type of a, b and c?

```
int* a, b, c;
```

- There's no default input/output of structs

```
cin >> p1;      // Compile error: The compiler doesn't
cout << p1;     // know how to read/print a Point
```

- Passing structs to functions:
  - by value (copies the struct)

```
void func(Point pt) { … }
```

  - by reference (doesn't copy the struct)

```
void func(Point& pt) { … }
```

When do we use const Point& ?