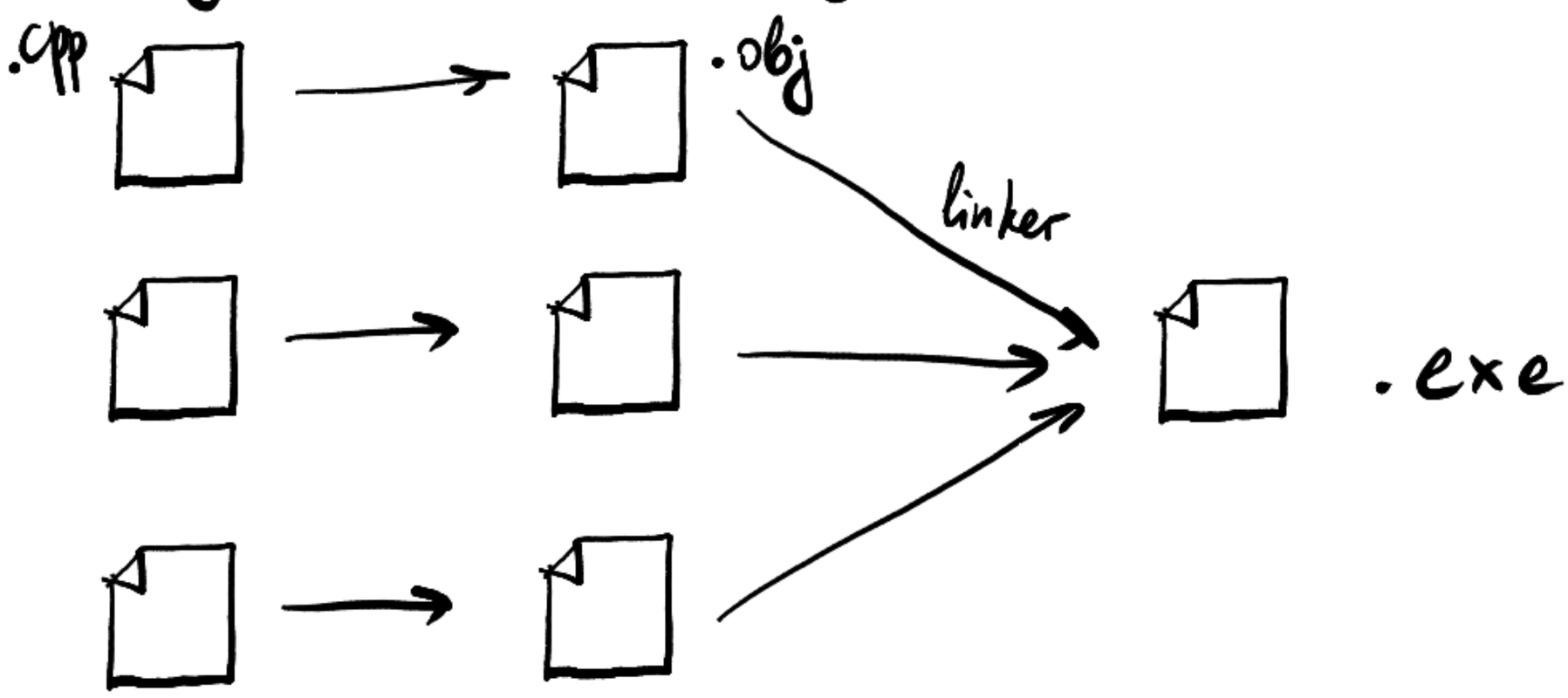
Тема 5.

Разделна компилация. Претоцесор. Копиращ конструктор и оператор =. Композиция и агрегация. Пример с клас Event.

Разделна компилация



→ Проектът се разделя на множество изходни (.cpp) файлове, като всеки се компилира независимо от другите (тести време, особено при малки промени)

→ Всички "обещания" - forward declaration - ги слагаме в друг .h файл, който include-ваме

→ #pragma once - директива, която казва на компилатора да обработи този код само веднъж

→ #include - замества реда със съдържание на даден файл

→ #define - замества един стринг с друг в целия код (макроси)

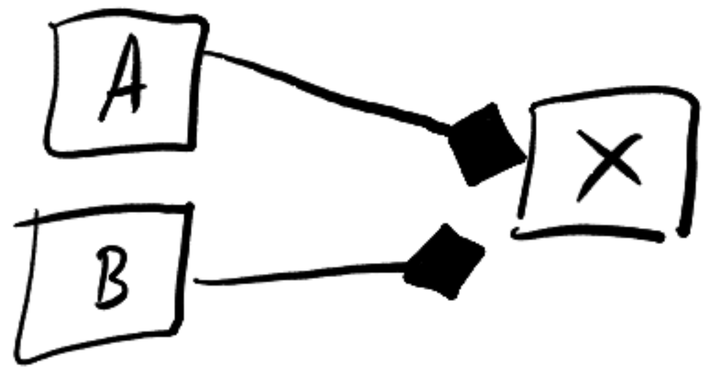
Стъпки на компилацията:

1) Претроцесор - обработка на string-ове
2) синтактичен анализ
3) семантичен анализ
4) междинни оптимизации
5) assembly code
6) машинен код (0 и 1)
7) линкване
8) CPU - dependant оптимизации

Композиция и агрегация - взаимоотношение м/у обекти

Композиция

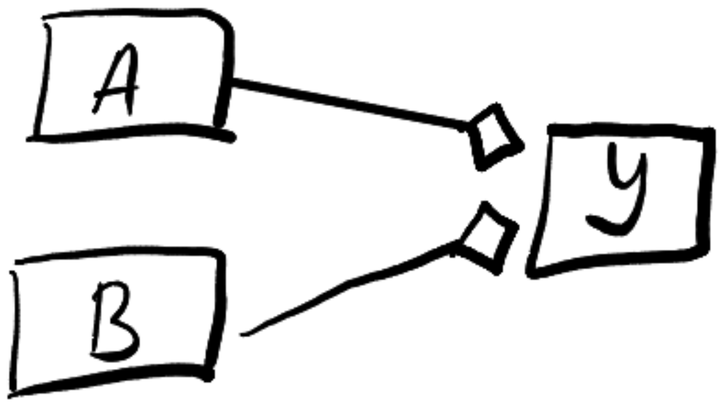
```
class X {
    A obj;
    B obj2;
}
```



Жизненият цикъл на A и B се контролира от X
(извиква конструктори и деструктори на A и B)

Агрегация

```
class Y {
    A*
    B&
}
```



A и B живеят извън рамките на Y.

Копиращ конструктор – конструктор, който приема обект от същия клас и текущият обект става негово копие

- ако не го разпишем, компилаторът създава такъв
- създава нов обект

```
class X
    A obj1
    B obj2

X(const X& obj):
    obj1(other.obj1),
    obj2(other.obj2)
```

- при композиция к.к. на обекта вика к.к. на член-данните си, ако са класове/структури, или използва memcpy, ако са примитивни

Оператор = (за присвояване) – приема обект от същия тип и текущият става негово копие

- текущият обект е съществувал преди това
- изчиства текущите данни и копира
- ако не го разпишем компилаторът създава такъв

```
X& operator = (const X& other)
```

- при композиция оп.= на обекта вика оп.= на член-данните си

```
A obj;
A obj2(obj); // к.к.
obj2 = obj; // оп. =
A obj3 = obj; // к.к.
```
(обектът е новосъздаден)

```
A create A(){
    return A();
}
A obj = create A();
```

} RVO (return value optimisation) спестява правенето на копия и вика само def. constr.

```
A create A(){
    A obj;
    ...
    return obj;
}
A obj = create A();
```

} NRVO (named return value optimisation) (обектът не е създаден в return)