Тема 4.

Указател this. Член-функции. Конструктор и деструктор. Извикване на конструктори и деструктори. Конвертиращи конструктори и деструктори при създаване на масиви (статични и динамични). Модификатори за достъп. Абстракция. Капсулация. Mutable.

Пример за клас Person с име (низ с дължина най-много 20 символа) и години [5, 90].

Член-функции → функции в тялото на клас/структура

→ работят директно с член-данните на обекта/инстанцията

→ извикват се от обект

→ преобразува се от компилатора до нормална функция, като подава и допълнителен параметър (конст. указател this към обекта, от който се извиква)

```
struct A {
    void g();          → void g(A* const this);
    void f() const;    → void f(const A* const this);
};
```

→ константните функции не променят член-данните

→ неконстантните функции не могат да се извикват в константни

Конструктор → член-функция, която се извиква при създаването на обекти

→ използва се за инициализация на променливите

→ заделя / отваря външни ресурси

→ има същото име като класа / структурата

→ не се указва тип на връщане експлицитно (връща const ref)

→ конструктор без параметри се нарича default-ен (ако не се разпише конструктор, компилаторът генерира автоматично default-ен, ако създадем друг конст., не се създава сам default-ен)

→ конструктор с 1 параметър се нарича конвертиращ (explicit указва, че не е такъв)

→ инициализиращ лист - задава начални стойности на данните, изпълнява се преди да влезе в тялото на конструктора

→ при A arr [10] се вика 10 пъти default-ен конструктор (задължителен за масиви)

→ при влагане:

```
struct A {
    B obj;
    C obj2;
    A(): B(), C() {}
};
```

Редът на викане е B(), C(), A()

```
X {
  A obj 1;
  B obj 2;
  C obj 3;
}
```

→ конструкторът на X е отговорен за създаването на A, B, C и трябва да им каже кои конструктори да им се извикат; ако не го направи се вика default-ния; ако нямат такъв, няма да се компилира

Деструктор

→ член ф-я, която е единствена
→ извиква се при изтриване / унищожаване на обекта
→ освобождава дин. памет и затваря външни ресурси
→ има същото име като класа / структурата, но с ~ отпред
→ няма тип на връщане
→ компилаторът генерира дестр. автоматично, ако не го разпишем експлицитно
→ използва се за унищожаване на външен ресурс
→ при влагане:

```
struct A {
    B obj;
    C obj2;
}
```

⇒ ~A(), ~C(), ~B()

→ Дефиниране конструктори и деструктори, понеже тези по подразбиране не правят винаги това, което бихме искали (при работа с дин. памет)

Абстракция - използваме нещо без да се интересуваме как работи
- скриване на ненужните детайли

Капсулация - ограничаване на достъпа
- постига се чрез модификатори за достъп
  → private - достъп само в класа
  → protected - достъп в класа и наследниците
  → public - достъп навсякъде

→ struct - по default е public
→ class - по default е private

- get-функции - връщат копие / конст. реф./указ. към член данните
- set-функции - позволяват промяна, но под контрол

- mutable - член-данни, които могат да се модифицират дори в const функции
  - не влияят на видимото състояние на обекта