7. Предефиниране на оператори. Приятелски класове и ф-ции

Оператори

- ф-ции със специален синтаксис
- operatorX (X е оператор)

Пр:
```
class A {int n;}
A ob;1, ob;2;
ob;1 + ob;2;
```

Характеризират се с:

- асоциативност - при изрази с оператори от еднакъв тип да се дефинира последователността на операциите
  - а) ляво (((a $ b) $ c) $ d) → (+, -, *, /)
  - б) дясно (a $ (b $ (c $ d))) → (=)
- приоритет - при израз с повече от един опер. определя последователността на изпълнение

  Пр: * > / > + > -
- позиция (на кои спрямо аргументите)
  - а) префикси ++a, --a, !a, ~a
  - б) инфикси a+b, a-c
  - в) суфикси a++, a--

⚠ не можем да предефинираме несъществуващи оператори (да създаваме нови)

⚠ не можем да променяме приоритета, асоциативността и позицията на оператор

⚠ не можем да предефинира следните оп:

(.), (::), (?:), (sizeof), (typeid), (alignof)

Тип на операторите

- унарни (1 аргумент) ++a, --a
- бинарни (2 аргумента) a+b, a-b
- тернарен - той е само 1 - ?:

- при бинарните оп, лявата операнда се приема като първи парам, а дясната като втори.

⚠ изключение прави op() (function call), който може да приема произволен брой

⚠ при предефиниране на && и || губим short-circuit eval

⚠ при наличието на наследяване трябва оп>, както и другите (>, <=, ==, !=, >=) - те могат да използват оп<

⚠ затова при проверка за равенство имаме оп== и оп!=

⚠ при предефиниране трябва да запазим return type

Начини за предефиниране

- външна ф-ция
  - няма достъп до private член-данните
  - ние го използваме като когато нямаме да променяме левия операнд

Пр:

```
Complex operator+(const Complex& c, const Complex& r)
    Complex n(c);
    n += r;
    return n; → NRVO
}
```

- член ф-ция
  - имаме достъп до private член-данните
  - левия аргумент е this

Пр:

```
Complex& operator+=(const Complex& r) {
    _real = r._real;
    _im = r._im;
    return *this
}
```

⚠ +=, -=, /=, *= - не дефинират като член-функции

+, -, /, * - не дефинират като външни функции

⚠ има функции, които могат да са само вътрешни: (=), ([]), (()), (->) - трябва да връщат *this

Предефиниране на оп за инкрементация (++), (--)

++a - връща променената стойност (&a) → lvalue

a++ - връща старата стойност → rvalue

Пр:
```
class X{
    int n;

    X& operator++(){
        ++n;
        return *this
    }

    X operator++(int dummy){
        X temp = *this;
        ++n;
        return temp;
    }
};
```

Предефиниране на I/O оп.

- когато имаме I/O операции с класа трябва да предефиниране оп за ~~всякакъв~~ поток

Пр:
```
A ob;
cin >> ob;
cout << ob;
```

- трябва да връщат istream& / ostream& за да може да има chaining

Пр:
```
ostream& operator<<(ostream& os, const A& ob){
    return os;
}
istream& operator>>(istream& is, A& ob){
    return is;
}
```

⚠ тези ф-ции трябва да имат като аргументи потоци, защото да са вътрешни:

Пр istream & operator>> (istream&) {..}

A obj;
obj >> cin; // не отговаря на стандарта

но, те трябва и да могат да променят полетата на а

=> ф-ция, която трябва е вътрешна и да променя полетата => friend

Приятелски класове и ф-ции

- класове / ф-ции вътрешни за класа, които имат достъп до private член-данните на класа
- friend
- не са транзитивни, не се наследяват
- няма значение дали е в private / public / protected частта

Пр: struct A {
    friend ~~ostream~~ istream& operator>> (istream&, A&) {}
}

class SharedPtr {
    struct Counter {..}
    friend class WeakPtr;
}