Предефиниране на оператори

титуляр на курса: д-р Тодор Цонков (ttsonkov@gmail.com)

От какво ще се състои настоящата лекция?

Предефиниране на оператори. Приятелски класове и функции. Defaulted comparison operators и <=> (C++20). Ключова дума auto. Пример за реализация на клас Student и School.

Какво са операторите в езика C++?

Операторите са:
- унарни (с един операнд)
- бинарни (с два операнда).
- Всеки оператор се характеризира с:
- Позиция на оператора спрямо операнда (операндите) му;
- Приоритет;
- Асоциативност
- В езика C++ не е възможно да бъдат създавани нови оператори, но са дадени средства за предефиниране на съществуващи оператори.

Какво са операторите в езика C++?

Позицията на оператора спрямо операнда (операндите) му го определя като:

- префиксен (операторът е пред единствения си операнд),
- инфиксен (операторът е между двата си операнда)
- постфиксен (операторът е след единствения си операнд).

Пример. Операторът * е инфиксен (a * b), операторът + е както инфиксен, така и префиксен, а операторът ++ е както постфиксен (a++), така и префиксен (++a).

- Приоритетът определя реда на изпълнение на операторите в израз. Оператор с по-висок приоритет се изпълнява преди оператор с по нисък приоритет.

Закачка: Какво ще изпечати следния код: a+++++b - еквивалентно на a++ ++ +b , a++ + ++b?

Какво е предефиниране на оператори?

Какво е предефиниране на оператори?

Operator Overloading позволява стандартните C++ оператори (+, ==, <<, < и др.) да работят с потребителски типове.

Кодът става по-четим
Обектите се държат като вградени типове

Пример

```
Student a, b;
if (a == b) { ... }

if (a != b) {... }

if (a < b) -> какво означава? в случая със Student не става ясно

Student operator+(const Student& a, const Student& b);
Какво означава това?
```

Кога да предефинираме оператори?

1)Когато операторът има естествен смисъл
2)Когато семантиката е ясна
3)Не за „изненадващо“ поведение

Добър пример: ==, <, <<
Лош пример: operator&& с нетипично значение

Не трябва да се променя смисъла на оператора
И трябва да се избира non-member функции, когато е възможно.

```cpp
#include <string>
#include <vector>

class Student {
    std::string name_;
    int facultyNumber_;
    std::vector<double> grades_;

public:
    Student(std::string name, int fn, std::vector<double>
grades)
        : name_(std::move(name)), facultyNumber_(fn),
grades_(std::move(grades)) {}
};


bool operator==(const Student& a, const Student& b) {
    return a.facultyNumber_ == b.facultyNumber_;
}
```

Кои оператори не може да бъдат предефинирани?

. - оператор за избор на член на клас ●
* - оператор за избор на член на клас чрез указател
● :: - оператор за присъединяване - примерно към namespaces
● ?: - троен условен израз
● sizeof

ключова дума friend

```
class Student {
    std::string name_;
    int facultyNumber_;
    std::vector<double> grades_;

public:
    Student(std::string, int, std::vector<double>);

    friend bool operator==(const Student&, const
Student&);
};

bool operator==(const Student& a, const Student& b) {
    return a.facultyNumber_ == b.facultyNumber_;
}

За симетрични оператори (==, <<)
Когато не искаме getter-и
Не за „всичко“ – нарушава енкапсулацията
```

предефиниране на оператори

```cpp
#include <ostream>

class Student {
    friend std::ostream& operator<<(std::ostream&, const
Student&);
};

std::ostream& operator<<(std::ostream& os, const Student&
s) {
    return os << s.name_ << " (" << s.facultyNumber_ <<
")";
}

bool operator<(const Student& a, const Student& b) {
    return a.facultyNumber_ < b.facultyNumber_;
}

std::sort(students.begin(), students.end());
```

Още примери за предефиниране

Примери

```
class Student {
public:
   bool operator==(const Student& other)
const;
};

bool operator==(const Student&, const
Student&); -> по-добре
```

Reflection C++ 26

```
template <typename T>
std::ostream&
print(std::ostream& os, const
T& obj) {
   constexpr auto meta =
reflexpr(T);

   for constexpr (auto m :
meta.members()) {
      os << m.name() << "=" <<
obj.*(m.pointer()) << " ";
   }
   return os;
}
```

Кога friend е оправдан?

✔ operator<<, operator>>
✔ operator==, <=> (ако не е defaulted)
✔ Тясно свързани помощни класове

Defaulted operator in C++ 20

Какъв проблем решава?

Трябва да предефинираме всички оператори: ==, !=, <, >, <=, >=

но имаме много boilerplate код.

Дефиниция

```
class Student {
    std::string name_;
    int facultyNumber_;
    std::vector<double>
grades_;

public:
    auto operator<=>(const
Student&) const = default;
};
```

Обединява всички оператори в една функция

Примери за функции

Започва с auto, защото резултатът е един от следните типове:

std::strong_ordering
std::weak_ordering
std::partial_ordering

Auto functions in C++

Какъв проблем решава?

```
auto add(int a, int b) {
    return a + b;
} //C++ 14 and above

auto multiply(int a, double b) ->
double {
    return a * b;
}
```

Дефиниция

В C++ думата auto е ключова дума за автоматично извеждане на типове (type deduction).

Тя се използва много често в модерния C++ (C++11 → C++23) и има няколко различни роли, като типът се извежда от компилатора.

Задължително е да има инициализация, иначе типът няма да бъде изведен.

Примери

auto за променливи (C++11)

```
auto x = 5;       // int
auto y = 3.14;    // double

decltype(auto) get(int& x) {
    return x;
}
```

- запазва точния тип, константност и т.н.

Ключова дума auto в езика C++

Какъв проблем решава?

auto извежда типа от инициализатора по същите правила като template type deduction.

```
int a = 5;
int& r = a;

auto x = r;     // int     ❌ reference се губи
auto& y = r;    // int&    ✅
```

Кога да я ползваме?

✅ Дълги и сложни типове:
auto it = myMap.begin();

✅ Template / generic код
✅ Iterator-и
✅ Lambda резултати
✅ Избягване на грешки при промяна на тип

Примери

```
std::vector<int> v = {1,2,3};

for (auto x : v)     // ❌ копие
for (auto& x : v)    // ✅ reference
for (const auto& x : v) // ✅ най-често правилното
```

Правила за auto в езика C++

```
1) Премахва се const и & по подразбиране
const int a = 10;
auto x = a;     // int (НЕ е const int)

Решение:
int a = 5;
int& f() { return a; }

auto x = f();          // int   (копие)
decltype(auto) y = f(); // int&
(референция)

2) Масиви и функции стават указател
int arr[10];
auto x = arr;   // int*

3)auto v = {1, 2, 3};     //
std::initializer_list<int>
```

Примери

```
4)auto [x, y] = std::pair{1, 2}; //C++17
```

Грешки

```
1)  auto x = 3.0 / 2;  // double
auto y = 3 / 2;    // int -> 1

2)  auto y = x;        // double
    auto& r = x;       // double&
    // но:
    auto z = {1,2,3};  //
    std::initializer_list<int>, не масив

3)  const int a = 10;
    auto x = a;      // int, НЕ е const int
    x = 5;           // позволено
```

имплементация на клас Student

```cpp
#include <ostream>

class Student {
    std::string name_;
    int facultyNumber_;
    std::vector<double> grades_;

public:
    auto operator<=>(const Student&) const = default;
};

//или да го предефинираме само по факултетен номер
auto operator<=>(const Student& other) const {
    return facultyNumber_ <=> other.facultyNumber_;
}
```

Предимства: По-малко код
✓ По-малко грешки
✓ Съвместимост със STL
✓ Ясна семантика

клас School

```cpp
#include <vector>

class School {
    std::string name_;
    std::vector<Student> students_;
public:
    explicit School(std::string name)
        : name_(std::move(name)) {}

    void addStudent(const Student& s) {
        students_.push_back(s);
    }

friend std::ostream& operator<<(std::ostream&, const
School&);

};

std::ostream& operator<<(std::ostream& os, const School&
sch) {
    os << "School: " << sch.name_ << "\n";
    for (const auto& s : sch.students_)
        os << "  " << s << "\n";
    return os;
}
```

Въпроси?

Благодаря за вниманието!