Обектно-ориентирано програмиране

Класове и обекти в C++

Иван Георгиев, Христо Иванов, Христо Стефанов

Технологично училище "Електронни системи",
Технически университет, София

18 юни 2019 г.

Съдържание

Съдържание

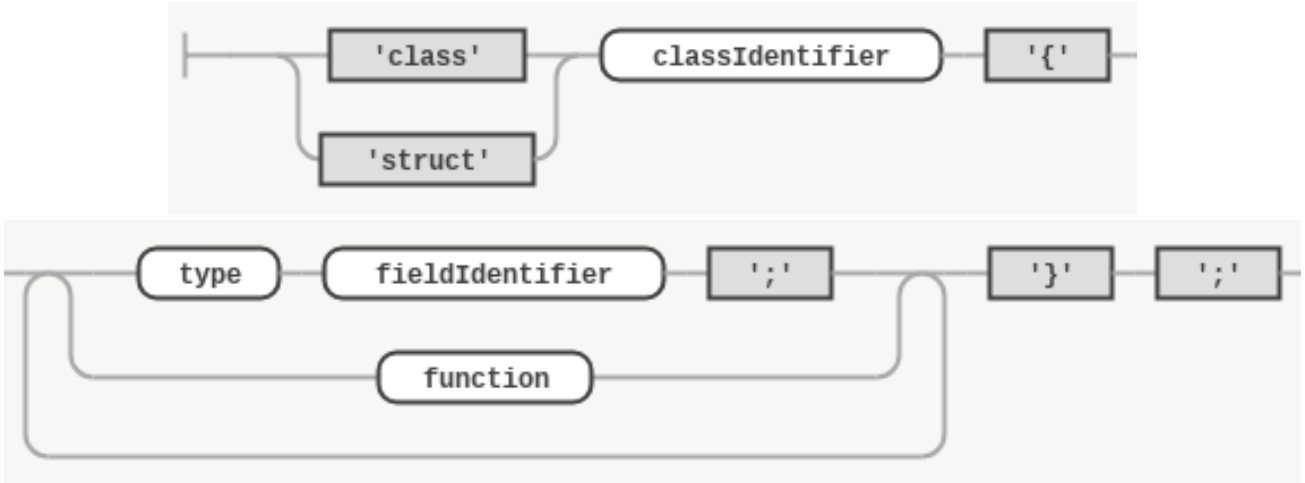
C++

- C++ е език, който надгражда C - всяка валидна конструкция на C е валидна конструкция и в C++[1]
- В C++ е добавена вградена поддръжка за обектно-ориентирано програмиране - т.е. езикът позволява по-лесно да се прилагат принципите на ООП

[1] Съществуват някои изключения, които в огромна част от случаите са пренебрежими

Класове и обекти в C++

- C++ предлага удобен синтаксис за деклариране на класове
- Класовете в C++ са типове, които могат да бъдат декларирани от програмиста, подобно на структурите в C
- Подобно на структурите, класовете в C++ имат полета, наречени *член-променливи*
- Освен член-променливи в класа могат да бъдат дефинирани и функции, наречени *член-функции* или *методи*. Методите не могат да бъдат използвани като обикновени функции



Фигура 1: Синтактична диаграма на декларация на клас

Класове и обекти в C++

```
class Triangle {
    double a;
    double b;
    double c;

    double perimeter() {
        // ...
    }
};
```

Фрагмент 1: Пример за декларация на клас

- За разлика от функциите, методите не могат да бъдат извикани без да се посочи обект, с който да работят

Създаване на обекти

- За да бъде създаден обект е нужно само да се създаде променлива с типа на предварително деклариран клас
- Класът се посочва с ключовата дума `class` или `struct` последвана от името на класа

```
int main() {
    class Triangle t;
    return 0;
}
```

Фрагмент 2: Пример за създаване на обект от клас Triangle

Достъп до член-променливи и член-функции на обект

- Подобно на структурите, членовете на обектите от определен клас се достъпват чрез оператора за достъп (точка)

```
class Rectangle {
public:
    int a;
    int b;
    int area() {
        // ...
    }
}
int main() {
    class Rectangle r;
    r.a = 10;
    r.b = 5;
    return 0;
}
```

Фрагмент 3: Достъп до член-променливи и член-функции на обект

Достъп до член-променливи и член-функции на обект

- За да бъде извикан един метод, той трябва да бъде извикан върху някакъв обект
- В C++ методите се извикват като се приложи оператор за достъп до поле на обект (точка) върху някой обект и се използва оператора за извикване на функция (кръгли скоби)

```
int main() {
    class Triangle t;
    t.a = 10;
    t.b = 5;
    printf("%f", t.perimeter());
    return 0;
}
```

Фрагмент 4: Пример за извикване на метод

Капсулация на класове в C++

- В C++ може да се ограничава достъпа до членовете (член-променливите и член-функциите (методите)) на класа, чрез модификатори на достъп
- Модификаторите на достъп се поставят чрез синтаксис подобен на поставяне на етикети с разликата, че могат да бъдат поставяни само преди началото на декларацията на член от тялото на класа
- Ограничаването на достъпа важи за членовете от мястото на модификатора до края на тялото на класа или докато не се срещне друг модификатор за достъп
- Съществуват три модификатора на достъп - `public`, `private` и `protected`
- Модификаторът `public` позволява неограничен достъп до членовете на класа, докато `private` забранява всякакъв достъп, освен от методите на класа
- По подразбиране, членовете в клас дефиниран с ключовата дума `class` са `private`, а със struct - `public`

Капсулация на класове в C++

```
class Triangle {
    double a;
public:
    double b;
private:
    double c;
    double perimeter() { /* ... */ };
};
int main() {
    class Triangle t;
    t.a = 11.1; // ERROR, 'a' is private member of Triangle
    t.b = 7.6; // OK, 'b' is public member
    t.c = 22.5; // ERROR, 'c' is private member
    double p = t.perimeter(); // ERROR, 'perimeter' is private member
    printf("%f\n", p);
    return 0;
}
```

Фрагмент 5: Пример с употреба на модификатори

Методи

- Методите в C++ винаги имат параметър с име `this`, който компилатора винаги автоматично добавя към декларациите на методите
- При извикване на метод, параметърът `this` автоматично се инициализира с указател към обекта, върху който метода е бил извикан
- Целта на параметъра `this` е метода да може да достъпва данните на обекта върху, който е бил извикан

```
class Triangle {
public:
    double a;
    double b;
    double c;
    double perimeter() {
        return this->a + this->b + this->c;
    }
};
```

Фрагмент 6: Пример за използване на `this` параметър в тяло на метод

Методи

```
int main() {
    class Triangle t;
    t.a = 11.1;
    t.b = 7.6;
    t.c = 22.5;
    printf("%f\n", t.perimeter());
}
```

Фрагмент 7: Извикване на метод, използващ `this` параметър

- Методите могат да приемат параметри, подобно на функциите
- Да се декларира от програмиста параметър `this` на метод е компилационна грешка в C++

Съдържание

Конструктори

- Тъй като (почти) винаги трябва един обект да се инициализира за да бъде използван успешно, в C++ е въведена възможността да се декларират специални методи на класа, които да служат за инициализация на обекта. Тези методи се наричат `конструктори`
- За да се декларира конструктор, името на метода трябва да съвпада точно с името на класа и да бъде изпуснат типа, който метода връща
- Причината за изпускането на типа е, че по замисъл конструктора трябва само да инициализира обекта - безсмислено е да връща резултат
- Извикването на конструктор се извършва чрез специален синтаксис при деклариране на обект. След името на обекта се поставят кръгли скоби и се предават в списък аргументите на конструктора. Ако конструкторът няма параметри, кръглите скоби трябва да се изпуснат - т.е. остава само декларацията на обекта - въпреки това конструктора ще бъде извикан

Конструктори

```
class Triangle {
    double a;
    double b;
    double c;
public:
    // declaration of constructor
    Triangle(double a, double b, double c) {
        this->a = a;
        this->b = b;
        thic->c = c;
    }
};
int main() {
    class Triangle t(3.2, 4.1, 7.1); // calls the constructor
    return 0;
}
```

Фрагмент 8: Пример за декларация и извикване на конструктор

Деструктори

- Понякога е удобно да може да се извика метод, точно преди обекта да се унищожи
- В C++ има специална поддръжка за деклариране на такъв метод, който се извиква автоматично точно преди паметта за обекта да се унищожи. Такъв метод още се нарича *деструктор*
- За да се декларира деструктор, името на метода трябва да съвпада точно с името на класа и да се сложи '~' (тилда) пред него. Методът не трябва да приема аргументи и типът, който връща, трябва да бъде изпуснат

Деструктори

```
class Triangle {
    double a;
    double b;
    double c;
public:
    Triangle() {
        printf("A triangle was created!\n");
    }
    // declaration of destructor
    ~Triangle() {
        printf("A triangle was destroyed...\n");
    }
};
int main() {
    class Triangle t; // the constructor is called,
                      // prints 'A triangle was created!'
    printf("Hello!\n");
    return 0; // destructor is called just before leaving the block,
              // prints 'A triangle was destroyed'
}
```

Фрагмент 9: Деклариране на деструктор и пример