Класове

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Информатика, 2019/20 г.

4 март 2020 г.

Какво са класовете?

- Основен инструмент на ООП
- Средство за дефиниране на абстрактни типове данни
- Синтактична конструкция, която позволява логическо групиране на данни и операциите над тях

Дефиниция на клас

Дефиницията на клас се състои от:

- Декларации на **член-данни (полета)**
- Декларации на **член-функции (методи)**
  - конструктори
  - селектори
  - мутатори
  - деструктор

Дефиниция на клас

- <клас> ::= `class` <име-на-клас> `{` <тяло> `};`
- <име-на-клас> често е съществително име с главна буква
- <тяло> ::= { <декларация>`;` }
- <декларация> ::= <член-данна> | <конструктор> | <мутатор> | <селектор> | <деструктор>
- <член-данна> ::= [<достъп>`:`] <тип> <име> {`,` <име>}
- <конструктор> ::= [<достъп>`:`] <име-на-клас>`(`<параметри>`)`
- <мутатор> ::= [<достъп>`:`] <тип> <име> `(`<параметри>`)`
- <селектор> ::= [<достъп>`:`] <тип> <име> `(`<параметри>`)` `const`
- <деструктор> ::= [<достъп>`:`] `~`<име-на-клас>`()`
- <достъп> ::= `private` | `protected` | `public`

Дефиниция на клас — заглавни файлове

- Може да присъства само един път в даден файл
- Обикновено се пише в заглавен (header) файл с разширение .hpp
- Файловете, които използват класа, включват дефиницията му чрез включване на заглавния файл с `#include`
- Пример:

```
class Rational {
private:
  int numer, denom;
public:
  Rational();
  int getNumerator() const;
  void read();
};
```

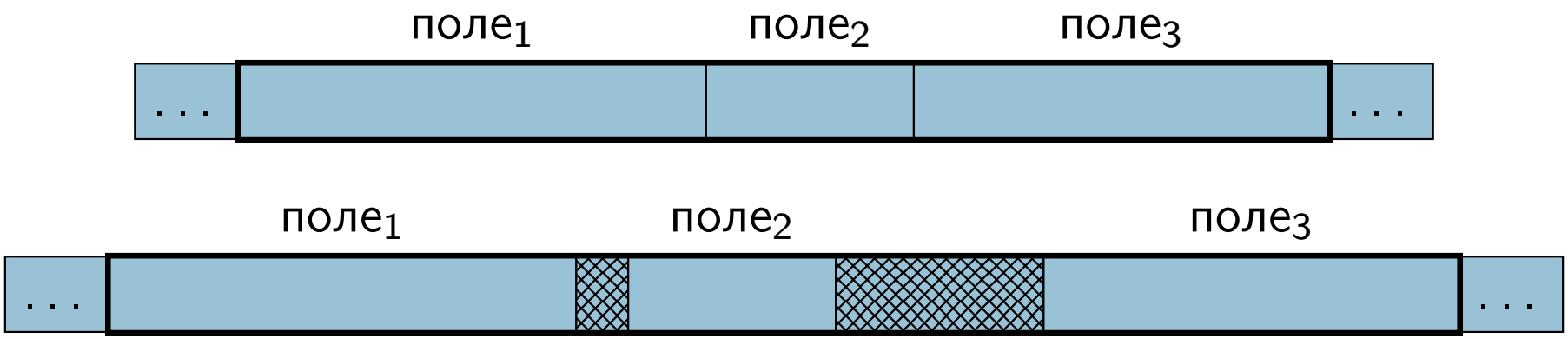
Дефиниция на клас — особености

- Конструкторите и деструкторите нямат тип
- Деструкторът няма параметри
- Прието е член-данни и член-функции да са разделени
- Директната рекурсията е забранена, както при записи
  - ~~`class Employee { Employee boss; ... };`~~
- Индиректната рекурсия (чрез указател) е позволена
  - `class Employee { Employee* boss; ... };`
- Член-функциите могат да са от всякакъв тип, включително и същия клас
  - `class Employee { ... Employee getBoss() const; };`

Обекти

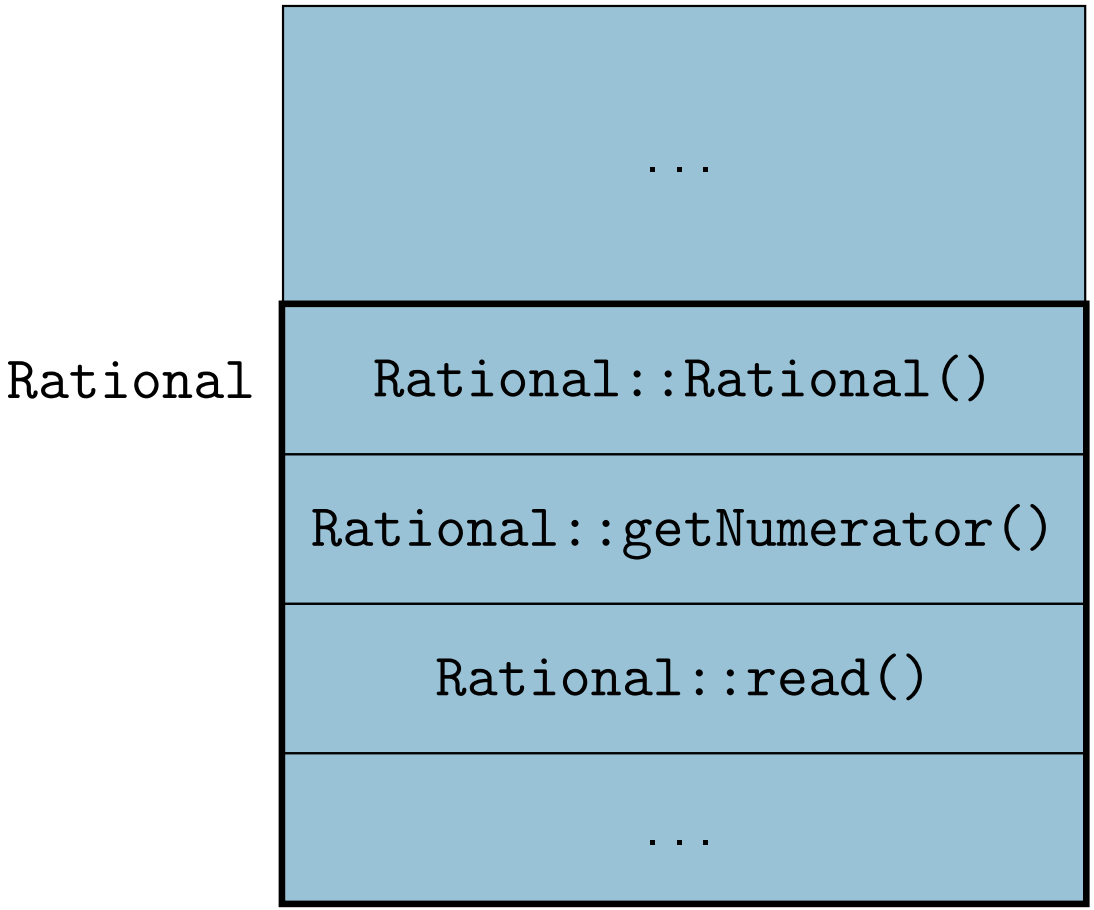
- Променливите от тип някой клас се наричат **обекти** или **инстанции на класа**
- <дефиниция-на-обект> ::= (<име-на-клас> | <клас>)
  <описание-на-обект> {, <описание-на-обект> };
- <описание на обект> ::=
  <име-на-обект> [= <израз>] |
  <име-на-обект>(<параметри>) |
  <име-на-обект> = <име-на-клас>(<параметри>)

Представяне на обекти в паметта



- паметта за даден обект представлява непрекъсната последователност от блокове памет за всяко то полетата му
- всеки обект от даден клас заема едно и също количество памет
- `sizeof`(<клас>) или `sizeof`(<обект>) връщат големина на <обект> от <клас> в байтове
- `sizeof`(<обект>) $\geq$ `sizeof`(<поле$_1$>) + `sizeof`(<поле$_2$>) + `sizeof`(<поле$_3$>)
- Полетата в класовете се подравняват до адрес кратен на големината им, както и при записите

Представяне на класове в паметта



Достъп до компонента на обект

- <обект>.<член-данна>
- <обект>.<член-функция>(<параметри>)
- Всеки обект има собствени стойности на член-данните
- Кодът на член-функциите е общ за всички обекти на класа

Достъп до компонента през указател към обект

- (*<указател-към-обект>).<член-данна>
- (*<указател-към-обект>).<член-функция>(<параметри>)
- <указател-към-обект>-><член-данна>
- <указател-към-обект>-><член-функция>(<параметри>)
- С указатели към обекти се работи както с указатели към обикновени променливи

Указател `this`

- В член-функциите имаме достъп до компонентите без да се указва обект
- Използва се обектът, за който е извикана член-функцията
- Как член-функциите разбират за кой обект са извикани?
- При всяко извикване на член-функция се създава автоматично специален константен указател с име `this`:
  <име-на-клас> `* const this`
- `this` винаги сочи към обекта, за който е извикана член-функцията
- За селекторите освен, че е константен, указателят `this` сочи към константа:
  <име-на-клас> `const * const this`

this като неявен параметър

Компилаторът автоматично и скрито от нас превежда член-функциите, така че:

- да получават this като първи параметър
- всяка компонента на обекта в тялото се достъпва през this

Пример 1:

```
void Rational::read() {
  cin >> numer >> denom;
}
```

... се превежда до ...

```
void Rational::read(Rational* const this) {
  cin >> this->numer >> this->denom;
}
```

```
r.read();    ... се превежда до ...    Rational::read(&r);
```

this като неявен параметър

Пример 2:

```
int Rational::getNumerator() const {
  return numer;
}
```

... се превежда до ...

```
int Rational::getNumerator(Rational const * const this) {
  return this->numer;
}
```

```
cout << r.getNumerator();
```

... се превежда до ...

```
cout << Rational::getNumerator(&r);
```

Режими на достъп

Имаме два режима за достъп:

- вътрешен достъп:
  Достъп до компоненти на класа от член-функции от същия клас
- външен достъп:
  Достъп до компоненти на класа от други функции:
  - обикновени функции
  - член-функции на друг клас

Спецификатори за достъп

В C++ имаме следните спецификатори за достъп:

- `private`
  - позволен е само вътрешен достъп
- `public`
  - позволен е вътрешен и външен достъп
- `protected`
  - позволен е вътрешен и ограничен външен достъп
  - подробностите: по-късно
- спецификатор по подразбиране е `private`
  - в `struct` е `public`

Указване на достъп

- След първото използване на спецификатор за достъп, той остава валиден за всички последващи декларации
- Спецификатор за достъп може да бъде използван произволен брой пъти
- **Пример:**

```
class Example {
  int a;                    // private
  double b;                 // private
public:
  Example();                // public
  int getA() const;         // public
private:
  void setB(double b);      // private
};
```

Операция за указване на област

- Всеки тип запис или клас дефинира **област (scope)**
- Имената на променливи и функции въведени в дадена област се виждат само в нея
- За да достъпим имената извън областта, в която са дефинирани е необходимо да укажем освен името и областта, която имаме предвид
- За целта използваме оператора за указване на област `::`
- [<област>]`::`<име>
- <област> може да е запис, клас или пространство от имена (namespace)
- Ако <област> е пропусната се подразбира глобалното пространство от имена
- Име за което е указана областта се нарича **квалифицирано име (qualified name)**

Примери за указване на област

- `Rational::read` — член-функцията read на класа `Rational`
- `Student::read` — член-функцията read на класа `Student`
- `::read` — глобалната функция `read`
- Операцията `::` се използва, когато има нужда да се разреши нееднозначност (ambiguity)

Дефиниция на член-функция

Синтаксис за дефиниране на член-функции:

- <член-функция> ::=
  [inline] [<тип>] <име-на-клас>::<име-на-член-функция>
  (<параметри>) { <тяло> }
- Прието е член-функциите да се дефинират в изходния (source, .cpp) файл, а не в заглавния (header, .hpp) файл
- Защо?
  - Заради принципа за капсулация
  - Потребителите на класа трябва да знаят какви член-функции има, но не и как са реализирани

Вградени (`inline`) член-функции

- По изключение се допуска член-функциите да се дефинират в дефиницията на класа
  `class Rational {... Rational() { numer = 0; denom = 1; } };`
- Такива функции се наричат вградени
- Вградените функции не се извикват със стекови рамки
- Тяхното тяло се замества при всяко тяхно извикване
- Една вградена функция може да е дефинирана извън дефиницията на класа
- Преди дефиницията се поставя запазената дума `inline`
- Окончателното решение дали една функция да е вградена е на компилатора!
- Препоръчително е да се вграждат само кратки функции

Примери за дефиниране на клас

- Точка в равнината
- Точка в пространството
- Пирамида