Обектно ориентирано програмиране

Класове. Дефиниране на класове

Дефиниране на класове

Класовете осигуряват механизми за създаване на напълно нови типове данни, които могат да бъдат интегрирани в езика, а също за обогатяване възможностите на вече съществуващи типове.

Дефинирането на един клас се състои от две части:

- декларация на класа и
- дефиниция на неговите член-функции (методи).

Декларация на клас

Декларацията на клас се състои от *заглавие* и *тяло*.

Заглавието започва със запазената дума **class**, следвано от името на класа.

Тялото е заградено във фигурни скоби. След скобите стои знакът “;” или списък от обекти. В тялото на класа са декларирани членовете на класа (член-данни и член-функции) със съответните им нива на достъп.

Декларация на клас ...

<декларация_на_клас> ::= <заглавие> <тяло>

<заглавие> ::= class [<име_на_клас>]

<тяло> ::= { <декларация_на_член>;

{<декларация_на_член>;}

};

<декларация_на_член> ::= [<спецификатор_на_достъп>:] <декларация_на_функция> |

<декларация_на_данна>

<спецификатор_на_достъп> ::= private | public | protected

<декларация_на_функция> ::= [<тип>] <име_на_функция>(<параметри>)

<декларация_на_данна> ::= <тип> <име_на_данна> {, <име_на_данна>}

<тип> ::= <име_на_тип> | <дефиниция_на_тип>

Декларация на клас ...

За имената на класовете важат същите правила, които се прилагат за имената на всички останали типове и променливи. Също като при структурите името на класа може да бъде пропуснато.

Имената на членовете на класа са **локални** за него, т.е. в различни класове в рамките на една програма могат да се дефинират членове с еднакви имена.

Член-данни от един и същ тип могат да се изредят, разделени със запетая и предшествани от типа им.

Декларация на клас ...

```
class Point
{
private:
    double x, y;
public:
    Point(double, double);
    void read();
    double getX() const;
    double getY() const;
    void print() const;
};
```

Декларация на клас ...

Препоръчва се член-данните да се декларират в нарастващ ред по броя на байтовете, необходим за представянето им в паметта. Така за повечето реализации се получава оптимално изравняване до дума.

- Типът на член-данна на клас **не може** да съвпада с името на класа, т. е. директна рекурсия е забранена

~~`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; };
```

Декларация на клас ...

В тялото, някои декларации на членове могат да бъдат предшествани от **спецификаторите на достъп** private, public или protected.

Областта на един спецификатор на достъп започва от спецификатора и продължава до следващия спецификатор.

Подразбиращ се спецификатор за достъп е **private**. Един и същ спецификатор на достъп може да се използва повече от веднъж в декларация на клас.

Препоръчва се, ако секция public съществува, да бъде първа в декларацията, а секцията private да бъде последна в тялото на класа.

Декларация на клас ...

Достъпът до членовете на класовете може да се разгледа на следните две нива:

- **вътрешен достъп**

  Достъп до компоненти на класа от член-функции на същия клас

- **външен достъп**

  Достъп до компоненти на класа от функции, които не са член-функции на същия клас
  - обикновени функции
  - член-функции на друг клас

Вътрешен достъп

По отношение на *член-функциите в класа* е в сила, че те имат достъп до всички членове на класа. При това не е необходимо тези компоненти да се предават като параметри. Този режим на достъп се нарича **режим на пряк достъп**.

Поради тази причина функциите Rational(), read(), print(), getNumerator() и getDenominator() са без параметри.

Вътрешен достъп

Освен това член-функцията print() може да бъде дефинирана и по следния начин:

```
void Rational::print() const {
    cout << getNumerator() << '/' <<
            getDenominator();
}
```

Външен достъп

По отношение на *функциите, които са външни за класа*, режимът на достъп са определя от начина на деклариране на членовете.

Членовете на даден клас, декларирани като **private** са видими (достъпни) само в рамките на класа. Външните функции нямат достъп до тях.

Чрез използването на членове, обявени като **private**, се постига скриване на членове за външната за класа среда. Процесът на скриване се нарича още **капсулиране на информацията**.

Членовете на клас, които трябва да бъдат видими извън класа (да бъдат достъпни за функции, които не са методи на дадения клас) трябва да бъдат декларирани като **public**.

Освен като private и public, членовете на класовете могат да бъдат декларирани и като **protected**.

Дефиниране на методите на клас

Дефинициите са аналогични на дефинициите на функции, но името на метода се предшества от името на класа, на който принадлежи метода, следвано от оператора за принадлежност **::** (Нарича се още оператор за област на действие). Такива имена се наричат **пълни**. (Операторът :: е ляво-асоциативен и с един и същ приоритет със (), [] и ->).

Дефиниция на метод на клас

<дефиниция_на_метод_на_клас> ::=

[<тип>] <име_на_клас>::<име_на_функция>(<параметри>) [const]

{ <тяло> }

<тяло> ::= <редица_от_оператори_и_дефиниции>

Дефиниране на методите на клас

Ще отбележим, че дефиницията на конструктор **не започва** с <тип>

Запазената дума const може да присъства само в дефинициите на функциите за достъп. Добрият стил на програмиране изисква използването на const в дефинициите на функциите за достъп и също в техните декларации. Ако се пренебрегне това изискване, могат да се създадат класове, които да не могат да се използват от други програмисти.

Дефиниране на методите на клас

Пример: Нека искаме да използваме класа Rational, но програмистът му е забравил или нарочно не е декларирал член-функцията print() като const и Rational има вида:

```
class Rational {
private:
    …
public:
    …
    void print();
};
```

Дефиниране на методите на клас

Нека декларираме класа MyClass, използващ класа Rational, коректно, т.е. функциите за достъп обявяваме като const.

```
class MyClass
{
private:
    int a;
    Rational p;// използване на класа Rational
    ...
public:
    ...
    void print() const;
};
```

Дефиниране на методите на клас

```
void MyClass::print() const
{
    cout << a << endl;
    p.print(); // тази print() е член-функцията
               // на класа Rational
};
```

Компилаторът ще съобщи за грешка в обръщението p.print(), защото p е обект на класа Rational, а член-функцията Rational::print() не е декларирана като const.

Дефиниране на методите на клас

Обикновено дефинициите на методите са разположени веднага след декларирането на класа, на който те са членове.

Възможно е обаче, дефинициите на методите на един клас да бъдат част от декларациите на този клас, т.е. в декларациите на член-функциите на класа могат да се зададат не само прототипите им, но и техните тела.

Дефиниране на методите на клас

```
class Rational {
private:
   int numer, denom;
   int gcd(int a, int b);
public:
// конструктори
Rational() {
   numer = 0;
   denom = 1;
};
```

Дефиниране на методите на клас

```
Rational(int n, int d) {
   if (n == 0 || d == 0) {
      numer = 0;
      denom = 1;
   } else {
      int g = gcd(abs(n), abs(d));
      if (n > 0 && d > 0 || n < 0 && d < 0) {
         numer = abs(n) / g;
         denom = abs(d) / g;
      } else {
         numer = -abs(n) / g;
         denom = abs(d) / g;
      }
   }
};
```

Дефиниране на методите на клас

```
// функции за достъп
int getNumerator() const {
   return numer;
};
int getDenominator() const {
   return denom;
};
void print() const {
   cout << getNumerator() << '/' << getDenominator();
};
```

Дефиниране на методите на клас

```
    // мутатор
    void read() {
        // Пример: 2/5
        cin >> numer;
        cin.ignore();
        cin >> denom;
    };
};
```

В този случай обаче член-функциите се третират като **вградени (inline) функции**.

Дефиниране на методите на клас

С цел повишаване на бързодействието, езикът C++ поддържа т.нар. **вградени (inline) функции.** Кодът на тези функции не се съхранява на едно място, а се копира на всяко място в паметта, където има обръщение към тях. Използват се като останалите функции, но при декларирането и дефинирането им заглавието им се предшества от модификатора inline.

Ще добавим, че дефиницията на inline функция трябва да се намира в същия файл, където се използва, т.е. не е възможна разделна компилация, тъй като компилаторът няма да разполага с кода за вграждане.

Използването на inline функции води до икономия на време, за сметка на паметта. Затова се препоръчва използването им само при “кратки” функции.

Дефиниране на методите на клас

Често член-функциите се реализират като inline функции. Това увеличава ефективността на програмата, използваща класа.

Декларацията на inline член-функции може да се осъществи и по следния начин:

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

Дефиниране на методите на клас

```
inline Rational::Rational() {
    numer = 0;
    denom = 1;
}
inline Rational::Rational(int n, int d) {
…
}
…
```

Дефиниране на методите на клас

В тялото на дефиницията на член-функция явно не се указва обектът, върху който тя ще се приложи. Този обект участва неявно - чрез член-данните на класа. Заради това се нарича **неявен параметър,** а член-данните **– абстрактни данни**.

Връзката между неявния параметър и обект ще бъде показана в по-късно. Параметри, които участват явно в дефиницията на член-функция се наричат **явни**.

Всяка член-функция има точно един неявен параметър и нула или повече явни.

Дефиниране на методите на клас

Обикновено декларацията на един клас се поставя в .h файл, а дефинициите на методите на класа – в съответен .cpp файл. Това позволява лесно да се създават библиотеки от класове.

Дефиниране на методите на клас

```
// файл Point.h
class Point
{
private:
   int x;
   int x;
public:
   Point(int, int);
   void read();
   …
};
```

Дефиниране на методите на клас

```
// файл Point.cpp
#include "Point.h"
Point::Point(int a, int b)
{
x = a;
y = b;
}
void point::read()
...
```

Дефиниране на методите на клас

```
// файл prog.cpp
#include <iostream>
#include "Rational.h"
#include "Point.h"
void main()
{
    Rational q(1, 3);
    Point a(5, 5);
    …
}
```

Област на класовете

За разлика от функциите, класовете могат да се декларират на различни нива в програмата: *глобално* (ниво функция) и *локално* (вътре във функция или в тялото на клас).

Областта на глобално деклариран клас започва от декларацията и продължава до края на програмата.

Ако клас е деклариран във функция, всички негови член-функции трябва да са inline. В противен случай ще се получат функции, дефинирани във функция, което не е възможно.

Област на класовете

```
void f(int i, int* p)
{
   int k;
   class CL
   {
      public:
      // всички методи са дефинирани в тялото на класа
      ...
      private:
      ...
   };
   // тяло на функцията f
   CL x;
   ...
}
```

Област на класовете

Областта на клас, дефиниран във функция, е функцията. Обектите на такъв клас са видими само в тялото на функцията.

Не е възможно в тялото на локално дефиниран клас да се използва функцията, в която класът е дефиниран.

```
void f(int i, int* p)
{
   int k;
   class CL
   {
      // не може да се използва функцията f
      ...
   };
   // тяло на функцията f
   ...
}
}
```

Обекти

След като даден клас е дефиниран, могат да бъдат създавани негови екземпляри, които се наричат **обекти**. Връзката между клас и обект в езика C++ е подобна на връзката между тип данни и променлива, но за разлика от обикновените променливи, обектите се състоят от множество **компоненти** (член-данни и член-функции).

Обекти

Дефиниция на обект на клас

<дефиниция-на_обект_на_клас> ::=

<име_на_клас> <обект>

[=<име_на_клас>(<фактически_параметри>)]

{, <обект>[=<име_на_клас>(<фактически_параметри>)] }

{, <обект>(<фактически_параметри>)}

{, <обект> = <вече_дефиниран_обект>};

<обект> ::= <идентификатор>

Обекти

Когато за даден клас явно са дефинирани конструктори, при всяко дефиниране на обект на класа те автоматично се извикват с цел да се инициализира обектът. Ако дефиницията е без явна инициализация (например `Rational p;`), дефинираният обект се инициализира според дефиницията на конструктора по подразбиране, ако такъв е определен, и се съобщава за грешка в противен случай. Ако дефиницията е с явна инициализация, обръщението към конструкторите трябва да бъде коректно.

Пример:

```
Rational p, q(2, 3), r = Rational(3, 8);
Rational t = q;
```

Обекти

Когато за даден клас явно не е дефиниран конструктор, компилаторът автоматично генерира **подразбиращ се конструктор**. Този конструктор изпълнява редица действия, като заделяне на памет за обектите, инициализиране на някой системни променливи и др. Дефиницията на обект от този клас трябва да е без явна инициализация.

Пример:

```
#include <iostream>
class Pom
{
private:
    int a;
```

Обекти

```
public:
   int b;
   void read();
   void print() const;
};
void main()
{
   Pom x;  // инициализация според подразбиращия се
           //конструктор, генериран от компилатора на C++
   x.read();
   x.print();
}
```

Обекти

```
void Pom::print()const
{
   cout << "a= " << a << " b=" << b << endl;
}
void Pom::read()
{
   cout << "a= ";
   cin >> a;
   cout << "b= ";
   cin >> b;
}
```

Обекти

Декларацията на клас не заделя памет за него. Памет се заделя едва при дефинирането на обект от класа. Дефиницията

```
Rational p, q(2, 3), r = Rational(3, 8);
```

заделя за обектите p, q и r по 8 байта ОП (по 4B за всяка от данните им numer и denom).

Достъпът до компонентите на обектите (ако е възможен) се осъществява чрез задаване на името на обекта и името на данната или метода, разделени с точка. Изключение от това правило правят конструкторите.

Обекти

Достъп до компонент на обект

<компонент_на_обект> ::=

<обект>.<данна> |

<обект>.<име_на_член_функция>() |

<обект>.<име_на_член_функция>(<параметри>)

<име_на_член_функция> е <идентификатор>, означаващ име на мутатор или име на функция за достъп.

Обекти

Пример:

```
Rational p(1, 2), q;
p.getNumerator(); // достъп до член-функцията
      // getNumerator() за обекта p
q.getNumerator() // достъп до член-функцията
      // getNumerator() за обекта q
```

Ще отбележим също, че на практика обектите p и q нямат свои копия на метода getNumerator (). И двете обръщения се отнасят за един и същ метод, но при първото обръщение се работи с данните за обекта p, а при второто – с данните за обекта q.

Обекти

При създаването на обекти на един клас кодът на методите на този клас не се копира във всеки обект, а се намира само на едно място в паметта.

Естествено възниква въпросът *по какъв начин методите на един клас "разбират" за кой обект на този клас са били извикани*. Отговорът на този въпрос дава указателят this. Всяка член-функция на клас поддържа допълнителен формален параметър - указател с име this и от тип <име_на_клас>*.

Указател this

- В член-функциите имаме достъп до компонентите без да се указва обект
- Използва се обекта, за който е извикана член-функцията
- Как член-функциите разбират за кой обект са извикани?
- При всяко извикване на член-функция се създава автоматично константен указател <име_на_клас> * const this
- this винаги сочи към обекта, за който е извикана членфункцията
- За селекторите: <име_на_клас> const * const this

this като неявен параметър

```
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

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);

Обекти

Обекти от един и същ клас могат да се присвояват един на друг. Присвояването може да е и на ниво инициализация.

Пример: Допустими са дефинициите

```
Rational p, q(4, 5), r = q;
p = q;
...
r = p;
```

При присвояването се копират всички член-данни на обекта. Така присвояването

```
r = p;
```

е еквивалентно на

```
r.numer = p.numer;
r.denom = p.denom;
```