Шаблони

Трифон Трифонов

Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

4 април 2019 г.

Повторение на код

```
class Point {
  double x, y;
  ...
  void translate(double a) {
    x += a; y += a;
  }
};
```

```
class UnsignedPoint {
  unsigned x, y;
  ...
  void translate(unsigned a) {
    x += a; y += a;
  }
};
```

```
class IntPoint {
  int x, y;
  ...
  void translate(int a) {
    x += a; y += a;
  }
};
```

```
class RationalPoint {
  Rational x, y;
  ...
  void translate(Rational a) {
    x += a; y += a;
  }
};
```

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч
  - функция с **параметри**

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч
  - функция с **параметри**
- Повторение на структура с различни стойности

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч
  - функция с **параметри**
- Повторение на структура с различни стойности
  - запис с **полета**

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч
  - функция с **параметри**
- Повторение на структура с различни стойности
  - запис с **полета**
  - клас с **член-данни**

Какво правим за да спестим повторенията?

- Повторение на изчисление с различни стойности
  - цикъл с **променлива** за брояч
  - функция с **параметри**
- Повторение на структура с различни стойности
  - запис с **полета**
  - клас с **член-данни**
- Какво се повтаря в предния пример?

Типови параметри

- **Шаблоните** в C++ позволяват дефинирането на “общи” функции и класове, които работят с неопределени типове по общ, унифициран начин

Типови параметри

- **Шаблоните** в C++ позволяват дефинирането на “общи” функции и класове, които работят с неопределени типове по общ, унифициран начин
- **Пример:**

```
template <typename T>
class Point {
  T x, y;
  ...
  void translate(T a) {
    x += a; y+= a;
  }
};
```

Типови параметри

- **Шаблоните** в C++ позволяват дефинирането на "общи" функции и класове, които работят с неопределени типове по общ, унифициран начин
- **Пример:**

```
template <typename T>
class Point {
  T x, y;
  ...
  void translate(T a) {
    x += a; y+= a;
  }
};
```

- Типът `T` може да бъде заместен с произволен тип, който поддържа операцията `+=`

Шаблони на функции

```
template <typename <параметър>[=<тип>]>
          {, typename <параметър>[=<тип>]}>
<сигнатура> { <тяло> }
```

Шаблони на функции

```
template <typename <параметър>[=<тип>]>
          {, typename <параметър>[=<тип>]}>
<сигнатура> { <тяло> }
```

- типовите параметри могат да участват в
  - тялото на функцията
  - типът на връщания резултат
  - типовете на параметрите
- типовите параметри могат да имат стойности по подразбиране

Шаблони на функции: примери

```
template <typename T>
void swap(T& a, T& b) {
  T tmp = a; a = b; b = tmp;
}
```

Шаблони на функции: примери

```
template <typename T>
void swap(T& a, T& b) {
  T tmp = a; a = b; b = tmp;
}

template <typename T>
void reverse(T* a, int n) {
  for(int i = 0; i < n/2; i++)
    swap(a[i], a[n - i - 1]);
}
```

Използване на шаблони на функции

- Явно указване на типовите параметри

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично
  - `int a[8] = { 0, 1, 2, 3, 4, 5, 6, 7 }; reverse<>(a, 8);`

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично
  - `int a[8] = { 0, 1, 2, 3, 4, 5, 6, 7 }; reverse<>(a, 8);`
- Шаблонът не се компилира

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично
  - `int a[8] = { 0, 1, 2, 3, 4, 5, 6, 7 }; reverse<>(a, 8);`
- Шаблонът не се компилира
- При всяко използване с различни типове генерира нова функция, която се компилира

Използване на шаблони на функции

- Явно указване на типовите параметри
  - `int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично
  - `int a[8] = { 0, 1, 2, 3, 4, 5, 6, 7 }; reverse<>(a, 8);`
- Шаблонът не се компилира
- При всяко използване с различни типове генерира нова функция, която се компилира
- Функция, генерирана от шаблон наричаме **шаблонна**

Специализации на шаблони на функции

- Можем да дефинираме “специална” версия на функцията за определени стойности на типовете

Специализации на шаблони на функции

- Можем да дефинираме “специална” версия на функцията за определени стойности на типовете
- **Пример:**

```
void swap(int& a, int& b) {
  a += b;
  b = a - b;
  a = a - b;
}
```

Специализации на шаблони на функции

- Можем да дефинираме "специална" версия на функцията за определени стойности на типовете
- **Пример:**

```
void swap(int& a, int& b) {
  a += b;
  b = a - b;
  a = a - b;
}
```

- Специализацията се използва вместо шаблона, освен при явно указване на параметрите

Специализации на шаблони на функции

- Можем да дефинираме "специална" версия на функцията за определени стойности на типовете
- **Пример:**

```
void swap(int& a, int& b) {
  a += b;
  b = a - b;
  a = a - b;
}
```

- Специализацията се използва вместо шаблона, освен при явно указване на параметрите
  - `swap<int>(a, b)` извиква шаблонната функция

Специализации на шаблони на функции

- Можем да дефинираме "специална" версия на функцията за определени стойности на типовете
- **Пример:**

```
void swap(int& a, int& b) {
  a += b;
  b = a - b;
  a = a - b;
}
```

- Специализацията се използва вместо шаблона, освен при явно указване на параметрите
  - `swap<int>(a, b)` извиква шаблонната функция
  - `swap(a, b)` извиква специализацията

Задачи

1. Да се напише функция, която въвежда масив

Задачи

1. Да се напише функция, която въвежда масив
2. Да се напише функция, която намира броя на срещанията на елемент в масив

Шаблони на класове

```
template <typename <параметър>[=<тип>]
          {, typename <параметър>[=<тип>]}>
  class <име> { <тяло> };
```

Шаблони на класове

```
template <typename <параметър>[=<тип>]
          {, typename <параметър>[=<тип>]}>
  class <име> { <тяло> };
```

Типовите параметри могат да се използват в:

- типовете на член-данните
- типовете на параметрите на член-функциите
- типа на връщан резултат на член-функциите
- в тялото на член-функциите

```
Point2D P;
```

Шаблони на класове

```
template <typename <параметър>[=<тип>]
          {, typename <параметър>[=<тип>]}>
  class <име> { <тяло> };
```

Типовите параметри могат да се използват в:

- типовете на член-данните
- типовете на параметрите на член-функциите
- типа на връщан резултат на член-функциите
- в тялото на член-функциите

Терминология:

- шаблон на клас: `template <typename T> class Point<T>`
- шаблонен клас: `Point<int>`

Член-функции на шаблонни класове

За член-функциите, които се дефинират извън класа:

- пред дефиницията се поставя:
  `template <typename` <параметър>`,`
  `{typename` <параметър>`}>`

Член-функции на шаблонни класове

За член-функциите, които се дефинират извън класа:

- пред дефиницията се поставя:
  `template <typename` <параметър>`,`
  `{typename` <параметър>`}>`
- пред името на функцията се поставя
  <шаблон>`<`<параметър>`{,` <параметър>`}>::`

Член-функции на шаблонни класове

За член-функциите, които се дефинират извън класа:

- пред дефиницията се поставя:
  `template <typename` <параметър>`,`
  `{typename` <параметър>`}>`
- пред името на функцията се поставя
  <шаблон>`<`<параметър>`{,` <параметър>`}>::`
- ако някой от типовете на параметрите или на връщаният резултат е шаблонен клас, също се указват всичките му типови параметри

Член-функции на шаблонни класове

За член-функциите, които се дефинират извън класа:

- пред дефиницията се поставя:
  `template <typename <параметър>,`
  `{typename <параметър>}>`
- пред името на функцията се поставя
  `<шаблон><<параметър>{, <параметър>}>::`
- ако някой от типовете на параметрите или на връщаният резултат е шаблонен клас, също се указват всичките му типови параметри

Примери:

```
template <typename T>
void Point<T>::translate(T a) {
  x += a; y += a;
}
```

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:
  - `Point<int> p;`

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:
  - `Point<int> p;`
  - `double distance (Point<double> p1, Point<double> p2) { ... }`

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:
  - `Point<int> p;`
  - `double distance (Point<double> p1, Point<double> p2) { ... }`
- Чрез дефиниране на потребителски тип

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:
  - `Point<int> p;`
  - `double distance (Point<double> p1, Point<double> p2) { ... }`
- Чрез дефиниране на потребителски тип
  - `typedef Point<double> DoublePoint;`

using DoublePoint = Point<double>;

Използване на шаблони на класове

- Шаблоните на класове се използват чрез явно указване на параметрите
  - параметрите по подразбиране могат да бъдат изпускани
- Директно инстанциране:
  - `Point<int> p;`
  - `double distance (Point<double> p1, Point<double> p2) { ... }`
- Чрез дефиниране на потребителски тип
  - `typedef Point<double> DoublePoint;`
- Използване в шаблон на функция

```
template <typename T>
double distance (Point<T> p1, Point<T> p2) { ... }
```

Особености на шаблоните на класове

- Шаблоните на класове не се компилират

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас
- Компилират се само член-функциите, които се използват от съответния шаблонен клас

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас
- Компилират се само член-функциите, които се използват от съответния шаблонен клас
  - докато шаблонът не бъде използван за конкретен тип, компилаторът не може да генерира код и да провери за грешки

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас
- Компилират се само член-функциите, които се използват от съответния шаблонен клас
  - докато шаблонът не бъде използван за конкретен тип, компилаторът не може да генерира код и да провери за грешки
  - може да не разберем, че има грешка в член-функция на шаблон, докато не я използваме!

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас
- Компилират се само член-функциите, които се използват от съответния шаблонен клас
  - докато шаблонът не бъде използван за конкретен тип, компилаторът не може да генерира код и да провери за грешки
  - може да не разберем, че има грешка в член-функция на шаблон, докато не я използваме!
- При всяко използване на шаблон се генерира нов програмен код

Особености на шаблоните на класове

- Шаблоните на класове не се компилират
- При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас
- Компилират се само член-функциите, които се използват от съответния шаблонен клас
  - докато шаблонът не бъде използван за конкретен тип, компилаторът не може да генерира код и да провери за грешки
  - може да не разберем, че има грешка в член-функция на шаблон, докато не я използваме!
- При всяко използване на шаблон се генерира нов програмен код
- `sizeof(T)` не е известен, затова не можем да правим обекти от шаблони на класове, а само обекти от шаблонни класове

Специализация на член-функции

Можем да дефинираме специални реализации на член-функциите при определени стойности на параметрите:

```
double Point<Rational>::distance(Point<Rational> const& p) const {
  Rational r = (p.x - x)*(p.x - x) + (p.y - y)*(p.y - y);
  return sqrt((double)r.getNumerator()/r.getDenominator());
}
```

Шаблони и приятели

- **приятел на шаблон**

```
template <typename T> class Point { ...  friend class Player; }
template <typename T> class Point
{ ...  friend operator<<(ostream&, Rational const& r); };
```

Шаблони и приятели

- **приятел на шаблон**

```
template <typename T> class Point { ...  friend class Player; }
template <typename T> class Point
{ ...  friend operator<<(ostream&, Rational const& r); };
```

- **шаблонен приятел**

```
class Player { ... friend class Point<int>; };
class Player { ... friend void swap(Point<int>&, Point<int>&);
```

Шаблони и приятели

- **приятел на шаблон**

```
template <typename T> class Point { ...  friend class Player; }
template <typename T> class Point
{ ...  friend operator<<(ostream&, Rational const& r); };
```

- **шаблонен приятел**

```
class Player { ... friend class Point<int>; };
class Player { ... friend void swap(Point<int>&, Point<int>&);
```

- **шаблонен приятел на шаблон**

```
template <typename T> class Point
{ ... friend class Stack<T>; };
template <typename T> class Point
{ ... friend void swap(Point<T>&, Point<T>&) };
```