

# Шаблони

# Повторение на код

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

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

- ```
class UnsignedPoint {  
    unsigned x, y;  
  
    ...  
    void translate(unsigned 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
  - цикъл с **променлива** за брояч
  - функция с **параметри**
- Повторение на структура с различни стойности
  - запис с **полета**
  - клас с **член-данни**
- Повторение на изчислителна схема с различни операции
  - функция от по-висок ред с **функции за параметри**
- Какво се повтаря в предния пример?

# Типови параметри

- **Шаблоните** в C++ позволяват дефинирането на „общи“ функции и класове, които работят с неопределени типове
- **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|class) <параметър>[=<тип>]>  
          {, (typename|class) <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

# Примери за шаблони на функция

- ```
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 параметрите  
`int a = 2, b = 3; swap<int>(a, b);`
- Подходящи типове могат да бъдат изведени автоматично  
`int a[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; reverse<>(a, 10);`
- 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
- ```
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

- Да се напише функция, която въвежда масив
- Да се напише функция, която намира броя на срещанията на елемент в масив

# Шаблони на класове

- `template <(typename|class) <параметър>[=<тип>]>  
 {, (typename|class) <параметър>[=<тип>]}>  
 class <име> { <тяло> };`
- типове параметри могат да участват в
  - типовете на член-данните
  - типовете на параметрите на член-функциите
  - типа на връщан резултат на член-функциите
  - в тялото на член-функциите

# Член-функции на шаблонни класове

- Ако функциите не са вградени
  - пред дефиницията се поставя  
`template <(typename|class) <параметър>,  
          {(typename|class) <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
- ```
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
  - 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) { ... }`

# Използване на шаблони 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
  - може да не разберем, че има грешка в член-функция на шаблон, докато не я използваме!

# 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());
}
```

# Особености на шаблоните

- `sizeof(T)` не е известен, затова не можем да правим обекти от шаблони на класове, а само обекти от шаблонни класове
- докато шаблонът не бъде използван за конкретен тип, компилаторът не може да генерира код и да провери за грешки
- при всяко използване на шаблон се генерира нов програмен код

# Шаблони и приятели

- **приятел на шаблон**

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

- **шаблонен приятел**

```
class Student { ... friend class Point<int>; };  
class Student { ... 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>&) };
```