

# Функции

## (част 1)

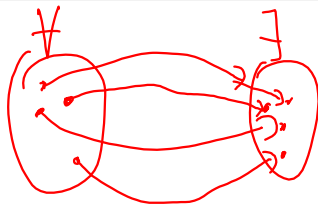
Трифон Трифонов

Увод в програмирането,  
спец. Компютърни науки, 1 поток,  
спец. Софтуерно инженерство,  
2016/17 г.

9 ноември 2016 г.

# Функциите в математиката

- Какво е функция в математиката?



# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$

# Функциите в математиката

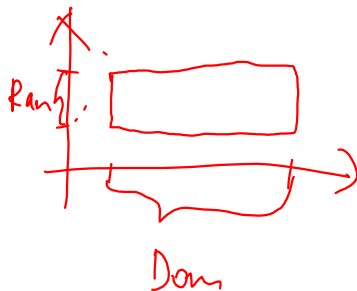
- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област

# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности

# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности
- Нека  $F \subseteq Dom \times Ran$



# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности
- Нека  $F \subseteq Dom \times Ran$
- така че  $\forall x \exists ! y(x, y) \in F$  (функционалност)



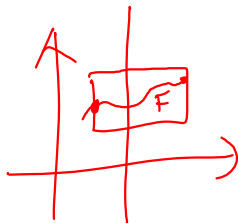
# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности
- Нека  $F \subseteq Dom \times Ran$
- така че  $\forall x \exists ! y(x, y) \in F$  (функционалност)
- еквивалентно: ако  $(x, y_1) \in F$  и  $(x, y_2) \in F$ , тогава  $y_1 = y_2$



# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности
- Нека  $F \subseteq Dom \times Ran$
- така че  $\forall x \exists y (x, y) \in F$  (функционалност)
- еквивалентно: ако  $(x, y_1) \in F$  и  $(x, y_2) \in F$ , тогава  $y_1 = y_2$
- Ако  $(x, y) \in F$  пишем  $f(x) = y$ .



$f(x)$

# Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \rightarrow Ran$ 
  - $Dom$  — дефиниционна област
  - $Ran$  — обхват, област от стойности
- Нека  $F \subseteq Dom \times Ran$
- така че  $\forall x \exists ! y(x, y) \in F$  (функционалност)
- еквивалентно: ако  $(x, y_1) \in F$  и  $(x, y_2) \in F$ , тогава  $y_1 = y_2$
- Ако  $(x, y) \in F$  пишем  $f(x) = y$ .
- $F$  — графика на функцията  $f$

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**
  - $f : \mathbb{R} \rightarrow \mathbb{R}$

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**

- $f : \mathbb{R} \rightarrow \mathbb{R}$

- $f(x) = x^2$

$$F = h(x, y) \mid y = x \cdot x, x, y \in \mathbb{R}$$



# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**

- $f: \mathbb{R} \rightarrow \mathbb{R}$

- $f(x) = x^2$

- `double square(double x) { return x * x; }`

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**
  - $f : \mathbb{R} \rightarrow \mathbb{R}$
  - $f(x) = x^2$
  - `double square(double x) { return x * x; }`
- **Пример 2**

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**
  - $f : \mathbb{R} \rightarrow \mathbb{R}$
  - $f(x) = x^2$
  - `double square(double x) { return x * x; }`
- **Пример 2**
  - $d : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане
- Може да бъде използвана многократно
- **Пример 1**
  - $f : \mathbb{R} \rightarrow \mathbb{R}$
  - $f(x) = x^2$
  - `double square(double x) { return x * x; }`
- **Пример 2**
  - $d : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
  - $d(x_1, y_1, x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

# Какво е програмна функция?

- Относително независима част от програмата, извършваща определено пресмятане

- Може да бъде използвана многократно

- **Пример 1**

- $f : \mathbb{R} \rightarrow \mathbb{R}$

- $f(x) = x^2$

- `double square(double x) { return x * x; }`

- **Пример 2**

- $d : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$

- $d(x_1, y_1, x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- `double distance(double x1, double y1, double x2, double y2)  
return sqrt(square(x2 - x1) + square(y2 - y1));  
}`

## Какво е подпрограма?

- Относително независима част от програмата, извършваща определена последователност от действия

## Какво е подпрограма?

- Относително независима част от програмата, извършваща определена последователност от действия
- Може да бъде изпълнявана многократно

## Какво е подпрограма?

- Относително независима част от програмата, извършваща определена последователност от действия
- Може да бъде изпълнявана многократно
- Още: процедура, метод



# Какво е подпрограма?

- Относително независима част от програмата, извършваща определена последователност от действия
- Може да бъде изпълнявана многократно
- Още: процедура, метод
- **Пример 1**

```
void printHello() {  
    cout << "Hello!\n";  
}
```

# Какво е подпрограма?

- Относително независима част от програмата, извършваща определена последователност от действия
- Може да бъде изпълнявана многократно
- Още: процедура, метод
- **Пример 1**

```
void printHello() {  
    cout << "Hello!\n";  
}
```

- **Пример 2**

```
void printReverseDigits(int n) {  
    while (n > 0) {  
        cout << n % 10;  
        n /= 10;  
    }  
}
```

# Процедури и функции

- Функцията извършва пресмятане и връща резултат

# Процедури и функции

- Функцията извършва пресмятане и връща резултат
- Процедурата изпълнява поредица от оператори

# Процедури и функции

- Функцията извършва пресмятане и връща резултат
- Процедурата изпълнява поредица от оператори
- Понякога двете понятия се смесват...

```
int readNumber(int from, int to) {  
    int n;  
    do {  
        cout << "n = "; cin >> n;  
    } while (n < from || n > to);  
    return n;  
}
```



readNumber(2, 5) → 3  
→ 4

# Процедури и функции

- Функцията извършва пресмятане и връща резултат
- Процедурата изпълнява поредица от оператори
- Понякога двете понятия се смесват...

```
int readNumber(int from, int to) {  
    int n;  
    do {  
        cout << "n = "; cin >> n;  
    } while (n < from || n > to);  
    return n;  
}
```

- В C++ се наричат просто “функции”

# Дефиниране на функция

- `<сигнатура> { <тяло> }`

# Дефиниране на функция



- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_результат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$



# Дефиниране на функция

- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_резултат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$ 
  - **void** = празен тип, не връща резултат

$$f: \mathbb{R} \rightarrow \emptyset$$

$$F \subseteq \mathbb{R} \times \emptyset$$

$$\downarrow$$

$$F = \emptyset$$

# Дефиниране на функция

- `<сигнатура> { <тяло> }`
- `<сигнатура> ::= [ <тип_результат> | void ]`  
`<идентификатор> ( <формални_параметри> )`
  - **void** = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`

# Дефиниране на функция

- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_резултат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$ 
  - **void** = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`
- $\langle \text{формални\_параметри} \rangle ::=$   
 $\langle \text{празно} \rangle \mid \text{void} \mid \langle \text{параметър} \rangle \{ , \langle \text{параметър} \rangle \}$

# Дефиниране на функция

- `<сигнатура> { <тяло> }`
- `<сигнатура> ::= [ <тип_результат> | void ]`  
`<идентификатор> ( <формални_параметри> )`
  - **void** = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`
- `<формални_параметри> ::=`  
`<празно> | void | <параметър> {, <параметър> }`
- `<параметър> ::= <тип> [<идентификатор>]`

# Дефиниране на функция

- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_резултат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$ 
  - `void` = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`
- $\langle \text{формални\_параметри} \rangle ::=$   
 $\langle \text{празно} \rangle \mid \text{void} \mid \langle \text{параметър} \rangle \{ , \langle \text{параметър} \rangle \}$
- $\langle \text{параметър} \rangle ::= \langle \text{тип} \rangle [ \langle \text{идентификатор} \rangle ]$ 
  - ако  $\langle \text{идентификатор} \rangle$  се пропусне, параметърът няма име и не се използва

# Дефиниране на функция

- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_резултат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$ 
  - `void` = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`
- $\langle \text{формални\_параметри} \rangle ::=$   
 $\langle \text{празно} \rangle \mid \text{void} \mid \langle \text{параметър} \rangle \{ , \langle \text{параметър} \rangle \}$
- $\langle \text{параметър} \rangle ::= \langle \text{тип} \rangle [ \langle \text{идентификатор} \rangle ]$ 
  - ако  $\langle \text{идентификатор} \rangle$  се пропусне, параметърът няма име и не се използва
  - Пример:  $f(x, y) = x + 5$

# Дефиниране на функция

- $\langle \text{сигнатура} \rangle \{ \langle \text{тяло} \rangle \}$
- $\langle \text{сигнатура} \rangle ::= [ \langle \text{тип\_резултат} \rangle \mid \text{void} ]$   
 $\langle \text{идентификатор} \rangle ( \langle \text{формални\_параметри} \rangle )$ 
  - `void` = празен тип, не връща резултат
  - ако типът на резултата се пропусне, подразбира се `int`
- $\langle \text{формални\_параметри} \rangle ::=$   
 $\langle \text{празно} \rangle \mid \text{void} \mid \langle \text{параметър} \rangle \{ , \langle \text{параметър} \rangle \}$
- $\langle \text{параметър} \rangle ::= \langle \text{тип} \rangle [ \langle \text{идентификатор} \rangle ]$ 
  - ако  $\langle \text{идентификатор} \rangle$  се пропусне, параметърът няма име и не се използва
  - Пример:  $f(x, y) = x + 5$
- $\langle \text{тяло} \rangle ::= \{ \langle \text{оператор} \rangle \}$

# Извикване на функция

- `<име>(<фактически_параметри>)`



# Извикване на функция

- $\langle \text{име} \rangle (\langle \text{фактически\_параметри} \rangle)$
- $\langle \text{фактически\_параметри} \rangle ::=$   
 $\langle \text{празно} \rangle \mid \text{void} \mid \langle \text{израз} \rangle \{, \langle \text{израз} \rangle \}$

# Извикване на функция

- `<име>(<фактически_параметри>)`
- `<фактически_параметри> ::=`  
`<празно> | void | <израз> {, <израз> }`
- извикването на функция всъщност е **операция** с много висок приоритет

*sqrt(5) + 7*

*sqrt(5)*

# Извикване на функция

- $\langle \text{име} \rangle (\langle \text{фактически\_параметри} \rangle)$
- $\langle \text{фактически\_параметри} \rangle ::= \langle \text{празно} \rangle \mid \text{void} \mid \langle \text{израз} \rangle \{, \langle \text{израз} \rangle \}$
- извикването на функция всъщност е **операция** с много висок приоритет
- типът на фактическия параметър се съпоставя с типа на съответния формален параметър

# Извикване на функция

- $\langle \text{име} \rangle (\langle \text{фактически\_параметри} \rangle)$
- $\langle \text{фактически\_параметри} \rangle ::= \langle \text{празно} \rangle \mid \text{void} \mid \langle \text{израз} \rangle \{, \langle \text{израз} \rangle \}$
- извикването на функция всъщност е **операция** с много висок приоритет
- типът на фактическия параметър се съпоставя с типа на съответния формален параметър
  - ако се налага, прави се преобразуване на типовете

# Извикване на функция

- $\langle \text{име} \rangle (\langle \text{фактически\_параметри} \rangle)$
- $\langle \text{фактически\_параметри} \rangle ::= \langle \text{празно} \rangle \mid \text{void} \mid \langle \text{израз} \rangle \{, \langle \text{израз} \rangle \}$
- извикването на функция всъщност е **операция** с много висок приоритет
- типът на фактическия параметър се съпоставя с типа на съответния формален параметър
  - ако се налага, прави се преобразуване на типовете
  - $\langle \text{формален\_параметър} \rangle = \langle \text{фактически\_параметър} \rangle$

# Връщане на резултат

- `return [<израз>];`

# Връщане на резултат

- `return` [<израз>;
- оператор за връщане на резултат на функция

# Връщане на резултат

- `return` [<израз>;
- оператор за връщане на резултат на функция
- типът на <израз> се съпоставя с типа на резултата на функцията



# Връщане на резултат

- `return` [<израз>;
- оператор за връщане на резултат на функция
- типът на <израз> се съпоставя с типа на резултата на функцията
  - ако се налага, прави се преобразуване на типовете

# Връщане на резултат

- `return` [<израз>;
- оператор за връщане на резултат на функция
- типът на <израз> се съпоставя с типа на резултата на функцията
  - ако се налага, прави се преобразуване на типовете
- работата на функцията се прекратява незабавно

# Връщане на резултат

- `return` [<израз>;
- оператор за връщане на резултат на функция
- типът на <израз> се съпоставя с типа на резултата на функцията
  - ако се налага, прави се преобразуване на типовете
- работата на функцията се прекратява незабавно
- стойността на <израз> е резултатът от извикването на функцията

# Деклариране на функция

*distance:  $\mathbb{R}^n \rightarrow \mathbb{R}$*

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$

# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция

# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна

# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти...

# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти...
- ...но може да бъде дефинирана **само веднъж**



# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти...
- ...но може да бъде дефинирана **само веднъж**
- Неизпълнените обещания водят до проблеми...

# Деклариране на функция

- $\langle \text{декларация\_на\_функция} \rangle ::= \langle \text{сигнатура} \rangle ;$
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти...
- ...но може да бъде дефинирана **само веднъж**
- Неизпълнените обещания водят до проблеми...
  - ...освен когато никой не разчита на тях