Функции
(част 1)

Трифон Трифонов

Увод в програмирането,
спец. Компютърни науки, 1 поток, 2018/19 г.

8 ноември 2018 г.

Функциите в математиката

- Какво е функция в математиката?

Функциите в математиката

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

Функциите в математиката

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

Функциите в математиката

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

Функциите в математиката

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

Функциите в математиката

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

дом. $\forall x \forall y_1, y_2 \quad (x, y_1) \in F \;\&\; (x, y_2) \in F \to y_1 = y_2$

имат. $\forall x_1, x_2 \forall y \quad (x_1, y) \in F \;\&\; (x_2, y) \in F \to x_1 = x_2$

Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \to Ran$
  - 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?
- $f : Dom \to Ran$
  - 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 $f(x) = y$.

Функциите в математиката

- Какво е функция в математиката?
- $f : Dom \to Ran$
  - 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
- **Пример 1**

Какво е програмна функция?

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

Какво е програмна функция?

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

Какво е програмна функция?

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

Какво е програмна функция?

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

Какво е програмна функция?

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

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

$f : \mathbb{R} \to \mathbb{R}$

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `void`]
  <идентификатор> (<**формални**_параметри>)

Дефиниране на функция

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `void`]
  <идентификатор> (<**формални**_параметри>)
  - `void` = празен тип, не връща резултат

Дефиниране на функция

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

Дефиниране на функция

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `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> }

Дефиниране на функция

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `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>]

int f(int, int)

Дефиниране на функция

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `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>]
  - ако <идентификатор> се пропусне, параметърът няма име и не се използва

Дефиниране на функция

- <сигнатура> `{` <тяло> `}`
- <сигнатура> ::= [<тип_резултат> | `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>]
  - ако <идентификатор> се пропусне, параметърът няма име и не се използва
  - **Пример:** $f(x, y) = x + 5$

double f(int x, int) { return x+5; }

Дефиниране на функция

- <сигнатура> { <тяло> }
- <сигнатура> ::= [<тип_резултат> | `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>]
  - ако <идентификатор> се пропусне, параметърът няма име и не се използва
  - **Пример:** $f(x, y) = x + 5$
- <тяло> ::= { <оператор> }

Извикване на функция

- <име>(<**фактически**_параметри>)

Извикване на функция

- <име>(<**фактически**_параметри>)
- <**фактически**_параметри> ::=
  <празно> | `void` | <израз> { , <израз> }

Извикване на функция

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

Извикване на функция

- <име>(<**фактически**_параметри>)
- <**фактически**_параметри> ::=
  <празно> | `void` | <израз> {, <израз> }
- 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

- <име>(<**фактически**_параметри>)
- <**фактически**_параметри> ::=
  <празно> | `void` | <израз> {, <израз> }
- 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

- <име>(<**фактически**_параметри>)
- <**фактически**_параметри> ::=
  <празно> | `void` | <израз> {`,` <израз> }
- 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
  - <формален_параметър> = <фактически_параметър>

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

- <декларация_на_функция> ::= <сигнатура>;

Деклариране на функция

- <декларация_на_функция> ::= <сигнатура>;
- Декларацията е “обещание” за дефиниция на функция

Деклариране на функция

- <декларация_на_функция> ::= <сигнатура>;
- Декларацията е “обещание” за дефиниция на функция
- Декларацията не е задължителна

Деклариране на функция

- <декларация_на_функция> ::= <сигнатура>;
- Декларацията е "обещание" за дефиниция на функция
- Декларацията не е задължителна
- Една функция може да бъде декларирана няколко пъти...

Деклариране на функция

- <декларация_на_функция> ::= <сигнатура>;
- 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

- <декларация_на_функция> ::= <сигнатура>;
- 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

- <декларация_на_функция> ::= <сигнатура>;
- 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

n = 5

1

2 3

3 4 5

4 5 6 7

5 6 7 8 9

n = 4

4 3 2 1 2 3 4

3 2 1 2 3

2 1 2

1

2 1 2

3 2 1 2 3

4 3 2 1 2 3 4

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

$$p = \frac{a+b+c}{2}$$

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