

Увод в програмирането

Рекурсия. Функции от по-висок ред

2017-2018 г.

ФМИ, специалност „Софтуерно инженерство“

Какво е рекурсия?



Какво е рекурсия?

- Повторение чрез самоизвикване
- “Приятелите на моите приятели са и мои приятели”
- PHP?
 - PHP Hypertext preprocessor
- GNU?
 - GNU is Not UNIX
- За да разберете какво е рекурсия, трябва да разберете какво е рекурсия

Рекурсивни функции в математиката

- Ако в дефиницията на някаква функция се използва самата функция, дефиницията на функцията се нарича рекурсивна.
- Примери:
 - а) Ако n е произволно естествено число, следната дефиниция на функцията факториел

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

- е рекурсивна. Условието при $n = 0$ не съдържа обръщение към функцията факториел и се нарича гранично (**още база или дъно на рекурсията**)

Рекурсивни функции в математиката ...

в) Функцията за повдигане на число на степен, се дефинира рекурсивно по следния начин:

$$x^n = \begin{cases} 1, & n = 0, \\ x \cdot x^{n-1}, & n > 0, \\ \frac{1}{x^{-n}}, & n < 0. \end{cases}$$

Рекурсивни функции в математиката ...

б) Функцията за намиране на най-голям общ делител на две естествени числа a и b може да се дефинира по следния рекурсивен начин:

$$\text{gcd}(a, b) = \begin{cases} a, & a = b \\ \text{gcd}(a - b, b), & a > b \\ \text{gcd}(a, b - a), & a < b. \end{cases}$$

Тук граничното условие е условието при $a = b$.

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

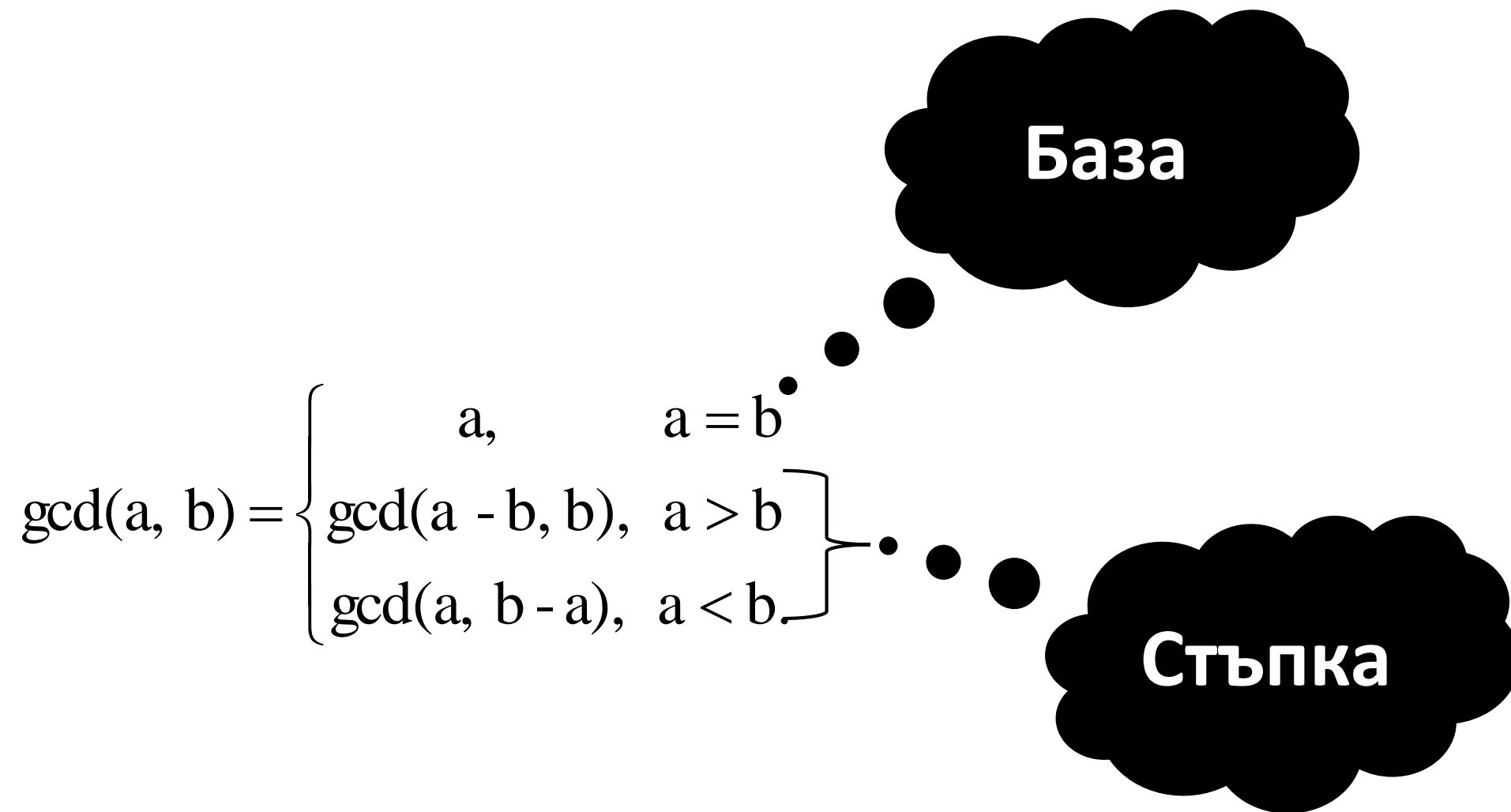
База

Стъпка

$$x^n = \left\{ \begin{array}{ll} 1, & n = 0, \\ x \cdot x^{n-1}, & n > 0, \\ \frac{1}{x^{-n}}, & n < 0. \end{array} \right\}$$

База

Стъпка



GCD - Greatest Common Divisor

Програмиране чрез рекурсия

- Рекурсията е друг начин на разделяне на дадена по-сложна задача на няколко на брой по-прости (еднакви) задачи, подобни на първоначалната
- Как работи:
 - Показваме решението на граничните, най-прости задачи (база, дъно)
 - Показваме как по-сложна задача се свежда към една или няколко по-прости (стъпка)
- Ако дадена програма може да се реализира с цикли, то тя може да се реализира и с рекурсия

Решаване на задачи с рекурсия

Пресмятане на $n!$ (за примера, нека $n=4$)

```
int factorial(int n) {  
    if (n < 0) return -1;  
    if (n == 0) return 1;  
  
    return n*factorial(n - 1);  
}  
  
int main()  
{  
    cout << factorial(4);  
  
    return 0;  
}
```

4! =
4.3! =
4.3.2! =
4.3.2.1! =
4.3.2.1.0! =
4.3.2.1.1 =
4.3.2.1 =
4.3.2 =
4.6 =
24

Решаване на задачи с рекурсия

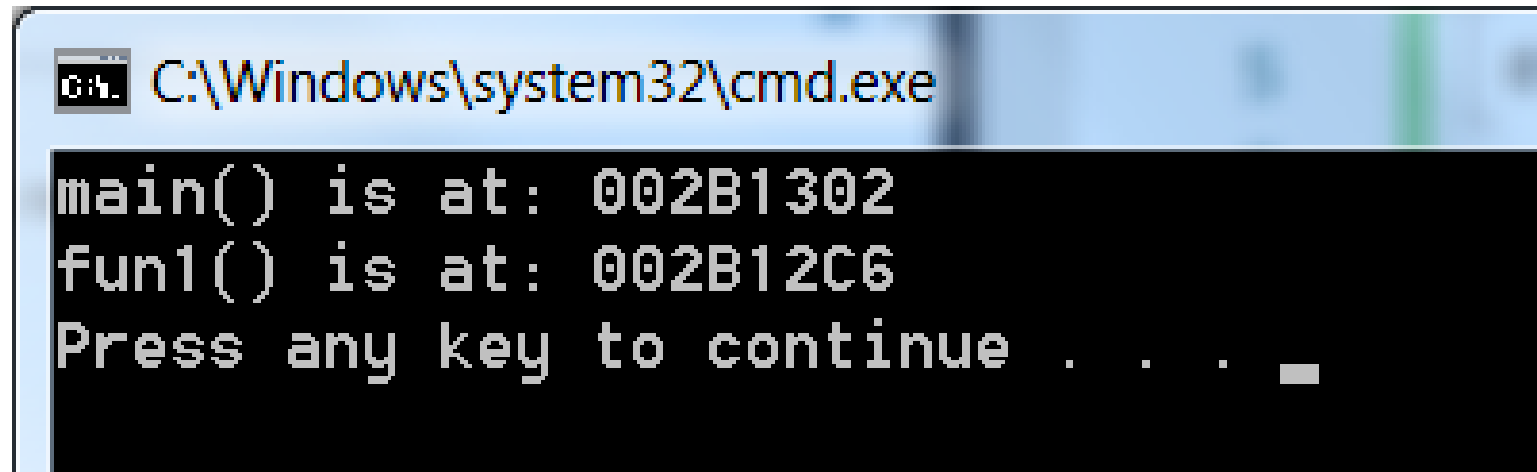
Пресмятане на най-голям общ делител

Пресмятане на степен

Функции от по-висок ред

```
int fun1(int par1) {  
    int a = 0;  
    a += par1;  
  
    return a;  
}
```

```
int main()  
{  
    cout << "main() is at: " << main << endl;  
    cout << "fun1() is at: " << fun1 << endl;  
}
```



A screenshot of a Windows command prompt window. The title bar shows the path "C:\Windows\system32\cmd.exe". The command prompt displays the output of the program: "main() is at: 002B1302", "fun1() is at: 002B12C6", and "Press any key to continue . . .".

Функции от по-висок ред

- Кодът на всяка функция на C++ се превежда до машинен код
- Машинният код на функциите е разположен в областта за програмен код
- Адрес на функцията се нарича адресът на първата инструкция във функцията
- Името на всяка функция може да се раглежда като константен указател към кода ѝ
- Можем да създаваме указатели към функции
- Стойността на указателя към функцията е адресът на нейния код

Указател към функция ...

- Дефиниция на указател към функция

<тип_на_функция> (*<указател_към_функция>)(<форм_пар>)
[= <име_на_функция>];

където

- <указател_към_функция> е идентификатор;
- <име_на_функция> е идентификатор, означаващ име на функция от тип <тип_на_функция> и параметри - <форм_пар>;
- <тип_на_функция> и <форм_пар> се дефинират аналогично на съответните от заглавието на дефиниция функция.
- Имената на параметрите могат да се пропуснат.

Указател към функция ...

- *Примери:*

```
double (*p)(double, double);
```

е дефиниция на променлива p от тип указател към функция от тип double с два аргумента също от тип double. В резултат за p се отделят 4 байта, които са с неопределена стойност.

```
int (*q)(int, int*);
```

дефинира променлива q от тип указател към функция от тип int, с два аргумента, единият от които цял, а другият – указател към int. За q се отделят 4 байта, които са с неопределена стойност.

Извикване на функция чрез указател

```
void(*f)(int&, int&) = swap;  
int x = 5, y = 8;
```

- Три еквивалентни начина за извикване на функцията са:
 - swap(x, y);
 - (*f)(x, y);
 - f(x, y);

Оператор typedef

- Използването на променливи, които са указатели към функции, усложнява записа на дефиницията на функция. Добре би било да дадем имена на типовете указател към функция и вместо дефиницията да използваме името на типа. Задаването на имена на типове може да се осъществи чрез оператора typedef.

- **Оператор typedef**

- *Синтаксис*

typedef <тип> <име>;

където

- <тип> е дефиниция на тип;
- <име> е идентификатор, определящ името на новия тип.

- *Семантика*

Определя <име> за синоним на типа от <тип>.

Оператор typedef...

- `typedef int number;`
- `number x; \iff int x;`
- `typedef double matrix[5][10];`
- `matrix a; \iff double a[5][10];`

```
typedef unsigned char BYTE;  
// BYTE е синоним на unsigned char  
typedef double REAL;  
// REAL е синоним на double
```

Оператор typedef ...

- Задаването на алтернативно име на тип указател към функция чрез typedef се осъществява по аналогичен начин на дефиниране на променлива от тип указател към функция като новото име на типа заема мястото на променливата.

- *Примери:*

```
typedef double (*mytype) (double) ;
```

определя mytype като синоним на типа double (*)(double);

```
typedef double (*newtype) (double, double) ;
```

определя newtype като синоним на типа double (*)(double, double).

Задача 1.

Да се пресметне сумата $\sin(1) + \sin(2) + \sin(3) + \dots + \sin(n)$.

```
double sum_sin(int n) {  
    double s = 0;  
    for(int i = 1; i <= n; i++)  
        s += sin(i);  
    return s;  
}
```

Задача 2.

Да се пресметне сумата $\cos(1) + \cos(2) + \cos(4) + \dots + \cos(n)$.

```
double sum_cos(int n) {  
    double s = 0;  
    for(int i = 1; i <= n; i *= 2)  
        s += cos(i);  
    return s;  
}
```



```
double sum_sin(int n) {  
    double s = 0;  
    for(int i = 1; i <= n; i++)  
        s += sin(i);  
    return s;  
}
```

```
double sum_cos(int n) {  
    double s = 0;  
    for(int i = 1; i <= n; i *= 2)  
        s += cos(i);  
    return s;  
}
```

Може ли да направим някакъв общ шаблон?

```
double <name>(int n) {  
    double s = 0;  
    for(int i = 1; i <= n; i = <next>(i))  
        s += <f>(i);  
    return s;  
}
```

Функциите като параметри

```
double sum(int n, double (*f)(double), int (*next)(int)) {  
    double s = 0;  
    for(int i = 1; i <= n; i = next(i))  
        s += f(i);  
    return s;  
}
```

```
int plus1(int i) { return i + 1; }
```

```
sum_sin(n)  $\iff$  sum(n, sin, plus1)
```

```
int mult2(int i) { return i * 2; }
```

```
sum_cos(n)  $\iff$  sum(n, cos, mult2)
```

Произведение от по-висок ред

Задача. Да се пресметне произведението $\tan(1)\tan(2)\tan(3)\dots\tan(n)$.

```
double product(int n, double (*f)(double), int (*next)(int)) {  
    double s = 1;  
    for(int i = 1; i <= n; i = next(i))  
        s *= f(i);  
    return s;  
}
```

Решение. `product(n, tan, plus1);`

```
double sum(int n, double (*f)(double), int (*next)(int)) {  
    double s = 0;  
    for(int i = 1; i <= n; i = next(i))  
        s += f(i);  
    return s;  
}
```

```
double product(int n, double (*f)(double), int (*next)(int)) {  
    double s = 1;  
    for(int i = 1; i <= n; i = next(i))  
        s *= f(i);  
    return s;  
}
```

Натрупване от по-висок ред

Да се напише функция, която пресмята натрупването

$$\perp \oplus f(a) \oplus f(next(a)) \oplus f(next(next(a))) \oplus \dots \oplus f(b)$$

където \oplus е двуместна операция,

а \perp е нейната “нулева стойност”, т.е. $x \oplus \perp = x$.

```
typedef int (*nextfun)(int);
typedef double (*mathfun)(double);
typedef double (*mathop)(double, double);

double accumulate (mathop op, double base_value,
                  double a, double b,
                  mathfun f, nextfun next) {
    double s = base_value;
    for(int i = a; i <= b; i = next(i))
        s = op(s, f(i));
    return s;
}
```

Как да го използваме

```
double plus(double a, double b) { return a + b; }
```

```
sum(n, f, next)  $\iff$  accumulate(plus, 0, 1, n, f, next)
```

```
double mult(double a, double b) { return a * b; }
```

```
product(n, f, next)  $\iff$  accumulate(mult, 1, 1, n, f, next)
```


- За подготовката на тази презентация са използвани слайдове на:
 - Доц. Александър Григоров
 - Доц. Трифон Трифонов