

# Функции от по-висок ред

Трифон Трифонов

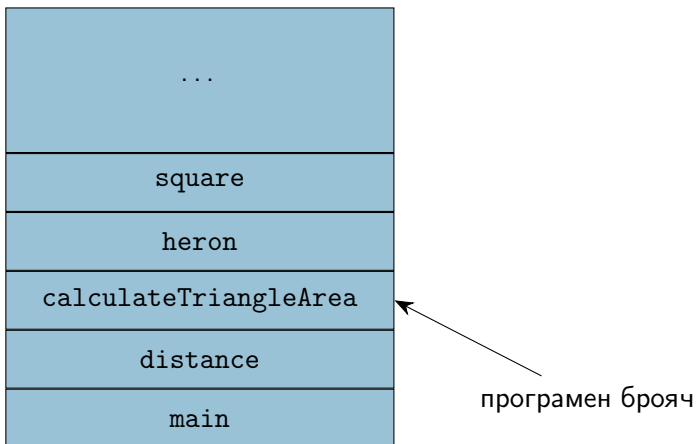
Увод в програмирането,  
спец. Компютърни науки, 1 поток,  
спец. Софтуерно инженерство,  
2016/17 г.

11 януари 2017 г.

# Схема на програмната памет



## Област за програмен код



# Указател към функция

- Кодът на всяка функция на C++ се превежда до машинен код

## Указател към функция

- Кодът на всяка функция на C++ се превежда до машинен код
- Машинният код на функциите е разположен в областта за програмен код

# Указател към функция

- Кодът на всяка функция на C++ се превежда до машинен код
- Машинният код на функциите е разположен в областта за програмен код
- **Адресът на функцията** наричаме адресът на първата инструкция във функцията

# Указател към функция

- Кодът на всяка функция на C++ се превежда до машинен код
- Машинният код на функциите е разположен в областта за програмен код
- **Адресът на функцията** наричаме адресът на първата инструкция във функцията
- Можем да създаваме **указатели към функции**

# Указател към функция

- Кодът на всяка функция на C++ се превежда до машинен код
- Машинният код на функциите е разположен в областта за програмен код
- **Адресът на функцията** наричаме адресът на първата инструкция във функцията
- Можем да създаваме **указатели към функции**
- Името на всяка функция може да се разглежда като константен указател към кода ѝ



# Указател към функция

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

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

## Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

### Примери:

- `void (*f)(int&, int&);`

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`
- `double (*op)(double) = sin;`

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`
- `double (*op)(double) = sin;`
- `op = cos;`

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`
- `double (*op)(double) = sin;`
- `op = cos;`
- `op = NULL;`



# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`
- `double (*op)(double) = sin;`
- `op = cos;`
- `op = NULL;`
- ~~`void (*p)(int, int&) = f;`~~

# Дефиниране на указатели към функции

<тип> (\*<идентификатор>)(<формални параметри>) [= <указател> ];

- имената на параметрите могат да се пропуснат

## Примери:

- `void (*f)(int&, int&);`
- `f = swap;`
- `double (*op)(double) = sin;`
- `op = cos;`
- `op = NULL;`
- ~~`void (*p)(int, int&) = f;`~~
- ~~`sin = op;`~~

# Извикване на функция през указател

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

# Извикване на функция през указател

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

Три еквивалентни начина за извикване на функцията:

- `swap(x, y);`
- `(*f)(x, y);`
- `f(x, y);`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <декларация_на_променлива>;
```

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

**Примери:**

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

**Примери:**

- `typedef int number;`



## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

**Примери:**

- `typedef int number;`
- `number x;  $\iff$  int x;`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x;  $\iff$  int x;`
- `number f(number y) { ... }  $\iff$  int f(int y) { ... }`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x;  $\iff$  int x;`
- `number f(number y) { ... }  $\iff$  int f(int y) { ... }`
- `typedef double matrix[5][10];`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

**Примери:**

- `typedef int number;`
- `number x;  $\iff$  int x;`
- `number f(number y) { ... }  $\iff$  int f(int y) { ... }`
- `typedef double matrix[5][10];`
- `matrix a;  $\iff$  double a[5][10];`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x;`  $\iff$  `int x;`
- `number f(number y) { ... }`  $\iff$  `int f(int y) { ... }`
- `typedef double matrix[5][10];`
- `matrix a;`  $\iff$  `double a[5][10];`
- `typedef int** pointer2;`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x;  $\iff$  int x;`
- `number f(number y) { ... }  $\iff$  int f(int y) { ... }`
- `typedef double matrix[5][10];`
- `matrix a;  $\iff$  double a[5][10];`
- `typedef int** pointer2;`
- `int x; int* p = &x; pointer2 q = &p;`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x;`  $\iff$  `int x;`
- `number f(number y) { ... }`  $\iff$  `int f(int y) { ... }`
- `typedef double matrix[5][10];`
- `matrix a;`  $\iff$  `double a[5][10];`
- `typedef int** pointer2;`
- `int x; int* p = &x; pointer2 q = &p;`
- `typedef int& ref;`

## Дефиниране на потребителски типове

В C++ е позволено дефиниране на потребителски типове:

```
typedef <тип> <име>;
```

- създава се нов тип <име>, който е еквивалентен на <тип>

Примери:

- `typedef int number;`
- `number x; ⇔ int x;`
- `number f(number y) { ... } ⇔ int f(int y) { ... }`
- `typedef double matrix[5][10];`
- `matrix a; ⇔ double a[5][10];`
- `typedef int** pointer2;`
- `int x; int* p = &x; pointer2 q = &p;`
- `typedef int& ref;`
- `ref y = x;`



# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`

# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`
- `mathfun p = exp; p = log;`

# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`
- `mathfun p = exp; p = log;`
- `typedef void (*procedure)();`

# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`
- `mathfun p = exp; p = log;`
- `typedef void (*procedure)();`
- `void h() { cout << "h()\n"; }`

# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`
- `mathfun p = exp; p = log;`
- `typedef void (*procedure)();`
- `void h() { cout << "h()\n"; }`
- `procedure q = h; q();`

# Потребителски типове за указатели към функции

- `typedef double (*mathfun)(double);`
- `mathfun p = exp; p = log;`
- `typedef void (*procedure)();`
- `void h() { cout << "h()\n"; }`
- `procedure q = h; q();`
- `void (*r)() = q;`

# Примерна сума 1

## Задача 1.

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

# Примерна сума 1

## Задача 1.

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

## Решение:

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



## Примерна сума 2

### Задача 2.

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

## Примерна сума 2

### Задача 2.

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

### Решение:

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

## Открийте разликите!

```
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;  
}
```

## Открийте разликите!

```
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;  
}
```

# Общият 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;  
}
```

## 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; }`

## 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)`



## 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; }`

## 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

```
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;  
}
```

## Произведение от по-висок ред

```
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;  
}
```

### Примери:

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

## Произведение от по-висок ред

```
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;  
}
```

### Примери:

- **Задача.** Да се пресметне произведението  $\tan(1)\tan(2)\tan(3)\dots\tan(n)$ .
- **Решение.** `product(n, tan, plus1);`

## Открийте разликите 2.0

```
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;  
}
```

## Открийте разликите 2.0

```
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 (accumulate)

Да се напише функция, която пресмята натрупването

$$\perp \oplus f(a) \oplus f(\text{next}(a)) \oplus f(\text{next}(\text{next}(a))) \oplus \dots \oplus f(b)$$

където  $\oplus$  е двуместна операция,

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



## Натрупване от по-висок ред (accumulate)

Да се напише функция, която пресмята натрупването

$$\perp \oplus f(a) \oplus f(\text{next}(a)) \oplus f(\text{next}(\text{next}(a))) \oplus \dots \oplus f(b)$$

където  $\oplus$  е двуместна операция,

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

Решение:

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

## Натрупване от по-висок ред (accumulate)

Да се напише функция, която пресмята натрупването

$$\perp \oplus f(a) \oplus f(\text{next}(a)) \oplus f(\text{next}(\text{next}(a))) \oplus \dots \oplus f(b)$$

където  $\oplus$  е двуместна операция,

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

Решение:

- `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);
```

## Натрупване от по-висок ред (accumulate)

```
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;  
}
```

## Натрупване от по-висок ред (accumulate)

```
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;
}
```

### Примери:

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

## Натрупване от по-висок ред (accumulate)

```
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;
}
```

### Примери:

- `double plus(double a, double b) { return a + b; }`
- `sum(n, f, next)  $\iff$  accumulate(plus, 0, 1, n, f, next)`

## Натрупване от по-висок ред (accumulate)

```
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;
}
```

### Примери:

- `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; }`

## Натрупване от по-висок ред (accumulate)

```
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;
}
```

### Примери:

- `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)`

## Задачи за accumulate

С помощта на `accumulate` да се пресметнат:

- $n!$



## Задачи за accumulate

С помощта на accumulate да се пресметнат:

- $n!$
- $x^n$

# Задачи за accumulate

С помощта на accumulate да се пресметнат:

- $n!$
- $x^n$
- $\sum_{i=0}^n \frac{x^i}{i!}$

# Задачи за accumulate

С помощта на accumulate да се пресметнат:

- $n!$
- $x^n$
- $\sum_{i=0}^n \frac{x^i}{i!}$
- $\binom{n}{k}$

# Задачи за accumulate

С помощта на accumulate да се пресметнат:

- $n!$

- $x^n$

- $\sum_{i=0}^n \frac{x^i}{i!}$

- $\binom{n}{k}$

- $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$