

Функции от по-висок ред

Трифон Трифонов

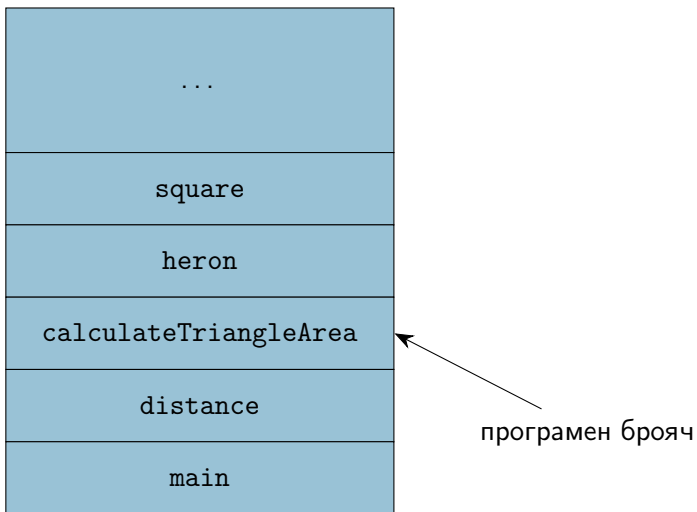
Обектно-ориентирано програмиране,
спец. Компютърни науки, 1 поток,
2018/19 г.

10 април 2019 г.

Схема на програмната памет



Област 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
- 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>];

- имената на параметрите могат да се пропуснат

Примери:

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

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:

```
using <име> = <тип>;
```

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

Примери:

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

Потребителски типове за указатели към функции

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

Примерна сума 1

Задача 1.

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

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 $\cos(1) + \cos(2) + \cos(4) + \dots + \cos(n)$.

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;  
}
```

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;
}
```

- `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;
}
```

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;
}
```

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:

- `using nextfun = int (*)(int);`
- `using mathfun = double (*)(double);`
- `using mathop = double (*)(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;
}
```

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:

1 $n!$

2 x^n

3 $\sum_{i=0}^n \frac{x^i}{i!}$

4 $\binom{n}{k}$

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

Анонимни функции в C++14

- `[](<параметри>) -> <тип> {<тяло>}`
- създава анонимна (λ) функция, дефинирана като:
`<тип> λ (<параметри>) {<тяло>}`
- ако `<тяло>` е от вида `return <израз>`; можем да пропуснем `<тип>`:
- `[](<параметри>) {<тяло>}`
- **Примери:**
- `sum(n, sin, [](int n) { return n + 1; })`
- `accumulate([](double a, double b) { return a * b; }, 1, 1, n, [](double x) { return x; }, [](int n) { return n + 1; })`

Функциите като върнат резултат

Задача. Да се напише функция, която по зададен числов код 1, 2, 3 или 4, връща съответно една от функциите \sin , \cos , e^x , \log .

Решение: `double (*choose_function(double))(int n);`

```
mathfun choose_function(int n) {  
    switch(n) {  
        case 1 : return sin;  
        case 2 : return cos;  
        case 3 : return exp;  
        case 4 : return log;  
        default : return nullptr;  
    }  
}
```

Производна

Задача. Да се напише функция, която по дадена едноаргументна функция f връща нейната производна.

Преговор:

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Решение.

```
matfun derive(matfun f) {  
    return ?  
}
```

Проблем: За различни f , трябва да връщаме различна функция, но предварително не знаем коя!

Намиране на производна

Идея №1. Ще използваме помощна функция.

```
const double EPS = 1E-10;
double derivative(double x) {
    return (function(x + EPS) - function(x)) / EPS;
}
```

Проблем №2: Как да подадем function?

Идея №3: Да използваме глобална променлива.

```
matfun function = nullptr;
matfun derive(matfun f) {
    function = f;
    return derivative;
}
```

Проблем №3. Грозно е.

Проблем №4. Работи само с една производна в даден момент.

Намиране на производна с анонимни функции

Идея №4: Да използваме анонимни функции!

```
auto derive(mathfun f) {
    return [f](double x) {
        return (f(x + EPS) - f(x)) / EPS;
    };
}
```

- `[f]` означава, че позволяваме на анонимната функция да използва копие на указателя `f`.
- `auto` означава, че искаме C++14 сам да се сети за типа на връщания резултат (**`mathfun` вече не върши работа**)

Примери:

- `auto mycos = derive(sin);`
- `cout << mycos(0) << ' ' << cos(0);`
- `cout << exp(1) << ' ' << derive(exp)(1);`