



УВОД В ПРОГРАМИРАНЕТО – УПРАЖНЕНИЕ №14

10.01.2023

СОФТУЕРНО ИНЖЕНЕРСТВО, ГРУПА 6

АСИСТЕНТ: ЕЛЕНА ТУПАРОВА

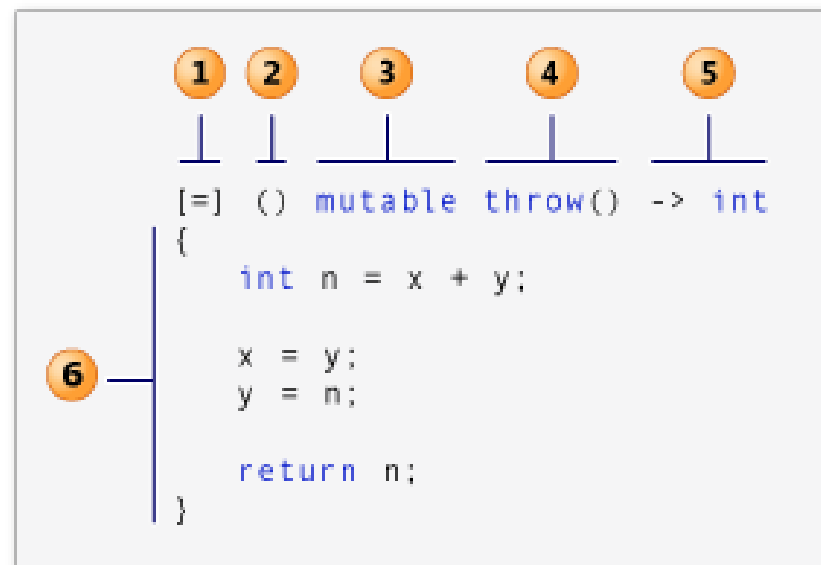


ЗА КАКВО ЩЕ СИ ГОВОРИМ ДНЕС?

- От миналия път - ламбда функции
- Header файлове
- Преговор

ЛАМБДА ФУНКЦИИ

- Т.нар. анонимни функции – дефинират се еднократно на мястото, на което се използват (извикват или подават като параметър на функция)



The diagram shows a C++ lambda function definition with six numbered annotations (1-6) pointing to specific parts of the code:

- 1 points to the opening square bracket `[=]`.
- 2 points to the opening parenthesis `()`.
- 3 points to the `mutable` keyword.
- 4 points to the `throw()` exception specifier.
- 5 points to the return type `-> int`.
- 6 points to the opening curly brace `{` of the function body.

```
[=] () mutable throw() -> int
{
    int n = x + y;

    x = y;
    y = n;

    return n;
}
```

КАК ДА ПИШЕМ „ХУБАВ“ КОД?

- Използвайте смислени имена на променливи и функции!
- НЕ пишете на шльокавица!
- НЕ използвайте магически числа – заменете ги с константи!
- Една функция трябва да прави (по възможност) едно нещо. Ако функцията ви прави повече неща, разделете я на подфункции! (принцип „разделяй и владей“)
- Пишете устойчиви функции – винаги проверявайте дали параметрите ви са валидни за конкретната задача, която решавате!
- НЕ пишете основната логика на задачата в `main()` – там трябва да са само входът на данните, валидацията им (идеално и тя е в отделна функция) и функция, която съдържа логиката на задачата (и която най-вероятно извиква други функции)!

УКАЗАТЕЛИ И ПСЕВДОНИМИ (РЕФЕРЕНЦИИ)

```
int number = 5;
```


```
int& referenceToNumber = number; // друго име на същата променлива  
std::cout << referenceToNumber << std::endl;
```

```
int* pointerToNumber = &number; // пази адреса на променливата number  
std::cout << pointerToNumber << std::endl;
```

```
std::cout << *pointerToNumber << std::endl;
```

УКАЗАТЕЛИ КЪМ КОНСТАНТА != КОНСТАНТНИ УКАЗАТЕЛИ

const int* != int* const



Указател към константа

Сочи към константа, т.е. не може да се променя стойността ѝ
Алтернативен запис - `int const*`

Константен указател

Веднъж свързан с дадена променлива, не може да се „пренасочва“ към друга променлива (като псевдоним)
Може да се променя стойността на самата променлива

НАЧИНИ ЗА ПРЕДАВАНЕ НА ПАРАМЕТРИ НА ФУНКЦИЯ

- По стойност
- Чрез указател
- Чрез псевдоним

УКАЗАТЕЛНА АРИТМЕТИКА

```
int a = 4;
```

```
int *p = &a;
```

```
p = p + 1; // ще „премести“ напред указателя с една  
„стъпка“, голяма колкото е размерът на типа, към който сочи  
указателят
```


УКАЗАТЕЛИ И МАСИВИ

- Указатели и едномерни масиви

```
int a[100]; // a -> указател към a[0]
```

```
*a == a[0];
```

```
*(a+1) == a[1]; ... *(a+n) == a[n];
```

- Указатели и двумерни масиви

```
int a[10][20]; // a -> указател към първия елемент на едномерния  
масив [a[0], a[1], ... a[9]], а всяко a[i] е указател към a[i][0]
```

```
**a == a[0][0]; *a == a[0];
```

```
a[i][j] == (*(a+i)+j);
```

КАКВО Е СИМВОЛЕН НИЗ И КАК ГО ПРЕДСТАВЯМЕ В C++?

- Редица от краен брой символи, заградена в кавички
- Масив от символи (char), в който след последния символ в низа е записан т.нар. терминиращ символ '\0'

```
char str[] = "FMI";
```

'F'	'M'	'I'	'\0'
-----	-----	-----	------

КАКВО ТРЯБВА ДА ЗАПОМНИМ ЗА СИМВОЛНИТЕ НИЗОВЕ?

- Винаги завършват с **терминиращ символ**, т.е. масив от тип `char` с размер `size` може да съдържа низ с максимална дължина (`size - 1`).
- Реално това са масиви => **не можем** да ги сравняваме с оператора „==“ или да ги копираме с оператора „=“.
- **Можем** да въвеждаме и извеждаме символни низове директно чрез `cin` и `cout`. Въвеждането със `cin` обаче се терминира при въвеждане на `whitespace` символ => в такъв случай ползваме `cin.getline()`.

УКАЗАТЕЛИ И НИЗОВЕ

- Връзката е като при едномерните масиви;
- Обхождане на низ с помощта на указатели:

```
char str[] = "Hello!";  
char* pstr = str;  
while (*pstr) {  
    cout << *pstr;  
    pstr++;  
}
```

ВИДОВЕ ПАМЕТ

Stack vs. Heap

- Заделя се при компилация
 - Освобождава се автоматично, когато се излезе от scope-а, в която е била дефинирана
- Заделя се по време на изпълнението на програмата
 - НЕ се освобождава се автоматично, трябва програмистът сам да се погрижи за това

КАК ЗАДЕЛЯМЕ И ОСВОБОЖДАВАМЕ ДИНАМИЧНА ПАМЕТ?

- Заделяме с `new` или `new[]`
- Освобождаваме с `delete` или `delete[]`
- **ВИНАГИ** трябва да освободим заделената с оператора `new` памет

КАК ФУНКЦИЯ ДА ВЪРНЕ МАСИВ?

- Масивът се връща като тип указател към съответния тип на елементите
- Важно е този указател да НЕ се унищожи след края на изпълнението на функцията
- За целта можем да използваме динамично създаден масив, който се намира в областта на *heap*-а и ще бъде изтрит от нея, когато ние експлицитно го направим

FUNCTION POINTERS – УКАЗАТЕЛИ КЪМ ФУНКЦИИ

- Указателят към функция има за стойност адреса на изпълнимия код на функцията
- Указателите към функции могат да се използват, за да се извикват функции и да се подават функции като параметри на други функции (функции от по-висок ред)
- НЕ може да се извършва указателна аритметика върху указатели към функции

ЗАДАЧА 1

- Да се напише **рекурсивна** функция, която намира сумата на елементите на масив от цели числа.