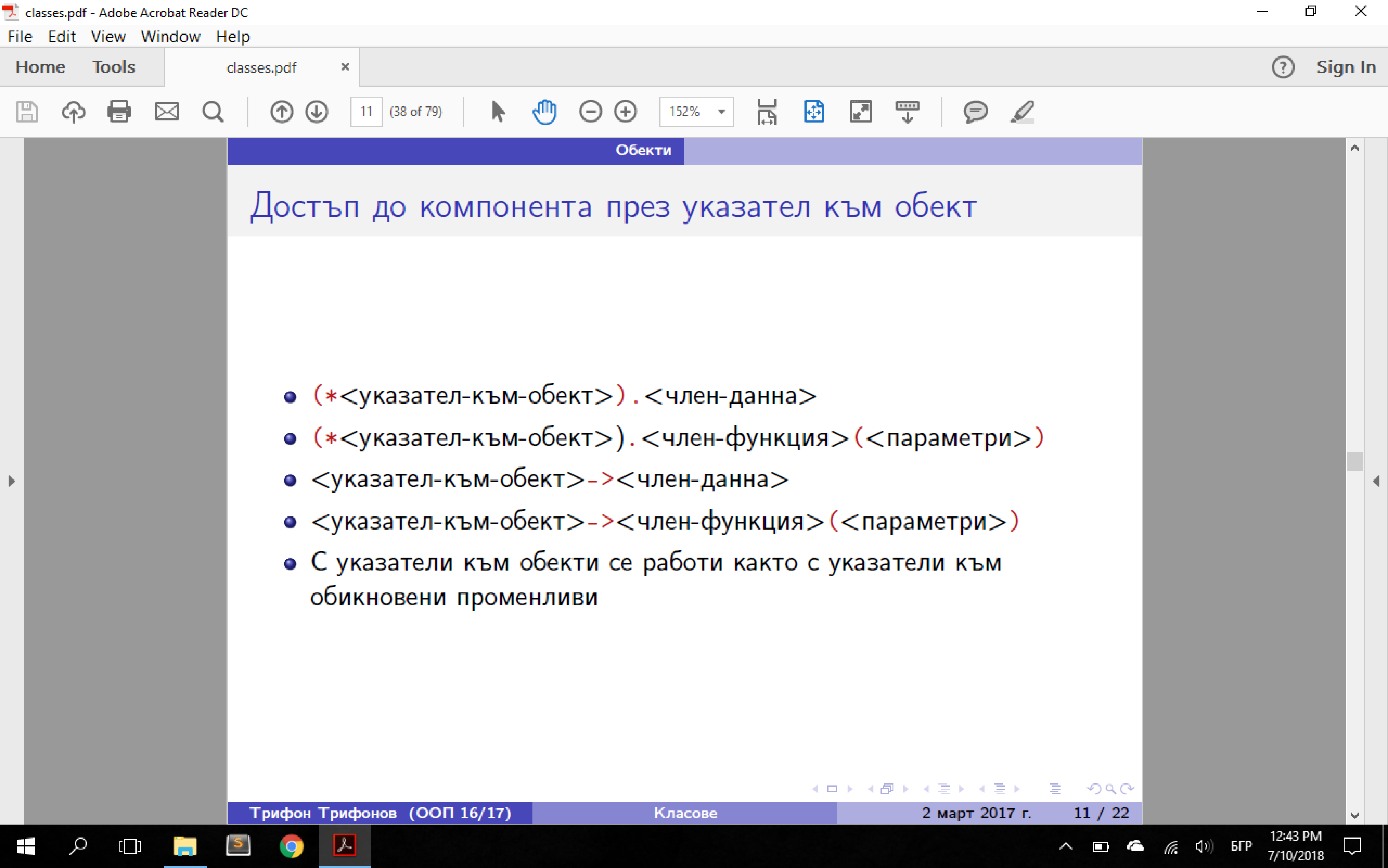
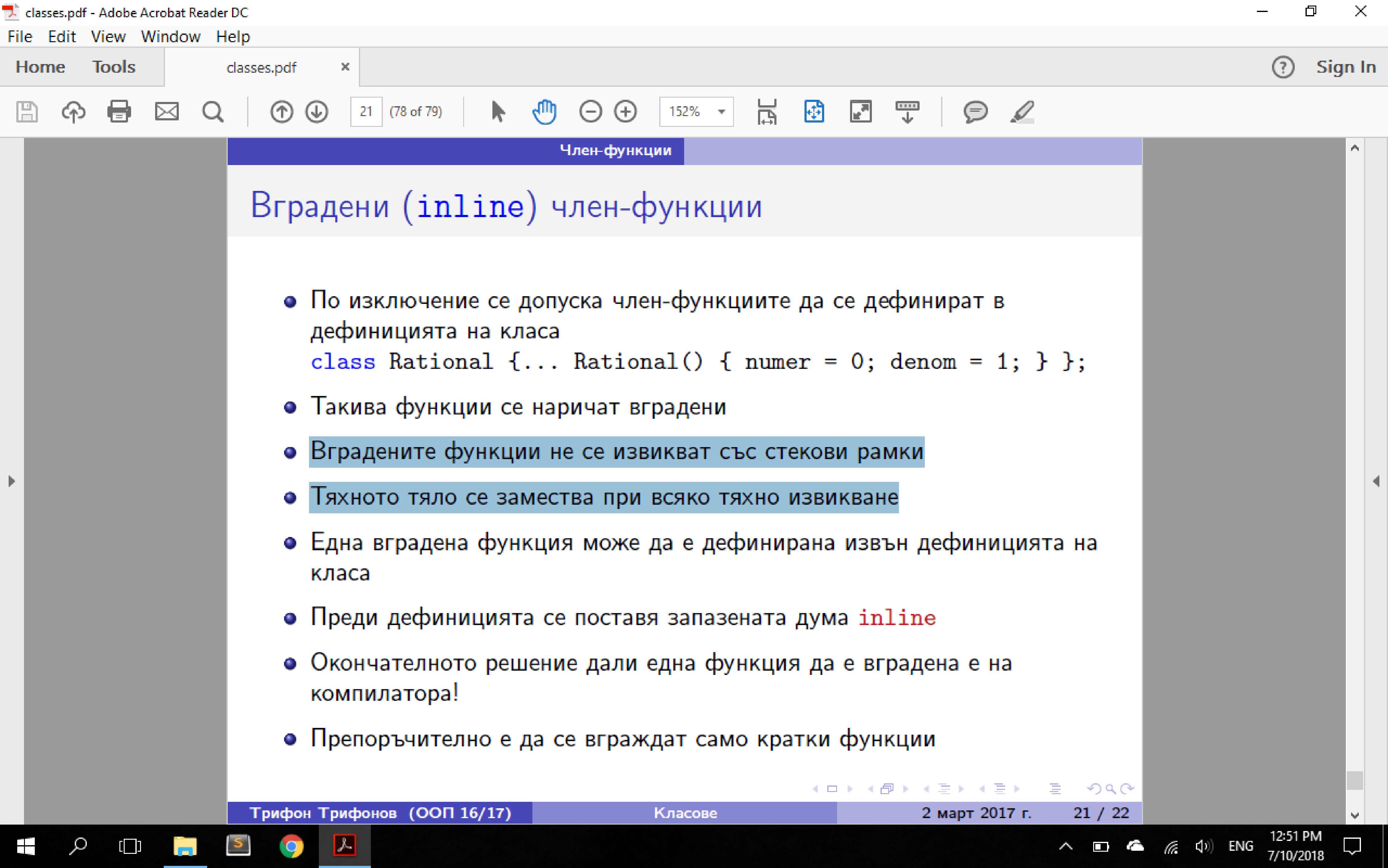


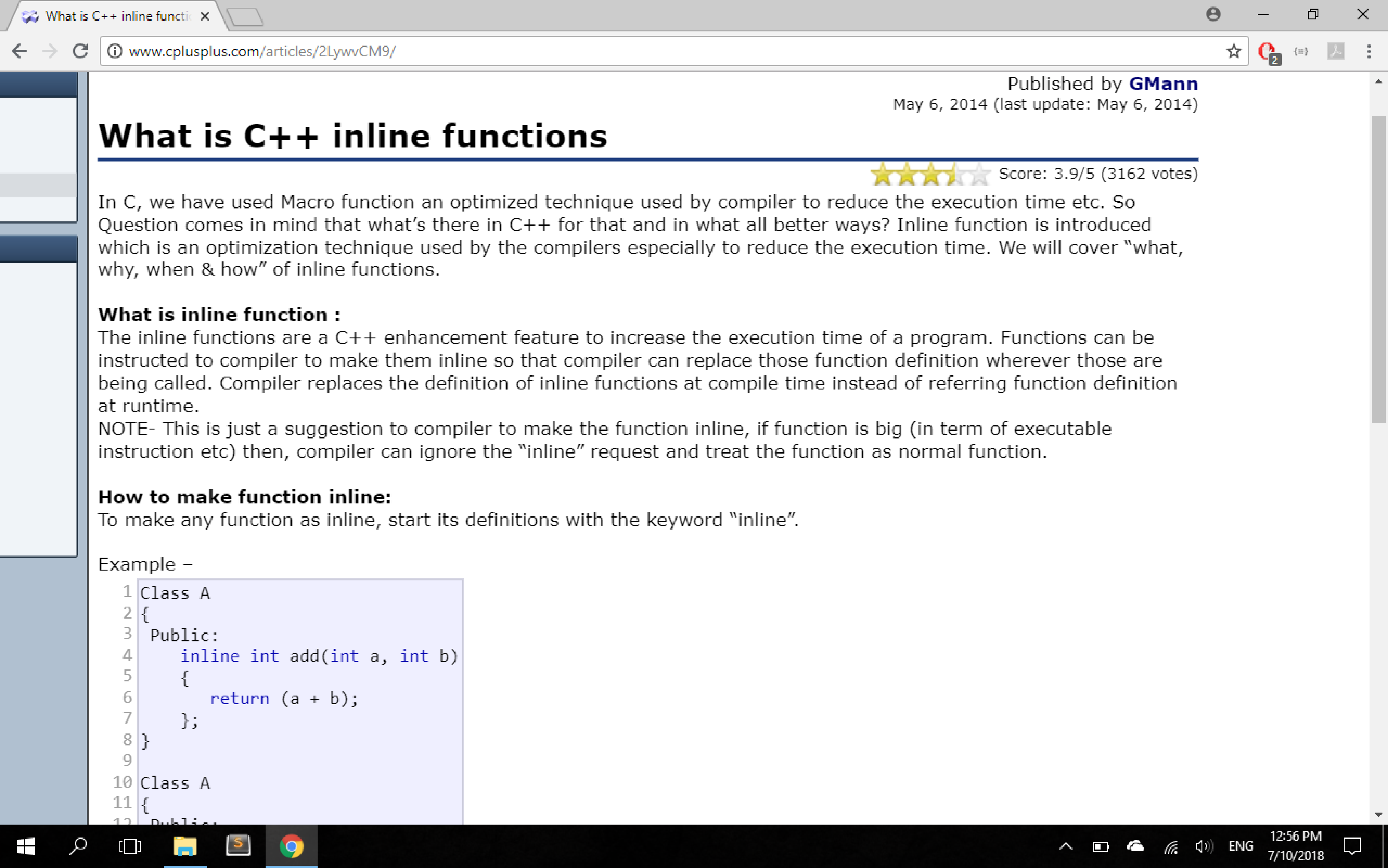
Достъп до компонента през указател към обект

- (*<указател-към-обект>).<член-данна>
- (*<указател-към-обект>).<член-функция>(<параметри>)
- <указател-към-обект>-><член-данна>
- <указател-към-обект>-><член-функция>(<параметри>)
- С указатели към обекти се работи както с указатели към обикновени променливи



Вградени (inline) член-функции

- По изключение се допуска член-функциите да се дефинират в дефиницията на класа
  `class Rational {... Rational() { numer = 0; denom = 1; } };`
- Такива функции се наричат вградени
- Вградените функции не се извикват със стекови рамки
- Тяхното тяло се замества при всяко тяхно извикване
- Една вградена функция може да е дефинирана извън дефиницията на класа
- Преди дефиницията се поставя запазената дума `inline`
- Окончателното решение дали една функция да е вградена е на компилатора!
- Препоръчително е да се вграждат само кратки функции


What is C++ inline functi
www.cplusplus.com/articles/2LywvCM9/
ENG
12:56 PM
7/10/2018

Published by **GMann**
May 6, 2014 (last update: May 6, 2014)

What is C++ inline functions

Score: 3.9/5 (3162 votes)

In C, we have used Macro function an optimized technique used by compiler to reduce the execution time etc. So Question comes in mind that what's there in C++ for that and in what all better ways? Inline function is introduced which is an optimization technique used by the compilers especially to reduce the execution time. We will cover "what, why, when & how" of inline functions.

What is inline function :
The inline functions are a C++ enhancement feature to increase the execution time of a program. Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. Compiler replaces the definition of inline functions at compile time instead of referring function definition at runtime.
NOTE- This is just a suggestion to compiler to make the function inline, if function is big (in term of executable instruction etc) then, compiler can ignore the "inline" request and treat the function as normal function.

How to make function inline:
To make any function as inline, start its definitions with the keyword "inline".

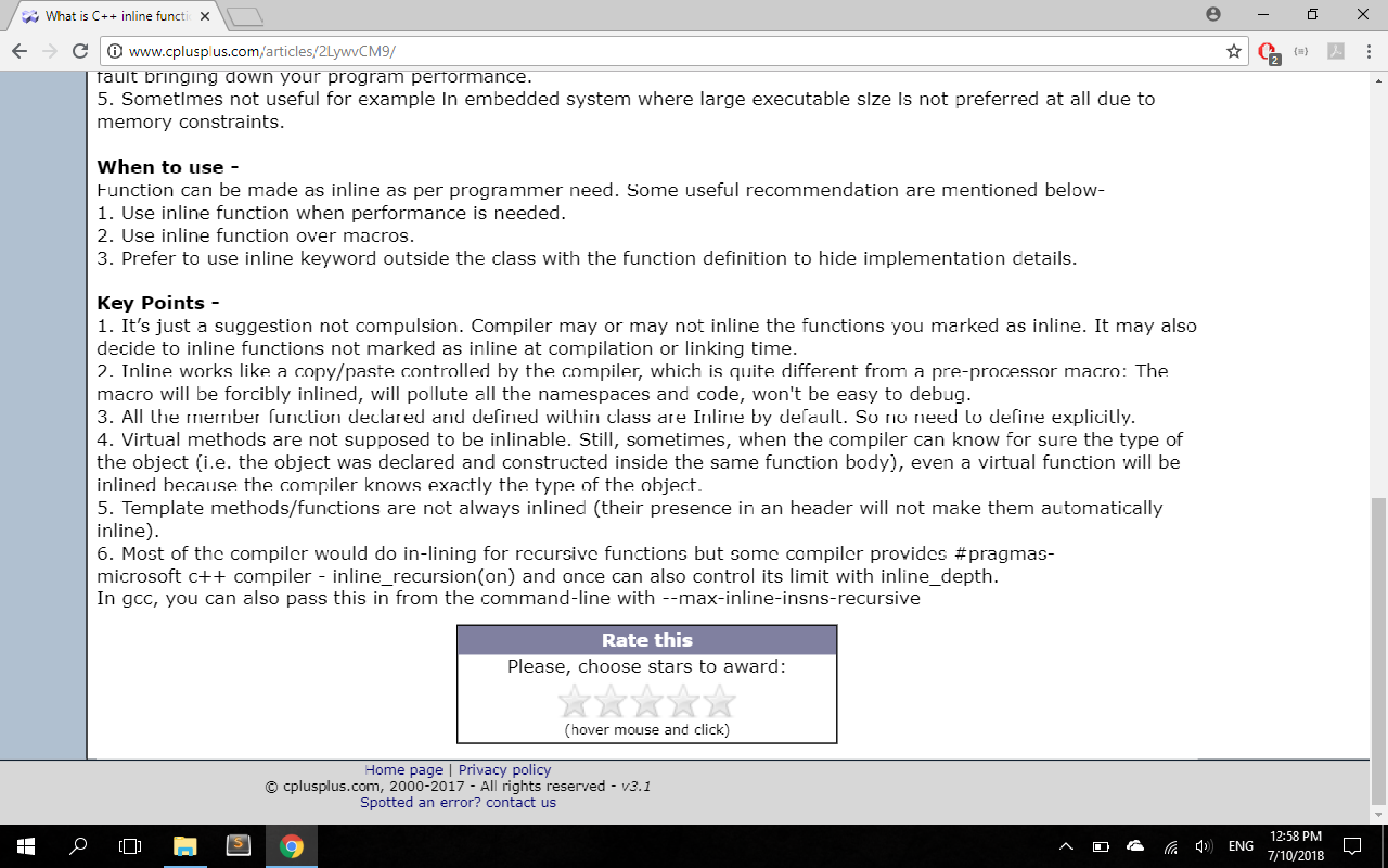
Example –

```
Class A
{
 Public:
    inline int add(int a, int b)
    {
       return (a + b);
    };
}

Class A
{
```

Why to use –
In many places we create the functions for small work/functionality which contain simple and less number of executable instruction. Imagine their calling overhead each time they are being called by callers.
When a normal function call instruction is encountered, the program stores the memory address of the instructions immediately following the function call statement, loads the function being called into the memory, copies argument values, jumps to the memory location of the called function, executes the function codes, stores the return value of the function, and then jumps back to the address of the instruction that was saved just before executing the called function. Too much run time overhead.
The C++ inline function provides an alternative. With inline keyword, the compiler replaces the function call statement with the function code itself (process called expansion) and then compiles the entire code. Thus, with inline functions, the compiler does not have to jump to another location to execute the function, and then jump back as the code of the called function is already available to the calling program.
With below pros, cons and performance analysis, you will be able to understand the "why" for inline keyword
Pros :-
1. It speeds up your program by avoiding function calling overhead.
2. It save overhead of variables push/pop on the stack, when function calling happens.
3. It save overhead of return call from a function.
4. It increases locality of reference by utilizing instruction cache.
5. By marking it as inline, you can put a function definition in a header file (i.e. it can be included in multiple compilation unit, without the linker complaining)

Cons :-
1. It increases the executable size due to code expansion.
2. C++ inlining is resolved at compile time. Which means if you change the code of the inlined function, you would need to recompile all the code using it to make sure it will be updated
3. When used in a header, it makes your header file larger with information which users don't care.
4. As mentioned above it increases the executable size, which may cause thrashing in memory. More number of page fault bringing down your program performance.
5. Sometimes not useful for example in embedded system where large executable size is not preferred at all due to memory constraints.

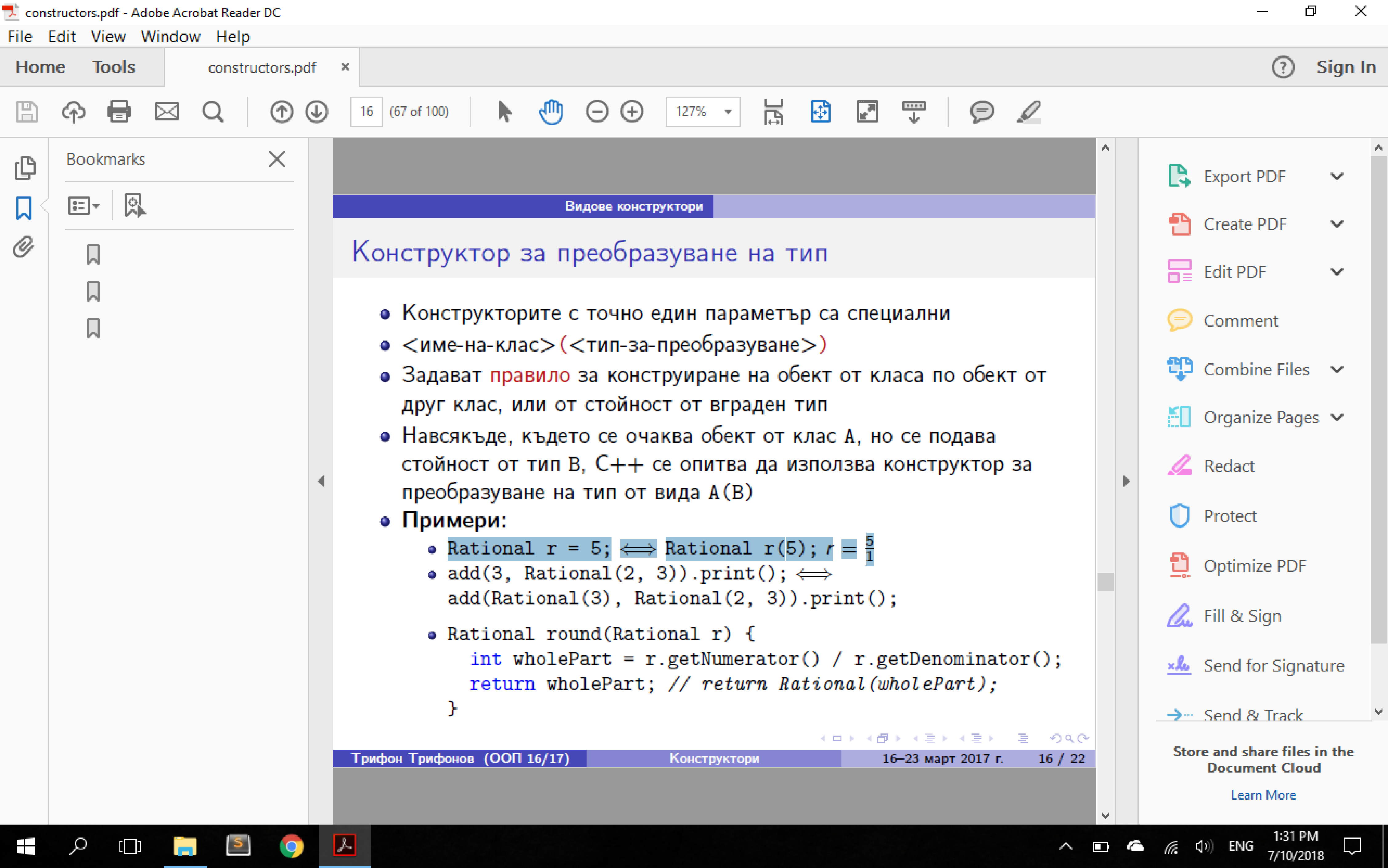

What is C++ inline functi
www.cplusplus.com/articles/2LywvCM9/
ENG
12:58 PM
7/10/2018

fault bringing down your program performance.
5. Sometimes not useful for example in embedded system where large executable size is not preferred at all due to memory constraints.

When to use -
Function can be made as inline as per programmer need. Some useful recommendation are mentioned below-
1. Use inline function when performance is needed.
2. Use inline function over macros.
3. Prefer to use inline keyword outside the class with the function definition to hide implementation details.

Key Points -
1. It's just a suggestion not compulsion. Compiler may or may not inline the functions you marked as inline. It may also decide to inline functions not marked as inline at compilation or linking time.
2. Inline works like a copy/paste controlled by the compiler, which is quite different from a pre-processor macro: The macro will be forcibly inlined, will pollute all the namespaces and code, won't be easy to debug.
3. All the member function declared and defined within class are Inline by default. So no need to define explicitly.
4. Virtual methods are not supposed to be inlinable. Still, sometimes, when the compiler can know for sure the type of the object (i.e. the object was declared and constructed inside the same function body), even a virtual function will be inlined because the compiler knows exactly the type of the object.
5. Template methods/functions are not always inlined (their presence in an header will not make them automatically inline).
6. Most of the compiler would do in-lining for recursive functions but some compiler provides #pragmas-
microsoft c++ compiler - inline_recursion(on) and once can also control its limit with inline_depth.
In gcc, you can also pass this in from the command-line with --max-inline-insns-recursive

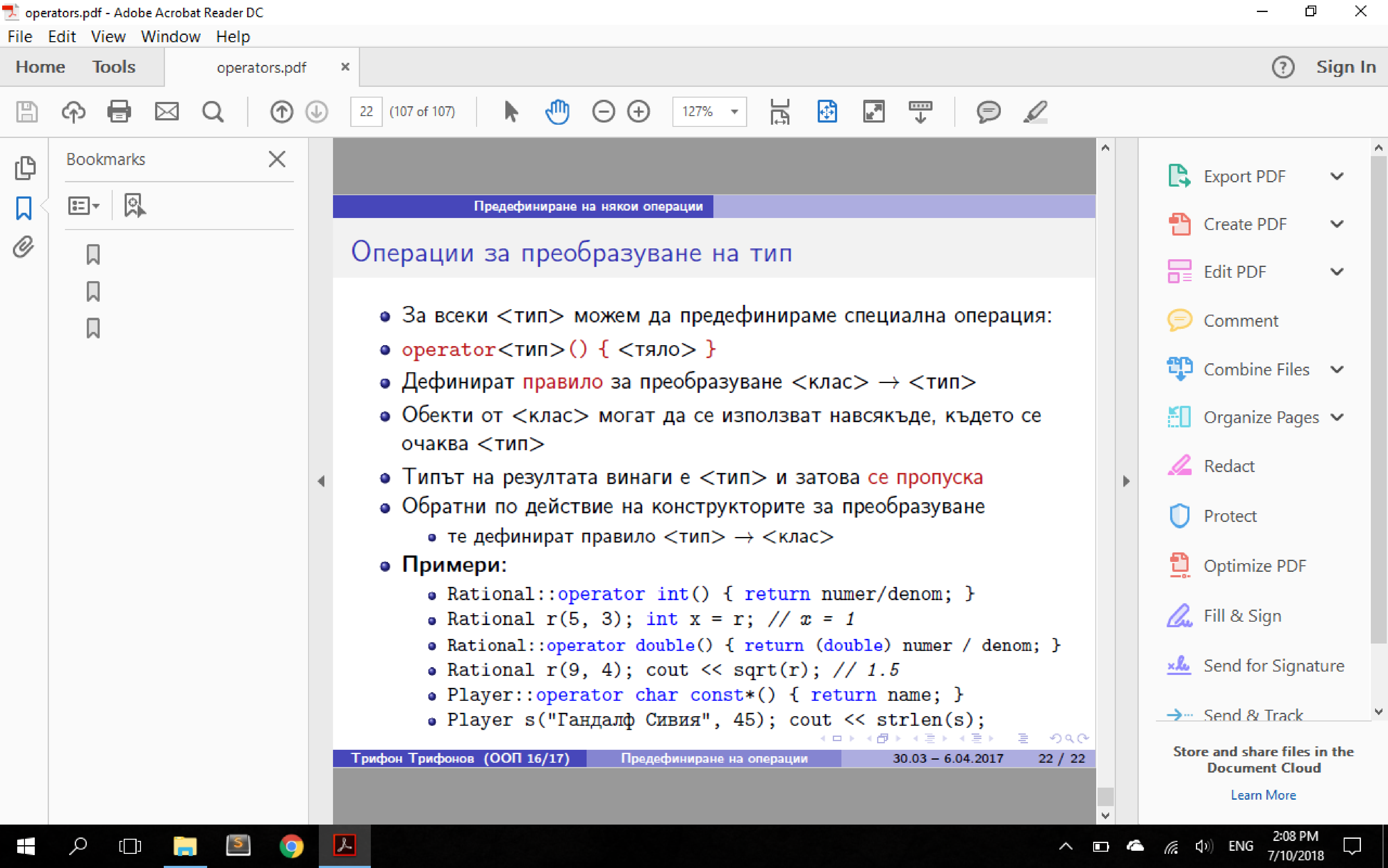
Rate this
Please, choose stars to award:
(hover mouse and click)



Конструктор за преобразуване на тип

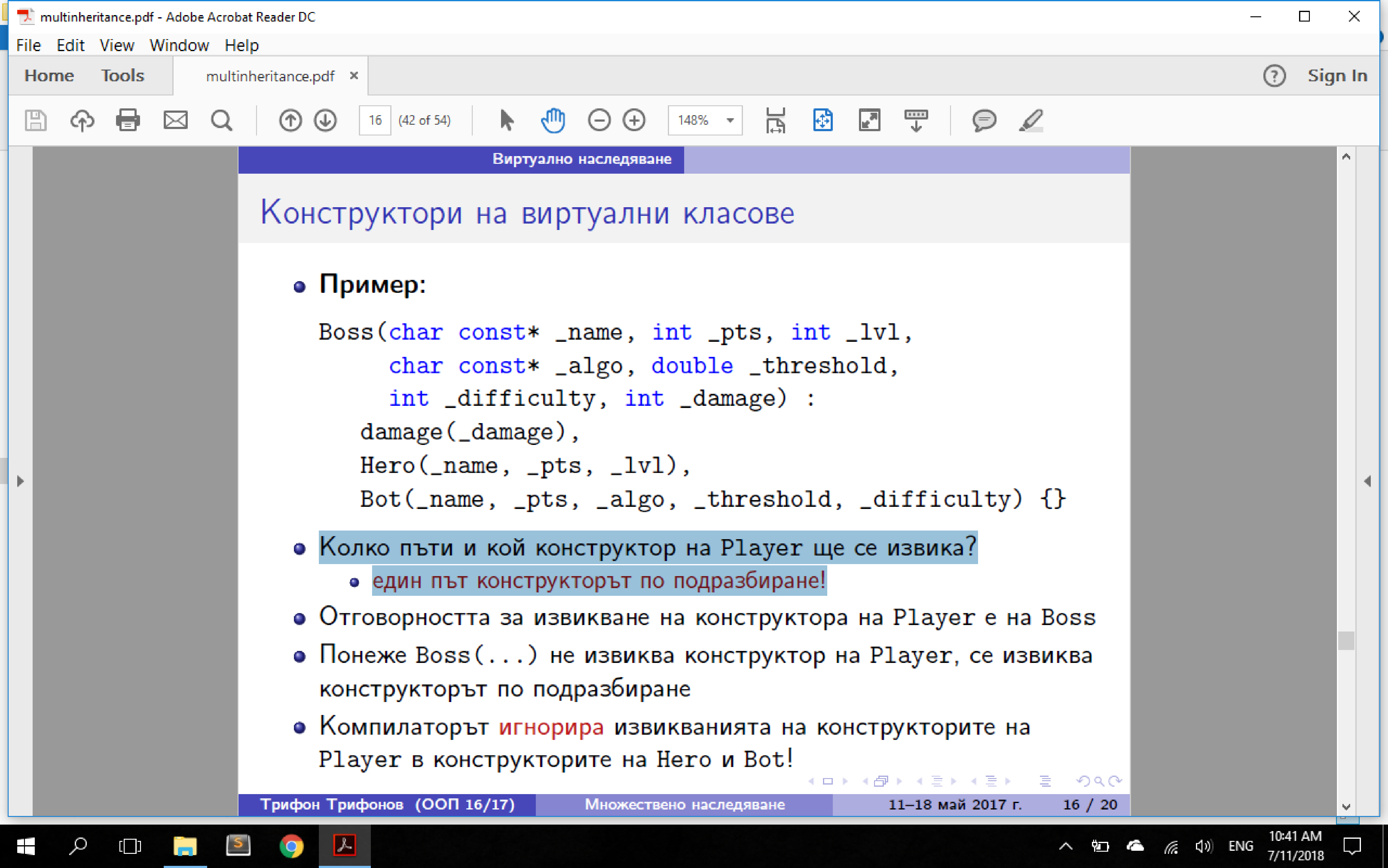
- Конструкторите с точно един параметър са специални
- <име-на-клас>(<тип-за-преобразуване>)
- Задават правило за конструиране на обект от класа по обект от друг клас, или от стойност от вграден тип
- Навсякъде, където се очаква обект от клас A, но се подава стойност от тип B, C++ се опитва да използва конструктор за преобразуване на тип от вида A(B)
- **Примери:**
  - `Rational r = 5;` $\Longleftrightarrow$ `Rational r(5);` $r = \frac{5}{1}$
  - `add(3, Rational(2, 3)).print();` $\Longleftrightarrow$ `add(Rational(3), Rational(2, 3)).print();`

  -
    ```
    Rational round(Rational r) {
      int wholePart = r.getNumerator() / r.getDenominator();
      return wholePart; // return Rational(wholePart);
    }
    ```



Операции за преобразуване на тип

- За всеки <тип> можем да предефинираме специална операция:
- `operator<тип>() { <тяло> }`
- Дефинират правило за преобразуване <клас> → <тип>
- Обекти от <клас> могат да се използват навсякъде, където се очаква <тип>
- Типът на резултата винаги е <тип> и затова се пропуска
- Обратни по действие на конструкторите за преобразуване
  - те дефинират правило <тип> → <клас>
- **Примери:**
  - `Rational::operator int() { return numer/denom; }`
  - `Rational r(5, 3); int x = r; // x = 1`
  - `Rational::operator double() { return (double) numer / denom; }`
  - `Rational r(9, 4); cout << sqrt(r); // 1.5`
  - `Player::operator char const*() { return name; }`
  - `Player s("Гандалф Сивия", 45); cout << strlen(s);`

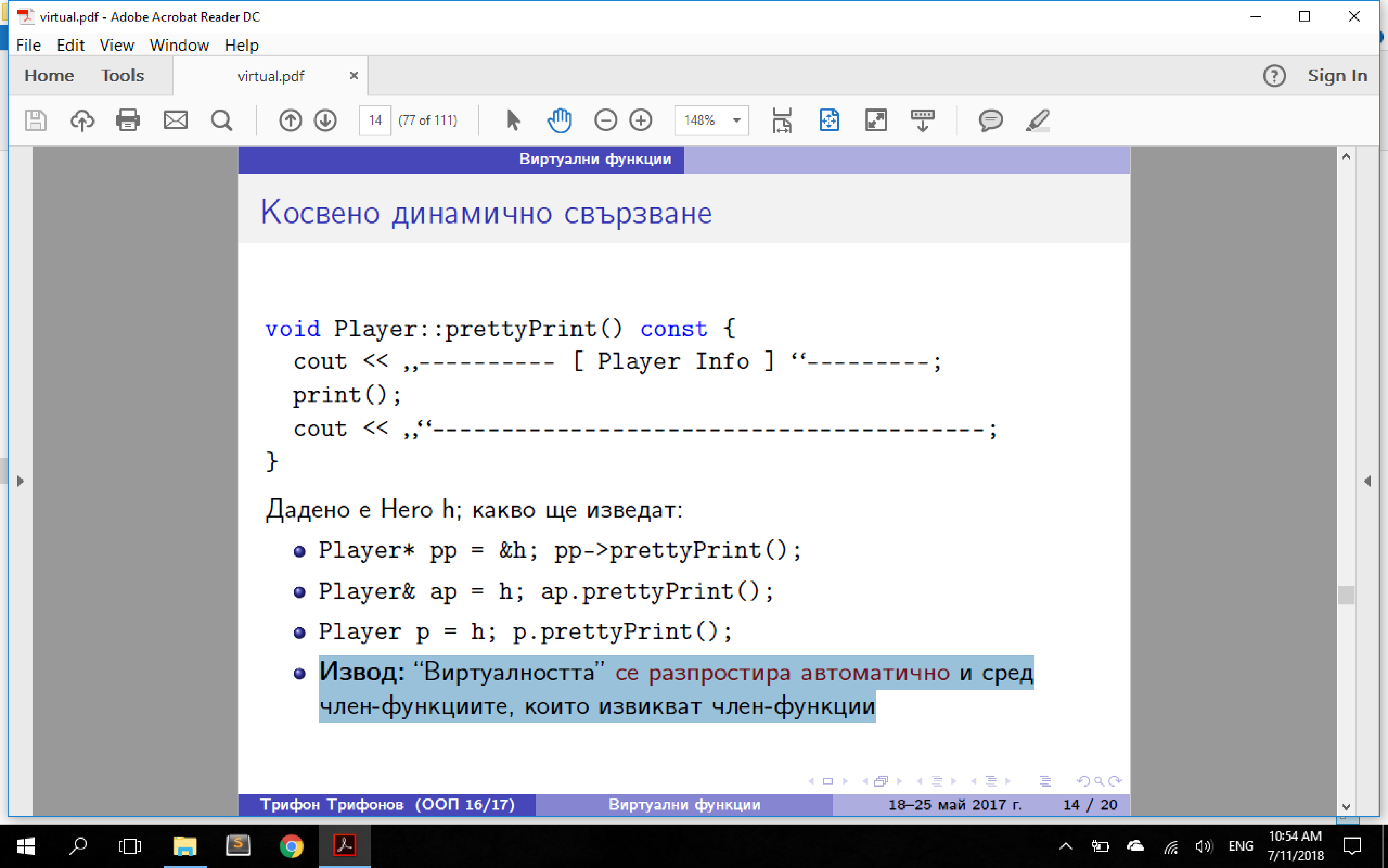

multinheritance.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools multinheritance.pdf
Sign In
16 (42 of 54)
148%
ENG
10:41 AM
7/11/2018

Конструктори на виртуални класове

- **Пример:**

```
Boss(char const* _name, int _pts, int _lvl,
     char const* _algo, double _threshold,
     int _difficulty, int _damage) :
   damage(_damage),
   Hero(_name, _pts, _lvl),
   Bot(_name, _pts, _algo, _threshold, _difficulty) {}
```

- Колко пъти и кой конструктор на Player ще се извика?
  - един път конструкторът по подразбиране!
- Отговорността за извикване на конструктора на Player е на Boss
- Понеже Boss(...) не извиква конструктор на Player, се извиква конструкторът по подразбиране
- Компилаторът игнорира извикванията на конструкторите на Player в конструкторите на Hero и Bot!

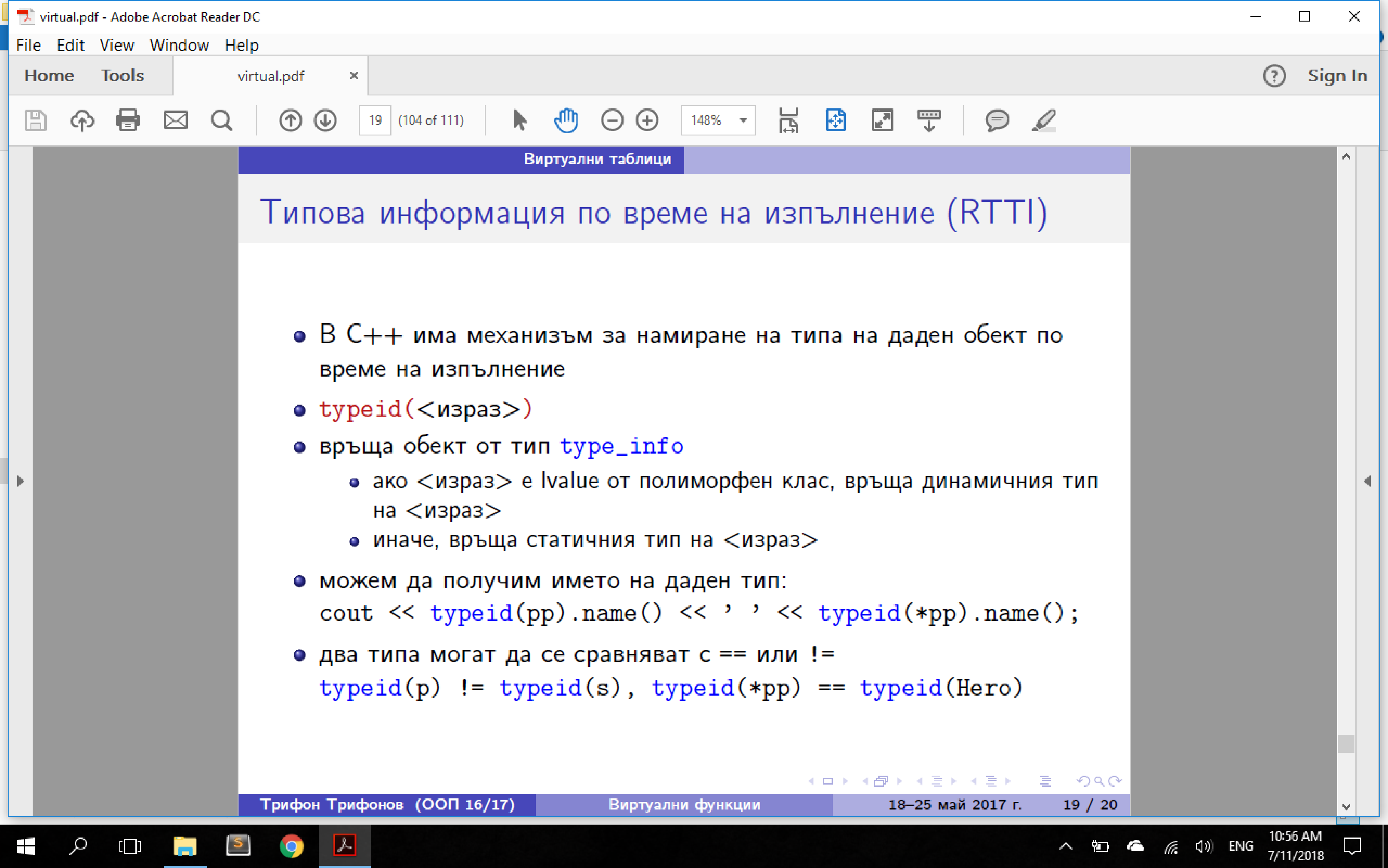


Косвено динамично свързване

```
void Player::prettyPrint() const {
  cout << ,,---------- [ Player Info ] ‘‘---------;
  print();
  cout << ,,‘‘-----------------------------------;
}
```

Дадено е Hero h; какво ще изведат:

- `Player* pp = &h; pp->prettyPrint();`
- `Player& ap = h; ap.prettyPrint();`
- `Player p = h; p.prettyPrint();`
- **Извод:** “Виртуалността” се разпростира автоматично и сред член-функциите, които извикват член-функции


virtual.pdf - Adobe Acrobat Reader DC
File Edit View Window Help
Home Tools virtual.pdf
Sign In
19 (104 of 111)
148%

Типова информация по време на изпълнение (RTTI)

- В C++ има механизъм за намиране на типа на даден обект по време на изпълнение
- typeid(<израз>)
- връща обект от тип type_info
  - ако <израз> е lvalue от полиморфен клас, връща динамичния тип на <израз>
  - иначе, връща статичния тип на <израз>
- можем да получим името на даден тип:
  `cout << typeid(pp).name() << ' ' << typeid(*pp).name();`
- два типа могат да се сравняват с == или !=
  `typeid(p) != typeid(s), typeid(*pp) == typeid(Hero)`