HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Introduction to Programming

Welcome to the Course!

Sergey Shershakov

#1/10 Jan 2019

Lecturer

Sergey Shershakov

Research Fellow @ PAISLab FCS HSE

Senior Instructor @ SSE FCS HSE

https://www.hse.ru/en/staff/sshershakov

sshershakov@hse.ru

@bjolyk

Office 419

by appointment

Drop me a line whenever you have questions or even if you just want to talk!

The Team

Sergey Shershakov

George Piatsky

Kirill Rudakov

Andrey Tatarnikov

Michael Diskin

Ilya Kostyuchenko

George Marshalko

There is simply no substitute for hard work
when it comes to achieving success.

COURSE GENERAL INFORMATION

What is Introduction to Programming-2?

- Learn how to write powerful and elegant code in C++.
- Problem solving oriented course with a lot of practical tasks.
- Based on the “classical” C++ but involving modern standards features.
- Incorporating basic data structures and algorithms — bridge to the 4th module.

Why Learn C++?

- Popularity.
- Multi-Paradigmatic High-Level Language.
- Extremely Powerful (and Dangerous — Isn't it intriguing?)
- Influenced the creation of a family of C++-styled programming languages such as Java, C#, …
  - excellent choice to be a first deeply studied language.

Why Learn C++? — TIOBE Index

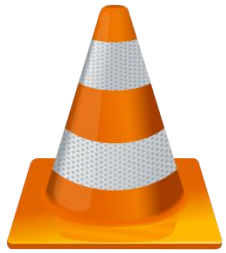
Long Tem History

Programming Language	2019	2014	2009	2004	1999	1994	1989
Java	1	2	1	1	14	-	-
C	2	1	2	2	1	1	1
C++	3	4	3	3	2	2	3
Python	4	7	5	10	20	21	-

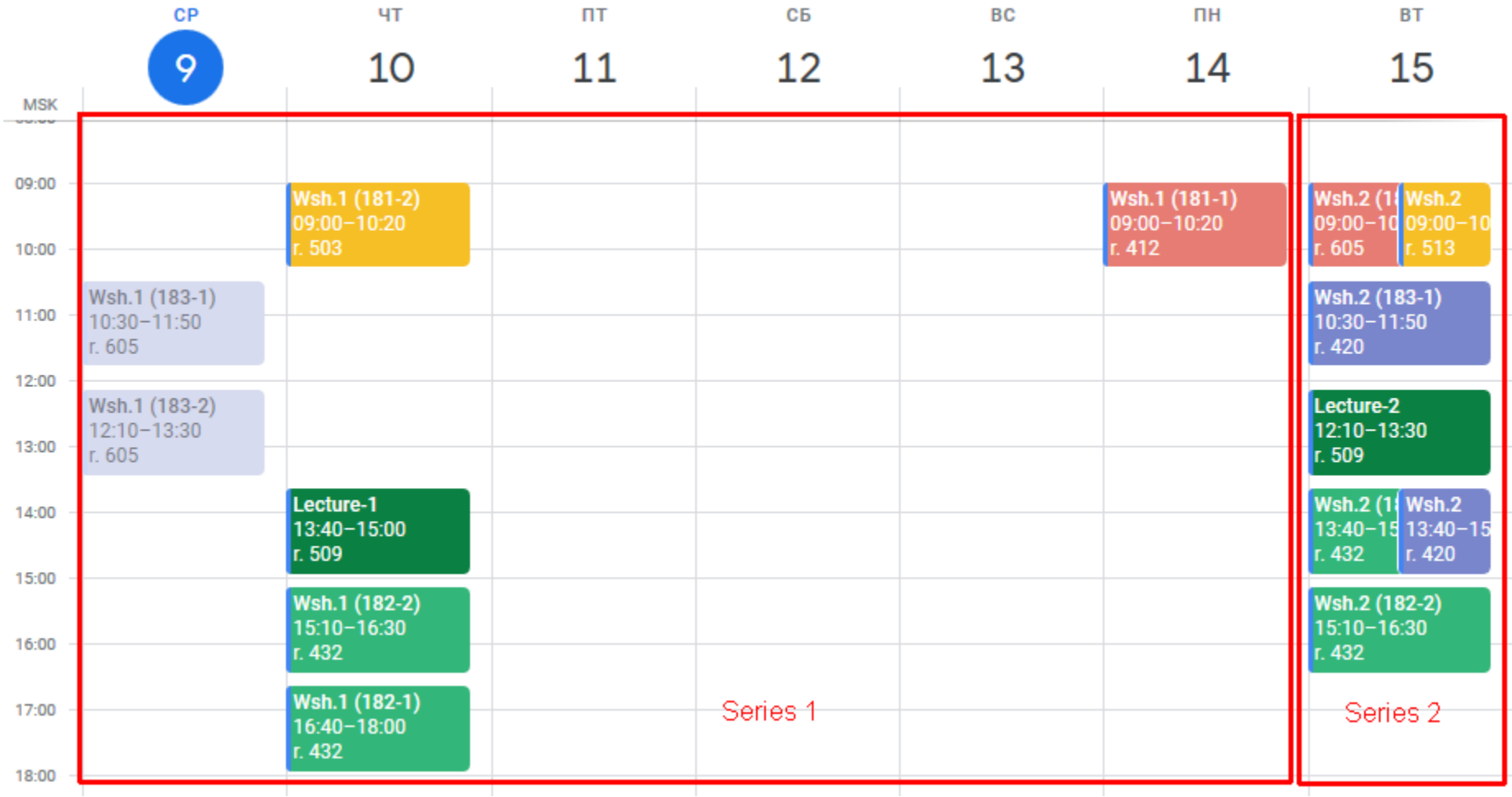
https://www.tiobe.com/tiobe-index/

Why C++: Companies

Why C++: Software



Logistics: Timetable



Logistics: Ongoing Activity and Assignments

- Working during practical classes.
- Solving problems by developing programs:
  - small and a few of large ones;
  - submitting to Я.Контест.
- Writing in-class tests.

Gives you points that constitute *OA* grade!

Logistics: Grading Details

The final grade P_2 for this part of the whole course:

$$P_2 = 0.4 \cdot E_2 + 0.6 \cdot OA$$

The ultimate grade for the whole course:

$$G = 0.7 \cdot \min(P_1, P_2) + 0.3 \cdot \max(P_1, P_2)$$

Logistics

Course Homepage:
http://wiki.cs.hse.ru/Introduction_to_programming_2

Telegram Channel (for announcements):
https://t.me/joinchat/AAAAAEhINr5SweyzcOrxow

Telegram Group (for discussions):
https://t.me/joinchat/DNq14VSd_oqlyGQrdXby4w

Lateness Policy: No late work will be accepted

Honor Code: Do not cheat, repercussions are severe!

Development

- You may use any setting of OS/Compiler/IDE/Editor/Debugger
- We refer to:
  - C++11 Standard implemented by gcc (ver. 4.8.1+)
    - MinGW ver. 8.1 port for gcc for Window is OK;
  - CMake as a (meta-)building system;
  - Qt Creator as a powerful minimalistic free IDE
    - CLion, CodeBlocks, MS VS are also OK, but on your own risk!

- You are strongly advised to bring your own laptop for all practical classes and especially control tests!

C++ HISTORY

Assembly Code

```
section   .text
global    _start          ;must be declared for linker (ld)

_start:                   ;tell linker entry point
  mov edx,len             ;message length
  mov ecx,msg             ;message to write
  mov ebx, 1              ;file descriptor (stdout)
  mov eax, 4              ;system call number (sys_write)
  int  0x80               ;call kernel
  mov eax, 1              ;system call number (sys_exit)
  int  0x80               ;call kernel

section .data
msg db 'Hello, world!',0xA            ;our dear string
len equ $ - msg           ;length of our dear string
```

Assembly Code

- Simple instructions
- Extremely fast (if written properly)
- Gives one complete control over your program

Why don’t we always use assembly?

- Requires lots of code to do simple tasks.
- Hard to understand other people’s code.
- Extremely unportable.

High-level program

- The idea is:
  - to write a program in a more intuitive language, mostly platform-independent
  - to translate the program into the low-level form that is appropriate to the CPU

C

- The C Programming Language:
  - created in 1972
  - fast, simple, cross-platform
  - the most used programming language so far

Ken Thompson and Dennis Ritchie

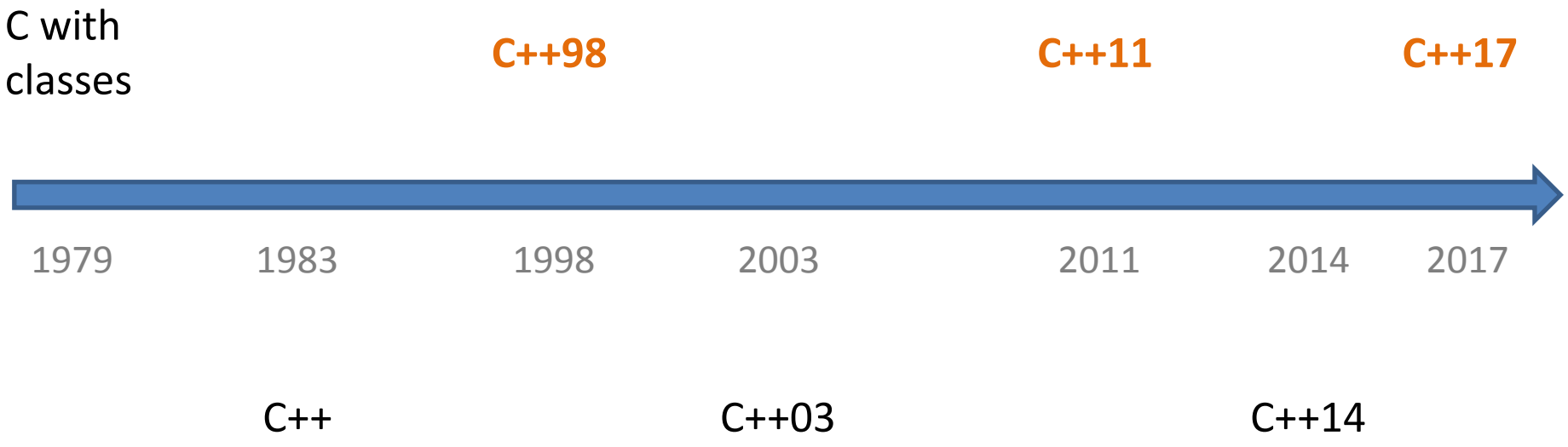
- Some weakness:
  - No objects or classes
  - Difficult to write code that worked generically
  - Tedious when writing large programs

Bjarne Stroustrup and C++

- The first version of C++ was created as an extension of C “with classes”
- It was designed to:
  - be fast
  - be simple to use
  - be cross-platform
  - have high-level features

C++ developers

Evolution of C++



Philosophy

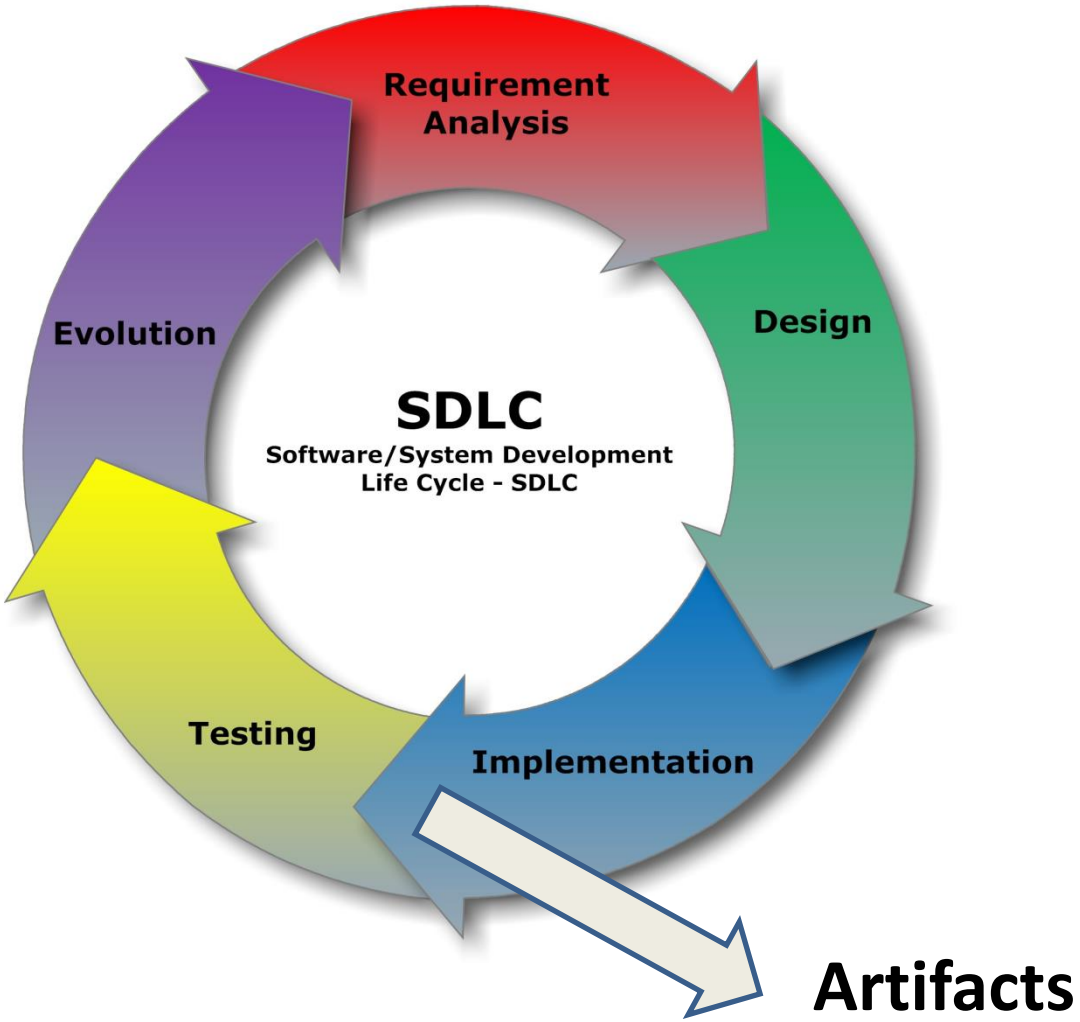
- It must be driven by actual problems and its features should be useful immediately in real world programs.
- Programmers should be free to choose their own style.
- Allowing a useful feature is more important than preventing every possible misuse of C++.
- No implicit violations of the type system.
- User-created types need to have the same support and performance as built-in types.
- Unused features should not negatively impact created executables.
- Enforce safety at compile time whenever possible.

- The Zero Overhead Principle:
  - «What you don't use, you don't pay for»
  - «What you do use, you couldn't hand code any better»

Targets and Building

BUILD YOU FIRST PROGRAM!

Application Development Lifecycle



Targets

- *Calculator* — stand-alone executable
- *Media Player* — pluggable app:
  - *MP Framework* — individual app
  - *MP3 Plugin* (shared lib, dll)
  - *AVI Plugin* (shared lib, dll)
  - …

Targets: Applications and Libraries

- *Application* — normally stand-alone executable (binary) file
- *Library* — contains common piece of compiled code (can also be given in the form of SC)
  - *Static Library* — is to be connected to an app during *linking* stage (compile time)
  - *Shared (Dynamic) Library* — is to be connected to an app during its run (runtime)

- Normally each target consists of a number of units. Each unit contains code related to some specific problem.
  - In source code a unit is represented by one or more source files

Interface and Implementation

Interface:

- *declare* smth w/o *defining* any specific way to implement this;
- specify a *contract* between two different parts;

Implementation:

- carry out exactly what an *interface* declares;
- there can be a number of ways to *implement* the same *interface* and, hence, satisfy the *contract*;

Interface and Implementation in C++

Interface:

- declare a prototype of a function;
- create a new *type*;
- declare a class with pure *virtual* functions — an *interface class*;
- often represented by *header files* (.h, .hpp)

Implementation:

- implement a function by writing its body;
- create a new object of a specific type;
- inherit the interface class and *override* virtual functions;
- often represented by *translation units* (.cpp)

Translation Unit

- *Translation Unit* is a text file with a program in C++, named normally with .cpp extension, that can be compiled by a C++-compiler.

- Header Files (.h, .hpp, …) are not translation units and, hence, cannot be compiled.

  DO NOT EVENT TRY to compile header files!

Building Stages