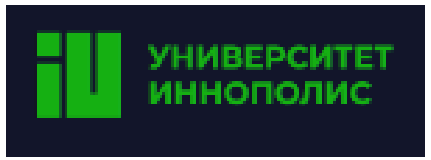
Университетский курс Compiler Construction: философия, задачи, структура, опыт

Евгений Зуев
Университет Иннополис



Конференция C++ Russia
Москва, 21 марта 2025

Outline

- Курс «Разработка компиляторов»: предпосылки, задачи, структура.
- Содержание и реализация курса.
- Учебные проекты: языки D, O, F, I.
- «Мастер-язык» Simple и его реализация

Предпосылки: философия

- Разработка компиляторов ЯП традиционно считается одним из трех краеугольных камней Computer Science наряду с базами данных и операционными системами.
- Эта область, наряду с практическими аспектами разработки ПО, включает дисциплины фундаментального характера: теория формальных грамматик, теория типов, теория алгоритмов, теория оптимизации и т.д.
- Курс compiler construction является непременной частью компьютерного образования практически во всех университетах.

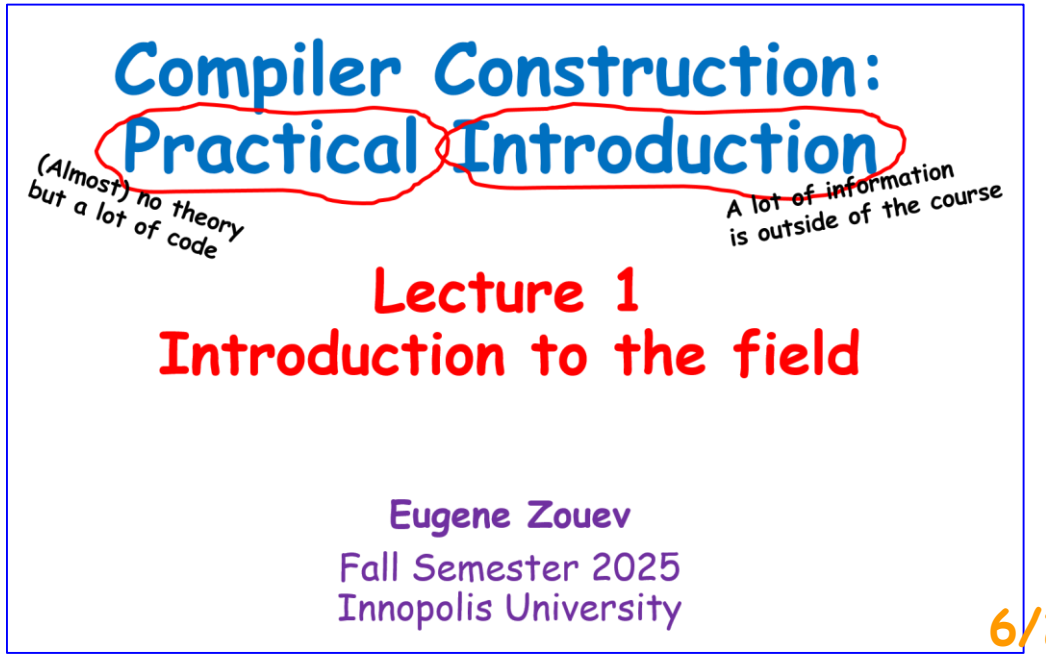
Личный опыт автора

- **C++** front-end compiler for the ISO 1998 Standard
  - Interstron Ltd., Moscow, 1999-2000.
- **Zonnon** language implementation for .NET & Visual Studio
  - ETH Zürich, 2005.
- **Swift** prototype compiler for Tizen & Android
  - Samsung Research, 2015
- Many other language-related projects and tools.
- Е. Зуев, «Редкая профессия», ДМК Пресс, 2014

Организация и содержание курса

Задачи и особенности курса

- Ознакомить студентов с существом процесса компиляции, принципами реализации и реальными примерами компиляторов.
- Курс носит выраженный практический характер
  - Продолжительность: (всего) один семестр
  - Педагогический прием: лекции + проекты + мастер-классы
  - Основная задача: пройти все этапы разработки вплоть до создания работающего компилятора/интерпретатора.
- Язык преподавания:
  Иннополис: английский
  ИТМО: русский



Содержание курса

- Introduction to the topic. Compilation stages
- Lexical analysis
- Syntax analysis
- Compilation structures
- Parser generators: Yacc/Bison
- Semantic analysis
- Source-level optimizations
- Compilation for Virtual Machines; .NET & JVM
- Bytecode interpretation techniques
- Python: Compiler & Virtual Machine
- The LLVM infrastructure for compiler construction
- Introduction to the language runtime support
- Evolution of the compiler architecture

Общая структура курса

- **Лекции**: 13-14, каждую неделю.
- **Доп. занятия**: 2-3 раза в течение курса

Общая структура курса

- **Лекции**: 13-14, каждую неделю.
- **Доп. занятия**: 2-3 раза в течение курса

- **Команды по 2-3 человека**
  выполняют семестровый проект:
  компилятор/интерпретатор учебного языка
- **Проект по выбору**:
  входной язык, язык реализации, технология фронтальной части, целевая платформа
- Регулярные (пять раз) **отчеты команд**
- **Финальная защита проекта**

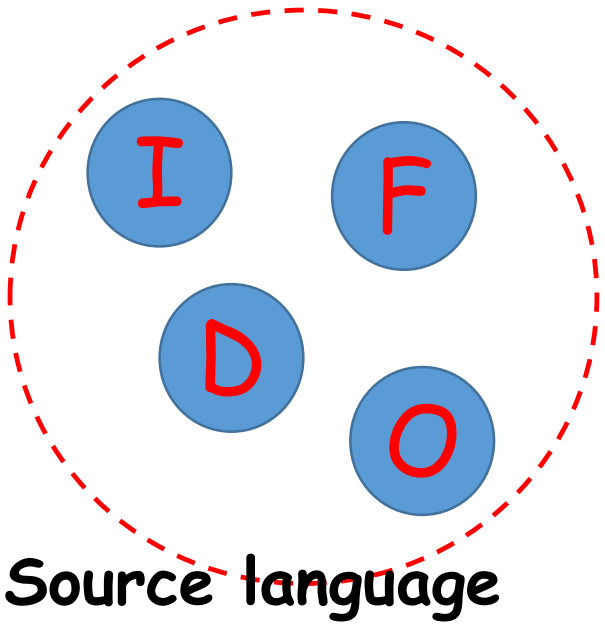
Общая структура курса

- **Лекции**: 13-14, каждую неделю.
- **Доп. занятия**: 2-3 раза в течение курса

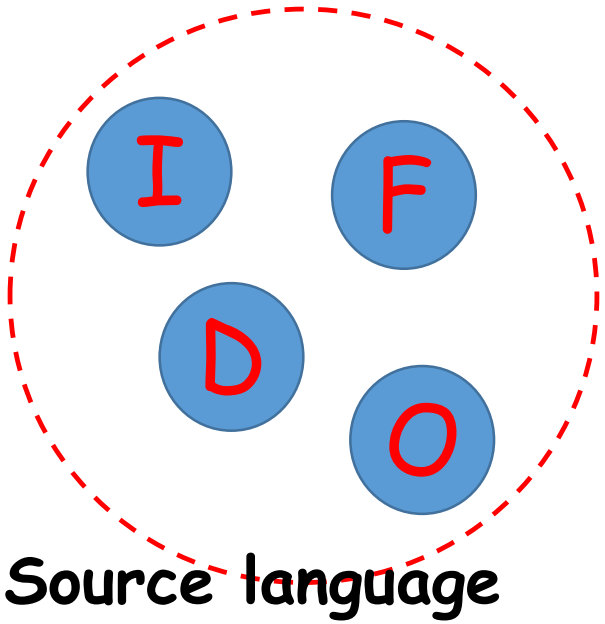
- **Команды по 2-3 человека**
  выполняют семестровый проект:
  компилятор/интерпретатор учебного языка
- **Проект по выбору**:
  входной язык, язык реализации, технология фронтальной части, целевая платформа
- Регулярные (пять раз) **отчеты команд**
- **Финальная защита проекта**

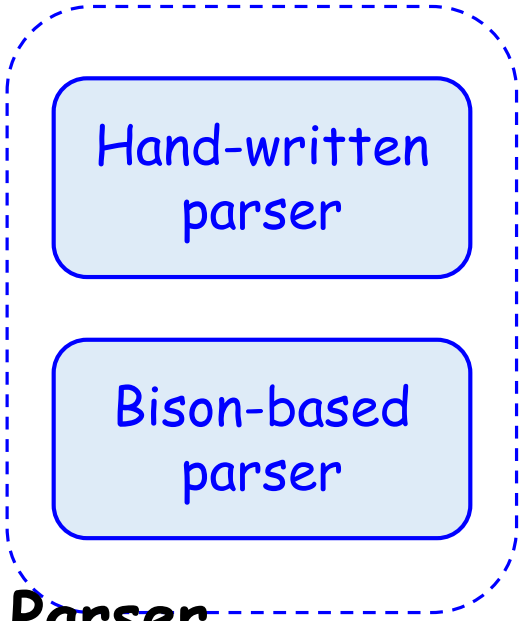
- **Мастер-классы**: master solutions (пять раз)
  по всем этапам процесса компиляции
  на основе специального модельного языка

Конфигурация проекта

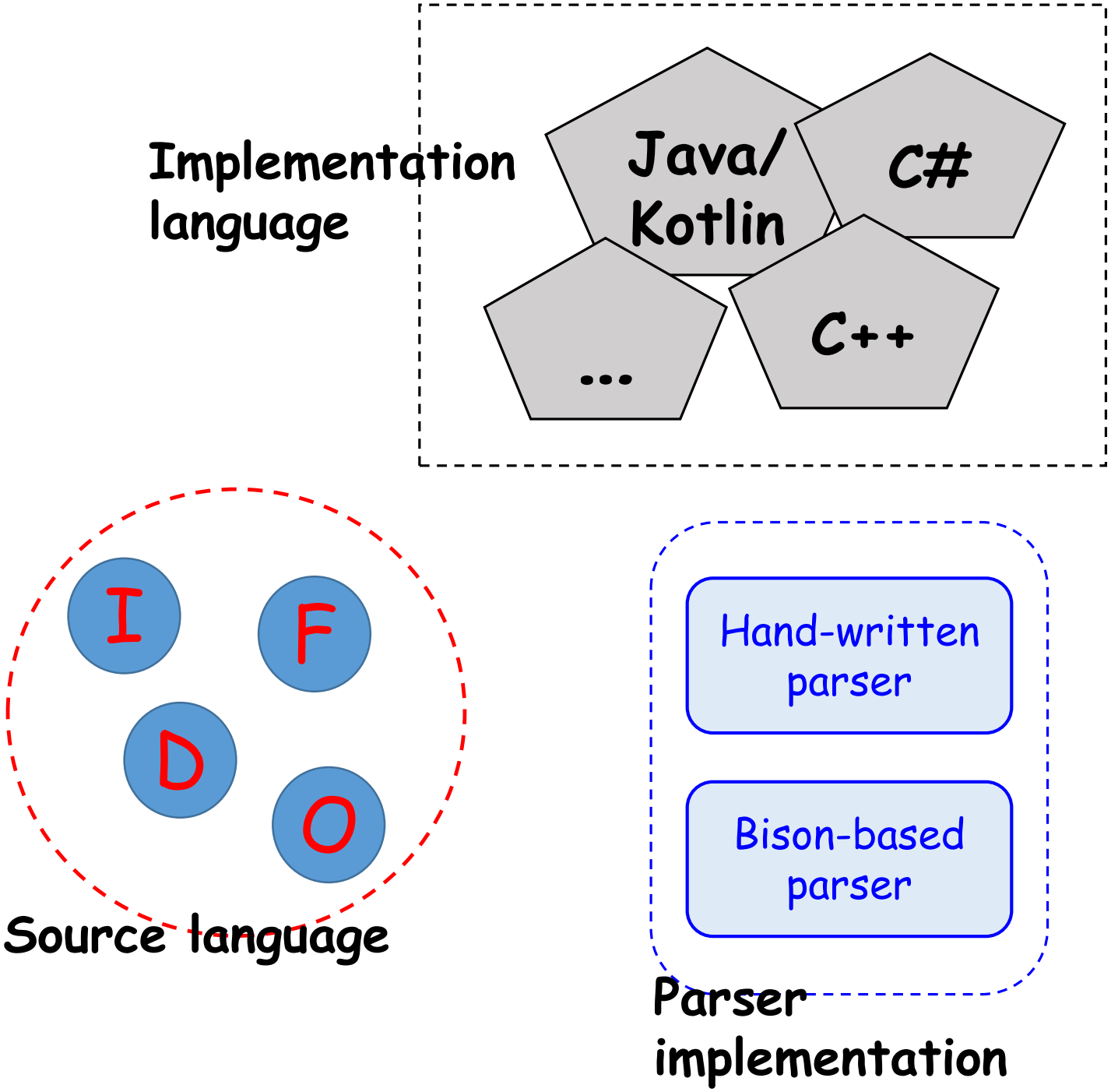


Конфигурация проекта

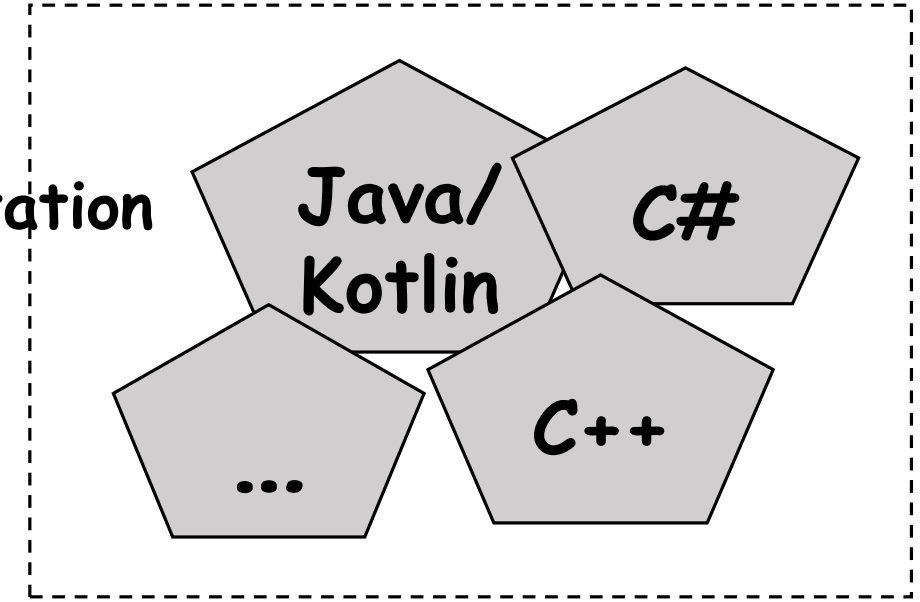


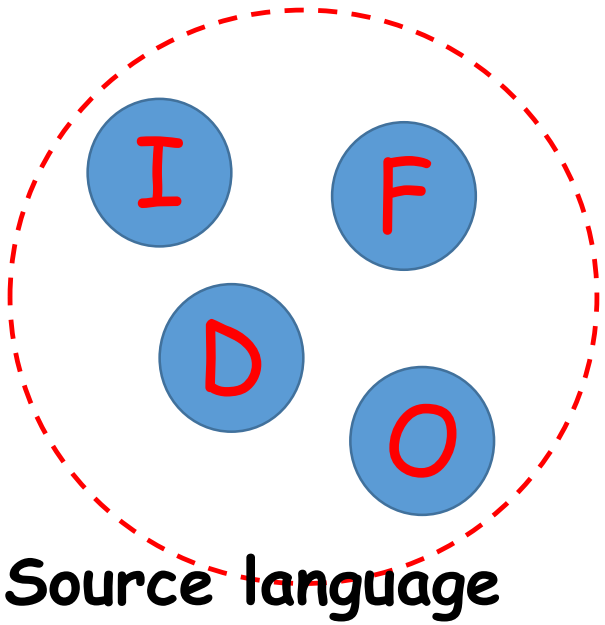


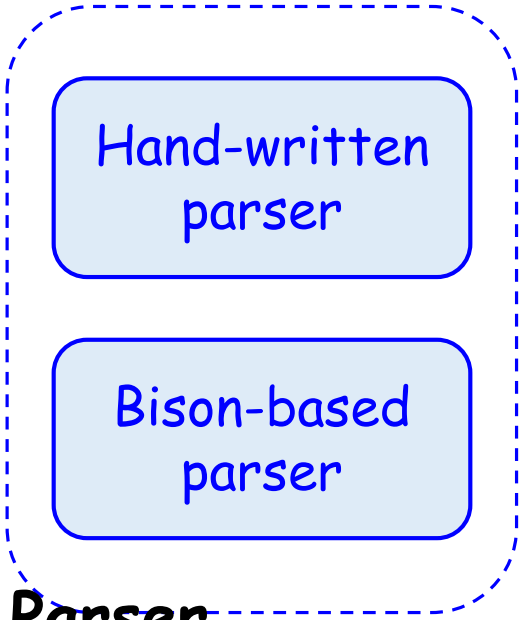
Конфигурация проекта



Конфигурация проекта







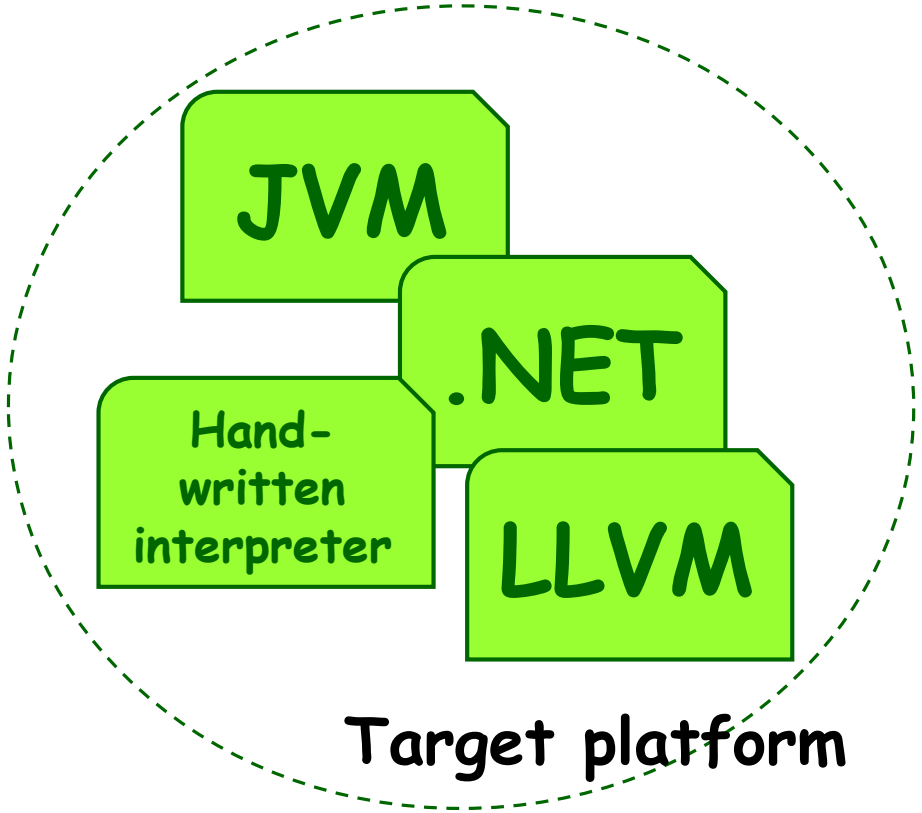
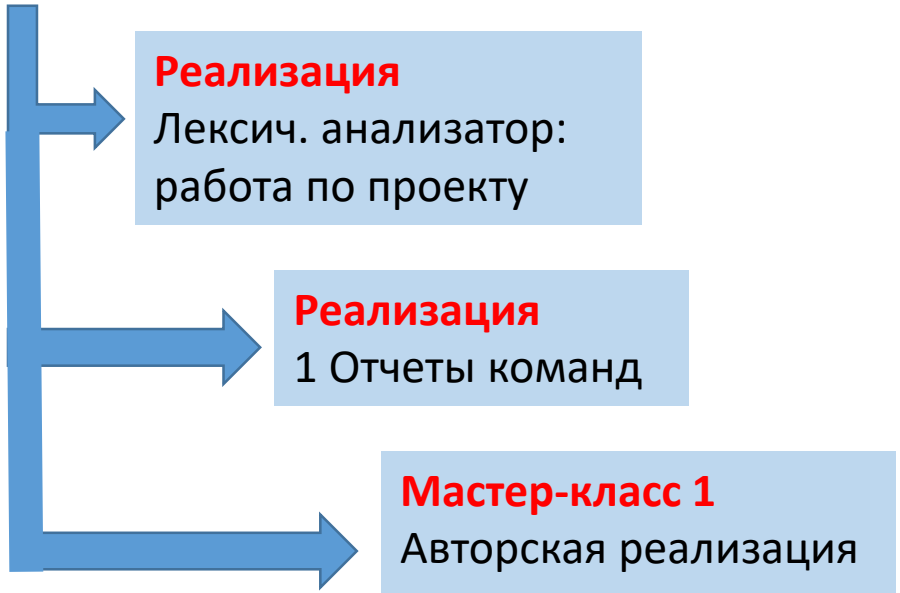


Схема организации активностей

Лекция 1
Лексический анализ: назначение, подходы, механизмы, инструменты

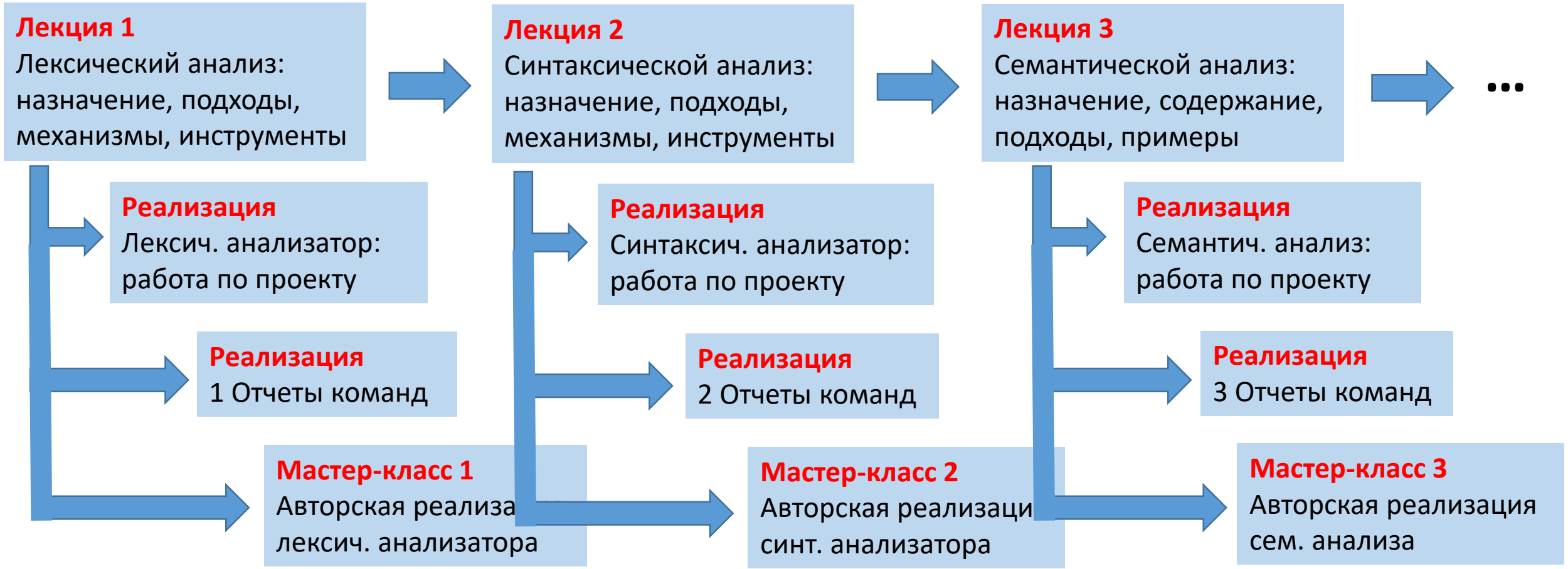


Реализация
Лексич. анализатор: работа по проекту

Реализация
1 Отчеты команд

Мастер-класс 1
Авторская реализация лексич. анализатора

Схема организации активностей



Проектные языки

The D Language

Prototype: ***JavaScript***

- Dynamic typing: object types are not specified and can change while program execution.
- The language assumes **interpretation.**
- Major notion: variable & literal (constant).
- Program structure: a sequence of declarations and statements.
- Builtin types:
  integer, real, boolean, string
  User-defined types: array, tuple, function.
- Implicit type conversions are supported.
- Statements: assignment, if/while/loop, exit/return, output.

```
var polygon := [];
polygon := polygon + [ {x :=  2, y :=  5} ];
polygon := polygon + [ {x := -1, y :=  1} ];
polygon := polygon + [ {x := -4, y :=  0} ];
polygon := polygon + [ {x :=  2, y := -3} ];
polygon := polygon + [ {x :=  3, y :=  1} ];

var j := 0;
var area := 0;
var cross := func(a, b) => a.x * b.y - a.y * b.x;

for point in polygon loop
    j := j + 1;
    if polygon[j] = none then
        j := 0;
    end
    area := area + cross(point, polygon[j]);
end
area := area / 2.0;
print area;
```

```
var make_node :=
 func(id, neighbors)
    => {id := id,
        neighbors := neighbors}
var graph := [
  make_node("j", [10, 5]),
  make_node("g", [13]),
  make_node("l", [6, 8]),
  make_node("b", []),
  make_node("i", [13]),
  make_node("d", [4, 12]),
  make_node("f", [13]),
  make_node("k", [7, 2, 1]),
  make_node("m", [11, 3]),
  make_node("h", [13]),
  make_node("a", []),
  make_node("c", []),
  make_node("e", [])
]
```

```
var result := []
var state := []
var visit := func(idx) is
  if not (state[idx] is none) then
    return state[idx]
  end
  state[idx] := false
  for neighbor_idx in graph[idx].neighbors loop
    if not visit(neighbor_idx) => return false
  end
  state[idx] := true
  result = result + [idx]
  return true
end

// Prints the first half of the alphabet in order.
if visit(9) then
  for idx in result loop
    print graph[idx].id
  end
else
  print "found a cycle"
end
```

The O Language

Prototype: **Oberon**

- The language is statically typed. Object types are fixed by object declarations and cannot change while program execution.
- Object types are classes – either predefined of user-defined.
- Classes contain data declarations and method declarations.
- The single inheritance and virtual functions are supported.
- **No expressions! – only function calls**.
- Program structure: a sequence of classes.
- Statements: a standard set (assignment, if/while, return, input/output)

```
class Main is
  this() is
    var arr : Array[Integer](5)
    arr.set(0, 10)
    arr.set(1, 20)
    arr.set(2, 30)
    arr.set(3, 40)
    arr.set(4, 50)
    var bs : BinarySearch()
    var target : Integer(30)
    var result : Integer(bs.search(arr, target))
    if result.Equal(-1) then
        Print("Target not found")
    else
        Print("Target found at index: ")
        Print(result)
    end
  end
end
```

The O Language

```
class BinarySearch is
  method search(arr: Array[Integer], target: Integer) : Integer is
    var low : Integer(0)
    var high : Integer(arr.Length.Minus(1))
    while low.LessEqual(high) loop
      var mid : Integer(low.Plus(high).Div(2))
      var midVal : Integer(arr.get(mid))
      if midVal.Equal(target) then
          return mid
      else if midVal.Less(target) then
          low := mid.Plus(1)
      else
          high := mid.Minus(1)
      end
    end
    return Integer(-1)
  end
end
```

The I Language

*Prototype: **Pascal***

- The language is statically typed. Object types are fixed by object declarations and cannot change while program execution.
- There are three predefined data types: integer, real and boolean.
  There are two predefined data structures: structs and arrays.
- Full expression syntax with usual set of operators.
- The language is **compiled**.
- Program structure: a sequence of data and routine declarations.
- Statements: a standard set (assignment, if/while, return, input/output)

```
// Решето Эратосфена
routine main() is
  for i in 2..100 loop
    var n is 2
    var is_simple is true
    while n*n < i loop
      is_simple := is_simple and (i%n /= 0)
      n := n + 1
    end
    if is_simple then print i end
  end
end
```

The I Language

```
routine is_sorted(arr: array[] integer): boolean is
  var result is true
  for i in 1..arr.length-1 loop
    result := result and (arr[i] <= arr[i+1])
  end
  return result
end
```

```
routine bubble_sort(arr: array[] integer) is
  while not is_sorted(arr) loop
    for i in 1..arr.length-1 loop
      if arr[i] > arr[i+1] then
        var tmp is arr[i]
        arr[i] := arr[i+1]
        arr[i+1] := tmp
      end
    end
  end
end
```

```
routine main() is
  var arr : array[5] integer
  arr[1] := 2
  arr[2] := 2
  arr[3] := 5
  arr[4] := 1
  arr[5] := 3
  bubble_sort(arr)
  print arr // 1 2 2 3 5
end
```

The F Language

Prototype: ***Lisp***

- There are three atomic predefined types: integers, reals & booleans.
- There are two basic structures: atoms and lists.
  Atom has the name and the value. The atom value can be either of a predefined type, or, or list. In some contexts atom represents itself.
  A list is an ordered sequence of atoms, literals and lists.
- Basic list access rules include taking the head of the list, the “tail” of the list, and constructing a list from its head and the “tail”..
- The language is dynamically typed and **interpreted**.
- Program structure: a sequence of lists and/or atoms
- Special forms (lists): functions, lambdas (unnamed functions), control structures.
- A minimal set of predefined functions: arithmetic, lists operations, etc.

```
(func mod (a b)
   (cond (equal b 0)
      null
      (minus a (times b (divide a b)))
   )
)
(func gcd (a b)
   (cond (equal b 0)
     a
     (gcd b (mod a b))
   )
)
(gcd 15 10)
```

```
(func length (list)
   (cond (not (islist list))
      0
      (plus 1 (length (tail list)))
   )
)
```

“Master Language”

Язык Simple

- Спроектирован специально для демонстрации основных проектных и программных решений при разработке компиляторов.
- Структурно простой, но полнофункциональный императивный язык со статической типизацией, развитыми структурами данных (массивы, кортежи, списки, функциональные типы), сопрограммным механизмом.
- Служит основой серии мастер-классов по всем аспектам разработки.
- Создан плагин для VsCode, который показывает все этапы компиляции/выполнения с демонстрацией AST и результирующего кода и с пошаговым исполнением кода.

Язык Simple: пример

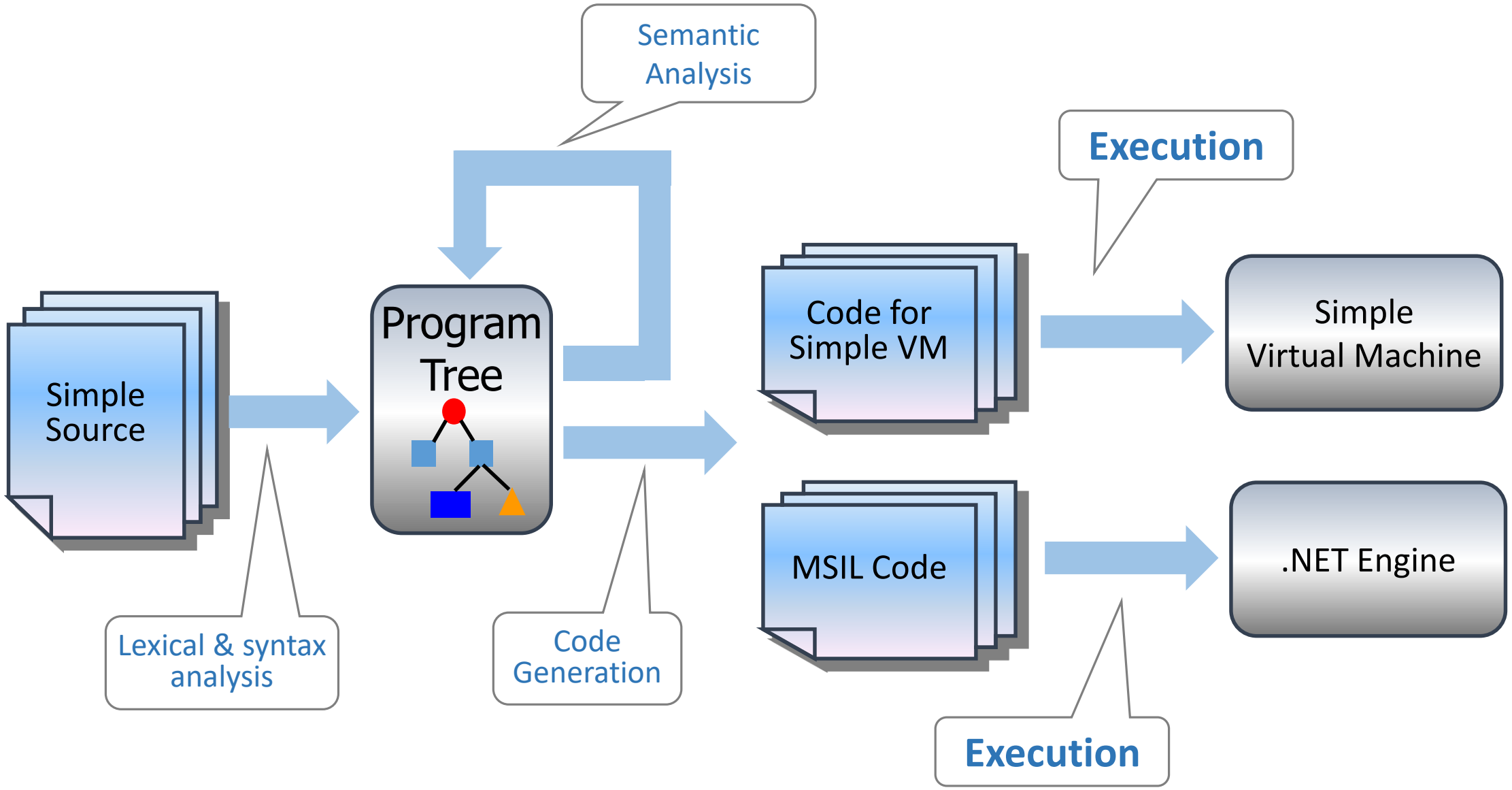
```
func merge(array: [Int], arrayCopy: [Int], l: Int, m: Int, r: Int)
  var iterL := l;
  var iterR := m + 1;
  for i in l..r loop
    if iterL > m then
      array[i] := arrayCopy[iterR];
      iterR := iterR + 1;
    elsif iterR > r then
      array[i] := arrayCopy[iterL];
      iterL := iterL + 1;
    elsif arrayCopy[iterL] > arrayCopy[iterR] then
      array[i] := arrayCopy[iterR];
      iterR := iterR + 1;
    else
      array[i] := arrayCopy[iterL];
      iterL := iterL + 1;
    end
  end
end
```

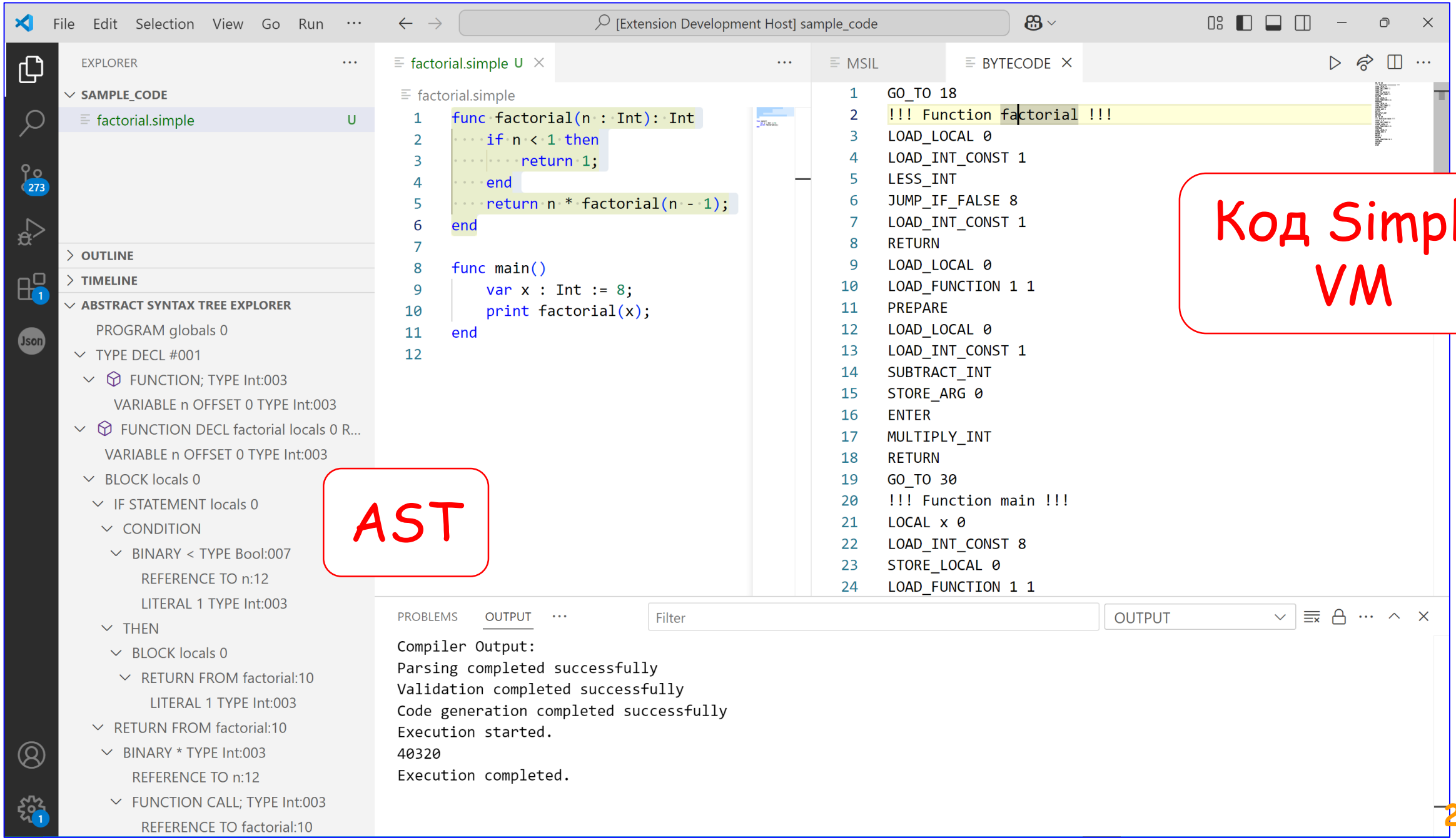
```
func split(array: [Int],
           arrayCopy: [Int], l: Int, r: Int)
    if r <= l then return; end
    const m := (l + r) / 2;
    print l, r, m;
    split(arrayCopy, array, l, m);
    split(arrayCopy, array, m + 1, r);
    merge(array, arrayCopy, l, m, r);
end

func mergeSort(array: [Int])
    split(array, copy(array), 1, #array);
end

func main()
    var array := [5, 8, 2, 4, 1, 1, 10, 5, 3, 7];
    mergeSort(array);
    print array; // [1, 1, 2, 3, 4, 5, 5, 7, 8, 10]
end
```

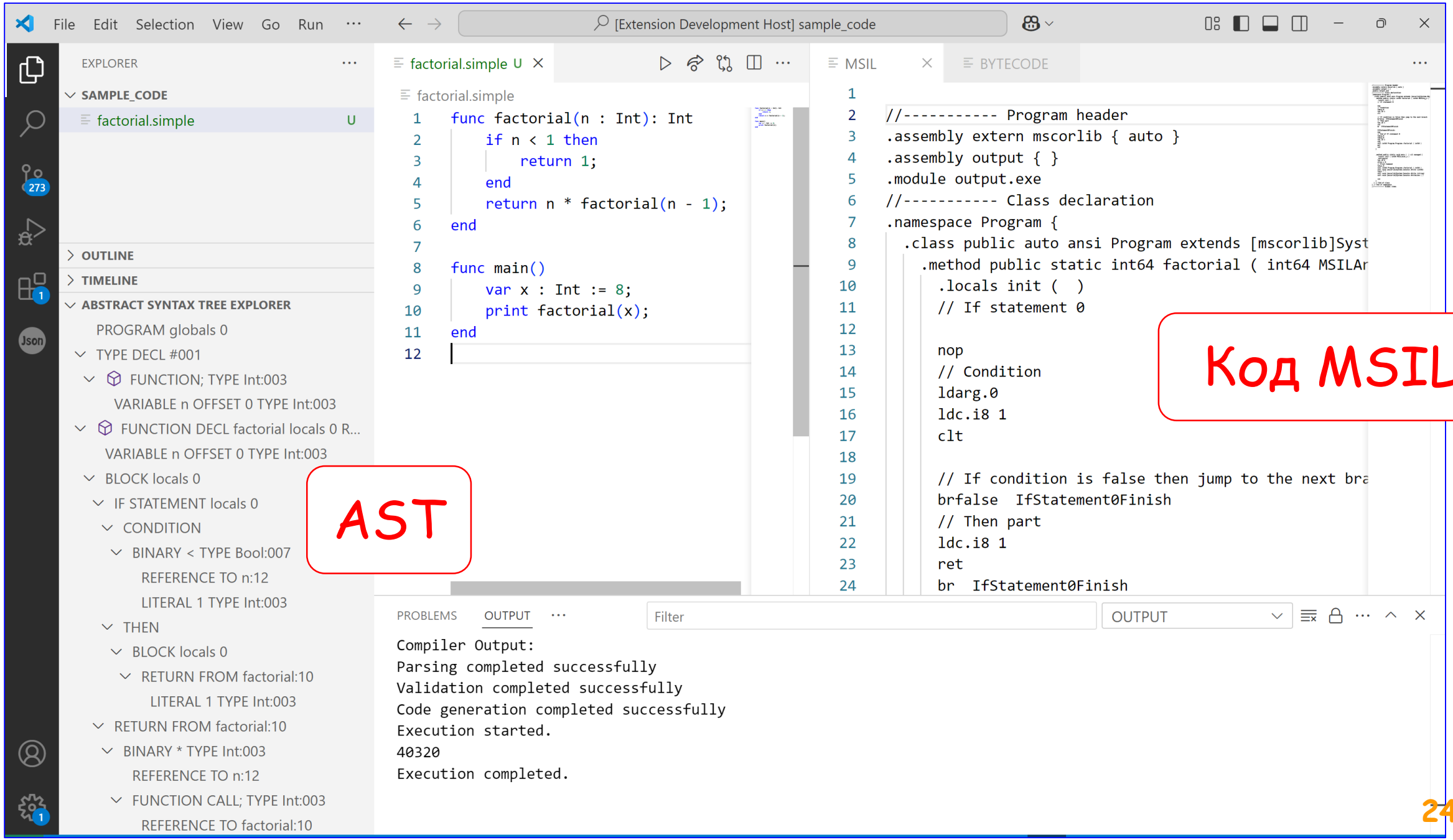
Язык Simple: Compilation & Execution





AST

Код Simple VM


File Edit Selection View Go Run
[Extension Development Host] sample_code
EXPLORER
SAMPLE_CODE
factorial.simple U
OUTLINE
TIMELINE
ABSTRACT SYNTAX TREE EXPLORER
PROGRAM globals 0
TYPE DECL #001
FUNCTION; TYPE Int:003
VARIABLE n OFFSET 0 TYPE Int:003
FUNCTION DECL factorial locals 0 R...
VARIABLE n OFFSET 0 TYPE Int:003
BLOCK locals 0
IF STATEMENT locals 0
CONDITION
BINARY < TYPE Bool:007
REFERENCE TO n:12
LITERAL 1 TYPE Int:003
THEN
BLOCK locals 0
RETURN FROM factorial:10
LITERAL 1 TYPE Int:003
RETURN FROM factorial:10
BINARY * TYPE Int:003
REFERENCE TO n:12
FUNCTION CALL; TYPE Int:003
REFERENCE TO factorial:10
factorial.simple U
factorial.simple
1 func factorial(n : Int): Int
2 if n < 1 then
3 return 1;
4 end
5 return n * factorial(n - 1);
6 end
7
8 func main()
9 var x : Int := 8;
10 print factorial(x);
11 end
12
MSIL
BYTECODE
1
2 //---------- Program header
3 .assembly extern mscorlib { auto }
4 .assembly output { }
5 .module output.exe
6 //---------- Class declaration
7 .namespace Program {
8 .class public auto ansi Program extends [mscorlib]Syst
9 .method public static int64 factorial (int64 MSILAr
10 .locals init ( )
11 // If statement 0
12
13 nop
14 // Condition
15 ldarg.0
16 ldc.i8 1
17 clt
18
19 // If condition is false then jump to the next bra
20 brfalse IfStatement0Finish
21 // Then part
22 ldc.i8 1
23 ret
24 br IfStatement0Finish
AST
Код MSIL
PROBLEMS
OUTPUT
Filter
OUTPUT
Compiler Output:
Parsing completed successfully
Validation completed successfully
Code generation completed successfully
Execution started.
40320
Execution completed.

Заключение и выводы

- Сокращенный курс с практической направленностью.
- **Интеграция лекционного материала, практических заданий и мастер-классов**.
- Проекты: достаточно полное покрытие входных языков, языков реализации и целевых платформ.
- **Инструментальная поддержка: «мастер-язык» с плагином для VsCode**.

Спасибо за интерес и внимание!