Факультет компьютерных наук

Образовательная программа
«Программная инженерия»

Москва
2026

Компилятор языка программирования Обольд

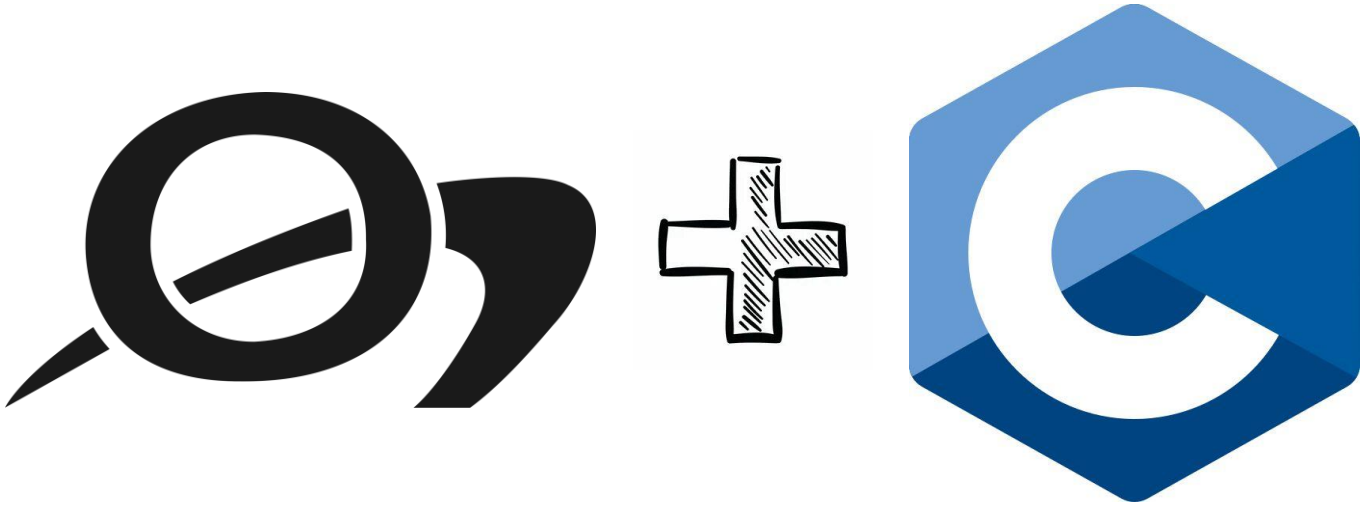
Выполнили: студент БПИ232 Петров Константин
студент БПИ232 Чубенко Семён

Научный руководитель: Легалов Александр Иванович, профессор департамента программной инженерии, доктор технических наук

Описание предметной области

Обольд - статически типизированный модульный язык, повторяющий семантику языка Оберон-07, но имеющий Си-подобный синтаксис.

Программа предназначена для трансляции исходного кода, написанного на языке Обольд, в промежуточное представление на языке Си и для компиляции в машинный код.



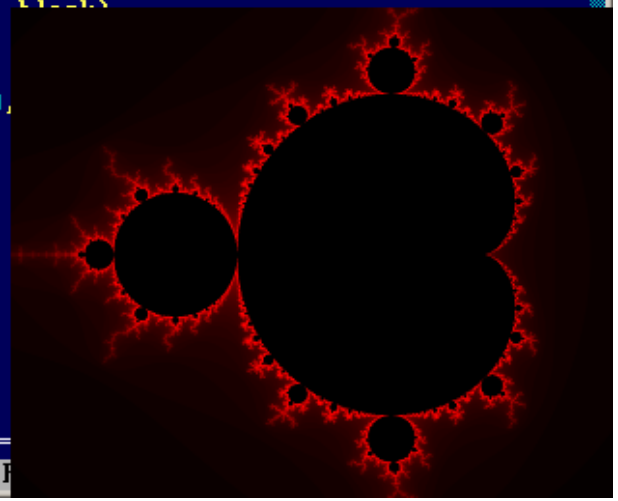
Актуальность работы

Free Oberon

File Edit Search Run Compile Debug Tools Options Window Help

Mandelbrot.Mod

```
MODULE Mandelbrot;
IMPORT G := Graph;
VAR s: G.Bitmap;
  black: INTEGER;

PROCEDURE Go(sx, sy: INTEGER; x, y: REAL);
CONST iter = 768;
VAR col, i: INTEGER;
  re, im, re2: REAL;
BEGIN re := x; im := y; i := 0;
  REPEAT re2 := re * re - im * im;
    im := 2 * re * im; re := re2;
    re := re + x; im := im + y; INC(i)
  UNTIL (i = iter) OR (re * re + im * im > 4);
  IF i = iter THEN G.PutPixel(s, sx, sy,
  ELSE i := i * 5;
    IF i > 255 THEN i := 255 END;
    G.PutPixel(s, sx, sy, G.MakeCol(i, 0,
  END
END Go;

PROCEDURE Do;
VAR x, y: INTEGER;
  x0, y0, x1, y1: REAL;
BEGIN G.Settings(0, 0, {G.fullscreen});
  s := G.Init();
  black := G.MakeCol(0, 0, 0);
  y0 := 1; y1 := -1;
```



F1 Help F2 Save F3 Open F9 Compile &

```
module FractalFlame

import Args, FlameCli, FlameRenderer, FlameTypes, Images, Out;

fn init() -> void
var {
    config: FlameTypes.Config;
    image: Images.Image;
    ok: bool;
}
{
    ok = FlameCli.LoadConfig(config);
    if ok {
        image = FlameRenderer.Render(config);
        if image == nil {
            Out.String("Rendering failed."); Out.Ln();
        } else {
            ok = Images.SavePNG(image, config.outputPath);
            if ok {
                Out.String("Saved image to ");
                Out.String(config.outputPath);
                Out.Ln();
            } else {
                Out.String("Failed to save image to ");
                Out.String(config.outputPath);
                Out.Ln();
            }
        }
    }
}
```

Цель: разработать компилятор для нового языка программирования Обольд

Задачи

1) Разработать синтаксис языка Обольд
2) Определить стек используемых технологий
3) Реализовать лексер - разбор текста на токены
4) Реализовать парсер рекурсивным спуском
5) Реализовать семантический анализ
6) Реализовать работу с символьными файлами
7) Реализовать транспилятор в Си
8) Реализовать генерацию машинного кода и объектных файлов
9) Написать тестовые и демонстрационные программы на языке Обольд

Общее
Константин
Семён

Синтаксис языка Обольд

```
Point: struct {
  x, y: i64;
}

Pixel: struct (Point) {
  color: i64;
}
```

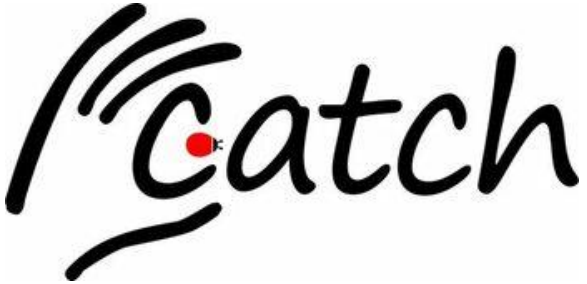
```
while j > 0 {
    j = j / 2;
    i = i+1;
}

while t != nil && t.key != i {
    t = t.left;
}

while m > n {
    m = m - n;
} elif n > m {
    n = n - m;
}

do {
    k = k / 2;
    n = n + 1;
} while k != 0;

for (v = beg, end, inc) {S}
```

```
if ch >= "A" && ch <= "Z" {
    ReadIdentifier();
} else if ch >= "0" && ch <= "9" {
    ReadNumber();
} else {
    ReadString();
}
```

```
fn SafeDiv(x, y: i64, res: &i64) -> bool
var isCorrect: bool;
{
  if y == 0 {
    isCorrect = false;
  } else {
    res = x / y;
    isCorrect = true;
  }
  return isCorrect;
}
```

```
switch k {
   case 0: x = x + y;
   case 1: x = x - y;
   case 2: x = x * y;
   case 3: x = x / y;
 }

 switch p {
   case P0: p.b = 10;
   case P1: p.b = 2.5;
   case P2: p.b = true;
 }
```

Стек используемых технологий



JSON for Modern C++

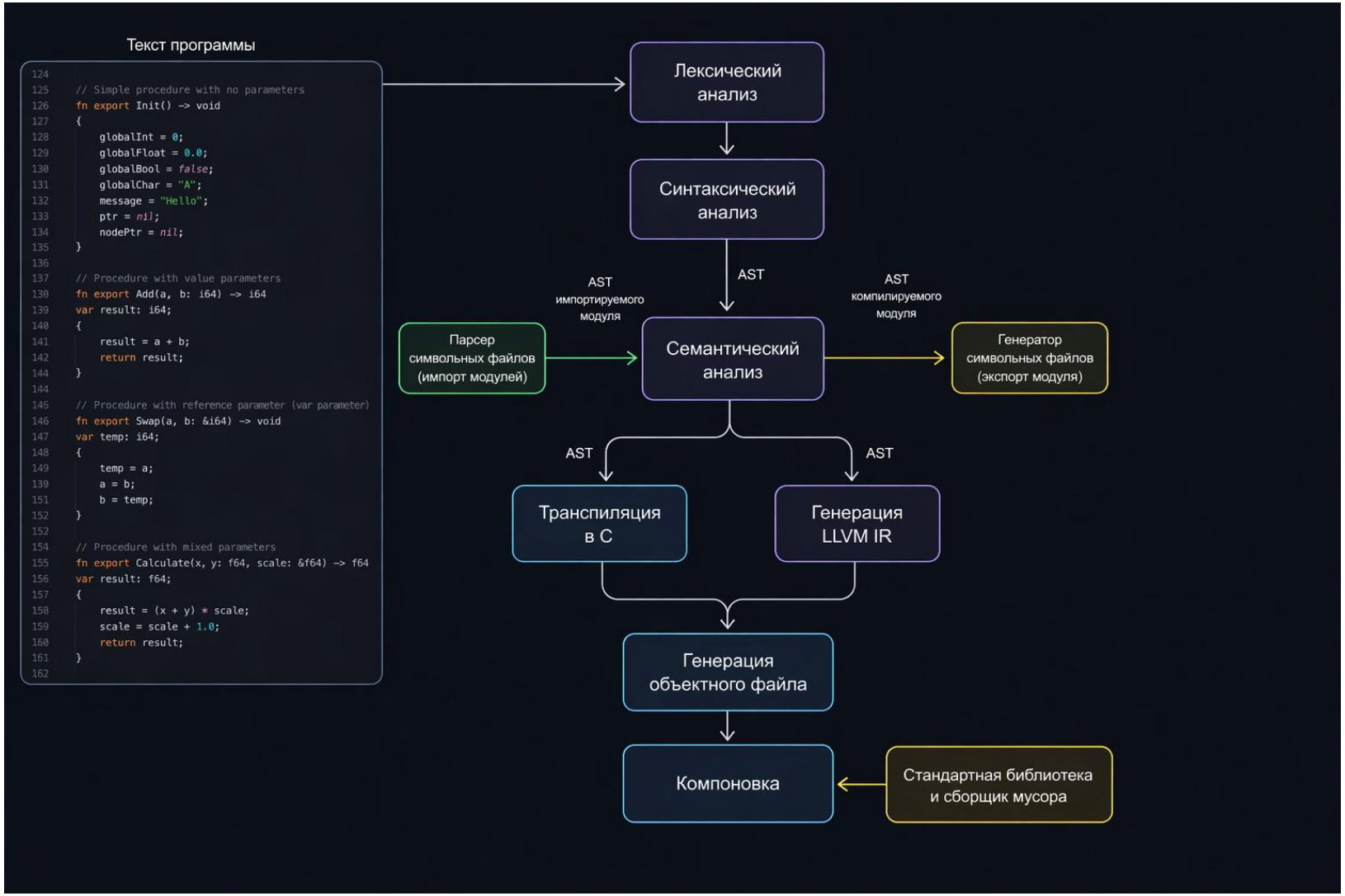
What if JSON was part of modern C++?

The "Boehm-Demers-Weiser" Conservative Garbage Collector

Архитектура программы

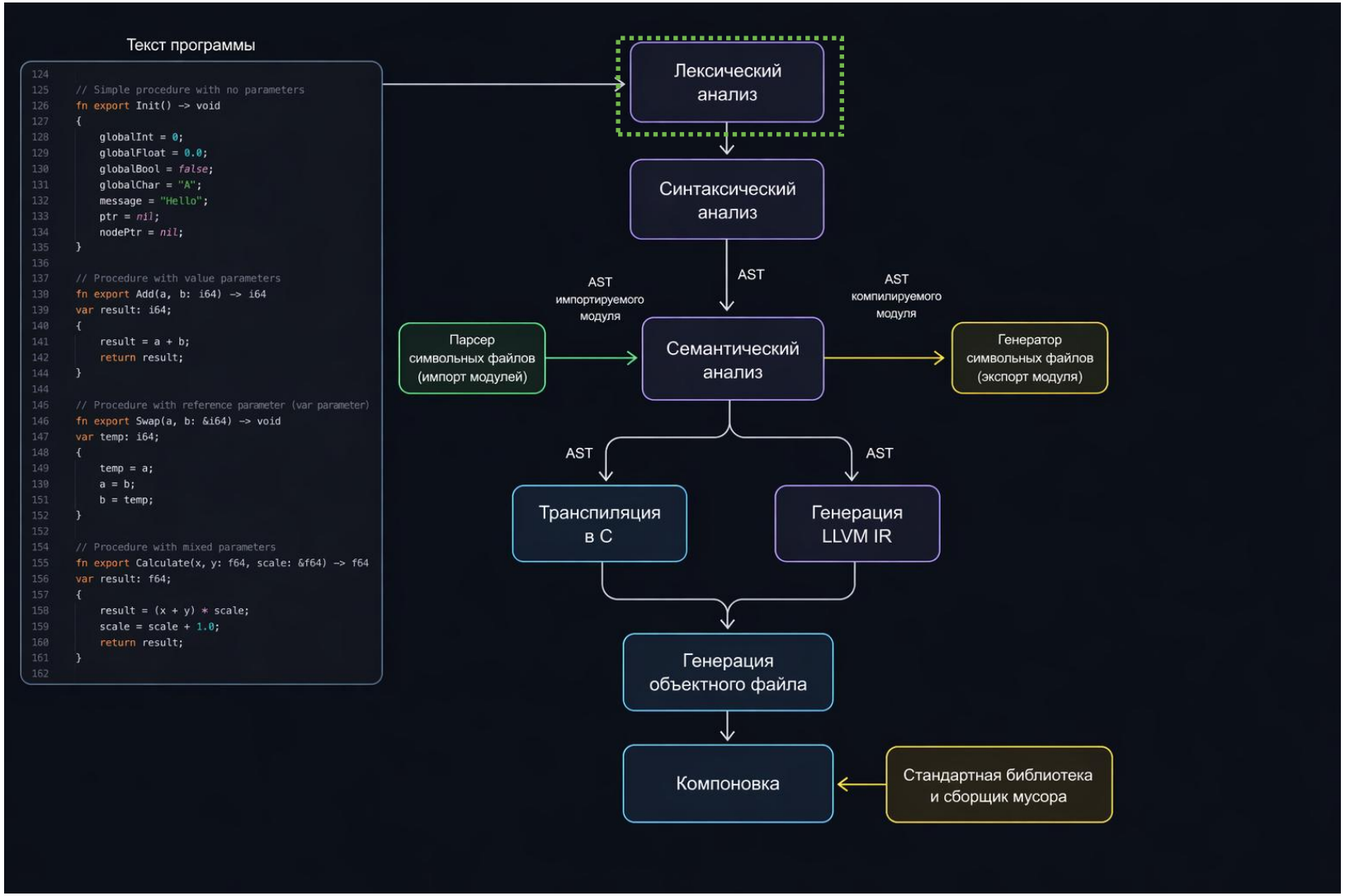
После разбора исходного кода на языке Обольд для передачи информации между модулями программы используется абстрактное синтаксическое дерево (AST).

Для передачи информации между единицами компиляции используются символьные файлы в формате JSON.



Лексер

Лексер разбивает исходный код на лексемы (токены), чтобы с ними работал парсер.



Лексер

Варианты реализации:

- Ручной конечный автомат
- Табличный DFA
- Регулярные выражения

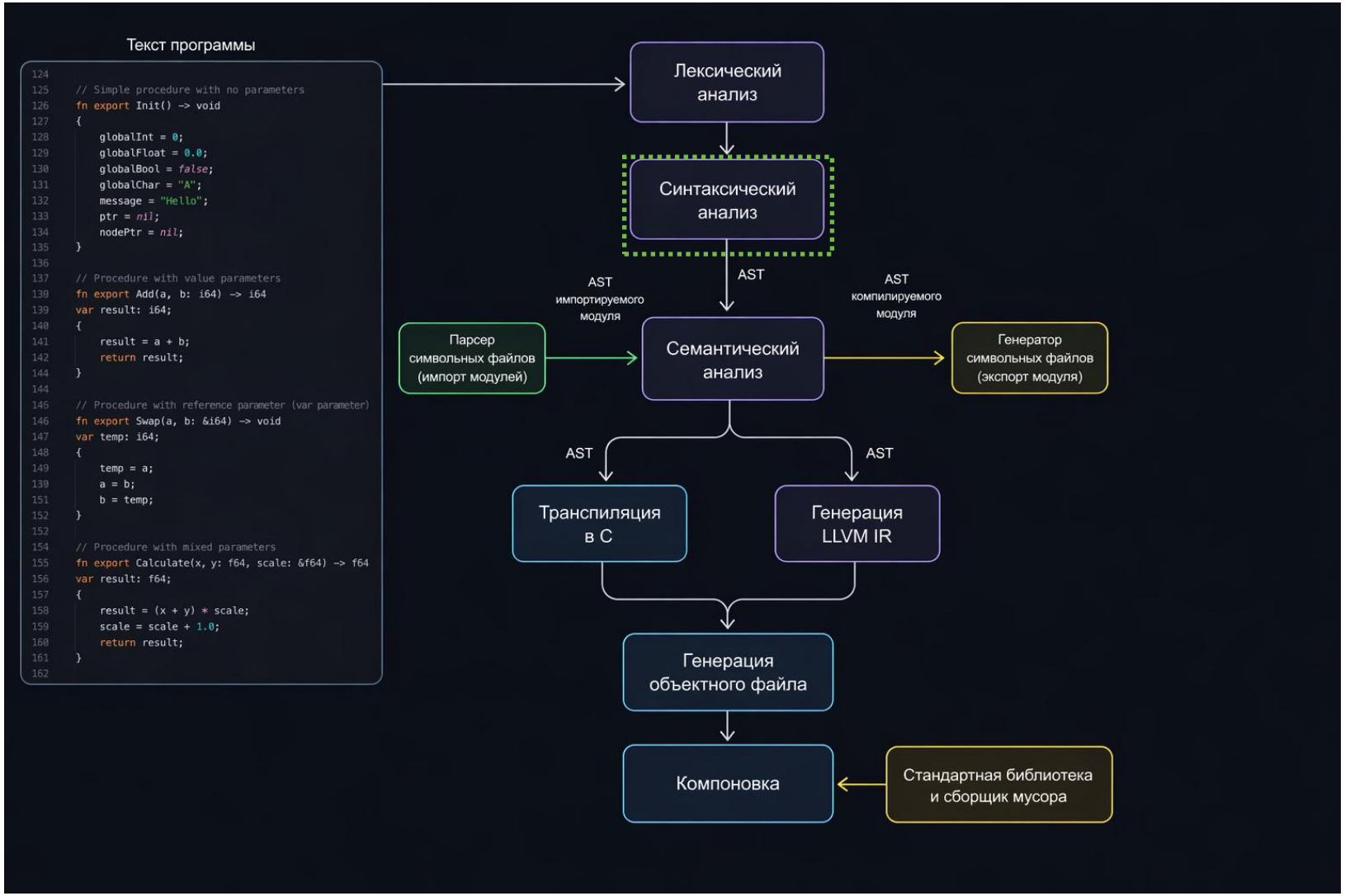
Выбор - ручной конечный автомат через проход с peek/advance.

Как в проекте - класс Lexer, метод tokenize(), типы Token/TokenType. При ошибке - токен Error и остановка пайплайна.

```
enum class TokenType {
    // Literals
    INTEGER,
    FLOAT,
    STRING,

    // Identifier
    IDENTIFIER,

    // Keywords
    KW_SWITCH,
    KW_CASE,
    KW_IF,
    KW_ELSE,
    KW_ELIF,
    KW_WHILE,
    KW_DO,
    KW_FOR,
    KW_FN,
    KW_MODULE,
    KW_RETURN,
    KW_VAR,
    KW_TYPE,
    KW_CONST,
    KW_TRUE,
    KW_FALSE,
    KW_IS,
    KW_NIL,
    KW_DIV,
    KW_MOD,
    KW_STRUCT,
    KW_EXPORT,
    KW_IMPORT,
    KW_VOID,
    KW_SET,
    KW_IN,
```

Парсер

Парсер по набору лексем строит абстрактное синтаксическое дерево, которое отражает структуру программы.



Парсер

Выбор

Варианты реализации:

- Рекурсивный спуск - каждый нетерминал = функция, просто читать и отлаживать, понятные сообщения об ошибках
- LL-парсер по таблице - явный стек + таблица переходов, генерируется из грамматики автоматически, нельзя левую рекурсию, хуже сообщения об ошибках

Парсер

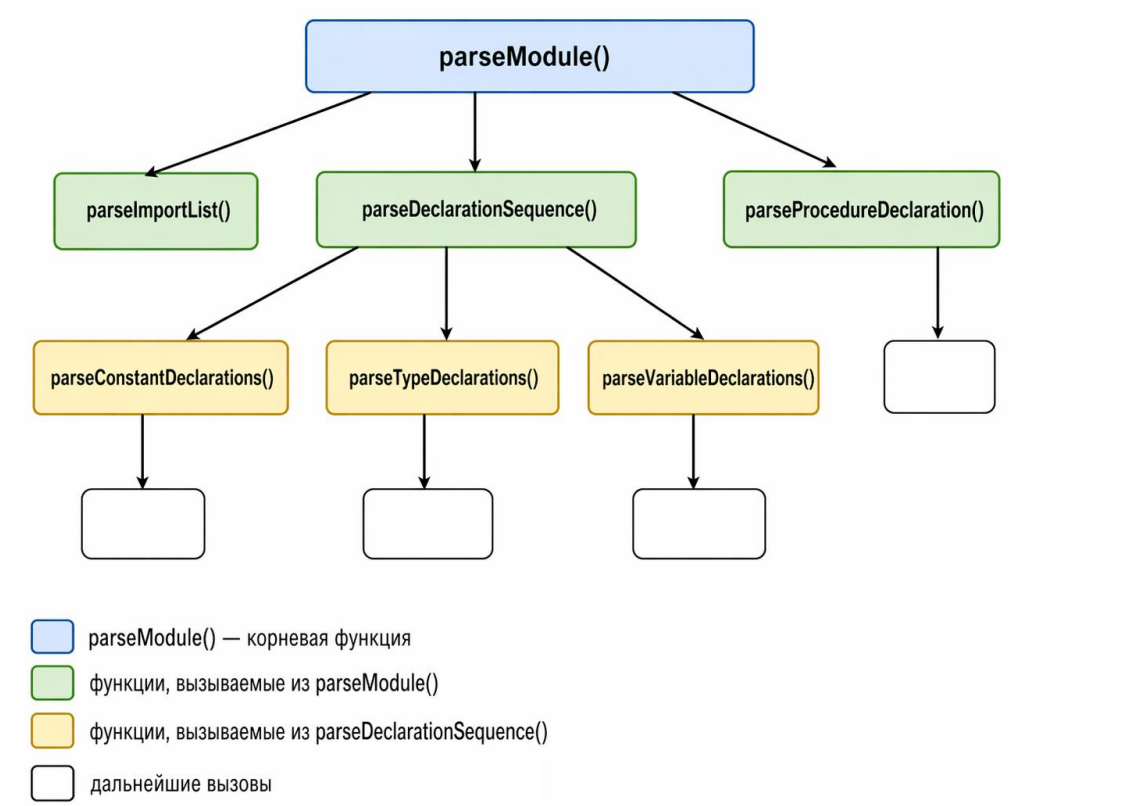


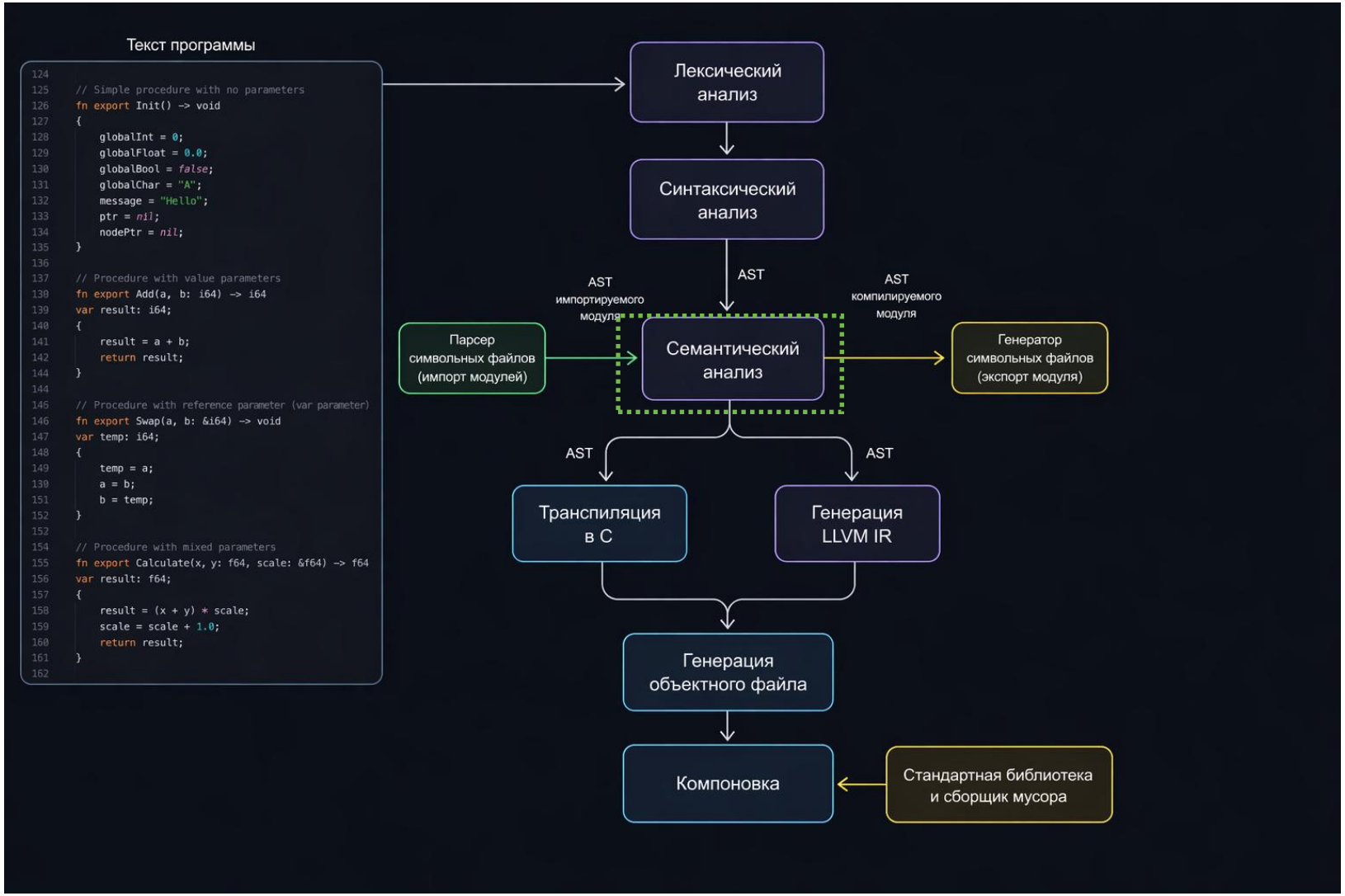
Схема рекурсивного спуска

```
module Math
├── Imports
│   └── Out
├── Declarations
└── Procedures
    ├── fn: factorial
    │   ├── Params
    │   │   └── : n
    │   │       └── i64
    │   ├── Returns
    │   │   └── i64
    │   ├── Declarations
    │   │   └── Variables
    │   │       └── variable: result
    │   │           └── i64
    │   └── Block
    │       └── If
    │           ├── BinaryExpr: <=
    │           │   ├── n
    │           │   └── 1 integer
    │           ├── Then
    │           │   └── Block
    │           │       └── Assign
    │           │           ├── result
    │           │           └── 1 integer
    │           └── Else
    │               └── Block
    │                   └── Assign
    │                       ├── result
    │                       └── BinaryExpr: *
    │                           ├── n
    │                           └── Call
    │                               ├── factorial
    │                               └── BinaryExpr: -
    │                                   ├── n
    │                                   └── 1 integer
    └── fn: init
        ├── Returns
        │   └── void
        ├── Declarations
        └── Block
            ├── Call
            │   ├── Out.Int
            │   └── Call
            │       └── factorial
```

Пример AST

Семантический анализ

На этапе семантического анализа проверяется смысловая корректность кода. В отличие от синтаксического анализа, который проверяет структуру программы, семантический анализ оценивает её смысловую логику: согласованность типов данных, корректность операций, правильное использование переменных и функций в контексте.

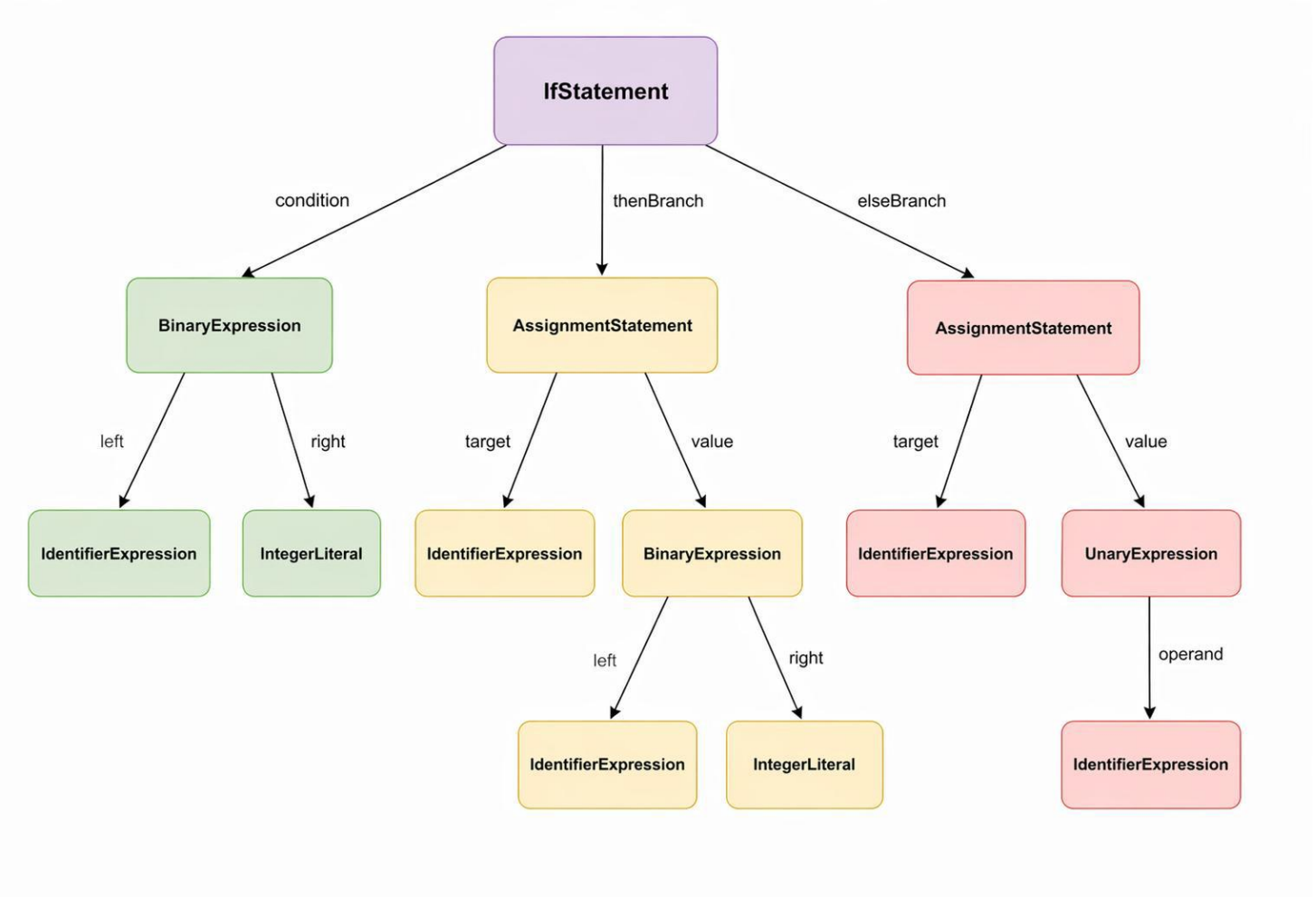


Семантический анализ

Абстрактное синтаксическое дерево обходится с использованием паттерна «Посетитель».

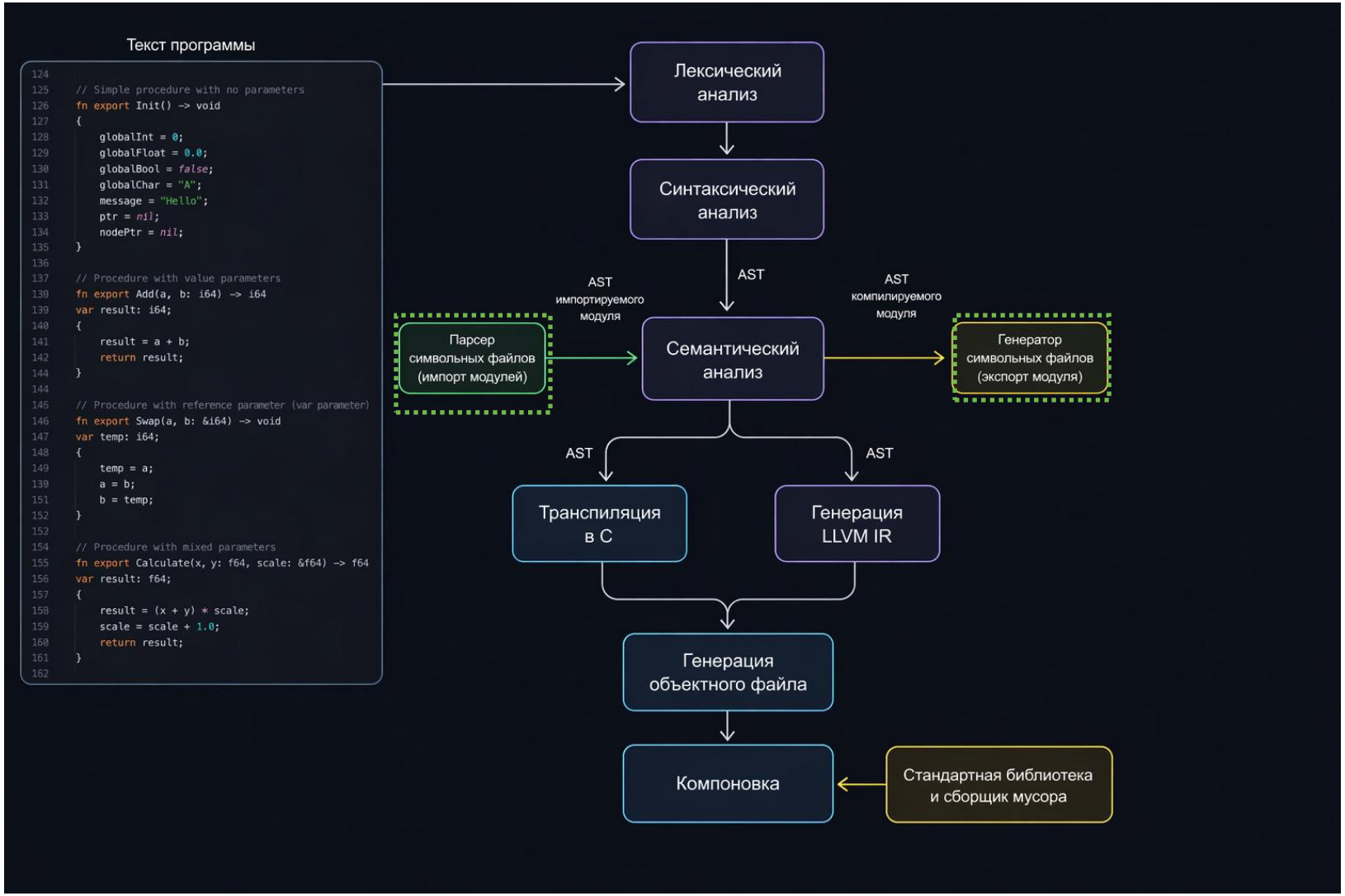
Во время обхода в узлах сохраняется информация о типах и значениях выражений.

Конструкции проверяются на соответствие правилам языка.



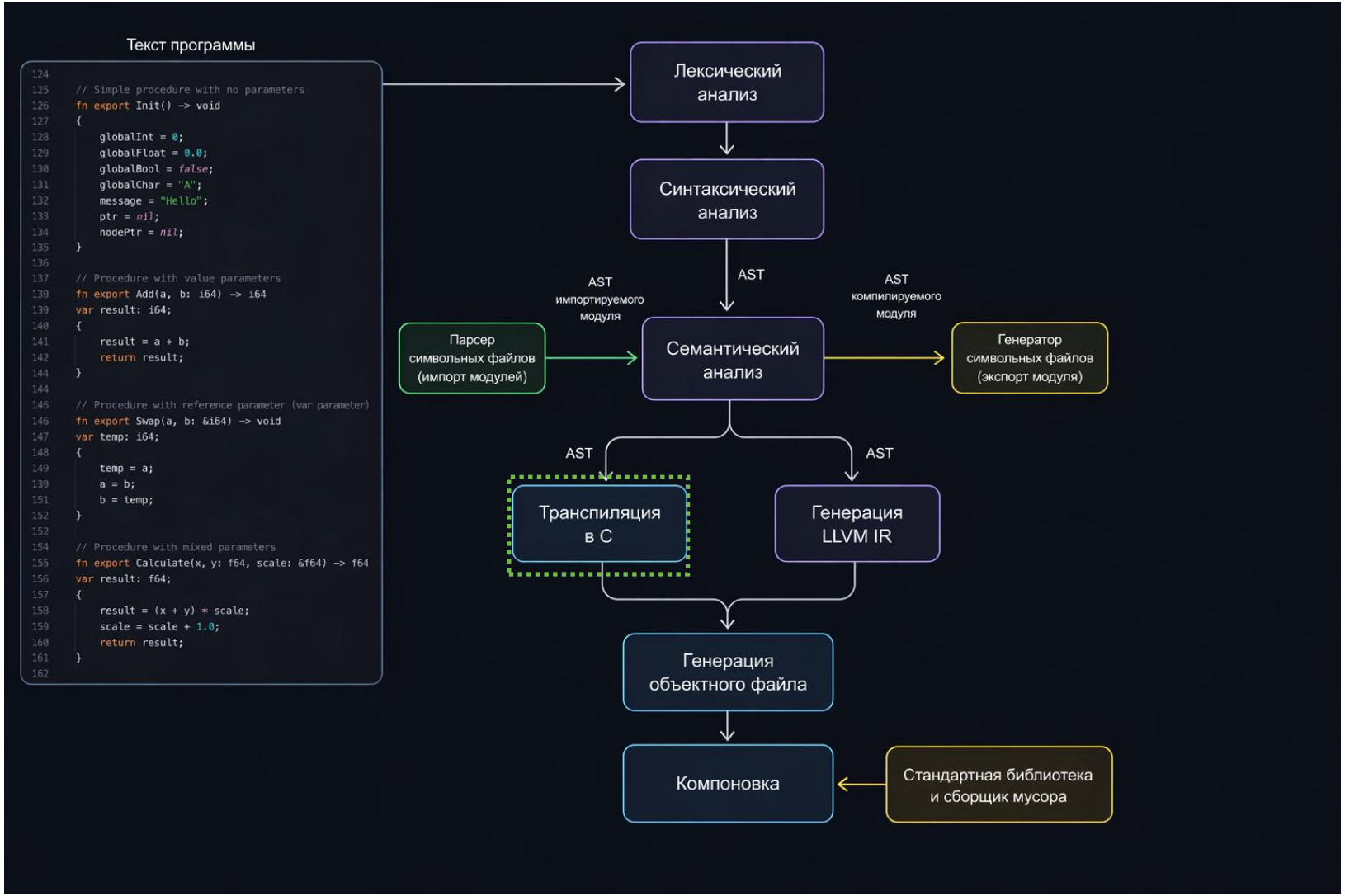
Символьные файлы

Парсер и генератор символьных файлов отвечают за чтение и создание JSON-файлов с описанием интерфейсов модулей и восстановление из них узлов абстрактного синтаксического дерева.



Транспиляция в Си

Зачем: получить переносимый Си-код и связку с существующими кодовыми базами.



```c
#include "FractalFlame.h"
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <gc.h>

/* Static function prototypes */
static void ob_12FractalFlame_init();

/* Function implementations */
static void ob_12FractalFlame_init() {
    ob_10FlameTypes_Config config;
    ob_6Images_Image image;
    bool ok;
    ok = ob_8FlameCli_LoadConfig(&(config));
    if (ok) {
        image = ob_13FlameRenderer_Render(config);
        if ((image == NULL)) {
            ob_3Out_String("Rendering failed.", 18);
            ob_3Out_Ln();
        } else {
            ok = ob_6Images_SavePNG(image, config.outputPath, 256);
            if (ok) {
                ob_3Out_String("Saved image to ", 16);
                ob_3Out_String(config.outputPath, 256);
                ob_3Out_Ln();
            } else {
                ob_3Out_String("Failed to save image to ", 25);
                ob_3Out_String(config.outputPath, 256);
                ob_3Out_Ln();
            };
        };
    };
}

extern void ob_4Args_4Args(void);
extern void ob_8FlameCli_8FlameCli(void);
extern void ob_13FlameRenderer_13FlameRenderer(void);
extern void ob_10FlameTypes_10FlameTypes(void);
extern void ob_6Images_6Images(void);
extern void ob_3Out_3Out(void);
void ob_12FractalFlame_12FractalFlame(void) {
    static int _initialized = 0;
    if (_initialized) return;
    _initialized = 1;
    ob_4Args_4Args();
    ob_8FlameCli_8FlameCli();
    ob_13FlameRenderer_13FlameRenderer();
```

Код на Си

```
module FractalFlame

import Args, FlameCli, FlameRenderer, FlameTypes, Images, Out;

fn init() -> void
var {
    config: FlameTypes.Config;
    image: Images.Image;
    ok: bool;
}
{
    ok = FlameCli.LoadConfig(config);
    if ok {
        image = FlameRenderer.Render(config);
        if image == nil {
            Out.String("Rendering failed."); Out.Ln();
        } else {
            ok = Images.SavePNG(image, config.outputPath);
            if ok {
                Out.String("Saved image to ");
                Out.String(config.outputPath);
                Out.Ln();
            } else {
                Out.String("Failed to save image to ");
                Out.String(config.outputPath);
                Out.Ln();
            }
        }
    }
}
```

Код на Обольд

Транспиляция в Си

Общий подход генерации кода:

- Один AST-визитор, два режима: CCodegenVisitor работает в режимах HEADER и SOURCE, поэтому из одного и того же AST генерируются согласованные .h и .c.
- HEADER генерирует интерфейс модуля: include guard, базовые #include, #include импортов, typedef/struct, объявления экспортируемых сущностей и прототипы процедур.
- SOURCE генерирует реализацию: подключает собственный .h, создаёт определения глобальных переменных/процедур, runtime-вспомогательные функции и код тел процедур.

Транспиляция в Си

Name mangling

- Формат префикса: ob_{len(module)}{module}_.
  Пример: Out.WriteLn -> ob_3Out_WriteLn.
- Это нужно, чтобы избежать коллизий имен между модулями и безопасно линковать символы в общей линковке.
- Применяется к глобальным переменным, константам, типам, процедурам и квалифицированным обращениям из импортов (Для локальных сущностей не нужен).

Транспиляция в Си

RTTI и перенос наследования в Си

- RTTI-метаданные: в header объявляется StructDescriptor (name, level, ancestors), а для каждого struct-типа создается глобальный дескриптор <Type>_desc
- Инициализация дескрипторов: в source генерируются helper-функции (createStructDescriptor, isInstanceOf) и _init_descriptors() с lazy single-init
- a is T транслируется в вызов isInstanceOf(actualDesc, &T_desc)
- Проверка идет через индексированный доступ по уровню наследования (ancestors[expected->level]). Если дескрипторы совпадают - результат истина. Если уровень экземпляра не глубже уровня проверяемого типа - результат ложь, иначе проверяется, совпадает ли предок экземпляра на уровне проверяемого типа с самим проверяемым типом

Генерация машинного кода

Из AST генерируется промежуточное представление LLVM IR.

Далее оно средствами LLVM преобразуется в объектный код.

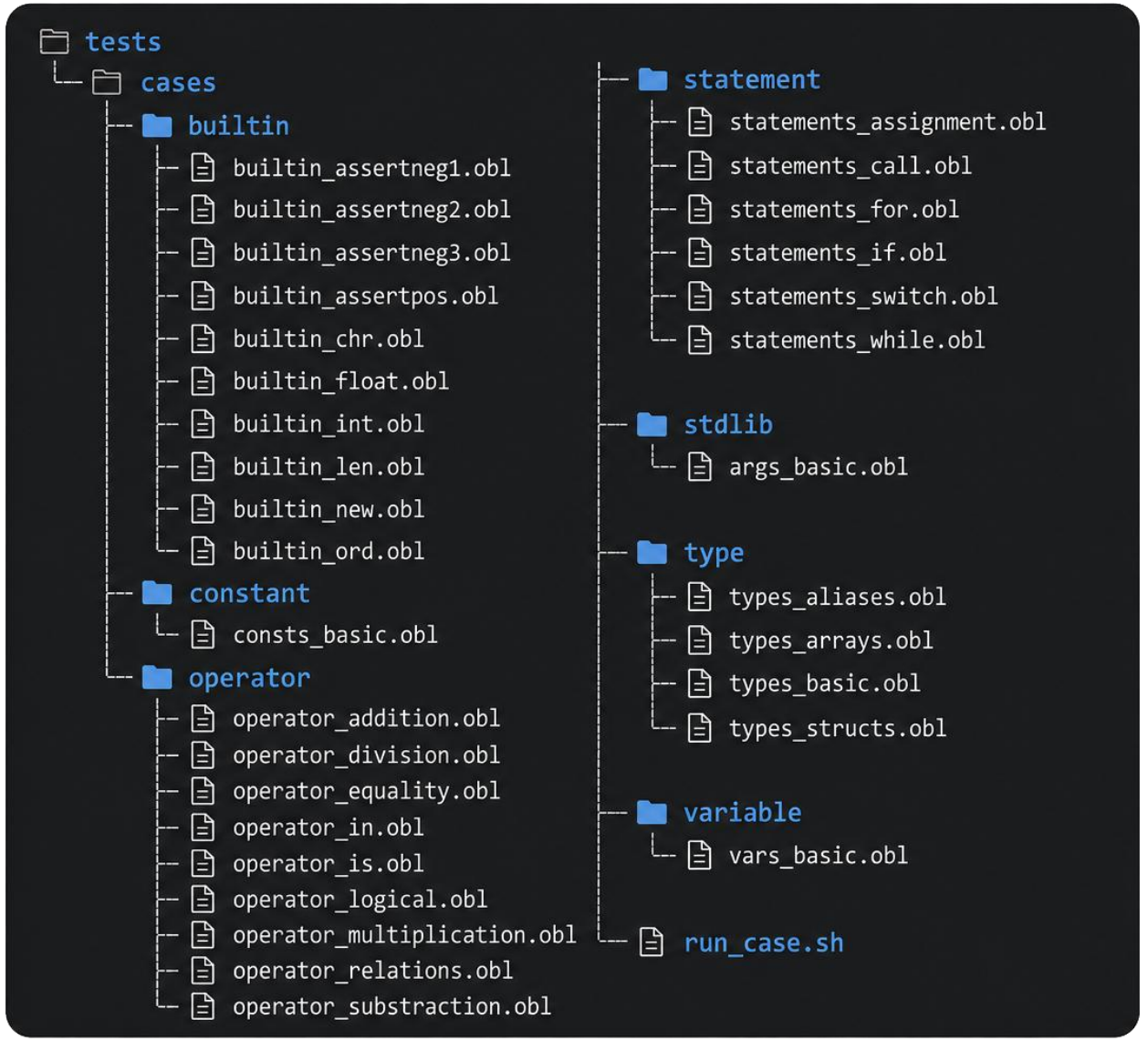
```
declare void @ob_3Out_Int(i64) local_unnamed_addr

declare void @ob_3Out_Ln() local_unnamed_addr

declare void @ob_3Out_3Out() local_unnamed_addr

; Function Attrs: nofree nosync nounwind memory(none)
define internal fastcc i64 @ob_4Math_factorial(i64 range(i64 1, 7) %n) unnamed_addr #0 {
entry:
  br label %tailrecurse

tailrecurse:                                      ; preds = %else, %entry
  %accumulator.tr = phi i64 [ 1, %entry ], [ %mul, %else ]
  %n.tr = phi i64 [ %n, %entry ], [ %sub, %else ]
  %lte = icmp samesign ult i64 %n.tr, 2
  br i1 %lte, label %merge, label %else

merge:                                            ; preds = %tailrecurse
  %accumulator.ret.tr = mul i64 %accumulator.tr, 1
  ret i64 %accumulator.ret.tr

else:                                             ; preds = %tailrecurse
  %sub = add nsw i64 %n.tr, -1
  %mul = mul i64 %accumulator.tr, %n.tr
  br label %tailrecurse
}

define void @ob_4Math_4Math() local_unnamed_addr {
entry:
  %0 = load i1, ptr @ob_Math_visited, align 1
  br i1 %0, label %common.ret, label %first

common.ret:                                       ; preds = %entry, %first
  ret void

first:                                            ; preds = %entry
  store i1 true, ptr @ob_Math_visited, align 1
  tail call void @ob_3Out_3Out()
  %1 = tail call fastcc i64 @ob_4Math_factorial(i64 6)
  tail call void @ob_3Out_Int(i64 %1)
  tail call void @ob_3Out_Ln()
  br label %common.ret
}

attributes #0 = { nofree nosync nounwind memory(none) }
```

```
.obould/factorial.o:    file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <ob_4Math_factorial>:
       0: b8 01 00 00 00               movl    $0x1, %eax
       5: 48 83 ff 01                  cmpq    $0x1, %rdi
       9: 76 12                        jbe     0x1d <ob_4Math_factorial+0x1d>
       b: 0f 1f 44 00 00               nopl    (%rax,%rax)
      10: 48 0f af c7                  imulq   %rdi, %rax
      14: 48 ff cf                     decq    %rdi
      17: 48 83 ff 01                  cmpq    $0x1, %rdi
      1b: 77 f3                        ja      0x10 <ob_4Math_factorial+0x10>
      1d: c3                           retq
      1e: 66 90                        nop

0000000000000020 <ob_4Math_4Math>:
      20: 80 3d 00 00 00 00 01         cmpb    $0x1, (%rip)            # 0x27 <ob_4Math_4Math+0x7>
      27: 75 01                        jne     0x2a <ob_4Math_4Math+0xa>
      29: c3                           retq
      2a: 50                           pushq   %rax
      2b: c6 05 00 00 00 00 01         movb    $0x1, (%rip)            # 0x32 <ob_4Math_4Math+0x12>
      32: e8 00 00 00 00               callq   0x37 <ob_4Math_4Math+0x17>
      37: bf 06 00 00 00               movl    $0x6, %edi
      3c: e8 bf ff ff ff               callq   0x0 <ob_4Math_factorial>
      41: 48 89 c7                     movq    %rax, %rdi
      44: e8 00 00 00 00               callq   0x49 <ob_4Math_4Math+0x29>
      49: 58                           popq    %rax
      4a: e9 00 00 00 00               jmp     0x4f <ob_4Math_4Math+0x2f>
      4f: 90                           nop

0000000000000050 <main>:
      50: 80 3d 00 00 00 00 00         cmpb    $0x0, (%rip)            # 0x57 <main+0x7>
      57: 75 28                        jne     0x81 <main+0x31>
      59: 50                           pushq   %rax
      5a: c6 05 00 00 00 00 01         movb    $0x1, (%rip)            # 0x61 <main+0x11>
      61: e8 00 00 00 00               callq   0x66 <main+0x16>
      66: bf 06 00 00 00               movl    $0x6, %edi
      6b: e8 90 ff ff ff               callq   0x0 <ob_4Math_factorial>
      70: 48 89 c7                     movq    %rax, %rdi
      73: e8 00 00 00 00               callq   0x78 <main+0x28>
      78: e8 00 00 00 00               callq   0x7d <main+0x2d>
      7d: 48 83 c4 08                  addq    $0x8, %rsp
      81: 31 c0                        xorl    %eax, %eax
      83: c3                           retq
```

Тестирование программы

Для проверки работы компилятора написаны тестовые программы, покрывающие все языковые конструкции.

```
TEST_CASE("Local variables and assignment", "[ccodegen][var]")
{
    std::string src = R"(module Vars
import Out;
fn init() -> void
var {
    a: i64;
    b: i64;
    c: i64;
}
{
    a = 10;
    b = 20;
    c = a + b;
    Out.Int(c);
    Out.Ln();
}
)";
    REQUIRE(transpileCompileRun("Vars", src, {"Out"}) == "30\n");
}
```



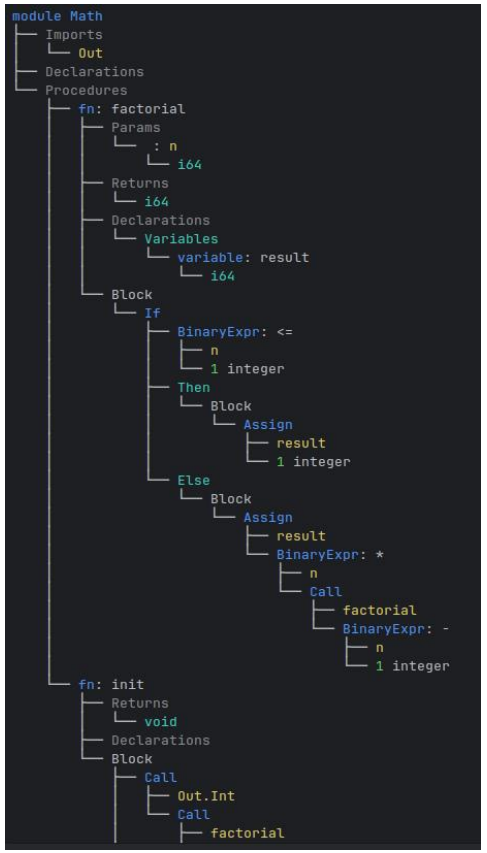
Результаты

1) Разработан синтаксис языка Обольд
2) Определён стек используемых технологий
3) Реализован лексер
4) Реализован парсер
5) Реализован семантический анализ
6) Реализована работа с символьными файлами
7) Реализован транспилятор в Си
8) Реализована генерация машинного кода и объектных файлов
9) Написаны тестовые и демонстрационные программы на языке Обольд

Общее
Константин Семён

Демонстрация

```
module Math
import Out;

fn factorial(n: i64) -> i64
var result: i64;
{
    if n <= 1 {
        result = 1;
    } else {
        result = n * factorial(n - 1);
    }
    return result;
}

fn init() -> void {
    Out.Int(factorial(6));
    Out.Ln();
}
```




```
  store i64 %n, ptr %n1, align 4
  store i64 0, ptr %result, align 4
  %0 = load i64, ptr %n1, align 4
  %lte = icmp sle i64 %0, 1
  br i1 %lte, label %then, label %else

then:                                             ; preds = %entry
  store i64 1, ptr %result, align 4
  br label %merge

merge:                                            ; preds = %else, %then
  %1 = load i64, ptr %result, align 4
  ret i64 %1

else:                                             ; preds = %entry
  %2 = load i64, ptr %n1, align 4
  %3 = load i64, ptr %n1, align 4
  %sub = sub i64 %3, 1
  %4 = call i64 @ob_4Math_factorial(i64 %sub)
  %mul = mul i64 %2, %4
  store i64 %mul, ptr %result, align 4
  br label %merge
}

define internal void @ob_4Math_init() {
entry:
  %0 = call i64 @ob_4Math_factorial(i64 6)
  call void @ob_3Out_Int(i64 %0)
  call void @ob_3Out_Ln()
  ret void
}

define void @ob_4Math_4Math() {
entry:
  %0 = load i1, ptr @ob_Math_visited, align 1
  br i1 %0, label %other, label %first

first:                                            ; preds = %entry
  store i1 true, ptr @ob_Math_visited, align 1
  call void @ob_3Out_3Out()
  call void @ob_4Math_init()
  ret void

other:                                            ; preds = %entry
  ret void
}
```

```
.obould/factorial.o:    file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <ob_4Math_factorial>:
       0: b8 01 00 00 00              movl    $0x1, %eax
       5: 48 83 ff 01                 cmpq    $0x1, %rdi
       9: 76 12                       jbe     0x1d <ob_4Math_factorial+0x1d>
       b: 0f 1f 44 00 00              nopl    (%rax,%rax)
      10: 48 0f af c7                 imulq   %rdi, %rax
      14: 48 ff cf                    decq    %rdi
      17: 48 83 ff 01                 cmpq    $0x1, %rdi
      1b: 77 f3                       ja      0x10 <ob_4Math_factorial+0x10>
      1d: c3                          retq
      1e: 66 90                       nop

0000000000000020 <ob_4Math_4Math>:
      20: 80 3d 00 00 00 00 01        cmpb    $0x1, (%rip)            # 0x27 <ob_4Math_4Math+0x7>
      27: 75 01                       jne     0x2a <ob_4Math_4Math+0xa>
      29: c3                          retq
      2a: 50                          pushq   %rax
      2b: c6 05 00 00 00 00 01        movb    $0x1, (%rip)            # 0x32 <ob_4Math_4Math+0x12>
      32: e8 00 00 00 00              callq   0x37 <ob_4Math_4Math+0x17>
      37: bf 06 00 00 00              movl    $0x6, %edi
      3c: e8 bf ff ff ff              callq   0x0 <ob_4Math_factorial>
      41: 48 89 c7                    movq    %rax, %rdi
      44: e8 00 00 00 00              callq   0x49 <ob_4Math_4Math+0x29>
      49: 58                          popq    %rax
      4a: e9 00 00 00 00              jmp     0x4f <ob_4Math_4Math+0x2f>
      4f: 90                          nop

0000000000000050 <main>:
      50: 80 3d 00 00 00 00 00        cmpb    $0x0, (%rip)            # 0x57 <main+0x7>
      57: 75 28                       jne     0x81 <main+0x31>
      59: 50                          pushq   %rax
      5a: c6 05 00 00 00 00 01        movb    $0x1, (%rip)            # 0x61 <main+0x11>
      61: e8 00 00 00 00              callq   0x66 <main+0x16>
      66: bf 06 00 00 00              movl    $0x6, %edi
      6b: e8 90 ff ff ff              callq   0x0 <ob_4Math_factorial>
      70: 48 89 c7                    movq    %rax, %rdi
      73: e8 00 00 00 00              callq   0x78 <main+0x28>
      78: e8 00 00 00 00              callq   0x7d <main+0x2d>
      7d: 48 83 c4 08                 addq    $0x8, %rsp
      81: 31 c0                       xorl    %eax, %eax
      83: c3                          retq
```

Направления дальнейшей работы

1) Детализация обработки ошибок. В частности, указание на место ошибки в файле.
2) Внедрение динамического полиморфизма процедурно-параметрической парадигмы программирования.
3) Изменения синтаксиса языка для повышения удобочитаемости исходного кода.
4) Расширение стандартной библиотеки языка.


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •