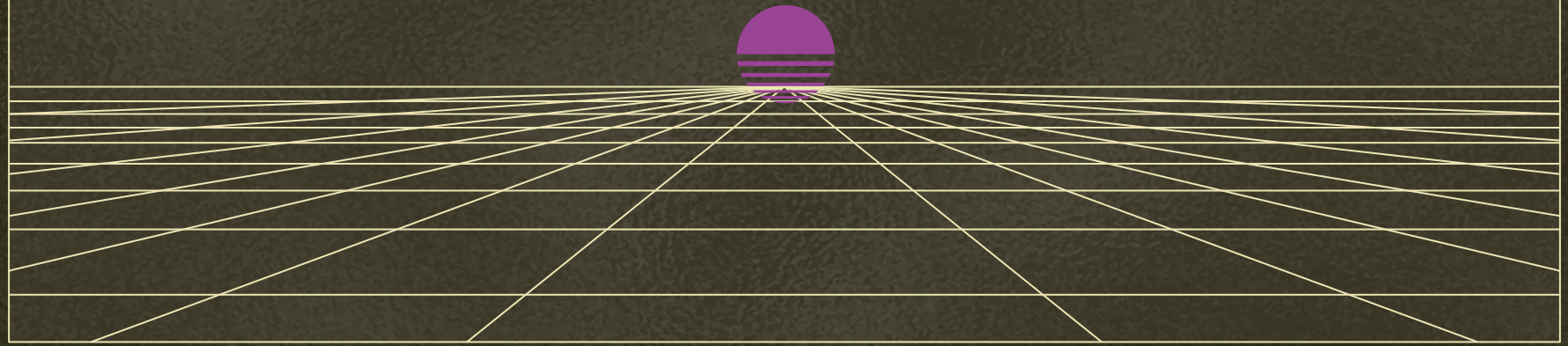
COMPILERS CONSTRUCTION

Report 4
Compilation



COMPILUL'KI


Gleb
Bugaev
Nail
Minnemulin
Dmitriy
Ekoneshnikov
Milana
Sirozhova

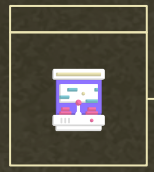
Technology stack

Project: Object-Oriented

Target platform: LLVM

Implementation lang/tool: C++, handwritten parser, CMake to build the project

Target language: LLVM bit-code

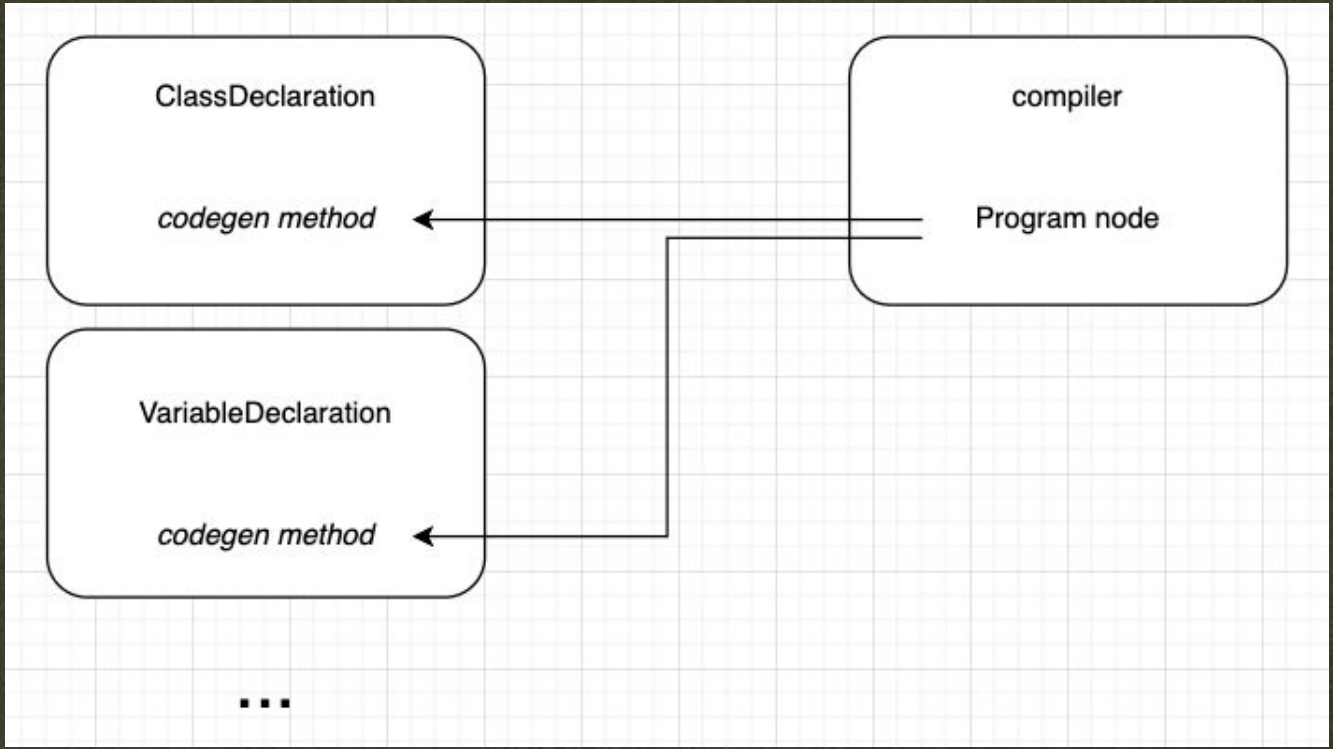


Task distribution

- Nail — LLVM logic
- Dima — Code fixing
- Gleb — LLVM code for standard library
- Milana — Presentation

Code generation logic

Code generation overall



Class Node Code Generation

The codegen function for a class node creates a custom LLVM type that encapsulates all fields of the class.

If a class is derived from a parent class, it will also contain all inherited fields and methods.

Furthermore, if the derived class does not override a method from the base class, this method is copied to the derived class.

Variable Declaration Code Generation

The codegen for a variable declaration involves allocating memory using malloc.
After allocation, it invokes the constructor and passes a pointer to the newly created object.

Method Declaration Code Generation

The codegen function for a method declaration creates a function with a specified signature. It also supports polymorphism to handle method variations.

Examples for method Declaration

```
class Main is
    method test(a: Integer): Integer is
        return a
    end
end
```

will be stored as
Main_test_Integer

```
class Main is
    method test(a: Integer, b: Real):
Integer is
        return a
    end
end
```

will be stored as
Main_test_Integer_Real

Simple example

```
Program: Main()

class Main is
    this() is
        var a : Integer(5)
        var b : Integer(10)
        var sum : a.Plus(b)
        sum.print()
    end

end
```

```
define void @Main_Constructor(ptr %this) {
entry:
  call void @Main_Init(ptr %this)
  %mallocCall = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall, i32 5)
  %mallocCall1 = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall1, i32 10)
  %loaded_arg = load %Integer, ptr %mallocCall1, align 4
  %call_Plus = call %Integer @Integer_Plus_Integer(ptr %mallocCall, %Integer %loaded_arg)
  %alloca_return_val = alloca %Integer, align 8
  store %Integer %call_Plus, ptr %alloca_return_val, align 4
  call void @Integer_print(ptr %alloca_return_val)
  ret void
}
```

While loop

```
Program: Main()

class Main is
    this() is

        var a : Integer
        var b : Integer

        a.scan()
        b.scan()

        while a.Greater(0) loop
            b := b.Plus(1)
            a := a.Minus(1)
        end

        b.print()

    end

end
```

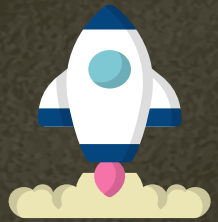
```
define void @Main_Constructor(ptr %this) {
entry:
  call void @Main_Init(ptr %this)
  %mallocCall = call ptr @malloc(i64 1)
  call void @Boolean_Constructor_Boolean(ptr %mallocCall, i1 true)
  %mallocCall1 = call ptr @malloc(i64 1)
  call void @Boolean_Constructor_Boolean(ptr %mallocCall1, i1 false)
  %boolFieldPtr = getelementptr inbounds %Boolean, ptr %mallocCall, i32 0, i32 0
  %loadBoolValue = load i1, ptr %boolFieldPtr, align 1
  %ifcond = icmp ne i1 %loadBoolValue, false
  br i1 %ifcond, label %btrue, label %bfalse

btrue:                                            ; preds = %entry
  call void @Boolean_print(ptr %mallocCall1)
  br label %end

bfalse:                                           ; preds = %entry
  call void @Boolean_print(ptr %mallocCall)
  br label %end

end:                                              ; preds = %bfalse, %btrue
  ret void
}
```

Deriving

```
define void @Base_show(ptr %this) {
entry:
  %mallocCall = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall, i32 1)
  call void @Integer_print(ptr %mallocCall)
  ret void
}

define void @Base_foo(ptr %this) {
entry:
  %mallocCall = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall, i32 3)
  call void @Integer_print(ptr %mallocCall)
  ret void
}

define void @Base_Constructor(ptr %0) {
entry:
  call void @Base_Init(ptr %0)
  ret void
}

define void @Derived_Init(ptr %0) {
entry:
  call void @Base_Init(ptr %0)
  ret void
}

define void @Derived_show(ptr %this) {
entry:
  %mallocCall = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall, i32 2)
  call void @Integer_print(ptr %mallocCall)
  ret void
}
```

```
define void @Derived_bar(ptr %this) {
entry:
  %mallocCall = call ptr @malloc(i64 4)
  call void @Integer_Constructor_Integer(ptr %mallocCall, i32 4)
  call void @Integer_print(ptr %mallocCall)
  ret void
}

define void @Derived_Constructor(ptr %0) {
entry:
  call void @Derived_Init(ptr %0)
  ret void
}

define void @Derived_foo(ptr %0) {
entry:
  call void @Base_foo(ptr %0)
  ret void
}
```

Examples



GitHub

THANKS!