CLP Project Group 007

Methods and C

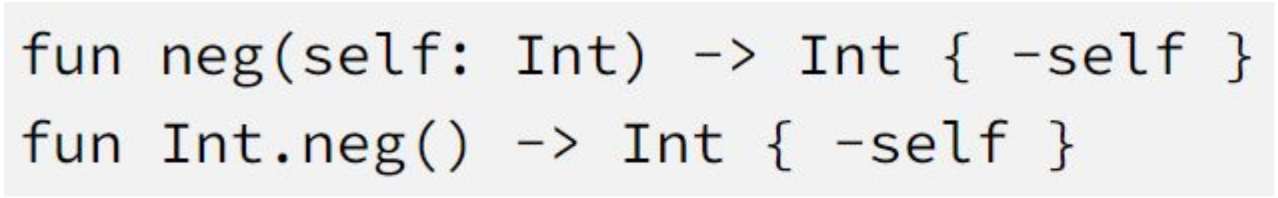
Noah Robert - Sirapat Charukamnertkanok - Sydney Hauke

What we implemented

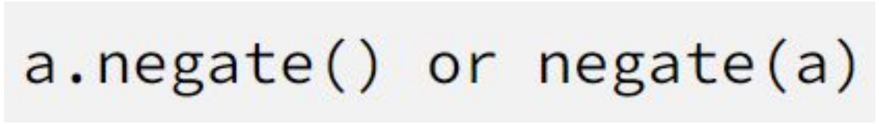
- Methods
- C transpiler

Methods as free functions

- Methods can be used as free functions.
- Free functions can be used as methods as long as we use “self” as the variable name.
- Possible because Methods are Functions in the parser.
- No need to change anything in the typer or any other file.



```
fun neg(self: Int) -> Int { -self }
fun Int.neg() -> Int { -self }
```



```
a.negate() or negate(a)
```

Methods - challenges encountered

- Methods can only be declared on basic types for now
- Declaring a method on a record type doesn’t work
- We can still declare it as a free function and call it as a method

Methods - challenges encountered

- Methods cannot be called on a literal

```
3.add(4) doesn’t work
```

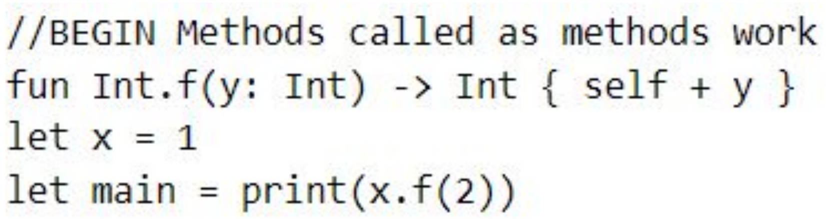
- we have to instantiate a variable

```
let x = 3
x.add(4)
```

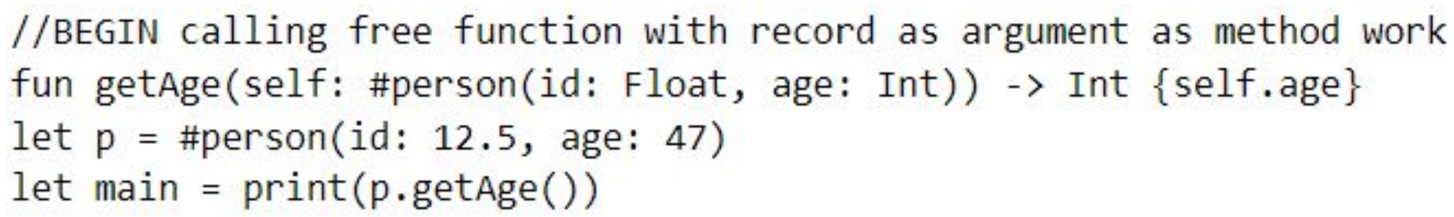
Methods - testing

tests of the parser

tests of the code generator



```
//BEGIN Methods called as methods work
fun Int.f(y: Int) -> Int { self + y }
let x = 1
let main = print(x.f(2))
```



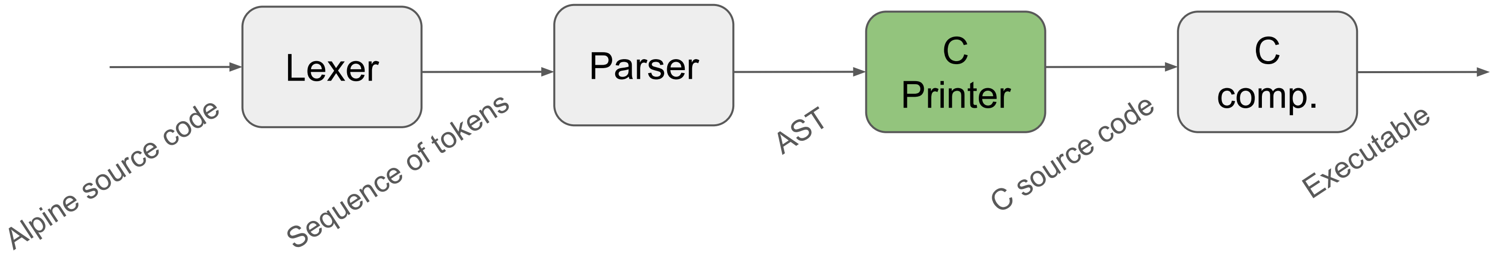
```
//BEGIN calling free function with record as argument as method work
fun getAge(self: #person(id: Float, age: Int)) -> Int {self.age}
let p = #person(id: 12.5, age: 47)
let main = print(p.getAge())
```

Methods - example of working code

```
fun Int.add(other: Int) -> Int { self + other }
fun Int.square() -> Int { self * self }
fun sub(self: Int, other: Int) -> Int { self - other }
let x = 1
let main = print(square(self: x.add(2).sub(3)))
```

C transpiler

The process of transpiling to C is essentially the same as in the Scala Printer we previously worked on.

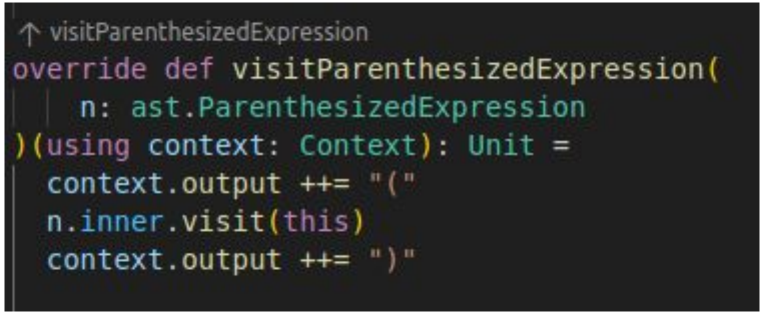


C transpiler - supported Alpine constructs

Alpine constructs	Support by the C transpiler
Conditionals	Full
Bindings	Full
Ascriptions	Buggy
Let expressions	None
Applications	Full (regular functions)
Infix/Prefix applications	Full
Lambdas	None
Records	Partial (no support for nested records)
Pattern matching	Partial (no support for records)
Field selection in records	Full

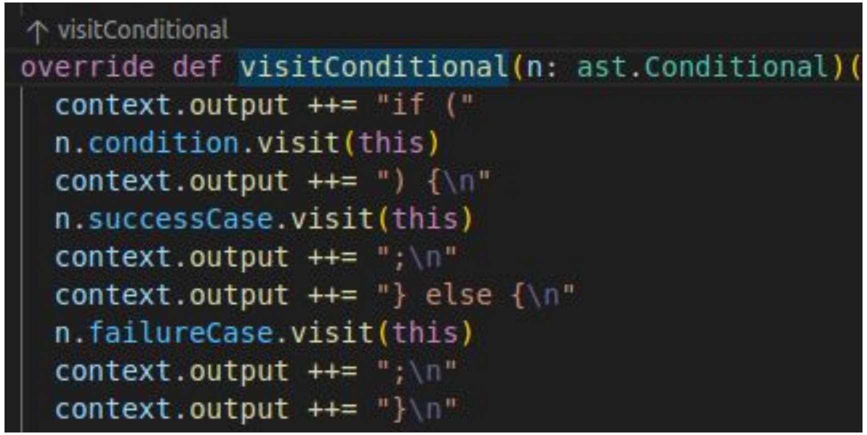
C transpiler - principle

- Print the resulting C program on-the-fly while traversing/visiting the AST
- But, some Alpine constructs cannot be translated trivially in C (records, functions, pattern matching…) !



```
↑ visitParenthesizedExpression
override def visitParenthesizedExpression(
    n: ast.ParenthesizedExpression
)(using context: Context): Unit =
  context.output ++= "("
  n.inner.visit(this)
  context.output ++= ")"
```

Trivial transpilation of parenthesized expression



```
↑ visitConditional
override def visitConditional(n: ast.Conditional)(
  context.output ++= "if ("
  n.condition.visit(this)
  context.output ++= ") {\n"
  n.successCase.visit(this)
  context.output ++= ";\n"
  context.output ++= "} else {\n"
  n.failureCase.visit(this)
  context.output ++= ";\n"
  context.output ++= "}\n"
```

Trivial transpilation of conditionals

C transpiler - transpiling records

Records are very similar in nature with C structs.

- C structs cannot be defined inline as with records in Alpine
- (Standard) C is sensitive to the order of struct definitions ! => non trivial problems arise when dealing with nested records

C transpiler - pattern matching

- Similar to the switch-case construct in C but the latter is primitive compared to pattern matching.

Pattern matching on fixed values of primitive types are straightforward in C

Simulating pattern matching on composite types are an other story…

- Pattern matching on primitive types :

Option 1 : Box primitives in their own struct

Option 2 : Use _Generics in C11

C transpiler - ascription

Alpine :

- (Un)conditional narrowing
- Widening

C :

- implicit conversions
- explicit conversions
- most importantly, no runtime checks inserted by the C compiler

Questions ?