Improved String Support

Improve string support for the Amy compiler

By Suhas Shankar and Edouard Michelin - Group 07

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

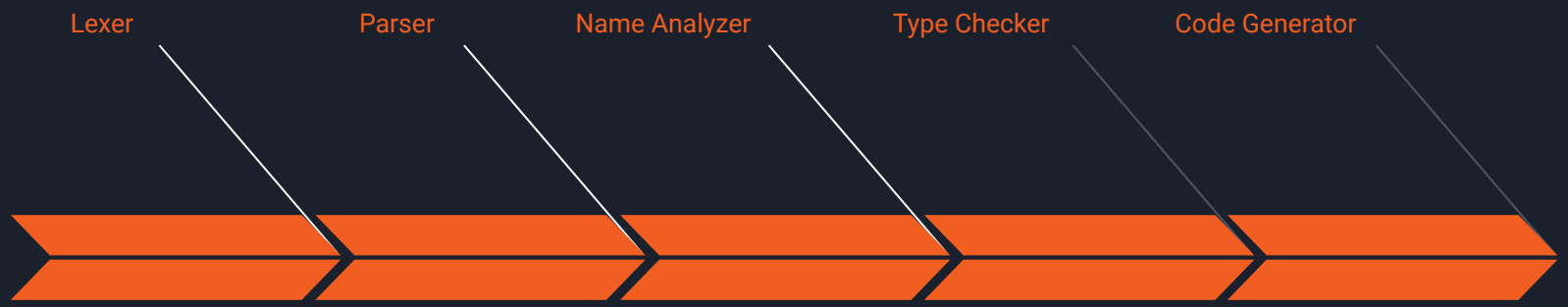
Summary

- Types
  - Char
- Built-in functions
  - StringImpl
    - substring(input: String, start: Int(32), end: Int(32)): String
    - length(input: String): Int(32)
    - replace(input: : String, c1: Char, c2: Char): String
    - strip(input: String): String
    - toLowerCase(input: String): String — toUpperCase(input: String): String
    - startsWith(input: String, search: String): Boolean — endsWith(input: String, search: String): Boolean
    - indexOf(input: String, search: Char): Int(32)
    - charAt(input: String: index: Int(32)): Char
    - split(input: String, separator: Char): List[String]
    - toCharArray(input: String): List[Char]
  - Std
    - printChar(input: Char): Unit
    - charToString(input: Char): String

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

Added Type — Char

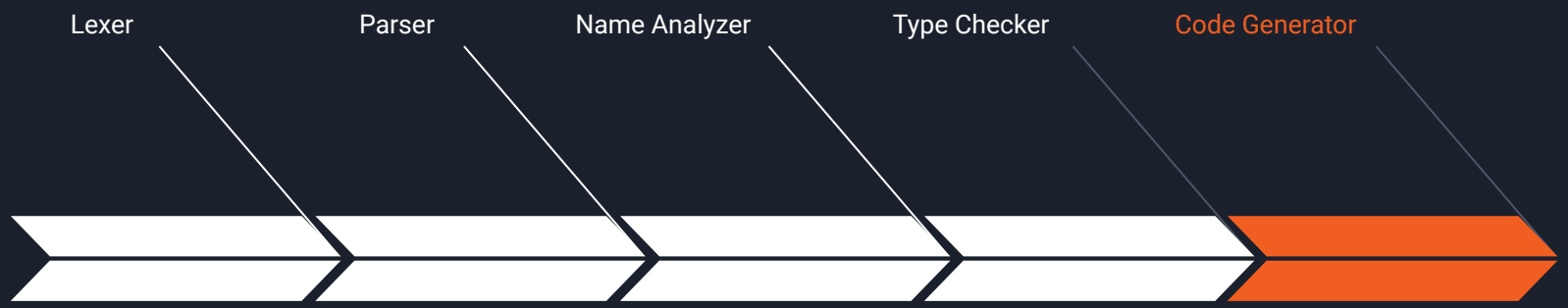


Lexer	Parser	Name Analyzer	Type Checker	Code Generator
parsing/Lexer.scala parsing/Tokens.scala	ast/TreeModule.scala ast/Printer.scala parsing/Parser.scala	analyzer/NameAnalyzer.scala	analyzer/TypeChecker.scala	codegen/CodeGen.scala
Add “Char” to the primitive type names.	Addition of type CharType and literal expression CharLiteral.	Add support for Char in step 3 (type discovering) and step 6 (module reconstruction).	Generate constraint for Char literals.	Push the char code (ASCII) on the stack.
Addition of the new token CharLitToken.	Add support in the printer.	Modifications were trivial: - no name, only literal values		
Addition of the pattern to match CharLitToken:	Add “Char” to the list of primitive types.			
(')(([^\\\n']) | ([\\][^\n]))(') ⇒ '{Char}' or '\{Char}'	Parse token CharLitToken to expression CharLiteral.			

Phase was modified

Phase was NOT modified

Added Built-in Functions



wasm/Module.scala
codegen/utils.scala

Functions that need access to global memory bound were written in codegen/utils.scala eg: toLowerCase

Functions that don't need access were added to the JS wrapper eg: length

Phase was modified

Phase was NOT modified

Added Built-in Functions

Why?

- Human interaction is mainly with Strings
- Optimising these operations results in huge speedup

Why these functions?

- Personal choice based on our experience
- Among multiple website's list of important string methods (ex: MDN, crio.do)

Built-in Functions

substring(

input: String,

start: Int(32),

end: Int(32)

): String

replace(

input: String,

searchValue: Char,

replaceValue: Char

): String

```
object StringOps

  Std.printString("Length of string <abcdefsg> ");
  Std.printInt(StringImpl.length("abcdefsg"));

  Std.printString("Replace a with b in  string <abcdefsaga> ");
  Std.printString(StringImpl.replace("abcdefsaga", 'a' , 'b'));

  Std.printString("Strip whitespace   string <  abcdefsg  > ");
  Std.printString(StringImpl.strip("  abcdefsg  ") );

  Std.printString("To lower case :   string <ABCaa234__Ssc> ");
  Std.printString( StringImpl.toLowerCase("ABCaa234__Ssc") )

end StringOps
```

Built-in Functions

strip(

input: String,

): String

toLowerCase(

input: String

): String

Console Output

```
Length of string <abcdefsg>
8
Replace a with b in  string <abcdefsaga>
bbcdefsbgb
Strip whitespace   string <  abcdefsg  >
abcdefsg
To lower case :   string <ABCaa234__Ssc>
abcaa234__ssc
```

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

Future improvements / improvement ideas

- Support for new-line character and other escape sequence.
- Perform optimisations for concatenation of multiple strings (Perhaps something similar to stringBuilder)

Plan

1. Introduction
2. Overview of the added features
3. What? Why? Where? How? And examples!
4. Future improvements
5. Conclusion

Thank you!

Merci !

ಧನ್ಯವಾದ!

```
object Presentation

  val isPresentationDone: Boolean = true;

  val thankYouMessage: String = "Thank you for listening!";

  if (isPresentationDone) {
    Std.printString(thankYouMessage)
  }

end Presentation
```

Questions?