Compiler

Bachelor Project

Aarhus Institut for Elektro- og Computerteknologi

Authors:
Sune Andreas Dyrbye, 201205948
Morten Høgsberg, 201704542

Supervisor:

Anslag:

Afleveringsdato: Eksamineringsdato:

Resumé

Contents

1 Abbreviations

Abbreviations

Table 1

Abbreviation	Meaning
CLI	Command Line Interface
AST	Abstract Syntax tree
IR	Intermediate Representation

2 Introduction

3 Requirements

The compiler will compile a custom programming language, that is inspired by Rust.
Requirements for the compiler, and accompanying language is listed below, using MoSCoW prioritisation:

3.1 Language requirements

- The language **must** be Turing complete
  - The language **must** contain loops
    - The loops **could** be recursive functions
  - The language **must** contain variables
    - The variables **must** be immutable, unless explicitly made mutable
  - The language **must** allow conditional code execution
  - The language **must** be able to do basic arithmetic[1]
- The language **must** contain functions
- The language **must** be strongly typed[2] and statically typed[3]
- The language **must** allow an output
- The language **must** support the following type primitives:
  - 32-bit Integer
  - 64-bit Floating point
  - 8-bit Character
  - Boolean
- The language syntax **must** follow the language specification in appendix A
- The language **should** have memory-management with Rust inspired borrow checker
- The language **should** have native array support
- The language **should** allow a runtime input
- The language **should** have error handling
- The language **could** allow access to system resources
- The language **wont** have classes
- The language **wont** have tooling/ecosystem/debugger

[1] Addition, subtraction, multiplication, division and modulus
[2] Strongly typed: Variable types does not change, except by explicit casting
[3] Statically typed: All variables must have an explicit type at initialisation

3.2 Compiler requirements

- Compiler **must** be able to compile cross-platform
- Compiler **must** crash on invalid inputs
- Compiler **must** pass all test cases
- Compiler **must** generate an intermediary language from an AST
- Test coverage **should** be above 80%
- Compiler **should** have clear error messages
- Compiler **wont** have intuitive CLI
- Compiler **wont** support several national languages

Appendix

A Language Syntax

NOTE: things as expressions?

A.1 Variables

let mut Identifier: type = value

Identifier ::= StartChar SubsequentChars*
StartChar ::= [a-zA-Z_]
SubsequentChars::= [a-zA-Z0-9_]

type: [int, float, char, bool]

A.2 Expressions

A.3 Conditionals

IfExpression ::= "if" expression Block ?ElseExpression?
ElseExpression ::= "else" (Block — IfExpression)
let a:int = if(bool) return 5
fn fun():void
let a:int = 1
if(!test()) return
do something else
let a = fun
else if block
else block

A.4 Functions

FunctionExpr ::= "fn" FunctionName ParamList ":" type Block
FunctionName ::= Identifier ParamList ::= "(" Param ("," Param)* ")"
Param ::= Identifier ":" Type
Block ::= "" Statement* ""
Statement ::= /* definition of what constitutes a statement in your language /
Type ::= / definition of what constitutes a type in your language /
Identifier ::= StartChar SubsequentChars*
StartChar ::= [a-zA-Z_]
SubsequentChars::= [a-zA-Z0-9_]

fn func(param:bool):int code

fn test(param_ fun:(name:int, var:int)- int):(int,int)- int
param_ fun(1,1)
return param_ fun

let a:(int,int)- int = test(func)

A.5 Types

int: 32-bit integer
float: 64-bit floating point
char: 8-bit integer
bool: 8-bit
void