Compiler Part 1 – Compiler Internals

Interpreter, a program itself, takes input program (e.g. Python) and takes a program input. Interpreter then spits out a program output.

Compiler gets input program and outputs intermediate language (e.g. Machine Code). This then goes to interpreter which needs a program input to take out a program output

Transpiler/ transpiled – High level to high level language

Compiler has multiple phases-

1. Tokenizer/Lexer – tokens are like individual words (if,while,for)
2. Parser – Group tokens together and represent to a tree data structure, output Abstract Syntax Tree(AST), EX 1
3. TypeChecker – Goes through the given code and checks if it is typed well or there is some errors (Annotated AST)
4. Code Generator – Can be broken up depending on your language you are using, the harder part of the compiler

Optimization?

Context Free Grammar/Grammar – BNF (Backus-Naur Form), expressions that are made to represent given tasks or units

Language Design Creation – Integers, Booleans, Declare and Initialize Variables, Perform arithmetic/ logical operations, EX 2

AND doesn't have a symbol because it is common we just write one after the other unlike OR

Meta Language – What we are writing the compiler in (Java)

Object Language – Our Language

Target Language – What we are compiling to (JavaScript)

You can make a target language as same as the object language

Emacs*

56:00 Minute for coding

Compiler Part 2 – Tokenization/Lexing

Defining tokens shortly after grammar (ifToken)

HashTables/Hash Code

EX 3

Compiler Part 3 - Parsing

Interface Type, Stmt, Exp, Op, Program, AST

Recursive Decent Parsing

S-expressions (LISP language) - (while(< 7 4)

Problems in Parsing 1)Left Recursion exp ::= num | exp+exp 2) Precedence 1+3 *2

Compiler Part 4 – Type Checking

Types – describe data and operations for the data

Ill-typed = program with type errors, well-typed = without

Examples

(EX 1) if (1 < 2){

```
return 7
}else{
Return 3;
}
```

(EX 2)

Var is a variable

Num is Number

type ::= ‘int’ | ‘bool’

vardecc ::= ‘(‘ ‘vardec’ type var expression ‘)’

loop ::= ‘(‘ ‘while’ expression statement ‘)’

assign ::= ‘(‘ ‘=’ var expression ‘)’

expression ::= num | true| false | ‘(‘ op expression expression ‘)’

op ::= ‘+’ | ‘-‘| ‘&&’|’||’|’<’

//

program ::= vardec*

(vardec int x 7)

(vardec bool y true)

(vardec int a (+ 1 2))

(vardec bool b (&& false true))

(vardec int x 0)

(while (< x 10)

(= x (+ x 1)))

(EX 3) Possible Tokens with given code from EX 2:

IdentifierToken(String), NumberToken(int), IntToken , BoolToken, LeftParenToken, VardecToken, RightParenToken, TrueToken, FalseToken, WhileToken, SinglesEqualToken, PlusToken, MinusToken, LogicAndToken, LogicOrToken, LessThanToken