Phases of a Compiler

Wednesday, February 3, 2021 9:23 AM

Translation of a programming language is divided into **phases.**

Compilation = Analysis + Synthesis

1. Lexical Analysis

Reads input and groups characters into meaningful words to prepare it for the next phase
Done by a **lexical analyzer**
Lexemes: (words)

Output: token for every lexeme, stream of tokens

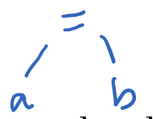
2. Syntax analysis: check if token stream obeys grammatical structure of the programming language

Using context-free grammar, we define how every sentence in your source program or programming language looks like

```
a=b
If (stmt) then else (stmt)
```

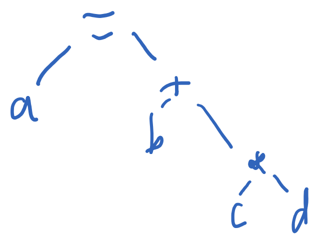
Output of syntax analyzer: tree-like structure or a **syntax tree**

a=b



Internal node - operation (-)
Children - operands (A,B)

a=b+c*d



Captures the operations in the statement to create a syntax tree from the given context free grammar

3. Semantic analysis

Input: syntax tree
"A said B left **his** assignment at home"
His is **ambiguous**

- Eliminate ambiguity

```
{
    //block a
    A = 0;
    ..
    ..
    {
        //block b;
        A = 10;
        ..
        ..
    }
}
```

- Variable binding: associate A with 0 in block A and A with 10 in block B
- Type checking
- Coercions (of types)

Output: unambiguous, semantically correct syntax tree

4. Intermediate code generation

- Not machine code/assembly language code
- Input: unambiguous, semantically correct syntax tree
- Loses similarity to the english language
- Using some intermediate code specifications/languages generate 3AC
- **3 address code** (3AC, TAC)
- Easy to generate and translate into final code
- 3AC is the output of the analysis of the compilation process

Synthesis

Code Generation Phase:
- Input: 3AC
- Output: target code in assembly language

Code Optimization Phase:
- Can be performed on 3AC and target code
- Optimization in terms of saving space in memory or time
- X = 45 + 2 ==> X = 47
- X = 45 * 1 ==> X = 45
- X = 6 * 2 ==> X = 6 + 6 (multiplication is costlier)
- Should not change logic
- Simple optimization can be done by compiler

- Generates a more efficient code

3AC is not specific to a processor
- Machine **independent** code optimization for 3AC

Target code is specific to a machine
- Machine **dependent** code optimization for target code

Symbol table:
- Data structure(hash table) to hold information about the program
-

Variable name	Type	Initial value	Scope
A	Int	0	Class name

- Used to generate 3AC and final code (by code generator, code optimizer, code generator)
-

Procedure name	Type Of arguments	Actual arguments

- Checks if program is correct
- Pass by ref, by value

Token:
<Token_name, attribute.value>
Token_name: symbol/label [can be keyword, identifier, numeric constants, operator]

```
int a = 10;
<keyword, int>
<id, pointertosymboltable>
<op, =>
<constant, 10>
```