TitoCC - C-kääntäjä ttk-91 -arkkitehtuurille.

T.A.

3. tammikuuta 2013

1 Aihe

Tavoitteena on luoda yksinkertainen kääntäjä C-kielestä ttk-91 -arkkitehtuurin symboliselle assembly-kielelle. Lisäksi tehdään graafinen käyttöliittymä, jonka kautta voi avata C:n lähdekooditiedostoja editoitavaksi ja kääntää niitä.

Kääntäjä

- Luokkakirjasto, joka ottaa vastaan tekstimuodossa C-käännösyksikön, ja tuottaa siitä tekstimuotoisen ttk91-assembly-listauksen.
- Tuetut C-kielen ominaisuudet:
  - `int`-tyyppiset globaalit ja paikalliset muuttujat
  - muuttujien näkyvyysalueet
  - sijoitusoperaatiot
  - aritmeettiset ja loogiset lausekkeet
  - `if`/`while` silmukat
  - funktiot ja niiden parametrit
  - `return`, paluuarvot
  - (muita ominaisuuksia jos aikaa on, esim. pointterit, taulukot)
- Ttk-91:n `in` ja `out` -käskyjen tuottaminen esim. `in()`/`out()` intrinsic-funktioiden avulla.
- Virheilmoitukset käännösvirheistä.

Käyttöliittymä

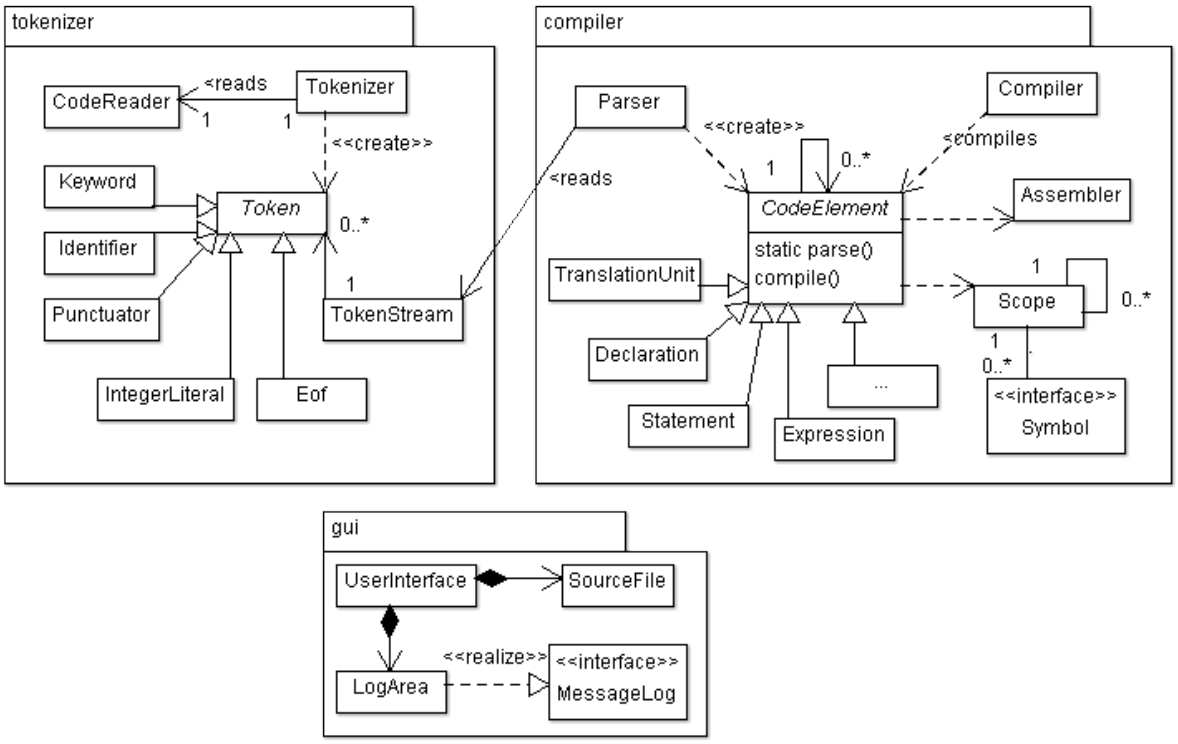
- C-lähdekooditiedoston avaus, editointi ja tallennus.
- Avatun tiedoston kääntäminen.
  - Tuotettu ttk-91-koodi näytetään omassa kentässään.
  - Optionaalisesti voidaan kirjoittaa levylle .k91-tiedosto.
  - Kääntäjän virheilmoitukset näytetään omassa kentässään.

2 Rakenne ja toiminta

Käännös suoritetaan kolmessa osassa. Ensin lähdekoodi analysoidaan tokenisoijan avulla, joka jakaa sen pienimpiin mahdollisiin palasiin, eli välimerkkeihin, tunnisteisiin, vakioihin, ja avainsanoihin.

Tämän jälkeen parseri muodostaa token-listasta puurakenteen, jossa jokaista koodielementtiä vastaan oma `CodeElement`-tyyppinen objekti. Lopuksi kääntäjä käy puurakenteen rekursiivisesti läpi ja generoi niitä vastaavan käskyt. Se myös tarkistaa, että jokaisen elementin tyyppi on oikea, ja että kaikki viitatut muuttujat ovat olemassa. Käännöksen aikana pidetään kirjaa kaikista määritellyistä muuttujista ja funktioista erillisessä `Scope`-hierarkiassa.

Kuva 1: Luokka- ja pakettikaavio.



Tokenisoijan tunnistama syntaksi voidaan määritellä BNF-tyylisen notaation avulla:

```
punctuator := "+" | "++" | "+=" | "-" | "--" | "-=" | "*" | "*=" | "/"
        | "/=" | "%" | "%=" | "|" | "|=" | "||" | "&" | "&=" | "&&" | "~" | "~="
        | "^" | "^=" | "!" | "=" | ">>" | ">>=" | "<<" | "<<=" | "==" | "!="
        | "<" | "<=" | ">" | ">=" | "{" | "}" | "(" | ")" | ";" | ","
integer_literal := digit+ word
identifier := word - keyword
keyword := "while" | "if" | "else" | "return" | "int" | "void"
word := identifier_start_char identifier_char*
identifier_start_char := "_" | ascii_letter
identifier_char := "_" | ascii_letter | digit
```

Ja vastaavasti parserin tunnistamat koodielementit:

```
translation_unit := declaration*
declaration := variable_declaration | function
variable_declaration := type identifier ("=" expression)? ";"
function := type identifier parameter_list block_statement
parameter_list := "(" (parameter ("," parameter)*)? ")"
parameter := type identifier
block_statement := "{" statement* "}"
statement := expression_statement | declaration_statement |
        if_statement | while_statement | block_statement |
        return_statement | ";"
expression_statement := expression ";"
declaration_statement := variable_declaration
if_statement := "if" "(" expression ")" statement ("else" statement)?
while_statement := "while" "(" expression ")" statement
return_statement := "return" (expression)? ";"
type := "void" | "int"
expression := assignment_expression
assignment_expression := binary_expression (assignment_operator
        assignment_expression)?
binary_expression := (binary_expression "||") binary_expression2
binary_expression2 := (binary_expression2 "&&") binary_expression3
...
binary_expression11 := (prefix_expression ("*" | "/" | "%"))
        binary_expression11
prefix_expression := ("++" | "--" | "+" | "-" | "!" | "~")? prefix_expression |
        postfix_expression
postfix_expression := postfix_expression ("++" | "--") | function_call_expression |
        intrinsic_call_expression | primary_expression
function_call_expression := postfix_expression argument_list
intrinsic_call_expression := ("in" | "out") argument_list
argument_list := "(" (expression ("," expression)*)? ")"
primary_expression := identifier | integer_literal | "(" expression ")"
```