Labo 1: lexing

Doel: starten van broncode en alle tokens hieruit halen en waar deze voorkomen. Error messages genereren qua syntaxfouten

Gebruik van **flex**: lexicale analyse generator

make\_tarball.sh => gebruiken om project in te dienen

./make\_tarball.sh 05 "Bert De Saffel & Xandro Vermeulen"

# Docker

*docker run --rm -it -v /c/Users/xandr/lexer\_files:/pract1 tbesard/compilers:pract1*

bestaande map $(pwd):/pract1 mappen op tbesard/compilers:pract1 en komt daar in pract1 terecht

tbesard/compilers:pract1 is de image (stuk na ‘:’ is een tag voor welke versie uit zijn dockerhub je neemt)

# algemeen

uitvoeren: make && ./main test/dummy.c die c klasse meegeven als argument aan de /main

ctrl+d voor uit een docker sessie te gaan

# Lexer.l

Notes:

yytext bevat de string waarde van de laatst ingelezen TOKEN

{newline} moet ook in brackets zitten, niet zonder brackets doen dus aje geen ‘+’ ofzo nodig hebt

Begin.column = end.column het gaat van waar vorige lijn gestopt is

# Basic lexing

**lexer.l**:

Definitie toevoegen:

DIGIT [0-9]

Regel toevoegen:

{DIGIT}+ {return INTEGER}

* INTEGER wordt geinclude door lexer.hpp
* Hetgeen wat in DIGIT staat 1 of meer keer (+)

Voor whitespace [ \t\s] moet er ook een definitie/regel zijn of het wordt niet geabsorbeerd en dan zal hij klagen over een unknown symbol found.

# Location Information

* Wordt uitgevoerd voor elke uitvoering van een lijntje bij de ‘regels’
* In lexer.hpp is er een location struct waarin line (huidige lijn) en column (huidig karakter op die lijn) wordt bijgehouden voor zowel begin als einde van een token.
* Aanroepen in de code gewoon via begin.line / begin.column / end.line / end.column
* De huidige gevonden token vind je via het beschikbaar attribuut yytext

# Error reporting

* YY\_BUF\_SIZE kan je gebruiken als bufsize
* Niet vergeten <memory.h> te includen voor malloc te kunnen doen
* Niet vergeten bij malloc het naar (\*char) te casten, same met de buffer leeg te maken
* Want buffer bijhouden voor de huidige regel code bij te houden
* Printf(“%\*c^”,(int)n,’ ‘) zal n spatiekarakters printen en dan een ^

# Context-sensitive rules

* In *Definitions* moet de definitie van een **toestand** gedefinieerd worden, zodat die kan gebruikt worden in *Rules*. Algemene vorm:  
  %x <NAAM>. Wij willen een toestand hebben voor als we in een block comment zitten:  
    
  %x betekent dat indien de huidige toestand BLOCK\_COMMENT is, enkel regels die bij die toestand horen geldig zijn. Je kan ook %s gebruiken, zodat alle regels nog steeds geldig zijn.
* In *Rules* kan gebruik gemaakt worden van de **INITIAL** toestand, die door flex zelf al gedefinieerd is:  
    
  De **BLOCK\_COMMENT** toestand is dan  
  

Labo 2: parsing

Labo 3: code generation

Labo 4: IR optimalisatie