Test Cases:

a) Compiling test

The test case was used to ensure the compiler was able to detect the basic tokens and grammars.

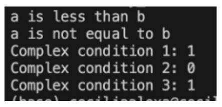
test.py : ensuring the tokens are detected

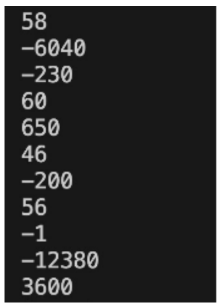
```
from lexx import Lexx

text_input = """

ngeb(5 + 5)ski

"""

lexx = Lexx().get_lexer()

tokens = lexx.lex(text_input)

for token in tokens:

    print(token)
```

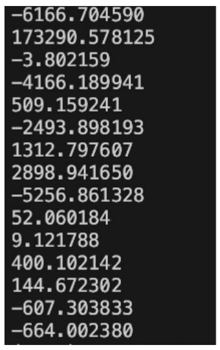
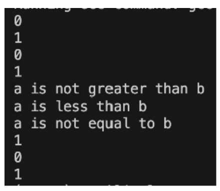
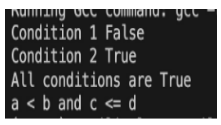
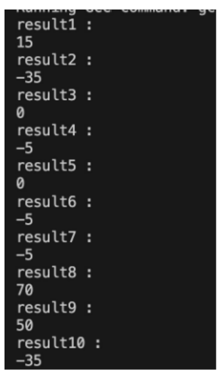
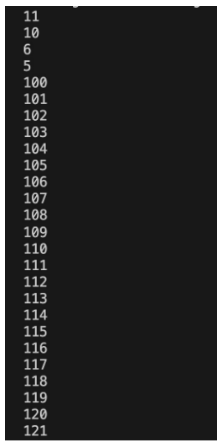
b) Language feature test

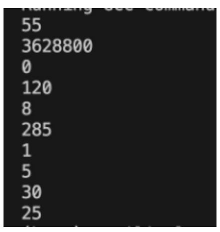
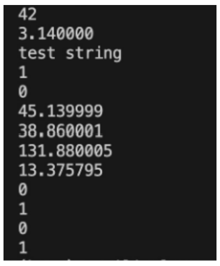
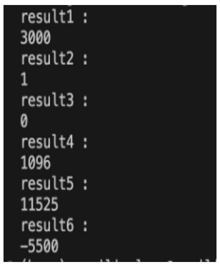
The test cases used to ensure the integrated language features are correct and can give expected output.

Test case type	Description	Test step	Expected result	Status

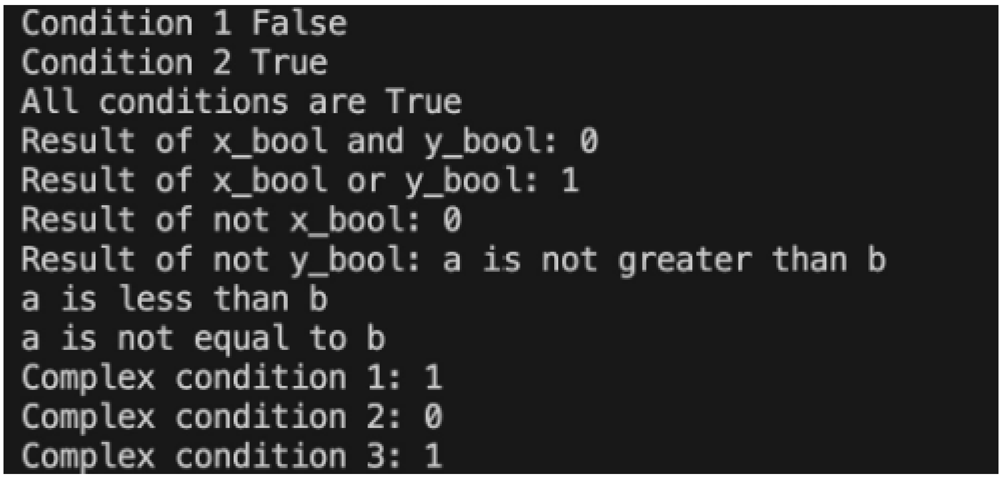
string.ngeb	Printing string, characters and sentences.	test/string	 `a is less than b`
			`a is not equal to b`
			`Complex condition 1: 1`
			`Complex condition 2: 0`
			`Complex condition 3: 1`
				Pass

arithmetic1.ngeb	Variable usage for arithmetic expressions with complex multiplications, divisions, sums and subs in integers.	test/arithmetic1	 `58`
			`-6040`
			`-230`
			`60`
			`650`
			`46`
			`-200`
			`56`
			`-1`
			`-12380`
			`3600`
				Pass

arithmetic2.ngeb	Variable usage for arithmetic expressions with complex multiplications, divisions, sums and subs in floats.	test/arithmetic2	 -6166.704590 173290.578125 -3.802159 -4166.189941 509.159241 -2493.898193 1312.797607 2898.941650 -5256.861328 52.060184 9.121788 400.102142 144.672302 -607.303833 -664.002380	Pass
boolean.ngeb	Implementation of boolean expressions and operators.	test/boolean	 0 1 0 1 a is not greater than b a is less than b a is not equal to b 1 0 1	Pass
conditional.ngeb	Implementation of If, else. then with boolean expressions and variables.	test/conditional	 Running GCC command: gcc - Condition 1 False Condition 2 True All conditions are True a < b and c <= d	Pass
IfThenElse.ngeb	If then else usage with simple variables.	test/IfThenElse	 result1 : 15 result2 : -35 result3 : 0 result4 : -5 result5 : 0 result6 : -5 result7 : -5 result8 : 70 result9 : 50 result10 : -35	Pass
incDec.ngeb	Increment and decrement usage	test/incDec	 11 10 6 5 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121	Pass

integration.ngeb	Complex case consists of assigning statements, variables. arithmetic, data types. conditionals and loops.	test/integration	 55 3628800 0 120 8 285 1 5 30 25	Pass
variable.ngeb	Variable usage in numerical data and strings.	test/variable	 42 3.140000 test string 1 0 45.139999 38.860001 131.880005 13.375795 0 1 0 1	Pass
while.ngeb	While loop implementation	test/while	 result1 : 3000 result2 : 1 result3 : 0 result4 : 1096 result5 : 11525 result6 : -5500	Pass

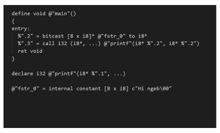
string.ngeb:

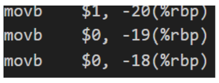


```
Condition 1 False
Condition 2 True
All conditions are True
Result of x_bool and y_bool: 0
Result of x_bool or y_bool: 1
Result of not x_bool: 0
Result of not y_bool: a is not greater than b
a is less than b
a is not equal to b
Complex condition 1: 1
Complex condition 2: 0
Complex condition 3: 1
```

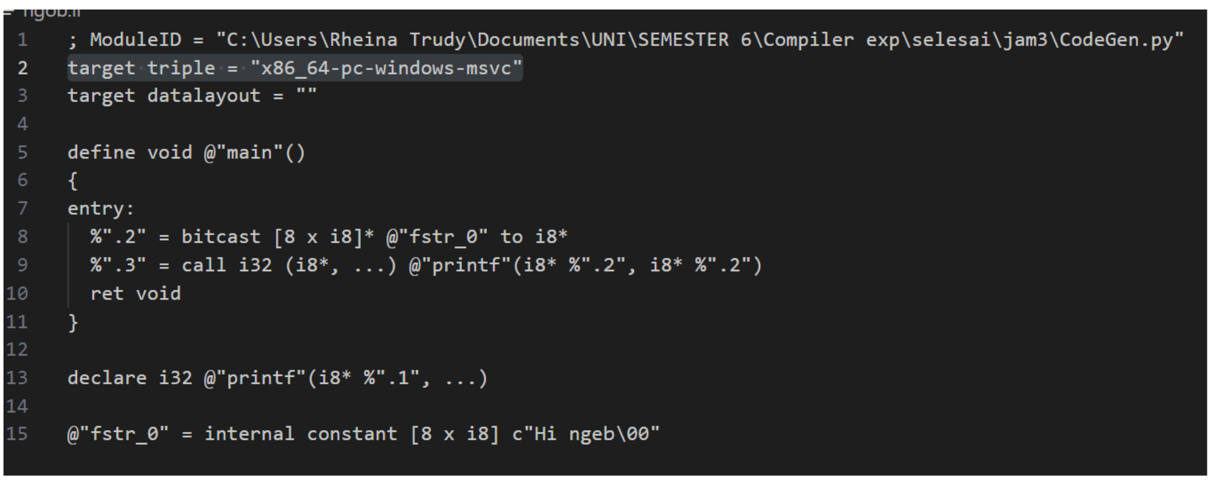
Additional feature:

Feature type	Description	Test step	Expected result	Status

IR code generator	Generate intermediate representational language llvm (ngob.ll)	run main.py	 `define void @"main"()`
			`{`
			`entry:`
			`%".2" = bitcast [8 x i8]* @"fstr_0" to i8*`
			`%".3" = call i32 (i8*, ...) @"printf"(i8* %".2", i8* %".2")`
			`ret void`
			`}`
			
			`declare i32 @"printf"(i8* %".1", ...)`
			
			`@"fstr_0" = internal constant [8 x i8] c"Hi ngeb\00"`
				Pass

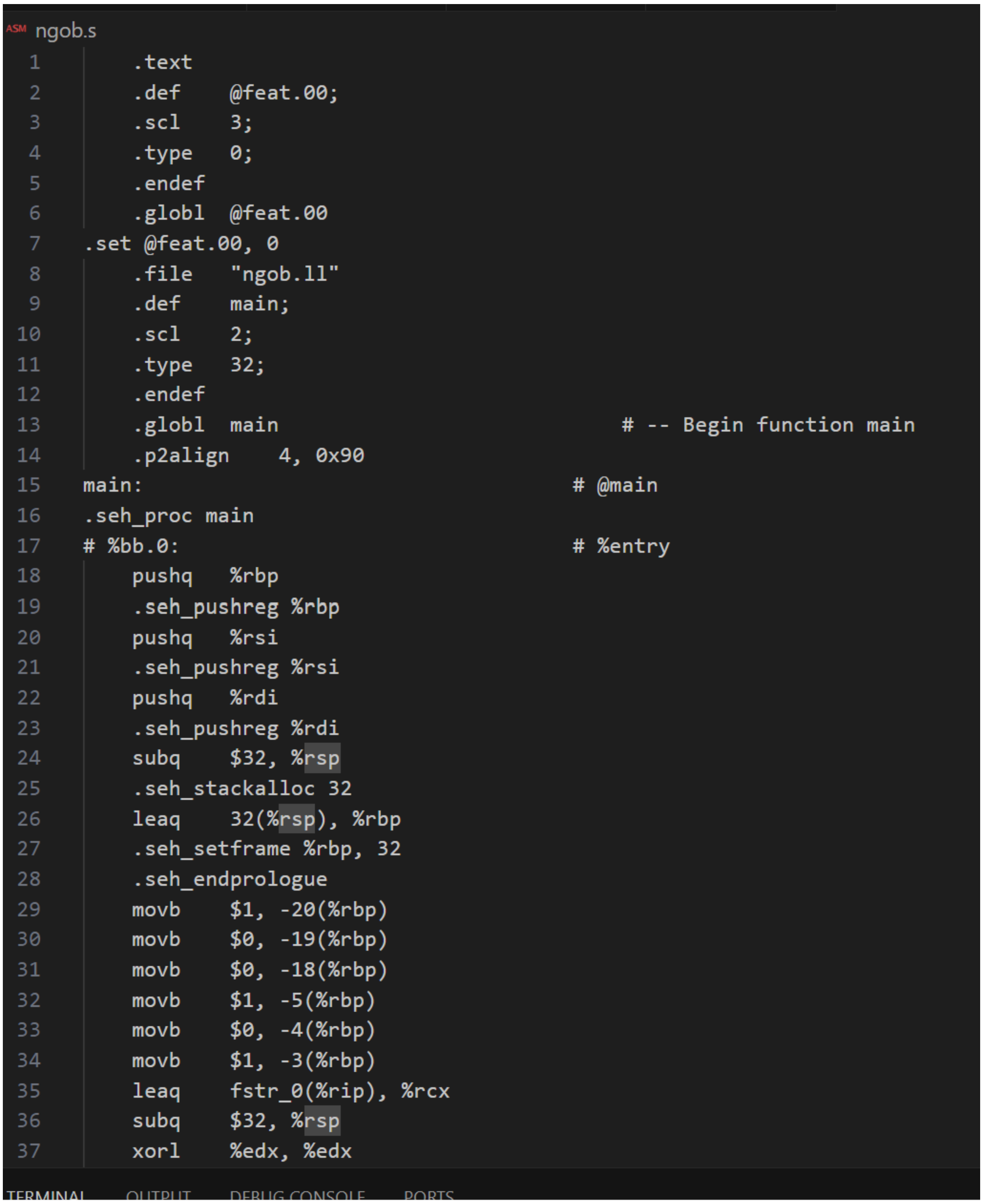
Assembly code generator	Generate assembly code with .s and .asm extension (ngob.s/ngob.asm)	run compile.py, type asm	 `movb $1, -20(%rbp)`
			`movb $0, -19(%rbp)`
			`movb $0, -18(%rbp)`
				Pass

Sample of IR code generated:



```
; ModuleID = "C:\Users\Rheina Trudy\Documents\UNI\SEMESTER 6\Compiler exp\selesai\jam3\CodeGen.py"
target triple = "x86_64-pc-windows-msvc"
target datalayout = ""

define void @"main"()
{
entry:
  %".2" = bitcast [8 x i8]* @"fstr_0" to i8*
  %".3" = call i32 (i8*, ...) @"printf"(i8* %".2", i8* %".2")
  ret void
}

declare i32 @"printf"(i8* %".1", ...)

@"fstr_0" = internal constant [8 x i8] c"Hi ngeb\00"
```

Sample of ASM code generatyed:



```
	.text
	.def	@feat.00;
	.scl	3;
	.type	0;
	.endef
	.globl	@feat.00
.set @feat.00, 0
	.file	"ngob.ll"
	.def	main;
	.scl	2;
	.type	32;
	.endef
	.globl	main                            # -- Begin function main
	.p2align	4, 0x90
main:                                   # @main
.seh_proc main
# %bb.0:                                # %entry
	pushq	%rbp
	.seh_pushreg %rbp
	pushq	%rsi
	.seh_pushreg %rsi
	pushq	%rdi
	.seh_pushreg %rdi
	subq	$32, %rsp
	.seh_stackalloc 32
	leaq	32(%rsp), %rbp
	.seh_setframe %rbp, 32
	.seh_endprologue
	movb	$1, -20(%rbp)
	movb	$0, -19(%rbp)
	movb	$0, -18(%rbp)
	movb	$1, -5(%rbp)
	movb	$0, -4(%rbp)
	movb	$1, -3(%rbp)
	leaq	fstr_0(%rip), %rcx
	subq	$32, %rsp
	xorl	%edx, %edx
```