Computer Science III — Omar Armando Neira Ordoñez (20192020110)

Workshop No. 2

1. For each one of next cases define a regular expression as used in a compiler based on the Python re library:

(i) Identifiers: A regular expression to match valid identifiers (variable names, function names, etc.)

```
r"\b[a-zA-Z]+[a-zA-Z|0-9|_]*\b"
```

(ii) Integer Literal: A regular expression to match integer literals

```
r"-?[0-9]+"
```

(iii) Floating Point Literal: A regular expression to match floating-point literals

```
r"-?[0-9]+(\.[0-9]+)?"
```

(iv) String Literal: A regular expression to match string literals enclosed in doble quotes

```
r'^\"[^\"]*\"$'
```

Note: [^\"] means that " has to be excluded

(v) Single-line Comment: A regular expression to match single-line comments started with '//'.

```
r'^//.*$'
```

Note: the .$ makes that doesn't make a newline (. not matches newline)

(vi) Multi-line Comment: A regular expression to match multiple-line comments enclosed in '/* */'

```
r'^/\*([^*]|\*[^/])*\*/$
```

Note: here we exclude the combination of */

(vii) Whitespace: A regular expression to match whitespace characters (spaces, tabs, newlines)

```
r"^\s+$"
```

Note: \s with s in lowercase is predefined in py re for whitespaces
Note2: \S with s in uppercase is the opposite

(viii) Operators: A regular expression to match common operators (e.g., '+', '-', '*', '/', '==', '!=')

```
r'(\+|\-|\*|\/|==|!=)
```

(ix) Keywords: A regular expression to match reserved Keywords (e.g., 'if', 'else', 'while', 'return').

r'(\b(if | else | while | return)\b' Note: \b ... \b ensures to match the whole word

(x) Hexadecimal Literal: A regular expression to match hexadecimal literals

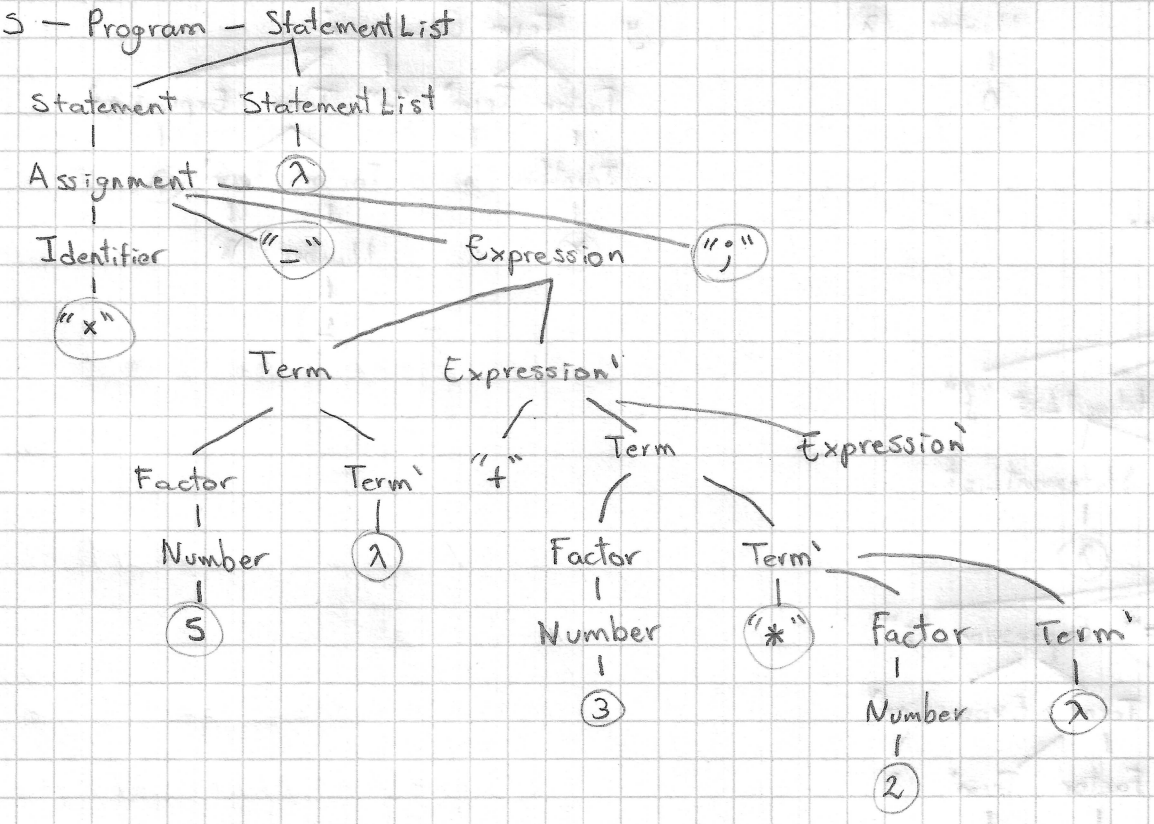
r'-?[0-9a-fA-F]+'

2. Be G a context-free grammar with the following productions:

S	→	Program
Program	→	StatementList
StatementList	→	Statement StatementList | <lambda>
Statement	→	Assignment | IfStatement | WhileStatement | ReturnStatement
Assignment	→	Identifier "=" Expression ";"
IfStatement	→	"if" "(" Expression ")" "{" StatementList "}" ElsePart
ElsePart	→	"else" "{" StatementList "}" | <lambda>
WhileStatement	→	"while" "(" Expression ")" "{" StatementList "}"
ReturnStatement	→	"return" Expression ";"
Expression	→	Term Expression'
Expression'	→	"+" Term Expression' | "-" Term Expression' | <lambda>
Term	→	Factor Term'
Term'	→	"*" Factor Term' | "/" Factor Term' | <lambda>
Factor	→	"(" Expression ")" | Identifier | Number
Identifier	→	[a-z A-Z _] [a-z A-Z 0-9 _]*
Number	→	[0-9]+

(a) Excersice 1

x = 5 + 3 * 2

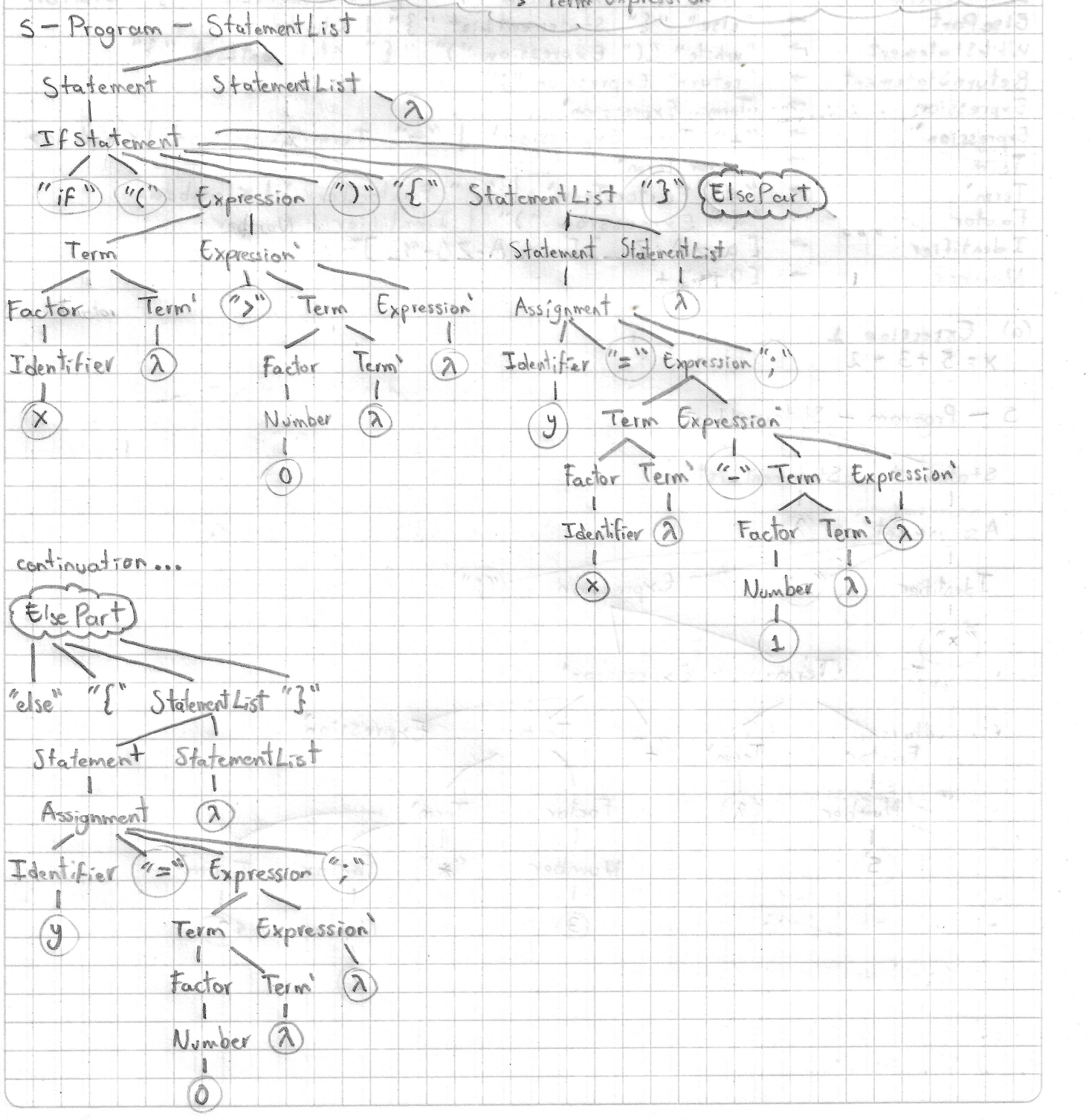


(b) Excercise 2:

```
if (x>0) {
    y = x - 1;
} else {
    y = 0;
}
```

Note: For this excercise the given grammar doesn't work because it doesn't have anything for the comperator "<" and ">", so we will use the next expression definition:

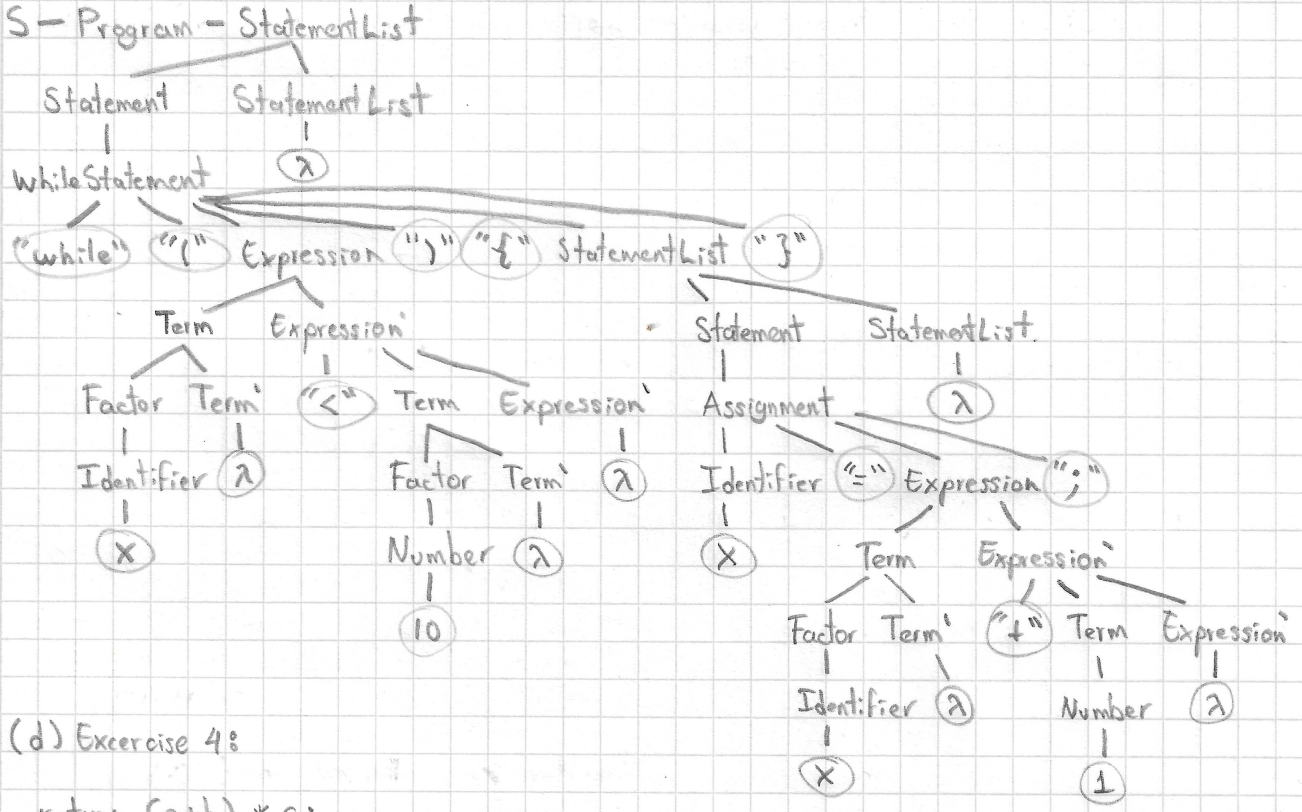
Expression' → "+" Term Expression' | "-" Term Expression' | "<" Term Expression' | ">" Term Expression'



(c) Excercise 3:

```
while (x<10){
    x = x+1;
}
```

Note: For this excercise the give grammar doesn't work either, so we will use same Expression definition as before.



(d) Excercise 4:

```
return (a+b) * c;
```