Problem 1:

<Comand> := <expr> \n

<expr> := <term> | <term> + <expr>

<term> := <factor> | <factor> \* <term>

<factor> := ( <expr> ) | <number>

<number> := <digit> <number> | <digit>

<digit> := 0|1|2|3|4|5|6|7|8|9

Right recursive.

Problem 2:

<Comand> := <expr> \n

<expr> := <minus> | <minus> + <expr>

<minus> := <term> | <term> - <minus>

<term> := <average> | <average> \* <term>

<average> := <factor> | <factor> @ <average>

<factor> := ( <expr> ) | <number>

<number> := <digit> <number> | <digit>

<digit> := 0|1|2|3|4|5|6|7|8|9

Problem 3:

1. 1+2-2

Command

Expr

Minus+expr

Term+expr

Avg+expr

Factor+expr

Number+expr

Digit+expr

1+expr

1+minus

1+term-minus

1+average-minus

1+factor-minus

1+number-minus

1+digit-minus

1+2-minus

1+2-term

1+2-average

1+2-factor

1+2-number

1+2-digit

1+2-2

1. 2-3x5

Command

Expr

Minus

Term – minus

Avg – minus

Factor – minus

Number – minus

Digit – minus

2 – minus

2 – term

2 – avg \* term

2 – factor \* term

2 – number \* term

2 – digit \* term

2 – 3 \* term

2 – 3 \* avg

2 – 3 \* factor

2 – 3 \* number

2 – 3 \* digit

2 – 3 \* 5

c)10-5@2

cmd

expr

minus

term – minus

avg – minus

factor – minus

number – minus

10 – minus

10 – term

10 – avg

10 – factor @ avg

10 – number @ avg

10 – 5 @ avg

10 – 5 @ factor

10 – 5 @ number

10 – 5 @ 2

d)10 – 7 – 2

cmd

expr

minus

term – minus

avg – minus

factor – minus

number – minus

10 – minus

10 – term – minus

10 – avg – minus

10 – factor – minus

10 – number – minus

10 – 7 – minus

10 – 7 – term

10 – 7 – avg

10 – 7 – factor

10 – 7 – number

10 – 7 - 2

Problem 4:

Check .c file attached

Problem 5:

<Comand> := <expr> \n

<expr> := <term> {+<term>}

<term> := <factor> {\*<factor>}

<factor> := ( <expr> )| <number>

<number> := <digit> {<digit>}

<digit> := 0|1|2|3|4|5|6|7|8|9

**Left recursive.**

**The associativity of plus and mult would be left associative since its left recursive**

Problem 6:

<Command> := <expr> \n

<expr> := <minus> {+<minus>}

<minus> := <term> {-<term>}

<term> := <average> {\*<average>}

<average> := <factor> {@<factor>}

<factor> := ( <expr> )| <number>

<number> := <digit> {<digit>}

<digit> := 0|1|2|3|4|5|6|7|8|9

Problem 7:

1. **1 + 2 – 2**

Command

Expr

Minus + minus

Term + minus

Average + minus

Factor + minus

Number + minus

1 + minus

1 + term – term

1 + avg – term

1 + factor – term

1 + number – term

1 + 2 – term

1 + 2 – avg

1 + 2 – factor

1 + 2 – number

1 + 2 – 2

**b)2 – 3 x 5**

command

expr

minus

term – term

avg – term

factor – term

number – term

2 – term

2 – avg \* avg

2 – factor \* avg

2 – number \* avg

2 – 3 \* avg

2 – 3 \* factor

2 – 3 \* number

1. – 3 \* 5
2. **10 – 5 @ 2**

Cmd

Expr

Minus

Minus – minus

Term – minus

Avg – minus

Factor – minus

Number – minus

10 – minus

10 – term

10 – avg

10 – factor @ factor

10 – number @ factor

10 – 5 @ factor

10 – 5 @ number

10 – 5 @ 2

1. **10 – 7 -2**

Cmd

Expr

Minus

Term – term -term

Avg – term – term

Factor – term – term

Number – term – term

10 – term – term

10 – avg – term

10 – factor – term

10 – number – term

10 – 7 – term

10 – 7 – avg

10 – 7 – factor

10 – 7 – number

10 – 7 - 2

Problem 8:

Check .c file attached

Problem 9: (used google for help such as the site <https://www.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html>

1) ^[a-zA-Z\_][\w]\*

2)/”([^\\”]|\\\\/\\”)\*/

3) (\*a)(a\*ba\*ba\*)\*

4)((ab)\*c(ab)\*)((ab)\*c(ab)\*c)\*

Problem 10:

1. \_Bool is an unsigned integer type and bool as true and false values which turns into \_Bool.
2. They both exist because bool is an alias to\_Bool. \_Bool was added onto c. bool is a macro for \_Bool. You are better off including the proper library and the word bool to prevent from breaking code. Also, it depends where you are coding since if you are dealing with old code you might have to take a different approach

Problem 11:

Pragmas are like really advanced comments. It provides the compiler with directives and hints. The reason why they are processed by the scanner is that they can be anywhere. This means that the grammar would be overly complicated so we let the scanner handle this for simplicity.Some benefits of pragmas is that it can tell the compiler of common and repetitive information. Some drawbacks are that program semantics might change overcomplicating compilation.

Problem 12:

1. 2
2. 4