CSc 600-01 (Section 1)

Homework 1 - Syntax

prepared by Ilya Kopyl

CSc 600 Homework 1 - Syntax

February 15, 2018

Homework is prepared by: Ilya Kopyl.
It is formatted in LaTeX, using TeXShop editor (under GNU GPL license).
Syntax diagrams are created in LucidChart online editor (lucidchart.com).

1. Using BNF write the syntax definitions of the following objects:

a) Natural number (1, 2, 3, ...). The answer:

⟨*natural number*⟩ ::= ⟨*non-zero digit*⟩ | ⟨*natural number*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | ⟨*non-zero digit*⟩

⟨*non-zero digit*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

b) Unsigned integer (0, 1, 2, 3, ...). The answer:

⟨*unsigned integer*⟩ ::= ⟨*digit*⟩ | ⟨*unsigned integer*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Example of BNF definition of unsigned integer in languages that do not support leading zeroes (e.g. Python):

⟨*unsigned integer*⟩ ::= 0 | ⟨*natural number*⟩

⟨*natural number*⟩ ::= ⟨*non-zero digit*⟩ | ⟨*natural number*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | ⟨*non-zero digit*⟩

⟨*non-zero digit*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

c) Integer (..., -2, -1, 0, 1, 2, ...). The answer:

⟨*integer*⟩ ::= ⟨*sign*⟩ ⟨*unsigned integer*⟩

⟨*sign*⟩ ::= + | - | ⟨*empty*⟩

⟨*empty*⟩ ::=

⟨*unsigned integer*⟩ ::= ⟨*digit*⟩ | ⟨*unsigned integer*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Example of BNF definition of an integer in languages that do not support leading zeroes (e.g. Python):

⟨*integer*⟩ ::= ⟨*sign*⟩ ⟨*unsigned integer*⟩

⟨*sign*⟩ ::= + | - | ⟨*empty*⟩

⟨*empty*⟩ ::=

⟨*unsigned integer*⟩ ::= 0 | ⟨*natural number*⟩

⟨*natural number*⟩ ::= ⟨*non-zero digit*⟩ | ⟨*natural number*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | ⟨*non-zero digit*⟩

⟨*non-zero digit*⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

d) Odd number (..., -3, -1, 1, 3, ..., 101, ..., 2047, ...). The answer:

⟨*odd number*⟩ ::= ⟨*sign*⟩ ⟨*unsigned odd number*⟩

⟨*sign*⟩ ::= + | - | ⟨*empty*⟩

⟨*empty*⟩ ::=

⟨*unsigned odd number*⟩ ::= ⟨*odd digit*⟩ | ⟨*unsigned integer*⟩ ⟨*odd digit*⟩

⟨*unsigned integer*⟩ ::= ⟨*digit*⟩ | ⟨*unsigned integer*⟩ ⟨*digit*⟩

⟨*digit*⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Example of BNF definition of an odd number in languages that do not support leading zeroes (e.g. Python):

```
⟨odd number⟩             ::= ⟨sign⟩ ⟨unsigned odd number⟩
⟨sign⟩                   ::= + | - | ⟨empty⟩
⟨empty⟩                  ::=
⟨unsigned odd number⟩    ::= ⟨odd digit⟩ | ⟨natural number⟩ ⟨odd digit⟩
⟨natural number⟩         ::= ⟨non-zero digit⟩ | ⟨natural number⟩ ⟨digit⟩
⟨digit⟩                  ::= 0 | ⟨non-zero digit⟩
⟨non-zero digit⟩         ::= 2 | 4 | 6 | 8 | ⟨odd digit⟩
⟨odd digit⟩              ::= 1 | 3 | 5 | 7 | 9
```

e) Even number (..., -4, -2, 0, 2, 4, ..., 332, ..., 1022, ...). The answer:

```
⟨even number⟩            ::= ⟨sign⟩ ⟨unsigned even number⟩
⟨sign⟩                   ::= + | - | ⟨empty⟩
⟨empty⟩                  ::=
⟨unsigned even number⟩   ::= ⟨even digit⟩ | ⟨unsigned integer⟩ ⟨even digit⟩
⟨unsigned integer⟩       ::= ⟨digit⟩ | ⟨unsigned integer⟩ ⟨digit⟩
⟨digit⟩                  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Example of BNF definition of an even number in languages that do not support leading zeroes (e.g. Python):

```
⟨even number⟩            ::= ⟨sign⟩ ⟨unsigned even number⟩
⟨sign⟩                   ::= + | - | ⟨empty⟩
⟨empty⟩                  ::=
⟨unsigned even number⟩   ::= ⟨even digit⟩ | ⟨natural number⟩ ⟨even digit⟩
⟨natural number⟩         ::= ⟨non-zero digit⟩ | ⟨natural number⟩ ⟨digit⟩
⟨digit⟩                  ::= 0 | ⟨non-zero digit⟩
⟨non-zero digit⟩         ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨even digit⟩             ::= 0 | 2 | 4 | 6 | 8
```

f) Integer divisible by five (..., -10, 5, 0, 5, 10, ...). The answer:

⟨*integer div-by-5*⟩ `::=` ⟨*sign*⟩ ⟨*unsigned int div-by-5*⟩

⟨*sign*⟩ `::=` `+` | `-` | ⟨*empty*⟩

⟨*empty*⟩ `::=`

⟨*unsigned int div-by-5*⟩ `::=` ⟨*div-by-5 suffix*⟩ | ⟨*unsigned integer*⟩ ⟨*div-by-5 suffix*⟩

⟨*unsigned integer*⟩ `::=` ⟨*digit*⟩ | ⟨*unsigned integer*⟩ ⟨*digit*⟩

⟨*div-by-5 suffix*⟩ `::=` `0` | `5`

⟨*digit*⟩ `::=` `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

Example of BNF definition of an integer divisible by 5 in languages that do not support leading zeroes (e.g. Python):

⟨*integer div-by-5*⟩ `::=` ⟨*sign*⟩ ⟨*unsigned int div-by-5*⟩

⟨*sign*⟩ `::=` `+` | `-` | ⟨*empty*⟩

⟨*empty*⟩ `::=`

⟨*unsigned int div-by-5*⟩ `::=` ⟨*div-by-5 suffix*⟩ | ⟨*natural number*⟩ ⟨*div-by-5 suffix*⟩

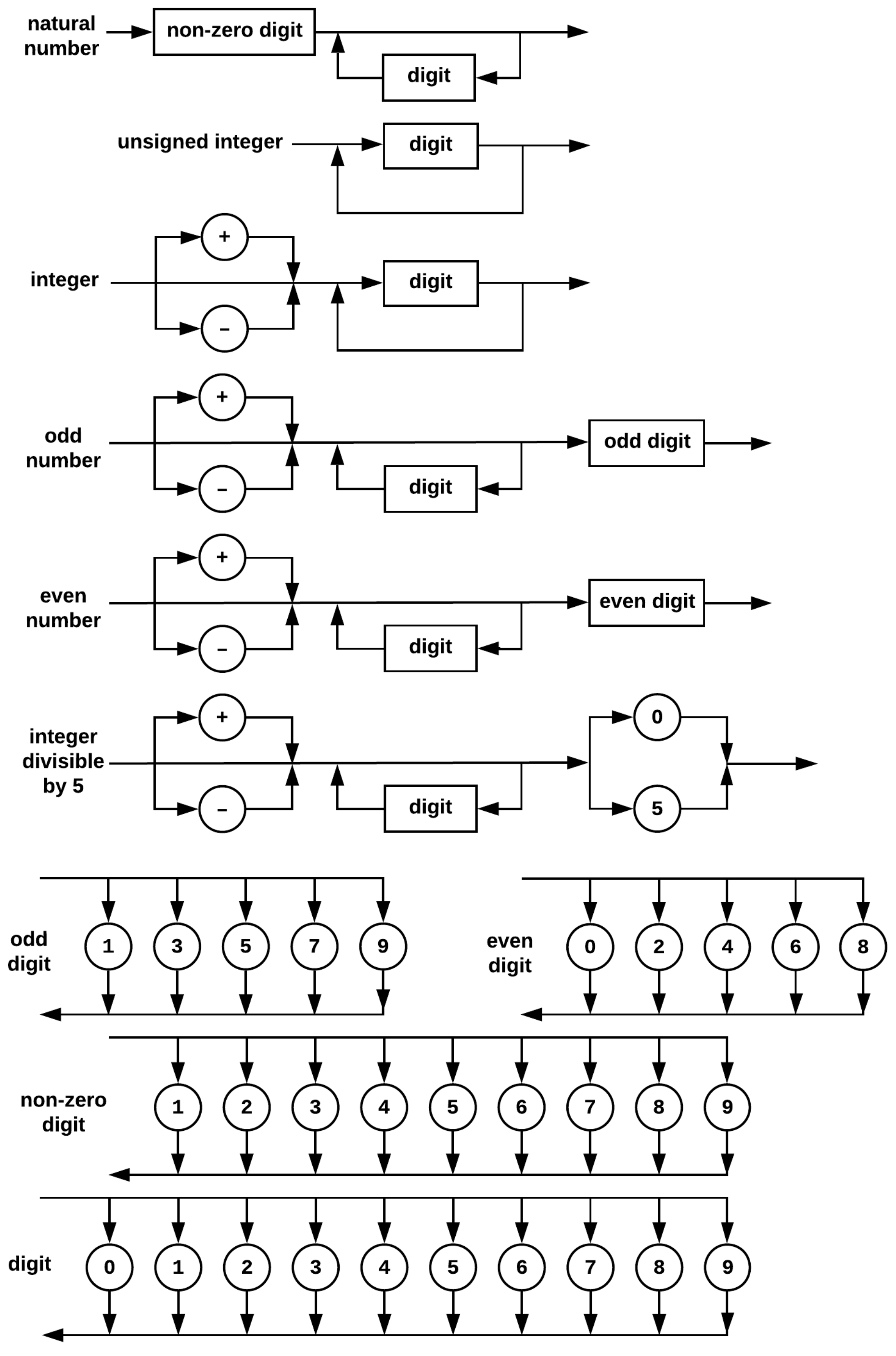
⟨*natural number*⟩ `::=` ⟨*non-zero digit*⟩ | ⟨*natural number*⟩ ⟨*digit*⟩

⟨*div-by-5 suffix*⟩ `::=` `0` | `5`

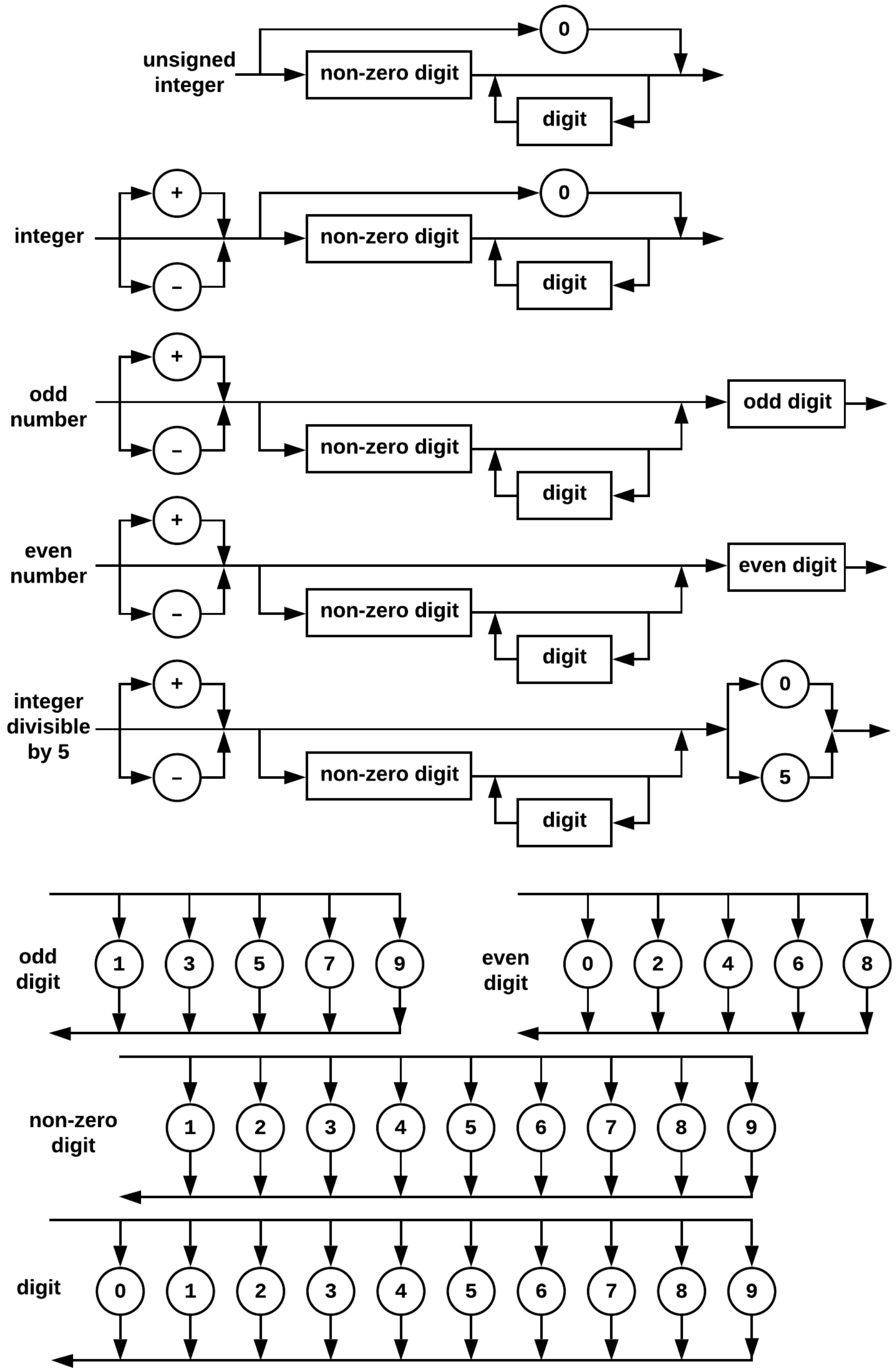
⟨*digit*⟩ `::=` `0` | ⟨*non-zero digit*⟩

⟨*non-zero digit*⟩ `::=` `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

2. Show syntax diagrams for questions (a), ..., (f) of problem 1.



Example of syntax diagrams for integers with no support of leading zeroes.



3. Write a BNF definition of the syntax of (all possible) input statements in C++.

Following is an example of input statement in C++:

```
cin >> sclr >> vec[2 * i - 1] >> mat[f(i)][j + k] >> t[i/3][j][k];
```

The answer:

```
⟨input statement⟩          ::= cin ⟨input arguments⟩ ;

⟨input arguments⟩          ::= >> ⟨input value⟩ | ⟨input arguments⟩ >> ⟨input value⟩

⟨input value⟩              ::= ⟨struct member⟩ | ⟨array element⟩ | ⟨identifier⟩

⟨struct member⟩            ::= ⟨identifier⟩ . ⟨identifier⟩
                             | ⟨struct member⟩ . ⟨identifier⟩

⟨array element⟩            ::= ⟨identifier⟩ ⟨array indicies⟩

⟨identifier⟩               ::= ⟨non-digit character⟩ | ⟨identifier⟩ ⟨digit⟩

⟨array indicies⟩           ::= ⟨array index⟩ | ⟨array indicies⟩ ⟨array index⟩

⟨array index⟩              ::= [ ⟨numerical expression⟩ ]

⟨numerical expression⟩     ::= ⟨arithmetic expression⟩
                             | ⟨function call⟩

⟨arithmetic expression⟩    ::= ⟨compound operand⟩
                             | ⟨arithmetic expression⟩ ⟨operator⟩ ⟨arithmetic expression⟩

⟨operator⟩                 ::= + | - | * | / | %

⟨compound operand⟩         ::= ⟨operand⟩ | ⟨unary expression⟩

⟨unary expression⟩         ::= ++ ⟨operand⟩ | ⟨operand⟩ ++
                             | -- ⟨operand⟩ | ⟨operand⟩ --

⟨operand⟩                  ::= ⟨identifier⟩ | ⟨integer number⟩ | ⟨floating point number⟩

⟨integer number⟩           ::= ⟨digits⟩ | ⟨digits⟩ L | ⟨digits⟩ LL

⟨floating point number⟩    ::= ⟨real number⟩ | ⟨real number⟩ F

⟨real number⟩              ::= ⟨digits⟩ . ⟨digits⟩ | ⟨digits⟩ . | . ⟨digits⟩

⟨function call⟩            ::= ⟨identifier⟩ ( ⟨function arguments⟩ )

⟨function arguments⟩       ::= ⟨argument⟩ | ⟨function arguments⟩ , ⟨argument⟩

⟨argument⟩                 ::= ⟨function call⟩ | ⟨expression⟩
```

```
⟨expression⟩            ::= ⟨numerical expression⟩
                          | ⟨string⟩
                          | ⟨character literal⟩
                          | ⟨void⟩

⟨string⟩                ::= " ⟨characters⟩ "

⟨character literal⟩     ::= ' ⟨char⟩ ' | ' \ ⟨char⟩ '

⟨characters⟩            ::= ⟨char⟩ | ⟨characters⟩ ⟨char⟩

⟨char⟩                  ::= ⟨non-digit character⟩ | ⟨digit⟩ | ⟨whitespace⟩

⟨whitespace⟩            ::= ' '

⟨non-digit character⟩   ::= A | B | C | D | E | F | G | H | I | J
                          | K | L | M | N | O | P | Q | R | S | T
                          | U | V | W | X | Y | Z | a | b | c | d
                          | e | f | g | h | i | j | k | l | m | n
                          | o | p | q | r | s | t | u | v | w | x
                          | y | z | _

⟨digits⟩                ::= ⟨digit⟩ | ⟨digits⟩ ⟨digit⟩

⟨digit⟩                 ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

⟨void⟩                  ::= ⟨empty⟩

⟨empty⟩                 ::=
```

4. Write a BNF definition of the syntax of (all possible) output statements in C++.

Following is an example of output statement in C++:

```
cout << 12.34 * a / rate << " " << 43.21 << " "
<< alpha + x[2*i-1] << " " << (p && q) << " "
<< pow(t[i][j],1.2) << " string " << 's'
<< " " << myfun(x, sin(x+y), third_argument) ;
```

The answer (the definitions for auxiliary BNF productions are listed in the previous answer):

```
⟨output statement⟩      ::= cout ⟨output arguments⟩ ;

⟨output arguments⟩      ::= << ⟨output value⟩ | ⟨output arguments⟩ << ⟨output value⟩

⟨output value⟩          ::= ⟨expression⟩
```