Exam1 Solution – Penn State CMPSC 461 Spring 2022, Prof. G. Tan.

Question about grammar derivations:

- Left-most derivation of 5E+2
  <SNFloat> -> <Float>E<Exponent>
  -> <NonZeroDigit>E<Exponent>
  -> 5E<Exponent>
  -> 5E+<Num>
  -> 5E+<Digit>
  -> 5E+<NonZeroDigit>
  -> 5E+2

- Right-most derivation of 3.14E3
  <SNFloat> -> <Float>E<Exponent>
  -> <Float>E<Digit>
  -> <Float>E<NonZeroDigit>
  -> <Float>E3
  -> <NonZeroDigit>.<Num>E3
  -> <NonZeroDigit>.<Digit><Digit>E3
  -> <NonZeroDigit>.<Digit><NonZeroDigit>E3
  -> <NonZeroDigit>.<Digit>4E3
  -> <NonZeroDigit>.<NonZeroDigit>4E3
  -> <NonZeroDigit>.14E3
  -> 3.14E3

- Why 100.4E+1 is not part of the language defined by the grammar?
  There are multiple ways to answer this question.
  (1) **Natural language** According the grammar, for 100.4E+1, 100.4 must be derived from <Float>, but <Float> can have only one <NonZeroDigit> before its "."; thus 100.4E+1 cannot be derived from the grammar.
  (2) **Derivation** The derivation for the string will get stuck at the following step, with a left-most derivation.
  <SNFloat> ->* <NonZeroDigit>.<Num>E<Exponent>
  (3) **Parse tree** A stuck parse tree would also be a valid answer.

Question about scoping:

- For the version with first line being “int a b;”
  (a) main: <b,10>, <a,1>
  foo: <a,2>, <b,1>
  bar: <a,6>, <b,1>
  (b) <a,6>, <b,10>
  (c) <a,6>, <b,10>

- For the version with first line being “int a;”

(d) main: <b,10>, <a,1>
foo: <b,2>, <a,3>
bar: <a,5>

(e) <a,5>, <b,10>

(f) <a,5>, <b,2>

Question about recursive descent parsing:

- For the version with grammar “<term> -> <term> * <id> | <id>; <id> -> a | b | c”

(a) The grammar contains left recursion in the rules of <term>. A recursive-descent parser results in an infinite loop when parsing a string from such a grammar.

(b)

<term> -> <id> {* <id>}
<id> -> a | b | c
or
<term> -> <id> * <term> | <id>
<id> -> a | b | c

(c)

```
void term() {
  id(); // parses an <id> and advance token to nextToken ()
  while ( token.type == timesOp ) {
     token = nextToken();
     id();
   }
}
```

or

```
void term() {
  id(); // parses an <id> and advance token to nextToken ()
  if ( token.type == timesOp ) {
     token = nextToken();
     term();
   }
}
```

- For the version with grammar “<clause> -> <clause> and <phrase> | <phrase>”

(a) The grammar contains left recursion in the rules of <clause>. A recursive-descent parser results in an infinite loop when parsing a string from such a grammar.

(b)

<clause> -> <phrase> {and <phrase>}
<phrase> -> …
or
<clause> -> <phrase> and <clause> | <phrase>
<phrase> -> …

(c)
```
void clause() {
  phrase(); // parses a <phrase> and advance token to nextToken ()
  while ( token.type == andOp ) {
    token = nextToken();
    phrase();
  }
}
```

or

```
void clause() {
  phrase(); // parses a <phrase> and advance token to nextToken ()
  if ( token.type == andOp ) {
    token = nextToken();
    clause();
  }
}
```

Question about no nonleading zeros in <Exponent>

- One possible answer:
  <Exponent> -> [(+|-)] <CNum>
  <CNum> -> <Digit> | <NonZeroDigit><Num>

- Another possible answer:
  <Exponent> -> [(+|-)]<Digit> | [(+|-)]<NonZeroDigit> {<Digit>}