Taylor Roberts
CS354-2

Java PL examples

Q: 1.1

```
(a) int&& = 55;

(b) int;

(c) int num;
    String str = “pikachu”;
    num = str;

(d) int num;
    num = 6;
    num = num/0;

(e) int[] A;
    A = new int[4];
    A[5] = 3;
```

C PL examples

Q: 1.8

(A)

Q: How accurate is this sort of dependence management?

A: That depends on how well you make the make file pertaining to what is being compiled in the make file.
Make files can be really complicated, but there is almost always to tool to compensate. If not properly
made, the make file can be very inaccurate.

(B)

Q: Under what circumstances will it lead to unnecessary work?

A: If the program is viewed, then saved. This changes the timestamp for the program which is what
make uses to check if anything has been changed. If the timestamp is different make will recompile
files that haven't been changed.

(C)

Q: Under what circumstances will it fail to recompile something that needs to be recompiled?

A: If the header file is changed, then the C files that the header file is getting its code from
need to be recompiled. By default, this does not happen unless a DEPS = headerfile.h rule is made.
So, if that rule is not within the make file, and the header file is changed. Running make will
not effectively compile the program suite effectively.

Q: 2.1

(A)

Q: What's the regular expression for a string in C PL

```
A:  ? “”(!{\, “, \n} | \!{\n})* ”
```

(B)

Q: What’s the regular expression for a pascal comment?

```
A: ? (* (!(*) | (*!()))))* *+)
```

(C)

Q: What are the regular expressions for all numeric constants in C PL

A:

```
C_constant → int_const | fp_const
int_const → (oct | dec | hex) int_suffix
oct_int → 0 oct_digit*
dec_int → nonzero_digitdec_digit*
hex_int → (0x | 0X) hex_digithex_digit*
oct_digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
nonzero_digit → 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
dec_digit → 0 | nonzero_digit
hex_digit → dec_digit A | B | C | D | E | F | a | b | c | d | e | f
dec_float → dec_digit*|.dec_digit*|E | e
hex_float → e | ϵ exponent | ϵ
type → long | unsigned long | longlong |unsigned longlong
exponent → + | - | ϵ
unsigned → U | u
float → F | f
long → L | l
longlong → LL | ll
```

(D)

Q: What are the regular expressions for floating-point constants in Ada

A:

```
Ada_int → digit ( ( _ | ε ) digit )*
Extended_digit → digit | a | b | c | d | e | f | A | B | C | D | E | F
Ada_extended_int → extended_digit ( ( _ | ε ) extended_digit )*
AdaFP_num → ( ( Ada_int ( ( . Ada_int | ε ) )
   | ( Ada_int # Ada_extended_int
   ( ( . Ada_extended_int ) | ε ) # ) )
   ( ( ( e | E ) ( + | - | ε ) Ada_int ) | ε )
```

(E)

Q: What are the regular expressions for inexact constants in Scheme?

```
A: digit + # * (.# * | ϵ ) | digit* .digit + # *
```

(F)

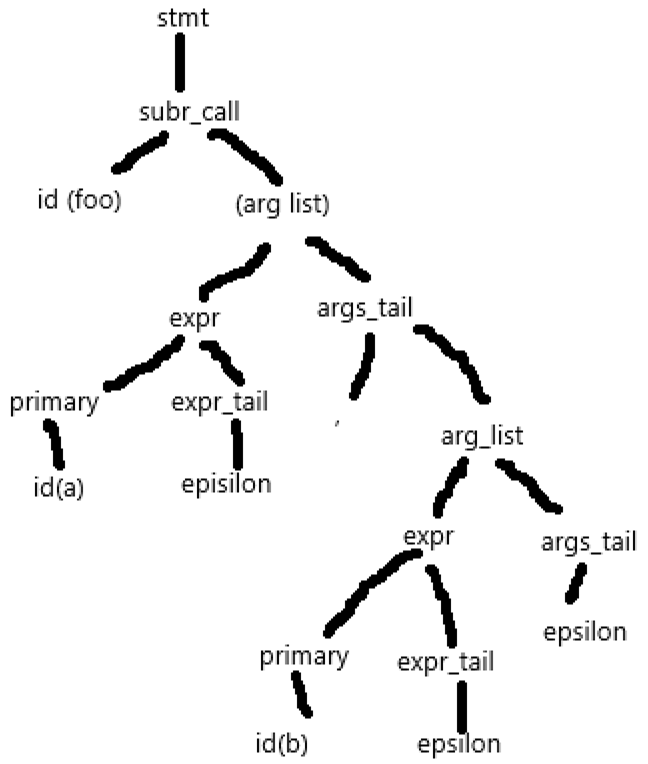
Q: RE for Financial quantities in American notation?

```
A: nzerodigit → 1 | 2 | 3 | 4 | 5 | 4 | 7 | 8 | 9
digit → 0 | nzerodigit
group → , digit digit digit
number → $ * * ( 0 | nzerodigit ( ε | digit | digit digit ) group* ) ( ε
| . digit digit )
```

Q: 2.13

(A)

Q: Construct a parse tree for the input string foo (a,b)
A:



(B)

Q: Give a canonical derivation
A:
stmt → subr_call
subr_call → id (arg_list)
arg_list → expr,args_tail
args_tail → ,,arg_list
expr → primary,expr_tail
expr_tail → op,expr
expr →id
expr →primary, expr_tail
primary → id,expr
assignment →id

Q: 2.17

```
program → stmt list $$
stmt list → stmt list stmt
stmt list → stmt
stmt → id := expr
stmt → read id
stmt → write expr
expr → term
expr → expr add op term
term → factor
term → term mult op factor
factor → ( expr )
factor → id
factor → number
add op → +
add op → -
mult op → *
mult op → /
stmt → if condition them stmt_list fi
     → while condition do stmt_list od
condition → expr relation expr
relation → <
         → >
         → <=
         → >=
         → =
         → !=
```