Jason Egbert
CS 354-002
Fall 2018
09/26/2018

Textbook Assignment #1

1.1) Errors in a computer program can be classified according to when they are detected and, if they are detected at compile time, what part of the compiler detects them. Using your favorite imperative language, give an example of each of the following

(a) A lexical error, detected by the scanner

```
int String aWord = “a word”;
```

(b) A syntax error, detected by the parser

```
int notAnInt = 103.99527698e142;
```

(c) A static semantic error, detected by semantic analysis

```
int badSemantics;
badSemantics = “not an int”;
```

(d) A dynamic semantic error, detected by code generated by the compiler

```
int intOverflow = Integer.Max();
intOverflow += Integer.Max();
```

(e) An error that the compiler can neither catch nor easily generate code to catch (this should be a violation of the language definition, not just a program bug)

```
int example[] = new int[14];
for(int i = 0; x <= 14; x++){
    example[i] = 42;
}
```

1.8) The Unix make utility allows the programmer to specify dependences among the separately compiled pieces of a program. If file A depends on file B and file B is modified, make deduces that A must be recompiled, in case any of the changes to B would affect the code produced for A. How accurate is this sort of dependence management? Under what circumstances will it lead to unnecessary work? Under what circumstances will it fail to recompile something that needs to be recompiled?

a) This kind of dependence management is conditionally accurate. Under some circumstances, like when the input and output of the functions in B are not changed by the modifications made in the file B, i.e. when changes are made to make the algorithms more efficient, there was no necessity to recompile A, as from the point of view of the code within A nothing has changed.

b) If there is a third file, file C, which is modified, and requires a recompilation of file B and A, file B will be recompiled because file C upon which it is directly dependent has been modified. However, if A only relies upon C indirectly, and calls directly on B, and the input/output between B and C have changed, A may not be recompiled and may produce incorrect output.

2.1) Write regular expressions to capture the following.

(a) "$(A \cup N \cup S \cup P \cup \backslash" \cup \backslash\backslash)^*$"

$A = \{A, B, C, \dots, Y, Z\} \cup \{a, b, c, \dots, Y, Z\}$

$N = \{0,1,2,3,4,5,6,7,8,9\}$

$S = \{' '\}$

$P = \{all\ x | x = legal\ punctuation, exluding\ backslash\ and\ "\}$

(b) $[F_1(A \cup N \cup S \cup P)^* F_2] \cup [\{(A \cup N \cup S \cup P)^*\}]$

$A = \{A, B, C, \dots, Y, Z\} \cup \{a, b, c, \dots, Y, Z\}$

$N = \{0,1,2,3,4,5,6,7,8,9\}$

$S = \{' '\}$

$P = \{all\ x\ | x = legal\ punctionation, excluding\ '(*', '*)', '\{', and\ '\}'\}$

$F_1 = (*$

$F_2 = *)$

(c) $[[[(RN^*) \cup (0OO^*) \cup (0(x \cup X)(N \cup A)^*)][U \cup u \cup L \cup l \cup LL \cup ll \cup \varepsilon]] \cup \Big[\Big[(RN^*)(.NN^* \cup \varepsilon)\Big(((e \cup E)(+ \cup - \cup \varepsilon)NN^*) \cup \varepsilon\Big)\Big] \cup [0(x \cup X)(N \cup A)(N \cup A)^*(.\cup \varepsilon)(p \cup P)(+ \cup - \cup \varepsilon)NN^*]\Big](F \cup f \cup L \cup l \cup \varepsilon)]$

$R = \{1, 2, 3, 4,5,6,7,8, 9\}$

$N = \{0\} \cup R$

$A = \{A, B, C, D, E, F, a, b, c, d, e, f\}$

$O = \{0,1,2,3,4,5,6,7\}$

(d) $\big[N(\_N \cup N)^*(e \cup E)(+ \cup - \cup \varepsilon)N(\_N \cup N)^* \cup N(\_N \cup N)^*(.)N(\_N \cup N)^*\big((e \cup E)(+ \cup - \cup \varepsilon)N(\_N \cup N)^* \cup \varepsilon\big)\big] \cup [(2 \cup 8 \cup 10 \cup 16)\#N(\_N \cup N)^*(e \cup E)(+ \cup - \cup \varepsilon)N(\_N \cup N)^*\# \cup (2 \cup 8 \cup 10 \cup 16)\#N(\_N \cup N)^*(.)N(\_N \cup N)^*\#\big((e \cup E)(+ \cup - \cup \varepsilon)N(\_N \cup N)^* \cup \varepsilon\big)]$

$N = \{0,1,2,3,4,5,6,7,8,9\}$

$A = \{A, B, C, D, E, F, a, b, c, d, e, f\}$

(e) $[N^*.N^*\#^*] \cup [N^*\#^*.\#^*] \cup [N^*\#^*]$

$N = \{0,1,2,3,4,5,6,7,8,9\}$

(f) $[\$ *^* \Big(0 \cup R \cup RN \cup RNN \cup \big((R \cup RN \cup RNN)(,NNN)^*\big)\Big)(.NN \cup \varepsilon)]$

$R = \{1,2,3,4,5,6,7,8,9\}$

$N = \{0\} \cup R$

2.13) parts a,b pp108

(a) foo(a,b)

```
stmt: foo(a, b) -> subr_call
   |- subr_call: id(a, b) -> id(arg_list)
        |- id(arg_list): a b -> expr args_tail
              |- expr: a -> primary
              |     |- primary: a -> id
              |- args_tail: b ->expr args_tail
                   |- expr: b -> primary
                        |- primary: b -> id
```

(b) Right-most derivation of foo(a, b)

```
stmt: foo(a, b) -> subr_call
  |- subr_call: id(a, b) -> arg_list
     |- arg_list: a, b -> expr args_tail
              |- args_tail: , b -> , arg_list
                    |    |- arg_list: b -> primary
                    |       |- primary: b -> id
                    |- expr: a -> primary
                       |- primary: a -> id
```

2.17) extend the grammar

1. program -> stmt_list $$
2. stmt_list -> stmt_list stmt
3. stmt_list -> if stmt_list
4. stmt_list -> while stmt_list
5. stmt_list -> stmt
6. stmt -> if expr comp_op expr
7. stmt -> while expr comp_op expr
8. stmt -> id := expr
9. stmt -> then id := expr
10. stmt -> read id
11. stmt -> write expr
12. expr -> term
13. expr -> expr add_op term
14. term -> factor
15. term -> term mult_op factor
16. factor -> (expr)
17. factor -> id
18. factor -> number
19. add_op -> +
20. add_op -> -
21. mult_op -> *
22. mult_op -> /
23. comp_op -> >
24. comp_op -> <
25. comp_op -> ==
26. comp_op -> >=
27. comp_op -> <=
28. comp_op -> !=