Assignment 4 Part 2

PART 2

1. Consider a two-dimensional array with 7 columns and 8 rows stored starting at location 2001 in row-major order. Its elements occupy 4 bytes each. The location of the 3rd element of the 4th row is:

   ((((Rows above) x (Columns)) + (Elements to left)) x Bytes) + (TopLeft Location)
   (((3 x 7) + 2) x 4) + 2001
   = 2093

2. In what way is static type checking better than dynamic type checking?

   The big benefit of static type checking is that it allows errors to be caught early in the development cycle, unlike dynamic type checking. Static type checking applies to languages that know the types of variables at compile time. By knowing the types of variables, the compiler can produce optimized machine code. Overall, static type checking benefits in optimizing machine code and detect errors, even in rarely used code paths.

3. What are the advantages and disadvantages of the ability to change objects in Ruby (language evaluation criteria)?

   The advantages of the ability to change objects in Ruby is that it adds flexibility to the language. This allows the program to be able to do more things than languages that don't allow this. The drawback is that Ruby trades readability for flexibility, thus affecting the languages reliability.

4. Write a prolog predicate definition that compares the two lists of numbers and return their number of common elements.

   Returns the number of common elements

```
numofcommon(L1,L2,Z) :-
        intersection(L1,L2,X),
        length(X,Z).
```

   Returns a list of common elements

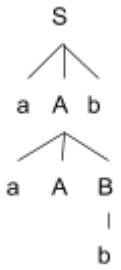
```
common (L1,L2,X):-
```

intersection(L1,L2,X).

5. Using labels and gotos give an operational semantic definition of the C for, while structures.

For (expr1; expr2, expr3)
    Evaluate (expr1);
    Loop = control = evaluate (expr2)
        If control == 0 goto out
    Evaluate (expr 3)
    Goto loop
Out: ...

6. Given the following grammar and the right sentential form, draw a parse tree and show the phrases and simple phrases, as well as the handle.
    a. aaAbb

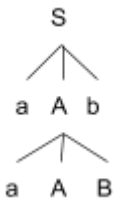


Handle : Left most simple phrase, since there is only one simple phrase, therefore b
Phrases: Starting from bottom up, all are sentential forms after derivations, starting bottom up, therefore aaAbb, aAb, b
Simple Phrases: All right most derivations, therefore b

    b. bBab

Handle : Left most simple phrase, therefore ab
Phrases: Starting from bottom up, sentential forms after derivations, therefore bBab, bBA
Simple Phrases: Right most derivations, therefore ab

    c. aaAbBb



Not possible since the string cannot be generated by this grammar. Therefore its phrase, simple phrase or handle cannot be determined.

7. Use left-factoring and/or elimination of left recursion to convert the following grammars into LL(1) grammars. Assume that these grammars are unambiguous.

    a.

    E → E + T | T | E!
    T → int | (E)

    Answer:
    E → T E’
    E’ → + T E’ | ! E’ | ε
    T → int | (E)

    b.

    L → X | L;X
    X → int | string | (L)

    Answer:
    L → X L’
    L’ → ;X L’ | ε
    X → int | string | (L)

    c.

    P → P H 4 U | p
    H → h
    U → u | u P

    Answer:
    P → p P’
    P’ → H 4 U P’ | ε
    H → h
    U → u | u P

8. How many 8-digit binary numbers can be generated by the following grammar assuming the central pair of digits is always 11? (e.g 00011000 and 10111001 are part of the desired subset.

S → 0 S 0 | 1 S 0 | 0 S 1 | 1 S 1 | 0 1 | 1 1

Answer: 16 possible 8 bit numbers can be generated by the grammar

9. Let O = {⊕, ⊙, ⊗} be three right-associative infix binary operators in an expression language which has following precedence rules.
    a. ⊙ has the highest precedence and ⊕ has the lowest precedence with ⊗ having precedence strictly in between ⊙ and ⊕.
    b. The operators are left associative.

    Let I = {a, b, c} be the set of identifier tokens and B = {(,)} is the set of brackets that are allowed for grouping sub-expressions together and overriding precedence. Then T = I ∪ O ∪ B is the set of terminal tokens. Design a context-free grammar to generate all valid expressions in this language which allows for any expressions (with or without the use of brackets) to be unambiguously parsed in such a way that the precedence rules are respected.

    Answer:
    <assign> → <id> = <expr>
    <I> → a | b | c
    <expr> → <term> ⊕ <expr> | <term>
    <term> → <term> ⊗ <factor> | <factor>
    <factor> → <factory>⊙ <B> | <B>
    <B> → (<expr>) | <I>

10.
    a. $(((a > b)_1$ **xor** $c)_3$**or** $(d <= 17)_2)_4$
    b. $(- (a + b)_1)_2$

11.
    a. $(a * (b - (1 + c)_1)_2)_3$
    b. $(a * ((b - 1)_2 / (c$ **mod** $d)_1)_3)_4$
    c. $((a – b)_5 / (c \& (d * (e / (a– 3)_1)_2)_3)_4)_6$
    d. $(-(a$ **or** $(c = (d$ **and** $e)_1)_2)_3)_4$

12.
    a. Evaluated left to right

    $sum1 = (i_{old} / 2) + fun(\&i_{old})$
    $= (10 / 2) + (3 * (10 + 4)\ \ 1)$
    $= 5 + 41$
    $= 46$

    $sum2 = fun(\&j_{old}) + (j_{new} / 2)$

= (3 * (10 + 4) – 1) + (14 / 2)
= 41 + 7
= 48

b. Evaluated right to left

sum1 = (i_{new} / 2) + fun(&i_{old})
= 14 / 2 + (3 * (10 + 4) 1)
= 7 + 41
= 48

sum2 = fun(&j_{old}) + (j_{old} / 2)
= (3 * (10 + 4) – 1) + (10 / 2)
= 41 + 5
= 46

13.

Static-Scope rules the value of x = 5
Dynamic-Scope rules the value of x = 10

14.

sub1

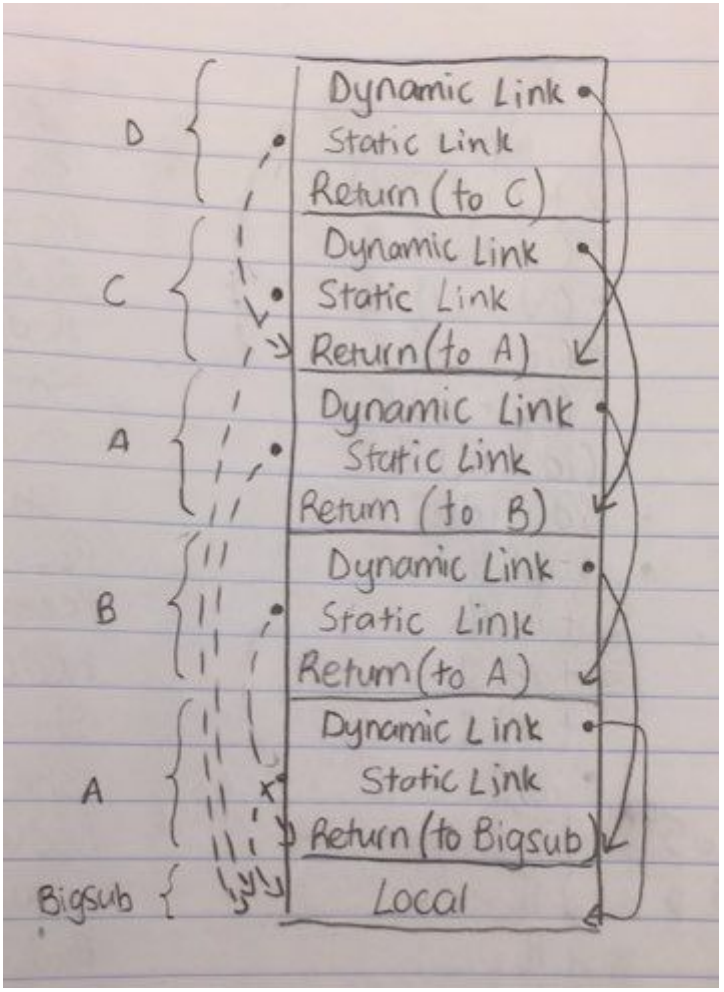
a declared in sub1
y declared in sub1
z declared in sub1
x declared in main

sub2

a declared in sub2
b declared in sub2
z declared in sub2
x declared in main
y declared in main

sub3

a declared in sub3
x declared in sub3
w declared in sub3
y declared in main
z declared in main

15. 

16. In the context of concurrent systems, explain busy waiting and itsa method whereby a task waits for a given event by continuously checking for that event to occur. What is the main problem with this approach?

    Busy-waiting in concurrent systems occurs when threads depend on the outcome of others to continue. Meaning that some threads may be put on hold until the result from other threads are available. The main problem with this approach is when the ready threads are the threads that the busy-wait loop is waiting on. This can make the busy-wait loop wait for a long amount of time, since the operating system has no way of knowing which thread the busy-waiting thread is waiting for. For short, busy-waiting in concurrent systems is bad because it could lead to extremely long wait times for the busy-waiting thread.

17. Sequence of events:
Performed by Task A
Fetch current value of Buf_size
Buf_size = 16
Perform Operation
16+2=18

Saving updated value
Buf_Size=18
Fetch current value of Buf_size
Buf_size = 18
Perform Operation
18-1=17
Saving the updated value
Buf_size=17
Final Value Buf_size = 17