Grammars and Reg-exp:

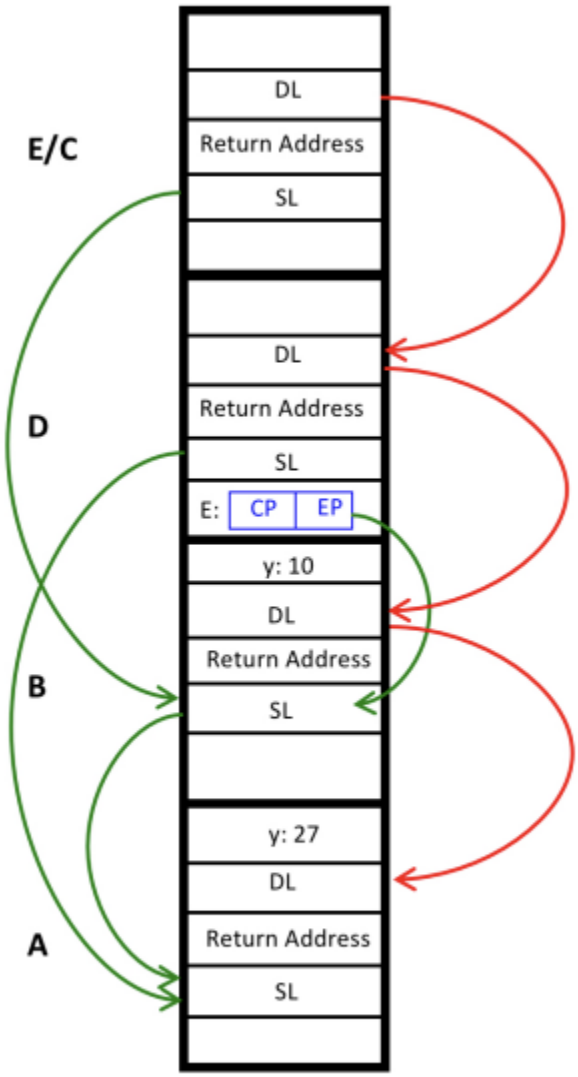
1. The set of all strings consisting of a's and b's whose length is an even number (i.e. consisting of an even number of characters).
   (aa | ab | ba | bb)*
2. The set of all strings representing integers and real numbers (e.g 27 and 34.68). There must be at least one digit before a decimal point and after a decimal point.
   [0-9][0-9]* ((.[0-9][0-9]*))
3. Define a regular expression for the set of all strings starting with an A, ending with a Z, and having any number of lower case letters and an even number of digits in between. For example, Abc34def5gh6Z would be in the set. Abc3def5gh6Z would not be in the set, because it has an odd number of digits
   A[a-z]*([0-9][a-z]*[0-9][a-z]*)*Z
4. Write a context free grammar that defines the set of all strings of the form "$a^m b^n c^n d^m$", for m ≥ 0 and n ≥ 1
   S →aSd | Q
   Q →bQc | bc
5. Write a CFG defining the set of all properly nested sequences of begin and end (e.g. begin end, begin begin end end, begin end begin end).
   S -> begin S end | SS | ∈

Static and Dynamic scoping:

1.
```
procedure A()
        y: integer := 27;
        procedure B()
        y: integer := 10;
                procedure C()
                begin
                        print(y);
                end;
        begin (* B *)
                D(C);
        end;
        procedure D(procedure E)
        begin
                E();
        end; (* D *)
begin (* A *)
        B();
end;
```

What does the program above print, assuming the language is statically scoped? Explain your answer

Ans: **10**

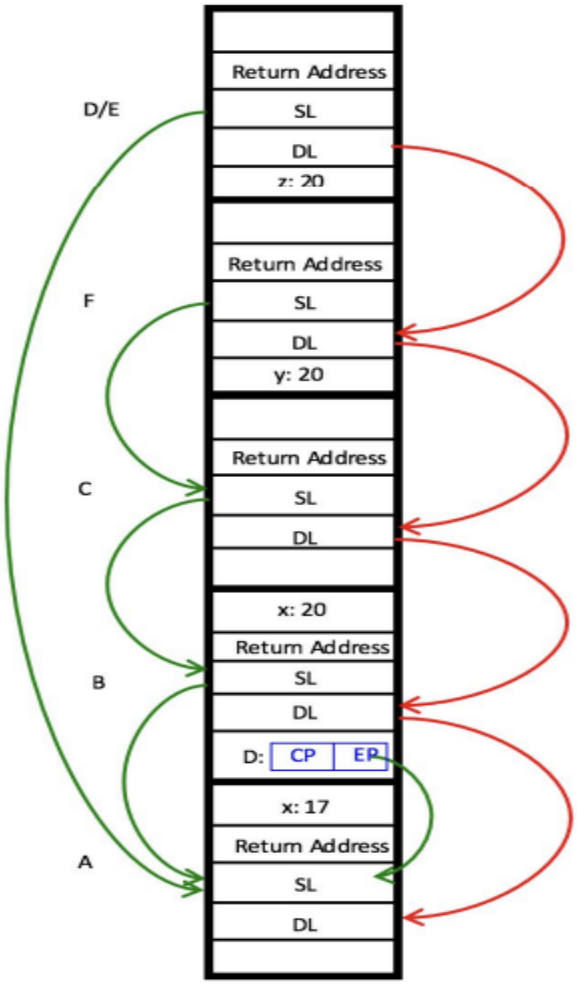


2. What does the program below print? Explain your answer.

```
procedure A()
        x: integer := 17;
        procedure B(procedure D)
                x: integer := 20;
                procedure C()
                        procedure F(y:integer)
                        begin (* F *)
                                D(y)
```

```
                end; (* F *)
        begin (* C *)
                F(x);
        end; (* C *)
begin (* B *)
        C();
end (* B *)
procedure E(z:integer)
begin (* E *)
        print(z, x);
end; (* E *)
begin (* A *)
        B(E);
end; (* A *)
```

a. What if it's statically scoped? **20 17**
b. What if it's dynamically scoped? **20 20**



Parameter passing methods:

1. x: integer := 2;
   y: array 1..3 of integer;
   i: integer;
   procedure f(a: integer)
   begin
   x:= x + 1;
   print(a);
   end;

   begin for i := 1 to 3
   do
   y[i] := i*2;
   done
   f(y[x]);
   end

a. What does the program print if the language uses pass-by-reference parameter passing?
   Ans: **4**
b. (b) What does the program print if the language uses pass-by-name parameter passing?
   Ans: **6**

2.
```
program foo;
    var i,j: integer;
    a: array[1..10] of integer;
    procedure f(x,y:integer)
    begin
        x := x * 4;
        i := i + 1;
        y := a[i] * 5;
    end
begin
    for j := 1 to 10 do a[j] = j;
    i := 2;
    f(i,a[i]);
    for j := 1 to 10 do print(a[j]);
end.
```

Ans:
Pass by Value: 1 2 3 4 5 6 7 8 9 10
Pass by Reference: 1 45 3 4 5 6 7 8 9 10
Pass by Value-Result: 1 15 3 4 5 6 7 8 9 10
Pass by Name: 1 2 3 4 5 6 7 8 45 10

Scheme:

1. Define a function, (average L), where L is a list of numbers, that computes the average of the elements of L.
   For example, > (average '(2 4 6 8)) = 5

   Answer:

   (define (average L) (

   letrec ((sum (lambda (L2)
   (if (null? L2)
   0
   (+ (car L2) (sum (cdr L2)))))))

   (/ (sum L) (length L))))

2. Define a function (bar x) that returns a function that takes a list L and adds x to each element of L, returning a list of the results.
   For example, > ((bar 3) '(1 2 3 4)) (4 5 6 7)

   Answer:
   (define (bar x)
   (lambda (L) (map (lambda (y) (+ x y)) L)))

3. Write the Scheme function (reduce f L), where f is a function and L is a list of the form (x1 x2 ... xN−1 xN), that computes (f x1 (f x2 ... (f xN−1 xN)...)).

   For example, (reduce + '(1 2 3 4)) returns the value of (+ 1 (+ 2 (+ 3 4))) = 10. You can assume that L has at least one element

   Answer:
   (define (reduce f L)
   (cond ((null? (cdr L)) (car L))
   (else (f (car L) (reduce f (cdr L))))))

4. Write an Scheme expression that calls reduce in the above example to return the largest number in L, where L is a list of numbers. That is, your expression should look like: (reduce ... L)

   Answer**: (reduce (lambda (x y) (if (> x y) x y)) L)**

Lambda calculus:

1. (λx. (+ x x)) ((λy. y) 3)
   Under normal order evaluation:
   (λx. (+ x x)) ((λy. y) 3) ⇒ (+((λy. y) 3)((λy. y) 3)) ⇒ (+ 3 ((λy. y) 3)) ⇒ (+ 3 3) δ ⇒ 6
   Under applicative order evaluation:
   (λx. (+ x x)) ((λy. y) 3) ⇒ (λx. (+ x x)) 3 ⇒ (+ 3 3) ⇒ 6

2. Reduce: (λx. x x) ((λy. y y) (λz.λw.3))
   Applicative order:
   (λx. x x) ((λy. y y) (λz.λw. 3))
   ⇒ (λx. x x) ((λz.λw. 3) (λz.λw. 3))
   ⇒ (λx. x x) (λw. 3)
   ⇒ (λw. 3) (λw. 3)
   ⇒ 3

   Normal order:
   (λx. x x) ((λy. y y) (λz.λw. 3))
   ⇒ ((λy. y y) (λz.λw. 3)) ((λy. y y) (λz.λw. 3))
   ⇒ ((λz.λw. 3) (λz.λw. 3)) ((λy. y y) (λz.λw. 3))
   ⇒ (λw. 3) ((λy. y y) (λz.λw. 3))
   ⇒ 3