1. Functional Programming:

Consider the following sum of 4 terms:

\[ a_1 + a_2 + a_3 + a_4 \]

The sum can be parenthesized in five ways:

\[ a_1 + (a_2 + (a_3 + a_4)) \]
\[ a_1 + ((a_2 + a_3) + a_4) \]
\[ (a_1 + a_2) + (a_3 + a_4) \]
\[ ((a_1 + a_2) + a_3) + a_4 \]
\[ (a_1 + (a_2 + a_3)) + a_4 \]

In general, the number of ways a sum of \( n \) terms can be parenthesized is given by the \((n - 1)\text{th}\) Catalan number.

The Catalan numbers are defined recursively as follows:

\[
C_0 = 1 \\
C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}
\]

Your task: Write a scheme program to compute the Catalan number \( C_n \), on input \( n \).

Hint: Use two functions: one to implement the base and general case for the Catalan numbers and another one to implement the summation. E.g.:

```scheme
; Catalan C_n
;
(define (C n) ... implementation here ... )

; Helper function to compute the required sum
;
(define (Csum i n) ... implementation here ... )
```

The first dozen Catalan numbers (index starts with zero) are:

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012

Another Hint: Re-write the second line in the definition as:

\[
C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1}
\]
Answer:

; C_0 = 1
;
; C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}
;
(define (Csum i n)
  (if (= i 0) (C n)
    else
    (+ (* (C i) (C (- n i))) (Csum (- i 1) n)))
  )
)

; Catalan

(define (C n)
  (if (= n 0) 1
    (Csum (- n 1) (- n 1))
  )
)

A sample session is shown below:

  gosh> (load "./catalan")
  #t
  gosh> (C 0)
   1
  gosh> (C 1)
   1
  gosh> (C 2)
   2
  gosh> (C 3)
   5
  gosh> (C 4)
   14
  gosh> (C 5)
   42
  gosh> (C 6)
   132
  gosh> (C 7)
   429
  gosh> (C 8)
   1430
  gosh> (C 9)
   4862
  gosh> (C 10)
   16796
  gosh> (exit)
Consider the graph $G = (V, E)$ in Figure 1 (left)

![Graph](image)

Figure 1: An Undirected Graph $G$ (left) and a Coloring of $G$ (right)

A **coloring** of a graph $G$ is an assignment of colors to the nodes such that if two nodes $v_i$ and $v_j$ are connected by an edge, then those two nodes must be different colors. The **chromatic number** of a graph is the minimum number of colors required to color it. The graph in Figure 1 (left) can be colored using four colors as shown in Figure 1 (right).

**Your Tasks:**

1. Write Prolog rules to verify the correctness of the coloring shown in Figure 1 (right). I.e.:
   (a) Write Prolog facts to express the connectivity of the graph in Figure 1 (left). Use the predicate `edge` e.g., `edge(v0,v1)`. Include only one pairing. I.e., do not include both `edge(v0,v1)` and `edge(v1,v0)` in your collection of edge facts.
   (b) Write a Prolog rule `connected(V,W)` to establish the symmetry of the edges in the undirected graph.
   (c) Write a set of Prolog facts named `color` to establish the coloring shown in Figure 1 (right).
   (d) Write a Prolog rule `wrong(V)` to decide if the color of vertex $V$ violates the coloring rules. *Hint:* Rule `wrong(V)` results in `true`, whenever vertex $V$ breaks the rules for coloring (i.e. has a `connected` vertex with the same color).
   (e) Write a Prolog rule `accept` to decide if the given coloring is acceptable. Notice that a coloring is acceptable if none of the vertices are “wrong”.

2. Modify your result from part 1 as follows:
   (a) Change the color of $v_1$ to yellow.
   (b) Run `accept` to show that this new coloring is also acceptable.
   (c) Change the color of $v_1$ to pink.
   (d) Run `accept` again to show that this new coloring is not an acceptable coloring.

**Hint:** You do not need any recursive rules for this problem.
Answer:

/* Edge facts. */
edge(v0,v1).
edge(v0,v2).
edge(v0,v3).
edge(v1,v2).
edge(v1,v4).
edge(v2,v3).
edge(v2,v4).
edge(v2,v5).
edge(v3,v5).
edge(v4,v5).

/* Symmetry rule for edges. */
connected( V, W ) :- edge( V, W ); edge( W,V ).

/* Color facts. */
color(v0,blue).
color(v1,green).
color(v2,pink).
color(v3,green).
color(v4,blue).
color(v5,yellow).

/* Coloring rules. */
wrong( V ) :- color(V,C),connected(V,W),color(W,D),C == D.
accept :- \+ wrong( _ ).

A sample session is shown below:

gottlieb% swipl -s t.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- accept.
true.

If we change color(v1,green) to color(v1,pink) we obtain:

gottlieb% swipl -s f.pl

;.

?- accept.
false.