1. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```c
int x = p + q;
for (int i = 1; i <= r; i++)
{x = x * q;}
```

You can use `add(_, _)` (sum) and `mult(_, _)` (product) in this expression.

What is the value of this function when `p = 1`, `q = 2`, and `r = 2`?

Let's define:

- \( x(p, q, o) = p + q \)
- \( x(p, q, m+1) = x(p, q, m) \times q \)
- \( f(n_1, n_2, o) = f(n_1, n_2) \)
- \( f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m)) \)

So, by comparing them, we have \( p = n_1, q = n_2, g = \text{add}(x_1, x_2) \)

Let \( h = \text{mult}(x_3, x_4) \)

\( \Rightarrow \) \( x = \text{PR}(\text{add}(x_1, x_2), \text{mult}(x_3, x_4)) \)

- \( p = 1, q = 2, r = 2 \)
- \( x(1, 2, 0) = 1 + 2 = 3 \)
- \( x(1, 2, 1) = 3 \times 2 = 6 \)
- \( x(1, 2, 2) = 6 \times 2 = 12 \)
2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```c
int z = p + q;
for(int i = 1; i <= r; i++)
    {for (int j = 1; j <= s; j++)
        {z = z * q;}}
```

You can use add(., .) and mult(., .) in this expression.

What is the value of this function when p = q = r = s = 2?

First, we consider the inner loop.

```c
int z = z0;
for (int j = 1; j <= s; j++)
    {z = z * q;}
```

\[ z(z_0, q, 0) = z_0. \]

\[ z(z_0, q, m+1) = z(z_0, q, m) \times q \]

\[ f(n_1, n_2, 0) = g(n_1, n_2) \]

\[ f(n_1, n_2, m+1) = h(n_1, n_2, m, f(n_1, n_2, m)) \]

\[ \Rightarrow n_1 = z_0, \quad n_2 = q, \quad k = 2, \quad q = \pi_1^2, \quad h = \text{mult}(\pi_1^2, \pi_1^4) \]

\[ \Rightarrow \text{aux} = \text{PR}(\pi_1^2, \text{mult}(\pi_1^2, \pi_1^4)) \]

Then for the outer loop.

```c
int z = p + q;
for(int i = 1; i <= r; i++)
    {z = aux(z, q, s)}
```

\[ z(p, q, s, 0) = p + q. \]

\[ z(p, q, s, m+1) = \text{aux}(z(p, q, s, m), q, s) \]

\[ f(n_1, n_2, n_3, 0) = g(n_1, n_2, n_3) \]

\[ f(n_1, n_2, n_3, m+1) = h(n_1, n_2, n_3, m, f(n_1, n_2, n_3, m)) \]

\[ \Rightarrow n_1 = p, \quad n_2 = q, \quad n_3 = s. \]

\[ g = \pi_1^3 + \pi_1^2 = \text{add}(\pi_1^3, \pi_1^2) \]

\[ h = \text{aux}(\pi_1^2, \pi_1^5, \pi_1^3) \]

\[ \Rightarrow z = \text{PR}(\text{add}(\pi_1^3, \pi_1^2), \]

\[ \text{aux}(\pi_1^5, \pi_1^3, \pi_1^5)) \]
If \( p = q = r = s = 2 \) then
3. Translate, step-by-step, the following primitive recursive function into a for-loop:

\[ f = \sigma(\text{PR}(\text{add}(\pi_2^2, \sigma(0)), \text{add}(\pi_4^1, \pi_3^4))). \]

For this function \( f \), what is the value \( f(2, 0, 1) \)?

Let \( F = \text{PR}(\text{add}(\pi_2^2, \sigma(0)), \text{add}(\pi_4^1, \pi_3^4)) \)

then \( f = \sigma(F) \)

from \( F = \text{PR}(\text{add}(\pi_2^2, \sigma(0)), \text{add}(\pi_4^1, \pi_3^4)) \)

we have \( k = 2 \) \quad \gamma(n_1, n_2) = \text{add}(\pi_2^2, \sigma(0)) = n_2 + 1; \)

\( h(n_1, n_2, m; F(n_1, n_2, m)) = \text{add}(\pi_4^1, \pi_3^4) = n_1 + m. \)

\[
\begin{align*}
\text{int } F &= n_2 + 1, \\
\text{for } (\text{int } i = 1; i < m; i++) \\
& \quad \text{if } F = n_1 + i - 1; \}
\]

\[
\begin{align*}
\text{int } f &= F + 1, \\
F(2, 0, 0) &= 0 + 1 = 1, \\
F(2, 0, 1) &= 2 + 0 \\ &= 1 = 2, \\
F(2, 0, 1) &= F(2, 0, 1) + 1 = 2 + 1 = 3.
\end{align*}
\]
4-5. Prove, from scratch, that the function \( f(p, q) = p \% (q / p) \) is primitive recursive. Start with the definitions of a primitive recursive function, and use only this definition in your proof -- do not simply mention results that we proved in class, prove them.

**Definition:** A function is called primitive recursive if it can be obtained from 0, \( \sigma \) and \( \pi_i \) by using composition and primitive recursion.

1. **add (+)** is P.R.
   
   int \( \text{sum} = a \).
   
   +\( (n, 0) = g(n) \) \quad +\( (a, 0) = a \)
   
   for \( (\text{int } i=1; i<=b; i++) \) \( +\( (n, m+1) = h(n, m, f(n, m)) \) \quad +\( (a, m+1) = f(a, m) + 1 \)
   
   \( \frac{1}{2} \text{sum} = \text{sum} + 1; \}
   
   \( \Rightarrow k = 1, a = n, g = \pi_1^1, h = \sigma(\pi_3^3) \)
   
   \( \Rightarrow \text{add} = \text{PR}(\pi_1, \sigma \circ \pi_3^3) \), so \( \text{add} (+) \) is P.R.

2. **mult (*)** is P.R.
   
   int \( \text{mult} = 0 \).
   
   +\( (n, 0) = g(n) \) \quad +\( (a, 0) = 0 \)
   
   for \( (\text{int } i=1; i<=b; i++) \) \( +\( (n, m+1) = h(n, m, f(n, m)) \) \quad +\( (a, m+1) = f(a, m) + a \)
   
   \( \frac{1}{2} \text{mult} = \text{mult} + a; \}
   
   \( \Rightarrow k = 1, a = n, g = 0, h = \text{add}(\pi_1^1, \pi_3^3) \)
   
   \( \Rightarrow \text{mult} = \text{PR}(0, \text{add}(\pi_1^1, \pi_3^3)) \), so \( \text{mult} (*) \) is P.R.

3. **prev** is P.R.
   
   \( \text{prev}(a) = \begin{cases} 0 & \text{if } a = 0 \\ a-1 & \text{if } a = 1, 2, 3 \end{cases} \)
   
   +\( (0) = g \) \quad +\( (0) = 0 \)
   
   +\( (m+1) = h(m, f(m)) \) \quad +\( (m+1) = m \)
   
   \( \Rightarrow k = 0, g = 0, h = \pi_1^1 \)
   
   \( \Rightarrow \text{prev} = \text{PR}(0, \pi_1^1) \), so \( \text{prev} \) is P.R.

4. **sub (-)** is P.R.
   
   int \( \text{sub} = a \).
   
   +\( (n, 0) = g(n) \) \quad +\( (a, 0) = a \)
   
   for \( (\text{int } i=1; i<=b; i++) \) \( +\( (n, m+1) = h(n, m, f(n, m)) \) \quad +\( (a, m+1) = \text{prev}(f(a, m)) \)
   
   \( \frac{1}{2} \text{sub} = \text{prev}(\text{sub}); \}
   
   \( \Rightarrow k = 1, a = n, g = \pi_1^1, h = \text{prev} \circ \pi_3^3 \)
   
   \( \Rightarrow \text{sub} = \text{PR}(\pi_1^1, \text{prev} \circ \pi_3^3) \), so \( \text{sub} (-) \) is P.R.

5. **And (\&\&)** is P.R.
   
   \[
   \begin{array}{c|ccc}
   \text{a} & 0 & 1 \\
   \hline
   0 & 0 & 0 \\
   1 & 0 & 1 \\
   \end{array}
   \]
   
   so \( \text{a}\&\&b = a \times 1 \), since \( \times \) is P.R., so \( \&\& \) is P.R.
6. If P then Q, else R is p.r.
   
   If P, then Q else R can be expressed as
   
   \[ P \land Q + (1 - P) \land R \]  since, +, -, \*, \# are p.r.
   
   so if P then Q, else R is p.r.

7. equal to 0 (eq(0)) is p.r.
   
   \[ eq(0) = 1 \land f(0) = 1 \land f(0) = g \]
   
   \[ eq(m+1) = 0 \land f(m+1) = 0 \land f(m+1) = h(m, f(m)) \]
   
   \[ \Rightarrow k = 0, \ g = 1, \ h = 0 \]
   
   \[ \Rightarrow eq = PR(S(0, 0), 0) \] so eq(0) is p.r.

8. equal (\(=\)) is p.r.
   
   \[ a = b \iff (a \leq b) \& (b \leq a) \iff eq(a - b) \& eq(b - a) \]
   
   \[ \iff eq(a - b) \land eq(b - a) \] since \(\land\), \(-\), and eq are p.r.
   
   so \(=\) is p.r.

9. rem (\(\%\)) is p.r.
   
   \[ rem(a, 0) = 0 \]
   
   \[ rem(a, m+1) = \begin{cases} 
   a & \text{if } (rem(a, m) + 1) = a \\
   \text{return } (rem(a, m) + 1) & \text{else return } (rem(a, m) + 1) 
   \end{cases} \]
   
   since if P then Q else R, + and \(=\) are p.r. so rem is p.r.

10. div (\(/\)) is p.r.
   
   \[ div(a, 0) = 0 \]
   
   \[ div(a, m+1) = \begin{cases} 
   a & \text{if } (rem(a, m+1) = 0) \text{ return } (div(a, m) + 1) \\
   \text{return } div(a, m) & \text{else return } div(a, m) 
   \end{cases} \]
   
   since if P then Q else R, \(=\) and + are p.r. so div is p.r.
11. \( f(p.q) = p \% (q \div p) \)

since we have proved that both \( \text{rem}(\%) \) and \( \text{div}(\) are p.r. so \( f(p.q) \) can be obtained from \( \sigma \) by using composition and primitive recursion, it means, \( f(p.q) \) is p.r.
6. Prove that the following function \( f(p, q) \) is \( \mu \)-recursive: \( f(p, q) = p \mod \left( \frac{q}{p} \right) \) when each of the values \( p \) and \( q \) is either 1 or 2, and \( f(p, q) \) is undefined for other pairs \((p, q)\).

There are several cases.

- If \( p = 1, q = 1 \) then \( q/p = 1/1 = 1, p \mod \left( \frac{q}{p} \right) = 1 \mod 1 = 0 \).
- If \( p = 1, q = 2 \) then \( q/p = 2/1 = 2, p \mod \left( \frac{q}{p} \right) = 1 \mod 2 = 1 \).
- If \( p = 2, q = 1 \) then \( q/p = 1/2 = 0, p \mod \left( \frac{q}{p} \right) = 2 \mod 0 = \text{undefined} \).
- If \( p = 2, q = 2 \) then \( q/p = 2/2 = 1, p \mod \left( \frac{q}{p} \right) = 2 \mod 1 = 0 \).

So \( f(p, q) = \begin{cases} 
0, & \text{if } (p = 1 \text{ and } q = 1) \text{ or } (p = 2, q = 2) \\
1, & \text{if } (p = 1 \text{ and } q = 2) \\
\text{undefined}, & \text{otherwise} 
\end{cases} \)

\[
\begin{align*}
\mu(n) = \{ & (p = 1, q = 1, m = 0) \} \\
& \quad \cup \{ (p = 2, q = 2, m = 0) \} \\
& \quad \cup \{ (p = 1, q = 2, m = 1) \}
\end{align*}
\]

So \( f(p, q) \) is \( \mu \)-recursive.
7. Translate the following μ-recursive expression into a while-loop:
\[ f(a, b) = \mu m. (m \cdot a = b). \]
For this function \( f \), what is the value of \( f(2, 4) \)? \( f(2, 5) \)?

```c
int m = 0;
while (! (m + a == b))
    m++;
```

```c
to f(2, 4)
    int m = 0;
    m * a = 0 * 2 = 0 \neq 4
    m = 1
    m * a = 1 * 2 = 2 \neq 4
    m = 2
    m * a = 2 * 2 = 4 \Rightarrow f(2, 4) = 4 = m.
```

```c
to f(2, 5) since no m value will meet the stop condition so it is undefined the loop will not stop.
```
8-9. Suppose that someone comes up with a new proof that not every computable function is primitive recursive, by providing a new example of a function $N(n)$ which is computable but not primitive recursive. What if, in addition to $0$, $\pi^k$, and $\sigma$, we also allow this new function $N(n)$ in our constructions? Let us call functions that can be obtained from $0$, $\pi^k$, $\sigma$, and $N(n)$ by using composition and primitive recursion $N$-primitive recursive functions. Will then every computable function be N-primitive recursive? Prove that your answer is correct.

In order to prove that there exist a computable function which is not $N$-primitive recursive ($N$-p.r), at first, we need an auxiliary notion of $N$-p.r code.

We start with an expression then.
1. We translate all symbols into ASCII code, for example $
\sigma \rightarrow \backslash \sigma \, m, \ \pi^k \rightarrow \backslash p; \backslash k \_ \_ 1, \ \sigma \rightarrow \backslash \sigma$
2. Use ASCII code to translate them into a binary string of Os and Is.
3. Put a 1 before the resulting binary string and interpret this string as a binary integer.
This integer is called $N$-p.r code.

Lemma: There exist an algorithm that given a natural number $c$
1. It checks whether $c$ is a valid $N$-p.r code.
2. If $c$ is a valid $N$-p.r code, return a Java program that computes the corresponding $N$-p.r function, this program will be denoted as $f_c$.

Let's define a new function
\[
    f(c) = \begin{cases} 
    f_c(c) + 1 & \text{if } c \text{ is a valid } N\text{-p.r code} \\
    0 & \text{otherwise,} 
    \end{cases}
\]

Let's prove that this function is computable, but not $N$-p.r.
First, let's prove that $f$ is computable.

Given $C$,

- Check whether $C$ is a valid $N$-P.Y code
  - Yes: use the algorithm in lemma
  - No: return 0

Design a program $fc$

Apply $fc$ to $C$

Add 1 to the result

Return $fc(C) + 1$.

So, we have proved that $f$ is computable.

Now, let's prove that $f$ is not $N$-P.Y by contradiction.

We assume that $f$ is $N$-P.Y, then it has a $N$-P.Y code, we denote it by $Co$. By lemma, $f_{Co}$ computes the function $f$. This means that for every input $n$, $f_{Co}(n) = f(n)$. In particular, if we let $Co$ be the input, then we have $f_{Co}(Co) = f(Co)$. By the definition of $f(Co)$ since $Co$ is a $N$-P.Y code, we have $f_{Co}(Co) = f_{Co}(Co) + 1$, so $f_{Co}(Co) = f_{Co}(Co) + 1 \Rightarrow 0 = 1$, which is a contradiction, thus, $f$ is not a $N$-P.Y function.
10. Design Turing machines for computing \( n + 1 \) in unary and in binary codes.

For unary:

- **Start**, \# \rightarrow **Working**, R.
- **Working**, 1 \rightarrow R.
- **Working**, \# \rightarrow 1, return, L
- return, 1 \rightarrow L
- return, \# \rightarrow halt.

For binary:

- **Start**, \# \rightarrow **Working**, R.
- **Working**, 0 \rightarrow 1, return, L
- **Working**, 1 \rightarrow 0, right, R.
- right, 1 \rightarrow 0, R.
- Right, 0 \rightarrow 1, return, L
- Right, \# \rightarrow 1, return, L
- return, \# \rightarrow halt.
- return, 1 \rightarrow L
- return, 0 \rightarrow L.