Theory of Computations,
Test 1 for the course
CS 5315, Spring 2020

Name: ____________________________

Up to 5 handwritten pages are allowed.

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`?

---

**Solution:**

In mathematical terms:

\[
X(p, q, 0) = p + q
\]

\[
X(p, q, m + 1) = X(p, q, m) \times q
\]

The general expression for \( k = 2 \) is:

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

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

In this case:

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

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

Thus we have:

\[
g = n_1 + n_2 = \text{sum}(T_1^2, T_2^2)
\]

\[
h = \text{mult}(T_1^4, T_2^4)
\]

\[
\therefore X(p, q, m) = PR[\text{sum}(T_1^0, T_2^0), \text{mult}(T_1^4, T_2^4)]
\]

```
4 8
p
2
```

```
1 2
q
```

```
4 2
x
```

The value for \( p = 1 \), \( q = 2 \), and \( r = 2 \) is 12.

```
X = 12
```
2. Translate, step-by-step, the following for-loop into a primitive recursive expression:

```
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?

**Solve:**

First we breakdown the nested loop

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

```
int z = z;
for(int j = 1; j <= s; j++)
  z = z * q;
```

In general, for \( k = 2 \)

\[
\begin{align*}
  f(n_1, n_2, 0) &= g(n_1, n_2) \\
  f(n_1, n_2, m+1) &= h(n_1, n_2, m, f(n_1, n_2, m))
\end{align*}
\]

```
\[
\begin{align*}
  Z(p, q, s, 0) &= p + q \\
  Z(p, q, s, m+1) &= aux(z(p, q, s, m), q, s)
\end{align*}
\]

In general, for \( k = 3 \)

\[
\begin{align*}
  f(n_1, n_2, n_3, 0) &= g(n_1, n_2, n_3) \\
  f(n_1, n_2, n_3, m+1) &= h(n_1, n_2, n_3, m, f(n_1, n_2, n_3, m))
\end{align*}
\]

```
\[
\begin{align*}
  j &= \Pi_1^3 + \Pi_2^3 \\
  h &= aux(\Pi_5^5, \Pi_2^5, \Pi_3^5) \\
  Z &= aux(\Pi_2^5, \Pi_5^5, \Pi_3^5)
\end{align*}
\]

In general, for \( k = 4 \)

\[
\begin{align*}
  f(n_1, n_2, n_3, n_4, 0) &= g(n_1, n_2, n_3, n_4) \\
  f(n_1, n_2, n_3, n_4, m+1) &= h(n_1, n_2, n_3, n_4, m, f(n_1, n_2, n_3, n_4, m))
\end{align*}
\]

```
\[
\begin{align*}
  j &= \Pi_1^3 + \Pi_2^3 \\
  h &= aux(\Pi_5^5, \Pi_2^5, \Pi_3^5) \\
  Z &= aux(\Pi_2^5, \Pi_5^5, \Pi_3^5)
\end{align*}
\]

In general, for \( k = 5 \)

\[
\begin{align*}
  f(n_1, n_2, n_3, n_4, n_5, 0) &= g(n_1, n_2, n_3, n_4, n_5) \\
  f(n_1, n_2, n_3, n_4, n_5, m+1) &= h(n_1, n_2, n_3, n_4, n_5, m, f(n_1, n_2, n_3, n_4, n_5, m))
\end{align*}
\]

```
\[
\begin{align*}
  j &= \Pi_1^3 + \Pi_2^3 \\
  h &= aux(\Pi_5^5, \Pi_2^5, \Pi_3^5) \\
  Z &= aux(\Pi_2^5, \Pi_5^5, \Pi_3^5)
\end{align*}
\]
\[
Z = PR\left[\text{Sum}(\Pi_1^3, \Pi_2^3) \times \left( PR\left[\Pi_1^2, \text{mul}(\Pi_4^4, \Pi_2^4)\right] \right) \left( \Pi_5^5, \Pi_2^5, \Pi_3^5 \right)\right]
\]

\[
\begin{array}{cccc}
2 & 2 & 2 & 2 \\
P & q & r & s \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 3 & 1 & 3 \\
& 3 & 1 & \\
& & & \\
& & & \\
\end{array}
\]

\[Z(2,2,2,2) = 64\]

- \(j = 0\), \(Z = 2+2 = 4\)
- \(j = 1\), \(Z = 4\times2 = 8\)
- \(j = 1\), \(Z = 8\times2 = 16\)
- \(j = 2\), \(Z = 16\times2 = 32\)
- \(j = 2\), \(Z = 32\times2 = 64\)
3. Translate, step-by-step, the following primitive recursive function into a for-loop:

\[ f = \sigma(PR(add(x^2, \sigma(0)), add(x^4, x^3))). \]

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

Let \( F = PR[add(x^2, \sigma(0)), add(x^4, x^3)] \)

\[ f = 6 \cdot F \]

\( \Rightarrow \) The general formula.

\[
\begin{align*}
f(n, n_2, \ldots, n_k, 0) &= g(n, n_2, \ldots, n_k) \\
f(n, n_2, \ldots, n_k, m+1) &= h(n, n_2, \ldots, n_k, m, f(n, n_2, \ldots, n_k, m)) \\
\end{align*}
\]

\( \text{int } f = g(n, \ldots, n_k) \)

\( \text{for (int } i = 1; \ i \leq m; \ i++) \)

\( \text{if } F = h(n, \ldots, n_k, m, f) \) \{

\[ \text{int } F = n + 1 \]

\( \text{for (int } i = 1; \ i \leq m; \ i++) \)

\( F = F + (i - 1) \) \}

\( \text{int } F = F + 1 \)

\[ F(2, 0, 0) = 0 + 1 = 1 \\
F(2, 0, 1) = 2 + 0 = 2 \\
F(2, 0, 1) = F(2, 0, 1) + 1 = 2 + 1 = 3 \\
\]
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.

**Sln:** Definition: A function is called primitive recursive (PR) if it can be obtained from 0, 1, and \( \pi_1^1 \) by using composition (\( \circ \)) and primitive recursion (PR).

To prove \( f(p, q) = p \% (q / p) \) is primitive recursive we have to prove that:

\( \Rightarrow \) Remainder (\( \% \)) is PR
\( \Rightarrow \) Division (\( / \)) is PR

\( \Rightarrow \) Remainder is PR.

\[
\begin{align*}
\text{rem}(0, 0) & = 0 \\
\text{rem}(a, m+1) & = \begin{cases} 
\text{rem}(a, m) + 1 & \text{if } \text{rem}(a, m) + 1 < a \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
f(n, 0) & = g(n) \\
f(n, m+1) & = h(n, m, f(n, m)) \\
f(n, 1) & = 0 \\
f(n, m+1) & = \begin{cases} 
\text{if } f(n, m) < n, \text{ then } f(n, m) + 1 & \text{else } 0
\end{cases}
\end{align*}
\]

Thus,

\[
\text{rem} = \text{PR}(0, \text{if } 6 \bigcup 3 < \bigcup 1 \text{ then } 6 \bigcup 3 \text{ else } 0)
\]

Thus remainder will be PR if:

\( \text{if } p(a) \text{ then } f(a) \text{ else } g(a) \) is PR

Sum is PR < is PR.

\( \Rightarrow \) If \( p(a) \) then \( f(a) \) else \( g(a) \) is PR:

Mathematically:

\[
p(a), f(a) + (1 - p(a)) \times g(a)
\]

Proof:

\[
\begin{align*}
\text{if } p(a) = 1 & \text{ then } f(a) \\
\text{if } p(a) = 0 & \text{ then } g(a)
\end{align*}
\]

Thus we have \( \text{if } p(a) \text{ then } f(a) \text{ else } g(a) \) is PR provided \( p(a), f(a) \) and \( g(a) \) are PR, sum is PR, subtraction is PR, product is PR.
⇒ \text{Sum is P.R.}\hspace{1cm} a+b = a+1+1 \ldots +1 \hspace{1cm} b\text{-times}\hspace{1cm} \text{int sum=0}\hspace{1cm} \text{for} \hspace{0.1cm} (i=1; i < b; i++)\hspace{1cm} \text{? sum=sum+i} \hspace{1cm} \text{? sum=PR(TT', 0'')}\hspace{1cm} \text{Thus sum=PR(TT', 0')}\hspace{1cm} \text{Prev(n) is P.R.}\hspace{1cm} \text{prev(0) = 0}\hspace{1cm} \text{prev(m+1) = m}\hspace{1cm} f(0) = 0\hspace{1cm} f(m+1) = h(m, f(m))\hspace{1cm} f(0) = 0\hspace{1cm} f(m+1) = h(m, f(m))\hspace{1cm} \text{Thus prev(n) = PR(0, TT')}\hspace{1cm} \therefore \text{Subtraction is P.R.}\hspace{1cm} a-b = a-1-1 \ldots -1 \hspace{1cm} b\text{-times}\hspace{1cm} \text{int sub=0}\hspace{1cm} \text{for} \hspace{0.1cm} (i=1; i < b; i++)\hspace{1cm} \text{? sub=prev(sub)} \hspace{1cm} \text{? sub(sub, 0) = 0}\hspace{1cm} \text{sub(m+1) = sub(m, f(m))}\hspace{1cm} f(0) = 0\hspace{1cm} f(m+1) = h(m, f(m))\hspace{1cm} f(0) = 0\hspace{1cm} f(m+1) = h(m, f(m))\hspace{1cm} \text{Thus sub(sub, 0) = PR(TT', prev(TT'''))}\hspace{1cm} \text{Thus we have Subtraction is P.R. provided prev(n) is P.R.}\hspace{1cm} \Rightarrow \text{< is P.R.}\hspace{1cm} \therefore a < b \iff a-b = 0 \iff \text{eq0(a-b)}\hspace{1cm} \text{to prove < is P.R. we have to prove that eq0 is P.R.}\hspace{1cm} \Rightarrow \text{eq0 is P.R.}\hspace{1cm} \text{eq0(0) = 1}\hspace{1cm} \text{eq0(1) = 0}\hspace{1cm} f(0) = 1\hspace{1cm} f(m+1) = 0\hspace{1cm} \text{eq0 = PR(0, 0, 0)}\hspace{1cm} \Rightarrow \text{Next page for 2'}
2) Division is p.v.:

\[
\text{div}(a, 0) = 0
\]
\[
\text{div}(a, m+1) = \begin{cases} 
\text{div}(a, m) + 1 & \text{if } \text{rem}(a, m+1) = 0 \\
\text{div}(a, m) & \text{otherwise}
\end{cases}
\]

\[
f(n, 0) = 0
\]
\[
f(n, m+1) = \begin{cases} 
q & \text{if } \text{rem}(n, m+1) = 0 \text{ then } f(n, m) + 1 \\
\text{else } f(n, m)
\end{cases}
\]

Thus \( \text{div} = \text{PR}(0, q, \text{rem}(a, m+1) = 0 \text{ then } 6o \Pi^3_3 \text{ else } \Pi^3_3) \)

Thus division is p.v. provided remainder is p.v. (which we have already proved), \( n \leq q \leq a \) then \( f(a) \) else \( q(a) \) is p.v. (which we have also already proved), and \( = \) is p.v.

\[= \] is p.v.:

\[a = b \iff (a \leq b) \& f(b \leq a)\]

To prove = is p.v. we have to prove \& is p.v. and \( \leq \) is p.v.

\[\Rightarrow \] \& is p.v.:

\& is product so it is p.v.

\[\Rightarrow \] \( \leq \) is p.v.:

\[a \leq b \iff a \div b = 0 \iff \emptyset o(a \div b)\]

Thus \( \leq \) is p.v.

Hence we can say that \( p \div (a/p) \) is p.v.
6. Prove that the following function $f(p, q)$ is μ-recursive: $f(p, q) = \frac{p}{q}$ when each of the values $p$ and $q$ is either 1 or 2, and $f(p, q)$ is undefined for other pairs $(p, q)$.

$$f(p, q) = \begin{cases} p \div q \div (p \div q) & \text{if } p \in \{1, 2\} \text{ and } q \in \{1, 2\} \\ \text{Undefined otherwise} \end{cases}$$

I don't think this function would be defined for $P=2$ and $q=1$ as we would have $f(2, 1) = 2 \div (1/2)$ for integer division $(1/2)$ in 0 and $2 \div 0$ is undefined. Assuming it is $2 \div 0.5$ and thus equal to 0 the solution would be as given below; however, we cannot have non-natural numbers like 0.5.

$$P \div (2/q) = \begin{cases} 0 & \text{if } p = 1 \text{ and } q = 1 \\ 1 & \text{if } p = 2 \text{ and } q = 1 \\ 0 & \text{if } p = 2 \text{ and } q = 2 \\ 1 & \text{if } p = 1 \text{ and } q = 2 \\ 0 & \text{if } p = 1 \text{ and } q = 2 \end{cases}$$

$$\mu m \left[ (p = 1 \land q = 1 \land m = 0) \land (p = 2 \land q = 2 \land m = 0) \land (p = 1 \land q = 2 \land m = 1) \lor (p = 2 \land q = 2 \land m = 1) \right]$$

$P \div (2/q)$ calculation:

- $p=1, q=1 \rightarrow \div 1\div(1/1) = 1 \div 1 = 0$
- $p=2, q=1 \rightarrow 2 \div (1/2) = 2 \div 0 \rightarrow$ *This should be undefined for integer division, not sure how this function would be defined.* If we assume $2/2 = 0.5 + 2 \mod 0.5 = 0$, then the result would be as shown above.
- $p=1, q=2 \rightarrow 1 \div (2/1) = 1 \div 2 = 1$
- $p=2, q=2 \rightarrow 2 \div (2/2) = 2 \div 1 = 0$
7. Translate the following μ-recursive expression into a while-loop:
   \( f(a, b) = \mu m.(m \times a = b) \).

   For this function \( f \), what is the value of \( f(2, 4) \)? \( f(2, 5) \)?

   **Solution:**

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

   \( \Rightarrow f(2, 4) \)?

   \[ m \begin{array}{c} a \ b \\ 0 \times 2 = 4 \times \text{Not Satisfied} \\ 1 \times 2 = 4 \times \text{Not Satisfied} \\ 2 \times 2 = 4 \times \checkmark \text{Satisfied (exit loop)} \end{array} \]

   Thus \( f(2, 4) = 2 \)

   \( \Rightarrow f(2, 5) \)?

   \[ m \begin{array}{c} a \ b \\ 0 \times 2 = 5 \times \text{Not Satisfied} \\ 1 \times 2 = 5 \times \text{Not Satisfied} \\ 2 \times 2 = 5 \times \text{Not Satisfied} \\ 3 \times 2 = 5 \times \text{Not Satisfied} \end{array} \]

   (Loop runs continuously)

   \( f(2, 5) \) is undefined

   This loop never ends as no integer value of \( m \) will satisfy the given condition thus \( f(2, 5) \) is undefined. This while loop will never end.
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, \( \pi^k_i \), and \( \sigma \), we also allow this new function \( N(n) \) in our constructions? Let us call functions that can be obtained from 0, \( \pi^k_i \), \( \sigma \), and \( N(n) \) by using composition and primitive recursion \( N\text{-}primitive \text{ recursive} \) functions. Will then every computable function be \( N\text{-}primitive \text{ recursive} \)? Prove that your answer is correct.

\textbf{S\textsc{ol}n:} \\
Before we start the proof we need to describe the following two things:

\underline{Definition of a \( N\text{-}pr \) code:} we have stated that a \( N\text{-}pr \) function can be obtained from 0, 6, \( \pi^k_i \), and \( N(n) \) using composition (1) and primitive recursion (2) and every \( N\text{-}pr \) function can be described by an expression containing (1), 0, 6, \( \pi^k_i \), and \( N(n) \): Now to define a \( N\text{-}pr \) code we start with this expression and then assign an integer number to this expression in following way:

1. \( 6 \rightarrow \backslash \sigma \text{ma} \)
   \( \pi^k_i \rightarrow \backslash \nu \text{pu} \text{nu} \)
   \( 0 \rightarrow \backslash \text{circ} \)

2. Now we use ASCII to transform each symbol into 0's and 1's
   \( \text{eg:} \ P \rightarrow 01000000 \)
   \( 1 \rightarrow 01010000 \)

3. Now we place 1 in front (so that we can re-construct the string back)

4. Finally interpret this binary string as an integer and this integer will be the \( N\text{-}pr \) code for the corresponding expression.
Lemma (w/o proof): there exists an algorithm that given a natural number $c$, checks whether $c$ is a N-pr-code of some N-pr expression and if yes returns a java program for computing the corresponding N-pr expression. The java program will be denoted by $f_c$.

How this algorithm works:
- convert the natural number to binary string
- strip off the first 1
- check if each byte is ASCII character
- check if all back symbols are correct
- compiles to get $f_c$

⇒ Main part of the proof:
Let us define the following function
$$f(c) = egin{cases} f_c(c) + 1 & \text{if } c \text{ is a valid N-pr-code} \\ 0 & \text{otherwise} \end{cases}$$

How can we compute $f(c)$?

To prove that this function $f(c)$ is not N-pr we will use proof by contradiction.

Let us assume that $f(c)$ is N-pr and let us deduce a contradiction from this assumption.

Since $f(c)$ is N-pr it has a N-pr code, let us denote this code by $c_0$.
for this N-pr code the algorithm from lemma will produce a Java program $f_{c_0}$

$$\forall c \left( f_{c_0}(c) = f(c) \right)$$

In particular for $c = c_0$

$$f_{c_0}(c_0) = f(c_0) = 0$$

On the other hand, since $c_0$ is a valid N-pr code from the definition of function $f(c)$, we have

$$f(c) = f_{c_0}(c) + 1$$

From (0) and (11) we have

$$f_{c_0}(c_0) = f(c_0) + 1$$

$$0 = 1$$

Thus we have a contradiction which means that our assumption that the function is N-pr was wrong hence we conclude that $f(c)$ is not N-pr.
10. Design Turing machines for computing \( n + 1 \) in unary and in binary codes.

⇒ In Unary:

We have

\[ \begin{array}{c}
\text{Start} \\
\text{Step 1} \\
\text{Step 2} \\
\text{Step 3} \\
\text{Step 4} \\
\text{Step 5} \\
\end{array} \]

We want

\[ \begin{array}{c}
\text{Start} \\
\text{Working} \\
\text{Working} \\
\text{Working} \\
\text{Return} \\
\text{Return} \\
\end{array} \]

Directives:

\[ \begin{array}{c}
\text{Start} \# \rightarrow \text{working, R} \\
\text{working, } \# \rightarrow \text{R} \\
\text{working, } \# \rightarrow 1, \text{Return, L} \\
\text{Return, } 1 \rightarrow \text{L} \\
\text{Return, } \# \rightarrow \text{halt} \\
\end{array} \]

⇒ In Binary:

Motivation: In binary addition of 1 means we move from least significant bit to most significant bit changing all the 1’s to 0’s until we get a 0 and we change that 0 to 1 and return.

Also in turing machine the least significant bit is stored first i.e., numbers are stored in reverse order.

\[ \begin{array}{c}
\text{Start} \\
\text{Step 1} \\
\text{Step 2} \\
\text{Step 3} \\
\text{Step 4} \\
\text{Step 5} \\
\text{Step 6} \\
\text{Step 7} \\
\text{Step 8} \\
\text{Step 9} \\
\text{Step 10} \\
\text{Step 11} \\
\text{Step 12} \\
\end{array} \]

We have

\[ \begin{array}{c}
\text{Start} \\
\text{Working} \\
\text{Working} \\
\text{Working} \\
\text{Return} \\
\text{Return} \\
\end{array} \]

We want

\[ \begin{array}{c}
\text{Start} \\
\text{Working} \\
\text{Working} \\
\text{Working} \\
\text{Return} \\
\text{Return} \\
\end{array} \]

Directives:

\[ \begin{array}{c}
\text{Start, } \# \rightarrow \text{R, Working} \\
\text{working, } \# \rightarrow 0, \text{ R} \\
\text{working, } 0 \rightarrow 1, \text{ Return, L} \\
\text{Return, } 0/1 \rightarrow \text{L} \\
\text{Return, } \# \rightarrow \text{halt} \\
\end{array} \]