CS 3360 Design and Implementation of Programming Languages

Summer 2017, Final Exam

Name: ________________________________

1a. Use bottom-up approach to parse the expression \((a \times b) + (c + d \times e) \times f + (g \times h)\).

1b. Use the dependency graph to check which operations can be performed in parallel. If you had unlimited number of processors, how many steps do we need to compute the value of this expression?

1c. Describe this expression in LISP prefix form.

\[
\begin{align*}
(a \times b) + (c + d \times e) \times f + (g \times h) & \\
1. r_1 & \leftarrow a \times b \\
r_1 + (c + d \times e) & \leftarrow f + (g \times h) \\
2. r_2 & \leftarrow d \times e \\
r_1 + (c + r_2) & \leftarrow f + (g \times h) \\
3. r_3 & \leftarrow c + r_2 \\
r_1 + r_3 & \leftarrow f + (g \times h) \\
4. r_4 & \leftarrow r_2 \times f \\
r_1 + r_4 + (g \times h) & \leftarrow r_5 + r_4 \\
5. r_5 & \leftarrow r_1 + r_4 \\
rs + (g \times h) & \leftarrow r_5 + r_6 \\
6. r_6 & \leftarrow g \times h \\
r & \leftarrow rs + r_6
\end{align*}
\]
With infinite processors (infinite parallelization), we can reduce 7 steps in series to 5 steps in parallel.

c) \((a \cdot b) + (c + d \cdot e) \cdot f + (g \cdot h)\)

\[
\begin{align*}
&\left( + \left( \begin{array}{c} + \left( \begin{array}{c} a \cdot b \end{array} \end{array} \end{array} \right) f + \left( + \left( \begin{array}{c} c \cdot \left( \begin{array}{c} d \cdot e \end{array} \right) \end{array} \end{array} \right) \right) \right) \\
\Rightarrow &\left( + \left( + \left( a \cdot b \right) \left( + c \left( d \cdot e \right) f \right) \right) \right) \left( g \cdot h \right)
\end{align*}
\]
2. Fibonacci numbers are defined as follows: \( \text{fib}(0) = \text{fib}(1) = 1 \), and \( \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \).

2a. Write a Java method \( \text{fib} \) for computing Fibonacci numbers. In the main program, there is a following call to this method:

\[
\text{int } c = \text{fib}(3);
\]

Assuming that the last byte of \( c \) was AED, describe, step-by-step, how memory will be allocated as we run this program.

2b. Write a Lisp function for computing \( \text{fib}(n) \). Show, step by step, how it will compute \( \text{fib}(3) \).

2c. Write a Prolog program for computing \( \text{fib}(n) \). Show how it will compute \( \text{fib}(3) \).

2d. What is the main inefficiency of all these implementations?
2b) \texttt{fib(n)} in \texttt{clisp} trace \texttt{fib(3)}

\begin{verbatim}
_DEFUN fib(n)
 (COND
 ( (= n 0) 1)
 ( (= n 1) 1)
 (T (+ (fib(-n 1))(fib(-n 2)))))
\end{verbatim}

e.g. \texttt{fib(3)}\n\begin{itemize}
\item \[\Rightarrow (+ (fib(\text{-3}1))(fib(\text{-3}2)))]
\item \[\Rightarrow (+ (fib \text{2})(fib 1))]\n\item \[\Rightarrow (+ (+ (fib(-21))(fib(-22))) 1 )\]
\item \[\Rightarrow (+ (+ (fib 1)(fib 0)) 1 )\]
\item \[\Rightarrow (+ ( 1 1) 1)]\n\item \[\Rightarrow (+ 2 1)\]
\item \[\Rightarrow 3\]
\end{itemize}
2c) Prolog Fib(n, X), trace fib(3, X)

\[
\begin{align*}
\text{fib}(0, 1), \\
\text{fib}(1, 1), \\
\text{fib}(n, X) \leftarrow \\
\quad \text{fib}(n-1, X_1), \\
\quad \text{fib}(n-2, X_2), \\
\quad X \text{ is } X_1 + X_2.
\end{align*}
\]

```
trace fib(3, X)
```

```
n=3 \\
\rightarrow \text{fib}(2, X_1) \\
\quad \text{fib}(3-2, X_2) \\
\quad X \text{ is } X_1 + X_2 \\
\rightarrow n=2 \\
\rightarrow \text{fib}(1, X_1') \\
\quad \text{fib}(2-2, X_2') \\
\quad X_1' \text{ is } X_1' + X_2' \\
\rightarrow n=1 \\
\rightarrow \text{fib}(1, 1) \\
\rightarrow X = 3
```

2d) The main inefficiency is that we compute certain values of fib(n) when \( n > 1 \) more than once; we do the same work multiple times.
3. Let us consider the following program for component-wise addition of one array to the other one:

```java
for (int i = 0; i < a.length; i++)
    a[i] += b[i];
```

3a. Use pre- and post-conditions to prove that the program is correct. *Hint:* For each value i, before we go into the i-th loop, we have \( a[j] = a_0[j] + b[j] \) for all \( j < i \) and \( a[j] = a_0[j] \) for all \( j \geq i \), where \( a_0[i] \) are the original values of the array \( a[i] \).

3b. Write down the sequence of quadruples implementing this code.

3c. Explain why in Java, the memory allocated to each data type is a power of two: 1, 2, 4, 8, 16 bytes, etc.

3d. Can we use parallel-for in this program? Explain your answer.

3e. Write a generic Java method for performing the above operation, so that we should be able to use the same method for integers, doubles, etc.
3b) \( \text{for (i=0; i<a.length; i++)} \)
\[ a[i] = b[i]; \]

\( i \leftarrow 0 \)
\( n \leftarrow a.\text{length} \)
\( p \leftarrow < i \quad \text{n} \)

L2: if \( (r1) \) else goto L1
\[ q[i] \leftarrow a[i] \quad b[i] \]
\( i \leftarrow i+1 \)
goto L2

L1: ...

3c) why in Java memory allocated to each data type is a power of 2?

Addresses in memory follow an alignment, and the contents of these addresses are binary numbers. It follows that the most efficient way to store it is in powers of 2, which maintains this alignment without sacrificing much space.

3d) Can we use parallel-for in this program?

Yes; consider the steps:

\[ a[0] = a[0] + b[0] \]

\[ a[a.\text{length}-1] = a[a.\text{length}-1] + b[a.\text{length}-1] \]

We can see that regardless of which order we perform these operations, each value \( a[i] \) will be assigned the correct value.
3e) public <T>[] CWS(<T>[] a, <T>[] b) {
    for (int i = 0; i < a.length; i++) {
        a[i] += b[i];
    }
    return a;
}
4a. Describe Java integers in BNF and EBNF forms.

4b. Give an example of ambiguous and unambiguous BNF forms for Java integers.

4c. Why do we need BNF? Why do we need EBNF?

4d. Describe a state diagram that would enable a computer to recognize Java integers.

\[
\text{BNF} \Rightarrow \begin{align*}
\text{<digit>} & : = 0 | \ldots | 9 \\
\text{<nn>} & : = \text{<digit>} | \text{<digit>} \text{<nn>} \\
\text{<int>} & : = \text{<nn>} | + \text{<nn>} | - \text{<nn>}
\end{align*}
\]

\[
\text{EBNF} \Rightarrow \\
\text{<int>} : = (\text{<digit>} 1+1-\text{<nn>}) \\
\text{<digit>} : = 01 \ldots 19 \\
\text{<nn>} : = \text{<digit>} [\text{<nn>}] \\
\]

4b. ambiguous: \text{<digit>} : = 0 | \ldots | 9 \\
\text{<nn>} : = \text{<digit>} | \text{<nn>} | \text{<nn>}

\text{<int>} : = \text{<nn>} | + \text{<nn>} | - \text{<nn>}

\begin{itemize}
  \item example: 
  \begin{itemize}
    \item +4670
    \item int
    \item + nn
  \end{itemize}

  because \text{<nn>} has a rule

  at \text{<nn>} \text{<nn>}, it's

  ambiguous where the recursion

  would take place

  this is just one

  example, but it still

  doesn't make the recursion

  clear.
\end{itemize}

Same#

Different tracing
Unambiguous:  

\[<\text{digit}> ::= 01 \ldots 9\]
\[<\text{nn}> ::= <\text{digit}>|<\text{digit}><\text{nn}>\]
\[<\text{int}> ::= <\text{nn}>|+<\text{nn}>|-<\text{nn}>\]

The recursion is clear in this BNF:

\[
\begin{align*}
\text{int} & \\
+4670 & \\
+ & \\
n & \\
4670 & \\
\text{digit} & \\
n & \\
6 & \\
\text{digit} & \\
n & \\
0 & \\
\text{digit} & \\
\text{digit} & \\
0 & \\
\end{align*}
\]

BNF gives the compiler a formal description since it doesn't understand natural language, which is why we need it. EBNF makes BNF concise and if the rules are known, it also makes the rules more readable.
5. Beyond Java: for each of the following features, describe its advantages and disadvantages, and give an example:

5a. declarative programming
5b. multiple assignment (as in Perl)
5c. guarded commands as an alternative to if-then-else statements
5d. Algol's call by name
5e. methods as parameters of other methods
5f. reflection
5g. wave algorithm

5a) Declarative Programming

- Advantages:
  - Can be used to find results even if a method to solve it hasn't been developed
- Disadvantages:
  - Is very inefficient

- e.g. Prolog, OBJ3

5b) Multiple Assignment:

- Advantages: allows to assign values to multiple variables in a comprehensive, versatile way.
  - e.g. \((x, y) \leftarrow (y, x)\) is a way to assign the previous value of \(y\) to \(x\) and vice versa; in a single expression

- Disadvantages: it increases the complexity of the language, and the operation is inefficient (takes time)

- e.g. \((x, y) \leftarrow (y, x)\)
5c) Guarded commands as an alternative to **IF-ELSE**
- Advantages:
  - Parallelization becomes possible, and it simplifies the formal analysis of code
- Disadvantages:
  - A set of guarded commands that are not rid of intersections between its conditions may yield incorrect results, as well as if possibly being ambiguous.

5d) Algol's call by name
- Advantages:
  - Makes the notation an synthax of the language closer to mathematical notation, which may simplify certain tasks.
- Disadvantages:
  - It makes compiling take more time, inefficient.

5e) Methods as parameters of other methods
- Advantages:
  - Similar to algol's call by name, being able to use a method as a parameter simplifies implementation.
- Disadvantages:
  - Again, it makes compiling more difficult, and inefficient.
5f) Reflection:
- Advantages:
  - as part of the meta-language, it makes programs
  - robust, being able to handle multiple types of
    parameters that can be handled differently.
- Disadvantages:
  - Makes languages more complex, is often
    inefficient

5g) Wave algorithm
- Advantage
  - A way to avoid looping infinitely on logic
    languages (i.e. Prolog)
- Disadvantages
  - "A shot in the dark", since you are
    approaching the problem using what you
    know to try to find what you're looking
    for, you are not guaranteed results.