HTML and LaTeX are two document formatting languages. We want to build a translator to convert documents structured using HTML to LaTeX. In HTML, the text of the document to be formatted is specified between <html> and </html>. In LaTeX, the text is specified between \begin{document} and \end{document}. In HTML, a list of numbered bullet items is given using <ol> and </ol>; in LaTeX, the list is specified using \begin{enumerate} and \end{enumerate}. Each item in the list begins with \item in HTML, and \item in LaTeX. Font changing commands in HTML are specified as follows: the text to be displayed in bold face, italics, or teletype are marked with <b>, <i>, and <tt>, respectively, in the beginning, and </b>, </i>, and </tt>, respectively, in the end. In LaTeX, the text with changed font begins with \textbf{, \textit{ and \texttt{ respectively, and the change of font persists until the matching \textbf{ is found.}

A sample translation of a HTML document to LaTeX is shown below:

HTML:
```
This is a sample document.
A list of bullet items can be created as follows:
\begin{itemize}
\item Start an \textbf{itemize} environment;
\item Place each item with an \textit{item} command.
\end{itemize}
```

LaTeX:
```
This is a sample document.
A list of bullet items can be created as follows:
\begin{itemize}
\item Start an \textbf{itemize} environment;
\item Place each item with an \textit{item} command.
\end{itemize}
```

a. (2 points) List the smallest set of tools you will use to construct a translator to convert input HTML documents into LaTeX documents. Explain your choice.

b. (3 points) If the input documents may not all be well formed HTML documents, will the tools you choose in part (a) above be still sufficient to construct a translator? Explain.
BigSoft Inc, the world's largest supplier of useless software, announces a new product called antitrust. The program converts documents (which are plain ASCII) as follows: each input sentence is copied to the output, except those sentences that contain the word competition.

BigSoft claims that antitrust is written completely using Lex, with no C code other than printf's.

Is this possible? If so, sketch the Lex specification; if not, explain why not.

The "normal" way to write arithmetic expressions is called infix form, since operators are sandwiched between the operands. For example, in (1 + 2) + 3, `+` operates on 2 and 3, and `/+/' works on 1 and the result of the product.

The same expression can be written in postfix form, as 1 2 3 +. The operators precede the operands.

For instance the infix expression (2 + 4)*6 will be written in postfix as 2 4 + 6 *.

Write the infix expression (1 + 2) + 3 in postfix form.

Can every infix expression be converted to postfix? Justify.

Let L be a language of postfix expressions over integer constants, with `,`, `+`, and `*` as binary operators, `,` as an unary operator, and `(` and `)` as delimiters.

Write a grammar that describes L precisely.
Let $G$ be an LL(1) grammar with $V_T$ as the set of terminal symbols and $V_N$ as the set of non-terminal symbols.

Note: For each of the following, you may write your answer in big-O notation (e.g., $O(n)$).

(a) (4 points) What is the maximum size of LL(1) parsing table for $G$? Why?

(b) (4 points) What is the size of the largest parse tree produced by a grammar in CNF for a string of length $n$? Why?

You may assume that $G$ has no $-$productions, and each production in $G$ is of the form $A \rightarrow B \alpha$ or $A \rightarrow \alpha$ where $\alpha$ is a terminal.

(c) (4 points) What is the maximum size of SLR(1) parsing table for $G$? Why?

(d) (4 points) How long does it take to parse a string of length $n$ with respect to $G$ using the LL(1) parsing algorithm? Why?

(e) (4 points) How long does it take to parse a string of length $n$ with respect to $G$ using the shift/reduce parsing algorithm with an SLR(1) table? Why?
Consider the following grammar, \( G \), that defines expressions over integers and a special identifier \( x \):

\[
\begin{align*}
E &::= E + T \\
E &::= E \times T \\
E &::= (E) \\
T &::= x \\
T &::= T^* F \\
T &::= F \\
F &::= \operatorname{int} F \\
F &::= \operatorname{int} \\
\end{align*}
\]

The expressions generated by \( G \) can be viewed as polynomials of \( x \). Consider the problem of differentiating the polynomials with respect to \( x \). For instance, for an input \( x \), the translation should yield \( 1 \). The output need not be simplified. For example, \( x^* x \) may be translated into \( x^* x + x^* x \).

Let \( dx \) be an attribute of non-terminal symbols \( E \), \( T \), and \( F \) that holds the derivative \( d \) of the corresponding symbol. For each of the following non-terminal symbols state whether \( dx \) is an inherited or synthesized attribute:

- \( E \):
- \( T \):
- \( F \):

Write a syntax directed definition to compute the value of attribute \( dx \) for grammar \( G \). Hint: Use the two laws of differentiation to compute the value of derivative of polynomials.

\[ \frac{d}{dx} (a + b + n) = \frac{d}{dx} a + \frac{d}{dx} b + \frac{d}{dx} n = \frac{d}{dx} a \]
Consider the following Decaf program:

```java
class A {
    public int i;
    public int m(int i) { return i; }
}
class B extends A {
    public float m;
    public float i(float x) { return x; }
}
class C extends B {
    public float m(float x) { return m; }
    public int i(int i) { return i(i); }
}
class D extends C {
    float i;
    int m;
    void f() {
        boolean i;
        A a; B b; C c; D d;
        a = new A();
        b = new B();
        c = new C();
        d = new D();
        Out.print(EXAMPLE/*/*/*/*);
    }
}
```

The table below lists expressions that may appear in the program in place of EXAMPLE/EXPRESSION (marked with /*/*/*/*/*). In the space provided in the table, fill at least 8 entries. Each correct entry in the table is worth 2 points (maximum 16 points).

<table>
<thead>
<tr>
<th>EXAMPLE/EXPRESSION</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.i</td>
<td>int</td>
</tr>
<tr>
<td>d.i</td>
<td>int</td>
</tr>
<tr>
<td>i</td>
<td>int</td>
</tr>
<tr>
<td>this.i</td>
<td>int</td>
</tr>
<tr>
<td>i(m)</td>
<td>float</td>
</tr>
<tr>
<td>m(i)</td>
<td>int</td>
</tr>
<tr>
<td>m(i+m)</td>
<td>float</td>
</tr>
<tr>
<td>c.m(i)</td>
<td>float</td>
</tr>
<tr>
<td>c.m(b.i)</td>
<td>float</td>
</tr>
<tr>
<td>c.m(this.i)</td>
<td>float</td>
</tr>
</tbody>
</table>

Give a syntax directed definition to generate code for the Java statement (C or Decaf).

Give a linker definition to generate code for the Java expression (C or Decaf).
Consider the statement:

\textit{continue} semantics of \textit{continue} is same as in C or Decaf, \textit{i.e.}, begin the next iteration of the loop.

Give a syntax directed definition to generate code for the \textit{continue} statement. List any modification(s) to the definition in part \textbf{a} (previous page) needed to support the \textit{continue} statement.

\textbf{c. (5 points)} Consider the extension of \textit{continue} statement of the form \textit{continue} \( E \), which means: begin next iteration of loop if \( E \) is true.

Sketch how you will modify the syntax directed definition of part \textbf{a} to handle this extension. Give any modification(s) needed to support \( E \).