Problem #1 (10 points; 1 point each)
Circle T or F to indicate the true or false statements.

T F The following grammar is ambiguous.
S-->SaS
S-->SbS
S-->c

T F It is possible to build a deterministic finite state automation for the
language \{ a* b* \} with only two states.

T F We use a scanner to convert characters to tokens rather than having the
parser do the conversion because it simplifies the parser and leads to
smaller, more time efficient compilers.

T F A sentential form is a string of terminals and non-terminals that can be
derived from the distinguished start symbol.

T F In OO93, two routines can have the same name or identifier.

T F In constructing a compiler, it is a good idea to use a program, like flex
or lex, because it simplifies the coding of a scanner.

T F A shared library contains code that can be shared by several processes
running different programs.

T F Dynamically allocated space is placed in the heap.

T F All objects in OO93 contain a pointer to an object that represents the
class of the object.

T F A grammar is a representation for a possibly infinite set of sentences.

Problem #2 (15 points)

(a) (5 points) Define a right-most derivation.

(b) (5 points) A possible definition for a floating point BINARY number input
format is the following: an optional + or - sign followed by a string of zero
or more 0's or 1's followed by a decimal point followed by a string of one or
more 0's or 1's.

Some valid numbers in this format are: +.1 10.01 -101.0

Some invalid numbers are: 1. -110

Show a finite state automation that will recognize BINARY floating point numbers.

(c) (5 points) Explain a reduce/reduce conflict and give an example grammar with one.

Problem #3 (25 points)
Answer the questions below base on the following program fragment:

```plaintext
uses "stdobj.ooh"
List: class object is
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp item: object;
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp next: List;
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp end;
Student: class object is
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp name: string;
    &nbsp&nbsp&nbspnbsp&nbsp&nbsp&nbsp&nbsp login: string;
end;
WorkStudy: class Student is
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp hours: int;
end;
BandMember: class Student is
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp instrument: string;
end;
Employee: class object is
    &nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp&nbsp name: string;
    &nbsp&nbsp&nbspnbsp&nbsp&nbsp&nbsp&nbsp jobtitle: string;
    &nbsp&nbsp&nbspnbsp&nbsp&nbsp&nbsp&nbsp hours: int;
end;
```

(a) (5 points) What is the type of "new(BandMember)"?

(b) What is the type of "new(Employee.classof)?"

(c) (10 points) Suppose you wanted to write a generic procedure named "works" that returned true if the object passed was a work study student or an employee and otherwise it returned false. Write the methods required to implement this procedure. Note that the solution should be modular so that if we add a new category of student who works, we don't have to modify your existing definitions.
What does the following procedure do?

```plaintext
foo: procedure (x: List, y: object) : int is
begin
    if (x nil) then
        if (x.item.classof = y) then
            return (1 + foo(x.next, y));
        else
            return (foo(x.next, y));
        fi;
    fi;
    return (0);
end
```

(Hint: an example call is "foo(a_var, BandMember)" where a_var is a variable of type List.)

Problem #4 (30 points)
Given the following LR parser tables and grammar rules, answer the following question.

(a) (10 points) Show the parse tree for the sentence ") ) id id ( ( (.

(b) (20 points) Show the parser configuration as it parses that input in the following table. You must use state numbers on the syntax stack. (Hint: 25 configurations are shown in the table -- the parse may take less than, more than, or equal to that number steps.)
Problem #5 (20 points) Given the following grammar, construct the finite state automation that represents the sets of collections of LR(0) items and the transitions between the sets.

\[
\begin{align*}
S' & \rightarrow S$
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow A 'a'
\end{align*}
\]

\[
\begin{align*}
S & \rightarrow A 'a' S
\end{align*}
\]

\[
\begin{align*}
A & \rightarrow 'b' 'c'
\end{align*}
\]

\[
\begin{align*}
A & \rightarrow 'b' A 'c'
\end{align*}
\]

Notice that the rule for S' has already been added to the grammar.