CS 450: Exam 2

April 19, 2002

Name: _____ **E-mail:** _____

This is a 50 minute test, and the time limit will be strictly enforced. You may use two 8.5"×11" pages of notes. Show all of your work to make it possible to give partial credit.

There are 5 questions on this test, on 8 numbered pages. It is your responsibility to make sure that you have all of the questions.

1. Use the grammar below for all parts of this queston.

1.	<system_goal>	$\longrightarrow$	<statements> **eof**
2.	<statements>	$\longrightarrow$	<statement> <statements>
3.	<statements>	$\longrightarrow$	λ
4.	<statement>	$\longrightarrow$	**id := id** <tail>
5.	<tail>	$\longrightarrow$	**+ id**
6.	<tail>	$\longrightarrow$	**- id**
7.	<tail>	$\longrightarrow$	*** id**
8.	<tail>	$\longrightarrow$	**/ id**
9.	<tail>	$\longrightarrow$	λ

(a) Describe in English the language generated by this grammar.

(b) What is the shortest valid string in the language represented by this grammar?

(c) Give the row labeled <tail> in the LL(1) table for this grammar.

(d) In rule 7, place any necessary semantic actions for generating code. List the semantic records needed in the call(s) and explain what they hold and how they got their values.

(e) Assuming that you are translating to an IR that uses a stack for arithmetic similar to your project, give (in general terms) the code that would be generated by the semantic action call(s) you inserted in answer to the previous part of this question.

2. Consider the following program in answering all parts of this question.

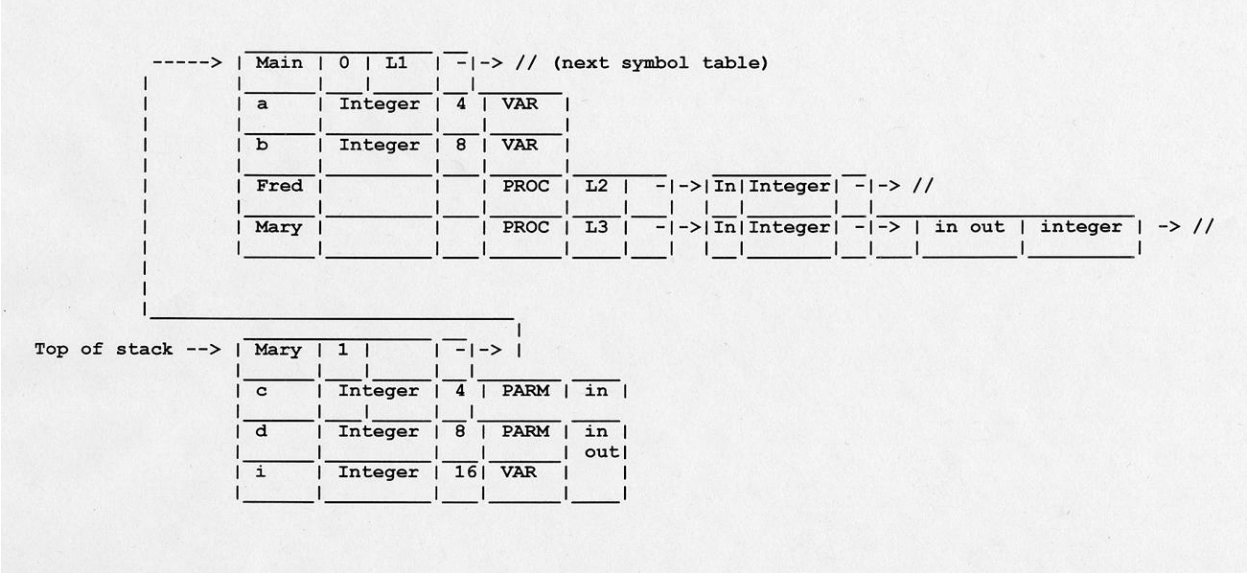
```
program Fuss;

  var a, b: integer;
      x:    float;
      name: character;

  begin --Fuss
    read(a, x);
    write(x);
    b := 3 + a * a;
  end. --Fuss
```

(a) Diagram the symbol table as it would appear at the point when the compiler has just encountered the `begin` statement in this program. Be sure to include all of the entries that we have been presenting in class recently for symbol tables. You can assume for this example that integer variables require four bytes, float variables require eight bytes, and character variables require one byte. Also for this example, variables are assumed to start at byte offset 4 in the activation record.

(b) Using your symbol table and stack based arithmetic, give the translation of the program statements inside the `begin` block of Fuss. Do not include the code for creating and removing the activation record for Fuss. Just translate the three instructions into IR that is similar to the IR for the virtual machine of your project.

(c) Suppose that this entire program has been translated and that the translated program is now executing on a computer. Diagram the activation record for Fuss the way it would appear at the point in the translated code right after the point where x is written (right after the translation of write(x)). You can assume that the first two values typed at the keyboard are 10 and 3.14159. For elements of the activation record whose values you do not know, just label those elements with a description of what they should contain at this point. Follow the general model given in class.

3. Consider the following symbol table structure in answering the various parts of this question.



Assume that you are generating IR code in the context of this symbol table structure. Translate the following statements using an IR similar to the one for your project. If the statement cannot be translated, explain why. If parameters are involved, follow the models given in the past few lectures for modes **in** and **in out**. Also, where necessary, follow the model given in class for construction of activation records.

(a) `Write(i);`

(b) `c := i * 3 + i;`

(c) `a := c + b * i;`

(d) `d := c + i;`

(e) `Fred(i);`

4. Use the program below for the following questions.

```
program Main
    a, b: integer;

    procedure Fred(x: in integer);
        b, c: real;

        function Ellen(a: in integer);
            x, y: real
                . . .
            end Ellen

        procedure Jack(z: in out integer)
            a, b: real;
            name: character;
                . . .
            end Jack

                . . .
        end Fred

                . . .
    end Main
```

(a) Look for the line with three dots (. . .) in Ellen. Show a diagram of the symbol table stack as it would appear at that point in the compiler. **Note: The** *only* **thing you are to show for an individual symbol table is its** *name* **(e.g., Main or Fred). Do** *not* **include any entries in the symbol table. Instead the purpose is to illustrate the stack structure of the individual symbol tables at various points in the program.**

(b) Look for the line with three dots (. . .) in Jack. Show a diagram of the symbol table stack as it would appear at that point in the compiler. **Note: The** *only* **thing you are to show for an individual symbol table is its** *name* **(e.g., Main or Fred). Do** *not* **include any entries in the symbol table. Instead the purpose is to illustrate the stack structure of the individual symbol tables at various points in the program.**

(c) Look for the line with three dots (. . .) in Fred. Show a diagram of the symbol table stack as it would appear at that point in the compiler. **Note: The** *only* **thing you are to show for an individual symbol table is its** *name* **(e.g., Main or Fred). Do** *not* **include any entries in the symbol table. Instead the purpose is to illustrate the stack structure of the individual symbol tables at various points in the program.**

(d) Suppose that the compiled program is now executing, and that Main calls Fred, then Fred calls Jack, then Jack calls Ellen, then Ellen returns to Jack. Diagram the activation records on the run time stack at this point in time, using the format given in class. Allow four bytes for each entry in the activation record. Label each activation record with the name of the routine it is for, and label each element of each activation record with what it contains (in general terms, not values). Start the run time stack at address 100.

5. (**Mind teaser: graded separately as extra credit. Attempt only if you are satisfied with your previous answers.**)

   Consider the classes and objects of an object oriented programming language such as Java or C++. Consider how you would go about compiling objects. Remember that objects declared to be instances of a class all have the same methods (procedures and functions) but not the same attributes (be careful to not confuse the word *attributes* as it applies to objects with how it applies to symbol table entries in a compiler).

   In the space below, describe in general terms how you would compile the following with respect to the symbol table and to code generation.

   (a) a class

   (b) an object declared as an instance of a class

   (c) a call to a method of an object

   You do not need to consider special things, like static methods.