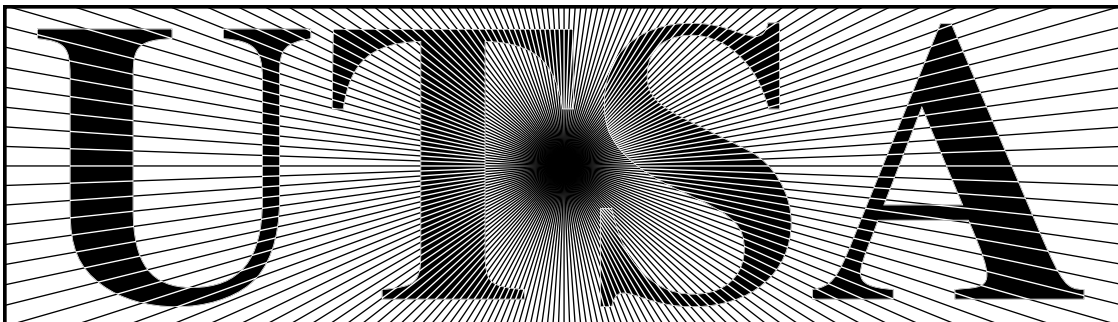


CS 3723, Programming Languages
Spring Semester, 2002
Final Examination

1. This is a question about *garbage collection*.
 - (a) What is meant by garbage collection? (Short answer.)
 - (b) Which of the following languages normally support garbage collection: C, C++, Java.
 - (c) What do the other languages do instead of garbage collection?
 - (d) What is being collected? (Needs more than answer “garbage.”)

2. Consider the following Postscript program:

```
%!PS-Adobe-2.0
/Times-Bold findfont 150 scalefont setfont
/path {
  newpath 0 0 moveto (UTSA) true charpath
} def
/online {
  0 0 moveto 250 0 lineto stroke
} def
/lines {
  0.5 setlinewidth
  gsave (UTSA) stringwidth pop 2 div 50 translate
  0 1 179 {pop 2 rotate online} for
  grestore
} def
/border {
  newpath
  -10 -10 moveto 0 120 rlineto
  (UTSA) stringwidth pop 20 add 0 rlineto
  0 -120 rlineto closepath
} def
/utsa {
  border clip lines
  border 2 setlinewidth stroke
  path fill
  path clip 1 setgray lines
} def
15 15 translate utsa
showpage
```



- (a) Do you have to do anything special to get a Postscript printer to print the figure below when it is given this code?
- (b) Explain in detail how the code above is producing the given picture. In particular
- What part is drawing the box around the outside?
 - How does the code arrange for the circle of lines to come from a point with x-axis in the middle of the UTSA?
 - What keeps the circle of lines from going outside the outer rectangle?
 - Describe the “for” loop inside “lines” in detail.
3. The language represented by the grammar below consists of fully parenthesized postfix expressions:

```

P ----> E
E ----> dig | '(' E E Op ')'
Op ----> '+' | '-' | '*' | '/' | '^'
dig ----> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

```

Here is a Java parser for the language:

```

// Parser.java: parser for parenthesized postfix language
public class Parser {
    private char next; // = ' ';

    // P: recognize a program
    public void P() {
        E();
    }

    // E: recognize a statement
    private void E() {
        scan();
        if (Character.isDigit(next)) {
            // next is a single digit here
        }
        else if (next == '(') {
            E();
            E();
            scan();
            scan(); // for the ')'
            if (next != ')') error(1);
        }
        else error(3);
    }

    // scan: put the next non whitespace char into next
    private void scan() {
        do {
            try {
                next = (char)System.in.read();
            } catch (Exception e){
                error(2);
            }
        } while ( next == ' ' || next == '\n');
    }
}

```

```

// error: error encountered during the parse
private void error(int n) {
    System.out.println("ERROR " + n + ", Token: " + next );
    System.exit(1);
}

public static void main(String[] args) {
    Parser parser = new Parser();
    parser.P();
}
}

```

You are to add code to the listing above so that the program will return the correct integer values when the operations have been carried out. Here is an interactive session showing how the language works and showing what your code should do. Each input is a single expression, either just a constant or a parenthesized expression starting with two similar expressions and ending with an operator.

```

% java Parser
6
6
% java Parser
( (2 3 +) (3 4 *) -)
-7
% java Parser
((2 4 ^) ( (8 2 /) (5 3 -) *) +)
24

```

- Evaluate the expression $((2\ 4\ \wedge)\ ((8\ 2\ /)\ (5\ 3\ -)\ *)\ +)$ and show that 24 is the correct answer.
- Supply the additional code needed to do the evaluation. If you wish, you may use the evaluating function below. (If you don't know any java, you can still do this problem by just pretending that you are coding in C, since for this application the two languages are nearly identical.)

```

private int eval(int r1, int r2, char op) {
    switch (op) {
        case '+': return r1 + r2;
        case '-': return r1 - r2;
        case '*': return r1 * r2;
        case '/': return r1 / r2;
        case '^': int res = r1;
                    while (r2 != 1) {
                        res = res*r1; r2--;
                    }
                    return res;
        default:
            error(4);
            return 0;
    }
}

```

- Using the reference sheet about the Quadruple Machine to help you, translate the following simple program *by hand* into quadruples:

```

n = 1;
WHILE (10 - n) {
    WRITE(n);
    WRITEC(32); /* a blank */
    n = n + 1;
}
WRITEC(10); /* a newline */

```

(If you wish, you may use the symbolic names for the quadruples you need, rather than the integer codes.)

5. Consider the grammar rule for a WHILE statement:

```
whilestmt ----> "WHILE" "(" expr ")" stmt
```

- Give the code for the function **whilestmt** in your parser. (Just give this code in rough form, using whatever kind of helpful pseudocode you need. You can employ the notation that you actually used in your own parser if you wish.)
- Show in outline form how to translate a general **WHILE** statement into quadruples, as you did (may have done) for the project for this course. This amounts to additions to the code for the **whilestmts** function in your parser. You should focus particularly on how you keep track of the numbers of key quadruples, and how you carry out the back-patching. You can assume a function **nextQuad()** that gives the quadruple number of the next quadruple to be generated. (Again, you can use any convenient pseudocode, and helpful additional functions, as needed.)

6. Consider the formal Grammar

```

P ----> E '$'
E ----> E '+' T | E '-' T | T
T ----> T '*' S | T '/' S | S
S ----> F '^' S
F ----> 'a' | 'b' | 'c' | '(' E ')'

```

and the sentence $a \wedge b * (c + b) \$$.

- Which are the terminal symbols and which are the non-terminals?
- Is this grammar ambiguous? (Just “Yes” or “No”.)
- Construct a parse tree for the given sentence.

7. Consider the following Prolog program:

```

reigns(henry7, 1485, 1509).
reigns(henry8, 1509, 1547).
reigns(edward4, 1547, 1553).
reigns(jane_grey, 1553, 1553).
reigns(mary1, 1553, 1558).
reigns(elizabeth1, 1558, 1603).
reigns(james1, 1603, 1625).

```

```

reigns(charles1, 1625, 1642).
reigns(oliver_cromwell, 1642, 1658).
reigns(richard_cromwell, 1658, 1659).
reigns(charles2, 1660, 1685).
reigns(james2, 1685, 1688).
reigns(mary2, 1688, 1694).
reigns(william3, 1688, 1702).
reigns(anne, 1702, 1714).
reigns(george1, 1714, 1724).
reigns(george2, 1727, 1760).
reigns(george3, 1760, 1811).
reigns(george4, 1811, 1830).
reigns(william4, 1830, 1837).
reigns(victoria, 1837, 1901).
reigns(edward7, 1901, 1910).
reigns(george5, 1910, 1936).
reigns(edward8, 1936, 1936).
reigns(george6, 1936, 1952).
reigns(elizabeth2, 1952, 2001).
house_of(henry7, tudor).
house_of(henry8, tudor).
house_of(edward4, tudor).
house_of(jane_grey, tudor).
house_of(mary1, tudor).
house_of(elizabeth1, tudor).
house_of(james1, stuart).
house_of(charles1, stuart).
house_of(oliver_cromwell, commonwealth).
house_of(richard_cromwell, commonwealth).
house_of(charles2, stuart).
house_of(james2, stuart).
house_of(mary2, orange).
house_of(william3, orange).
house_of(anne, stuart).
house_of(george1, brunswick).
house_of(george2, brunswick).
house_of(george3, brunswick).
house_of(george4, brunswick).
house_of(william4, brunswick).
house_of(victoria, brunswick).
house_of(edward7, saxe_coburg_gotha).
house_of(george5, windsor).
house_of(edward8, windsor).
house_of(george6, windsor).
house_of(elizabeth2, windsor).
ruler(N, Y) :- reigns(N, A, B), Y >= A, Y =< B.

```

Try to explain this as above.[Hint: In Prolog, identifiers starting with an uppercase letter are “unknowns”, while identifiers starting with a lowercase letter are “constants”.]

- Which are the facts above and which are the rules?
- Look at the following session (boldface and all semicolons are user input — a semicolon just means to try again):

```

ten42% pl
?- consult(rulers).
rulers compiled, 0.00 sec, 5,164 bytes.
Yes

```

```
?- ruler(X, 1560).
X = elizabeth1 ;
No
?- ruler(X, 1688).
X = james2 ;
X = mary2 ;
X = william3 ;
No
?- house_of(X, stuart).
X = james1 ;
X = charles1 ;
X = charles2 ;
X = james2 ;
X = anne ;
No
```

- (c) Explain why there are the three answers to **ruler(X, 1688)**.
- (d) Write a rule **years(H, A, B)** where **H** is a House and **A** and **B** are the years of someone's reign who belongs to the house **H**.
- (e) With the new rule, what would be the result of typing **years(orange, A, B)** .?