The Text has a few problems on grammars: Ex 4-5, 4-6, 4-7 on page 164, Ex 4-8, 4-9 on page 167 and problems 3, 4, 5?, 6, 7? on pages 173-174.

Some additional problems:

I. Given A ::= a|b|c
B ::= A|(C)
C ::= B|{B,C}
(Here a, b, and c are terminals, A, B, and C are non-terminals)
For each of the strings below, indicate all syntactic categories of which it is a member (i.e. A, B and/or C), if any. Give a derivation and draw derivation trees.

(i) c (ii) (a) (iii) {b}
(iv) ({a,b}) (v) {(a),b} (vi) {(a),{b,a}}

II. Write a BNF grammar for the language composed of all binary numbers which contain at least 3 consecutive 1's. (includes strings like 011110101 and 000011101010 but not 0100110101011)

III. Given T ::= W|T_W
W ::= Y|YWY
Y ::= P|PS|aP|aS
P ::= Sa
S ::= b|d
(Here T is the start symbol, a, b, d and _ are terminals and the capital letters are non-terminals)

Which of the following strings can be derived from the rules above.

(i) ba_ababadada_bad_dabbada
(ii) abdabaadab_ada
(iii) dad_ad_abaadad_badadbaad

be sure to try them first.

4-5 & 4-6 hint derive them from Figure 4.1 on page 165.
4-7 p164: One solution is:
<identifier> ::= L|L_T
T ::= A|A_T
L ::= <letter>A starts with a letter than letters $/or digits
A ::= <letter>A|<digit>A non-zero strings of letters &/or digits
4-9 p167: One solution is:
<identifier> ::= <letter><alphanumeric>*{_<alphanumeric>+}*
superscripts ^ ^ ^
4-8 p167: These are so simple they are hard. But here is what is wanted.
A ::= B+ is defined to be the same as A ::= B|AB
A ::= [+ on top of -]B is defined to be A ::= B|+B|-B
A ::= {B on top of C} is defined to be A ::= B|C
Simple substition will get you from Figure 4.1 to Figure 4.2 now.

3 p173: <legal subscript> ::= C|V|C+V|C-V|C*V|C*V+C|C*V-C
C integer constant, V an interger variable.
4 p173: hint see pages 147-148
6 p174: <compound statement> ::= begin <statement list> end
<statement list> ::= <statement>|<statement>;<statement list>
is BNF
<compound statement> ::= begin <statement>{;<statement>}* end
is extended BNF superscript ^
The last notation is of your choice perhaps
<compound statement> ::= begin <statement>[;<statement>]... end

answers to the additional problems
I. since C->B->A the possible answers for each string is either
None, C only, B and C only or all three of A, B and C.

(derivation trees are not fun to draw in ascii files, so i didn't)
(i) All three: C->B->A->c.
(ii) Only B and C: C->B->(C)->(B)->(A)->(a). Note only a, b or c is derivable from A.
(iii) None: every rule with a curely brace also has a comma, so every string in the language has # of '{' = # of ',' = # of '}' and {b} doesn't.
(iv) Only B and C: C->B->(C)->({B,C})->({B,B})->({A,A})->({a,b}).
(v) C Only: C->{B,C}->{(C),C}->{(B),B}->{(A),A}->{(a),b}. Note only strings which start and stop with (and) are derivable from B other than a, b & c.
(vi) C Only: C->{B,C}->{(C),{B,C}}->{(B),{B,B}}->{(A),{A,A}}->{(a),{b,a}}.

II. Let S be the start symbol and T be another non-terminal then one solution is S ::= T|0S|1S|S0|S1
T ::= 111

III. Because the only rule with the _ is T->T_W, each of the given strings is derivable from T if all the substrings between the '_' are derivable from W.
Second since the only way to obtain b or d is via S, then we can replace each b or d with S in all these substrings and pretend S is a terminal.
Third, we "expand" P out to eliminate it. (Since P->Sa is the only derivation possible from P.)

Thus the problem has become given W is the start symbol, Y non-terminal, and S and a are terminals with the rules:
W ::= Y|YWY
Y ::= Sa|SaS|aSa|aS
which of the following strings are derivable:
Sa aSaSaSaSa SaS SaSSaSa
aSSaSaaSaS aSa
SaS aS aSaaSaS SaSaSSaaS

Clearly the short strings: Sa SaS aSa and aS are derivable. Also
W->YWY->YYY->aSaYY->aSaSaY->aSaSaSa.
W->YWY->YYY->SaSYY->SaSSaY->SaSSaSa.
W->YWY->YYY->aSaYY->aSaaSY->aSaaSaS.
There are two more, for the long one:
W->YWY->YYWYY->YYYYY->aSYYYY->aSSaYYY->aSSaSaYY->aSSaSaaSY->aSSaSaaSaS.
which leaves only SaSaSSaaS which can't be derived. (Note that all strings must be derived from an odd number of Y's. From one Y we can get strings of length two or three. From YYY we can get strings of length six to nine. From YYYYY we get strings of length ten to fifteen.) Since SaSaSSaaS has length 9, if it were derivable, it would be derivable from YYY and SaS, aSS and aaS would have to be each derivable from Y. Neither aSS nor aaS is derivable from Y.

going back to the question which was asked, only the last one is not derivable (it was dad_ad_abaadad_badsdbaad)

under construction
due mon last week of classes. 4 dec 89

Start with the stuff in ~bellenot/lisp-part3
you will need "makefile" and "main.c"
note the new test-data files:
opTest, opTest2, propTest, oldTest, facTest
mapTest, setTest

NOTE that there are changes in the include files and
many of the other files.

things for you to do:

0. Improve your stuff from part2 as required to:
   a. make C-functions like isNull isAtom isEq return C-booleans
      actually calls to isNull(x) should be replaced by x==nil and
      similarly for isEq. isAtom should exist.
   b. remove the tail recursion from evlis evcon pairlis and assoc.
1. Change main.c so to include the functions in evaluate.c which
   you did for part2 of the project. You will need to do a few
   changes to reflex the change of "Token" from a structure to
   a C++ class. This part should be little or nothing unless you used
   a Token to make new numerical atoms (now use newNumNode in symtable.c)
2. In main.c there are "poorly implemented" calls to 4 new functions
   (which are coded in opsys.c) add these to your lisp interpreter:
   (shell) which "calls" evShell
   (load filename) which reads filename as input calls
      evLoad (filename)
   (edit filename) which calls evEdit (filename)
      The editor which is called is determine by your
      environment. in your .login file there is most
      likely a line "setenv EDITOR emacs" which determines
      which editor which will be called. If you don't have
      EDITOR defined you will get vi ":q" to quit or "ZZ"
   (ledit filename) which calls evLedit (filename) which
      edits and then loads the file.
   HINTS: strings don't work as filenames do you have ``"foo"''
   as a filename? These go in apply so 'filename or (quote filename)
   is needed.
3. (if you haven't yet done so) plus, diff, times, quotient, and mod
   NOTE no remainder, minus replaced with diff!
   This requires numbers and strings to be self-evaluating
   (plus 1 2 3 ...) = 1 + 2 + 3 + ... or 0 if no parameters
   (diff 1 2 3 ...) = 1 - 2 - 3 - ... or 0 if no parameters
   (times 1 2 3 ...) = 1 times 2 times 3 times ... or 1 if no parameters
   (quotient 1 2 3 ...) = 1 / 2 / 3 / ... or 1 if no parameters
   (mod x y) is "C" x%y (two parameters only)
4. set and setq functions
   (set x y) adds (x . y) to alist called environment and returns y.
   (setq x y) is the same as (set 'x y), the parameter x is not evaluated
5. property lists (see ch 9 in text)
   (a b c d e f) property a has value b, c has value d and e has value f

   (plist atom) returns the property list for atom (stored in pointer
      field "plist". see cell.h)
   (putprop atom exp1 exp2) adds exp1 as value to exp2 for atom
      returns exp1. (similar to the putprop of text)
   (get atom exp) return the value of property of exp for atom (similar
      to the getprop of text)
   (remprop atom exp) remove exp and its value from property list
      of atom. returns the plist after exp (see propTest file)
6. garbage collection
   the garbage collection routine in main.c will need updating. Set
   makes the global variable environment non-nil between calls.
7. making the code robust--hints and helps to be added.
   a. Make the interpreter functions check for NULL parameters first
      and if one is NULL, return NULL.
   b. Make eval check if assoc returns nil. If so it should print the
      error message "Eval: undefined symbol %s" and then return NULL
   c. Pairlis should check that the number of parameters match and
      printout error messages like too few or too many parameters
      And when such errors occur it should return NULL.
   d. To make everyone's lisp the same (car nil) and (cdr nil) will
      be ERRORS! (note this is different from the lisp on the vax)
      Indeed, (car x) and (cdr x) are now errors whenever x is an
      atom, and should print out error messages like "bad argument
      to car" (or cdr). Unlike isNull above, having functions like
      SExp * Car (SExp * x) can make the code more robust (because
      they can check for NULL and return NULL safely.)
8. "mapcar" (see text page 383 or run file "mapTest" via lisp < mapTest)

Should have:
(defprop atom exp1 exp2) like putprop but defprop does not
   evaluate its parameters
add "def"
   (def x y) adds (x . y) to alist called environment but evaluates
   neither x nor y.

ADA

```
with text_io; use text_io;
with integer_io; use integer_io;

package wraper is

        task counter is
                entry increment ( new_value : out integer );
                entry report_n_clear ( old_value : out integer );
        end;

        task wire is end;

        task recorder is end;

end wraper;


package body wraper is

task body counter is
        count_value :integer := 0;
begin
        loop
                select
                        accept increment ( new_value : out integer ) do
                                count_value := count_value + 1;
                                new_value := count_value;
                        end;
                        put_line ( "counter.increment" );
                or
                        accept report_n_clear ( old_value : out integer ) do
                                old_value := count_value;
                                count_value := 0;
                        end;
                        put_line ( "counter.report_n_clear" );
                end select;
        end loop;
end counter;

task body wire is
i : integer := 0;
value : integer;
begin
        loop
                delay 1.0; -- delay one second
                counter.increment ( value );
                put ( value ); put_line ( " wire" );
                i := i + 1;
                if i >= 100 then exit; end if;
        end loop;
end wire;

task body recorder is
i : integer := 0;
value : integer;
begin
        loop
                delay 10.0; -- delay ten seconds
                counter.report_n_clear ( value );
                put ( value ); put_line ( " recorder" );
                i := i + 1;
                if i >= 10 then exit; end if;
        end loop;
end recorder;

end wraper;
```

OUTPUT

```
counter.increment
          1 wire
counter.increment
          2 wire
counter.increment
          3 wire
counter.increment
          4 wire
counter.increment
          5 wire
counter.increment
          6 wire
counter.increment
          7 wire
counter.increment
          8 wire
counter.increment
          9 wire
counter.report_n_clear
counter.increment
          9 recorder
          1 wire
counter.increment
          2 wire
counter.increment
          3 wire
counter.increment
          4 wire
counter.increment
          5 wire
counter.increment
          6 wire
counter.increment
          7 wire
counter.increment
          8 wire
counter.increment
          9 wire
counter.increment
         10 wire
counter.report_n_clear
counter.increment
         10 recorder
          1 wire
counter.increment
          2 wire
counter.increment
          3 wire
counter.increment
          4 wire
counter.increment
          5 wire
counter.increment
          6 wire
counter.increment
          7 wire
counter.increment
          8 wire
counter.increment
          9 wire
counter.increment
         10 wire
counter.report_n_clear
counter.increment
         10 recorder
          1 wire
```

```
fsucs 1> cat cnc.c
char * Q = "\"", * B = "\\", * NL = "\n";
char * S0 = "char * Q = %s%s%s%s, * B = %s%s%s%s, * NL = %s%sn%s;%s";
char * S1 = "char * S0 = %s%s%s;%schar * S1 = %s%s%s;%schar * S2 = %s%s%s;%s";
char * S2 = "char * S3 = %s%s%s;%schar * S4 = %s%s%s;%schar * S5 = %s%s%s;%s";
char * S3 = "%s%smain()%s{%s    printf ( S0, Q, B, Q, Q, Q, B, B, Q, Q, B, Q, NL );%s   printf ( S1, Q, S0, Q, NL, Q, S1, Q, NL, Q, S2, Q, NL );%s";
char * S4 = "   printf ( S2, Q, S3, Q, NL, Q, S4, Q, NL, Q, S5, Q, NL );%s";
char * S5 = "   printf ( S3, NL, NL, NL, NL, NL, NL ); printf ( S4, NL, NL );%s printf ( S5, NL, NL, NL );%s}%s";


main()
{
        printf ( S0, Q, B, Q, Q, Q, B, B, Q, Q, B, Q, NL );
        printf ( S1, Q, S0, Q, NL, Q, S1, Q, NL, Q, S2, Q, NL );
        printf ( S2, Q, S3, Q, NL, Q, S4, Q, NL, Q, S5, Q, NL );
        printf ( S3, NL, NL, NL, NL, NL, NL ); printf ( S4, NL, NL );
        printf ( S5, NL, NL, NL );
}
fsucs 2> g++ cnc.c
In function int main ():
cnc.c:12: warning: implicit declaration of function 'printf'
fsucs 3> a.out > cnc.out
fsucs 4> diff cnc.c cnc.out
fsucs 5># no differences!!
fsucs 6> cat self-fun.lisp
(def srf (lambda ()
 ((lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed)))))
  (quote (lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed))))))))
))
(list (getd 'srf))
(srf)
(equal (list (getd 'srf)) (srf))
fsucs 7> lisp <self-fun.lisp
Franz Lisp, Opus 38.79
-> srf
-> ((lambda nil ((lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed))))) '(lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed)))))))
-> ((lambda nil ((lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed))))) '(lambda (seed) (list (list 'lambda 'nil (list seed (list 'quote seed)))))))
-> t
->
fsucs 8> # files in ~bellenot
```

C++

```
1
2
3  struct  Name
4  {
5     char*     n;
6     int       x,y,z;
7
8           Name(char*,int  xx=-1;  int  yy=-1;  int  zz=-1);
9           Name(Name&);
10          operator=(Name&);
11    void    ~Name();
12 };
13
14 extern void takeName ( Name );
15 main()
16 {
17    Name a( "Foo" );
18    Name b( "Foo", 23, 45, 12 );
19    Name* c = new Name ( "Foo", 23, 33 );
20    Name* d = new Name ( "Foo", 1);
21    takeName(a);
22    c = d;
23    delete  c;
24    delete  d;
25 };
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

C

```
42 struct  Name
43 {
44    char*                n;
45    int                  x,y,z;
46 };
47
48 void  setName(Name*,  char*);
49 void  setName1(Name*,  char*,  int);
50 void  setName2(Name*,  char*,  int,  int);
51 void  setName3(Name*,  char*,  int,  int);
52 void  copy(Name*,Name*);
53
54 extern void takeName ( Name );
55 main()
56 {
57    Name a;
58    Name b;
59    setName ( &a, "Foo" );
60    setName3 ( &b, "Foo", 23, 45, 12);
61    Name* c = malloc ( sizeof (Name) );
62    setName2( c, "Foo", 23, 33 );
63    Name* d = malloc ( sizeof (Name) );
64    setName1(d, "Foo", 1);
65    Name* temp = malloc ( sizeof(Name));
66    copy(temp,a);
67    takeName(a*);
68    delete c->n;
69    (c*) = (d*);
70    free(a.n);
71    free(b.n);
72    free(c->n);
73    free(d->n);
74 }
```