Answers to the exercises for chapter: *yacc*

1. First we observe that through the inclusion of C code, all these languages, whether regular, context-free, context-sensitive, can be parsed in *lex*. This means that *lex* and *yacc* are theoretically not restricted to regular and context-free languages, even though their basic mechanism is a FSA and PDA respectively.
   Let us now look at solutions that use both *lex* and *yacc*.
   For these languages, the *lex* program can be very simple:

```
%{

#include "anbn.h"

%}

%%

[ab] {return *yytext;}
\n   {return *yytext;}
.    {yyerror("unrecognized character");}
```

   The problem with the *yacc* code is how to give error messages for ungrammatical strings. This program recognizes the language, and lets *yacc* give its default error on ungrammatical strings:

```
%{

int depth=0;

%}

%%

S : AB '\n'     {printf("depth=%d\n",depth);}
AB : 'a' AB 'b' {depth++;}
   |            ;

%%
```

   Attempts to parse unbalanced strings of *a*s and *b*s such as

```
%{

int depth=0;

%}
```

```
%%

S : AB '\n'     {printf("depth=%d\n",depth);}
  | A AB '\n'   {printf("too many a's\n");}
  | AB B '\n'   {printf("too many b's\n");}
AB : 'a' AB 'b' {depth++;}
   |            ;
A : 'a'
  | 'a' A
B : 'b'
  | 'b' B

%%
```

invariably lead to conflicts, because *yacc* can not decide which rule to match on an a input. An unbalanced amounts of bs can be handled:

```
%{

int depth=0;

%}

%%

S : AB '\n'     {printf("depth=%d\n",depth);}
  | AB B '\n'   {printf("excess of b\n");}

AB : 'a' AB 'b' {depth++;}
   | ;

B : 'b'
  | 'b' B

%%
```

but for a general solution we really need to recognize $\{a^m b^n\}$ and impose the restriction $m \equiv n$ through included C code:

```
%{

int depth=0;

%}

%%

S : A B '\n'     {if (depth==0) printf("matched\n");
                  else if (depth>0) printf("too many a\n");
                  else printf("too many b\n"); }

A : 'a' A {depth++;}
  | ;
```

```
B : ;
  | 'b' B {depth--;}

%%
```

Without the error clauses, this recognizes $a^m b^n$, and it is easy to extend this program to $a^n b^n c^n$.

2.

```
%{

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define NCSNAMES 100
char *csnames[NCSNAMES]; int csnargs[NCSNAMES]; int ncs=0;
int env[100],nenv=0;
int verticalmode = 1;
int lineno = 1;

%}

%token LETTER CHAR WORD
%token BEGINCS ENDCS CONTROLSEQ CONTROLSYM CONTROLSPACE
%token GROUPOPEN GROUPCLOSE

%%

latexfile :
        documentclass environment
| error wordarg environment {printf("No document class\n");}
        ;
documentclass :
        CONTROLSEQ wordarg
            {int ics; ics = findcs("documentclass");
      if ($1!=ics) {
printf("Expecting \\documentclass\n"); YYABORT;}
      printf("Using documentclass <%s>\n",$2);}
environment :
        env_open text env_close ;
env_open :
        BEGINCS wordarg {env_push($2);}
env_close :
        ENDCS wordarg
            {int open=env_pop();
             if (!(strcmp((char*)open,(char*)$2)==0))
       yyerror("Environment mismatch");
    }
text :  ;
        | WORD text
        | environment text ;
wordarg :
```

```
        GROUPOPEN WORD GROUPCLOSE {$$ = $2;}
        ;
spaces : ;
        | ' ' spaces ;

%%

int registercs(char *name,int nargs)
{
  if (ncs==NCSNAMES-1) {
    printf("Can not register any more control sequences\n"); exit;}
  csnames[ncs] = strdup(name); printf("registring <%s> as %d\n",name,ncs);
  csnargs[ncs] = nargs;
  return ncs++;
}

int findcs(char *name)
{
  int loc,i;
  loc = -1; /*printf("finding <%s>",name);*/
  for (i=0; i<ncs; i++)
    if (strcmp(name,csnames[i])==0) {
      loc = i;
    }
  /*printf("=%d\n",loc);*/
  return loc;
}

void env_push(int e) {
  printf("opening environment <%s>\n",(char*)e);
  env[nenv++] = e;
}

int env_pop(void) {
  int e = env[--nenv];
  printf("need closing: <%s>\n",(char*)e);
  return e;
}

void output_char(int c)
{
  printf("%c",c);
  return;
}

void yyerror(char *s)
{
  printf("Parsing failed in line %d because of %s\n",lineno,s);
  return;
}
```

```
int main(void)
{
  registercs("documentclass",1);
  registercs("begin",1); registercs("end",1);
  yydebug=0;
  yyparse();
  return 0;
}
```