Correction de l'exercice 9 :

Fichier .y

```
%%
line :  list '\n'           {printf("OK!\n");YYACCEPT;}
list :  '(' in ')' | 'a'
in :    list | in ',' liste
%%
void yyerror(char* s) { fprintf("%s\n",s); exit(1); }
int yylex() { return getchar(); }
main() { yyparse (); }
```

Correction de l'exercice 10 :

Fichier .lex

```
%{
#include "y.tab.h"
%}
%%
"begin"|\(              {return DEBUT;}
"end"|\)                {return FIN;}
[a-zA-Z_][a-zA-Z0-9_]+  {return ID;}
,|\.|\;|\:              {return SEPARE;}
[ \t\n]+
<<EOF>>                 {return FINFIN;}
.                       {printf("lex erreur!\n");exit(1);}
%%
```

Fichier .y

```
%token DEBUT FIN ID SEPARE FINFIN
%%
fichier :  list FINFIN     {printf("OK\n");YYACCEPT;}
list :     DEBUT in FIN
|          ID
;
in :       list
|          in SEPARE list
;
%%
void yyerror(char* s) { fprintf("%s\n",s); exit(1); }
main () {yyparse ();}
```

Correction de l'exercice 11 :

Fichier .lex

```
%{
#include "y.tab.h"
%}
%%
"+"     {return PLUS;}
"-"     {return MINUS;}
"*"     {return MUL;}
"/"     {return DIV;}
[0-9]+  {yylval=atoi(yytext);return NOMBRE;}
\n      {return FIN;}
[ \t]+
.       {printf("lex erreur\n");exit(1);}
%%
```

Fichier .y

```
%token FIN PLUS MINUS MUL DIV NOMBRE
```

```
%%
line :  expr FIN          {printf("%i\n",$1);YYACCEPT;}
expr :  NOMBRE            {$$=$1;}
|       expr expr PLUS    {$$=$1+$2;}
|       expr expr MINUS   {$$=$1-$2;}
|       expr expr MUL     {$$=$1*$2;}
|       expr expr DIV     {$$=$1/$2;}
;
%%
void yyerror(char* s) { fprintf("%s\n",s); exit(1); }
main () {yyparse ();}
```

Correction de l'exercice 12 :

Au fichier `.lex` de l'exercice précédent, on enlève les définitions de `MINUS` et `DIV`, et on ajoute :

```
"("  {return OPEN;}
")"  {return CLOSE;}
```

Fichier `.y`

```
%token FIN PLUS MUL OPEN CLOSE NOMBRE
%left PLUS
%left MUL
%%
line :  expr FIN          {printf("%i\n",$1);YYACCEPT;}
expr :  NOMBRE            {$$=$1;}
|       expr PLUS expr    {$$=$1+$3;}
|       expr MUL expr     {$$=$1*$3;}
|       OPEN expr CLOSE   {$$=$2;}
;
%%
void yyerror(char* s) { fprintf("%s\n",s); exit(1); }
main () {yyparse ();}
```

Correction de l'exercice 13 :

Au fichier `.lex` précédent, on ajoute la définition :

```
"-"  {return MINUS;}
```

Fichier `.y`

```
%token FIN PLUS MINUS MUL OPEN CLOSE NOMBRE
%left PLUS MINUS
%left MUL
%nonassoc OPPOSE
%%
line :  expr FIN                  {printf("%i\n",$1);YYACCEPT;}
expr :  NOMBRE                    {$$=$1;}
|       expr PLUS expr            {$$=$1+$3;}
|       expr MINUS expr           {$$=$1-$3;}
|       expr MUL expr             {$$=$1*$3;}
|       OPEN expr CLOSE           {$$=$2;}
|       MINUS %prec OPPOSE expr   {$$=-$2;}
;
%%
main () {yyparse ();}
```

Correction de l'exercice 14 :

Fichier `.lex`

```
%{
```

```
#include "y.tab.h"
%}
%%
"+"                               {return PLUS;}
"*"                               {return MULT;}
"@"                               {return CAT;}
"("                               {return LPAR;}
")"                               {return RPAR;}
"len"                             {return LEN;}
"eval"                            {return EVAL;}
[0-9]+                            {yylval.i=atoi(yytext);return NOMBRE;}
\"([^\"\\]* (\\.)?(\\\n)?)* \"    {strncpy(yylval.s,yytext+1,strlen(yytext)-2);
                                   return CHAINE;}
\n                                {return FIN;}
[ \t]+
.                                 {printf("lex erreur\n");exit(1);}
%%
```

Fichier `.y`

```
%{
#include <string.h>
%}
%union   { int i; char s[10000]; }
%token FIN PLUS MULT CAT LEN LPAR RPAR EVAL
%token <i> NOMBRE
%token <s> CHAINE
%left CAT
%left PLUS
%left MULT
%type<i> iexpr
%type<s> sexpr
%%
expr :   sexpr FIN                {printf("resultat : %s\n",$1);YYACCEPT;}
iexpr :  NOMBRE                   {$$=$1;}
|        iexpr PLUS iexpr         {$$=$1+$3;}
|        iexpr MULT iexpr         {$$=$1*$3;}
|        LPAR iexpr RPAR          {$$=$2;}
|        LEN LPAR sexpr RPAR      {$$=strlen($3);}
;
sexpr :  CHAINE                   {strcpy($$,$1);}
|        LPAR sexpr RPAR          {strcpy($$,$2);}
|        sexpr CAT sexpr          {strcpy($$,$1); strcat($$,$3);}
|        EVAL LPAR iexpr RPAR     {sprintf($$,"%i",$3);}
;
%%
void yyerror(char* s) { fprintf("%s\n",s); exit(1); }
main () {yyparse ();}
```

Correction de l'exercice 15 :

La phrase `if` ··· `then if` ··· `then` ··· `else` ··· est ambiguë car le `else` peut correspondre au premier `if` ou au deuxième.
Bison indique donc un conflit *shift-reduce* et choisit automatiquement la rgle *reduce*. Cela veut dire qu'un `else` se rapporte toujours au `if` *le plus intérieur* (ici, le deuxième).
Ce comportement nous conviendra, d'autant plus qu'il est très difficile de supprimer ce conflit !