Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.

Übung 7: Parsererzeugung mit `yacc` / `bison`

(Besprechung am Montag, den 23. Juni 2003)

PL2 hat folgende Syntax (Lexeme sind fett und unterstrichen dargestellt):

```
program     ::= block .
block       ::= {declaration}
                begin statements end.
declaration ::= const id = number { , id = number} ;
              | var id {, id} : type ;
              | procedure id ; block ;.
type        ::= id
statements  ::= statement { ; statement }.
statement   ::=
              [ id := formula
              | id
              | ? id
              | ! ( formula | string )
              | if formula then statements
                { elsif formula then statements}
                [ else statements ]
                end if
              | while formula
                do statements
                end while
              ].
formula     ::= conjunction { or conjunction }.
conjunction ::= relation { and relation }.
relation    ::= expression [ ( = | # | < | > | <= | >= ) expression ]
expression  ::= [ + | - ] term { ( + | - ) term } .
term        ::= factor { ( * | / ) factor } .
factor      ::= id
              | number
              | ( ODD | NOT ) factor
              | ( formula ).
```

1. Erzeugt für diese Sprache einen Parser mit `yacc` oder `bison`. Dazu müsst Ihr die EBNF-Abkürzungen für Wiederholung und Option auflösen. Welche Konflikte (*shift-reduce* oder *reduce-reduce*) treten auf? Wie können sie behoben werden?

2. Vereinfacht die Syntax von Formeln, so dass sie mehrdeutig ist, und behebt die Konflikte mit entsprechenden Angaben zu Assoziativität und Präzedenz.