1 Syntax Analysis & Abstract Syntax

1. You're given the following concrete syntax for a subset of the ftp command-language:

```
Input   = { Command } [ BYE NL ]
        .
Command = ASCII NL
        | BIN NL
        | LS [ FILEID ] NL
        | MPUT { FILEID } NL
        | MGET { FILEID } NL
        .
```

   (a) Give a handwritten top-down parser that checks input correctness.

   (b) Give grammar rules as they are understood by JavaCup (eliminate {}[ ]).

   (c) Find an abstract tree representation for this language.

   (d) Give the class Tree with the subclasses for the abstract syntax tree (The subclasses for LS and MPUT and FILEID are enough). Do not forget the declaration of the visitor interface!

   (e) Add actions to the grammar rules so that the parser constructs the tree.

   (f) Write a pretty print visitor for this tree (The main routine and the caseLs, caseMput, and caseFileId are enough)!

2. Consider the following input

```
mget xxx.tex
mput yyy.tex zzz.tex
bye
```

   (a) Give the abstract syntax tree corresponding to the input!

   (b) Give a stack-trace, at the moment the pretty-printer calls caseMget and at the moment it prints zzz.tex.

3. Write an EBNF grammar for the following languages.

   (a) The language of all (possibly empty) sequences of (and) (no other characters) that have balanced parenthesis.

   (b) The language of all (possibly empty) sequences of (,), [, and] (again no other characters) that have balanced parenthesis.

   (c) The language of all (possibly empty) sequences of a's followed by some b's, but more b's than a's.

4. Give first, follow, and nullable for the non-terminals of the following grammar: Which of them are problematic for top-down parsing. How can you use left-recursion elimination and left-factoring to get rid of the problems?

   (a)
   E = T E | L.
   T = ident ident.
   L = ident { num }.

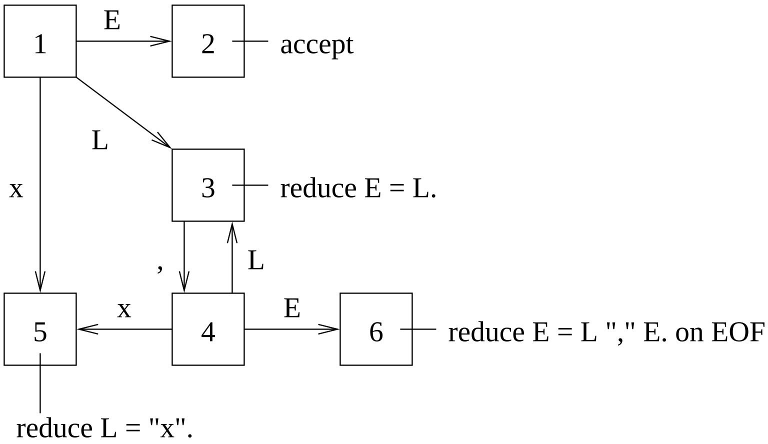
   (b)
   E = E T | T.
   T = ident ident.

   (c)
   E = T E | L
   T = ident ident.
   L = { num }.

5. Consider

   E = L "," E | L.
   E = "x".

   and the following stack-automaton.

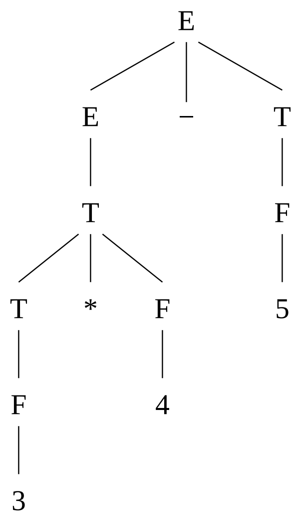


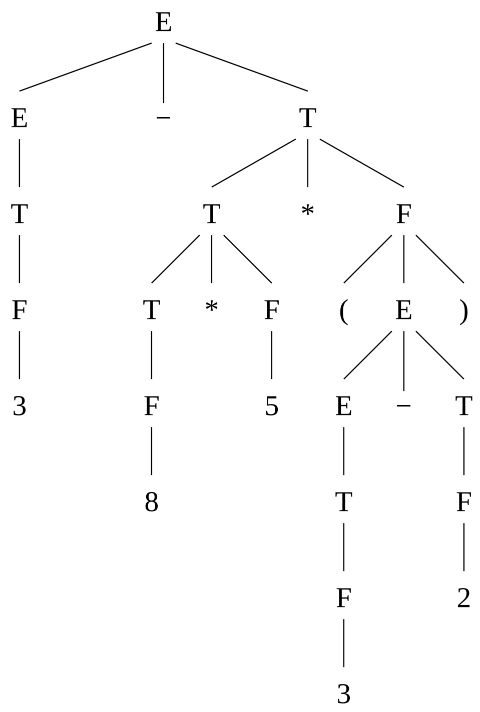
   Show how the input "x,x,x" gets accepted from the parser, that is continue the following table

Stack	Input	Action
$_1$	x,x,x$	shift
$_1x_5$	,x,x$	reduce L = x

6. Transform the following concrete syntax-trees into abstract syntax trees. Use the abstract syntax

```
Expr = OPERATION Expr Expr op
     | NUMLIT
     ;
```





2 Solutions

1. (a) The hand-written parser:

```
void Input() {
    nextToken();
    while (token == ASCII || token == BIN || token == LS
        || token == MPUT || token == MGET || token == NL) {
        Command();
    }
    if (token == BYE) {
        nextToken();
        if (token == NL) nextToken() else error();
    }
}
```

```
void Command() {
    switch (token) {
    case ASCII:
        nextToken();
        if (token == NL) nextToken() else error();
        break;
    case BIN:
        nextToken();
        if (token == NL) nextToken() else error();
        break;
    case LS:
        nextToken();
        if (token == FILEID)
            nextToken();
        if (token == NL) nextToken() else error();
        break;
    case MPUT:
        nextToken();
        while (token == FILEID)
            nextToken();
        if (token == NL) nextToken() else error();
        break;
    case MGET:
        nextToken();
        while (token == FILEID)
            nextToken();
        if (token == NL) nextToken() else error();
        break;
    default:
        error();
        break;
    }
}
```

(b) The grammar for JavaCup:

```
Input          = CommandList
               | CommandList BYE NL
               ;
CommandList    = /* empty */
               | CommandList Command
               ;
Command        = ASCII NL
               | BIN NL
               | LS FILEID NL
               | LS  NL
               | MPUT FileIds NL
               | MGET FileIds NL
               ;
FileIds        =  /* empty */
               |  FILEID FileIds
               ;
```

(c) The abstract syntax:

```
Input        = COMMANDLIST { Command }
             ;
Command      = SETBINASCII boolean
             | LS FileId
             | MPUT { FileId }
             | MGET { FileId }
             ;
FileId       = FILEID String
             ;
```

(d) The class Tree:

```
package ftp;
public abstract class Tree {
    int pos;

    public Tree(int pos) {
        this.pos = pos;
    }

    public abstract void apply(Visitor v);
```

```
public static class CommandList extends Tree {
        Tree[ ] commands;

        public CommandList(int pos, Tree[ ] commands) {
                this.commands = commands;
                super(pos);
        }

        public void apply(Visitor v) {
                v.caseCommandList(this);
        }
}
public static class SetBinAscii extends Tree {
        boolean ascii;

        public SetBinAscii (int pos, boolean ascii) {
                super(pos);
                this.ascii = ascii;
        }

        public void apply(Visitor v) {
                v.caseSetBinAscii(this);
        }
}
public static class Ls extends Tree {
        Tree fid;
        public Ls(int pos, Tree fid) {
                super(pos);

        }

        public void apply(Visitor v) {
                v.caseLs(this);
        }
}
```

```
    public static class Mput extends Tree {
        Tree[ ] fid;
        public Mput(int pos, Tree[ ] fid) {
            super(pos);
            this.fid = fid;
        }

        public void apply(Visitor v) {
            v.caseMput(this);
        }
    }
    public static class Mget extends Tree {
        Tree[ ] fid;
        public Mget(int pos, Tree[ ] fid) {
            super(pos);
            this.fid = fid;
        }

        public void apply(Visitor v) {
            v.caseMget(this);
        }
    }
    public static class FileId extends Tree {
        String name;
        public Mget(int pos, String name) {
            super(pos);
            this.fid = fid;
        }

        public void apply(Visitor v) {
            v.caseFileId(this);
        }
    }
    public interface Visitor {
        void caseCommandList(CommandList tree);
        void caseSetBinAscii(SetBinAscii tree);
        void caseLs(Ls tree);
        void caseMput(Mput tree);
        void caseMget(Mget tree);
        void caseFileId(FileId tree);
    }
}
```

(e) The grammar for JavaCup (including tree building):

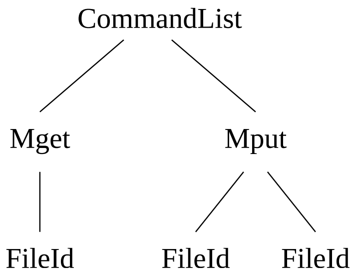
```
Input          =  CommandList:c
                  {: RESULT = new CommandList(cleft, c.toArray()); :}
               |  CommandList:c BYE NL
                  {: RESULT = new CommandList(cleft, c.toArray()); :}
               ;
CommandList    = /* empty */
                  {: RESULT = new TreeArrayBuffer(); :}
               |  CommandList:cl Command:c
                  {: RESULT = cl.append(c); :}
               ;
Command        = ASCII:a NL
                  {: RESULT = new SetBinAscii(aleft, true); :}
               |  BIN:b NL
                  {: RESULT = new SetBinAscii(bleft, false); :}
               |  LS:l FILEID:f NL
                  {: RESULT = new Ls(lleft, new FileId(fleft, f)); :}
               |  LS:l  NL
                  {: RESULT = new Ls(lleft, null); :}
               |  MPUT:l FileIds:f NL
                  {: RESULT = new Mput(lleft, f.toArray()); :}
               |  MGET:l FileIds:f NL
                  {: RESULT = new Mget(lleft, f.toArray()); :}
               ;
FileIds        =  /* empty */
                  {: RESULT = new TreeArrayBuffer(); :}
               |   FileIds:fs FILEID:f
                  {: RESULT = fs.append(new FileId(fleft, f)); :}
               ;
```

(f) The pretty print visitor:

```
package ftp;
import java.io.*;
public class Printer implements Tree.Visitor {
    public static void prettyPrint(Tree tree) {
        tree.apply(new Printer());
        System.out.println();
    }
    protected void print(Tree tree) {
        tree.apply(this);
    }
```

```
    public void caseCommandList(Tree.CommandList tree) {
        for (int i = 0; i < tree.commands.length; i ++) {
            print(tree.commands[i]);
            System.out.println();
        }
    }

    public void caseSetBinAscii(Tree.SetBinAscii tree) {
        if (tree.ascii)
            System.out.print("ascii");
        else
            System.out.print("bin");
    }

    public void caseMLs(Tree.Ls tree) {
        System.out.print("ls ");
        if (tree.fid != null)
            print(tree.fid);

    }

    public void caseMput(Tree.Mput tree) {
        System.out.print("mput ");
        for (int i = 0; i < tree.fid.length; i ++) {
            print(tree.fid[i]);
            System.out.print(" ");
        }
    }

    public void caseMget(Tree.Mget tree) {
        System.out.print("mget ");
        for (int i = 0; i < tree.fid.length; i ++) {
            print(tree.fid[i]);
            System.out.print(" ");
        }
    }

    public void caseFileId(Tree.FileId tree) {
        System.out.print(tree.name);
    }
}
```

2. (a) The abstract syntax tree:



(b) mget, mput, and fileid represent the objects of class Mget, Mput, and FileId respectively.

```
Printer.prettyPrint(CommandList)          // p = new Printer()
CommandList.apply(p)
p.caseCommandList(CommandList)
p.print(mget)
mget.apply(p)
p.caseMget(mget)
```

(c) Stack trace 2

```
Printer.prettyPrint(CommandList)          // p = new Printer()
CommandList.apply(p)
p.caseCommandList(CommandList)
p.print(mput)
mput.apply(p)
p.caseMput(mput)
p.print(fileid)
fileid.apply(p)
p.caseFileId(fileid)
System.out.print("zzz.tex")
```

3. The grammars are

(a) P = ϵ | "(" P ")" P

(b) P = ϵ | "(" P ")" P | "[" P "]" P

(c) C = "a" C "b" | C "b" | "b"

4. (a) We have to left-factor ident.

E = ident (ident E | { num }).

	first	follow
E	ident	EOF
T	ident	ident
L	ident	EOF

nullable = $\emptyset$

(b) We have to eliminate left-recursion

E = { T } T.
T = ident ident.

	first	follow
E	ident	ident, EOF
T	ident	ident, EOF

nullable = ∅

(c) That's already OK.

E = T E | L
T = ident ident.
L = { num }.

	first	follow
E	ident, num	EOF
T	ident	ident, num, EOF
L	num	EOF

nullable = { E, L }

5. The completed table:

Stack	Input	Action
${}_1$	x,x,x\$	shift
${}_1x_5$	,x,x\$	reduce L = x
${}_1L_3$	,x,x\$	shift
${}_1L_3,_4$	x,x\$	shift
${}_1L_3,_4x_5$	,x\$	reduce L = x
${}_1L_3,_4L_3$	,x\$	shift
${}_1L_3,_4L_3,_4$	x\$	shift
${}_1L_3,_4L_3,_4x_5$	\$	reduce L = x
${}_1L_3,_4L_3,_4L_3$	\$	reduce E = L
${}_1L_3,_4L_3,_4E_6$	\$	reduce E = L, E
${}_1L_3,_4E_6$	\$	reduce E = L, E
${}_1E_2$	\$	accept

6. We write O for Operation and N for NumLit.