REGULAR EXPRESSIONS

$address =~ m/(\d* .*)\n(.*?, ([A-Z]{2}) (\d{5})-?(\d{0,5})/

By,

Uzma Sulthana

Contents

- Atoms. Operators
- grep. Operation. grepFamily. Searching for File Content.
- Sed. Scripts. Operations. Addresses. Commands. Applications. grep and sed. awk.
- Execution. Fields and Records. Scripts. Operations. Patterns. Actions. Associative Arrays.
- String Functions. Mathematical Functions. User-Defined Functions. Using System Commands in awk.
- Applications

Introduction to Regular Expressions

Pattern Matching

- Regular Expression is a pattern consisting of a sequence of characters that is matched against text
- Unix evaluates text against the pattern to determine if the text and the pattern match.
- If matches, the expression is true and a command is executed
- If not matches, the expression is false and a command is not executed
- Unix utilities ---- grep and sed –use Reg

Introduction to Regular Expressions

- In this we need to learn basic concepts of Regular expression use to including searching for and replacement of text.
- A regular expression is a mathematical expression.
- Mathematical expression is made of operands (data) and operators
- Regular expression is made of atoms and operators.
- Atom: specifies what we are looking for and where in the text the match is to be made.
- Operator: which is not required in all expressions, combines atoms into complex expressions.

Introduction to Regular Expressions

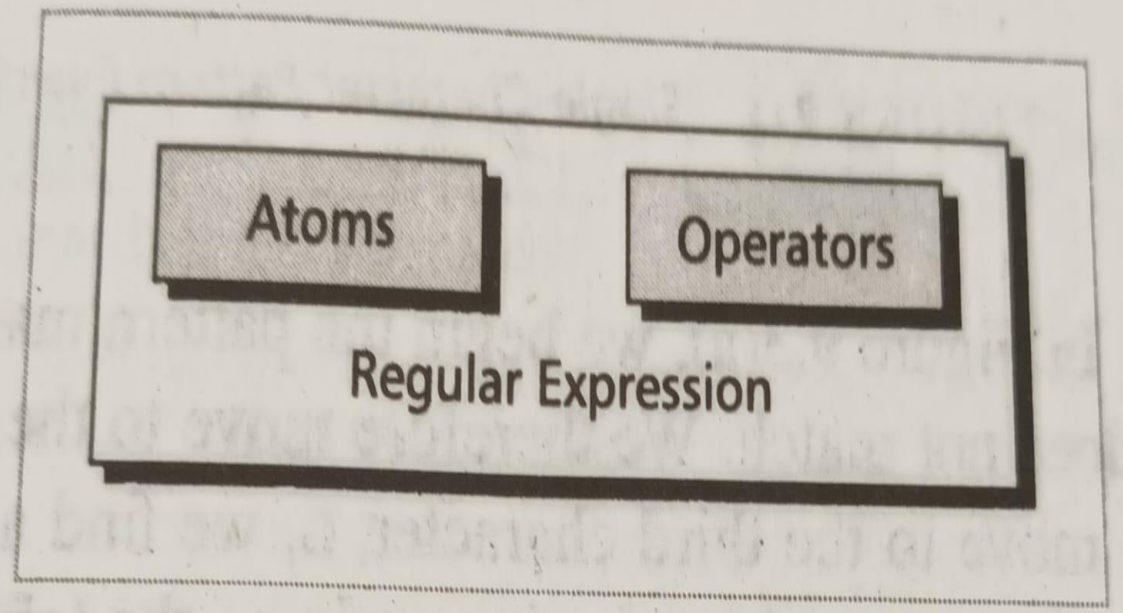


FIGURE 9.1 *Regular Expression*

Mathematical Expression consisting of Operands and Operators

Atoms

- As mentioned, an atom specifies what text is to be matched and where it is to be found.
- An atom in a regular expression can be one of five tyeps:
- 1. single character
- 2. a dot
- 3. a class
- 4. an anchor
- 5. back reference.

Atoms

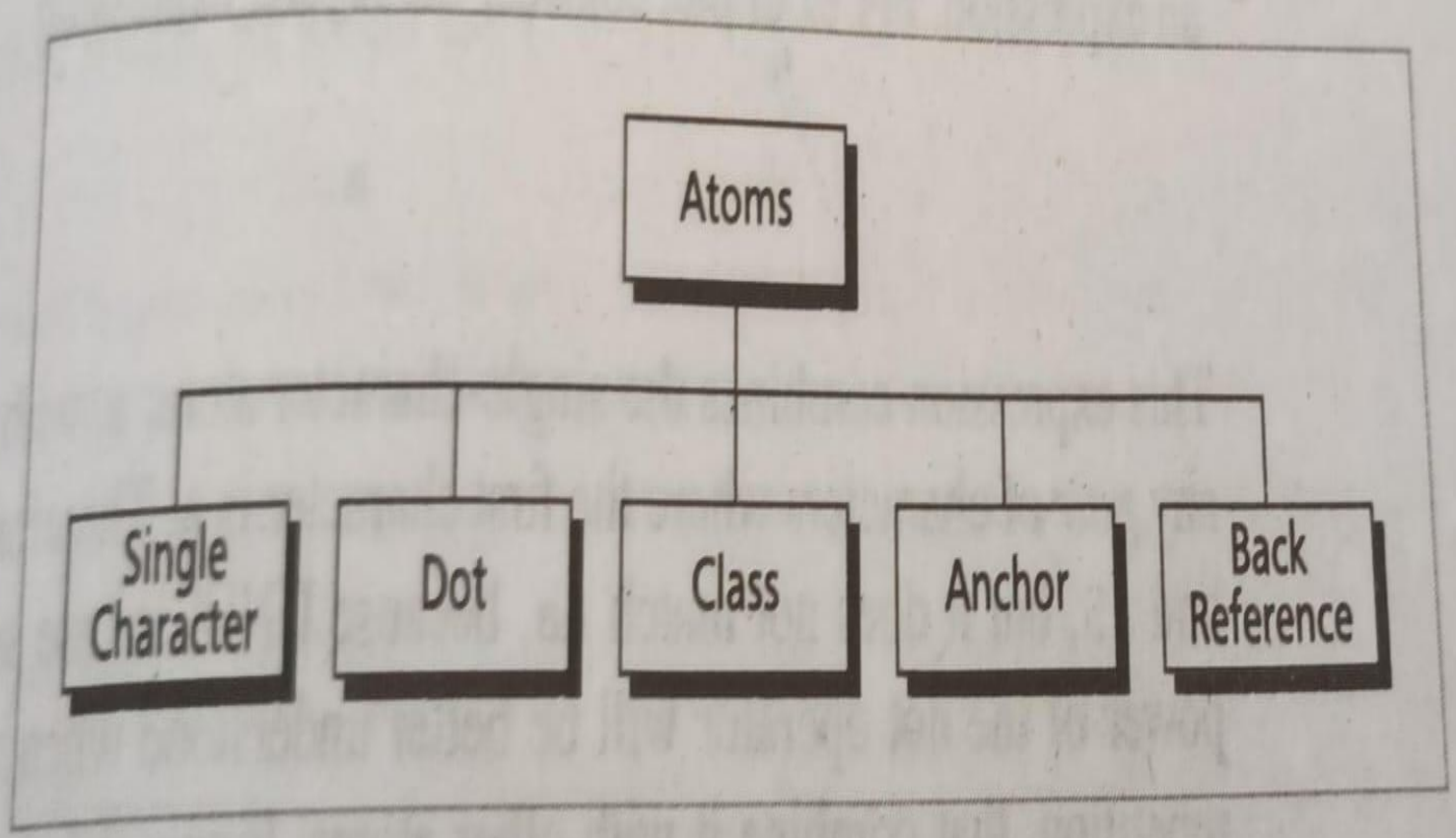


FIGURE 9.2 *Atoms*

Single character

- The simplest atom is a single character.
- When single character appears in a regular expression, it matches itself. In other words- if RE made of one single character, that character must be somewhere in the text to make the pattern match successful.
- Conversely, if the character does not appear in the text, the pattern match is unsuccessful.
- An example of successful and unsuccessful single character pattern match is presented in Below figure

Single Character

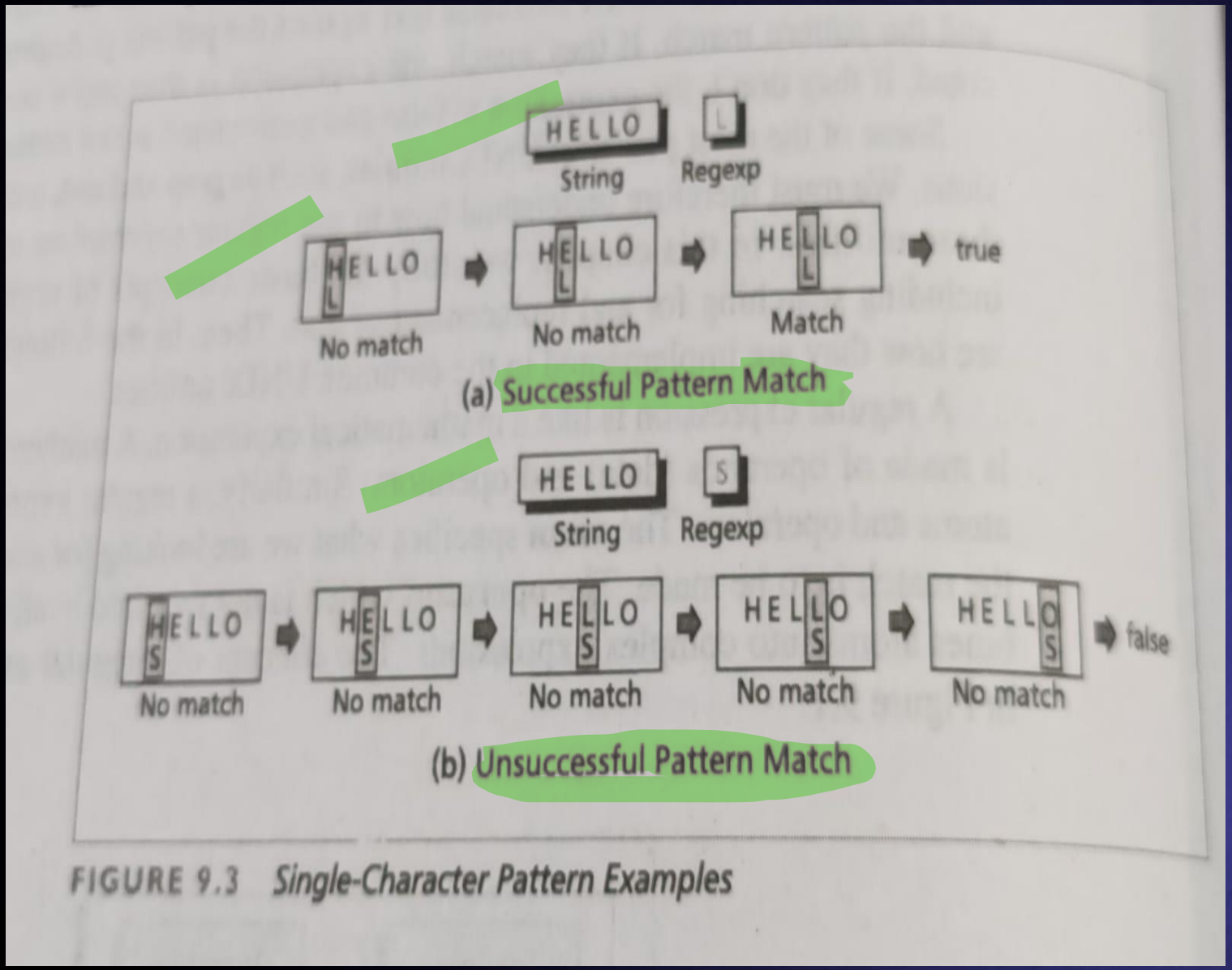


FIGURE 9.3 *Single-Character Pattern Examples*

Dot

- A dot matches any single character except the newline character (\n).
- Example : a.
- This expression combines the single-character atom a, with dot atom.
- It matches pair of character where the first character is a.
- It maches aa, ab,aX, a5 but it does not match Aa, because UNIX is case sensitive , or a\n.

DOT

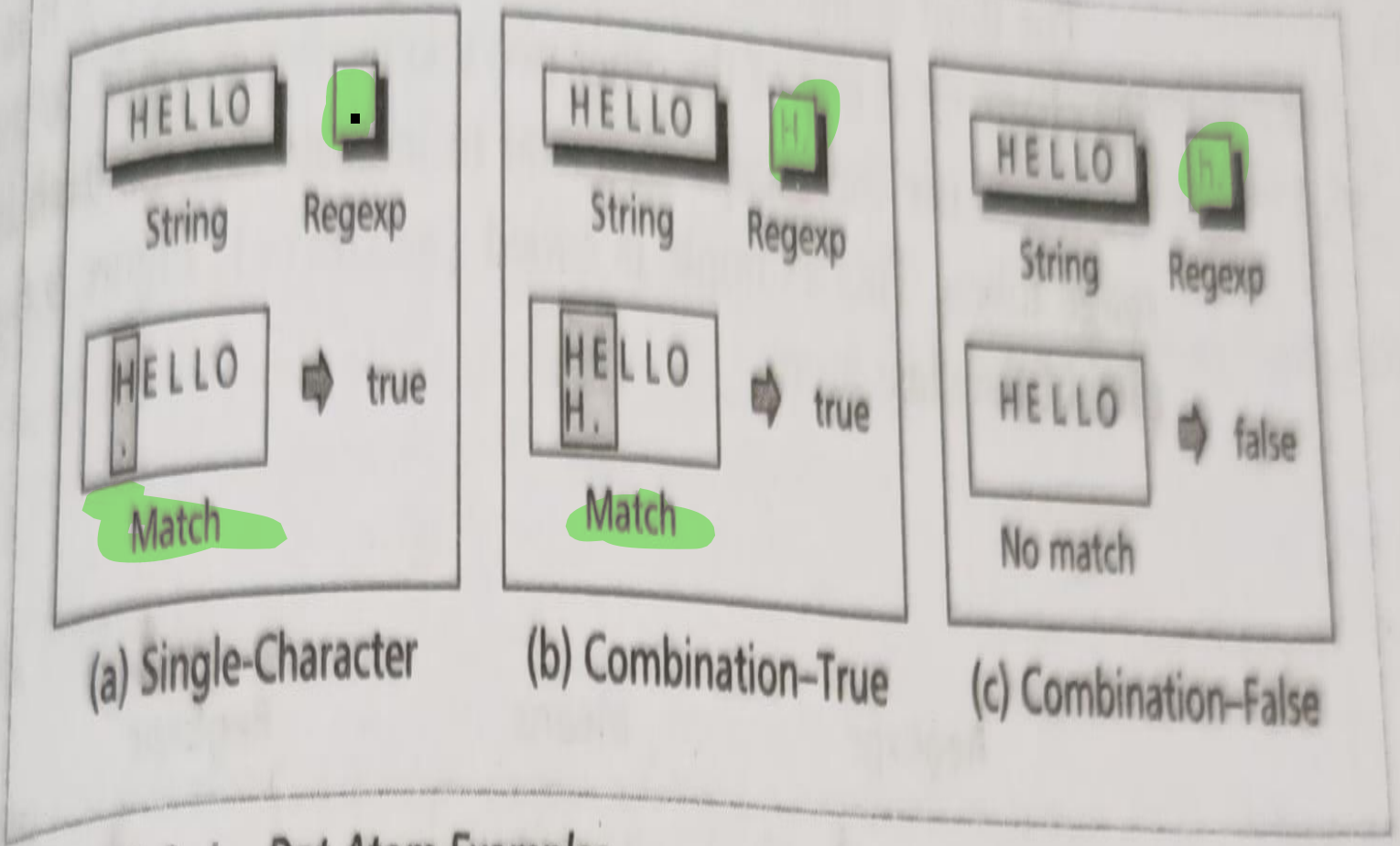


FIGURE 9.4 *Dot Atom Examples*

Class

- The dot matches any single character.
- The class atom defines a set of ASCII characters, any one of which may match any of the characters in the text.
- The character set to be used in the matching process is enclosed in the brackets
- The class set is a very powerful expression component.
- Its power is extended with three additional tokens: ranges, exclusion and escape characters.

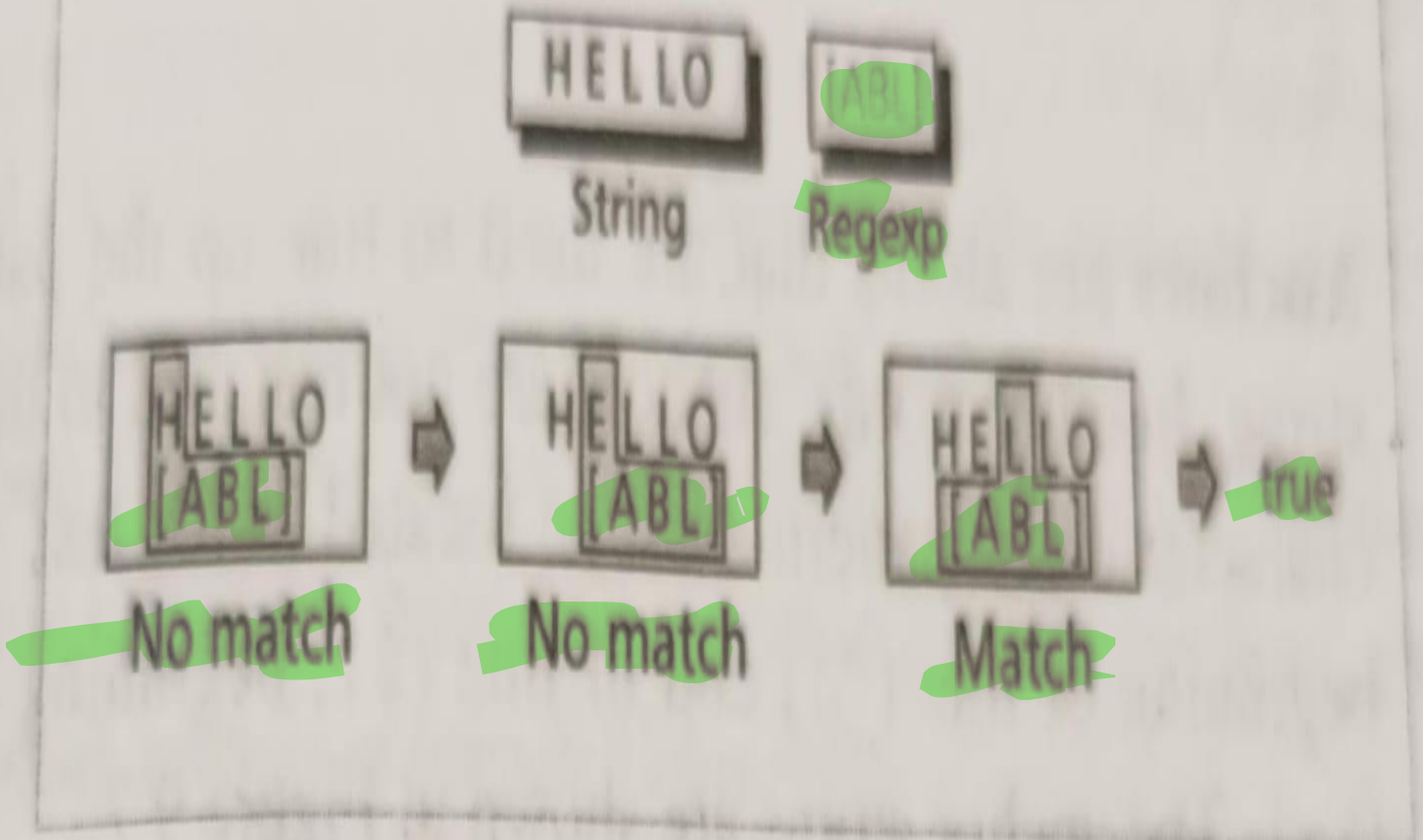


FIGURE 9.5 *Class Atom Example*

Class

- Range: text characters is indicated by a dash(-).
- Example : [a-d] --- indicates the characters a,b,c and d are included in the set.
- Exclusion: Specify which characters are to be excluded from the set. Operator used (^)
- Example: to specify any characters other than vowel, [^aeiou].
- The not operator can also be used with ranges.
- To specify any character other than a digit, [^0-9]

Class

- Escape character (\): It is used when the matching character is one of the other two tokens.
- Example: to match a vowel or a dash, we use the escape character to indicate that the dash is a character and not a range token.
- [aeiou\-]

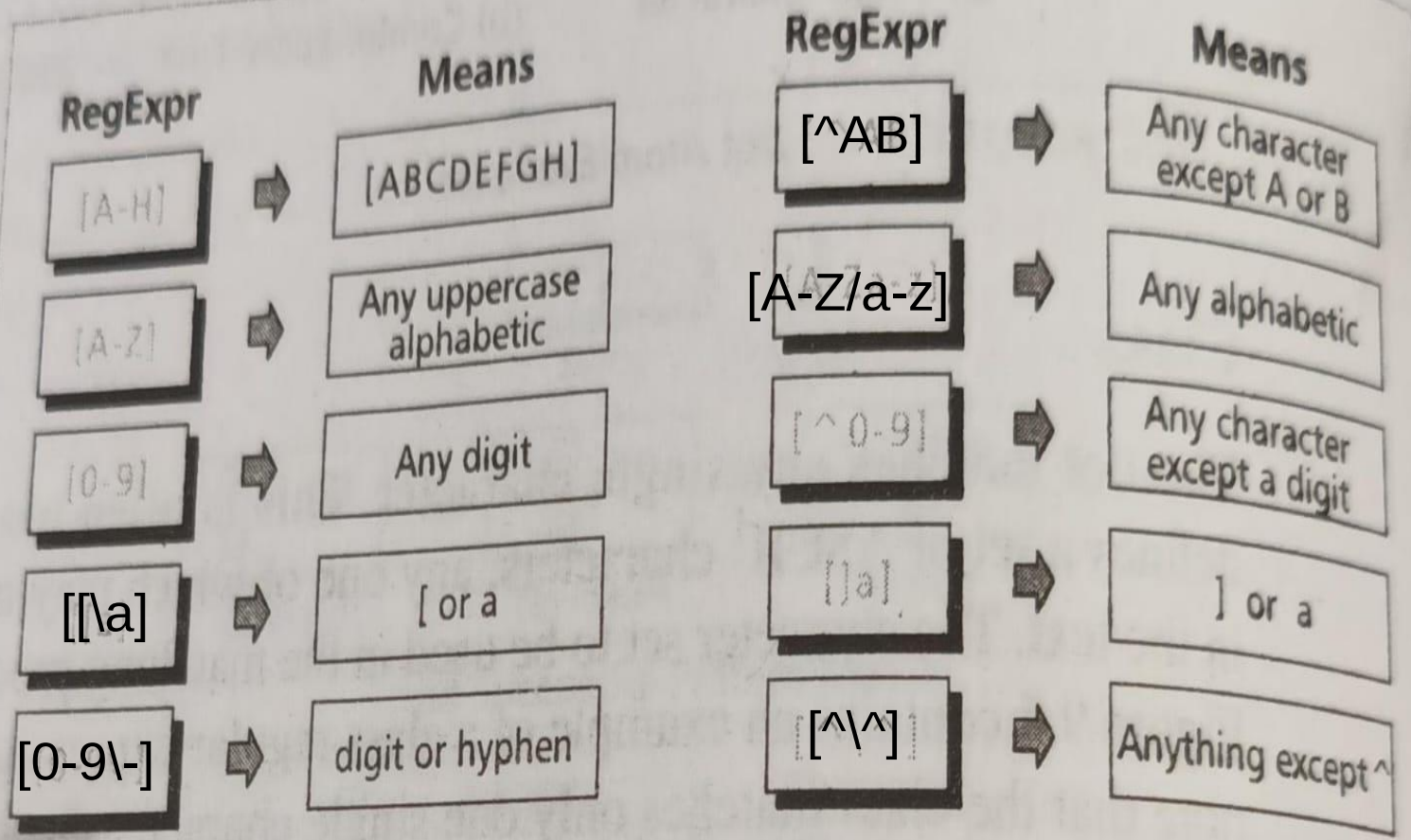


FIGURE 9.6 *Examples of Classes*

Anchors

- Anchors are atoms that are used to line up the pattern with a particular part of a string.
- There are four types of anchors:
- 1. Beginning of line (^)
- 2. end of line ($)
- 3. beginning of word (\<)
- 4. end of word (\>)

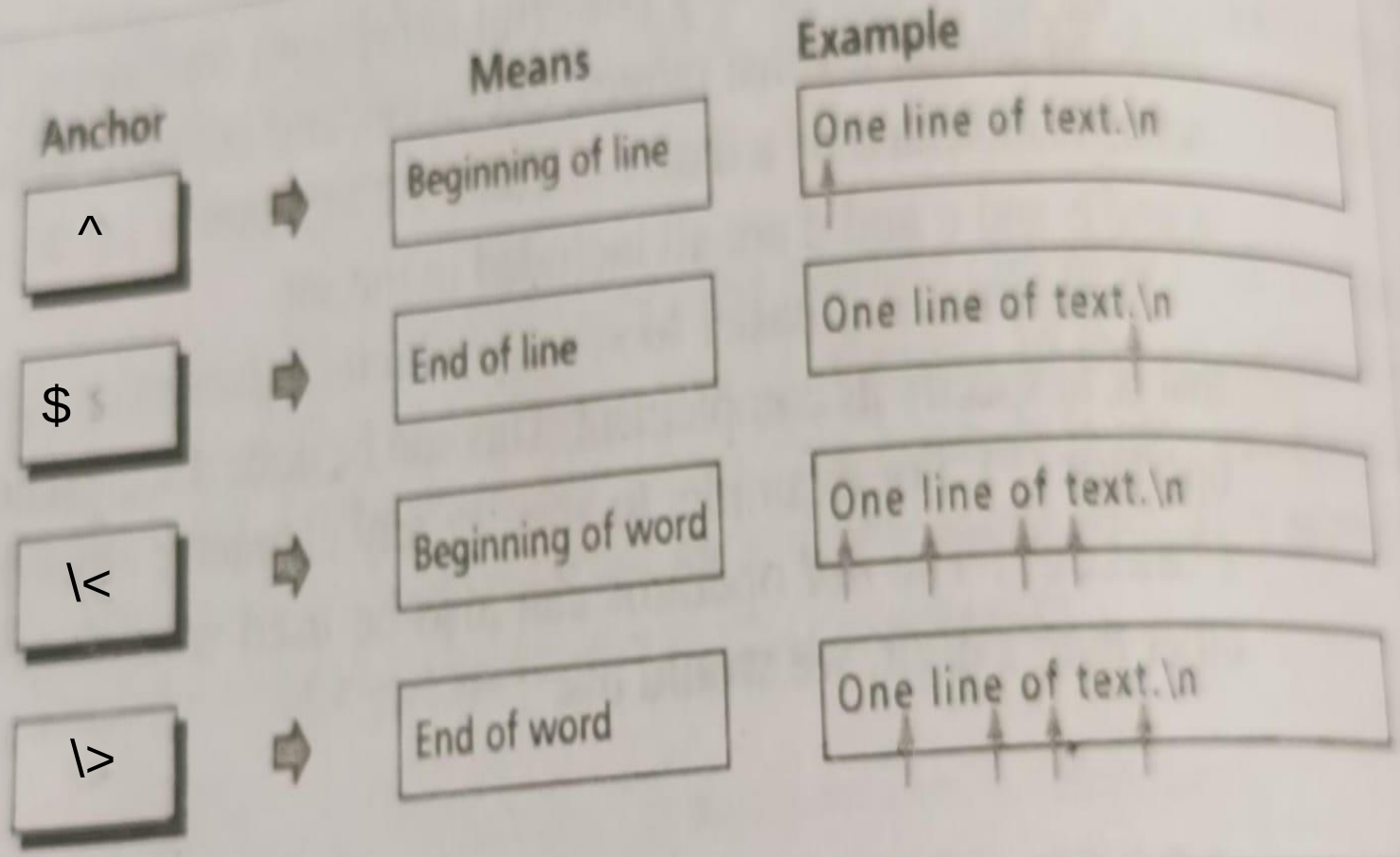


FIGURE 9.7 *Anchors*

Back References

- A back reference is coded using the escape character and a digit in the range of 1 to 9
- Example: \1,\2,\3…\9
- A back reference is used to match text in the current or designated buffer with text that has been saved in one of the systems's nine buffer.

The regex \([A-Z]\).*\1 is used to match a text that contains two identical uppercase letters separated by any number of characters. The \([A-Z]\) part matches and captures one uppercase letter, and the \1 part matches the same letter that was captured. The .* part matches any number of characters in between.

First char is stored in Buffer 1

For
\([A-Z]\).*\1
For text NACDEXGHIJABC

1. N is stored in Buffer 1
And then it fails to repeat anywhere
2. It checks A and saves A in Buffer 1
It repeats itself and then that start to end position is the pattern matched

Operators

- The atoms with operators are combined to make Regex more powerful
- Regex operators play the same role as the mathematical operators
- Mathematical expression operators combine mathematical atoms (data): regular expression operators combine regular expression atoms.
- Regular expressions into 5 different categories

1. Sequence operators
2. alternation operators
3. repetition operators
4. group operators
5. save operators

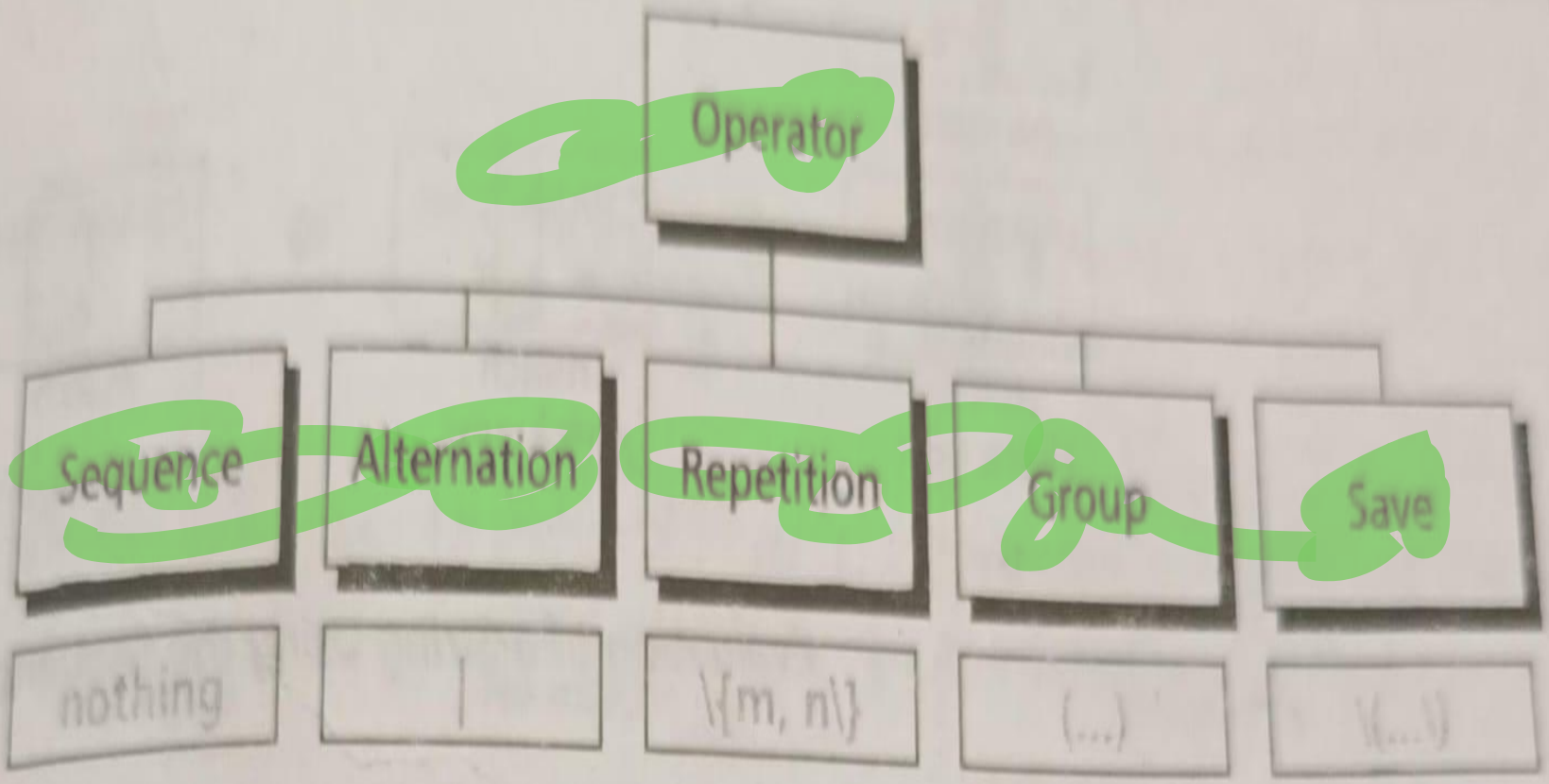


FIGURE 9.8 *Operators*

Sequence

- The Sequence operator is *nothing*.
- Example :

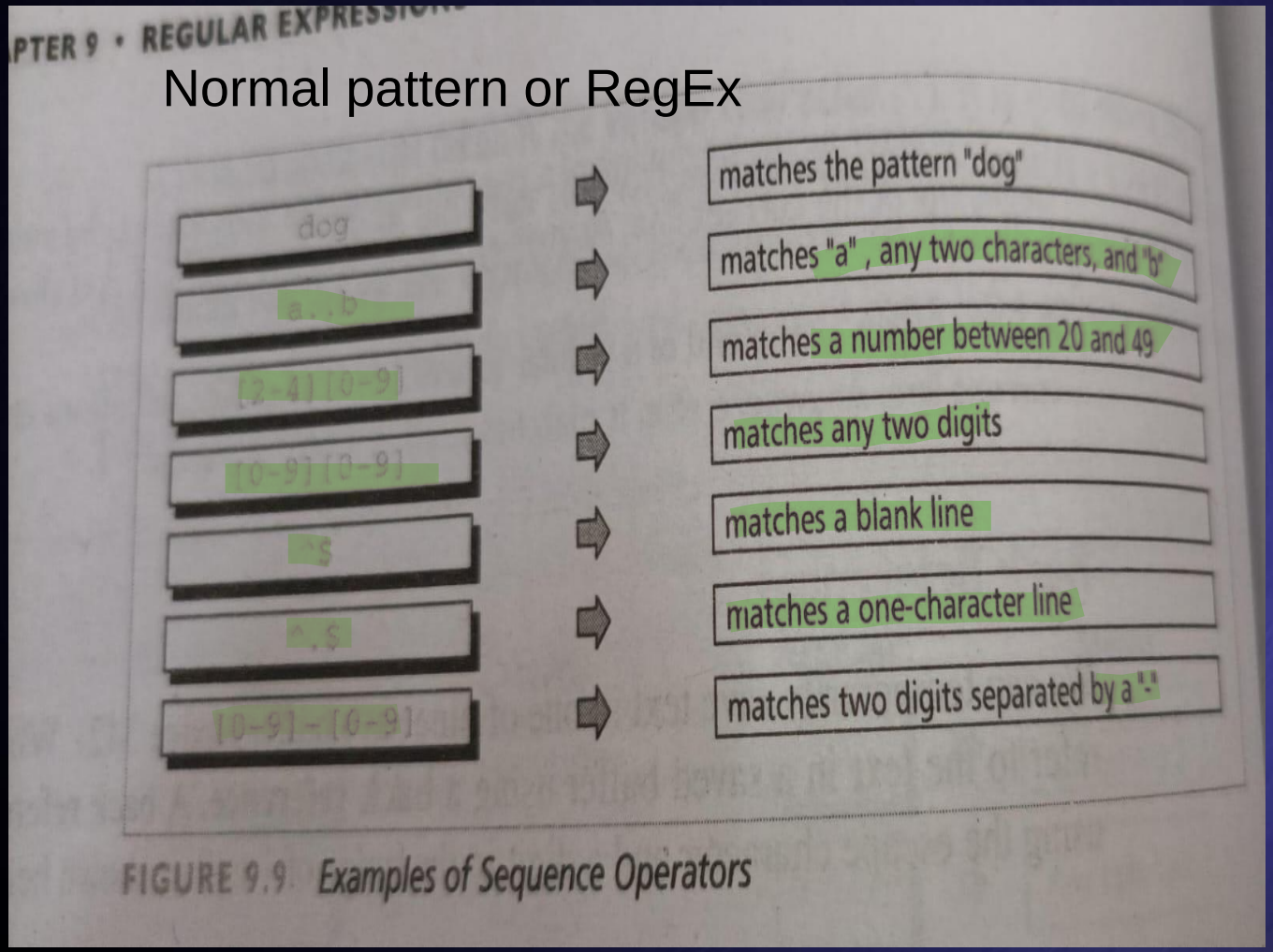


FIGURE 9.9 Examples of Sequence Operators

Sequence

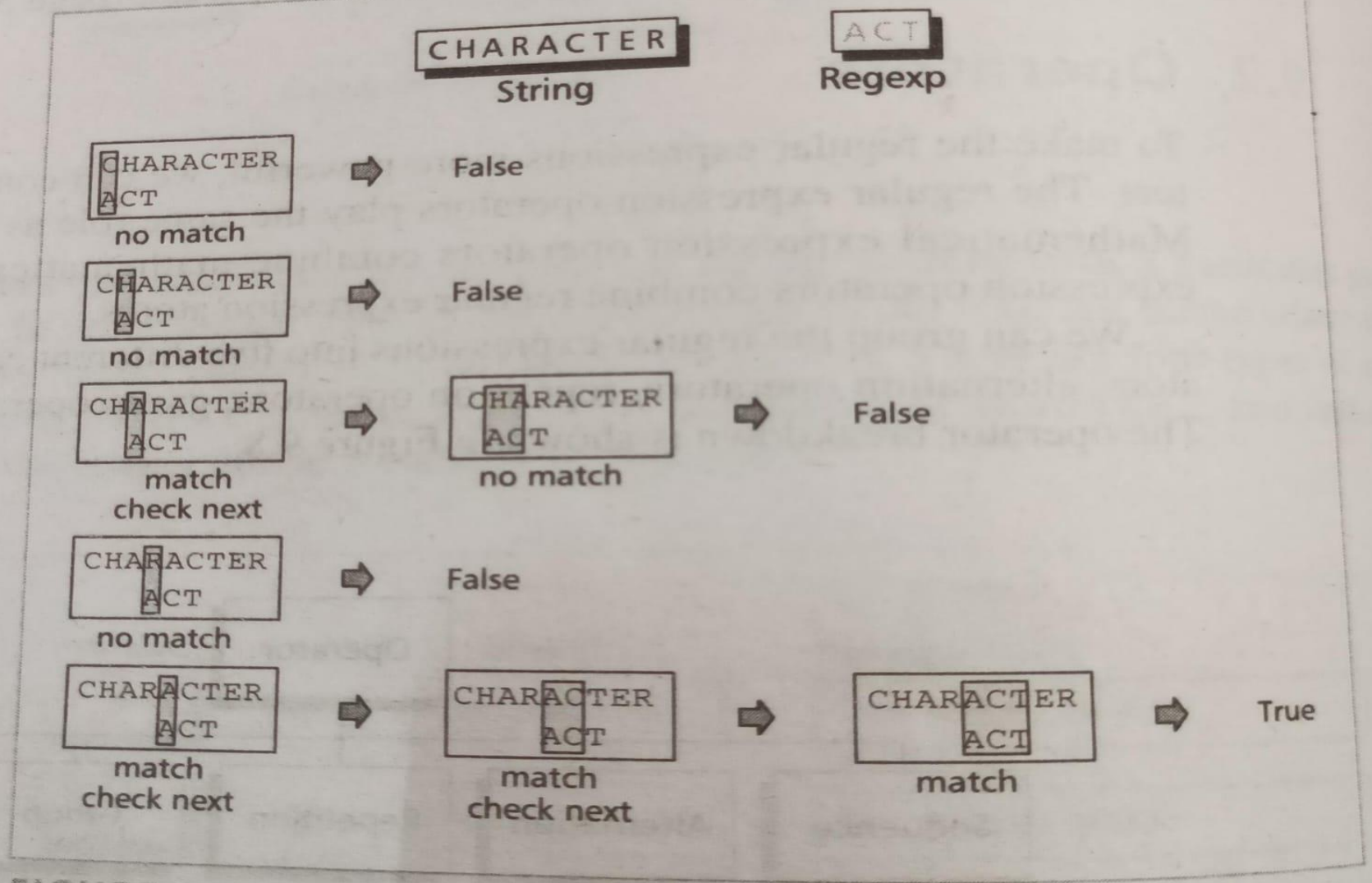


FIGURE 9.10 *Evaluation of a String Using the Sequence Operator*

Alternation

- Alternation operator (|) is used to define one or more alternatives.
- Example we need to select between A or B ---code as A|B

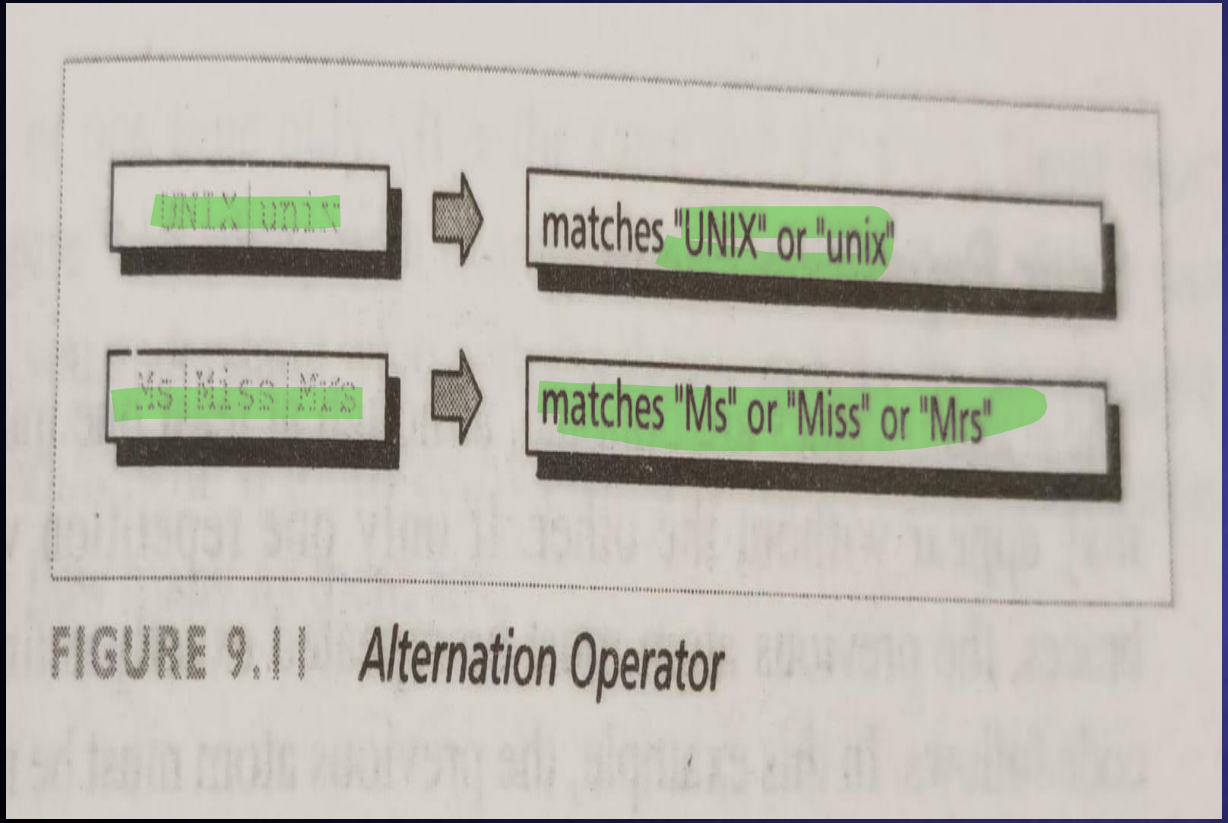


FIGURE 9.11 Alternation Operator

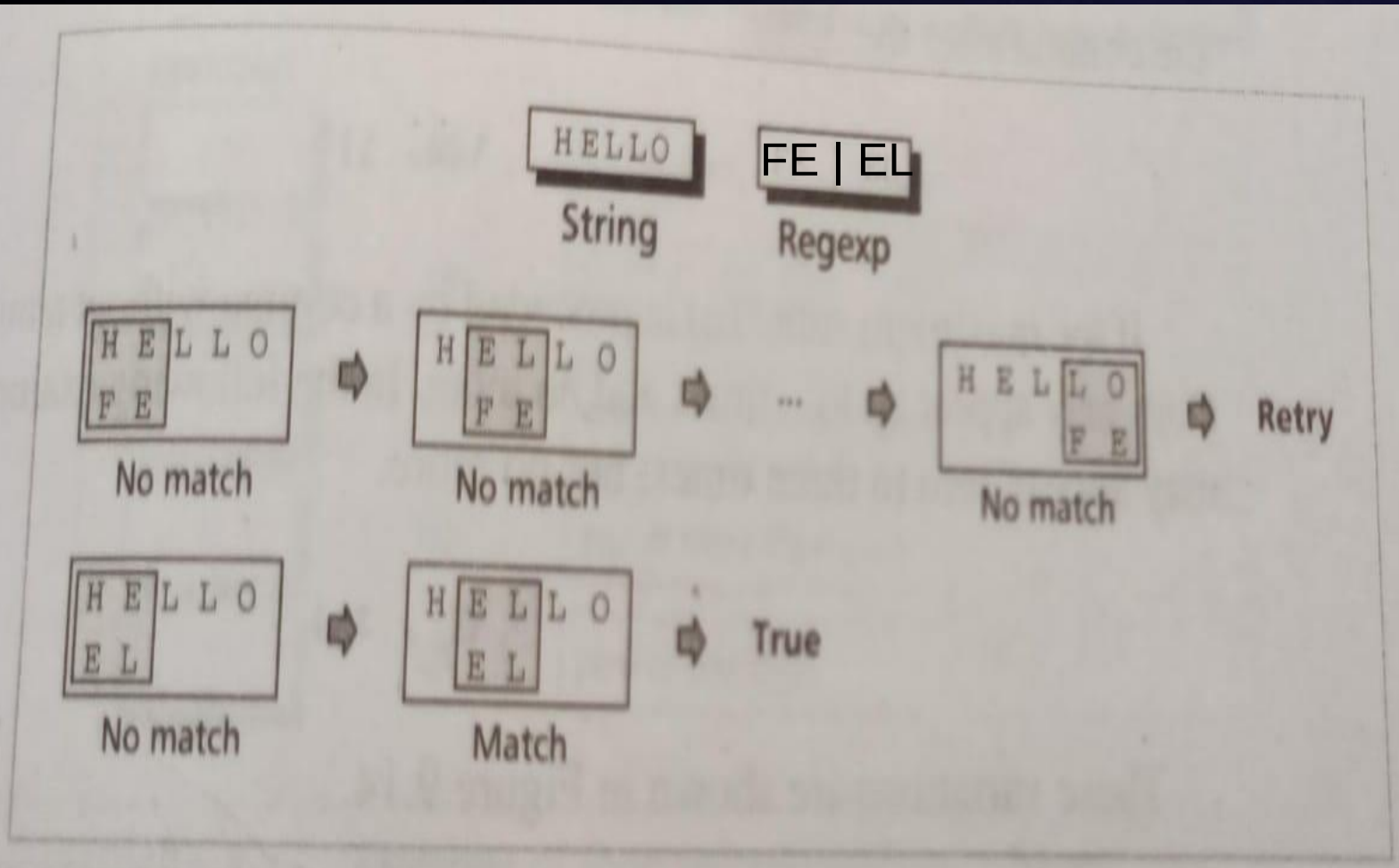


FIGURE 9.12 *Matching Alternation Operators*

Repetition

- The repetition operator is a set of escaped braces (\{….\}) that contains two numbers separted by a comma. A\{3,5\} or BA\{3,5\}

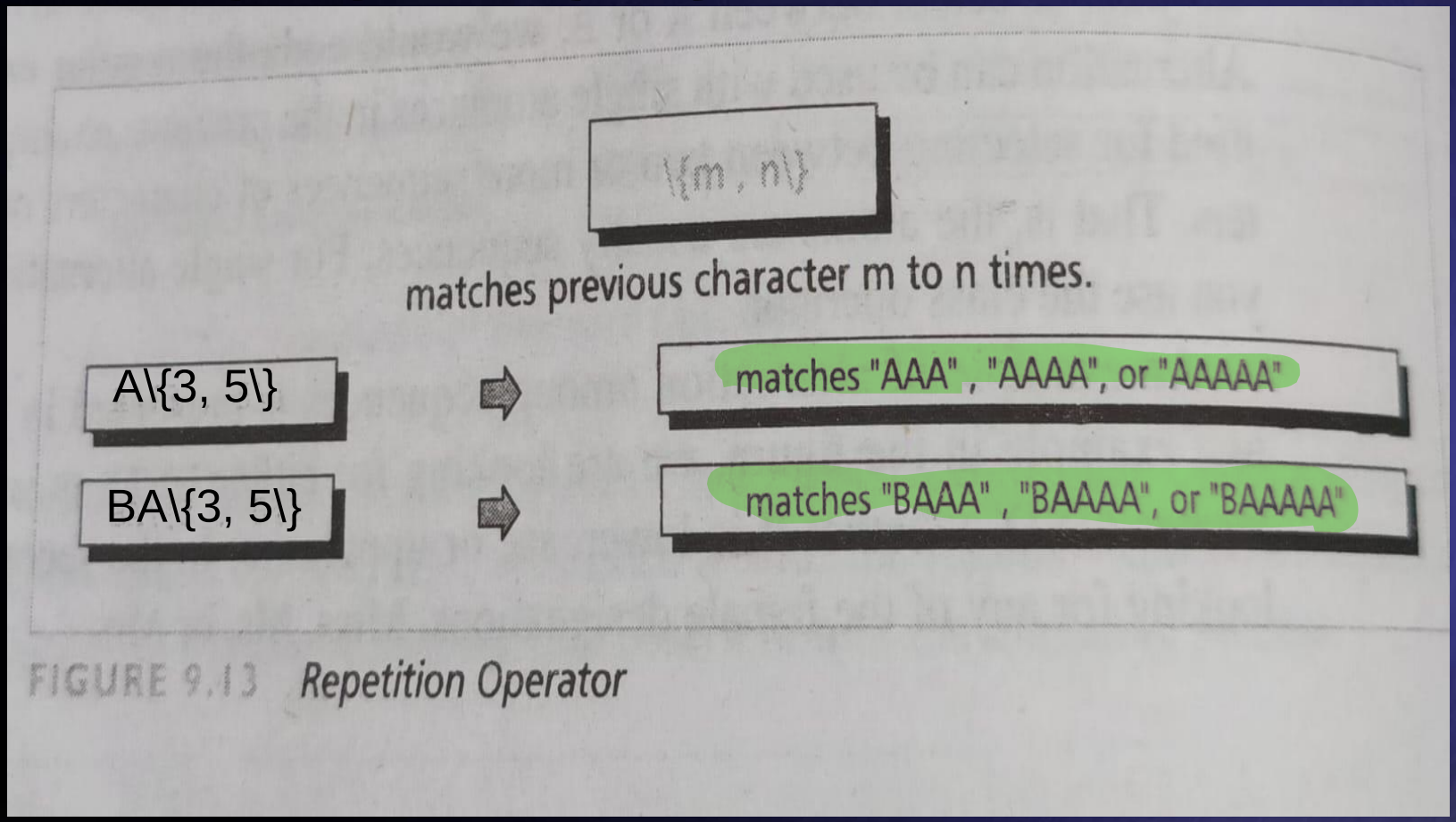


FIGURE 9.13 *Repetition Operator*

Repetition

- Basic repetition forms (m and n values are optinal
- At least one should be there.

- Short form Operators:
- Asterisk (*) may be used to repeat an atom zero or more times
- The Plus (+) is used to specify that the atom must appear one or more times
- The question mark (?) is used to repeat the pattern zero or one time only.

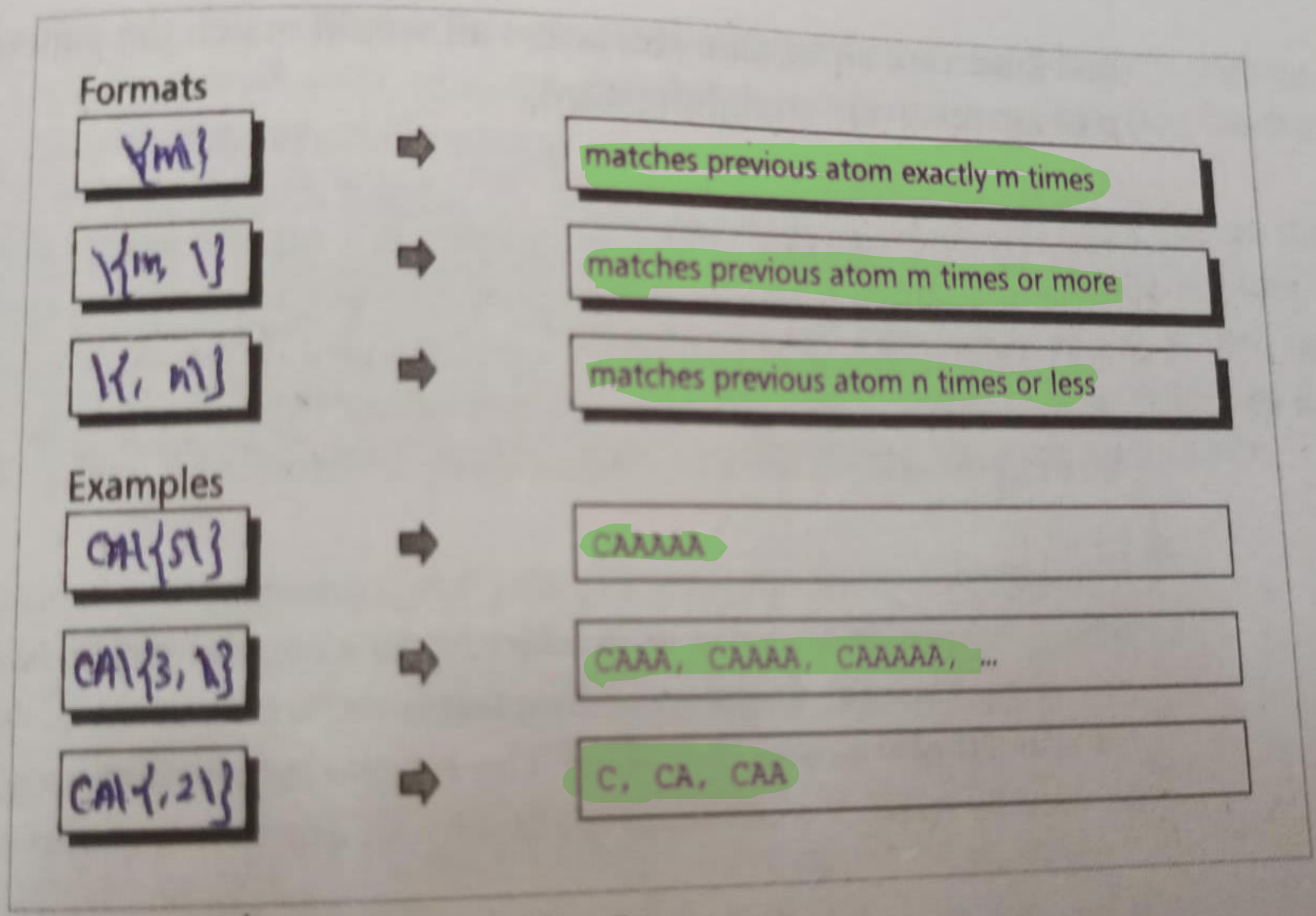


FIGURE 9.14 *Basic Repetition Forms*

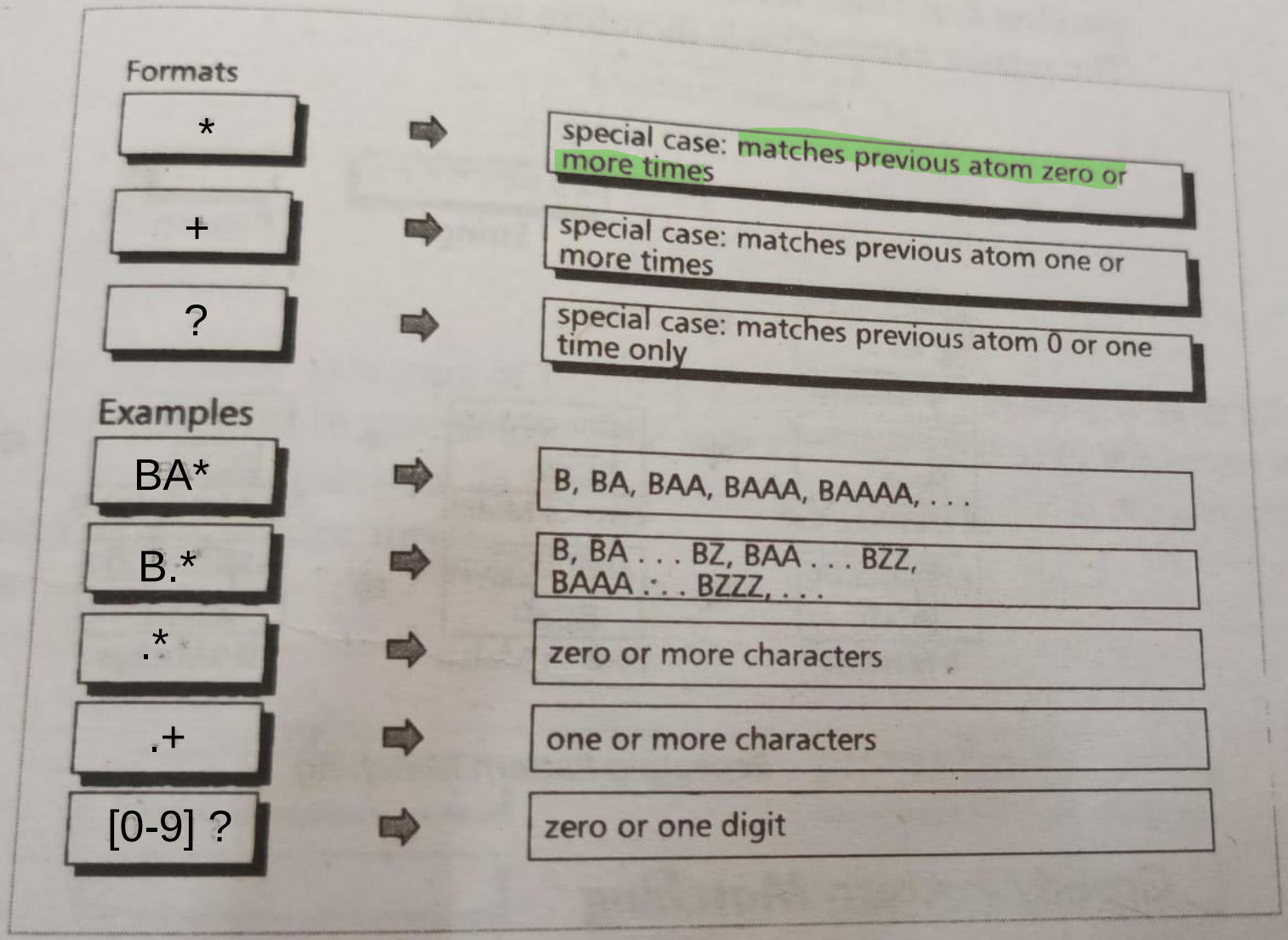


FIGURE 9.15 *Examples of Short Form Repetition Operators*

Group Operator

Group Operator

The **group operator** is a pair of opening and closing parentheses. When a group of characters is enclosed in parentheses, the next operator applies to the whole group, not only to the previous character. In the first example in Figure 9.18, the group (BC) must be repeated exactly three times.

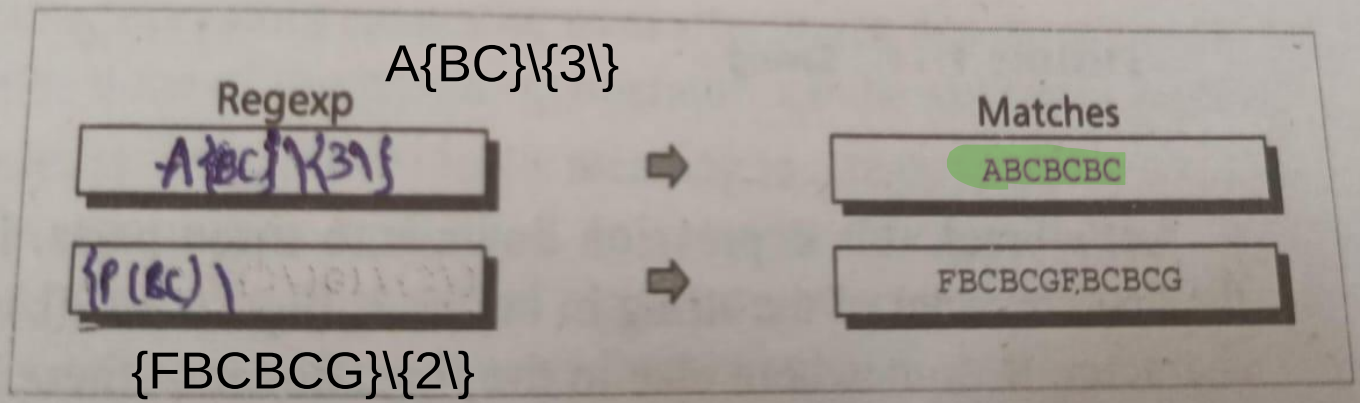


FIGURE 9.18 *Group Operator*

Save Operator

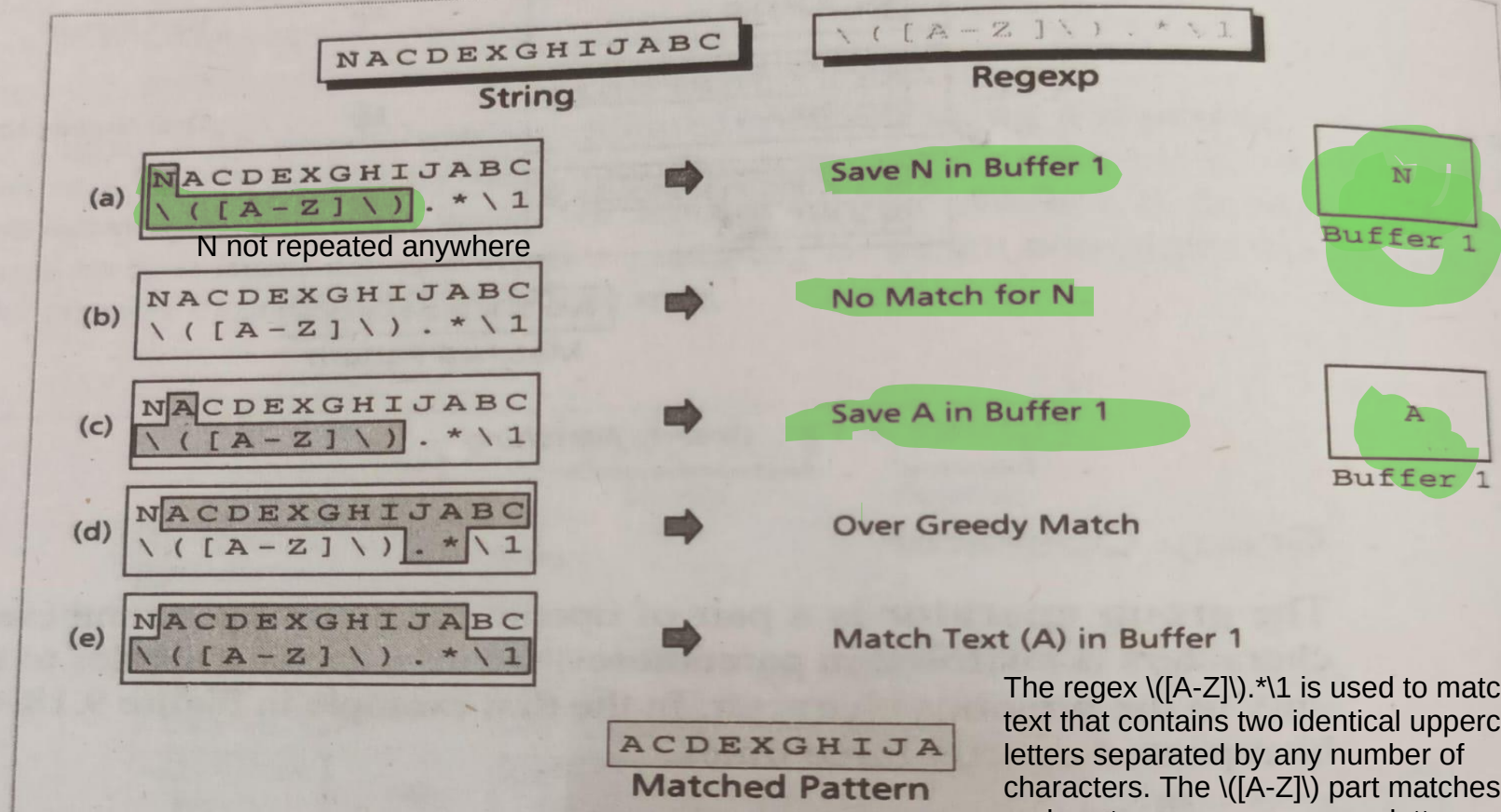


FIGURE 9.19 *Saving*

The regex \([A-Z]\).*\1 is used to match a text that contains two identical uppercase letters separated by any number of characters. The \([A-Z]\) part matches and captures one uppercase letter, and the \1 part matches the same letter that was captured. The .* part matches any number of characters in between.

GREP

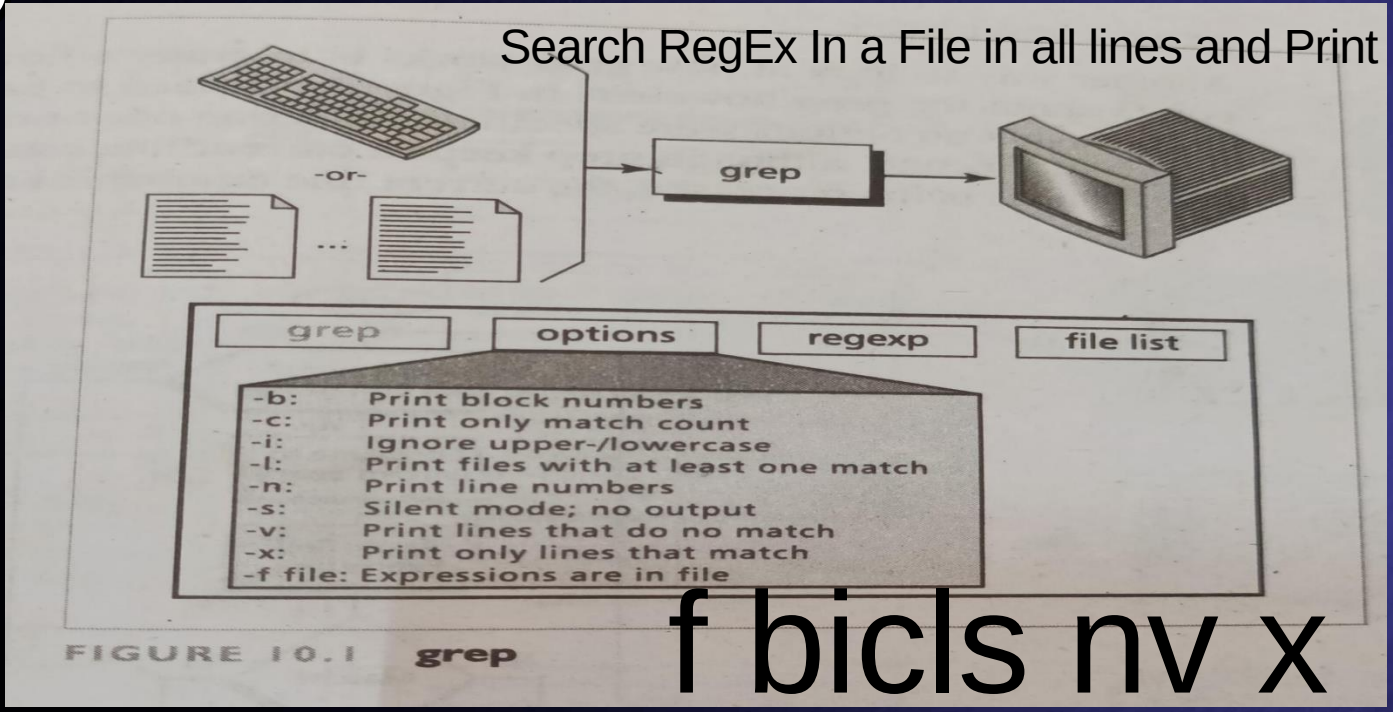
1. Operation
    - grep Flowchart
    - grep Operation Example
2. grep Family
    - grep
3. Examples
4. Searching for File Content

Introduction

GLOBAL REGEX PRINT

grep stands for **global regular expression print.**

It is family of programs that is used to search the input file for all lines that match a specified regular expression and write them to the standard output file (monitor)



Operation

For each line in the standard input(keyboard), grep performs the following operations:

1. Copies the next input line into the pattern space. The pattern space is a buffer that can hold only one text line.
2. Applies the regular expression to the pattern space.
3. If there is a match, the line is copied from the pattern space to the standard output.

The grep utilities repeat these 3 operations on each line in the input.

Operation

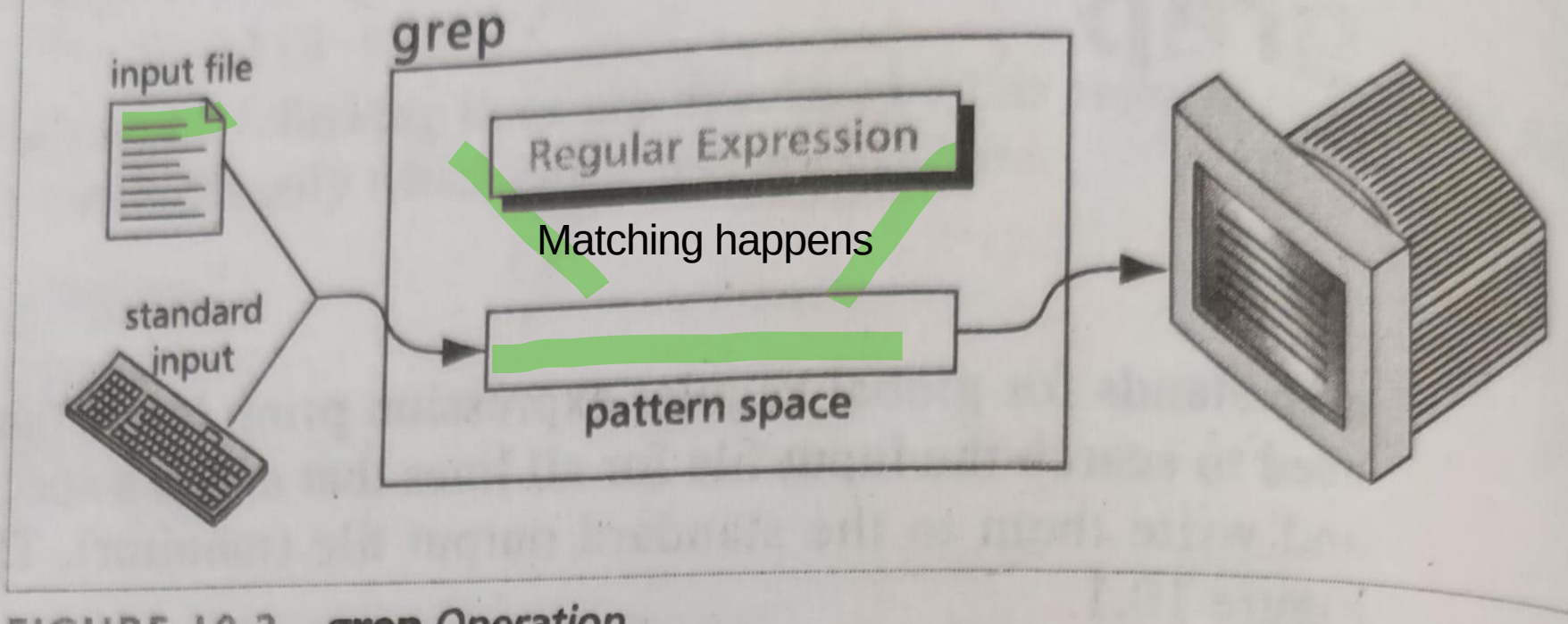


FIGURE 10.2 **grep** *Operation*

Grep flowchart

To know how grep works study flowchart of its operation.

1. Flowchart assumes that no options were specified. (selecting one or more options will change the flowchart)
2. grep keeps a current line counter so that it always knows which line is being processed, the current line number is not reflected in the flowchart.

Grep flowchart

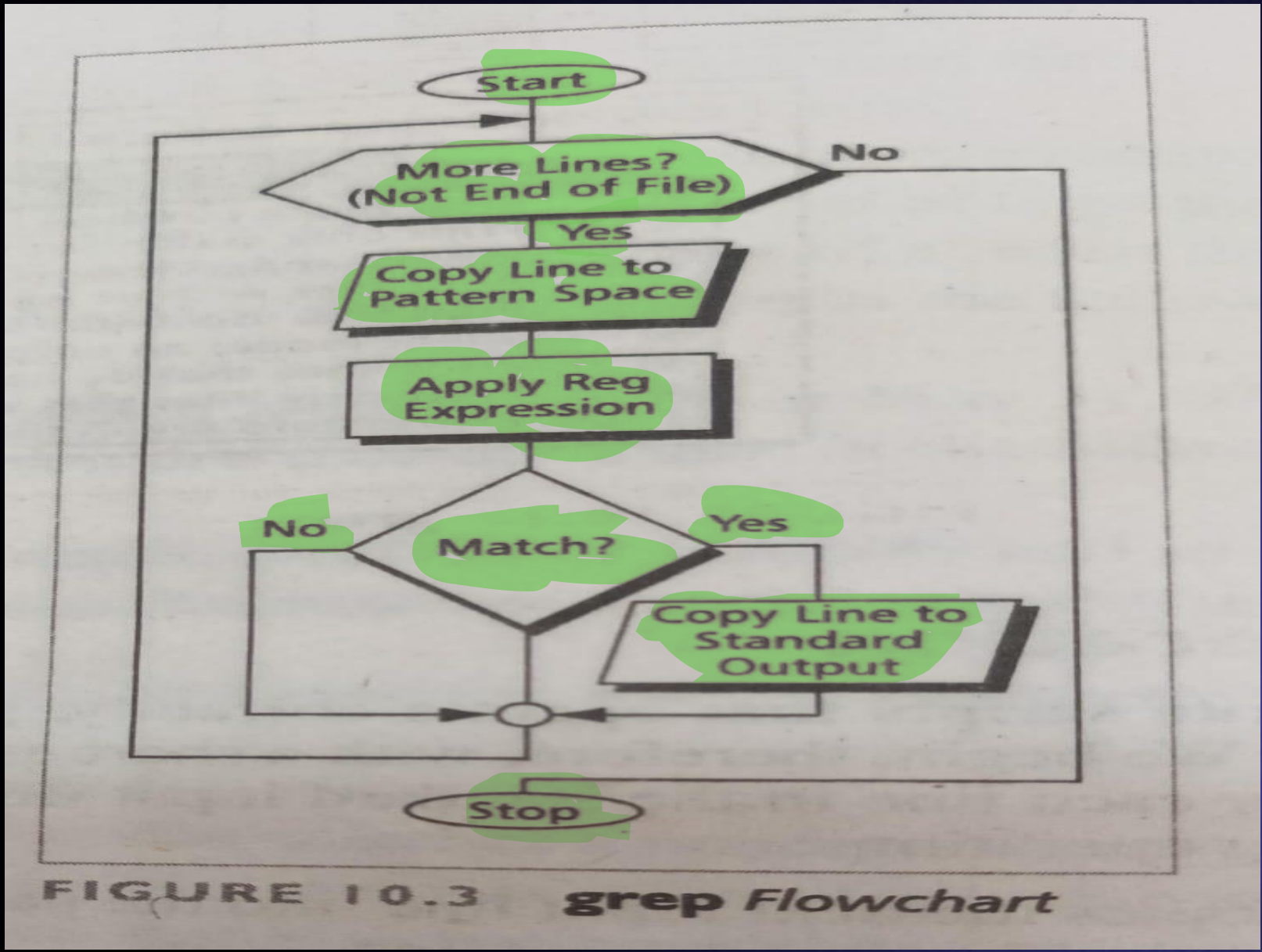


FIGURE 10.3 **grep** *Flowchart*

GREP OPERATION EXAMPLE

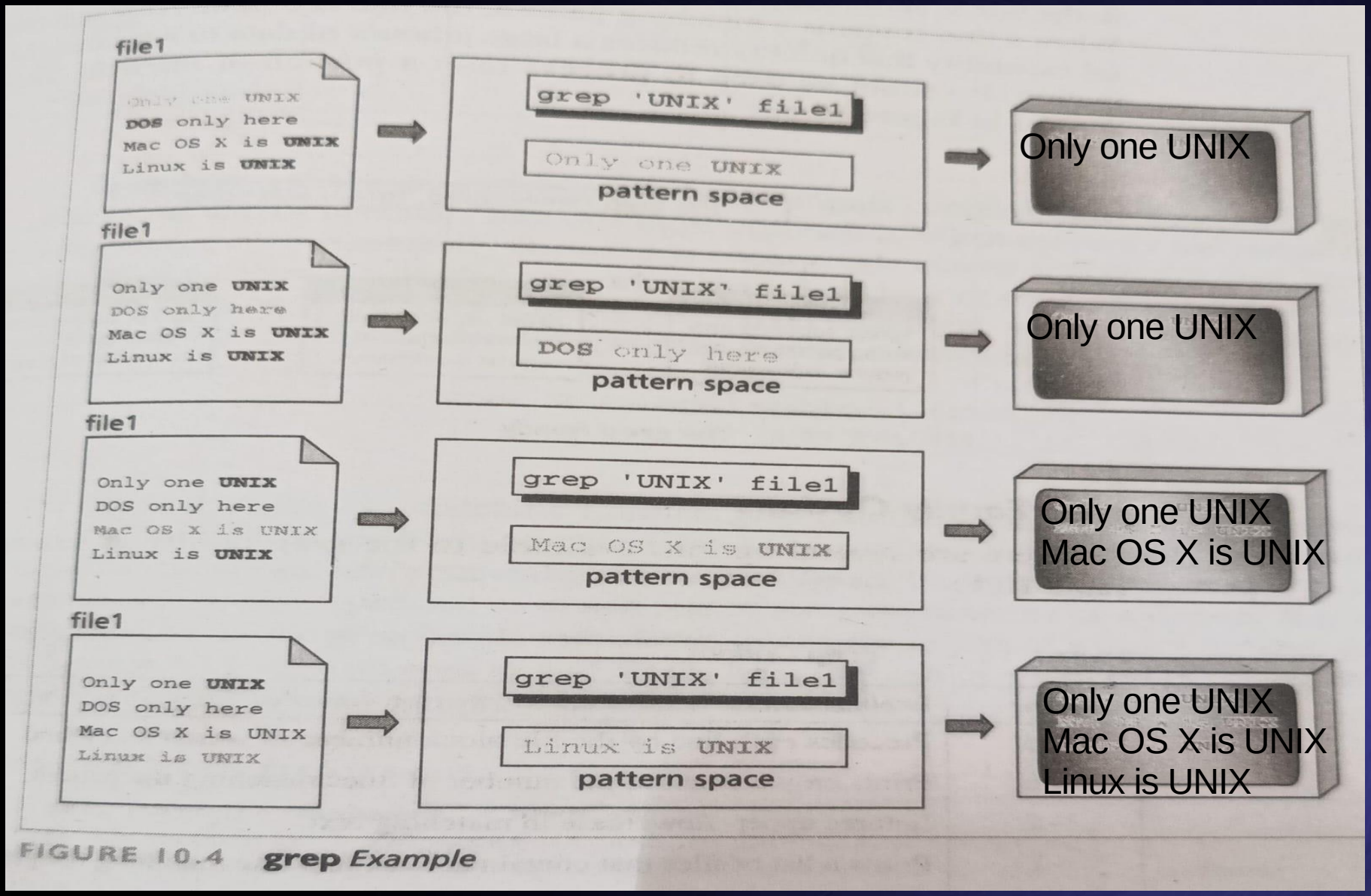


FIGURE 10.4 **grep** *Example*

Grep operation example

grep handles the following situations:

1. grep is a search utility; it can search only for the existence of a line that matches a regular expression.
2. The only action that grep can perform on a line is to send it to standard output. If the line does not match the regular expression, it is not printed.
3. The line selection is based only on the regular expression. The line number or other criteria cannot to used to select the line.
4. grep is a filter. It can be used at the left- or right-hand side of a pipe.

Grep operation example

grep limitations

1. grep cannot be used to add, delete, or change a line.
2. grep cannot be used to print only part of a line.
3. grep cannot read only part of file.
4. grep cannot select a line based on the contents of the previous or the next line. There is only one buffer, and it holds only the current line.

Grep family

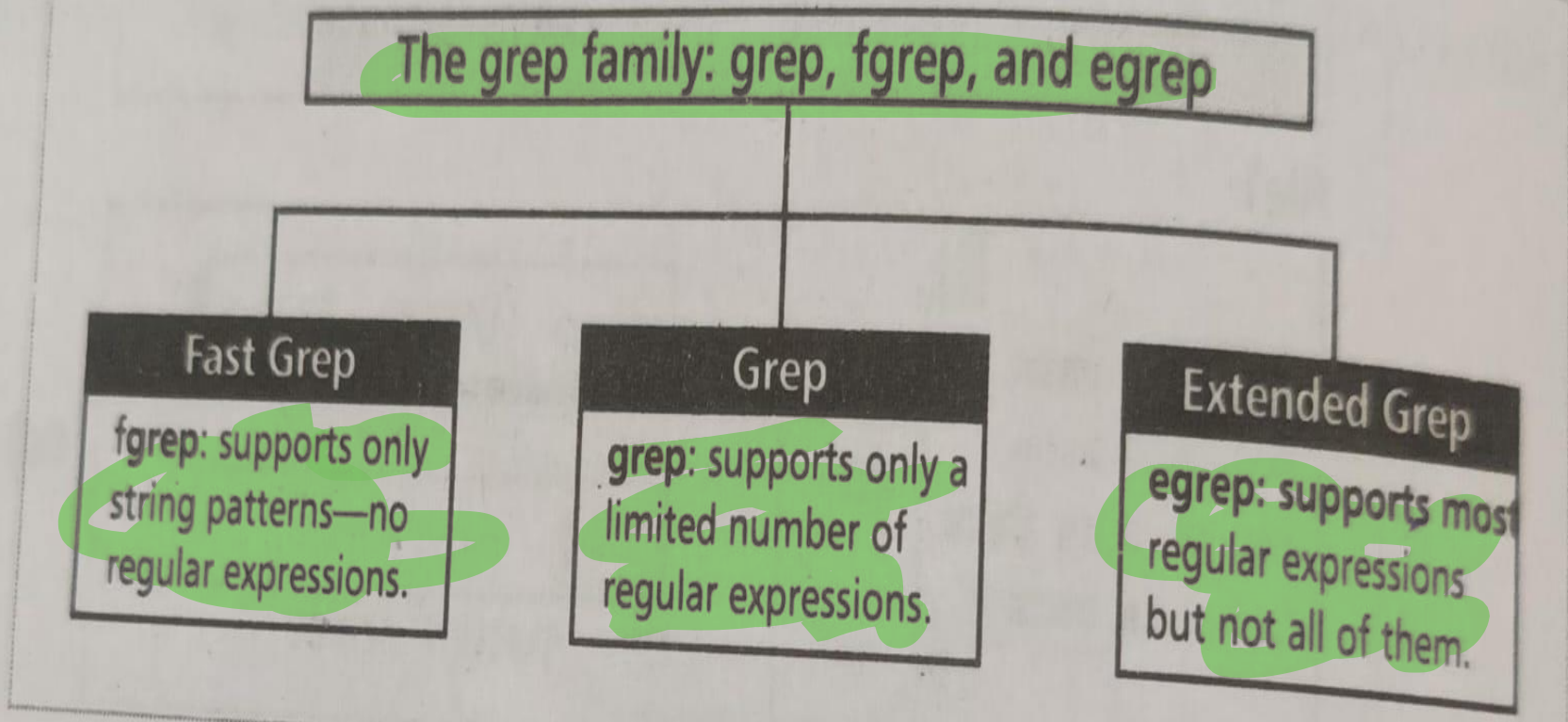


FIGURE 10.5 *The* **grep** *Family*

Grep family

TABLE 10.1 **grep** *Options*

f bicls nv x

Option	Explanation
`-b`	Precedes each line by the file block number in which it is found.
`-c`	Prints only a count of the number of lines matching the pattern.
`-i`	Ignores upper- /lowercase in matching text.
`-l`	Prints a list of files that contain at least one line matching the pattern.
`-n`	Shows line number of each line before the line.
`-s`	Silent mode. Executes utility but suppresses all output.
`-v`	Inverse output. Prints lines that do not match pattern.
`-x`	Prints only lines that entirely match pattern.
`-f file`	List of strings to be matched are in file.

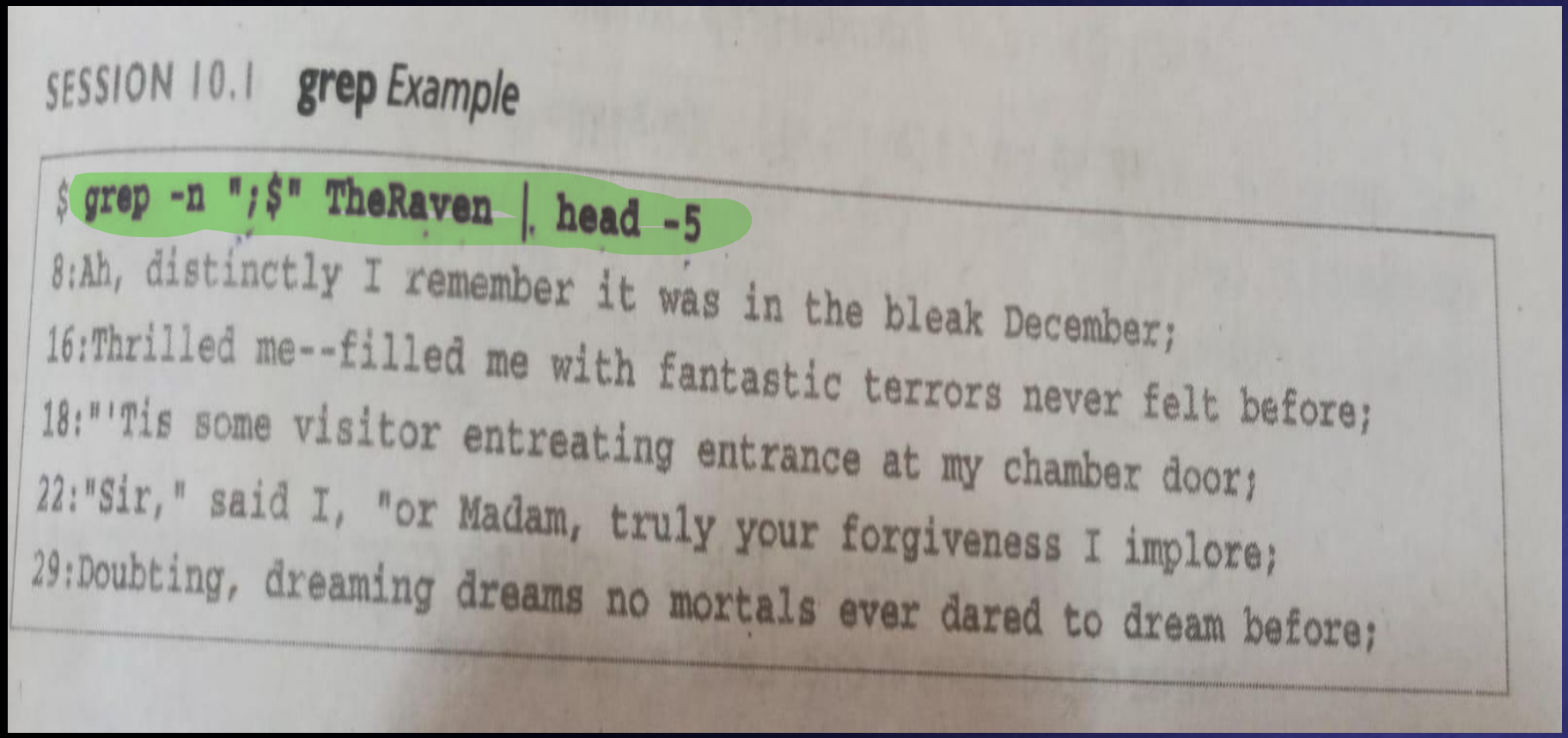
Grep family

TABLE 10.2 *Regular Expressions in the* **grep** *Family*

Atoms	grep	fgrep	egrep	Operators	grep	fgrep	egrep
Character	✓	✓	✓	Sequence	✓	✓	✓
Dot	✓		✓	Repetition	all but ?		* ? +
Class	✓		✓	Alternation			✓
Anchors	✓		^ $	Group			✓
Back Reference	✓			Save	✓		

Grep Example

Use grep to find all the lines that end in a semicolon(;) and then pipe results to head and print the first five.



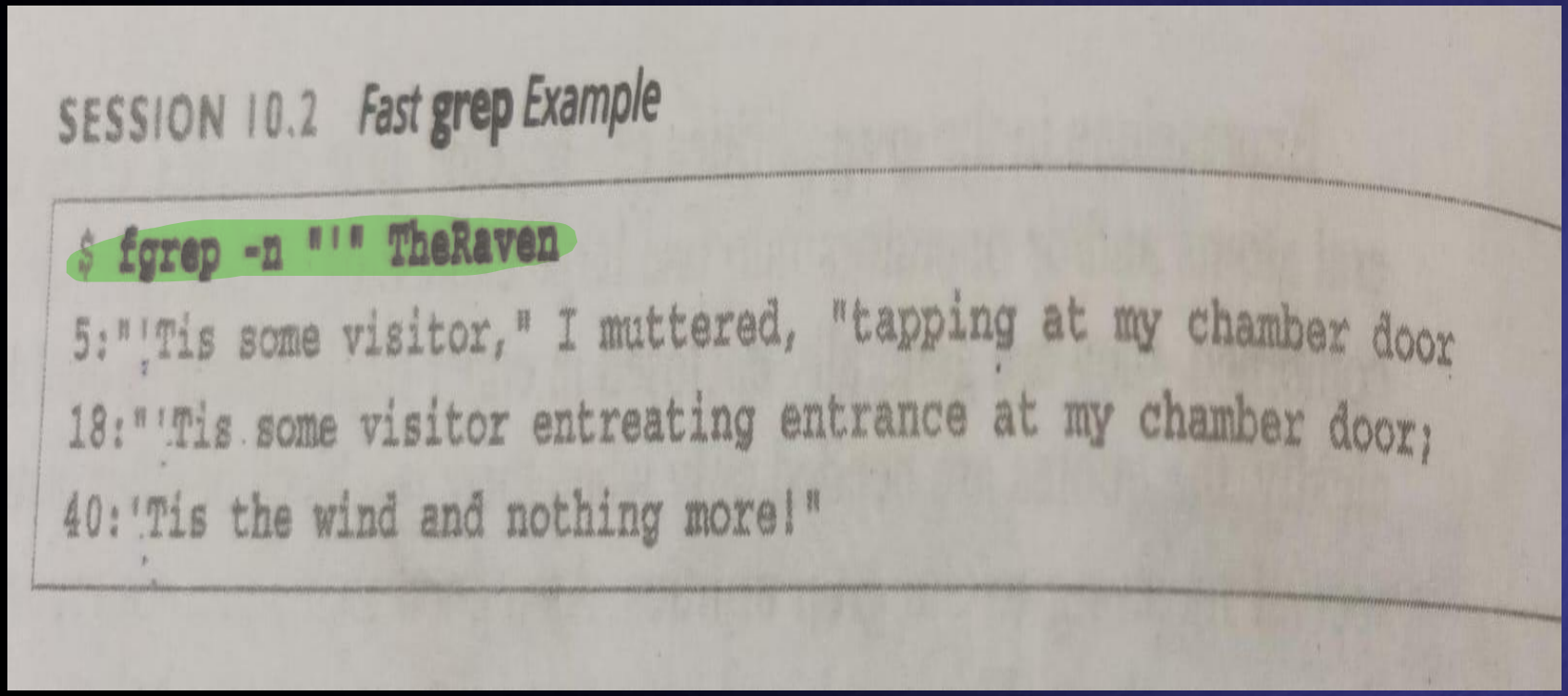
SESSION 10.1 **grep** *Example*

```
$ grep -n ";$" TheRaven | head -5
8:Ah, distinctly I remember it was in the bleak December;
16:Thrilled me--filled me with fantastic terrors never felt before;
18:"'Tis some visitor entreating entrance at my chamber door;
22:"Sir," said I, "or Madam, truly your forgiveness I implore;
29:Doubting, dreaming dreams no mortals ever dared to dream before;
```

Fast grep Example

Such as escape, parantheses, or quotes.---criteria requires only sequence expressions, fast grep(fgrep) is the best utility.

To extract all lines of “The Raven” that contain an apostrophe, we could use fgrep.



SESSION 10.2 *Fast grep Example*

```
$ fgrep -n "'" TheRaven
5:'Tis some visitor," I muttered, "tapping at my chamber door
18:'Tis some visitor entreating entrance at my chamber door;
40:'Tis the wind and nothing more!"
```

Extended grep Example

It is most powerful of the 3 grep utilities. It doesn't have the save option, it does allow more complex patterns.

Consider the case where we want to extract all lines that start with a capital letter and end in an exclamation point(!)

SESSION 10.3 *Extended* **grep** *Example*

```
$ egrep -n '^[A-Z].*!$' TheRaven
82:She shall press, ah, nevermore!
107:Leave no black plume as a token of that lie thy soul hath spoken!
116:Shall be lifted nevermore!
```

Searching for filecontent

- Forgotten the filename but know that it contains a specific expression or set of words.
- grep family used to accomplish the same thing.
- When we know the directory that contains the file. Example.

SESSION 10.7 Search for Text in Current Directory

```
$ ls
RavenII      TheRaven     man.ed       regexp.dat
$ grep -l 'Raven' *
RavenII
TheRaven
```

Searching for filecontent

- When we don't know the file located, we must use the find command Example.
- Home directory(~)

SESSION 10.8 *Using* **find** *and* **grep** *to Search All Directories*

```
$ find ~ -type f -exec grep -l "Raven" {} \;
/mnt/staff/gilberg/unix10grep/RavenII
/mnt/staff/gilberg/unix10grep/TheRaven
/mnt/staff/gilberg/unix6filters/TheRaven
/mnt/staff/gilberg/unix7comm/TheRaven
/mnt/staff/gilberg/unix11sed/TheRaven
/mnt/staff/gilberg/unix11sed/readScript
/mnt/staff/gilberg/unix11sed/change.sed
/mnt/staff/gilberg/unix11sed/raven.txt
/mnt/staff/gilberg/unix8vi/TheRaven
/mnt/staff/gilberg/unix5shells/TheRaven
/mnt/staff/gilberg/unix13/TheRaven
/mnt/staff/gilberg/unix3/TheRaven
/mnt/staff/gilberg/TheRaven
/mnt/staff/gilberg/unix9filters/TheRaven
/mnt/staff/gilberg/TheRavenV1
/mnt/staff/gilberg/unix12awk/TheRaven
/mnt/staff/gilberg/unix3files/TheRaven
/mnt/staff/gilberg/unix3files/TheRavenV1
/mnt/staff/gilberg/unix13bourne/TheRaven
/mnt/staff/gilberg/unix13bourne/midRaven.scr
/mnt/staff/gilberg/unix13bourne/midRaven.txt
$
```

SED

- Sed is an acronym for stream editor.Although the name implies editing. It is not a true editor; it does not change anything in the original file.
- Rather sed scans the input file, line by line, and applies a list of instructions (called a sed script) to each line in the input file.
- The script, which is usually a separate file, can be included in the sed command line if it is a one-line command.

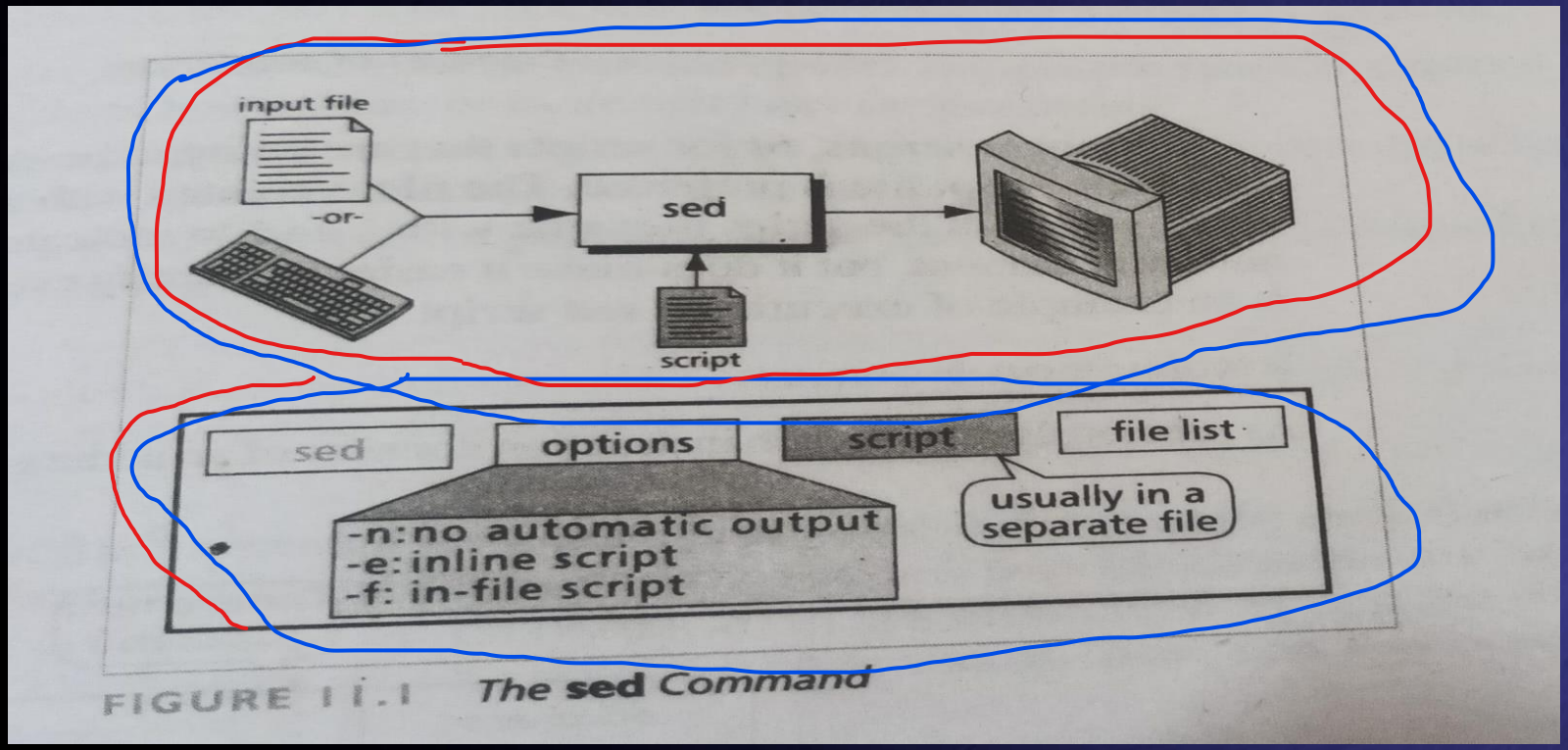
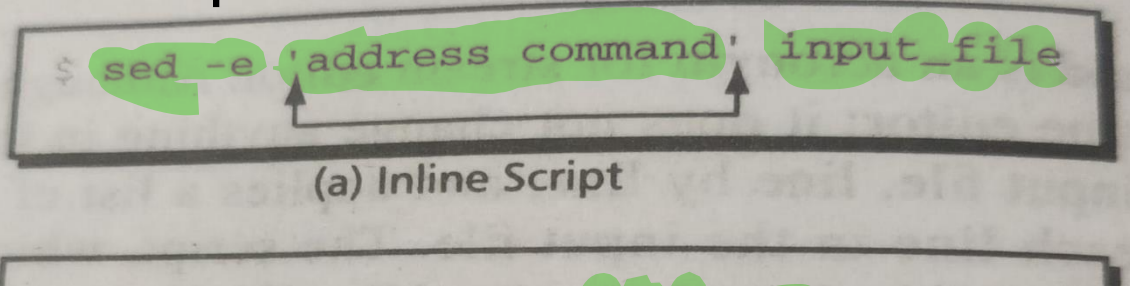


FIGURE 11.1 *The* **sed** *Command*

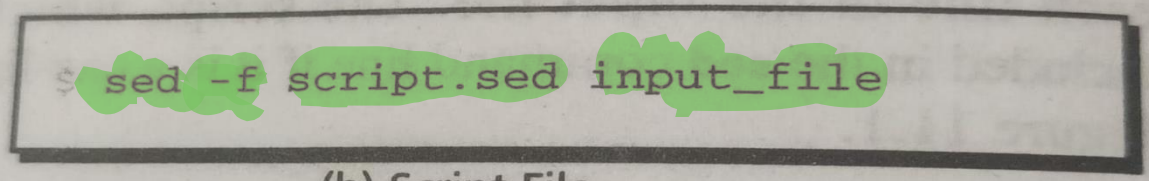
Scripts

- In addition to input data, sed also requires one or more instructions that provide editing criteria.
- When there is only one command, it may be entered from the keyboard.
- Instructions are placed in a file known as sed script (program).
- Each instruction in a sed script contains an address and command.
- Script formats

sed opt "address command" file1



(a) Inline Script



(b) Script File

sed opt script.sed file1

FIGURE 11.2 *Examples of* **sed** *Scripts*

Scripts

- Instruction format
- Instruction contains address and a command.

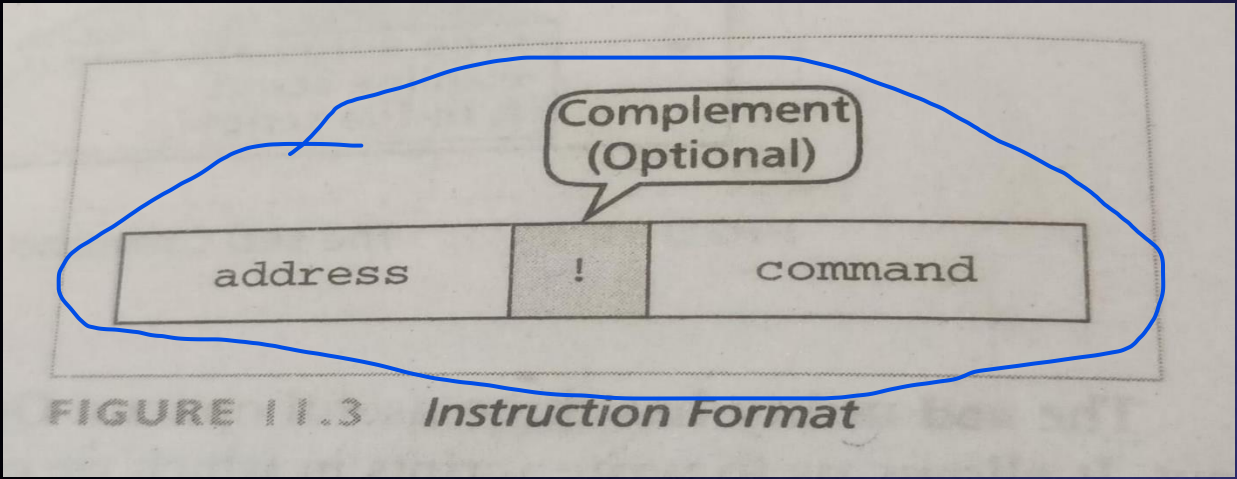
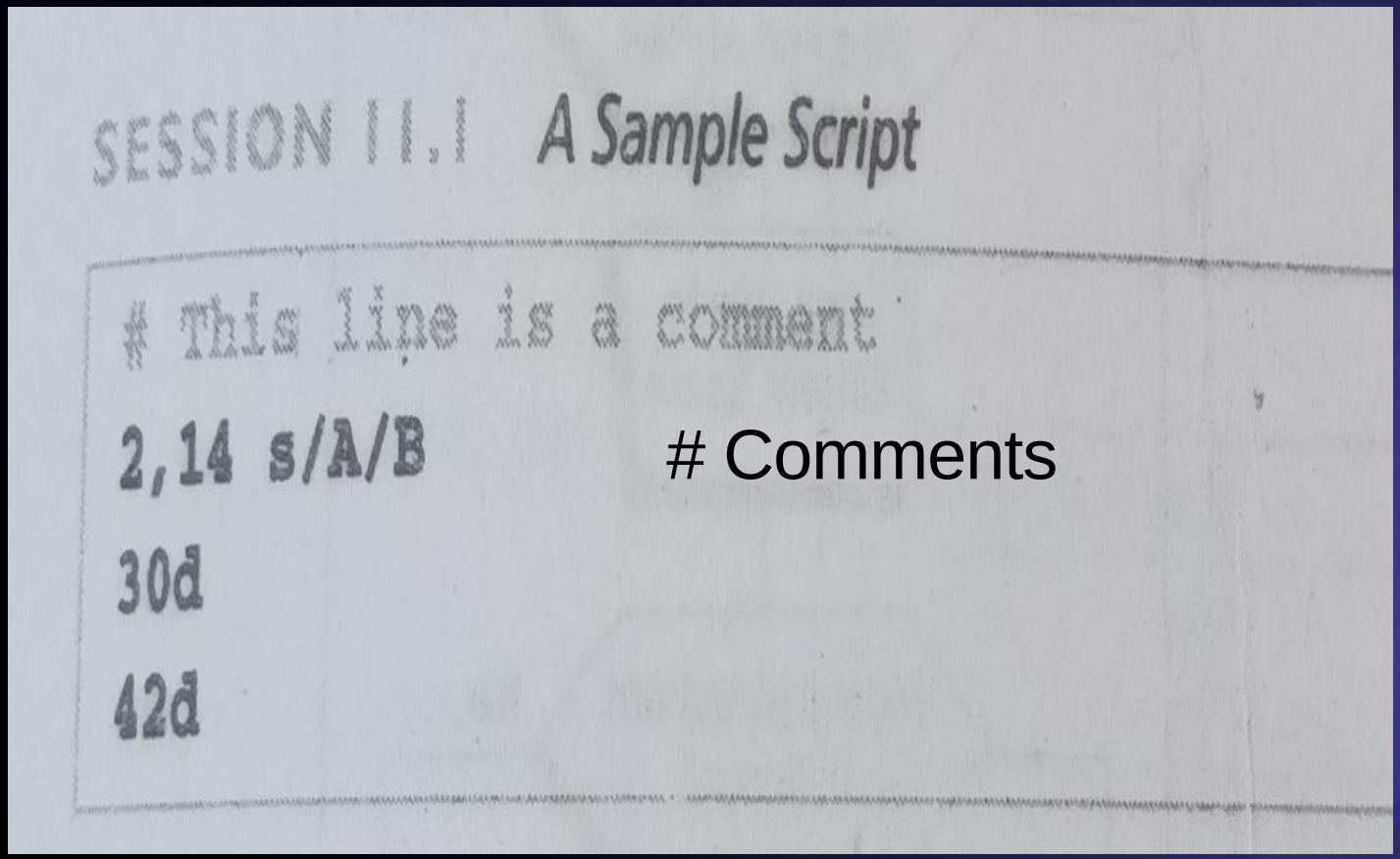


FIGURE 11.3 *Instruction Format*

- The address selects line to be processed(or not processed) by the command.
- The exclamation point (!) is an optional address complement. When it is not present, the address must exactly match a line to select the line.
- If present, any line that does not match the address is selected; lines that match the address are skipped
- The command indicates the action that sed is to apply to each input line that matches the address.

Scripts

- Comments
- A comment is a script line that documents or explains one or more instructions in a scipt.
- It is provided to assist the reader and is ignored by sed.



Operation

Following operations

1. Copies an input line to the pattern space. The pattern space is a specified buffer capable of holding one or more text lines for processing

2. Applies all the instructions in the scripts, one by one , to all patterns space lines that match the specified addresses in the instruction.

3. Copies the contents of the pattern space to the output file unless directed not to by the -n option flag.

Operation

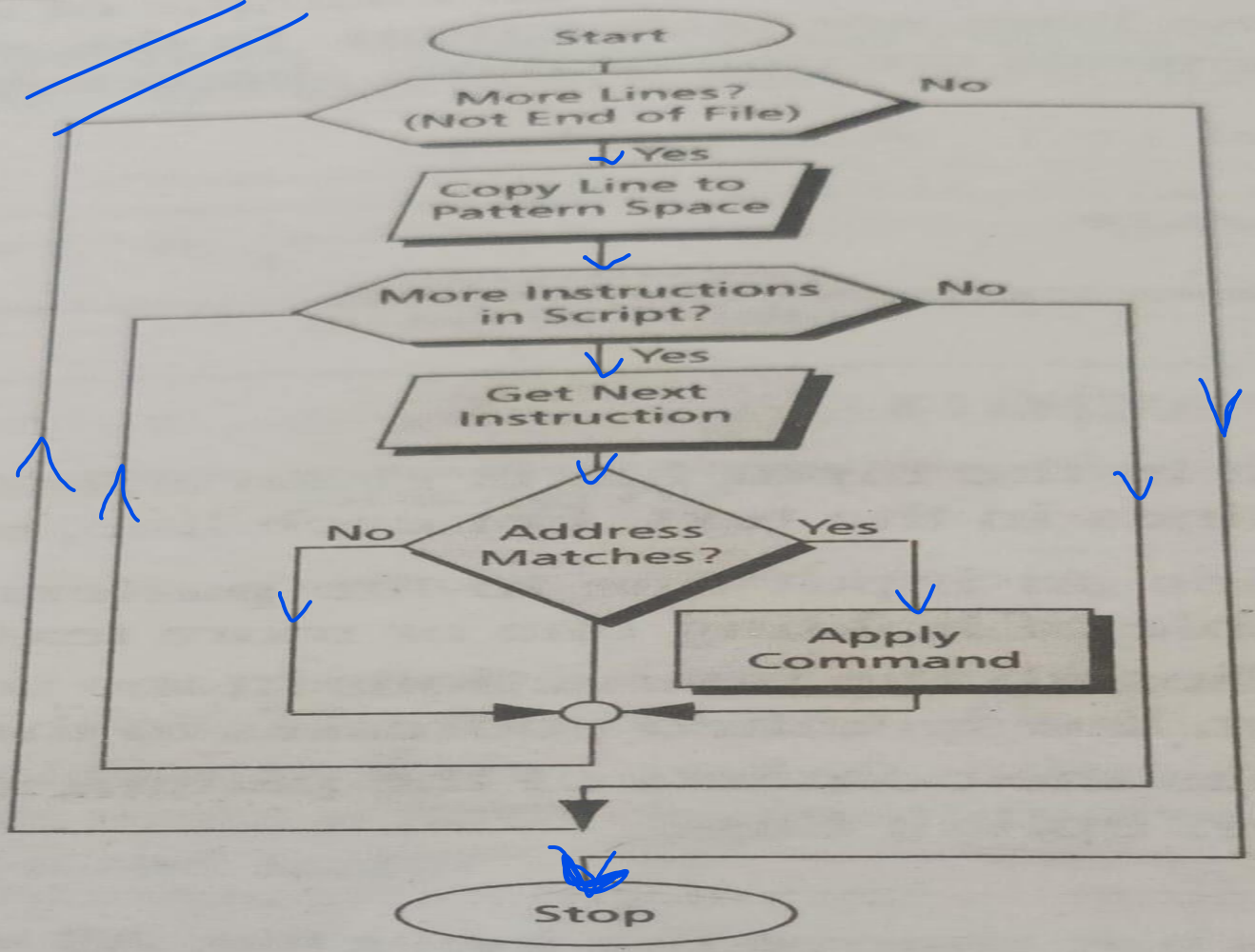


FIGURE 11.4 **sed**'s Processing Flow

Operation

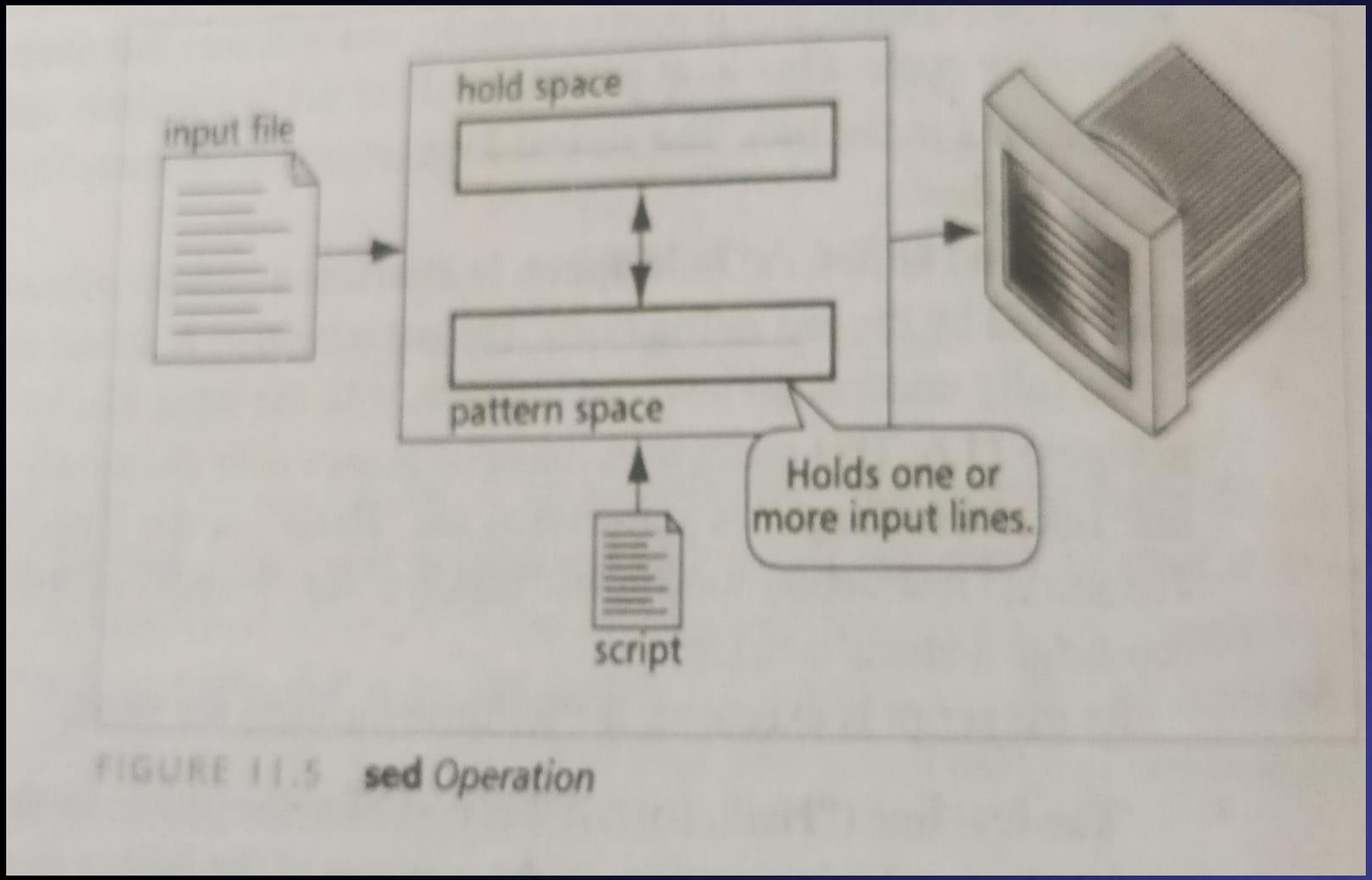


FIGURE 11.5 **sed** *Operation*

Operation

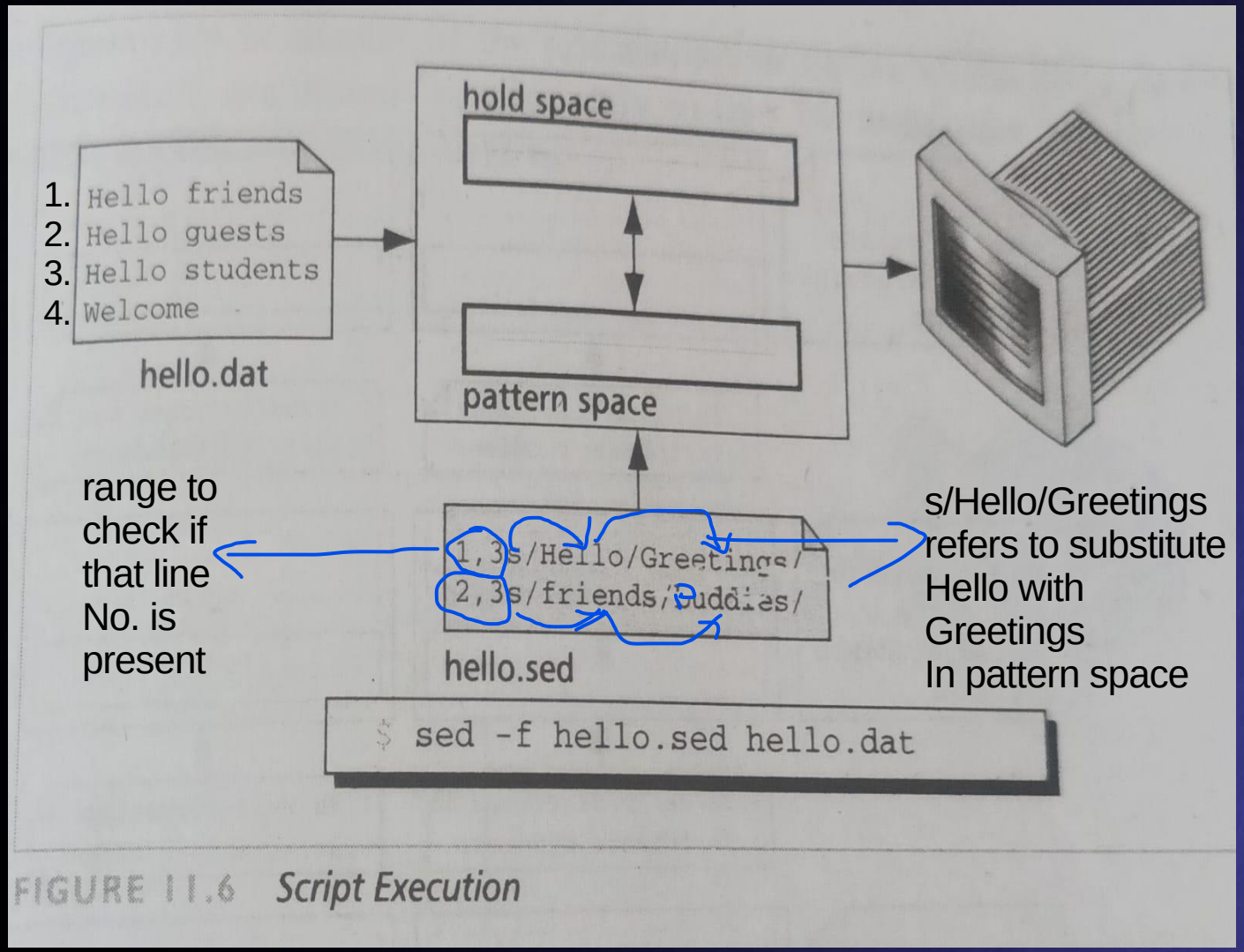


FIGURE 11.6 *Script Execution*

Operation

As the script is executed, the following actions are taken:

1. The first line ("Hello friends") is copied to the pattern space. The script is applied, instruction by instruction, to the contents of the pattern space.

a. Script line 1: The address (1-3) matches the input line (1), so the instruction is applied. The pattern space now contains "Greetings friends."

b. Script line 2: The address (2 or 3) does not match, so this instruction is not applied.

c. End of script: The pattern space, "Greetings friends," is sent to the monitor.

Operation

2. The second line ("Hello guests") is copied to the pattern space, replacing its contents. The script is applied, instruction by instruction, to the contents of the pattern space.

a. Script line 1: The address (1-3) matches the input line (2), so the instruction is applied. The pattern space now contains "Greetings guests."

b. Script line 2: The address (2 or 3) matches the input line (2), so this instruc- tion is applied. However, because there is no "friends" in the line, the pattern space is unchanged; it still contains "Greetings guests."

c. End of script: The pattern space is sent to the monitor.

Operation

3. The third line ("Hello students") is copied to the pattern space, replacing its contents. The script is applied, instruction by instruction, to the contents of the pattern space.

a. Script line 1: The address (1-3) matches the input line (3), so the instruction is applied. The pattern space now contains "Greetings students."

b. Script line 2: The address (2 or 3) matches the input line (3), so this instruc- tion is applied. However, because there is no "friends" in the line, the pattern space is unchanged; it still contains "Greetings students."

c. End of script: The pattern space is sent to the monitor.

Operation

4. The fourth line ("Welcome") is copied to the pattern space. The script is applied, instruction by instruction, to the contents of the pattern space.

a. Script line 1: The address (1-3) does not match the input line (4), so the instruction is not applied.

b. Script line 2: The address (2 or 3) does not match the input line (4), so this instruction is not applied.

c. End of script. The unchanged pattern space is printed.

Figure 11.7 shows the script execution instruction by instruction.

Interesting Piece Of Shit.....

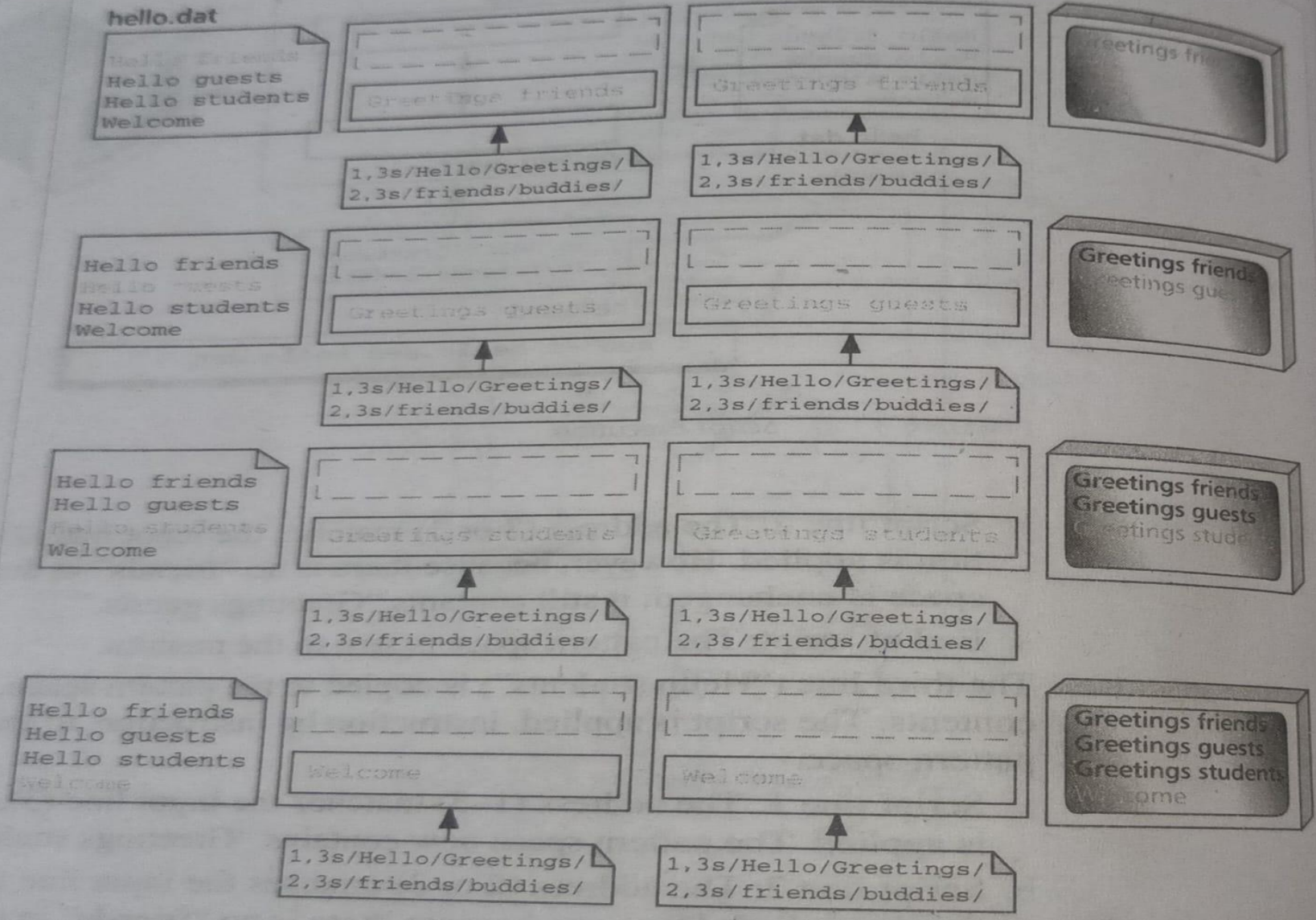


FIGURE 11.7 *Trace Script Execution*

Addresses

- The address in an instruction determines which lines in the input file are to be processed by the command in the instruction.

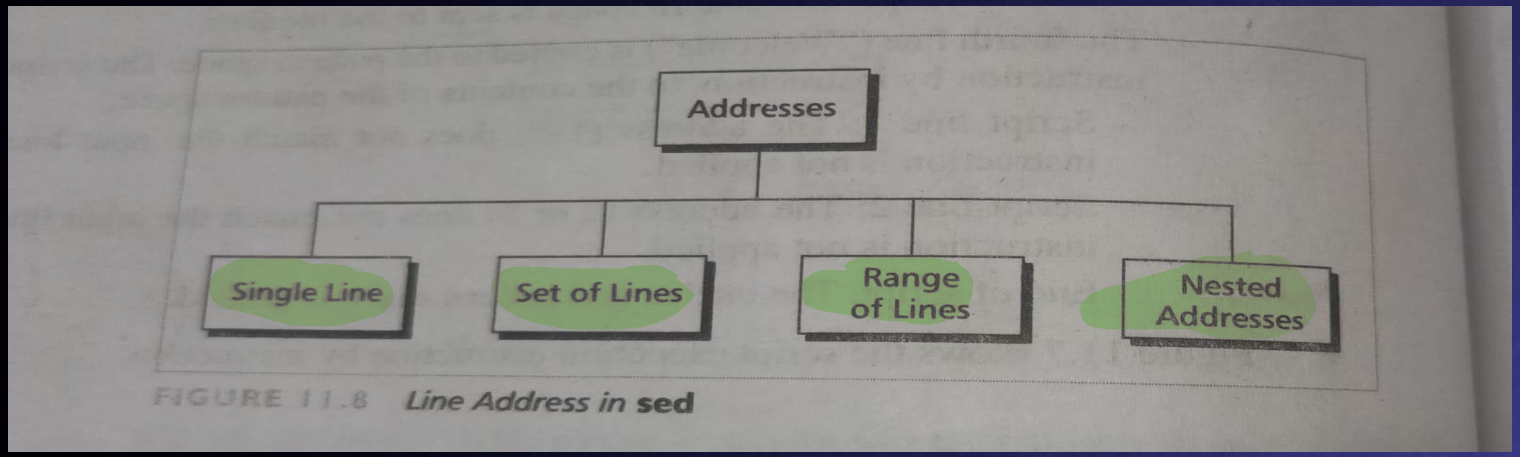


FIGURE 11.8 *Line Address in* **sed**

- Single Line Addresses
- Set-of-Line Addresses
- Range Addresses
- Nested Addresses

Single Line Addresses

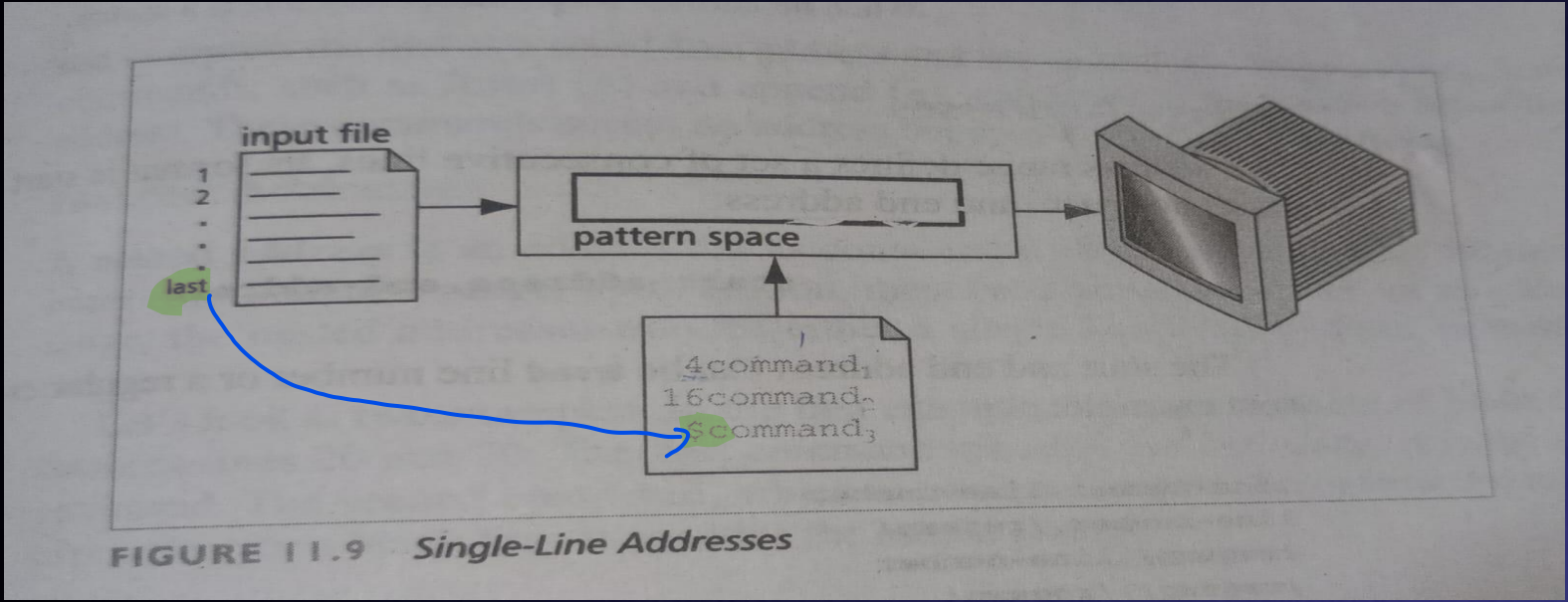


FIGURE 11.9 *Single-Line Addresses*

A Single-Line address specifies one and only line in the input file.

There are two single-line formats:

1. A line number
2. a dollar sign($)

Set of line addresses

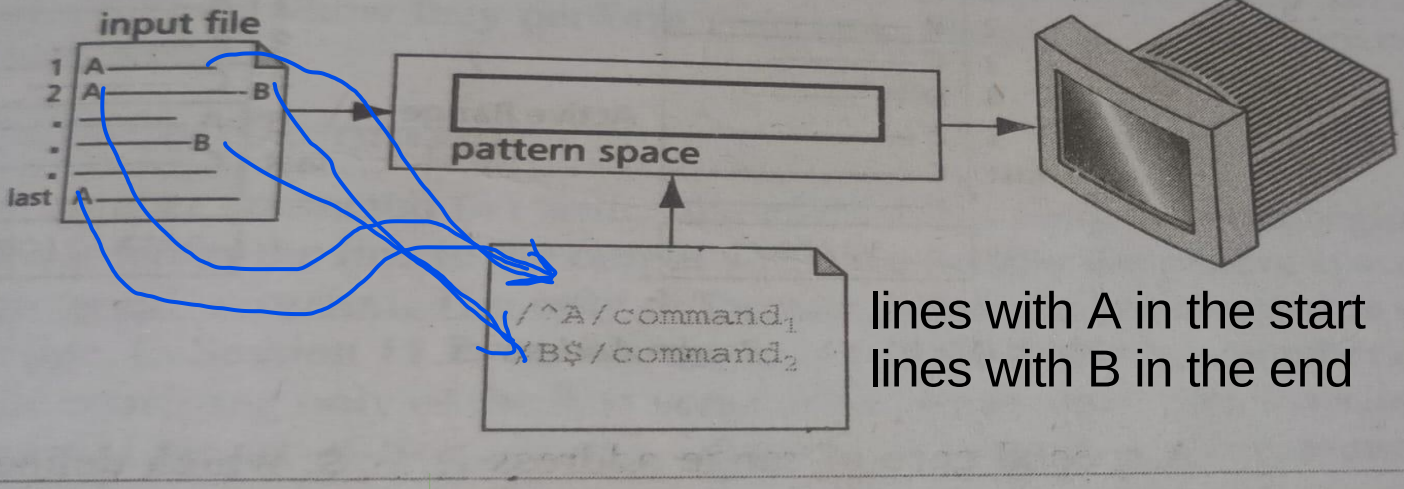


FIGURE 11.10 *Set-of-Line Addresses*

A Set-of-line address is a regular expression that may match zero or more lines, not necessarily consecutive, in the input file.

Range addresses

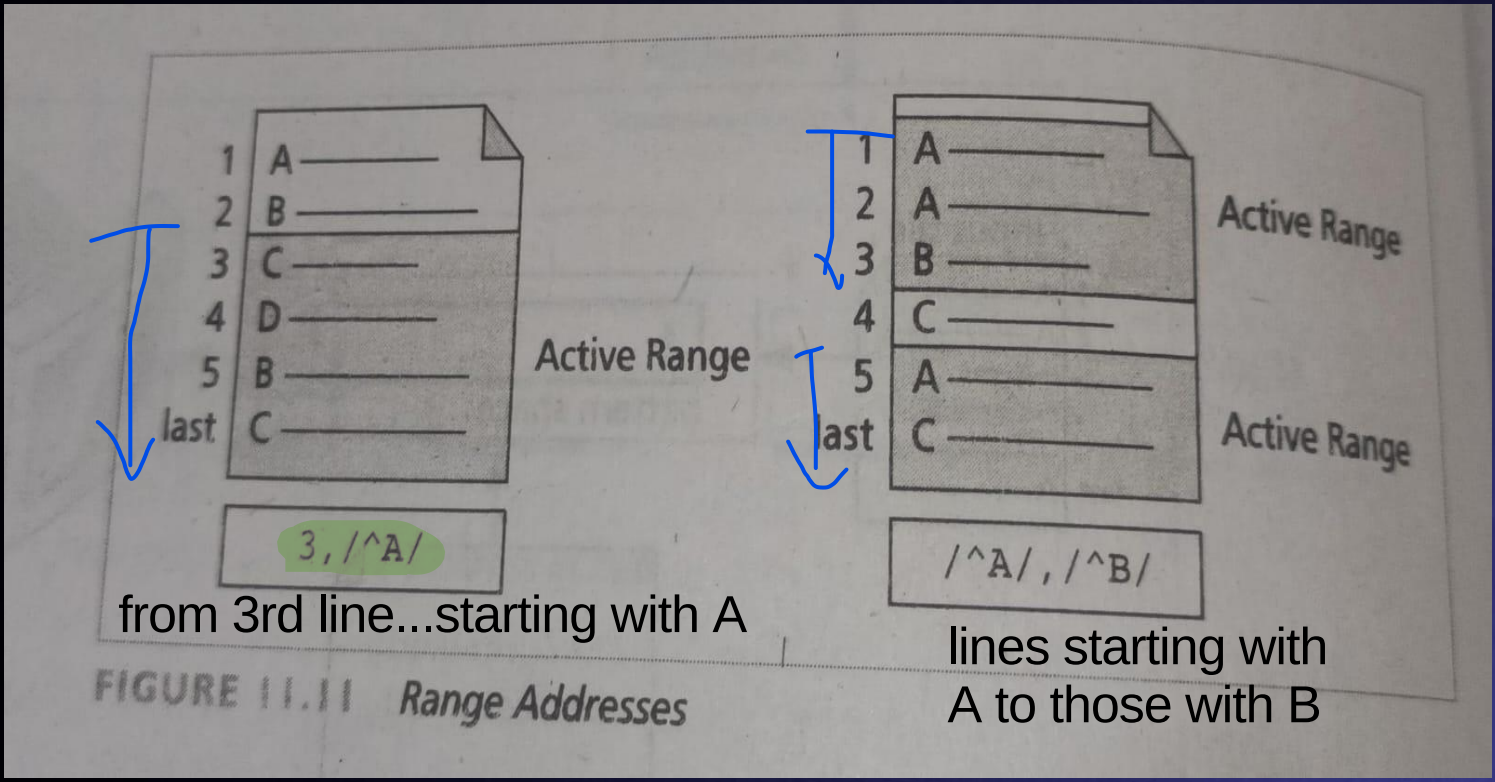


FIGURE 11.11 *Range Addresses*

An address range defines a set of consecutive lines.

Its format is start address, comma with no space ,and address

Start-address,end-address

Nested addresses

A nested address is an address that is contained within another address.

- The outer (first) address range, by definition, must be either a set of lines or an address range, the nested addresses may be either a single line, a set of lines or another range.

Example:

Questions: Delete all blank lines between lines 20 and 30.

- Solution: first command specifies the line ranges; it is outer command. The second command, which is enclosed in braces,contains the regular expression for a blank line.
- It contains the nested address.

```
20,30 {          deletes all blank lines from 20 to 30
        /^$/d
      }
```

Commands

- Line Number Command
- Modify Commands
- Substitute Command
- Substitute Flags
- Transform Command
- Input and Output Commands
- File Commands
- Branch Commands
- Hold Space Commands
- Quit

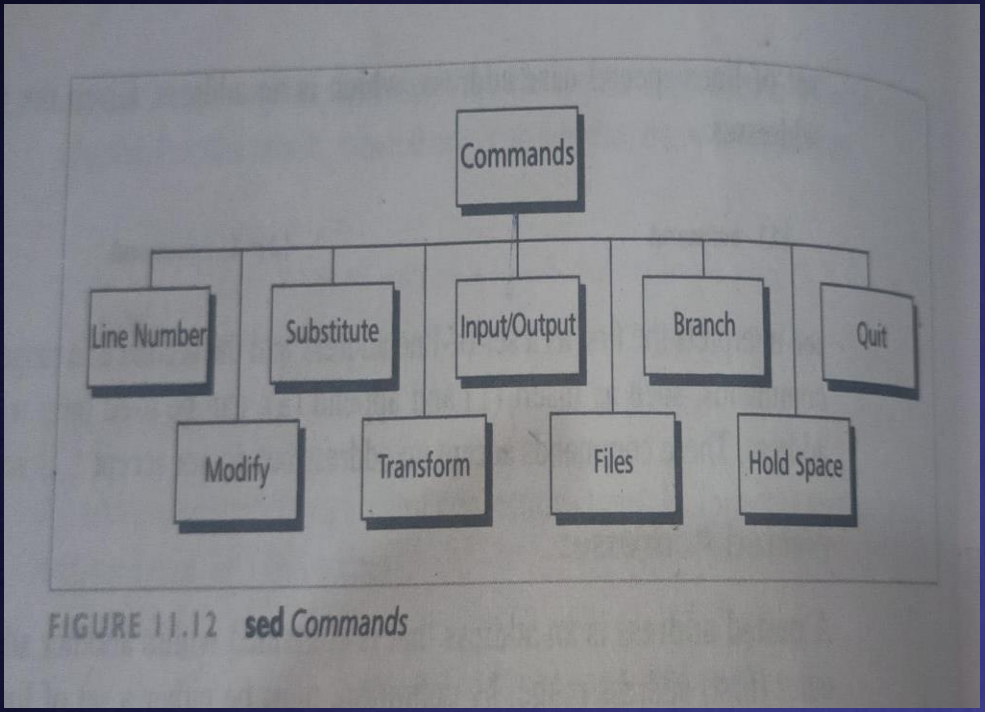


FIGURE 11.12 **sed** *Commands*

Line number command

= used to O/p line no. in the given file

sed '=' file1.txt

- The line number command (=) writes the current line number at the beginning of the line when it writes the line to the output without affecting the pattern space.

Example:

SESSION 11.2 *Display Line Numbers*

```
$ sed '=' TheRavenV1
```

Output:

```
1
Once upon a midnight dreary, while I pondered, weak and weary,
2
Over many a quaint and curious volume of forgotten lore
3
While I nodded, nearly napping; suddenly there came a tapping,
4
As of someone gently rapping, rapping at my chamber door.
5
"'Tis some visitor," I muttered, "tapping at my chamber door
6
Only this and nothing more."
```

Modify commands

sed -f (modifyOptName) Instn.sed file1

- This commands are used to inset, append,change or delete one or more whole lines.
- It require that any text associated with them by plaved on the next line in the script.

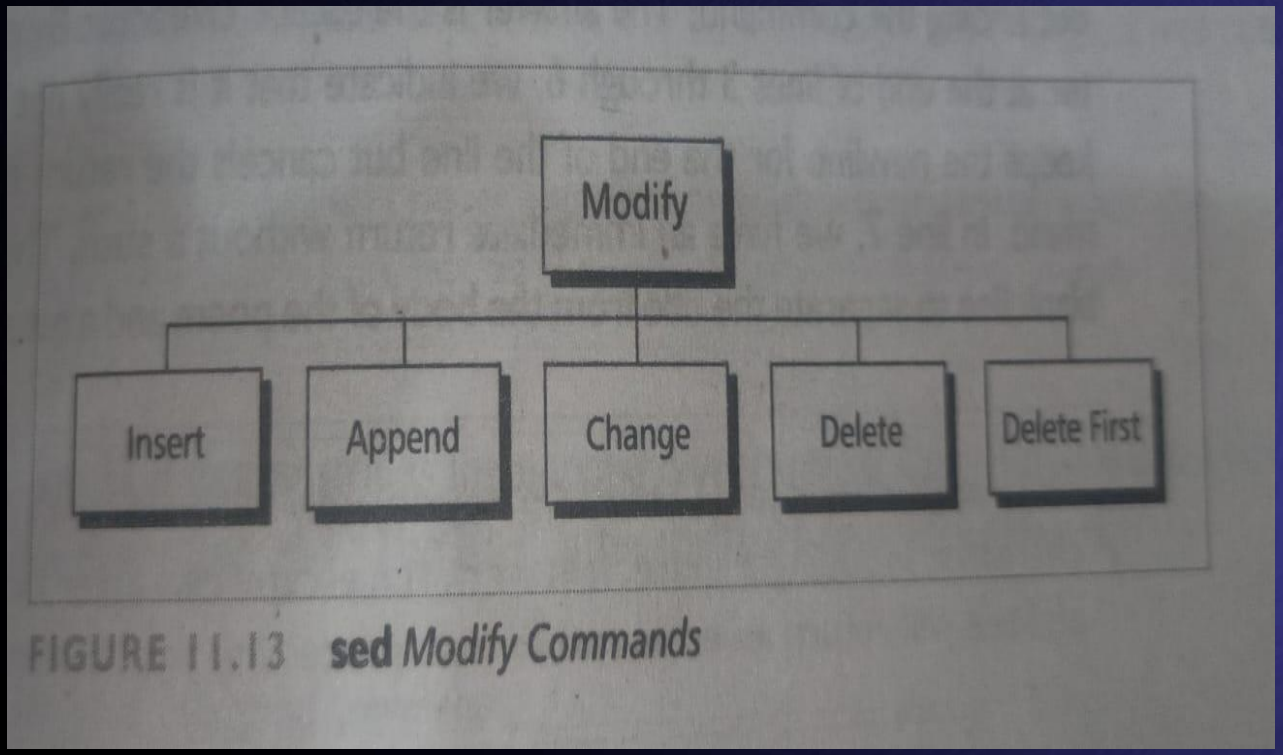


FIGURE 11.13 **sed** *Modify Commands*

Modify commands

before address

- **Insert command:**

sed -f insert Instn.sed file1

It adds one or more lines directly to the output before address.

This command can be used only with the single line and set of lines, not with the range.

Example:

Insert a title at the beginning of Poe's "The Raven"

Command:

Sed -f insertTitle.sed The RavenV1 | cat -n

Modify commands

after address

sed -f append Instn.sed file1

- **Append command:**

This command is similar to insert command except that it writes the text directly to the output after the specified line.

It cannot be used with range address.

Example:

Appending a dashed line separator after every line and “The End “ after the last line of “The Raven”

Command:

Sed -f appenLineSep.sed TheRavenV1

Modify commands

sed -f change Instn.sed file1

- **Change command:**

This command replaces a matched line with new text.

It accepts all 4 address types

Example:

Replace the second line of Po’es classic with a common thought expressed by many a weary calculus student.

Command:

Sed -f change.sed TheRavenV1

Modify commands

- **Delete Pattern Space command: (d)/Delete Only first line command(D)**

This delete command comes in two versions:

Lowercase delete command (Deletes the entire pattern space)and Upper case delete command(First line of pattern space is deleted)

Example:

Sed '/^O/d' TheRavenV1

Sed '/^O/D' TheRavenV1

Substitute command

- This command replaces text that is selected by a regular expression with a replacement string.
- Similar to search and replace a string in text file.
- Format:

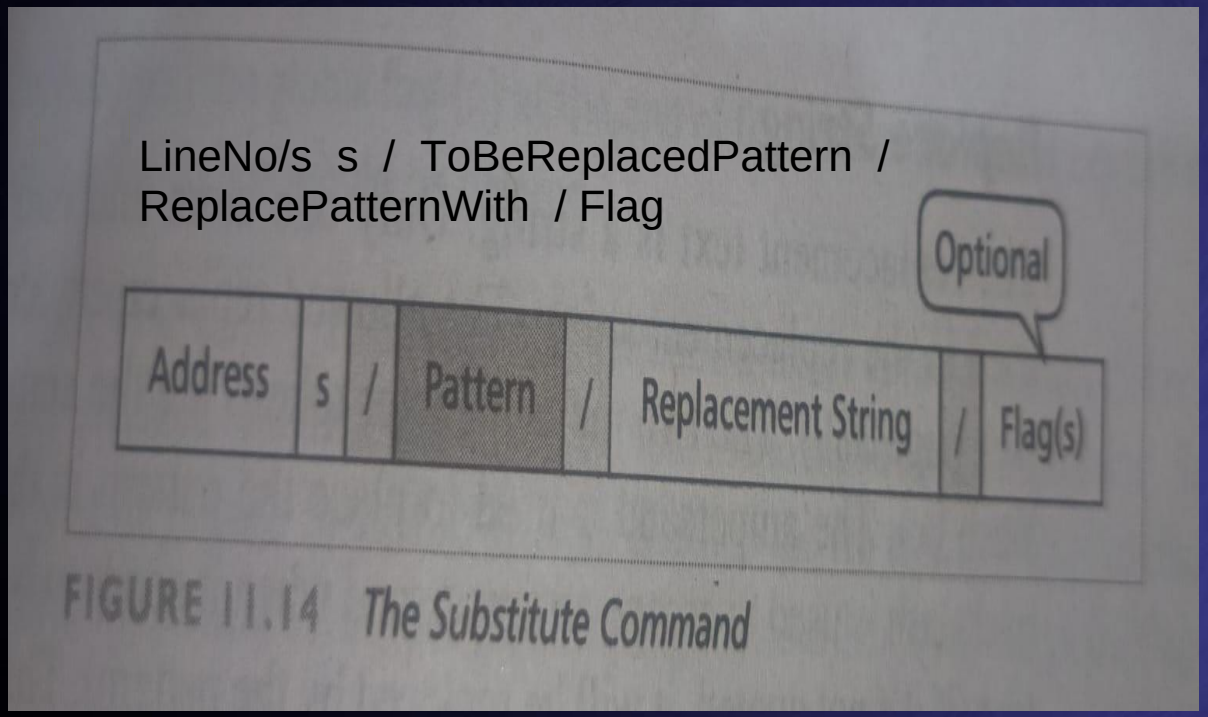


FIGURE 11.14 The Substitute Command

Substitute command

Search Pattern:

This sed search pattern uses only a subset of the regular expression atoms and pattern.

The allowable atoms and operators are listed.

TABLE 11.1 **sed's** *Regular Expressions*

Atoms	Allowed	Operators	Allowed
Character	✓	Sequence	✓
Dot	✓	Repetition	* ? \{...\}
Class	✓	Alternation	✓
Anchors	^ $	Group	✗
Back Reference	✓	Save	✓

Substitute command

Pattern Matches Address:

In this we don't need to repeat the regular expression in the substitute command.

Example:

Sed '/love/s//adore/' browing.txt

Replace String:

The replacement text is a string. Only one atom and two metacharacters can be used in the replacement string.

Example:

Sed 's?UNIX/*** & ***/' file1

Substitute command

Substitute Operation:

When the pattern matches the text, sed first deletes the text and then insets the replacement text.

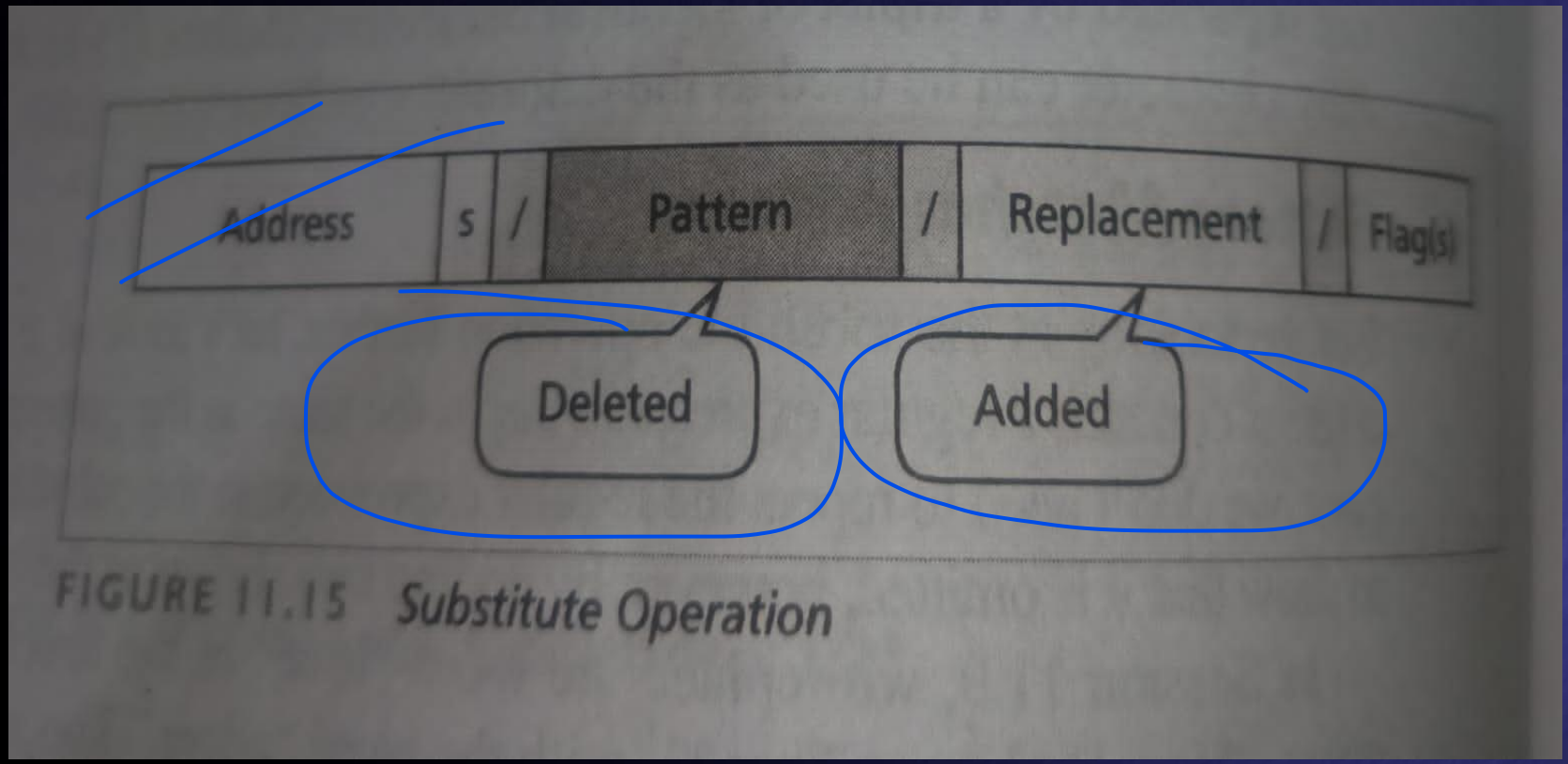


FIGURE 11.15 *Substitute Operation*

Delete part of a line

To delete part of a line, we leave the replacement text empty.

In other words, partial line deletes are a special substitution case in which the replacement is null.

Example:

Sed ‘s/[0-9]//g’

deletes all no.s

Change part of line

To change only part of line, we create a pattern that matches the part to be changed and then place the new text in the replacement expression

Example:

Sed ‘s/ / /g

Add to part of a line

To add text to a line requires both a pattern to locate the text and the text that is to be added.

Example:

Sed -f addPart.sed

Add to part of a line

Back References: Refer the text.

Whole Pattern Substitution

Numbered Buffer Substitution

Refer page number 391:

Substitute flags

There are 4 flags

- Global substitution(g)
- Specific occurrence substitution (digit)
- Print(p)
- Write file(w file-name).

Substitute flags

- **Global substitution(g)**

The substitute command only replaces the first occurrence of a pattern

If there are multiple occurrences, none after first are changes

Example:

sed '1 s/cat/dog/g'

only replaces one time in each matched line address

- **Specific occurrence substitution (digit)**

Change the first occurrence and all of the occurrences of a text pattern.

It changes any single occurrence of text that matches the pattern.

Example:

Sed '1 s/cat/dog/2'

- Print(p)
- Write file(w file-name).

Substitute flags

- Print(p)

When we do not want to print all of the output

Example when developing a script, it helps to view only the lines that have been changed.

- Write file(w file-name).

This command is similar to the print flag.

Instead of print command we used the write command.

Transform command y

- It is necessary to transform one set of characters to another.

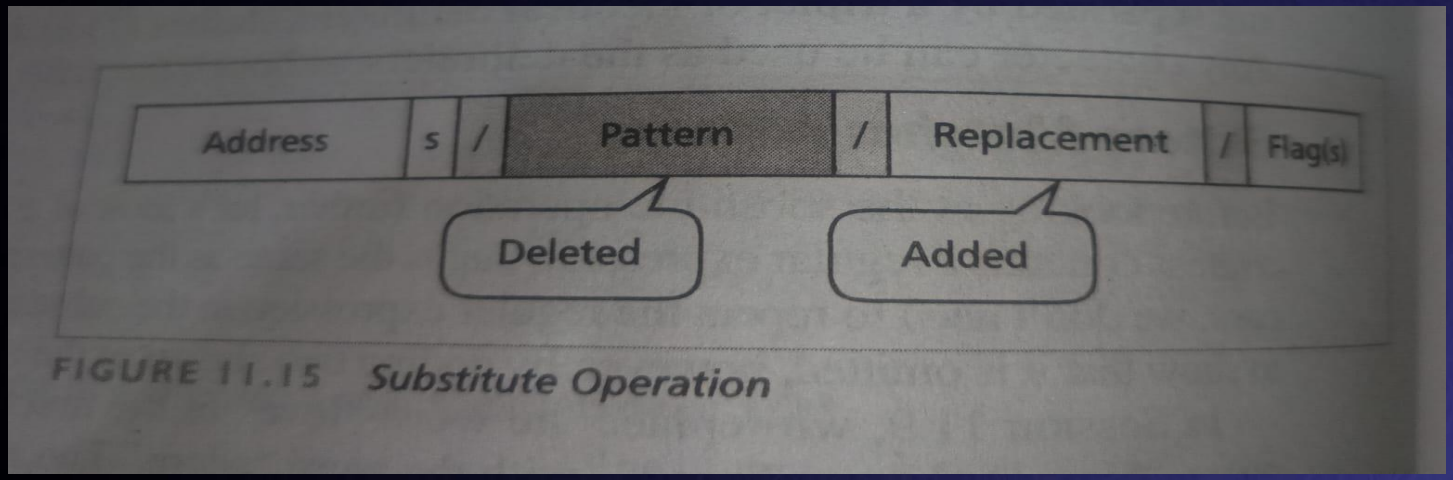


FIGURE 11.15 *Substitute Operation*

Input and Output Commands

- **The sed utility automatically reads text from the input file and writes data to the output file, usually standard output.**

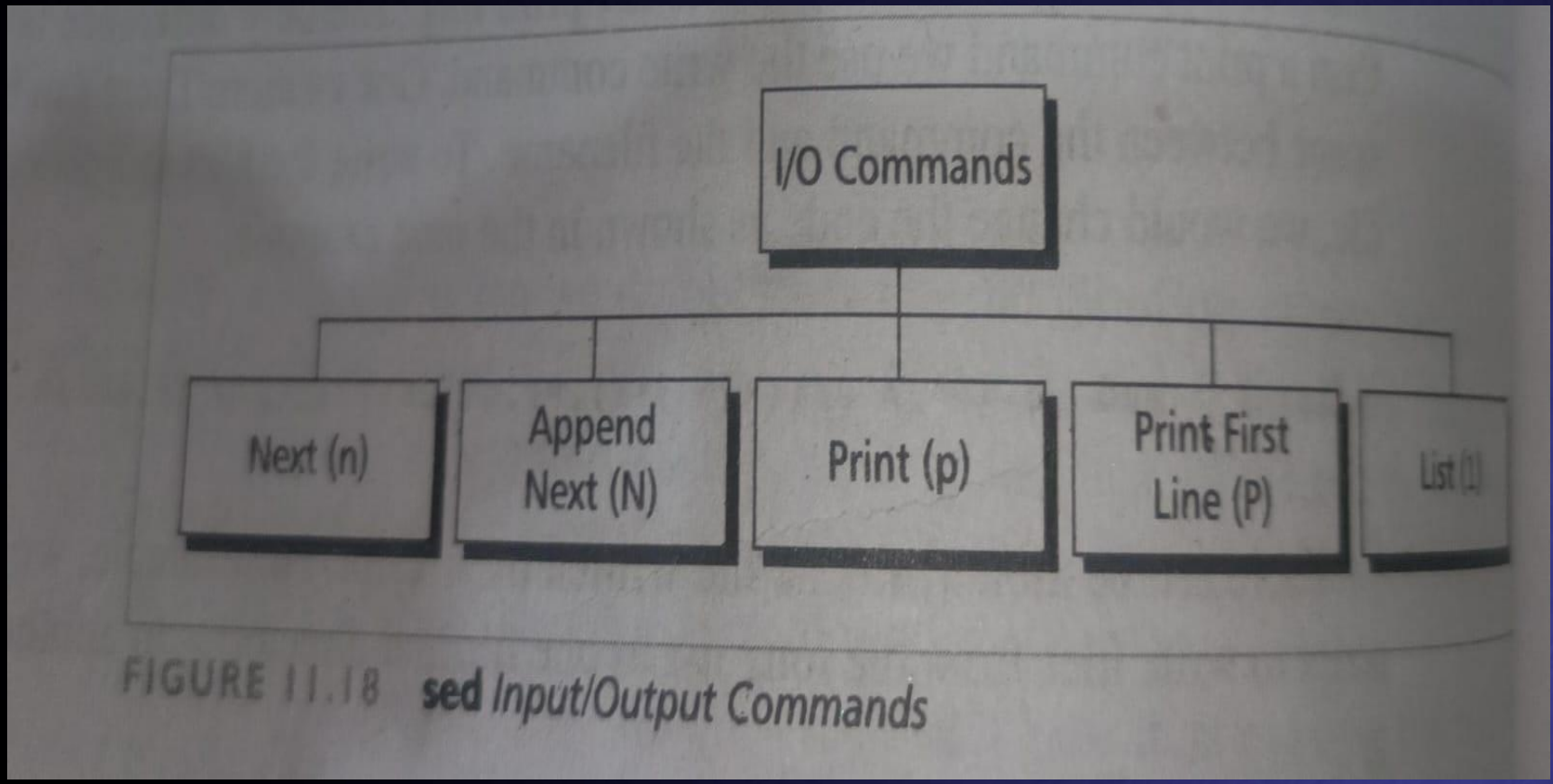


FIGURE 11.18 **sed** *Input/Output Commands*

File commands

- There are 2 file commands that can be used to read and write file.

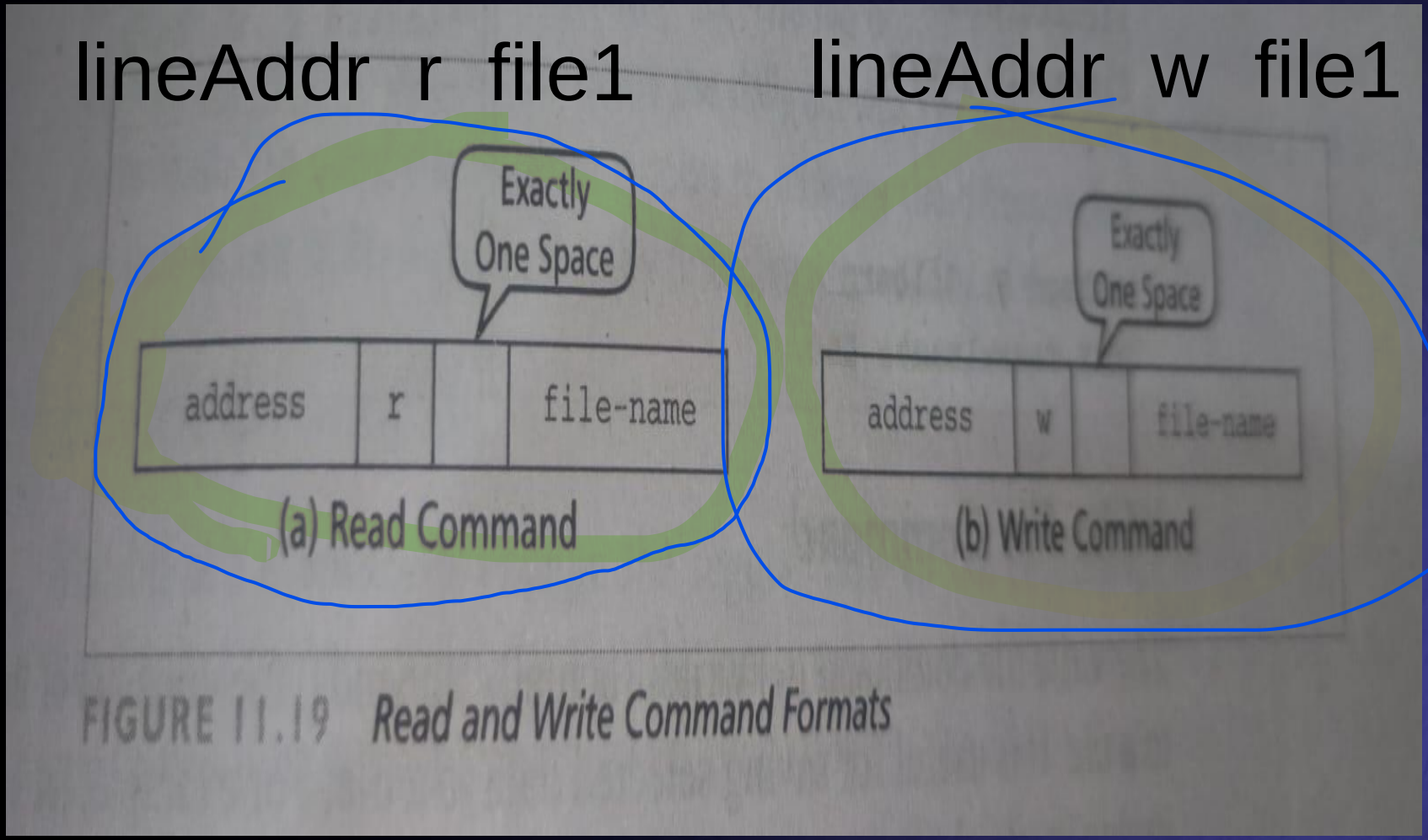


FIGURE 11.19 *Read and Write Command Formats*

Branch commands

- This command change the regular flow of the commands in the script file.
- There are 2 branch commands

1. branch (b)
2. branch on substitution (t)

Hold space commands

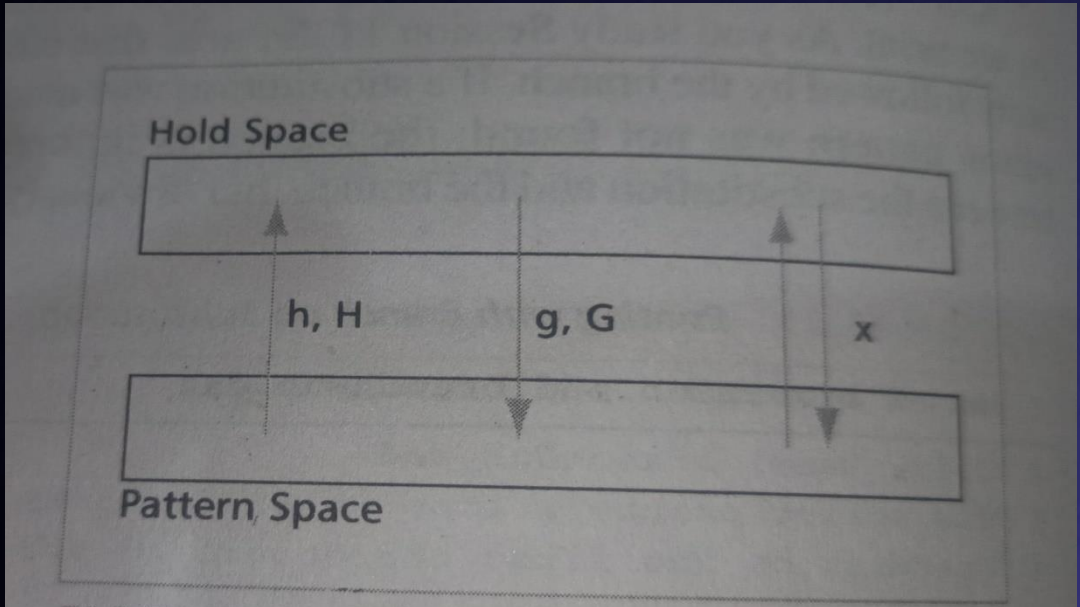


FIGURE 11.20 *Hold Space Operations*

Hold space commands

Hold and Destroy Command

The **hold and destroy** command (`h`) copies the current contents of the pattern space to the hold space and destroys any text currently in the hold space.

Hold and Append Command

The **hold and append** command (`H`) appends the current contents of the pattern space to the hold space.

Get and Destroy Command

The **get and destroy** command (`g`) copies the text in the hold space to the pattern space and destroys any text currently in the pattern space.

Get and Append Command

The **get and append** command (`G`) appends the current contents of the hold space to the pattern space.

Exchange Command

The **exchange** command (`x`) swaps the text in the pattern and hold spaces. That is, the text in the pattern space is moved to the hold space, and the data that were in the hold space are moved to the pattern space.

Quit

The quit command (q) terminates the sed utility.

Example:

Sed ‘/^$/q’ TheRaven

Applications

- **Delete Lines**

SESSION 11.34 *Delete BEGIN and END Lines*

```
$ sed '/BEGIN.*END/d' beginEnd.dat
```

Input:	Output:
1 This is the first line.	1 This is the first line.
2 This is the BEGINning of begin.	2 This is the BEGINning of begin.
3 It has several lines.	3 It has several lines.
4 This line has begin but not END.	4 This line has begin but not END.
5 This is just another line.	5 This is just another line.
6 This line has END in it.	6 This line has END in it.
7 But it is not the end.	7 But it is not the end.
8 This line has BEGIN and END in it.	9 This is the end--at last!
9 This is the end--at last!	

Applications

- **Delete Lines**

SESSION 11.35 *Delete BEGIN Not END*

```
$ sed -f beginEnd2.sed beginEnd.dat
```

```
# Delete lines that contain BEGIN but not END
/BEGIN/{
        /END/!d
       }
```

Input:

```
1 This is the first line.
2 This is the BEGINning of begin.
3 It has several lines.
4 This line has begin but not END.
5 This is just another line.
6 This line has END in it.
7 But it is not the end.
8 This line has BEGIN and END in it.
9 This is the end--at last!
```

Output:

```
1 This is the first line.
3 It has several lines.
4 This line has begin but not END
5 This is just another line.
6 This line has END in it.
7 But it is not the end.
8 This line has BEGIN and END in it.
9 This is the end--at last!
```

Applications

- **Delete Lines**

SESSION 11.36 *Delete BEGIN through END*

```
$ sed '/BEGIN/,/END/d' beginEnd.dat
```

Input:

```
1 This is the first line.
2 This is the BEGINning of begin.
3 It has several lines.
4 This line has begin but not END.
5 This is just another line.
6 This line has END in it.
7 But it is not the end.
8 This line has BEGIN and END in it.
9 This is the end--at last!
```

Output:

```
1 This is the first line.
5 This is just another line.
6 This line has END in it.
7 But it is not the end.
```

Applications

- **Delete Text**

SESSION 11.37 *Delete Text between BEGIN and END*

```
$ sed 's/BEGIN.*END//' beginEnd.dat
```

Input:	Output:
1 This is the first line.	1 This is the first line.
2 This is the BEGINning of begin.	2 This is the BEGINning of begin.
3 It has several lines.	3 It has several lines.
4 This line has begin but not END.	4 This line has begin but not END.
5 This is just another line.	5 This is just another line.
6 This line has END in it.	6 This line has END in it.
7 But it is not the end.	7 But it is not the end.
8 This line has BEGIN and END in it.	8 This line has in it.
9 This is the end--at last!	9 This is the end--at last!

Grep and sed

- Lines that match a Regular Expression
  - If we need to use sed instead of grep to find a line that matches a regular expression.
  - Print(p) command in sed and turn off the automatic output option (-n).
  - A grep regular expression command can be simulated using the regular expressions in a sed command.
  - Example:
  - grep ‘regular expressions’ file1
  - sed -a ‘/ regular experssion/p’ file1

Grep and sed

- Lines that do not match a Regular Expression
- If we want to use sed instead of grep (or egrep) to find lines that do not match a regular expression (activated in grep with the -v option)
- We again use the print command but this time we complement the address (!) in sed.
- Because we are again controlling the printing, we must use the no output option (-n)

- Example :
- grep -v ‘regular expression’ file1
- sed -n ‘/regular expression/!p’ file1

Awk

- Execution
- Fields and Records
- Scripts
- Operation
- Patterns
- Actions
- Associative Arrays
- String Functions
- Mathematical Functions
- User-Defined Functions
- Using System Commands in awk
- Applications

Refer Text books from page number 425 to 479

Unix and shell programming

Behrouz A. FOROUZAN Richard F Gilberg