Outline

MCS 260 Lecture 17
Introduction to Computer Science
Jan Verschelde, 19 February 2016

compression and conversion using buffers

compressing files with gzip or gunzip

File compression saves space on mass storage.

A session at the Linux command prompt `$`:

```
$ ls -g files.pdf
-rwxr-xr-x .. 487634 Oct 17 07:16 files.pdf
$ gzip files.pdf
$ ls -g files.pdf
ls: files.pdf: No such file or directory
$ ls -g files.pdf.gz
-rwxr-xr-x .. 236452 Oct 17 07:16 files.pdf.gz
```

Gzip reduces size of file from 487634 to 236452 bytes.

```
$ gunzip files.pdf.gz
$ ls -g files.pdf
-rwxr-xr-x .. 487634 Oct 17 07:16 files.pdf
```

Gzip uses the Lempel-Ziv encoding (LZ77).

compression and conversion using buffers

data compression

Two different data compression schemes:

- lossless: no loss of information in compression,
- lossy: minor errors are tolerated (e.g. images).

Popular compression schemes use dictionary encoding.
For example: replace all `the`'s in a text file by a symbol.

The Lempel-Ziv encoding used in Gzip

- finds duplicated strings in the input data.
- The second occurrence of a string is replaced by a pointer to the previous string.

compression and conversion using buffers

archiving files with the utility tar

To create an archive of several files, we use `tar`.

`tar` is an archiving program to store and extract files from an archive, called a *tarfile*. The `t` in `tar` stands for "tape" although we mostly write to a file.

Suppose we want to archive all our Python programs:

1. create a separate directory `pyprogs`;
2. copy all files with extension `.py` to `pyprogs`;
3. apply `tar` (eventually followed by gzip).

A session at the Linux command prompt `$`:

```
$ mkdir pyprogs
$ cp *.py pyprogs
$ tar -cf pyprogs.tar pyprogs
```

The `c` and `f` of `tar` are for `create` and `file`.

using file archives

To display the files in the archive file `pyprogs.tar`:

```
$ tar -tf pyprogs.tar
```

Again, the `f` is for `file`, while `t` is to display.

To extract the files for the archive file `pyprogs.tar`:

```
$ tar -xf pyprogs.tar
```

The `x` and `f` of `tar` are for `extract` and `file`.

If the directory `pyprogs` already exists,
then `tar` will update the files with newer versions,
otherwise the directory `pyprogs` will be created.

concatenation of files with the utility cat

Suppose we want to send small files to the printer.

As all files fit on one page, we better make one new file that contains everything.

At the command prompt `$`

```
$ cat g*.py > toprint
```

This command combines three actions:

1. `cat` concatenates files, given as arguments;
2. `g*.py`: all files starting with `g` and ending with `.py`;
3. instead of writing to screen, with `>` all output goes to the file `toprint`.

redirection and pipes: sorting misspelled words

`spell` reads text and writes misspelled words.

```
$ spell
hello
helo
helo
```

The first two lines were input, the last line was output.

To print all misspelled words in a the file `sometext`:

```
$ spell < sometext
```

To see a sorted list of misspelled words:

```
$ spell sometext | sort
```

The `|` indicates a pipe, setting up a pipeline redirecting the output of `spell` to the input of `sort`, without creating an intermediate file, gluing two different programs.

pattern matching: another example of a pipeline

The pipe functionality summarizes the power of Unix.

Another illustration:

```
$ cat *.py | grep "MCS 260" | more
```

is a pipe of three programs:

1. `cat *.py` shows the content of all `.py` files
2. `grep "MCS 260"` prints all lines in the files that match the string `"MCS 260"`
3. `more` pauses printing after first screen is full.

With Python we can glue several applications *independent* of the operating system.

compression and conversion using buffers

1. Compressing Files
   - gzip and gunzip
   - data compression
   - archiving files and pipes in Unix
2. File Methods in Python
   - format conversions
   - encrypting text
3. Using Buffers
   - counting and replacing words
   - using tell() to locate a word
   - using seek() to replace words
4. Summary + Assignments

format conversions

Convert the ad hoc formatting of the file books

```
1:1:Computer Science, an overview:
0:2:Python Programming in Context:
```

into one that uses lists

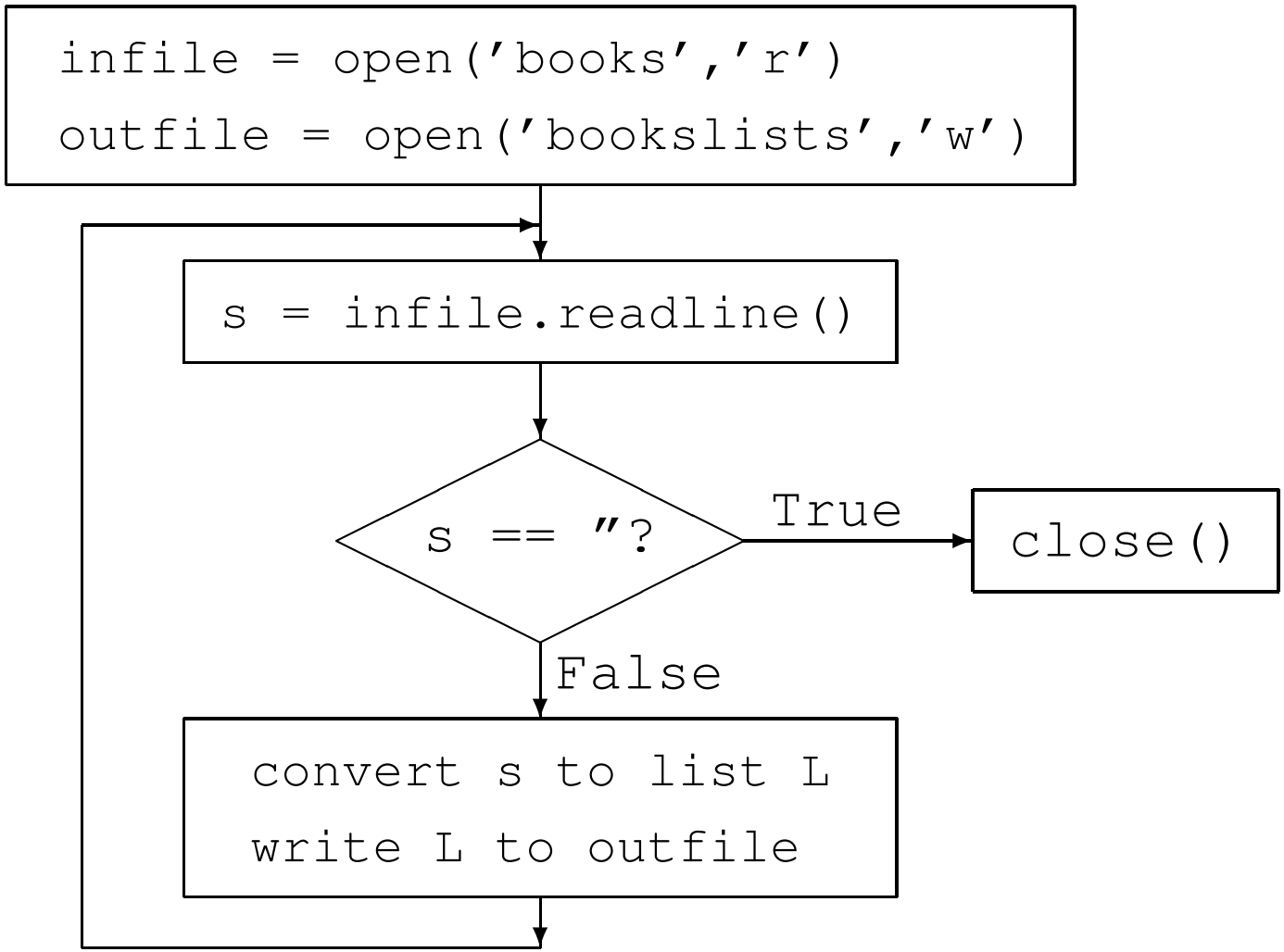
```
[True, 1, 'Computer Science, an overview']
[False, 2, 'Python Programming in Context']
```

and into using dictionaries

```
{'available': True, 'key': 1, \
'title': 'Computer Science, an overview}
{'available': False, 'key': 2, \
'title': 'Python Programming in Context'}
```

Benefit: the module `bkform` is no longer needed.

format conversion algorithm

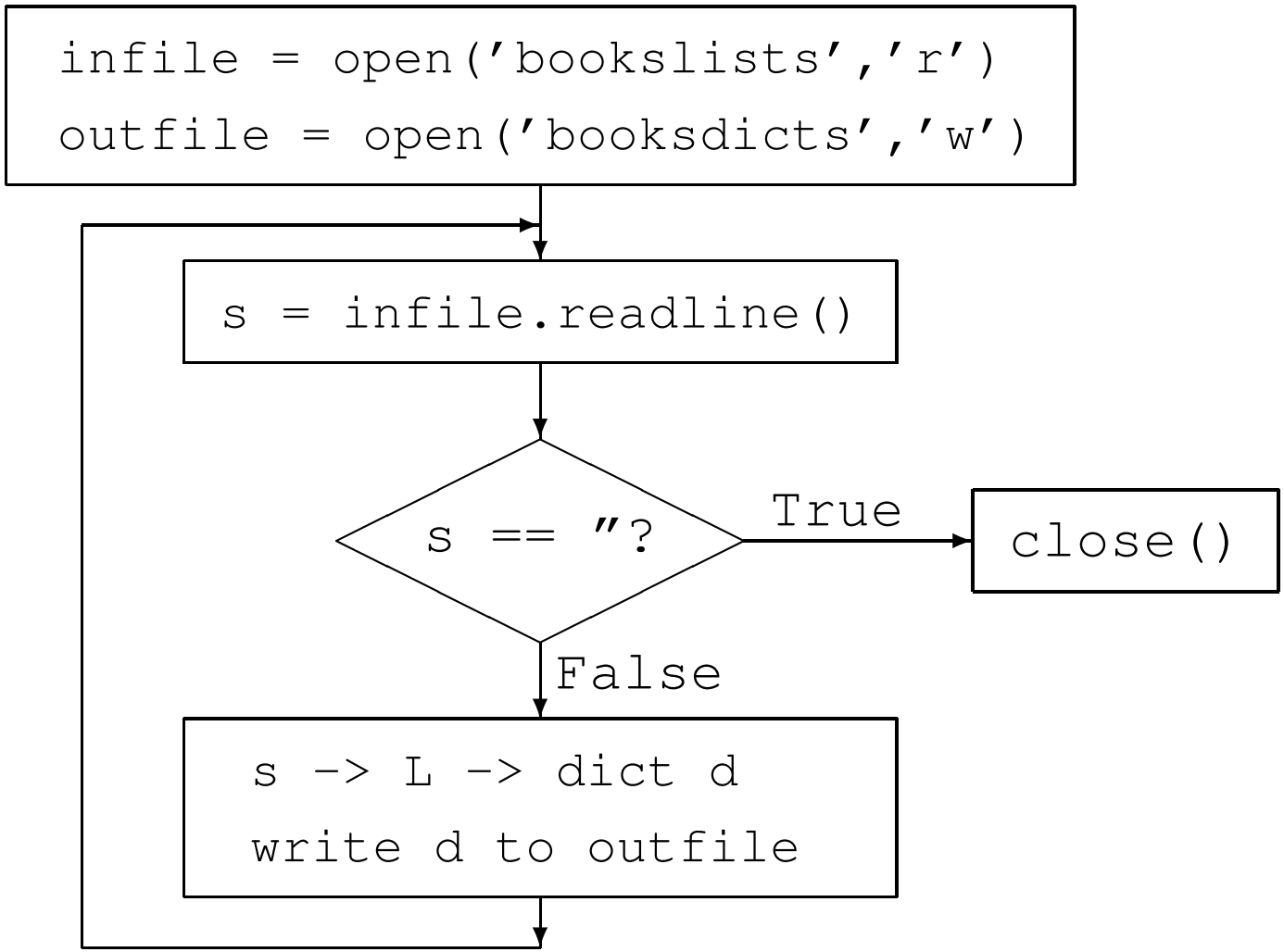


Python program bkform2list.py

```
INFILE = open('books', 'r')
OUTFILE = open('bookslists', 'w')
while True:
    S = INFILE.readline()
    if S == '':
        break
    L = S.split(':')
    R = [L[0] == '1', int(L[1]), L[2]]
    OUTFILE.write(str(R) + '\n')
INFILE.close()
OUTFILE.close()
```

Observe: `L[0] == '1'` returns bool for availability
`int(L[1])` returns integer for key

converting lists into dictionaries



Python program bklist2dict.py

```
from ast import literal_eval
INFILE = open('bookslists', 'r')
OUTFILE = open('booksdicts', 'w')
while True:
    S = INFILE.readline()
    if S == '':
        break
    L = literal_eval(S)
    D = {'available':L[0], 'key':L[1], \
         'title':L[2]}
    OUTFILE.write(str(D) + '\n')
INFILE.close()
OUTFILE.close()
```

Observe: `literal_eval(str(L)) == L`

`str(L)` converts a list to a string

`literal_eval()` evaluates a string into an object

compression and conversion using buffers

scrambling text – encrypting information

The content of the file sometext:

```
This is a sample text, used as an example
for a message whose vowels will be scrambled.
```

After scrambling vowels, we obtain codetext:

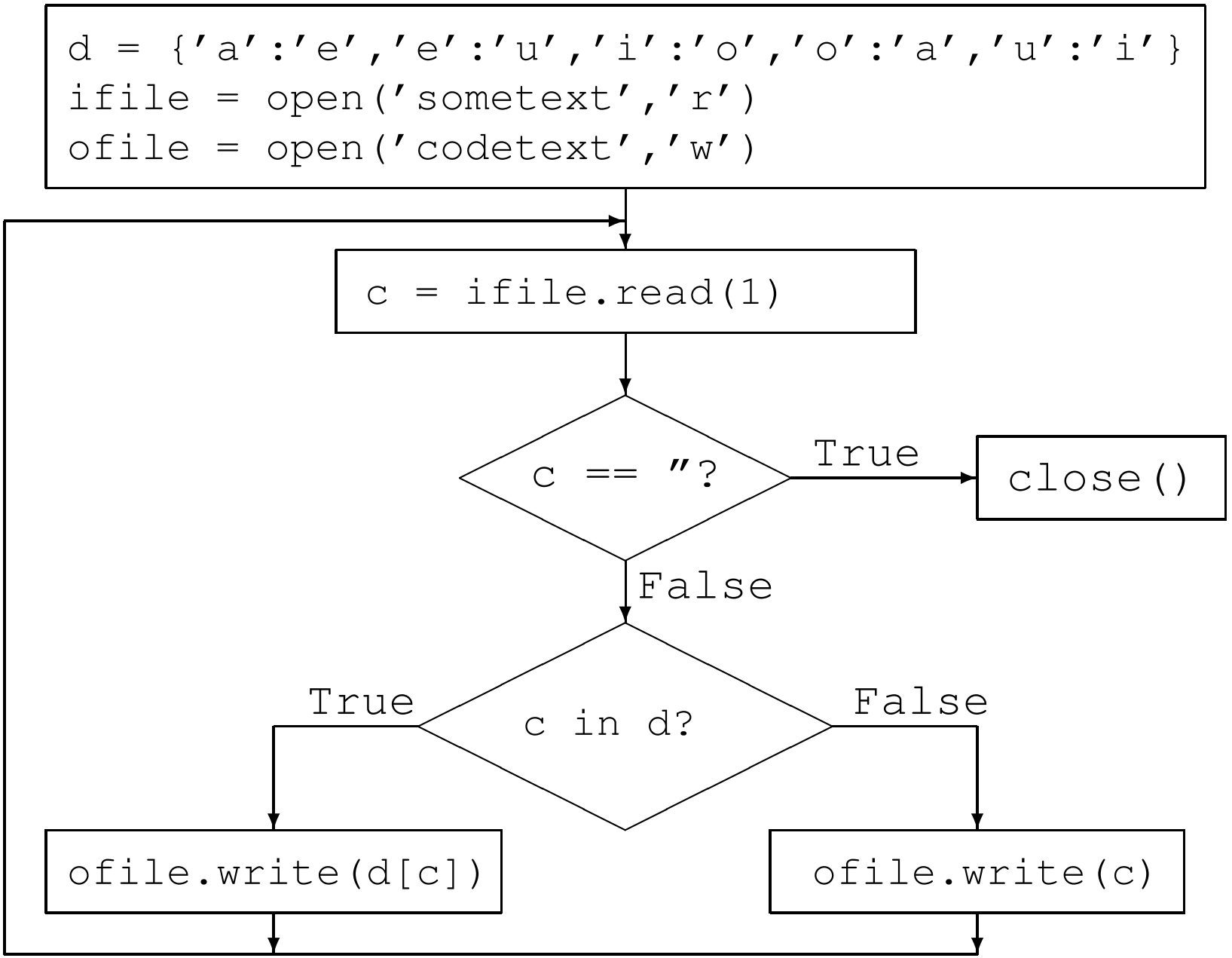
```
Thos os e semplu tuxt, isud es en uxemplu
far e mussegu whasu vawuls woll bu scremblud.
```

The cipher used:

```
['a','i','e','u','o'] -> ['e','o','u','i','a']
```

Python: `file.read(1)` returns one byte read from file.

the scrambling algorithm



scrambling vowels

```
D = {'a':'e', 'e':'u', 'i':'o', 'o':'a', 'u':'i'}
print D.keys(), '->', D.values()
NAME = input('name of input file : ')
INFILE = open(NAME, 'r')
NAME = input('name of output file : ')
OUTFILE = open(NAME, 'w')
while True:
    C = INFILE.read(1)
    if C == '':
        break
    if C in D:
        OUTFILE.write(D[C])
    else:
        OUTFILE.write(C)
INFILE.close()
OUTFILE.close()
```

compression and conversion using buffers

1. Compressing Files
   - gzip and gunzip
   - data compression
   - archiving files and pipes in Unix
2. File Methods in Python
   - format conversions
   - encrypting text
3. Using Buffers
   - counting and replacing words
   - using tell() to locate a word
   - using seek() to replace words
4. Summary + Assignments

counting words in a text

Problem: count number of times `the` occurs.

For example: if the file sometext2 has content

```
At the end of the lecture we go home,
taking the bus or the train.
```

then the number of times we count `the` is 4.

Algorithm:

1. maintain a buffer of 3 consecutive letters on file
2. after each new byte read, compare with `the`

updating a buffer with the function add_to_buffer

A buffer is a list of 3 characters.
We do not want to assign to a function argument.

```
def add_to_buffer(buf, let):
    """
    Adds let to a 3-letter buffer buf,
    on return is the updated buffer.
    """
    nbf = buf
    if len(buf) == 3:
        (nbf[0], nbf[1], nbf[2]) = (nbf[1], nbf[2], let)
    else:
        nbf.append(let)
    return nbf
```

We place the function `add_to_buffer`
at the start of the program `cntword.py`.

Counting Words: the main program cntword.py

```
def add_to_buffer(buf, let):
    ...

SEARCH = ['t', 'h', 'e']
BUFF = []
NAME = input('give file name : ')
FILE = open(NAME, 'r')
COUNT = 0
while True:
    C = FILE.read(1)
    if C == '':
        break
    BUFF = add_to_buffer(BUFF, C)
    if BUFF == SEARCH:
        COUNT += 1
FILE.close()
```

compression and conversion using buffers

Locating a Words using the method tell()

Suppose we want to know the positions in the file of all occurrences of the word `the`.

For example: if the file sometext2 has content

```
At the end of the lecture we go home,
taking the bus or the train.
```

then locword.py will print

```
the occurs at  [3L, 14L, 45L, 56L]
```

because `the` occurs at byte 3, 14, 45, and 56.

Same algorithm as counting words.
file.tell() returns current position of cursor in file.

the program locword.py uses function add_to_buffer

```
search = ['t','h','e']
buffer = []
name = input('give file name : ')
file = open(name,'r')
locations = []
while True:
    c = file.read(1)
    if c == '':
        break
    buffer = add_to_buffer(buffer, c)
    if buffer == search:
        locations.append(file.tell()-3)
file.close()
print 'the occurs at ', locations
```

compression and conversion using buffers

replacing words introducing the method seek()

Suppose we want to replace all occurrences of the word `the` by `one` in a text.

For example, if the file sometext2 has content

```
At the end of the lecture we go home,
taking the bus or the train.
```

then repword.py will change this file into

```
At one end of one lecture we go home,
taking one bus or one train.
```

Again same algorithm, using a 3-letter buffer.

file.seek(offset,direction) moves the cursor
as many bytes as the value of `offset`, starting from
(0) the beginning of the file, if `direction` = 0;
(1) the current position, if `direction` = 1; or
(2) the end of the file, if `direction` = 2.

repword.py uses the function add_to_buffer

```
def add_to_buffer(b,c):
    "adds c to 3-letter buffer"
    ...

SEARCH = ['t', 'h', 'e']
BUFF = []
NAME = input('give file name : ')
FILE = open(NAME, 'r+')
while True:
    C = FILE.read(1)
    if C == '':
        break
    BUFF = add_to_buffer(BUFF, C)
    if BUFF == SEARCH:
        CURSOR = FILE.tell()
        FILE.seek(-3, 1)
        FILE.write('one')
FILE.close()
```

Summary + Assignments

Read pages 155-165 in *Python Programming in Context.*
Read §1.8 in *Computer Science, an overview*.

Assignments:

1. Write a program `bkform2dict.py` to convert the formatting in the file books directly to booksdicts.
2. Rewrite the program `library.py` of the previous lecture, using dictionaries as formats for the file books.
3. Modify the `cntword.py` so that it considers a word as separated from others by spaces or commas and other punctuation symbols, i.e.: the `the` in `there` or `then` does not count as an occurrence of `the`.
4. Design a permutation of the vowels so that after applying `scramvow.py` twice we get the original message.

more assignments

5. Download
   `http://www.gutenberg.org/dirs/etext97/1ws3410.txt`
   and write a script to count the number of occurrences of the strings `"LADY MACBETH"` and `"MACBETH"`.
6. Download
   `http://ichart.finance.yahoo.com/table.csv?s=ibm`
   to get the evolution of the value of IBM stock as a file in csv (comma separated value) format. Download this file and write a script to find the days at which IBM stock was valued at its highest and lowest price.