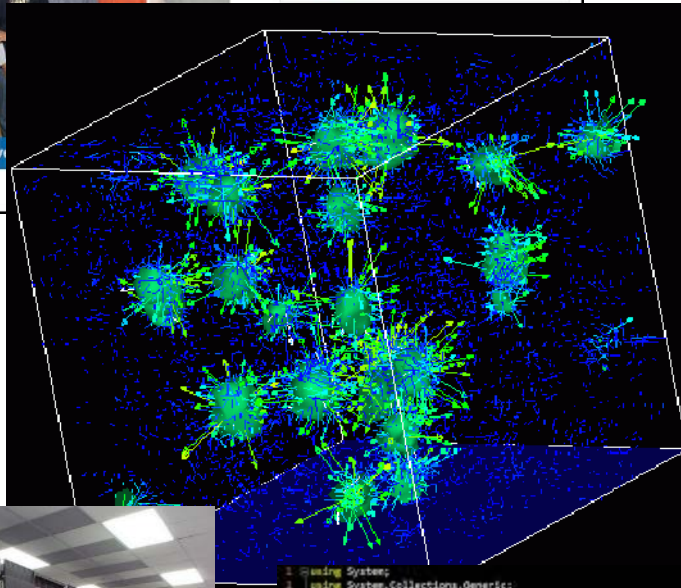




Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
```

Session 14
Cryptanalysis,
Anagrams

Session Goals

- Manage a C++ **string** as a **vector<char>**
 - Understand **ASCII** as an encoding mechanism
 - Read an ASCII **text file** stored on disk into a memory buffer
 - Generate a **histogram of character frequencies** within a file
- Encrypt and decrypt files using “**Caesar Shift**”
 - Perform **bigram analysis** on unreadable ciphertext to determine the author’s native language
- Generate and discover simple and compound **anagrams**
 - Avoid combinatorial (exponential) explosion in search space

C++ strings

- A C++ string is mostly equivalent to a **vector<char>**
 - A C++ **char** data type holds one “character”
 - There may be a difference between the length of a string (the number of "characters" in the string) and the actual number of bytes required to store it in memory or on disk
 - The memory size (number of bits) of a character can vary by encoding scheme and platform (Windows vs. Linux)
- A string has **.size()** or **.length()** methods to get the number of *characters* in the string
- We can access individual characters using the **.at()** method

Reverse a String

i	s.at()
0	F
1	o
2	r
3	e
4	v
5	e
6	r
7	(space)
8	Y
9	o
10	u
11	n
12	g

s.length()==13

```
reverse-string
File Edit View Terminal Tabs Help
Original string = Forever Young
Reversed string = gnuoY reveroF
STL reverse() = gnuoY reveroF

Process returned 0 (0x0)    execution time : 0.016 s
Press ENTER to continue.
```

We could try to **swap** each letter around the middle element, but that would require complicated code

Open Lab 1

Reverse String

```
main.cpp
1  #include "stdafx.h"
2
3  using namespace std;
4
5  string ReverseString(string a)
6  {
7      string b;
8      // Implement your algorithm here
9
10     return b;
11 }
12
13
14
15 int main()
16 {
17     string s = "Forever Young";
18     string r = ReverseString(s);
19
20     cout << "Original string = "
21           << s << endl;
22
23     cout << "Reversed string = "
24           << r << endl;
25
26     reverse(s.begin(), s.end());
27
28     cout << " STL reverse() = "
29           << s << endl;
30
31     return 0;
32 }
33
```

Work backwards from the *end* of string **a** appending each character to string **b**

Edit Lab 1 – Reverse String

- Only add code to the **ReverseString()** function
 - A **string** is an vector – you can access individual elements (characters) using the **.at()** method
 - Use **.length()** to get the number of characters in the string
 - You can append characters to the end of a string using the **+** operator
- On return, **b** should be the string **a** in reverse char order

```
5  string ReverseString(string a)
6  {
7      string b;
8
9      // Implement your algorithm here
10
11
12      return b;
13 }
```



Edit Lab 1 – Reverse String

- Only add code to the **ReverseString()** function
 - A **string** is an vector – you can access individual elements (characters) using the **.at()** method
 - Use **.length()** to get the number of characters in the string
 - You can append characters to end of a string using the **+** operator
- On return, **b** should be the string **a** in reverse char order

```
5  string ReverseString(string a)
6  {
7      string b;
8
9      for(int i = a.length() - 1; i >= 0; --i)
10         b += a.at(i);
11
12     return b;
13 }
```

Check Lab 1

Reverse String

main.cpp

```
1  #include "stdafx.h"
2
3  using namespace std;
4
5  string ReverseString(string a)
6  {
7      string b;
8
9      for(int i = a.length() - 1; i >= 0; --i)
10         b += a.at(i);
11
12     return b;
13 }
14
15 int main()
16 {
17     string s = "Forever Young";
18     string r = ReverseString(s);
19
20     cout << "Original string = "
21           << s << endl;
22
23     cout << "Reversed string = "
24           << r << endl;
25
26     reverse(s.begin(), s.end());
27
28     cout << " STL reverse() = "
29           << s << endl;
30
31     return 0;
32 }
33
```

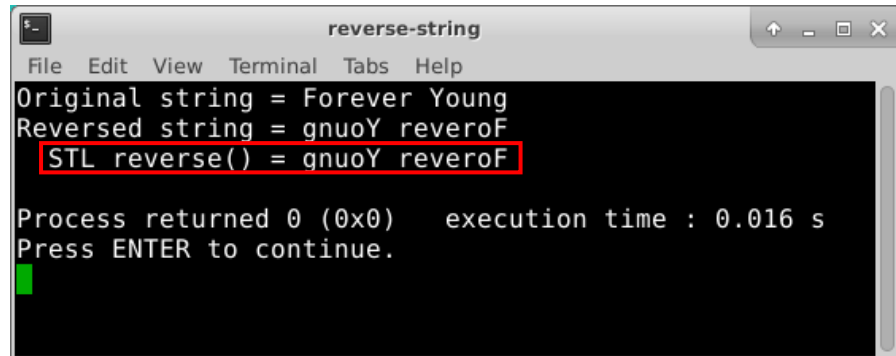
Walk backwards through string **a** while dynamically building string **b** one character at a time

The C++ Standard Template Library (**STL**) has a built-in function to reverse the order of the elements in **any** vector

Run Lab 1 – Reverse String

i	s.at()
0	F
1	o
2	r
3	e
4	v
5	e
6	r
7	(space)
8	Y
9	o
10	u
11	n
12	g

s.length()==13



```
reverse-string
File Edit View Terminal Tabs Help
Original string = Forever Young
Reversed string = gnuoY reveroF
STL reverse() = gnuoY reveroF

Process returned 0 (0x0)   execution time : 0.016 s
Press ENTER to continue.
```

STL = Standard Template Library

The **STL** is a set of free & open-source functions and classes to *reduce* the amount of code C++ programmers must write to solve common problems

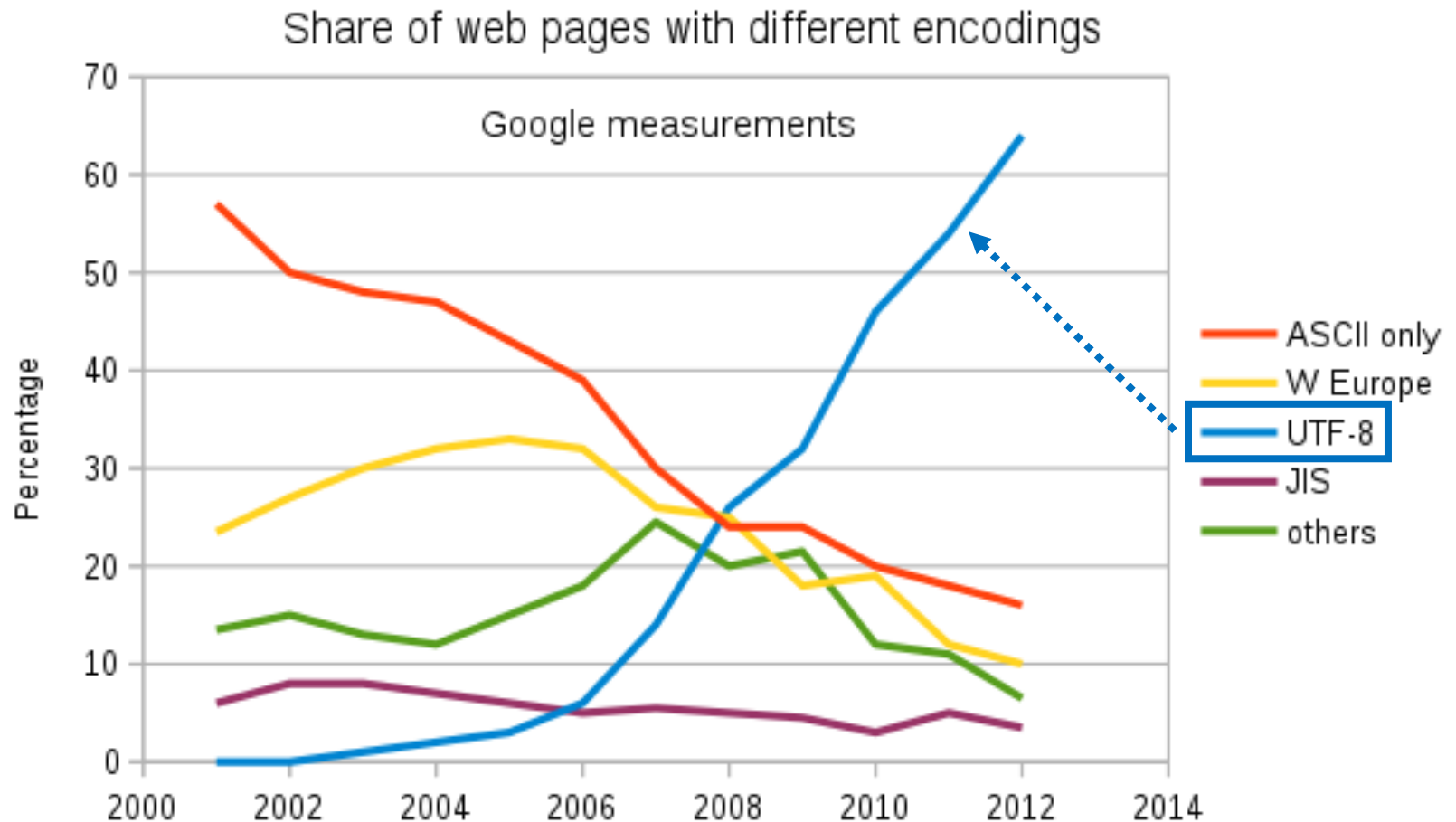


ASCII (“as-key”)

- **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
 - ASCII was the most common legacy International standard used across the Internet until 2007
 - Since 2008 ASCII has been surpassed by **UTF-8** (Universal Transformation Format), which includes ASCII as a subset
- ASCII is an 8-bit (**one byte**) character encoding scheme
 - ASCII maps most of the characters in the (Western) languages descending from Latin to a specific integer value
 - In ASCII there is a 1:1 correspondence between a letter and a number and every character is **always** one byte long
 - Inside a computer, **all** letters, punctuation marks, even numbers (when treated as strings) are encoded as either ASCII or UTF

Learn about **UTF-8**

<https://en.wikipedia.org/wiki/UTF-8>



ASCII range for common English characters

Dec	Char
32	(space)
33	!
34	"
35	#
36	\$
37	%
38	&
39	'
40	(
41)
42	*
43	+
44	,
45	-
46	.
47	/
48	0
49	1
50	2

Dec	Char
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?
64	@
65	A
66	B
67	C
68	D
69	E

Dec	Char
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X

Dec	Char
89	Y
90	Z
91	[
92	\
93]
94	^
95	_
96	`
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k

Dec	Char
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	
125	}
126	~

Creating a Frequency Histogram














- Consider Lincoln's Gettysburg Address:






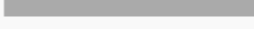
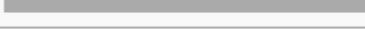






Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure...

- We want to perform a **histogram analysis** of Lincoln's speech
 - What **letter** do you think occurs most frequently in **English**?
 - Spaces (**ASCII value 32**) usually occur most often because we use spaces as a word breaker

Letter Frequencies in the English Language

Letter ↕	Relative frequency in the English language ↕	
a	8.167%	
b	1.492%	
c	2.782%	
d	4.253%	
e	12.702%	
f	2.228%	
g	2.015%	
h	6.094%	
i	6.966%	
j	0.153%	
k	0.772%	
l	4.025%	
m	2.406%	

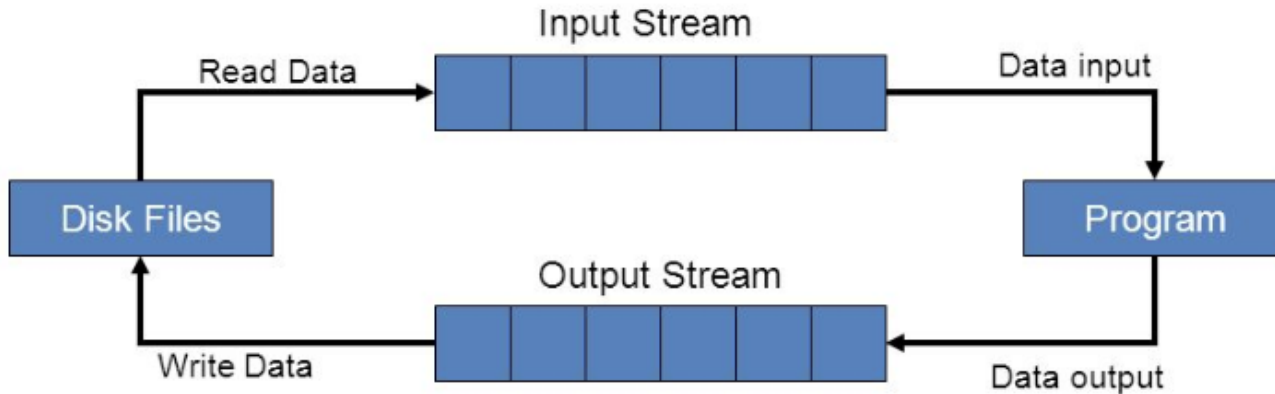
Letter ↕	Relative frequency in the English language ↕	
n	6.749%	
o	7.507%	
p	1.929%	
q	0.095%	
r	5.987%	
s	6.327%	
t	9.056%	
u	2.758%	
v	0.978%	
w	2.361%	
x	0.150%	
y	1.974%	
z	0.074%	

ETAOIN SHRDLU

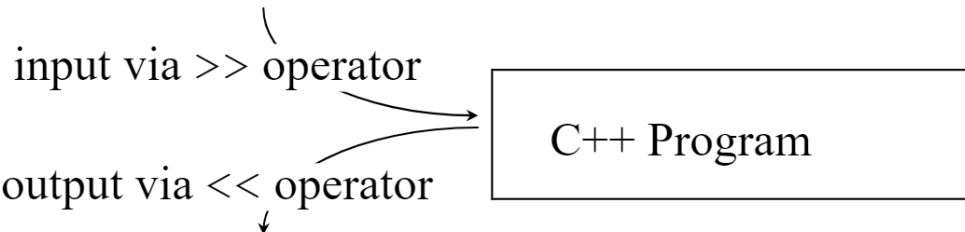
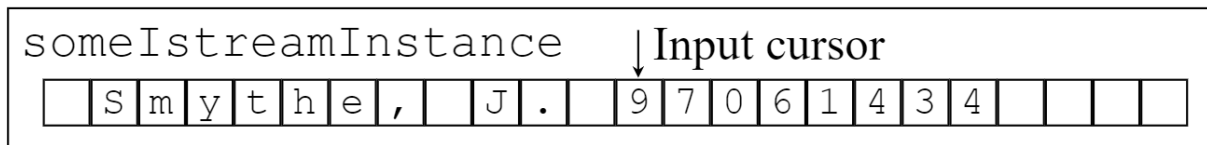
ASCII Text Files – A “stream”

- A file on disk is essentially just a **byte vector**
 - A **byte** is an 8-bit *unsigned integer* between 0 and 255 (**uint8_t**)
 - We can declare a byte vector and load it with the contents of a file
 - A **stream** of file bytes in memory is called a **buffer**
- We can then access any individual character within the file
 - Use the normal **.at() method** on this vector of **"fileBytes"**
 - We only need to specify an **index value** to get a specific byte

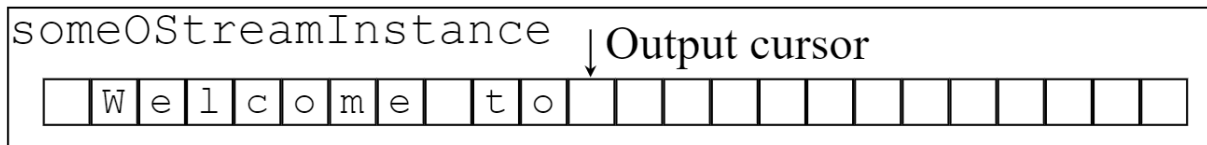
ASCII Text Files – A “stream”



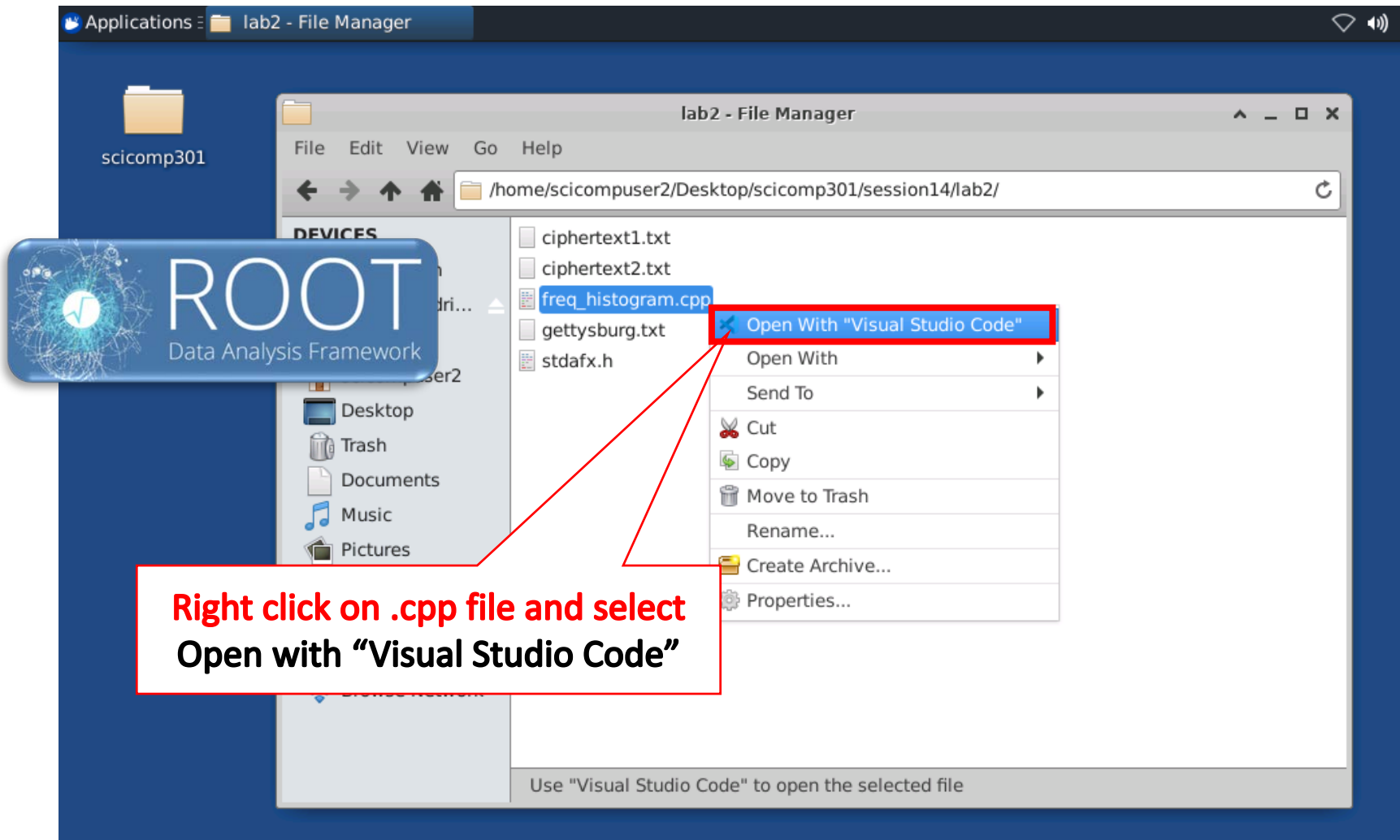
`cin >> n;`



`cout << n;`



Open Lab 2: freq_histogram.cpp



View Lab 2 Frequency Histogram

```
void freq_histogram(string filename)
{
    // Open file at the end (ios::ate) so the next "get"
    // position will return the actual file size
    ifstream ifs(filename, ios::binary | ios::ate);
    ifstream::pos_type pos = ifs.tellg();

    // Allocate a vector big enough to hold all the file bytes
    vector<unsigned char> fileBytes(pos);

    // Read in the file from the beginning straight into the vector
    ifs.seekg(0, ios::beg);
    ifs.read((char*)(fileBytes.data()), pos);

    // Create a ROOT chart
    string title = "Frequency Analysis";
    TCanvas* c1 = new TCanvas(title.c_str());
    c1->SetTitle(title.c_str());

    // Create a ROOT one dimensional histogram of integers
    TH1I* h1 = new TH1I(nullptr, title.c_str(), 256, 0, 257);
    h1->SetStats(kFALSE);

    TAxis* ya = h1->GetYaxis();
    ya->SetTitle("Count");
    ya->CenterTitle();

    TAxis* xa = h1->GetXaxis();
    xa->SetTitle("ASCII Value");
    xa->CenterTitle();
    xa->SetTickSize(0);
}
```

A file on disk is just a
vector of integers
(unsigned char = byte)

View Lab 2

Frequency Histogram

```
// Create a ROOT chart
string title = "Frequency Analysis";
TCanvas* c1 = new TCanvas(title.c_str());
c1->SetTitle(title.c_str());

// Create a ROOT one dimensional histogram of integers
TH1I* h1 = new TH1I(nullptr, title.c_str(), 256, 0, 257);
h1->SetStats(kFALSE);

TAxis* ya = h1->GetYaxis();
ya->SetTitle("Count");
ya->CenterTitle();

TAxis* xa = h1->GetXaxis();
xa->SetTitle("ASCII Value");
xa->CenterTitle();
xa->SetTickSize(0);

// Fill the histogram using the bytes in the file
for (auto item : fileBytes)
    h1->Fill((int)item);

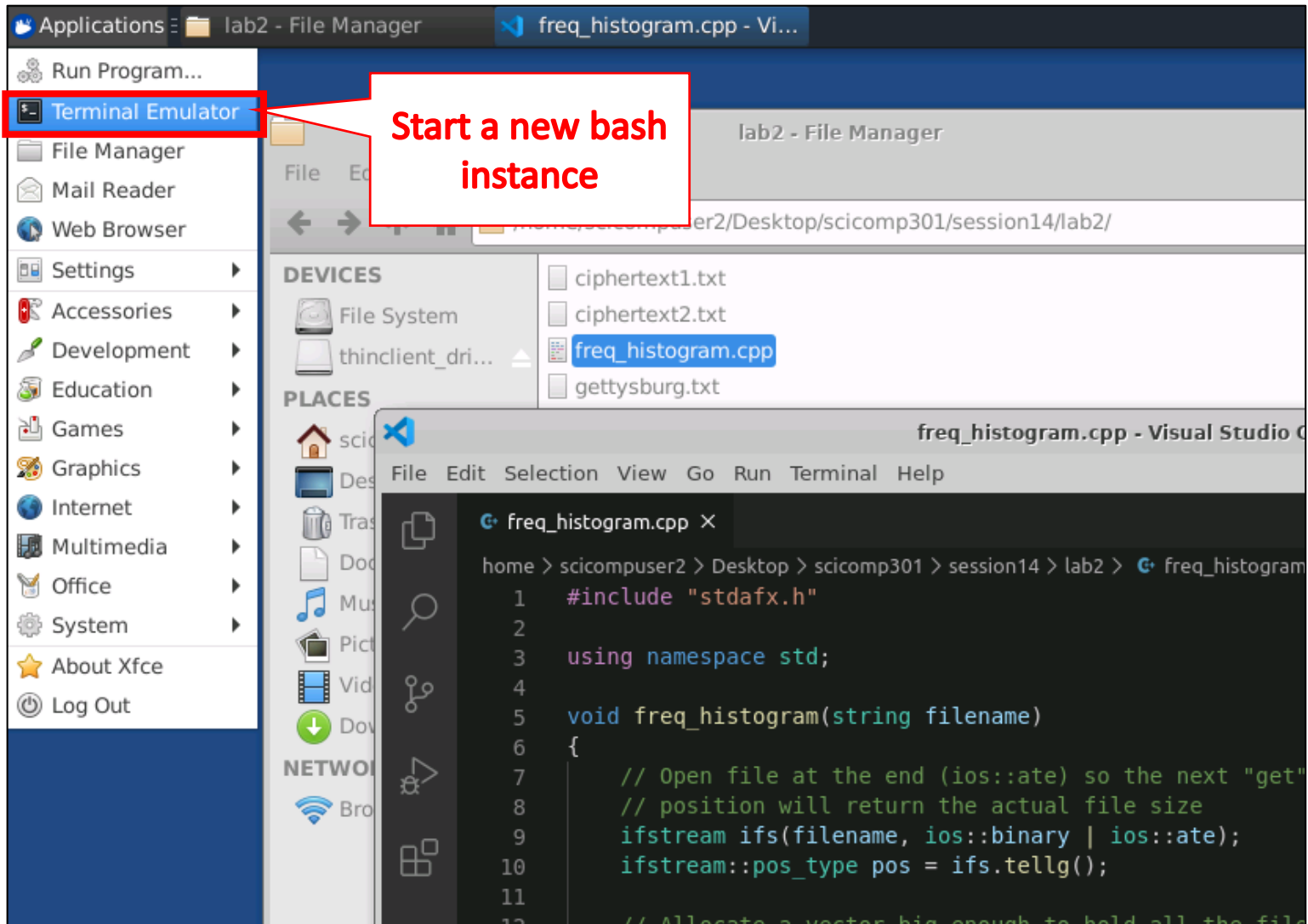
// Label any bin with the ASCII value
// if the bin count is > 6% of the file size,
// as these would be noteworthy occurrences
for (int i{1}; i < xa->GetNbins(); ++i)
    if (h1->GetBinContent(i) > fileBytes.size() * 0.06)
        xa->SetBinLabel(i, to_string(i).c_str());

h1->Draw();
}
```

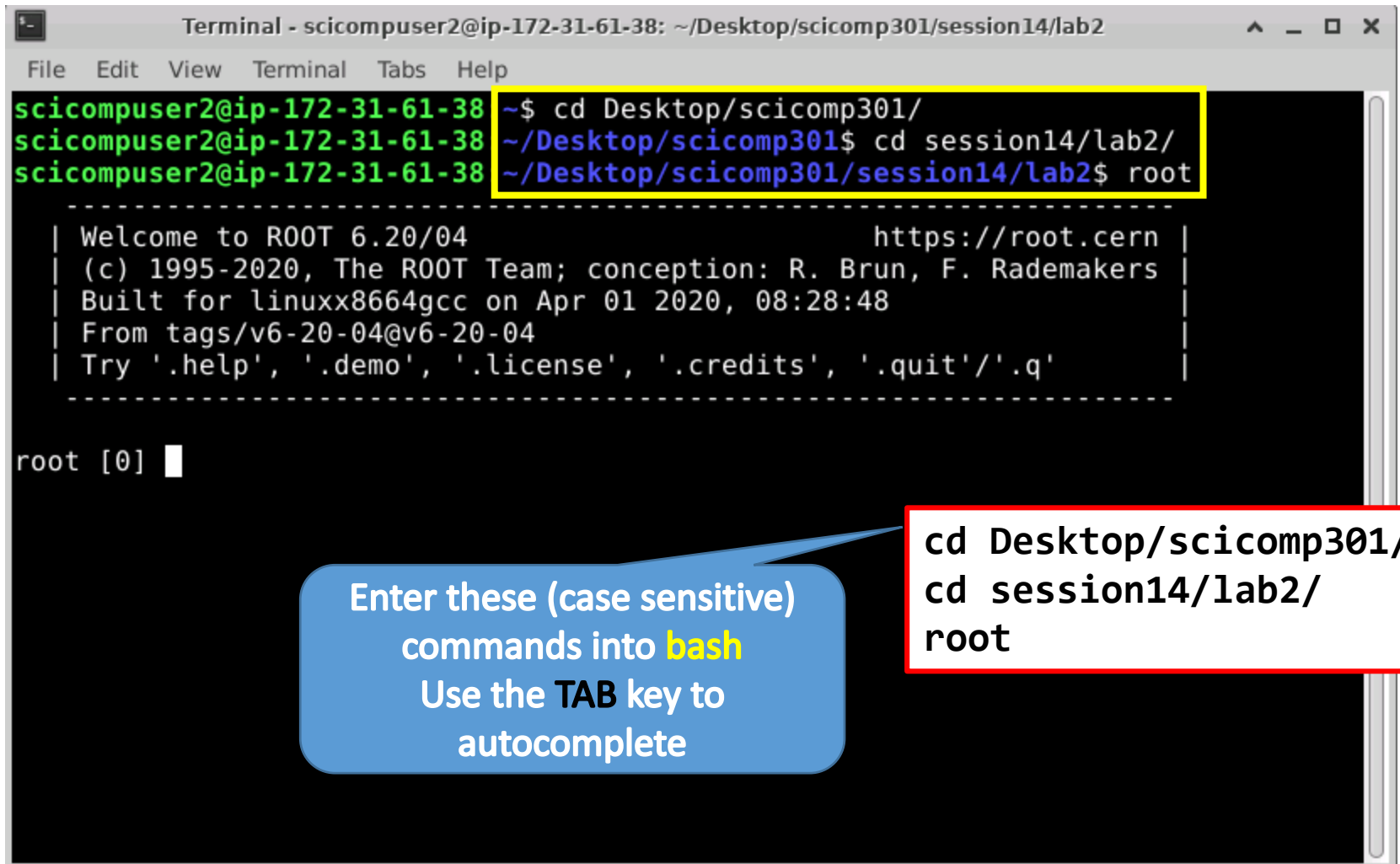
A **ROOT** histogram can automatically tally every occurrence in a vector

Only label ASCII values that occur more than 6% in the text file

Run Lab 2: freq_histogram.cpp



Run Lab 2: freq_histogram.cpp



```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session14/lab2
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38 ~$ cd Desktop/scicomp301/
scicompuser2@ip-172-31-61-38 ~/Desktop/scicomp301$ cd session14/lab2/
scicompuser2@ip-172-31-61-38 ~/Desktop/scicomp301/session14/lab2$ root

-----
| Welcome to ROOT 6.20/04                                     https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Apr 01 2020, 08:28:48             |
| From tags/v6-20-04@v6-20-04                                 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'  |
|-----|

root [0] █
```

Enter these (case sensitive) commands into **bash**
Use the **TAB** key to autocomplete

cd Desktop/scicomp301/
cd session14/lab2/
root

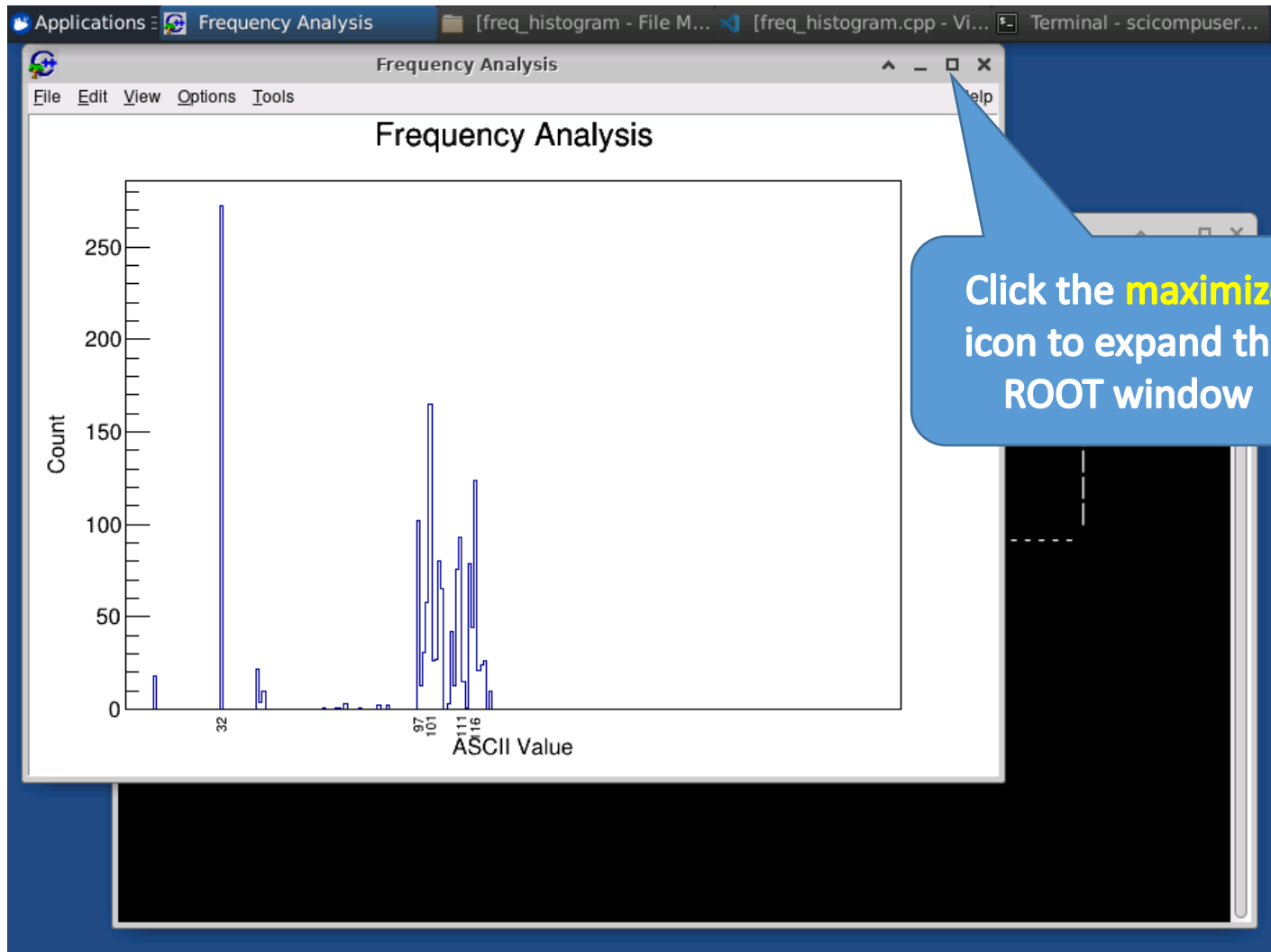
Run Lab 2: freq_histogram.cpp

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session14/lab2
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301$ cd session14/lab2/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session14/lab2$ root
-----
| Welcome to ROOT 6.20/04                                     https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Apr 01 2020, 08:28:48             |
| From tags/v6-20-04@v6-20-04                                 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.'q' |
|-----|
root [0] .x freq_histogram.cpp("gettysburg.txt")
root [1]
```

Enter this (case sensitive)
command into **ROOT**

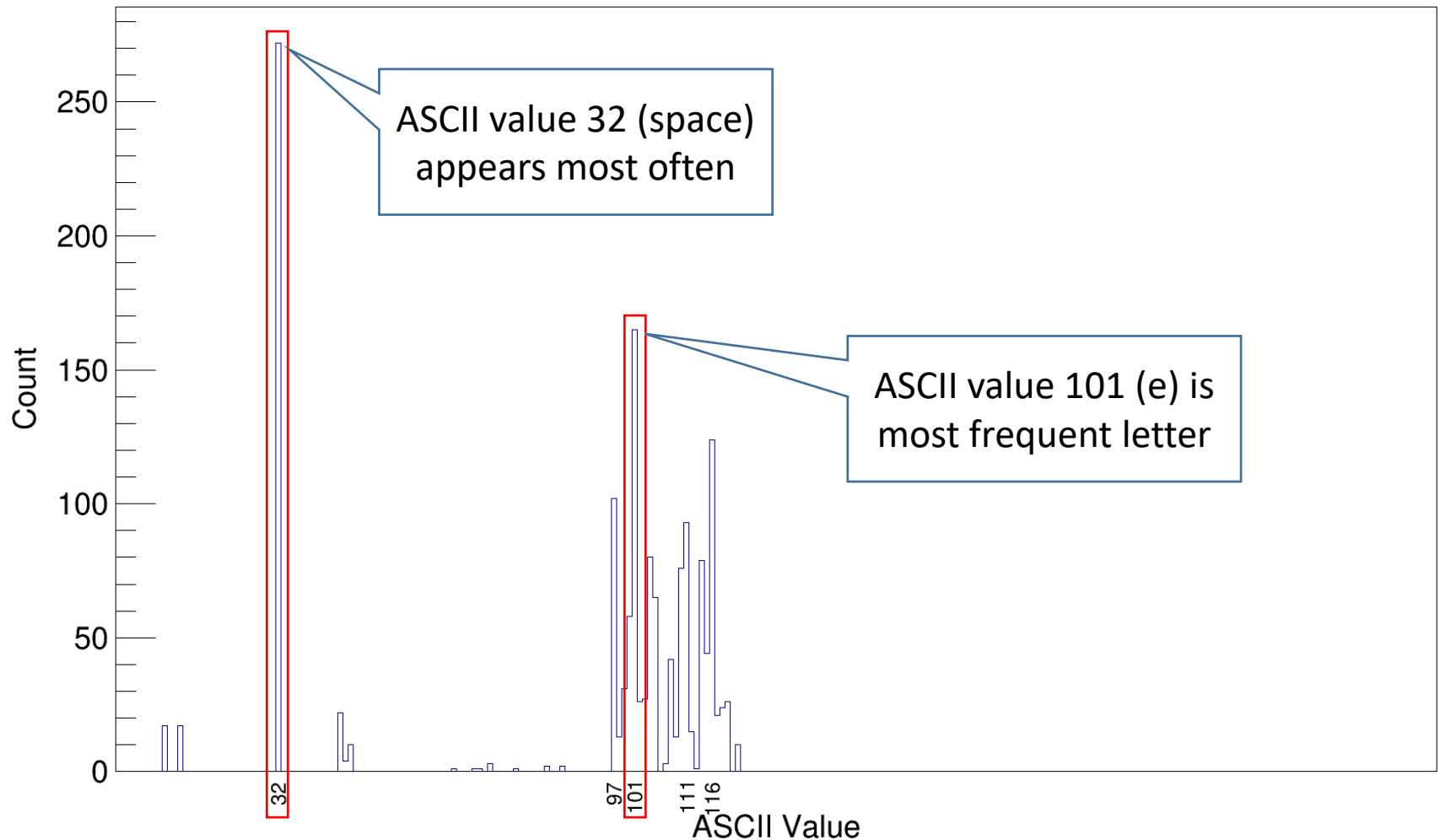
```
.x freq_histogram.cpp("gettysburg.txt")
```

Check Lab 2: freq_histogram.cpp

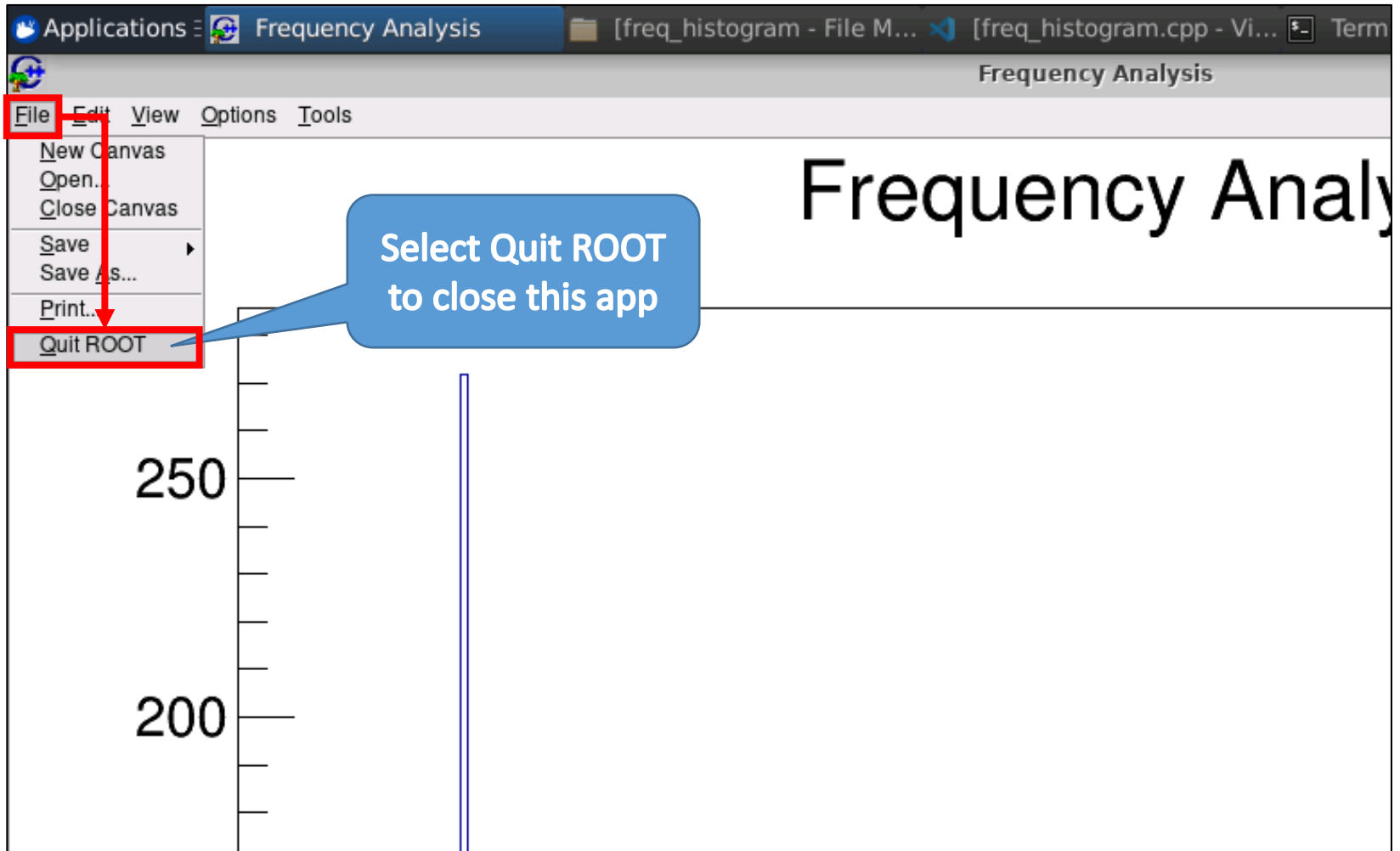


Histogram of Lincoln's Gettysburg Address

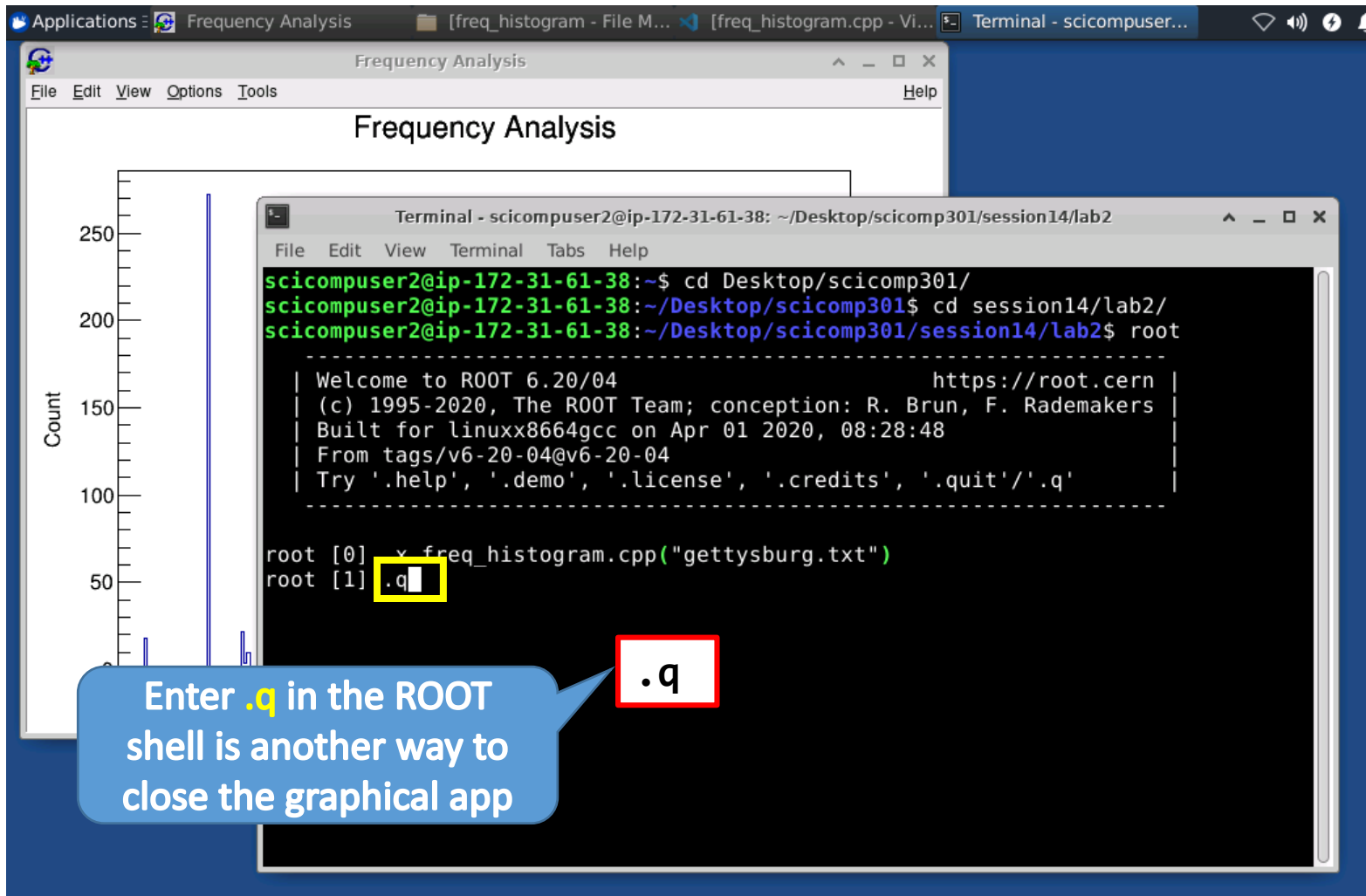
Frequency Analysis



Stop Lab 2: freq_histogram.cpp



Stop Lab 2: freq_histogram.cpp



The screenshot displays a desktop environment with a 'Frequency Analysis' window and a terminal window. The 'Frequency Analysis' window shows a histogram with a y-axis labeled 'Count' ranging from 0 to 250. The terminal window, titled 'Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session14/lab2', shows the following commands and output:

```
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301$ cd session14/lab2/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session14/lab2$ root

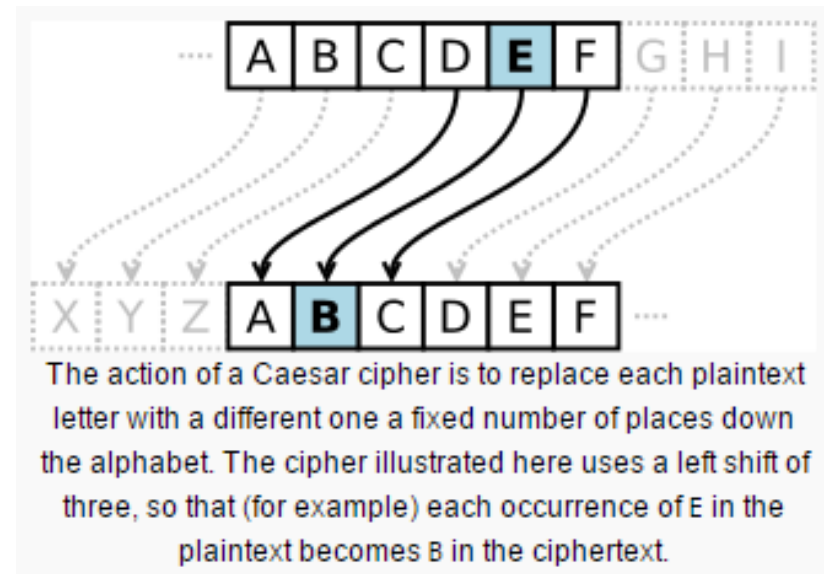
-----
| Welcome to ROOT 6.20/04                                     https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Apr 01 2020, 08:28:48             |
| From tags/v6-20-04@v6-20-04                                 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
-----

root [0] x freq_histogram.cpp("gettysburg.txt")
root [1] .q
```

A blue callout bubble points to the '.q' command in the terminal, stating: "Enter .q in the ROOT shell is another way to close the graphical app". A red box highlights the '.q' command in the terminal.

The Caesar Shift Cipher

- Roman Emperor **Julius Caesar** used a simple (but effective for its time) encryption scheme for his private correspondence
- To create “**cipher text**” from “**plain text**” simply shift the original letters **forward** (*or backward*) a given number of letters in the alphabet
- To decrypt the message, simply **reverse the sign of the shift**



The Caesar Shift Cipher

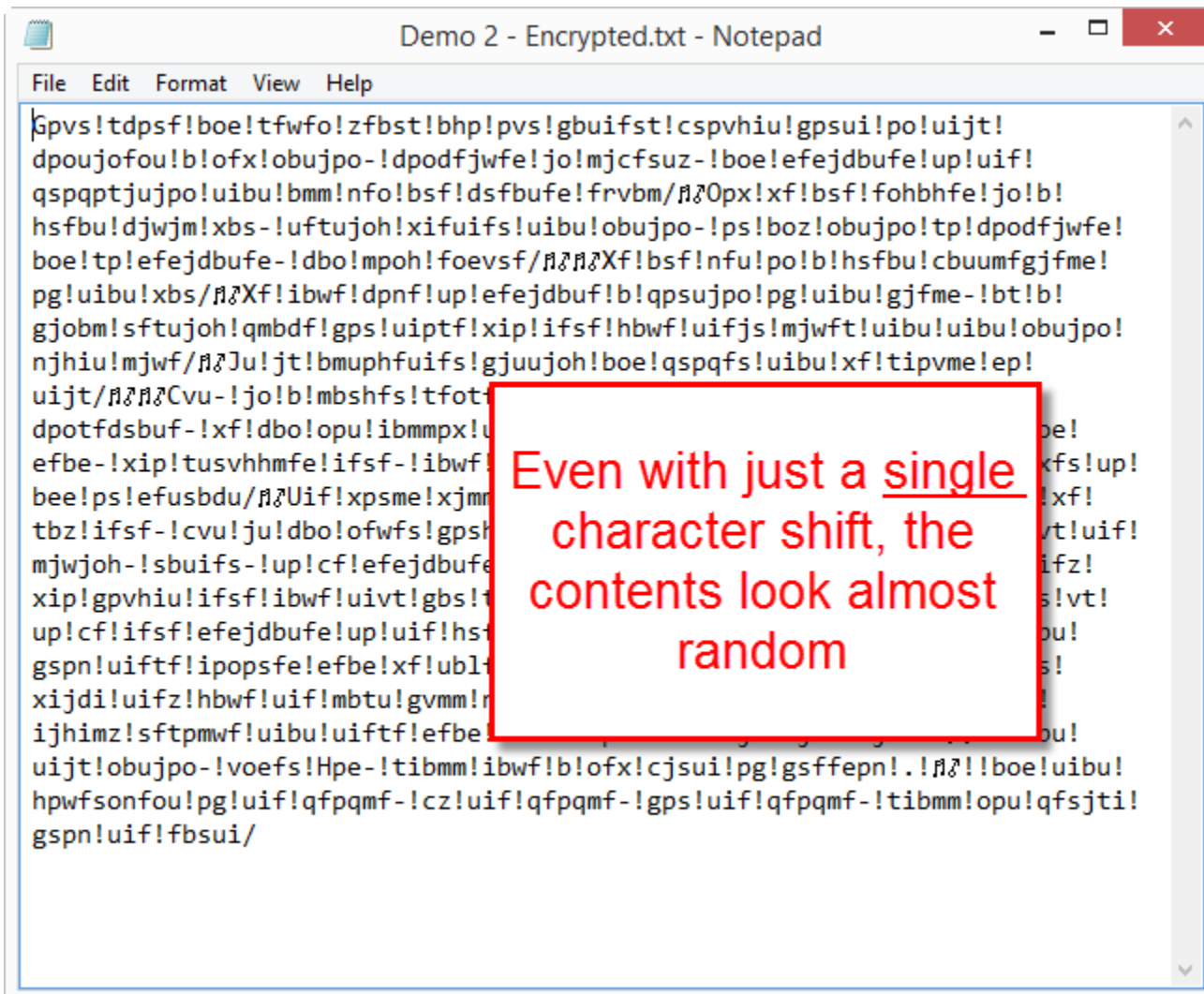
- Consider Lincoln's Gettysburg Address:

Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal.

Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure...

- How effective is the Caesar Shift at encrypting **plaintext**?
- What if we shifted each letter by just **one** position?
 - Simply add **+1** to the ASCII value of each plaintext character
 - Save the shifted values to a new **ciphertext** file

The Caesar Shift Cipher



Demo 2 - Encrypted.txt - Notepad

File Edit Format View Help

Spvs!tdpsf!boe!tfwfo!zfbst!bhp!pvs!gbuifst!cspvhiu!gpsui!po!uijt!
dpoujofou!b!ofx!obujpo-!dpodfjwfe!jo!mjcfesz-!boe!efejdbufe!up!uif!
qspqptjujpo!uibu!bmm!nfo!bsf!dsfbufe!frvbm/!Xf!bsf!fohbhfe!jo!b!
hsfbu!djwjm!xbs-!uftujoh!xifuijs!uibu!obujpo-!ps!boz!obujpo!tp!dpodfjwfe!
boe!tp!efejdbufe-!dbo!mpoh!foevsf/!Xf!bsf!nfu!po!b!hsfbu!cbuumfgjme!
pg!uibu!xbs/!Xf!ibwfi!dpnf!up!efejdbuf!b!qpsujpo!pg!uibu!gjme-!bt!b!
gjobm!sftujoh!qmbdf!gps!uiptf!xip!ifsf!hbwf!uifjs!mjwft!uibu!uibu!obujpo!
njhiu!mjwf/!Ju!jt!bmuphfuijs!gjuujoh!boe!qspqfs!uibu!xf!tipvme!ep!
uijt/!Cvu-!jo!b!mbshfs!tfot
dpotfdsbuf-!xf!dbo!opu!ibmmpx!
efbe-!xip!tusvhhmfe!ifsf-!ibwfi
bee!ps!efusbdu/!Uif!xpsme!xjmm
tbz!ifsf-!cvu!ju!dbo!ofwfs!gps!
mjwjoh-!sbuijs-!up!cf!efejdbufe
xip!gpvhiu!ifsf!ibwfi!uivt!gbs!
up!cf!ifsf!efejdbufe!up!uif!hsf
gspn!uiftf!ipopsfe!efbe!xf!ublf
xijdi!uifz!hbwf!uif!mbtu!gvmm!
ijhimz!sftpmwf!uibu!uiftf!efbe
uijt!obujpo-!voefs!Hpe-!tibmm!ibwfi!b!ofx!cjsui!pg!gsffepn!.!boe!uibu!
hpwfonfou!pg!uif!qfpqmf-!cz!uif!qfpqmf-!gps!uif!qfpqmf-!tibmm!opu!qfsjti!
gspn!uif!fbsui/

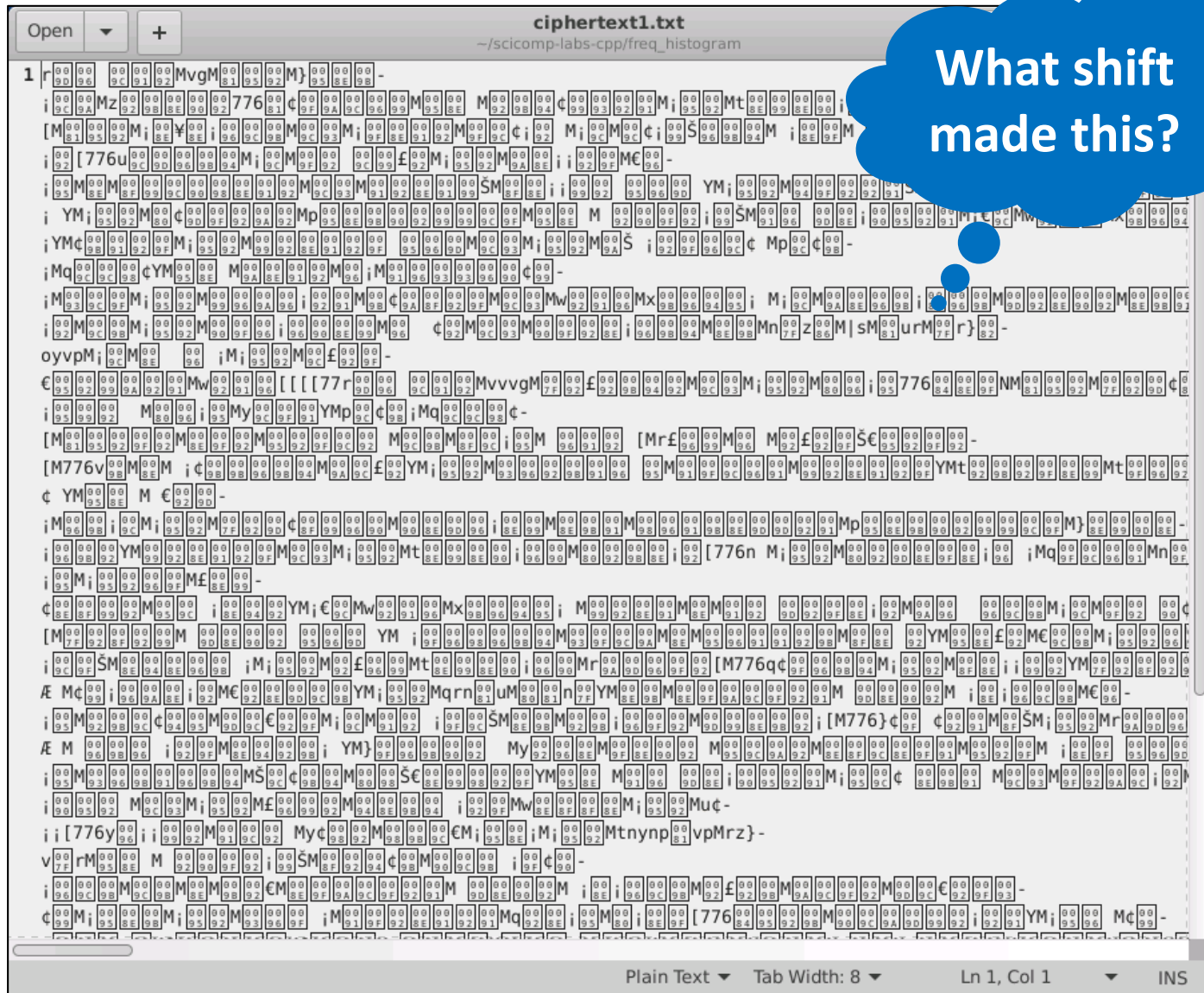
Even with just a single character shift, the contents look almost random

The Caesar Shift Cipher

- If we are given a text file written in **English** and encrypted by a **Caesar Shift**, can we figure out **what shift value** was used?
- We could use “**brute force**” and try every possible value to see what shift produces legible prose...
 - **But we'd need to possibly try every shift value from +1 to +255**
 - It would take a long time to sift through all the decrypted files because all incorrect **shift** values generate more *gibberish*
- Can we glean any insight from analyzing the **character histogram** of the encrypted file?

ciphertext1.txt

What shift made this?



Run Lab 2: freq_histogram.cpp

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session14/lab2
File Edit View Terminal Tabs Help

scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301$ cd session14/lab2/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session14/lab2$ root
-----
| Welcome to ROOT 6.20/04                                     https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Apr 01 2020, 08:28:48             |
| From tags/v6-20-04@v6-20-04                                 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.'q' |
|-----|

root [0] .x freq_histogram.cpp("ciphertext1.txt")
root [1] .q
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session14/lab2$ root
-----
| Welcome to ROOT 6.20/04                                     https://root.cern |
| (c) 1995-2020, The ROOT Team; conception: R. Brun, F. Rademakers |
| Built for linuxx8664gcc on Apr 01 2020, 08:28:48             |
| From tags/v6-20-04@v6-20-04                                 |
| Try '.help', '.demo', '.license', '.credits', '.quit'/'.'q' |
|-----|

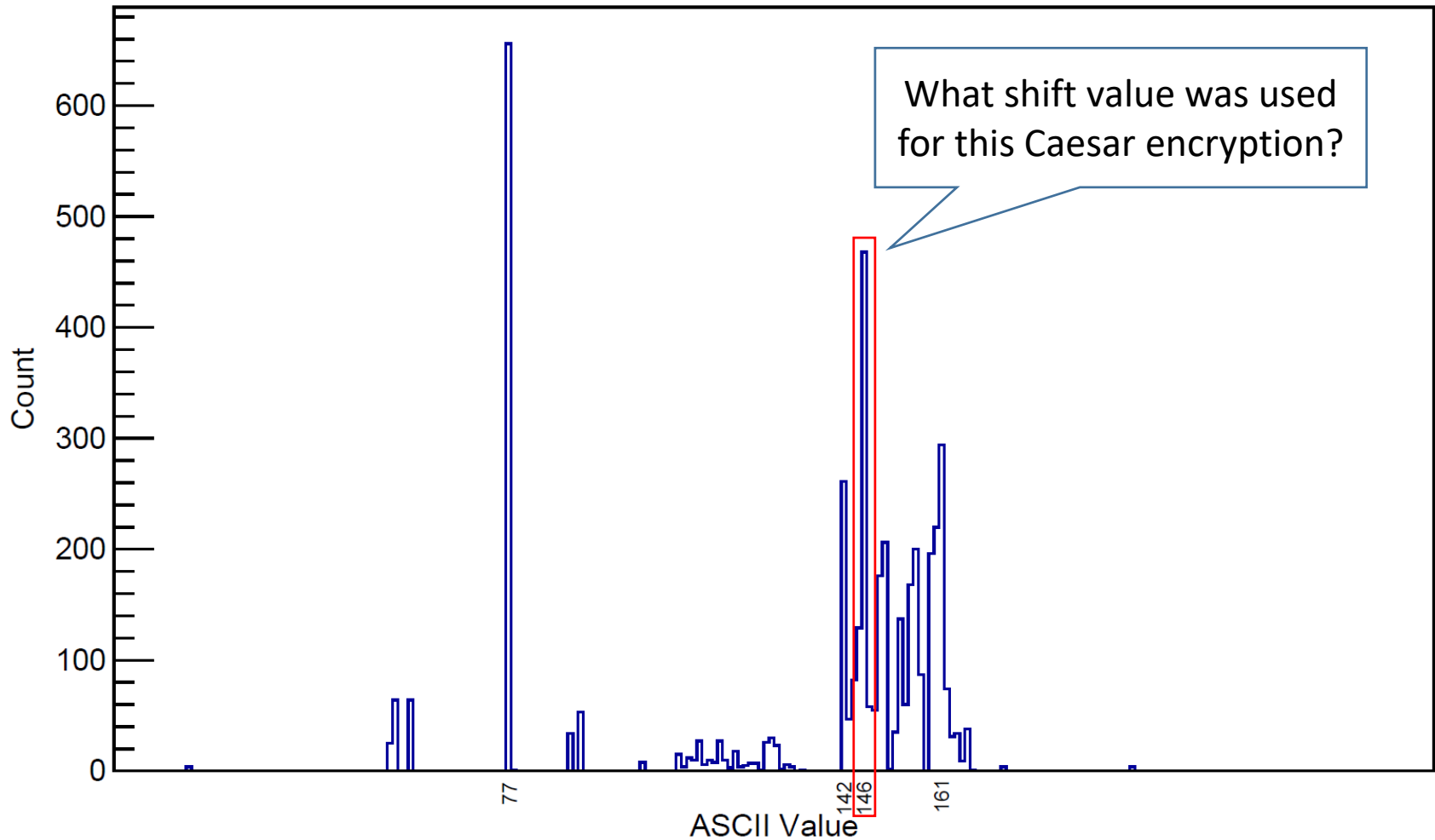
root [0] .x freq_histogram.cpp("ciphertext1.txt")
```

Enter this (case sensitive)
command into **ROOT**

```
.x freq_histogram.cpp("ciphertext1.txt")
```

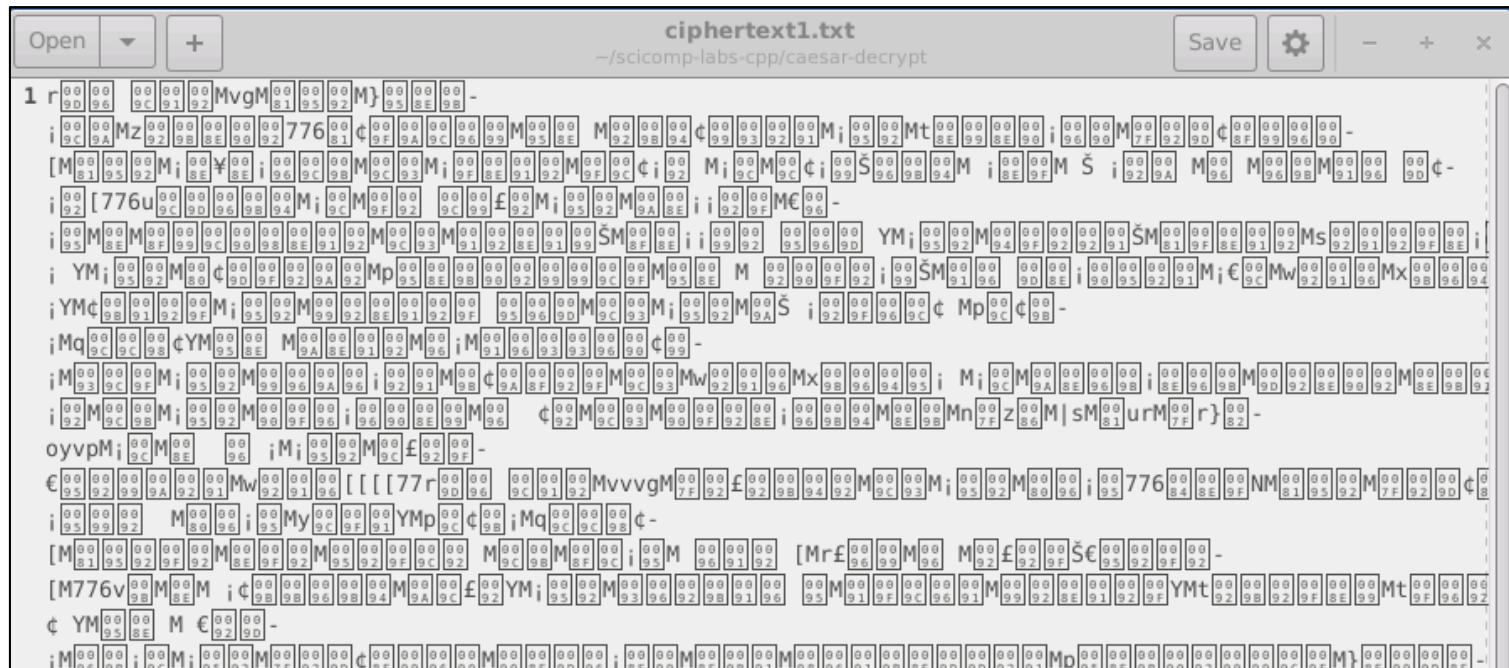

Histogram of Ciphertext #1

Frequency Analysis



Open Lab 3 – Caesar Decrypt

- **Your mission is to decrypt the ciphertext1.txt file**
- What if the survival of your country depended upon your ability to crack the encryption?



View Lab 3 – Caesar Decrypt

```
main.cpp [X]
1  #include "stdafx.h"
2
3  using namespace std;
4
5  int main()
6  {
7      // Open the ciphertext file
8      ifstream ifs("ciphertext1.txt", ios::binary | ios::ate);
9
10     // Read the input file straight into the buffer
11     ifstream::pos_type pos = ifs.tellg();
12     vector<unsigned char> buff(pos);
13     ifs.seekg(0, ios::beg);
14     ifs.read((char*)(buff.data()), pos);
15     ifs.close();
16
17     int shift = 0;
18
19     // Shift every character in cipher
20     for (auto c : buff)
21         cout << (char)(c + shift);
22
23     cout << endl << endl;
24
25     return 0;
26 }
27
```

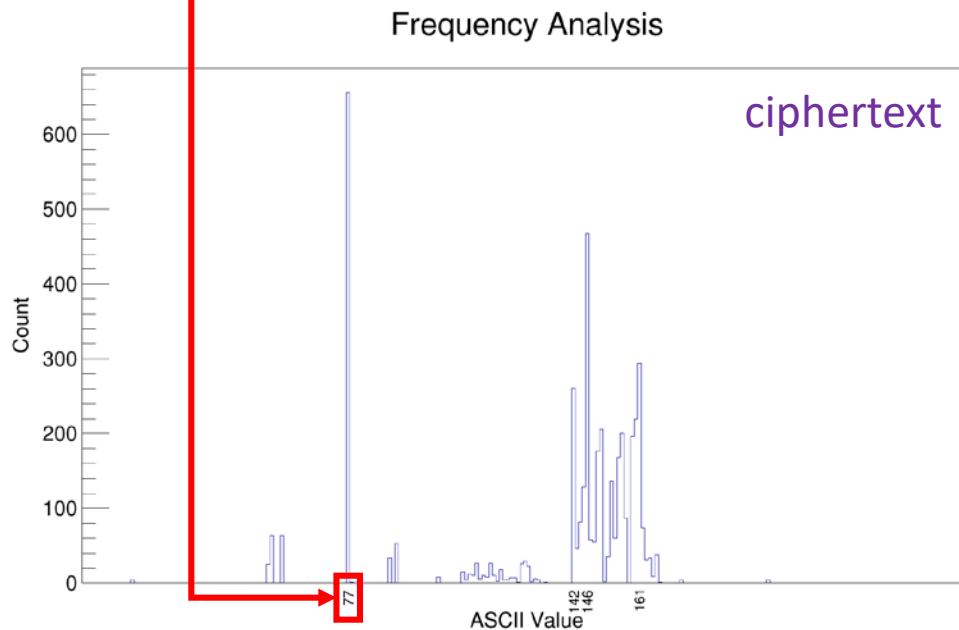
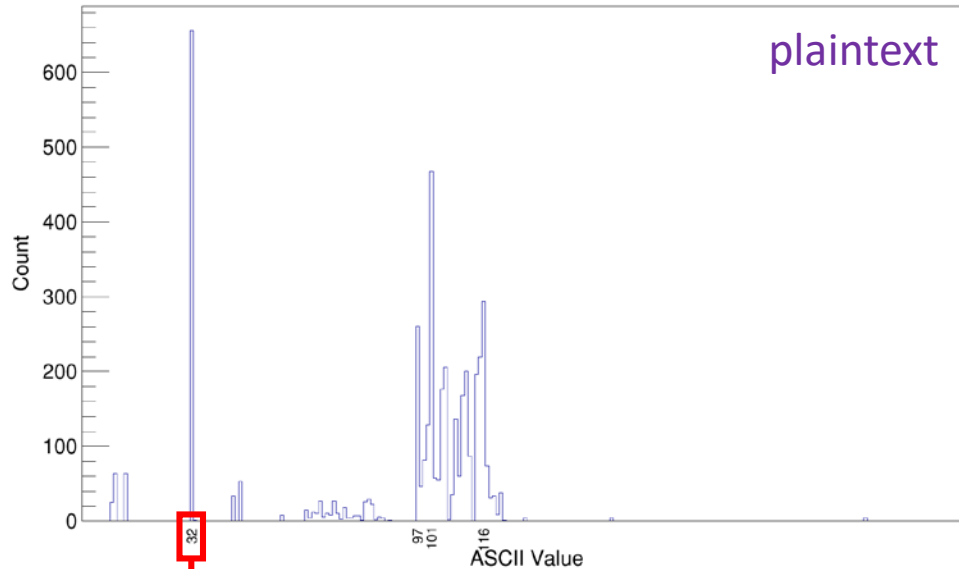
What will the output be if we leave **shift = 0** ?

Run Lab 3 – Caesar Decrypt

With shift = 0, the output text is just the same as the input text

Process returned 0 (0x0) execution time : 0.014 s
Press ENTER to continue.

Frequency Analysis



Notice the ***relative*** letter frequencies (individual bar heights) are the same before and after encryption

Edit Lab 3 – Caesar Decrypt

```
main.cpp [X]
1  #include "stdafx.h"
2
3  using namespace std;
4
5  int main()
6  {
7      // Open the ciphertext file
8      ifstream ifs("ciphertext1.txt", ios::binary | ios::ate);
9
10     // Read the input file straight into the buffer
11     ifstream::pos_type pos = ifs.tellg();
12     vector<unsigned char> buff(pos);
13     ifs.seekg(0, ios::beg);
14     ifs.read((char*)(buff.data()), pos);
15     ifs.close();
16
17     int shift = 0;
18
19     // Shift every character in cipher
20     for (auto c : buff)
21         cout << (char)(c + shift);
22
23     cout << endl << endl;
24
25     return 0;
26 }
27
```

What Caesar shift value will **reverse** the encryption that produced **ciphertext1.txt** ?

Check Lab 3 – Caesar Decrypt

Episode IX

THE RETURN OF CAESAR

Caesar speaks! A team of brave scientists use the tools of scientific computing to decipher a cryptic message in a desperate race to defend their nation...

The Caesar Shift Cipher

Because the **Caesar Shift** is a **monoalphabetic substitution cipher**, it is susceptible to **cryptanalysis** (breaking) by **frequency analysis**

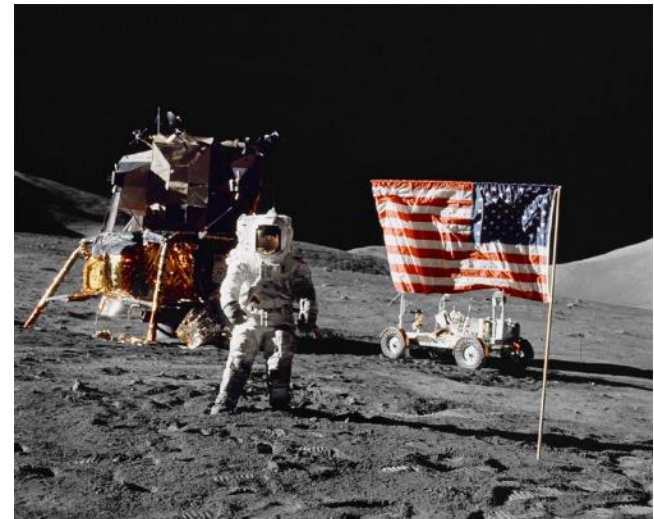


Bigram Analysis

- Most Western (Latin influenced) languages have a unique fingerprint from their most frequent **bigrams** (two-letter pair)
 - In **English** the bigrams **TH** and **HE** occur most often, since “the” is the most frequent word in English
 - All languages have definite articles like “the” but each language spells theirs differently, and this helps establish a **distinct statistical profile** for each language
- Linguists and statisticians have compiled lists of the most frequent **bigrams** *per* language
- We will analyze the bigrams in **President Kennedy’s Rice University Speech** - where he set the goal in 1962 for the USA to reach the moon before 1970!

Kennedy's Moon Speech in 1962

“We choose to go to the moon in this decade and do the other things, not because they are easy, *but because they are hard*, because that goal will serve to organize and measure **the best of our energies and skills**, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one which we intend to win.”



Bigram Analysis of Kennedy's Moon Speech (English)

```
freq-bigrams
File Edit View Terminal Tabs Help
Most recurring bigrams in file:
Kennedy Moon English.txt

ASCII      CHARS      FREQ
(65, 76)    AL         1.13
(65, 78)    AN         1.79
(65, 83)    AS         1.13
(65, 84)    AT         1.59
(69, 65)    EA         1.53
(69, 78)    EN         1.53
(69, 82)    ER         1.66
(69, 83)    ES         1.46
(69, 84)    ET         1.72
(72, 65)    HA         1.33
(72, 69)    HE         2.32
(73, 78)    IN         1.53
(76, 69)    LE         0.99
(76, 76)    LL         0.99
(78, 68)    ND         1.13
(78, 69)    NE         0.99
(78, 71)    NG         1.33
(78, 84)    NT         0.99
(79, 78)    ON         1.33
(82, 69)    RE         1.53
(83, 84)    ST         1.53
(84, 69)    TE         0.99
(84, 72)    TH         3.38
(84, 79)    TO         1.66

Process returned 0 (0x0)   execution time : 0.023 s
Press ENTER to continue.
```

Kennedy's Moon Speech Translated

Nous choisissons d'aller sur la lune.
Nous choisissons d'aller sur la lune
dans cette décennie et de faire
d'autres choses, non pas parce qu'ils
sont faciles, mais parce qu'ils sont
difficiles, parce que ce but servira à
organiser et mesurer le meilleur de
nos énergies et de compétences
parce que ce défi est l'un
sommes prêts à accepter,
ne sommes pas disposés à
et celui qui nous avons l'intention
gagner.

Elegimos ir a la Luna. Elegimos ir a la
Luna en esta década y hacer las
otras cosas, no porque sean fáciles,
sino porque son difíciles, porque esa
meta servirá para organizar y medir
lo mejor de nuestras energías y
habilidades, porque ese desafío es
una que estamos dispuestos a

Wir wählen, um zum Mond zu fliegen. Wir
wählen, um zum Mond in diesem
Jahrzehnt zu gehen und die anderen
Dinge, nicht weil sie leicht sind, sondern
weil sie hart sind, denn das Ziel wird dazu
dienen, zu organisieren und zu messen,
das Beste aus unserer Energien und
Fähigkeiten, denn das ist eine
Herausforderung dass wir bereit sind zu
akzeptieren, das wir nicht bereit sind, zu
verschieben, und eine, die wir
beabsichtigen, zu gewinnen.

Bigram Statistics by Language

Bigrams - Kennedy Speech

= Unique Indicators (for each language)
 = Relative Indicator (see German)

English		Speech
TH	2.71	3.38
HE	2.33	2.32
IN	2.03	1.53
ER	1.78	1.66
AN	1.61	1.79
RE	1.41	
ES	1.32	
ON	1.32	
ST	1.25	
NT	1.17	
EN	1.13	
AT	1.12	

Top 5: 10.46 10.68

Spanish		Speech
DE	2.57	2.41
ES	2.31	2.84
EN	2.27	1.75
EL	2.01	1.57
LA	1.80	1.69
OS	1.79	
ON	1.61	
AS	1.56	
ER	1.52	
RA	1.47	
AD	1.43	
AR	1.43	

Top 5: 10.96 10.26

French		Speech
ES	2.91	2.17
LE	2.08	2.17
DE	2.02	2.11
EN	1.97	1.61
ON	1.70	2.00
NT	1.69	
RE	1.62	
AN	1.28	
LA	1.25	
ER	1.21	
TE	1.19	
EL	1.15	

Top 5: 10.68 10.06

German		Speech
ER	3.90	3.29
EN	3.61	4.44
CH	2.36	1.67
DE	2.31	1.90
EI	1.98	1.73
TE	1.98	
IN	1.71	
ND	1.68	
IE	1.48	
GE	1.45	
ST	1.21	
NE	1.19	

Top 5: 14.16 13.03

German is the most consistent language for bigrams

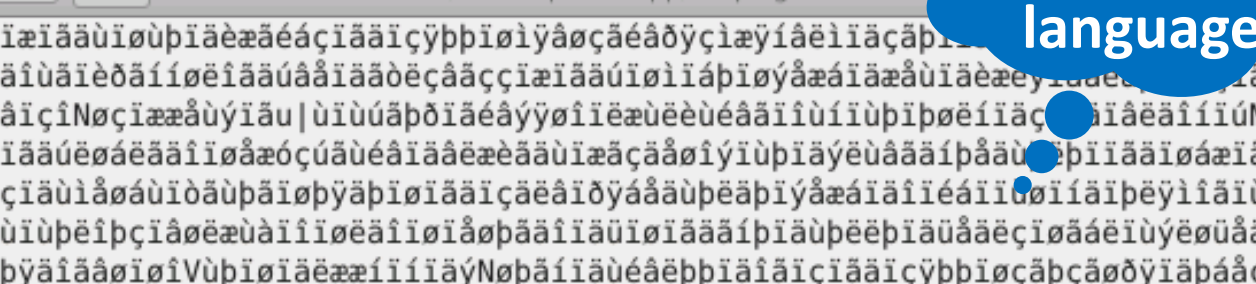
Lab 4 – Frequency of Bigrams

- You have been given another encrypted message
- It is encrypted with a **monoalphabetic substitution cipher**
 - Frequency analysis **does not** suggest a simple Caesar shift was used
 - Rather, a **different shift value** was used for each plaintext letter!
- Even if you are **unable** to break the encryption, **can you tell what language was used** when writing the plaintext?
 - Q: **Are you serious?** How can you possibly discern the author's language if you **cannot even read the contents** in the first place?
 - A: Enigma was also **unbreakable**



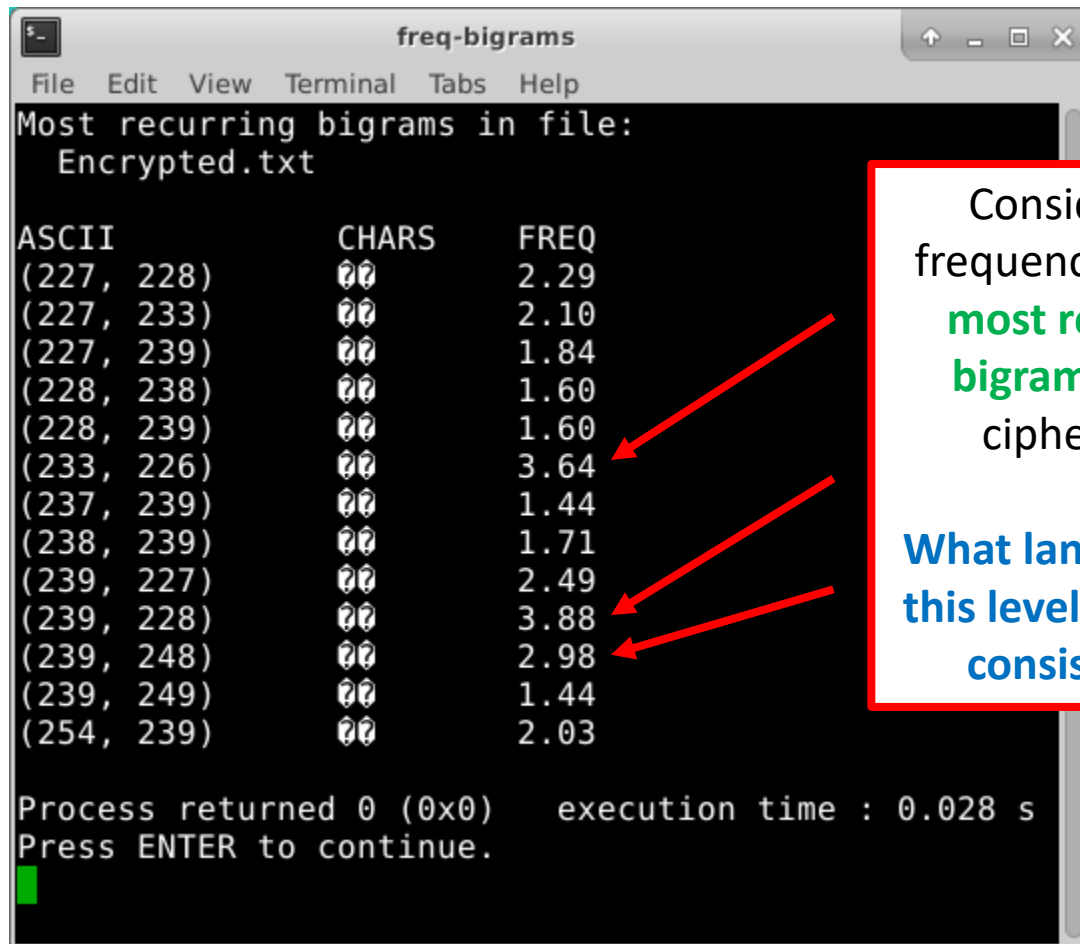
View Lab 4 – Encrypted.txt

What was the source language?



the source language?

Run Lab 4 – Frequency of Bigrams



```
freq-bigrams
File Edit View Terminal Tabs Help
Most recurring bigrams in file:
Encrypted.txt

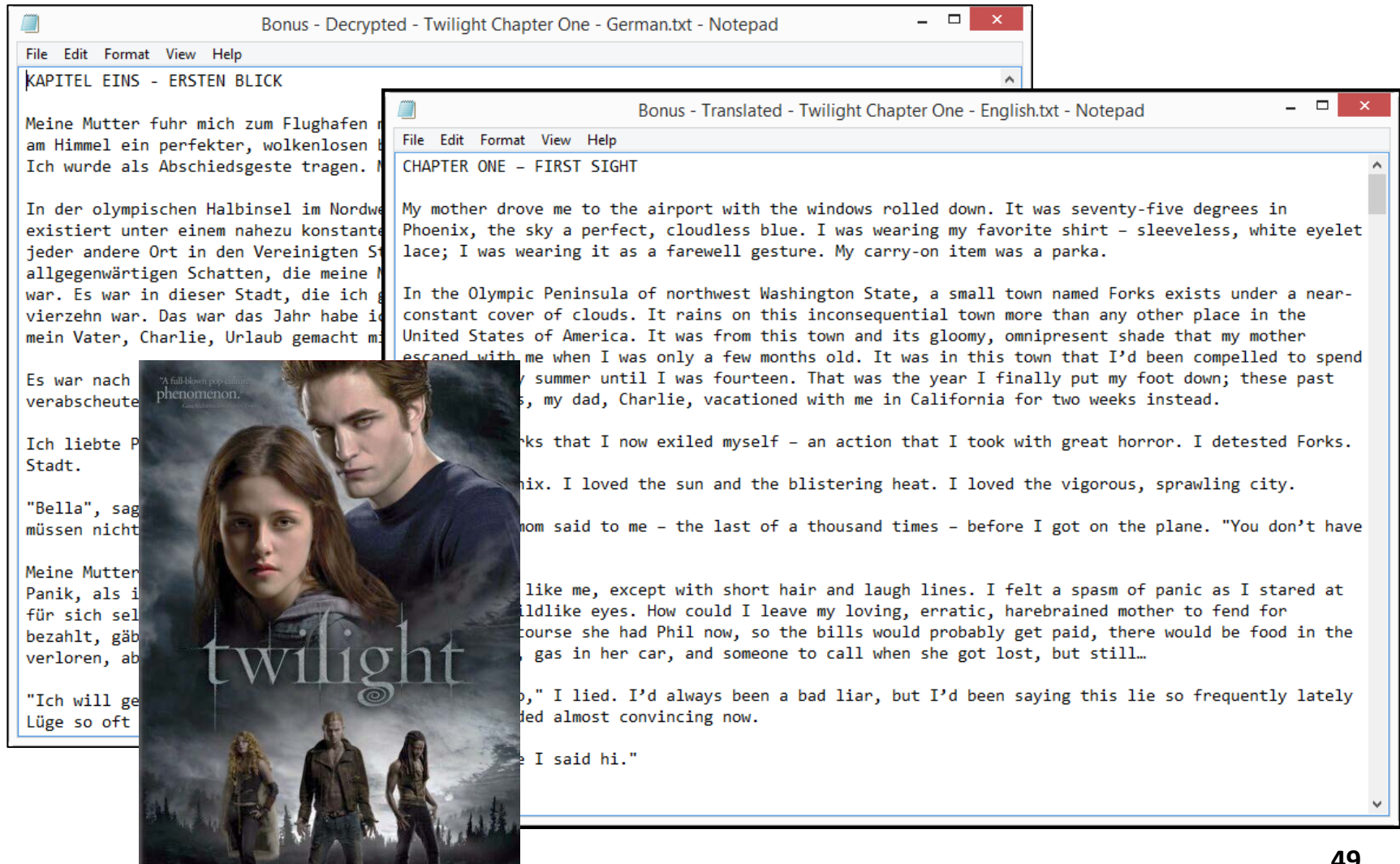
ASCII      CHARS      FREQ
(227, 228)  00        2.29
(227, 233)  00        2.10
(227, 239)  00        1.84
(228, 238)  00        1.60
(228, 239)  00        1.60
(233, 226)  00        3.64
(237, 239)  00        1.44
(238, 239)  00        1.71
(239, 227)  00        2.49
(239, 228)  00        3.88
(239, 248)  00        2.98
(239, 249)  00        1.44
(254, 239)  00        2.03

Process returned 0 (0x0)   execution time : 0.028 s
Press ENTER to continue.
█
```

Consider the frequencies of the **most recurring bigrams** in the cipher text.

What language has this level of bigram consistency?

Lab 4 – Plaintext



The image shows two overlapping Notepad windows. The background window is titled "Bonus - Decrypted - Twilight Chapter One - German.txt - Notepad" and contains German text. The foreground window is titled "Bonus - Translated - Twilight Chapter One - English.txt - Notepad" and contains the English translation. A movie poster for "Twilight" is overlaid on the bottom left of the English window.

German Window (Background):

File Edit Format View Help

KAPITEL EINS - ERSTEN BLICK

Meine Mutter fuhr mich zum Flughafen... am Himmel ein perfekter, wolkenloser... Ich wurde als Abschiedsgeste tragen.

In der olympischen Halbinsel im Nordwe... existiert unter einem nahezu konstante... jeder andere Ort in den Vereinigten St... allgegenwärtigen Schatten, die meine M... war. Es war in dieser Stadt, die ich g... vierzehn war. Das war das Jahr habe i... mein Vater, Charlie, Urlaub gemacht m...

Es war nach... verabscheute...

Ich liebte P... Stadt.

"Bella", sag... müssen nicht...

Meine Mutter... Panik, als i... für sich sel... bezahlt, gäb... verloren, ab...

"Ich will ge... Lüge so oft...

English Window (Foreground):

File Edit Format View Help

CHAPTER ONE – FIRST SIGHT

My mother drove me to the airport with the windows rolled down. It was seventy-five degrees in Phoenix, the sky a perfect, cloudless blue. I was wearing my favorite shirt – sleeveless, white eyelet lace; I was wearing it as a farewell gesture. My carry-on item was a parka.

In the Olympic Peninsula of northwest Washington State, a small town named Forks exists under a near-constant cover of clouds. It rains on this inconsequential town more than any other place in the United States of America. It was from this town and its gloomy, omnipresent shade that my mother escaped with me when I was only a few months old. It was in this town that I'd been compelled to spend my summer until I was fourteen. That was the year I finally put my foot down; these past years, my dad, Charlie, vacationed with me in California for two weeks instead.

Forks that I now exiled myself – an action that I took with great horror. I detested Forks. Phoenix. I loved the sun and the blistering heat. I loved the vigorous, sprawling city.

mom said to me – the last of a thousand times – before I got on the plane. "You don't have to like me, except with short hair and laugh lines. I felt a spasm of panic as I stared at wildlike eyes. How could I leave my loving, erratic, harebrained mother to fend for herself? Of course she had Phil now, so the bills would probably get paid, there would be food in the house, gas in her car, and someone to call when she got lost, but still...

"No," I lied. I'd always been a bad liar, but I'd been saying this lie so frequently lately lately it had almost convincing now.

... I said hi."

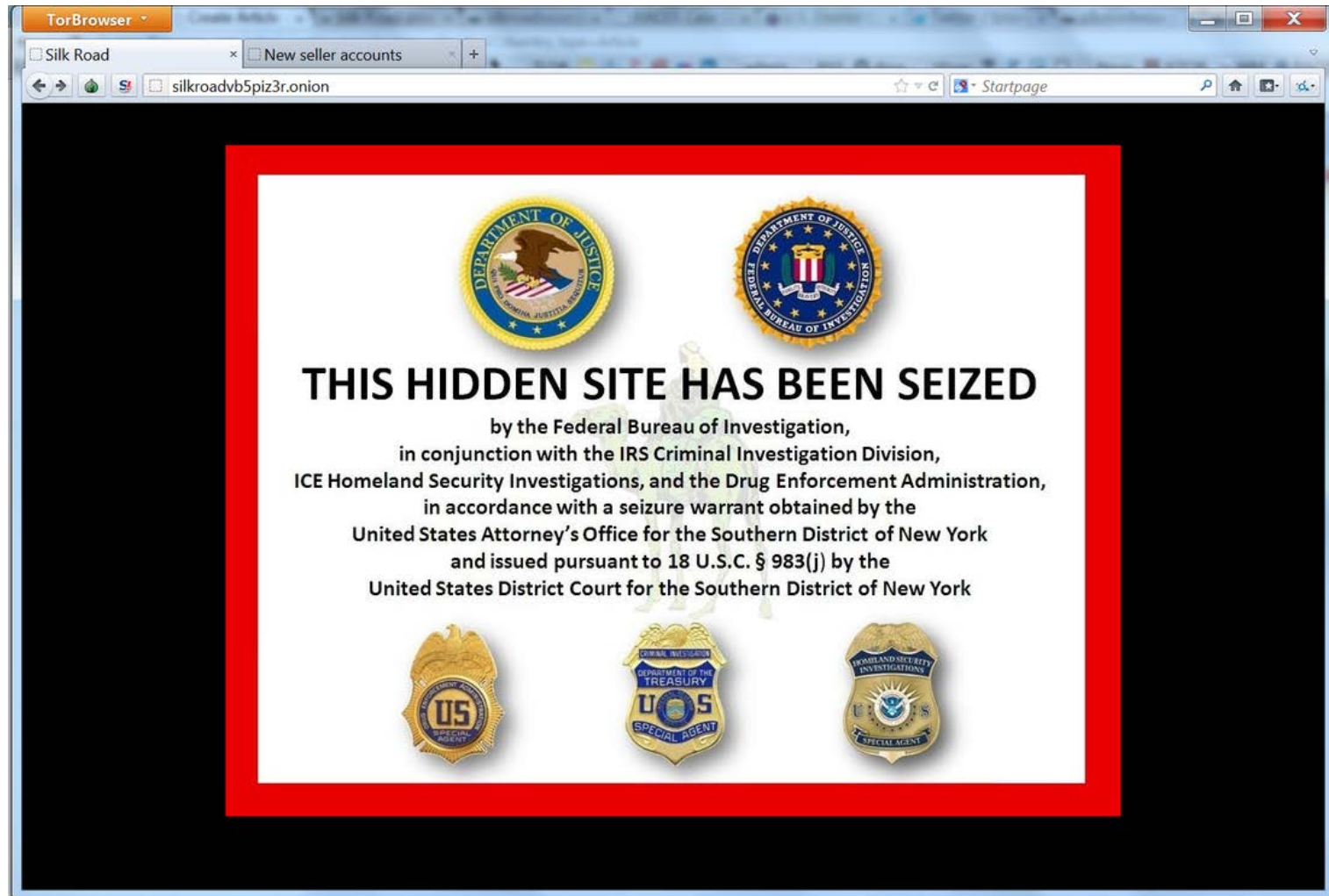
Twilight Movie Poster (Overlay):

A full-blown pop culture phenomenon.

twilight

The poster features Bella Swan and Edward Cullen, with other characters at the bottom.

The Fallacy of the Smarter Criminal



Anagrams

- Different words all spelled with the same set of letters are called **anagrams**
- Given an **English dictionary**, how could you **find all** the anagrams of a word?
- What algorithm would you use? **Trial and error?**

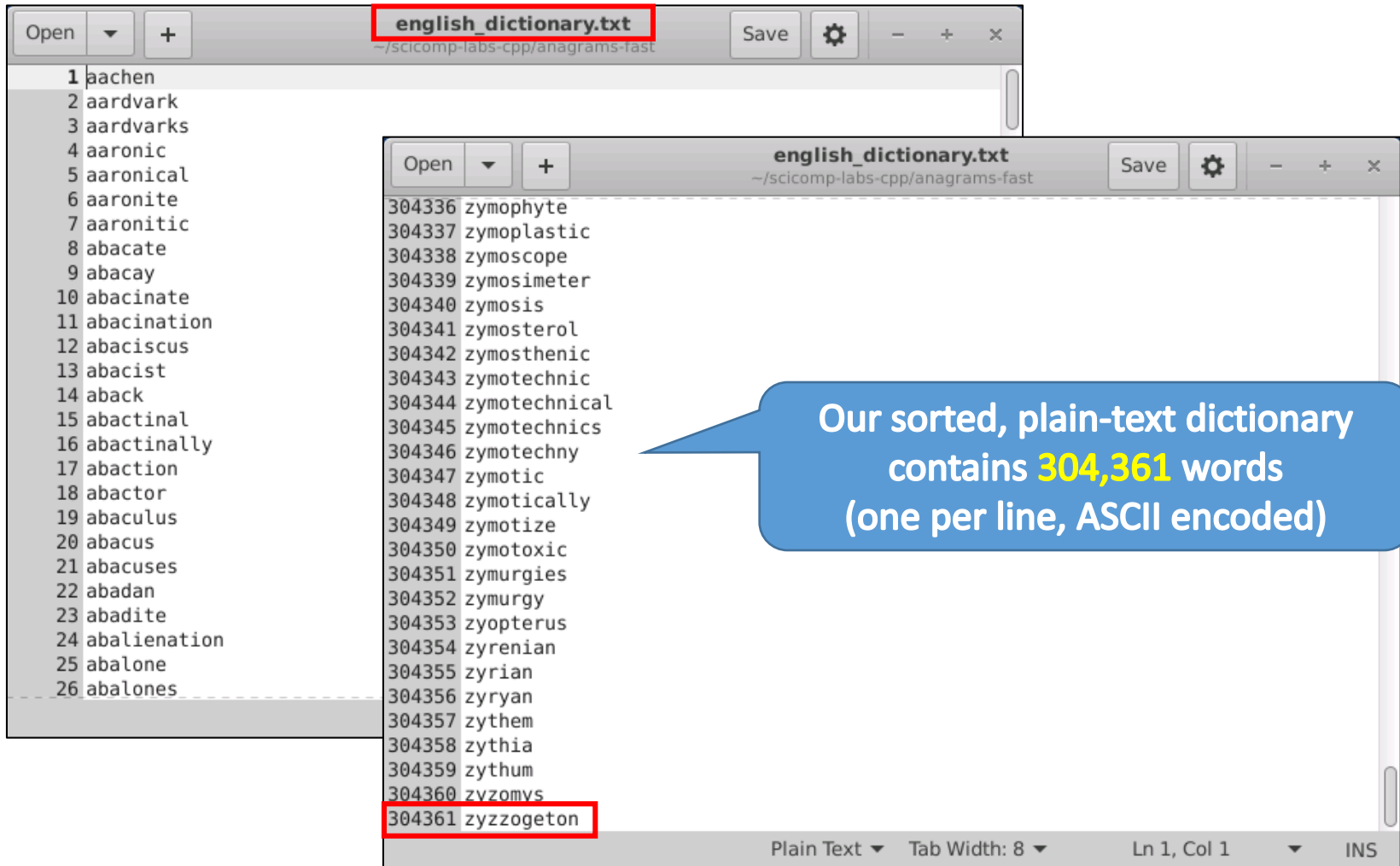
Word	Letters	Anagrams	Permutations
STOP	4	6	24
LEAST	5	10	120
TRACES	6	9	720
PLAYERS	7	7	5,040
RESTRAIN	8	6	40,320
MASTERING	9	4	362,880
SUPERSONIC	10	3	3,628,800

Anagrams

- stop = post, pots, spot, tops
- least = slate, stale, steal, tales
- traces = ???
- players = ???
- restrain = ???
- mastering = ???
- supersonic = ???

Naive Approach = Try every possible permutation of all given letters, checking in dictionary file to see if that permutation it is a valid English word

english_dictionary.txt



Open Lab 5 – Slow Anagrams

```
main.cpp [x]
1  #include "stdafx.h"
2
3  using namespace std;
4
5  vector<string> phrases
6  {
7      "Stop", "Least", "Traces", "Players", "Restrain"
8  };
9
10 vec
11 vec
12
49  int main()
50  {
51      // Read in the dictionary file
52      ifstream inputFile("english_dictionary.txt");
53      string line;
54      while (getline(inputFile, line))
55      {
56          boost::trim(line);
57          if (line.length() > 0)
58              dictionary.push_back(line);
59      }
60
61      // Start a timer
62      boost::timer timer;
63
64      cout << "Total search time = "
65           << fixed << setprecision(3)
66           << timer.elapsed() << " s" << endl;
```

BOOST Libraries for C++

<https://www.boost.org>

The screenshot shows the Boost Library Documentation website. The browser address bar displays `boost.org/doc/libs/?view=categorized`. The page header features the Boost logo and the text "C++ LIBRARIES". Below the header, the "BOOST LIBRARY DOCUMENTATION" title is followed by a red-bordered box labeled "ALGORITHMS". Under this box, a list of libraries is provided, including Algorithm, Foreach, Geometry, GIL, Graph, GraphParallel, Histogram, Min-Max, Polygon, QVM, Range, Sort, String Algo, and Utility. Another red-bordered box labeled "SYSTEM" is located below the algorithms list, containing a list of system-related libraries such as Chrono, Context, Date Time, DLL, Fiber, Filesystem, Nowide, Process, Stacktrace, System, and Thread.

BOOST LIBRARY DOCUMENTATION

ALGORITHMS

- **Algorithm:** A collection of useful generic algorithms.
- **Foreach:** In C++, writing a loop that iterates over a sequence is tedious. We can either use iterators, which requires a considerable amount of boiler-plate, or we can use the `std::for_each()` algorithm and move our loop body into a predicate, which requires no less boiler-plate.
- **Geometry:** The Boost.Geometry library provides geometric algorithms, primitives and spatial index.
- **GIL: (C++11) Generic Image Library**
- **Graph:** The BGL graph interface and graph components are generic, in the same sense as the Standard Template Library (STL).
- **GraphParallel:** The PBGL graph interface and graph components are generic, in the same sense as the the Standard Template Library.
- **Histogram:** Fast multi-dimensional histogram with convenient interface for C++14
- **Min-Max:** Standard library extensions for simultaneous min/max and min/max element computations.
- **Polygon:** Voronoi diagram construction and booleans/clipping, resizing/offsetting and more for planar polygons with integral coordinates.
- **QVM:** Generic {CPP} library for working with Quaternions Vectors and Matrices.
- **Range:** A new infrastructure for generic algorithms that builds on top of the new iterator concepts.
- **Sort:** High-performance templated sort functions.
- **String Algo:** String algorithms library.
- **Utility:** Class noncopyable plus `checked_delete()`, `checked_array_delete()`, `next()`, `prior()` function templates, plus base-from-member.

SYSTEM

- **Chrono:** Useful time utilities. C++11.
- **Context:** (C++11) Context switching library.
- **Date Time:** A set of date-time libraries based on generic programming concepts.
- **DLL:** Library for comfortable work with DLL and DSO.
- **Fiber:** (C++11) Userland threads library.
- **Filesystem:** The Boost Filesystem Library provides portable facilities to query and manipulate paths, files, and directories.
- **Nowide:** Standard library functions with UTF-8 API on Windows.
- **Process:** Library to create processes in a portable way.
- **Stacktrace:** Gather, store, copy and print backtraces.
- **System:** Operating system support, including the diagnostics support that will be part of the C++0x standard library.
- **Thread:** Portable C++ multi-threading. C++03, C++11, C++14, C++17.

Boost Software License

Latest version	1.0
Published	17 August 2003
FSF approved	Yes
OSI approved	Yes
GPL compatible	Yes
Copyleft	No
Linking from code with a different licence	Yes

View Lab 5

Slow Anagrams

```
// Find any anagrams for every requested phrase
for (auto& phrase : phrases) {

    // Convert phrase to all lowercase
    boost::to_lower(phrase);

    // Create vector of individual characters
    vector<char> letters;
    for (auto s : phrase)
        letters.push_back(s);

    // Add all permutations of letters to words vector
    words.clear();
    Permute<char>(&letters, letters.size());

    // Remove redundant permutations caused
    // by a phrase having duplicated letters
    sort(words.begin(), words.end());
    auto last = unique(words.begin(), words.end());
    words.erase(last, words.end());

    // Display only words that are found in the dictionary
    for (const auto& word : words)
        if (binary_search(dictionary.begin(),
                           dictionary.end(), word))
            cout << word << endl;

    cout << endl;
}
```


Binary Search

0	1	2	3	4	5	6	7	8	9	10
1	2	7	12	28	31	40	41	42	46	59

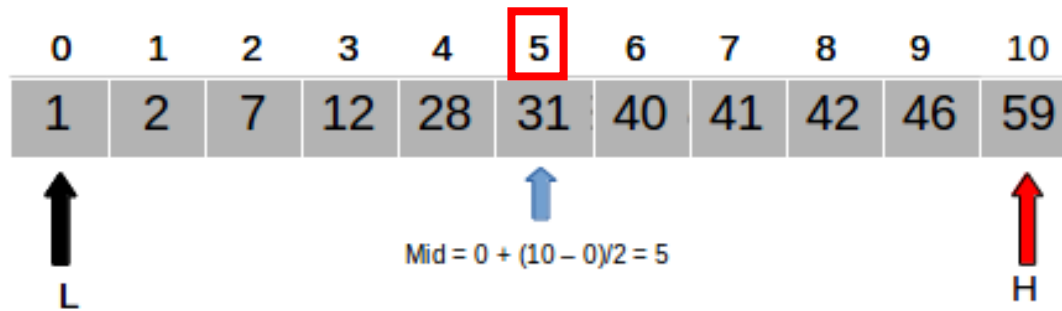
↑
L

↑
H

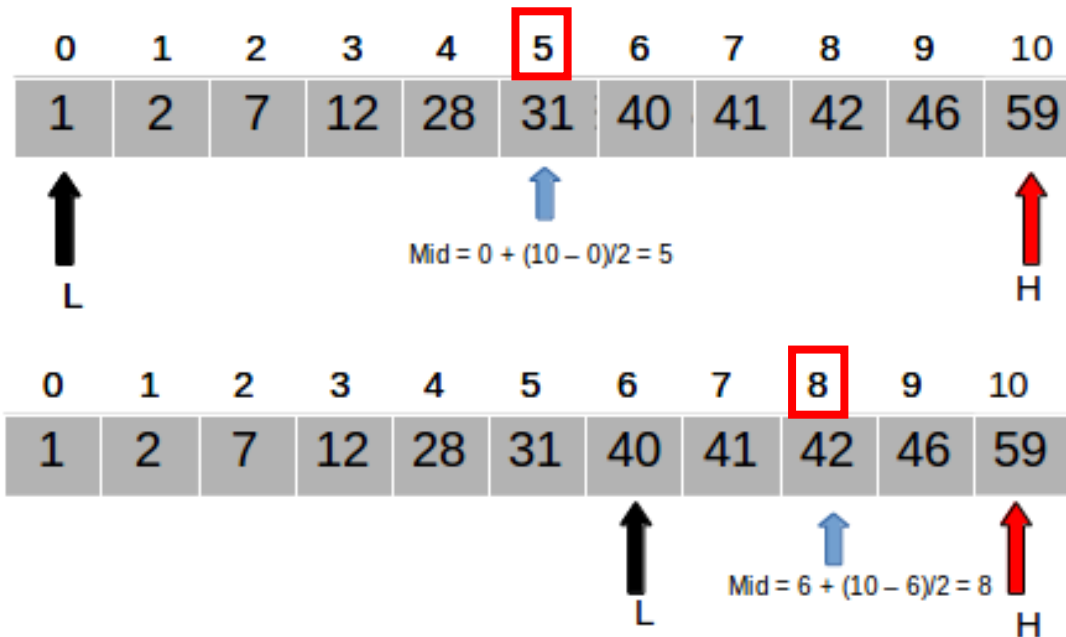
If the vector is
already **sorted**...

...what is the index
for element **46** ?

Binary Search

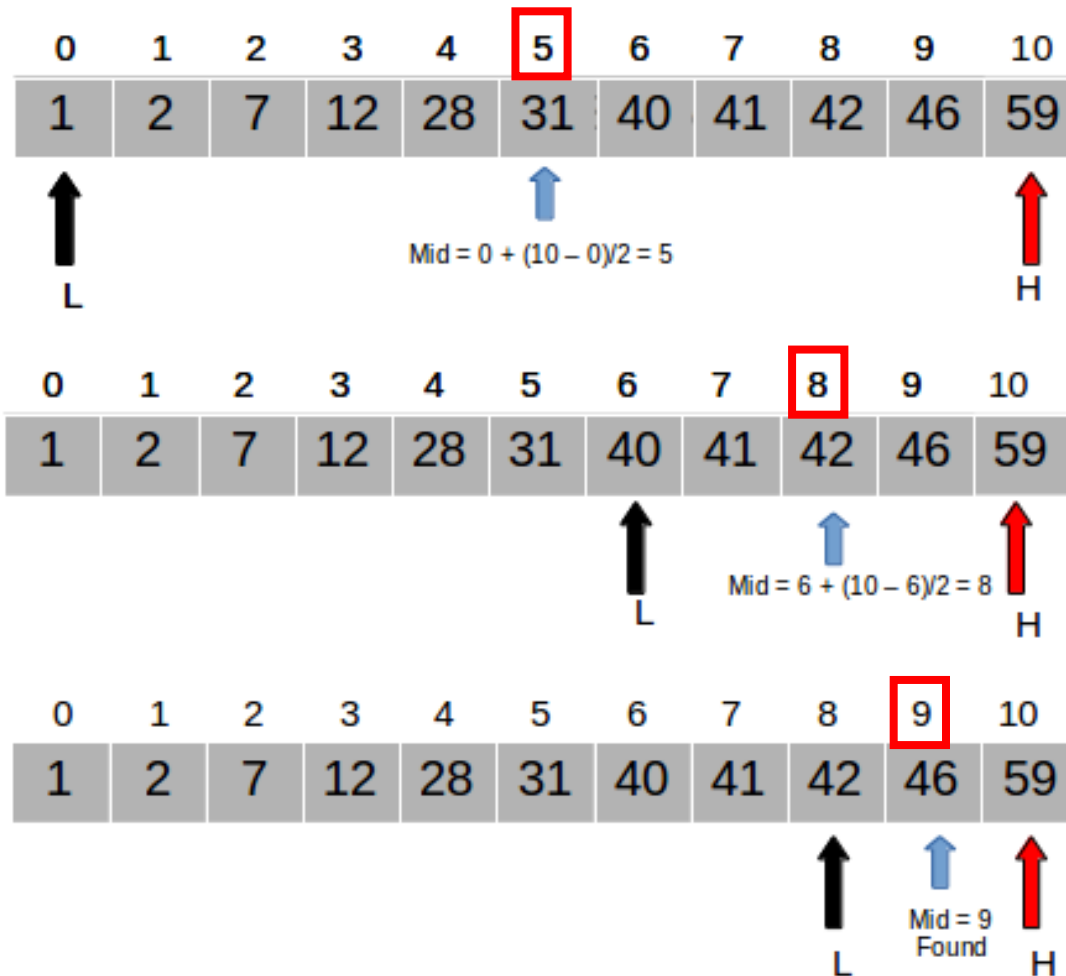


Binary Search



...what is the index
for element 46?

Binary Search



We had to check only **three** elements to find the **46**

View Lab 5

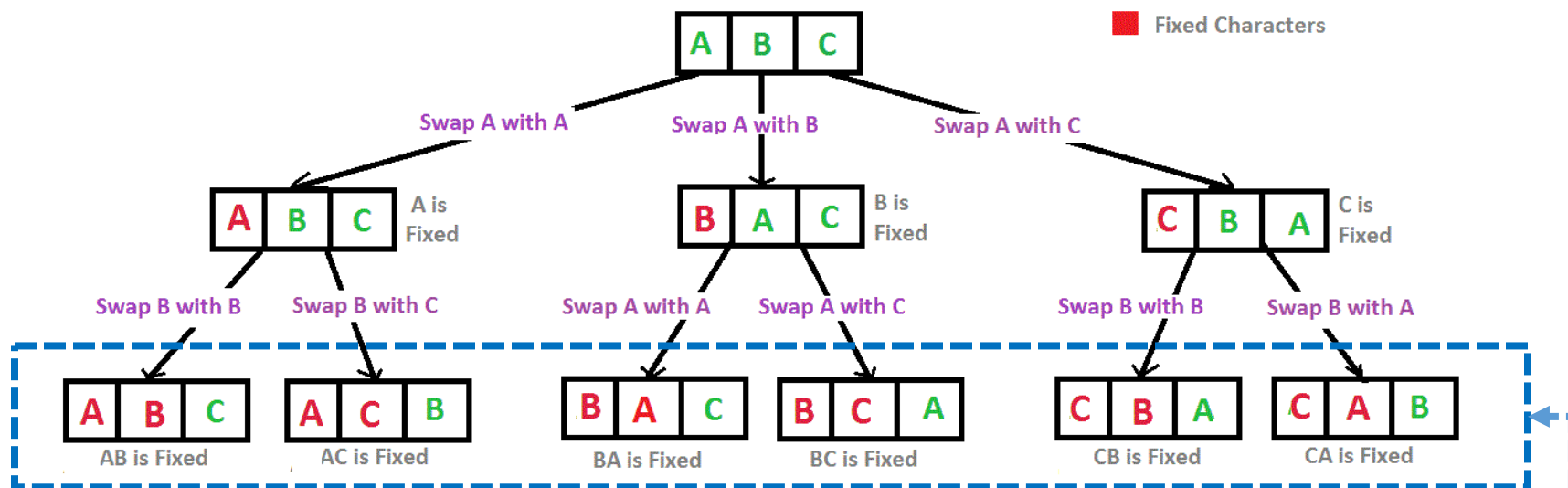
Slow Anagrams

```
template <typename T>
string Concat(vector<T>* set)
{
    string c{};
    for (const auto& item : *set)
        c += item;
    return c;
}

template <typename T>
void Swap(vector<T>* set, int a, int b)
{
    T tmp = set->at(a);
    set->at(a) = set->at(b);
    set->at(b) = tmp;
}

template <typename T>
void Permute(vector<T>* set, int level)
{
    // Heap's Algorithm
    if (level == 0) {
        // At this point, set contains a new permutation
        words.push_back(Concat(set));
    } else {
        for (int i{ 0 }; i < level; ++i) {
            Permute(set, level - 1);
            Swap(set, level % 2 == 1 ? 0 : i, level - 1);
        }
    }
}
```

Permutations



Recursion Tree for Permutations of String "ABC"

$${}^n P_k = \frac{n!}{(n-k)!}$$

Permutation
 Size of Set
 # Selected
 Permutation of Full Set
 Permutation of Left Behind Set

$${}_3 P_3 = \frac{3!}{(3-3)!} = \frac{3!}{0!} = \frac{3 \times 2 \times 1}{1} = 6$$

Permutations

Permutations by interchanges

By B. R. Heap

Methods for obtaining all possible permutations of a number of objects, in which each permutation differs from its predecessor only by the interchange of two of the objects, are discussed. Details of two programs which produce these permutations are given, one allowing a specified position to be filled by each of the objects in a predetermined order, the other needing the minimum of storage space in a computer.

In programs of a combinatorial nature, it is often required to produce all possible permutations of N objects. Many methods can be used for this purpose and a general review of them has been given by D. H. Lehmer in *Proceedings of Symposia in Applied Mathematics* (American Mathematical Society), Vol. 10, p. 179. In this note we shall describe methods for obtaining the permutations in which each permutation is obtained from its predecessor by means of the interchange of two of the objects. Thus $(N - 2)$ of the N objects are undisturbed in going from one permutation to the next.

We shall consider values of N up to $N = 12$, since the

of the first $(n - 1)$ objects and again permute the first $(n - 1)$ objects. Again interchange the n th object with one of the first $(n - 1)$ objects, making sure that this object has not previously occupied the n th cell. Now repeat the process until each of the objects has filled the n th position while the other $(n - 1)$ have been permuted, and clearly all $n!$ permutations have been found. Finally, it is clear that two objects can be permuted by a simple interchange, and so N objects can be so permuted. To achieve this one only needs to specify a total of

$$1 + 2 + 3 + \dots + (N - 1) = \frac{1}{2} N(N - 1)$$

1963

Run Lab 5 – Slow Anagrams

- Stop
- Least
- Traces
- Players
- Restrain

```
anagrams-slow
File Edit View Terminal Tabs Help
carest
carets
cartes
caster
caters
crates
reacts
recast
traces

parleys
parsley
players
pyrales
replays
sparely
splayer

restrain
retrains
strainer
terrains
trainers
transire

Total search time = 0.150 s

Process returned 0 (0x0)    execution time : 0.687 s
Press ENTER to continue.
```


Anagrams

- stop = post, pots, spot, tops
 - least = slate, stale, steal, tales
 - traces = carets, caster, caters, crates, reacts, recast
 - players = parsley, parleys, replays, sparely
 - restrain = retrain, strainer, terrains, trainers
 - mastering = ???
 - supersonic = ???
- 10 letters → 10! = 3,628,800 permutations!**

Novel Approach - Think in reverse! First, **SORT** each word in the given dictionary by **letter order**, then **SEARCH** for all words having matching letter orders – those words must be valid anagrams!

A way of finding anagrams that much faster than by trying every permutation!

Word in Letter Order

POSSUM	M	O	P	S	S	U		MOPSSU
POST	O	P	S	T				OPST
POSTAGE	A	E	G	O	P	S	T	AEGOPST
POTION	I	N	O	O	P	T		INOOPT
POTS	O	P	S	T				OPST
POUCH	C	H	O	P	U			CHOPU
SPOT	O	P	S	T				OPST
SPOUSE	E	O	P	S	S	U		EOPSSU
STOP	O	P	S	T				OPST
TOPICS	C	I	O	P	S	T		CIOPT
TOPS	O	P	S	T				OPST
TORCH	C	H	O	R	T			CHORT

Letter Order	Actual Word
MOPSSU	POSSUM
OPST	POST
AEGOPST	POSTAGE
INOOPT	POTION
OPST	POTS
CHOPU	POUCH
OPST	SPOT
EOPSSU	SPOUSE
OPST	STOP
CIOPST	TOPICS
OPST	TOPS
CHORT	TORCH

Sort list by
the **first**
column



AEGOPST	POSTAGE
CHOPU	POUCH
CHORT	TORCH
CIOPST	TOPICS
EOPSSU	SPOUSE
INOOPT	POTION
MOPSSU	POSSUM
OPST	POST
OPST	POTS
OPST	SPOT
OPST	STOP
OPST	TOPS

Matching word
orders are all
anagrams

Open Lab 6 – Fast Anagrams

```
main.cpp [x]
1  #include "stdafx.h"
2
3  using namespace std;
4
5  class Anagram
6  {
7  public:
8      Anagram(string word);
9      string word;
10     string letters;
11 };
12
13 Anagram::Anagram(string word)
14 {
15     boost::to_lower(word);
16     this->word = word;
17     sort(word.begin(), word.end());
18     this->letters = word;
19 }
20
21 auto compare_lambda = []
22                      (const Anagram& a, const Anagram& b) -> bool
23                      {
24                          return a.letters < b.letters;
25                      };
26
27 vector<string> phrases
```

A **lambda**
expression is
a *closure*

View Lab 6

Fast Anagrams

```
int main()
{
    // Load dictionary into the anagrams vector
    string line;
    ifstream inputFile("english_dictionary.txt");
    while (getline(inputFile, line))
    {
        boost::trim(line);
        if (line.length() > 0 )
            anagrams.push_back(Anagram(line));
    }

    // Sort the anagrams by their sorted letters
    sort(anagrams.begin(), anagrams.end(), compare_lambda)

    // Start a timer
    boost::timer timer;

    for (const auto& phrase : phrases)
    {
        Anagram input{ phrase };

        // Find *first* word in dictionary that has sorted letters
        // matching the current phrase also sorted by letters
        auto lower = lower_bound(anagrams.begin(),
                                anagrams.end(), input, compare_lambda);

        // Find *last* word in dictionary that has sorted letters
        // matching the current phrase also sorted by letters
        auto upper = upper_bound(lower, anagrams.end(),
                                input, compare_lambda);

        // Display all dictionary words matching the phrase's anagram
        for(auto& a{lower}; a < upper; ++a)
            cout << a->word << endl;

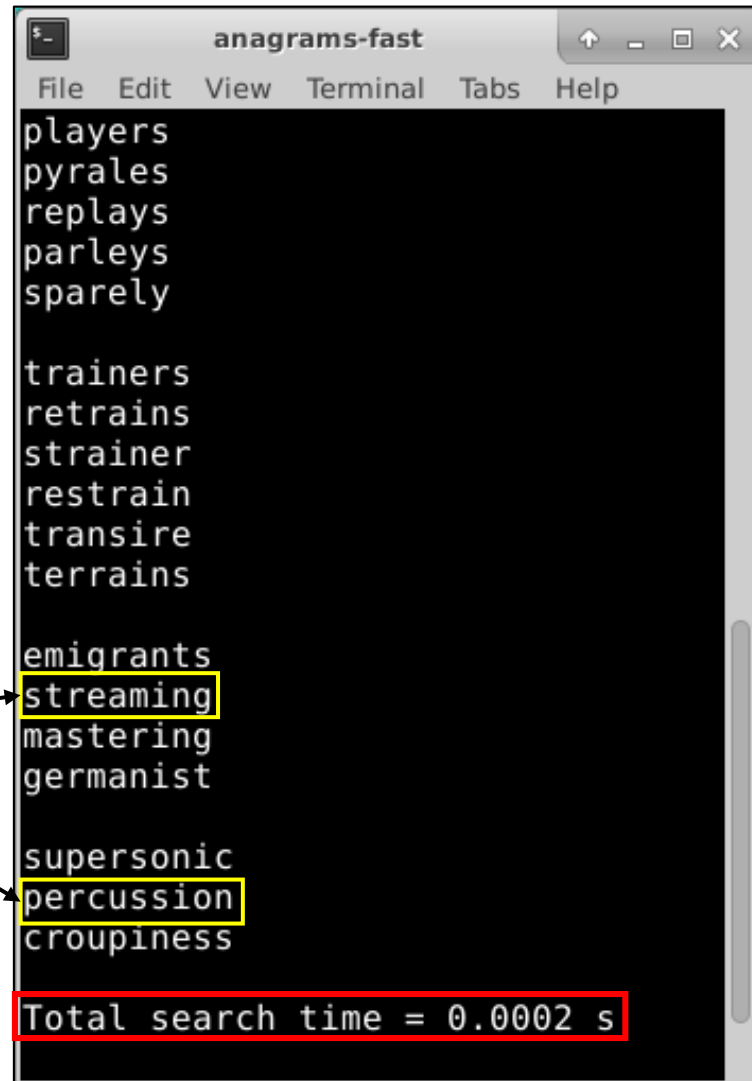
        cout << endl;
    }

    cout << "Total search time = "
        << fixed << setprecision(4)
        << timer.elapsed() << " s" << endl;

    return 0;
}
```

Run Lab 6 – Fast Anagrams

- Stop
- Least
- Traces
- Players
- Restrain
- Mastering
- Supersonic



```
anagrams-fast
File Edit View Terminal Tabs Help

players
pyrales
replays
parleys
sparely

trainers
retrains
strainer
restrain
transire
terrains

emigrants
streaming
mastering
germanist

supersonic
percussion
croupiness

Total search time = 0.0002 s
```

Slow vs. Fast Anagrams

- The Slow Anagram algorithm took **150** ms while the Fast Anagram algorithm took **0.2ms** – that is a **750X speedup** despite including **3 additional long words** in the search!
 - Even **binary searching** the dictionary cannot overcome the penalty of enumerating permutations which **could never be valid English words**
 - Wasted effort is the inherent problem with the Slow Anagram
- Don't expand the search space by testing **unconstrained permutations** – that leads to **combinational explosion**
 - The *dictionary* naturally constrains the **search space**
 - Mine the "answer key" for *any* information it possibly contains
 - That which constrains you often has hidden value

Open Lab 7

Compound Anagrams

What **two** smaller words can be made out of the letters of just **one** word?

```
class Anagram2
{
public:
    Anagram2(string word);
    Anagram2(string word1, string word2);
    string word1;
    string word2;
    string letters;
};

Anagram2::Anagram2(string word)
{
    sort(word.begin(), word.end());
    this->letters = word;
}

Anagram2::Anagram2(string word1, string word2)
{
    this->word1 = word1;
    this->word2 = word2;
    string word = word1 + word2;
    sort(word.begin(), word.end());
    this->letters = word;
}

auto compare_lambda = [](const Anagram2& a, const Anagram2& b) ->bool {
    return a.letters < b.letters; };

bool contained(string a, string b)
{
    // Is a fully & uniquely contined in b?
    if (a.length() > b.length())
        return false;

    for (size_t i{}; i < a.length(); i++)
    {
        auto pos = b.find(a[i], i);
        if (pos == string::npos)
            return false;
        b[pos] = ' ';
    }

    return true;
}
```

moor (room) \approx **dimoor**rtty (dormitory)

View Lab 7

Compound Anagrams

```
// Read in the dictionary file
ifstream inputFile("english_dictionary.txt");
string line;
while (getline(inputFile, line)) {
    boost::trim(line);
    if (line.length() > 0) {
        Anagram2 word(line);
        // Only add words from dictionary that could
        // possibly be in the anagram of the given phrase
        if (contained(word.letters, input.letters))
            dictionary.push_back(line);
    }
}

// Create compound anagram from every
// successive two words in the dictionary
for (size_t i{}; i < dictionary.size() - 1; i++)
    for (size_t j{ i + 1 }; j < dictionary.size(); j++)
        anagrams.push_back(
            Anagram2(dictionary.at(i),
                dictionary.at(j)));

// Sort the anagrams by their sorted letters
sort(anagrams.begin(), anagrams.end(), compare_lambda);

// Find *first* word in dictionary that has sorted letters
// matching the current phrase also sorted by letters
auto lower = lower_bound(anagrams.begin(), anagrams.end(),
    input, compare_lambda);

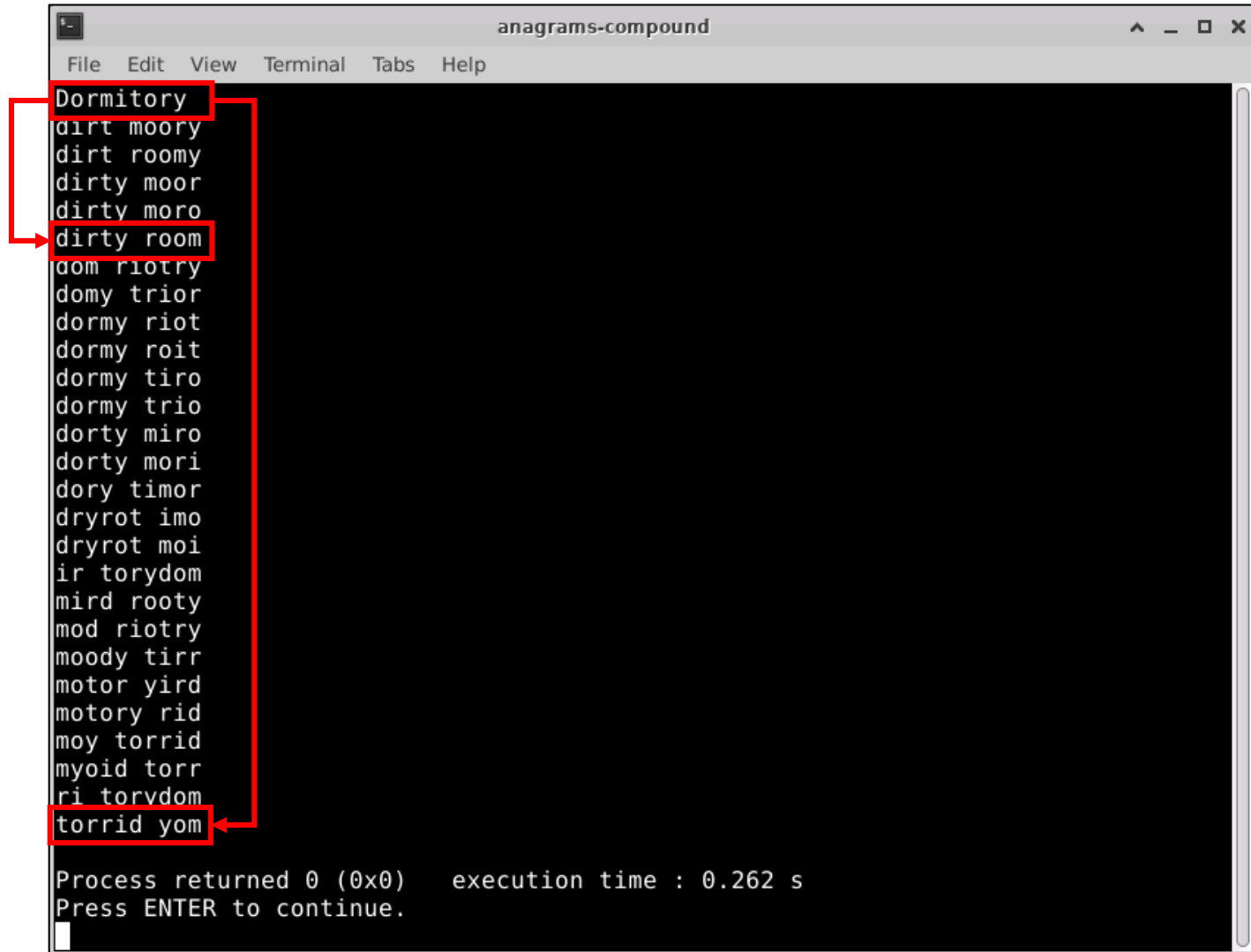
// Find *last* word in dictionary that has sorted letters
// matching the current phrase also sorted by letters
auto upper = upper_bound(lower, anagrams.end(),
    input, compare_lambda);

// Create a vector concatenating both words of each anagram
vector<string> phrases;
for (auto& a = lower; a < upper; a++)
    phrases.push_back(a->word1 + " " + a->word2);

// Sort & display the vector of the compound anagrams
sort(phrases.begin(), phrases.end());
for (auto& s : phrases)
    cout << s << endl;
```

**dimoorrtty (dormitory) =
dimoorrtty (dirty room)**

Run Lab 7 - Compound Anagrams



```
anagrams-compound
File Edit View Terminal Tabs Help
Dormitory
dirt moory
dirt roomy
dirty moor
dirty moro
dirty room
dom riotry
domy trior
dormy riot
dormy roit
dormy tiro
dormy trio
dorty miro
dorty mori
dory timor
dryrot imo
dryrot moi
ir torydom
mird rooty
mod riotry
moody tirr
motor yird
motory rid
moy torrid
myoid torr
ri torvdom
torrid yom

Process returned 0 (0x0)   execution time : 0.262 s
Press ENTER to continue.
```

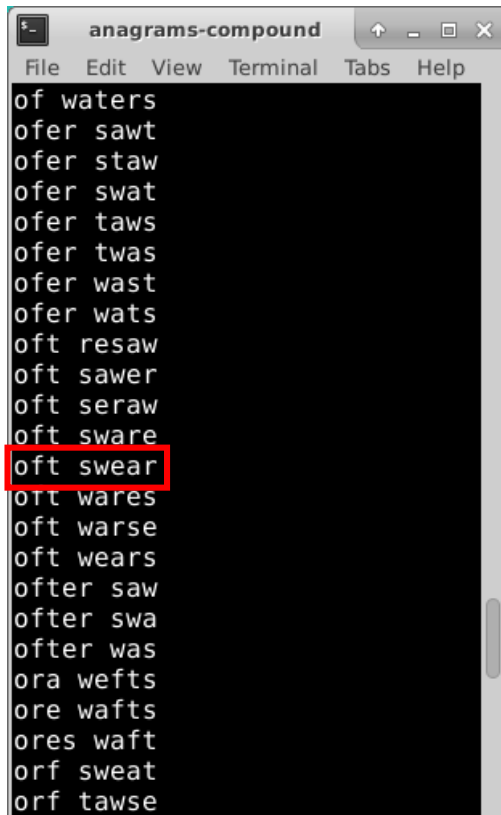
Edit Lab 7 - Compound Anagrams

Uncomment the other lines to reveal
lurking compound anagrams.. 😊

```
52  vector<string> dictionary;  
53  vector<Anagram2> anagrams;  
54  
55  int main()  
56  {  
57      string phrase{ "Dormitory" };  
58      //string phrase{ "Software" };  
59      //string phrase{ "Mother-In-Law" };  
60  }
```

Run Lab 7 - Compound Anagrams

```
int main()
{
    //string phrase{ "Dormitory" };
    string phrase{ "Software" };
    //string phrase{ "Mother-In-Law" };
```

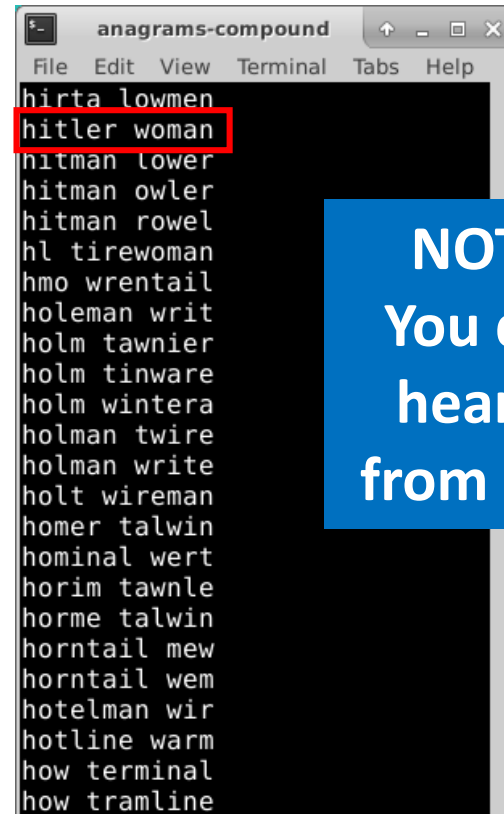


anagrams-compound

File Edit View Terminal Tabs Help

of waters
ofer sawt
ofer staw
ofer swat
ofer taws
ofer twas
ofer wast
ofer wats
oft resaw
oft sawer
oft seraw
oft sware
oft swear
oft wares
oft warse
oft wears
ofter saw
ofter swa
ofter was
ora wefts
ore wafts
ores waft
orf sweat
orf tawse

```
int main()
{
    //string phrase{ "Dormitory" };
    //string phrase{ "Software" };
    string phrase{ "Mother-In-Law" };
```



anagrams-compound

File Edit View Terminal Tabs Help

hirta lowmen
hitler woman
hitman lower
hitman owler
hitman rowel
hl tirewoman
hmo wrentail
holeman writ
holm tawnier
holm tinware
holm wintera
holman twire
holman write
holt wireman
homer talwin
hominal wert
horim tawnle
horme talwin
horntail mew
horntail wem
hotelman wir
hotline warm
how terminal
how tramline

NOTICE:
You didn't
hear that
from me! 😊

Who knew?

Listen = Silent

The Morse Code = Here come dots

The meaning of life = The fine game of nil

Statue of Liberty = Built to stay free

Now you know...

- C++ **strings** are essentially a vector of type **char**
 - **ASCII** is an international standard for encoding most Western language characters into a single byte
- Character histograms enable **frequency analysis**
 - Caesar-Shift ciphers are not very secure
 - All **monoalphabetic substitution ciphers** can be broken with **bigram analysis**
 - Using “brute force” to crack a cipher is often **intractable** – get statistics on your side to help you out!
- **Heap’s Algorithm** will generate all **permutations** of a set
- Consider problems *backwards*: don’t expand search spaces