More File IO

CIS 15 : Spring 2007

Functionalia

Office Hours Today 2 to 3pm - 0317 N (Bridges Room)

HW 2 due on Sunday March 11, 11:59pm

Note: Midterm is on MONDAY, March 12th

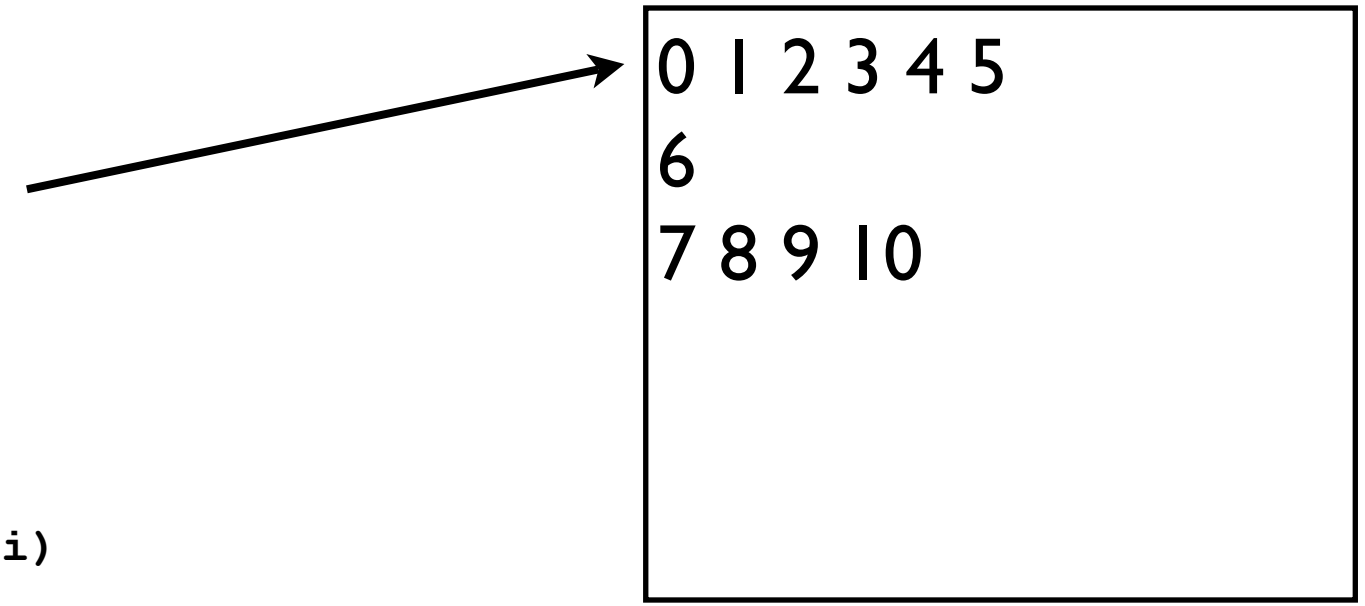
Review: Thursday

Today:

- Survey #1
- File IO

Single Integers or a Line at a Time?

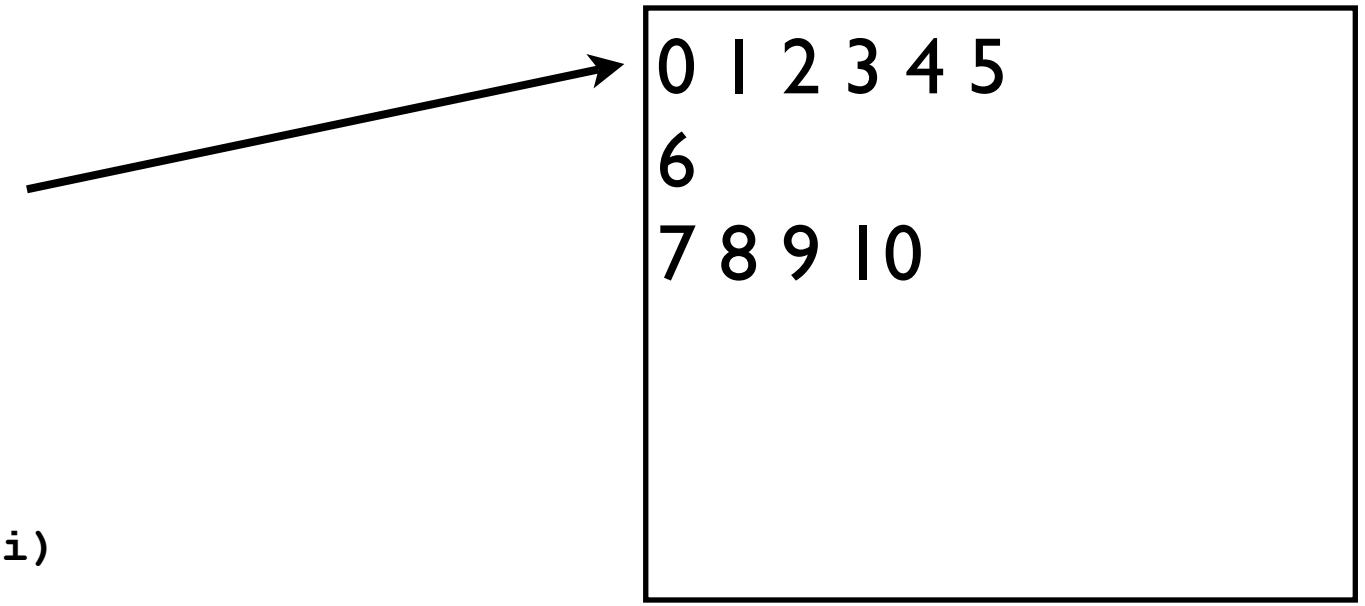
```
ifstream dataFile;

int i;


...

while(dataFile >> i)

   cout << i;
```



Output:

Single Integers or a Line at a Time?

```
ifstream dataFile;

int i;


...

while(dataFile >> i)

   cout << i;
```



Output:

```
012345678910
```

What is this behavior?

```
ifstream dataFile;

int i;

...

while(dataFile >> i)

   cout << i;
```

What is this behavior?

```
ifstream dataFile;

int i;

...

while(dataFile >> i)

   cout << i;
```

Integers are extracted (i.e. read in) from the dataFile file stream and placed into the variable i.

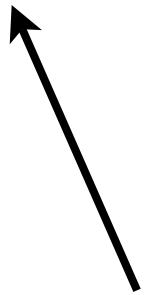
The expression `dataFile >> i` returns a non-negative number if integers are read in.

It will return a 0 when there was no more integers to extract.

A 0 value will break the `while(...)` loop.

Flexible Read/Write to Files

```
#include <fstream>

...

fstream myFile;

myFile.open("journal.txt", ios::out);
```

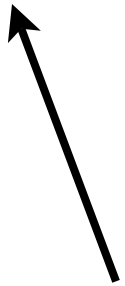


File Access Flags

```
myFile << "Writing out!";
```

Flexible Read/Write to Files

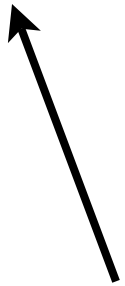
```
#include <fstream>

...

fstream myFile;

myFile.open("journal.txt", ios::in);


int code;

myFile >> code;
```



File Access Flag

Flexible Read/Write to Files

```
#include <fstream>

...

fstream myFile;

myFile.open("journal.txt", ios::in | ios::out);


int code;

myFile >> code;

code++;

myFile << code;
```



File Access Flags

File Access Flag(s)	Behavior
`ios::out`	Data written out. Existing file is deleted (by default). No file results in creation.
`ios::in`	Data read. No file results in error.
`ios::in | ios::out`	Read/Write. File preserved. No file results in creation.
`ios::out | ios::app`	Append. Go to the end of file, start writing. No file results in creation.

Excerpt from Table 12-2

<EOF>

```
fstream dataFile;

dataFile.open("demo.txt", ios::out);

dataFile.close();
```

Once `close()` is called, an end-of-file character is written at the end of the file.

It allows a program to know when the end of file is reached, or where to start *appending* characters to.

In C++, usually predefined as a negative number (non-printable number) and is represented by control-Z.

Opening Files with Constructor

```
fstream dataFile;
dataFile.open("demo.txt", ios::in | ios::out);
```

Using the `fstream` constructor:

```
fstream dataFile("demo.txt", ios::in | ios::out);
```

Checking if the File Exists

```
fstream dataFile("values.txt", ios::in);
if(dataFile.fail())
{
   // File does not exists, create it now...
   dataFile.open("values.txt, ios::out);

}
else
{
   // File already exists.
   dataFile.close();
}
```

Legal File Names

Depending on the OS, file names have certain rules associated to them.

OS	Character	Reserved	Max Length
MS-DOS	A-Z,0-9,-,_	Everything else	8 + 3
Win XP	any	|\?*<":>/	254
Mac X	any	: /	255
UNIX	any	/	255

Output Formating : Precision and Format

```
fstream demo;
double num = 3.1415929;

demo.open("wholeLottaPi.txt", ios::out);

demo << fixed;
demo << num << endl;
demo << setprecision(4) << num << endl;
demo << setprecision(2) << num << endl;
demo << setprecision(0) << num << endl;
demo << scientific;
demo << num;
```

What is the output?

Output Formating : Precision and Format

```
fstream demo;
double num = 3.1415929;


demo.open("wholeLottaPi.txt", ios::out);


demo << fixed;
demo << num << endl;
demo << setprecision(4) << num << endl;
demo << setprecision(2) << num << endl;
demo << setprecision(0) << num << endl;
demo << scientific;
demo << num;
```

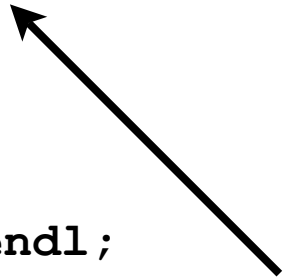
```
3.1415929
3.1416
3.14
3
3e+00
```

Using File Streams with Functions

```
void showContents(fstream &file)
{
   char line[MAX_LINE_SIZE];
   while(file >> line)
   {
      cout << line << endl;
   }
}


...


fstream dataFile;
...
showContents(dataFile);
```

Using File Streams with Functions

```
void showContents(fstream &file)
{
   char line[MAX_LINE_SIZE];
   while(file >> line)
   {
      cout << line << endl;
   }
}


...


fstream dataFile;
...
showContents(dataFile);
```



Where does this come from?

Refined Error Testing

Bit	Function	Behavior
`ios::eofbit`	`eof()`	end of input stream
`ios::failbit`	`fail()`	attempted operation failed
`ios::hardfail`	`fail()`	unrecoverable error occurred
`ios:badbit`	`bad()`	invalid operation attempted
`ios::goodbit`	`good()`	stream in good condition
	`clear()`	clears all above error flags

Show the File State

```
void showState(fstream &file)
{
   cout << "File Status:" << endl;
   cout << "   eof bit: " << file.eof() << endl;
   cout << "   fail bit: " << file.fail() << endl;
   cout << "   bad bit: " << file.bad() << endl;
   cout << "   good bit: " << file.good() << endl;
   file.clear(); // clears any error bits
}
...
fstream demo.open("numbers.txt" ios::in);
int i;
while(demo >> i)
  cout << i;
showState(demo);
showState(demo);
```

Show the File State

```
void showState(fstream &file)
{
   cout << "File Status:" << e
   cout << "   eof bit: " << f
   cout << "   fail bit: " <<
   cout << "   bad bit: " << f
   cout << "   good bit: " <<
   file.clear(); // clears any
}
...
fstream demo.open("numbers.txt
int i;
while(demo >> i)
  cout << i;
showState(demo);
showState(demo);
```

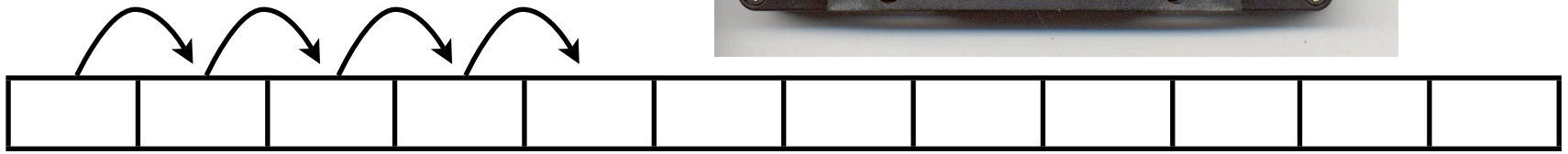
```
5
4
3
2
File Status:
   eof bit: 1
   fail bit: 1
   bad bit: 0
   good bit: 0
File Status:
   eof bit: 0
   fail bit: 0
   bad bit: 0
   good bit: 1
```

Reading and Writing Character by Character

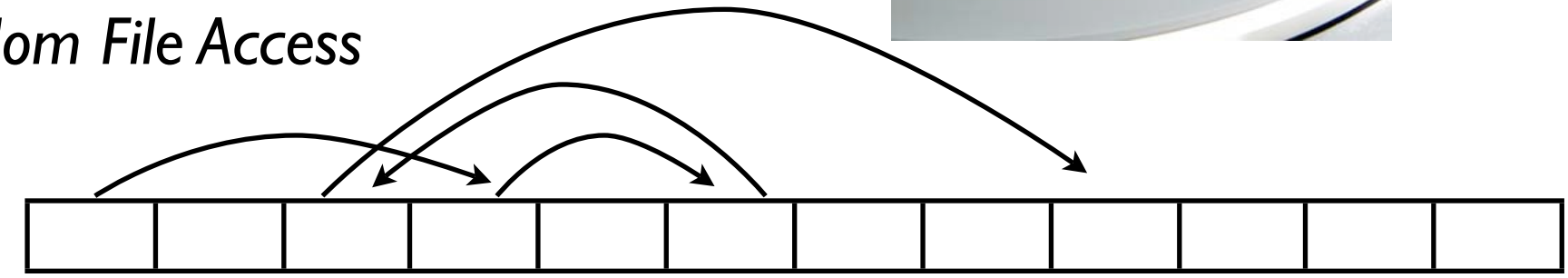
```
fstream file("demo.txt", ios::in);
fstream output("output.txt", ios::out);
char ch;

file.get(ch);
while(!file.eof())
{
   cout << ch;
   file.get(ch);
   output.put(toupper(ch));
}

file.close();
output.close();
```

Random Access Files

Sequential File Access



Random File Access



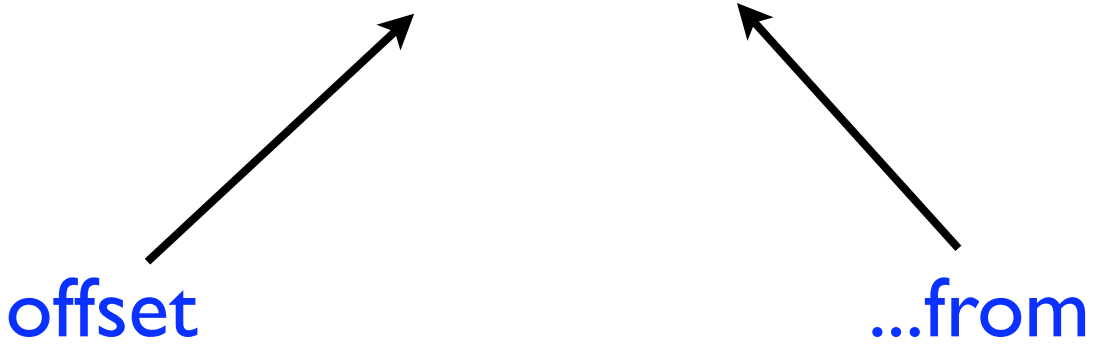
`seekp` and `seekg`

Use **`fstream`** member functions **`seekp`** and **`seekg`**.

`seekp` works with files opened for output (think **p** for **p**ut)

`seekg` works with files opened for input (think **g** for **g**et)

```
file.seekp(20L, ios::beg);
```



`seekp` and `seekg`

Offset is in bytes. (L stands for *long integer*)

Offset can be negative (i.e. move backwards.

Flag	**Offset from...**
ios::beg	beginning of file
ios::end	end of file
ios::cur	from current position

Example

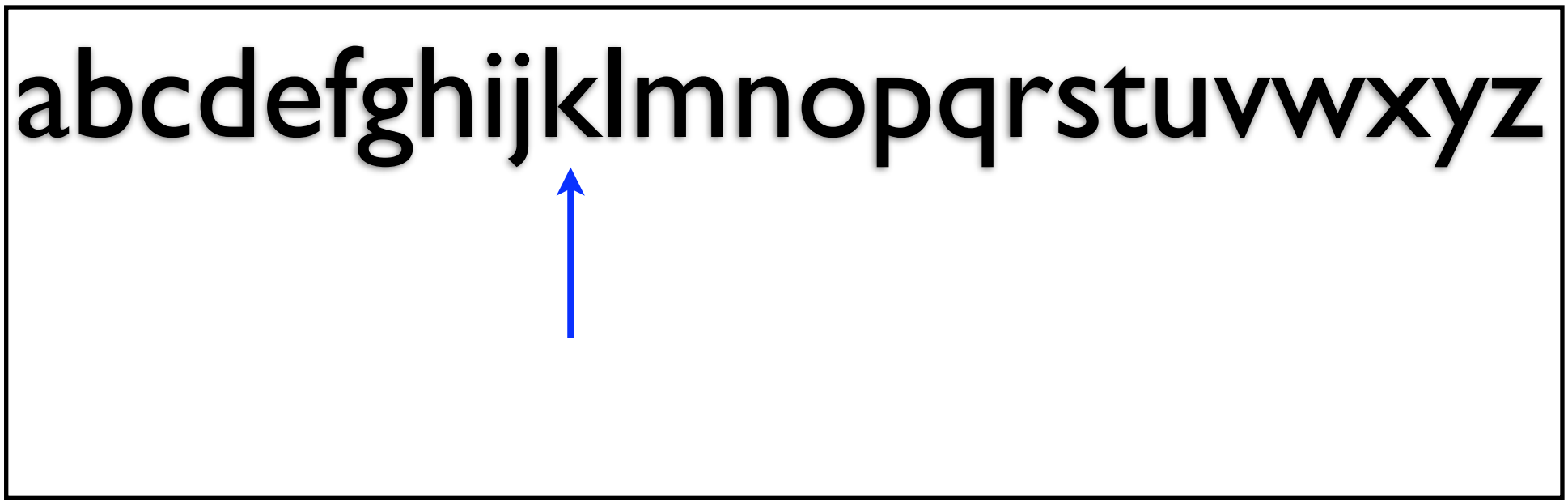
```
char ch;
file.seekg(10L, ios::beg);
file.get(ch);
cout << ch;
```

Where does this go?

abcdefghijklmnopqrstuvwxyz

Example

```
char ch;
file.seekg(10L, ios::beg);
file.get(ch);
cout << ch;
```



Example

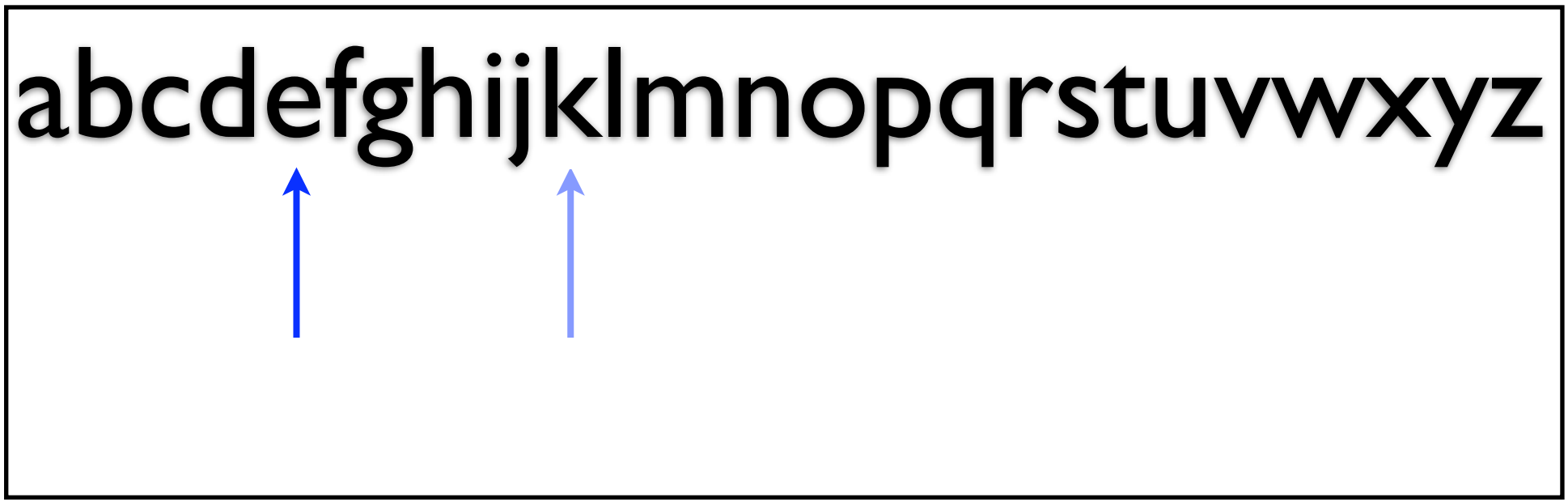
```
char ch;
file.seekg(10L, ios::beg);
file.seekg(-6L, ios::cur);
file.get(ch);
cout << ch;
```

Where does this go?

abcdefghijklmnopqrstuvwxyz

Example

```
char ch;

file.seekg(10L, ios::beg);

file.seekg(-6L, ios::cur);

file.get(ch);

cout << ch;
```



Example

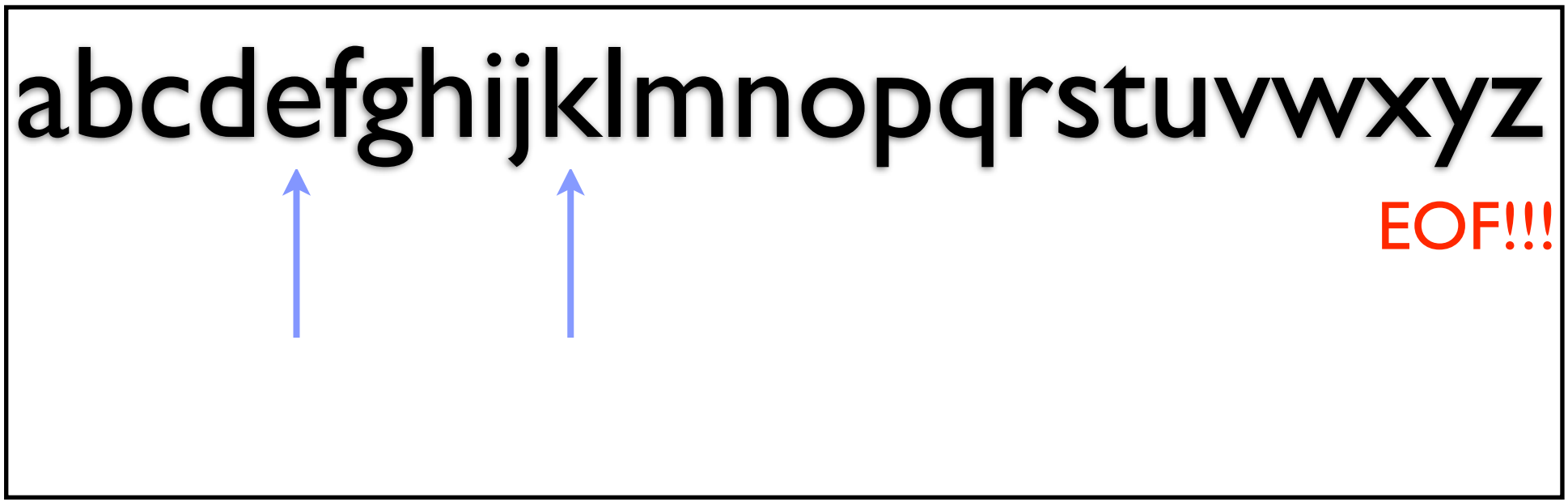
```
char ch;

file.seekg(10L, ios::beg);

file.seekg(-6L, ios::cur);

file.seekg(30L, ios::beg);

file.get(ch);

cout << ch;
```

Where does this go?

abcdefghijklmnopqrstuvwxyz

Example

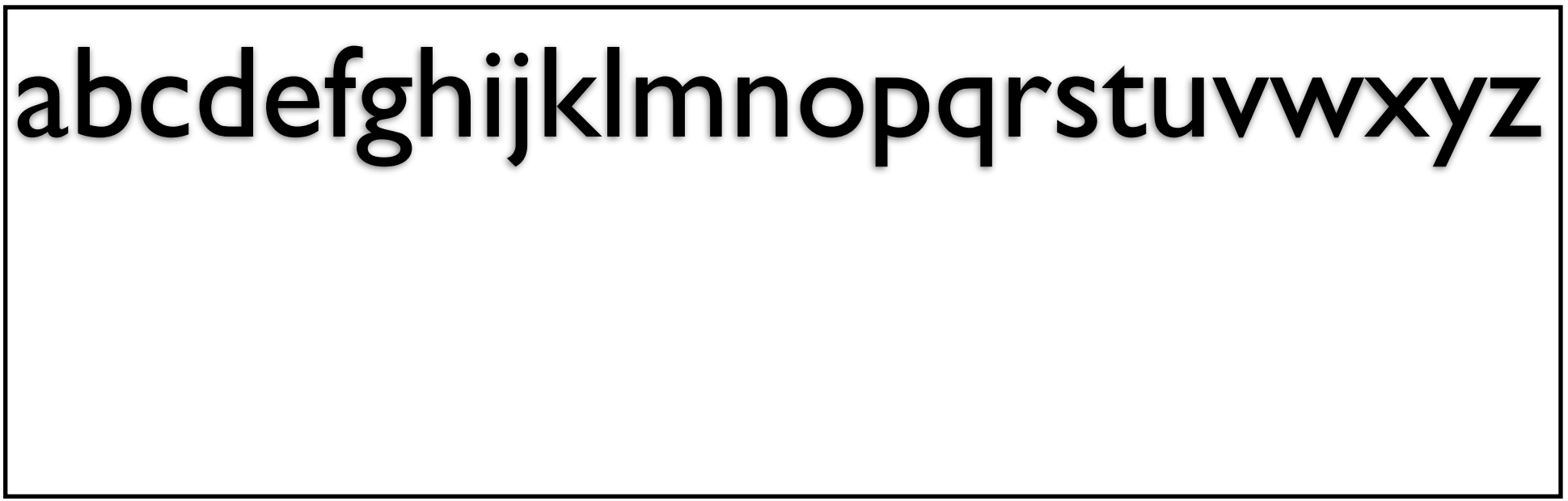
```
char ch;

file.seekg(10L, ios::beg);

file.seekg(-6L, ios::cur);

file.seekg(30L, ios::beg);

file.get(ch);

cout << ch;
```



Example

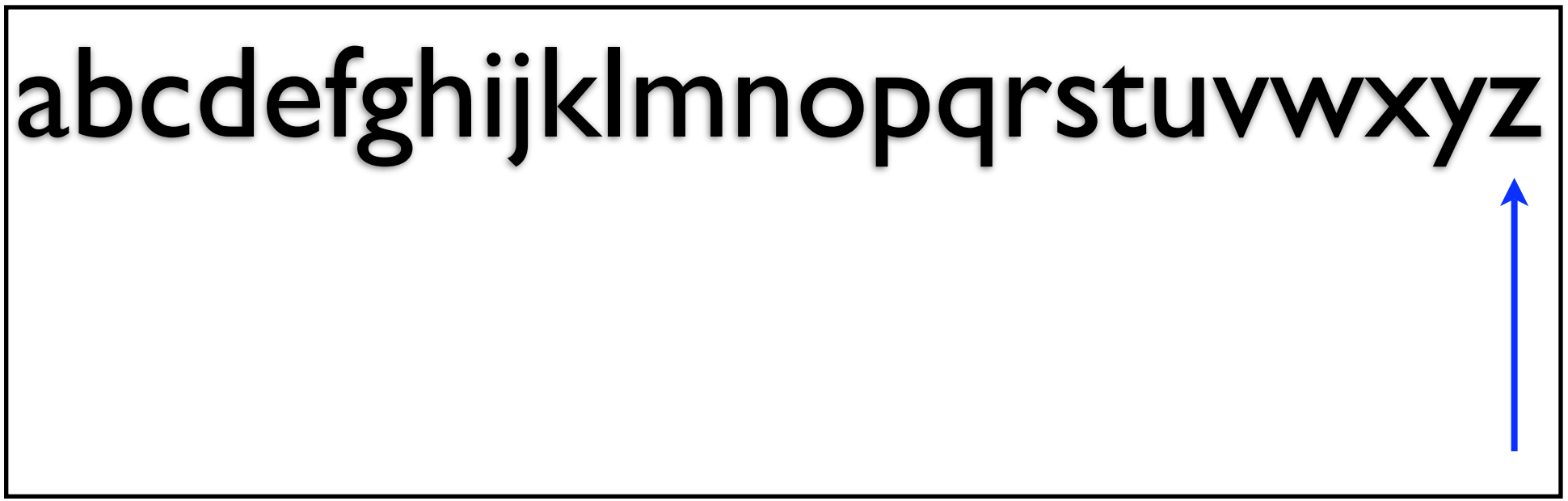
```
char ch;
file.seekg(10L, ios::beg);
file.seekg(-6L, ios::cur);
file.seekg(30L, ios::beg);
file.clear();
file.seekg(-1L, ios::end);
file.get(ch);
cout << ch;
```

Where does this go?



Example

```
char ch;
file.seekg(10L, ios::beg);
file.seekg(-6L, ios::cur);
file.seekg(30L, ios::beg);
file.clear();
file.seekg(-1L, ios::end);
file.get(ch);
cout << ch;
```



Example

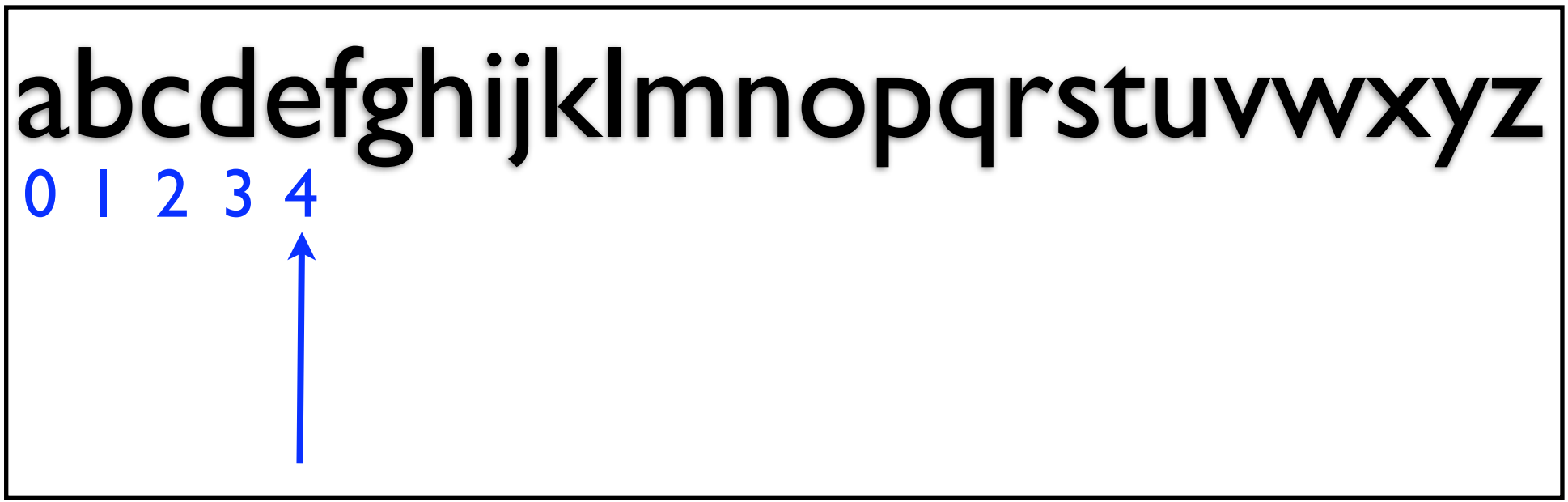
```
char ch;
file.seekg(10L, ios::beg);
file.seekg(-6L, ios::cur);
cout << file.tellg();

file.get(ch);
cout << ch;
```

Where does this go?

abcdefghijklmnopqrstuvwxyz

Example

```
char ch;
file.seekg(10L, ios::beg);
file.seekg(-6L, ios::cur);
cout << file.tellg();

file.get(ch);
cout << ch;
```



What does n represent?

```
int n;

file.seekg(0L, ios::end);
n = file.tellg();
```

What does n represent?

```
int n;

file.seekg(0L, ios::end);
n = file.tellg();
```

n contains the file size in bytes.

Review on Thursday

DONE.