Streams and Pipes

Along the Stream by Sharon France

Streams

As a leaf is carried by a stream, whether the stream ends in a lake or in the sea, so too is the output of your program carried by a stream not knowing if the stream goes to the screen or to a file.

Washroom Wall (1995)
Quoted by Savitch

Input/Output

People communicate interactively and instinctively.

Hello.

How are you?

Not bad . . .

Gota go!

When people communicate with computers, it is is natural and valuable to mimic this. However, it is more complicated than it seems to program this interaction.

Complicated

1. Synchronizing the actors complicates the arrow of time.

2. The multiple systems involved are difficult to control and anticipate.
3. Specifying, or communicating precisely, the interactivity is difficult.

Tt is useful to break I/O into simplier pieces and learn simplier patterns of communication.

Streams

Programming I/O is kind of tedious and we might be tempted to give up and move on to more “pure” computational challenges. But programs need input and output.

- A program without input would always give the same result.
- A program without output would not have worth running as the user would get no answer.

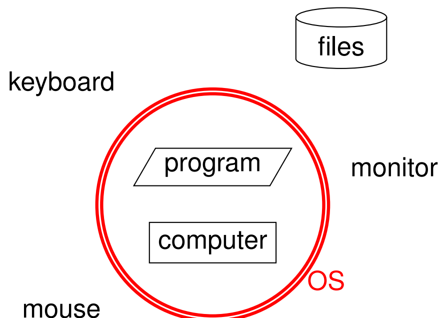
Streams invented with UNIX operating system in the 1960s are universally used today as an organizing I/O principle.
Perhaps because people don’t think much about I/O and are accustomed to complex GUIs some minor points (like end-of-file) cause more trouble than they should. So we are careful here to explain the whole concept and hope to demonstrate why something so simple is actually quite profound.

Streams

A stream is a convenient abstraction provided to a program by the operating system to make I/O easier and more uniform.

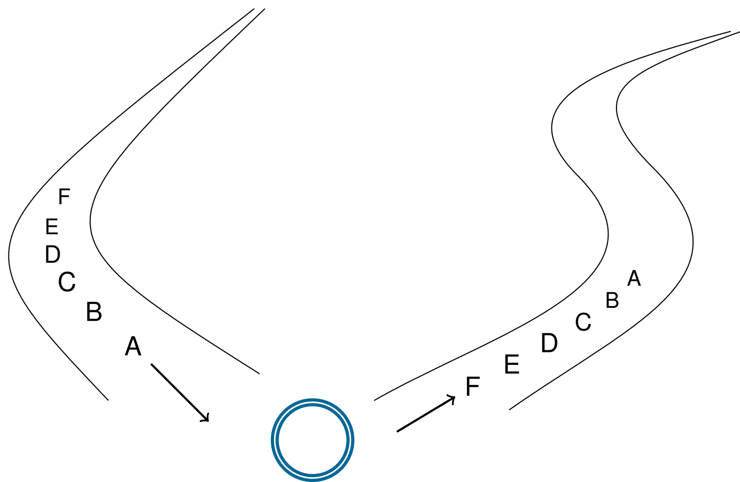
A *stream* is a conduit of data to or from a program. If the flow is into the program, the stream is called an *input stream*. If the flow is out of the program, the stream is called an *output stream*.

OS

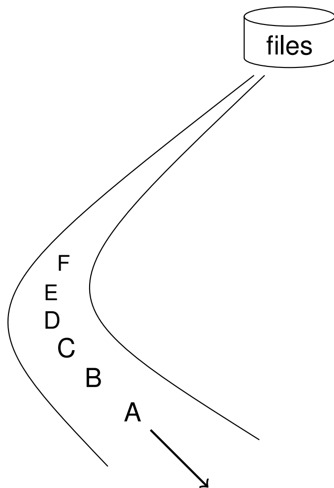


The program controls the computer, but needs the assistance of the operating system to communicate with the outside environment. A programming language is incapable of doing I/O except through the operating system.

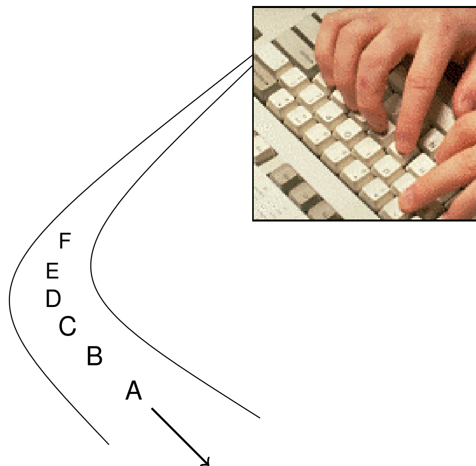
Input and Output Streams



Input Streams



Input Streams

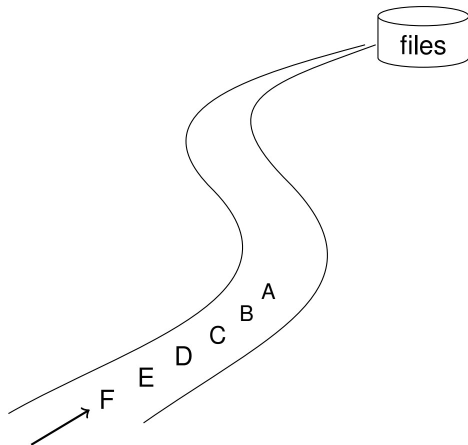


The data on an input stream can come from a person typing on the keyboard or from a file stored on the computer.

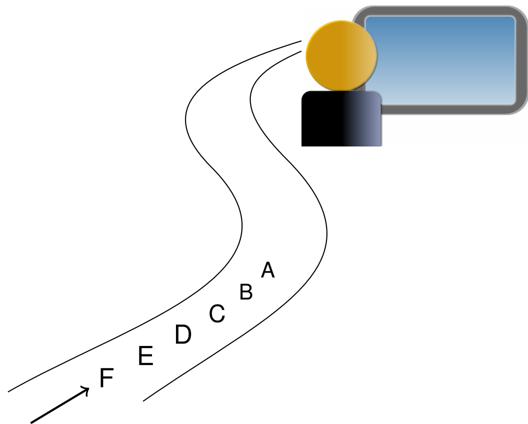
It is *more flexible* for the **user** to decide where the data comes from when the program is *executed*.

It is *less flexible* for the **programmer** to decide where the data comes from when the program is *written*.

Output Streams



Output Streams



The data on an output stream can go to a person at the monitor or to a file stored on the computer.

It is *more flexible* for the **user** to decide where the data goes to when the program is *executed*.

It is *less flexible* for the **programmer** to decide where the data goes to when the program is *written*.

Streams

Since the byte or octet (8-bits) is the smallest workable unit of (binary) data, the simplest view is that all data is a sequence of bytes. Files are a sequence of bytes; streams are a sequence of bytes.

These bytes make up the units of the common data types: characters, integers, etc.

```
1f 8b 08 40 d3 ad
3e f2 7d cd 55 27
65 87 43 c4
```

Medium/Ether

* k l m . ' + i j k = x y z , u v w) r s t (1 2 3 -> 4 5

a b c * a b c . , + i j k = x y z , u v w) r s t (1 2 3

a b c * a b c . , + i j k = x y z , u v w) r s t (1 2 3

Medium/Ether

A medium is an agency or means of doing something. Writing is a medium of communication.
Un medio es una agencia de hacer algo. La escritura es un medio de comunicacin.
Ein Medium ist eine Agentur oder ein Mittel, etwas zu tun. Schreiben ist ein Medium der Kommunikation.
Chinese
Arabic

In the process of communication, the medium is a channel or the means by which information (the message) is transmitted between a speaker or writer (the sender) and an audience (the receiver). In our case computation is expressed by the programmer to the computer by means of programming languages based on text.

Streams

A concrete stream as realized in a programming language can hide many of the complicated facets of data in addition to the mere hardware. I/O details like buffering, echoing, compressing, and encryption might easily be hidden from the programmer by a stream interface. The programmer in a language may have access to the details, but each language differs in how these facilities are exposed, if at all.

A stream can hide the details of data conversion (like byte-order) as well.
Since input and output is often written language, it is convenient to view a stream as text.
A *text stream* is a sequence of characters.

Streams

In fact, we often limit our attention to text streams as opposed to other kinds of binary I/O data.

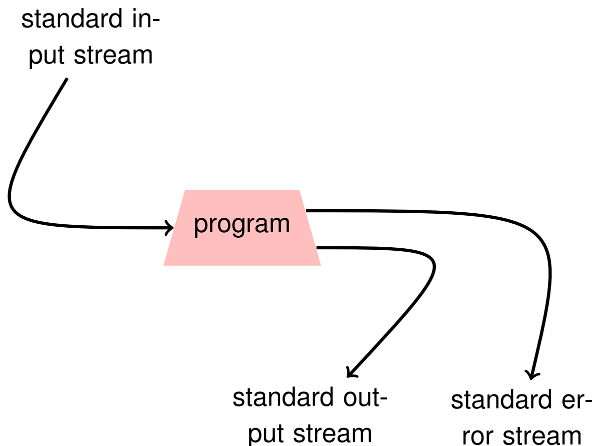
Do not assume that one byte encodes one character, although this is true for popular encodings like Latin-1, Mac-Roman, and others.

Standard I/O

A programming language these days normally assumes that the operating system provides three streams as a matter of course (and others streams upon request). This so-called *standard I/O* is divided into the *standard input stream* (associated with the keyboard by default), the *standard output stream* (associated with the display screen by default), and the *standard error stream* (also associated with the display screen by default).
One can completely ignore the existence of the standard error stream, if the program has no use for it.

Additional streams (both input and output) can be created by the programmer.

Standard I/O



Standard I/O in Languages

In any language there is a certain amount of mysterious boilerplate code required for taking advantage of the built-in standard I/O facilities. At first, one must find the right template and use it. All programs need I/O, even though the language mechanisms to enable the use of the I/O facilities are often less commonly used and may not be introduced to beginners.

The C Programming Language

```
#include <stdio.h>

int main (void) {
    int c;
    for (;;) {
        c=fgetc(stdin);
        if (c==EOF) break;
    }
    fputc('A', stdout);
    fputc('X', stderr);
}
```

The C++ Programming Language

```
#include <iostream>

using std::cout;
using std::cin;
using std::cerr;

int main (void) {
    char c;
    while (cin.eof()) {
        cin>>c;
    }
    cout << 'A';
    cerr << 'X';
}
```

Streams in Java

The concept of a stream is realized in Java with two classes

`InputStream OutputStream`

The standard I/O streams are defined in the Java class `System`, and they are:

`InputStream in; PrintStream out; PrintStream err;`

(A `PrintStream` is a special kind of `OutputStream`.)

Regrettably, Java does not treat the two kinds of streams equally.
The output streams are prepared in advance for simple text output.
The input stream is not prepared in advance for simple text input and so the user usually has some preparation to do in order to facilitate the use of the standard input stream.

Inside a Java program the end-of-file signal from the operating system can be detected like this.

```
// Create a reader for the standard input stream
final InputStreamReader reader =
   new InputStreamReader (System.in, "LATIN-1");

for (;;) {
   // read one character
   final int c = reader.read();
   if (c == -1) break;
}

System.out.print ('A');
System.err.print ('X');
```

Java uses a particular kind of input stream called “readers” especially for text.

Java uses Unicode internally for all characters and will translate from any character set, e.g, Latin-1, for you.

The Java method `read()` will return the integer -1 when the end of the input stream is reached.

For various reasons: convenience, efficiency, exception handling, etc; there is a better approach to text I/O that should used. It is illustrated in the next program.

```java
import java.util.Scanner;

public class CopyText {

    public static void main (String[] args) {

        // System.in  is a java.io.InputStream
        // System.out is a java.io.PrintStream

        final Scanner stdin =
             new Scanner(System.in, "US-ASCII");

        // Read standard input stream line by line
        while (stdin.hasNextLine()) {
            final String line = stdin.nextLine();
            System.out.println (line);
        }
    }
}
```

(This program does not buffer the standard input; for large input data, significant performance improvement can be obtained by buffering, i.e., using the Java class `BufferedInputStream`).
Avoid the method `String.split()` and the class `StringTokenizer`, when using the class `Scanner`, as `Scanner` does more and it is easier.

By default the scanner class breaks the input into token separated by white space. Occasionally, one needs something else like entries separated by commas (and new lines).

```
import java.util.Scanner;

public class CommaDelimited {

    // Reg ex:  comma, or Unix line or DOS line term
    private final static String DELIM = ",|\\n|\\n\\

    public static void main (String[] args) {

        final Scanner stdin =
           new Scanner(System.in, "US-ASCII").useDeli

        // Read comma separated tokens in standard in
        while (stdin.hasNext()) {
           final String entry = stdin.next();
           System.out.println (entry);
        }
    }
}
```

Never mislead the reader or yourself with your choice of names for variables. For example, `keyboard` is a particularly bad choice to name a scanner associated with the standard input stream. The program has no control on where the input on the standard input stream comes from. Just calling the scanner `keyboard` does make the input come the keyboad.

```
final Scanner stdin =
      new Scanner(System.in, "US-ASCII").useDelimit
```

NB. Chaining method in this way is called method cascading in object-oriented languages. The method `useDelimiter()` returns `this` rather than be a less flexible `void` method.

File Redirection

You can control where the operating system gets the characters it puts in the input stream and where it puts the characters from the output stream. This is called *file redirection*. For example, on the Unix and Windows command line you can request that the standard input come from a file.

```
java Program < data
```

Now the characters found in the standard input come from the file named data. (The first version of Unix in 1970 already had file redirection for two standard I/O streams.)

When the standard input comes from the keyboard a signal of some kind is needed to indicate from the interactive user when the end-of-file is reached because the length of the stream cannot be known in advance. This signal is understood by the operating system which then passes it to the program. Different operating systems provide different mechanisms for doing this. In Unix the usual signal is typing control-D (in Windows, control-Z) on a line by itself.

Interprocess communication in Unix is done through signals.

```
broadside> stty -a
speed 38400 baud; rows 38; columns 80;
intr = ^C; quit = ^\; erase = ^H; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; dsusp = ^Y; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; status = ^T; min = 1; time = 0;
parenb -parodd cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
-ixany -imaxbel
opost -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -tostop -echoprt echoctl
echoke
```

In operating systems class one learns more about signals.

The standard output is the stream of characters that the Java program writes to the `PrintStream` called `System.out`. The standard output is associated with the computer display by default, but can be associated with the file as in

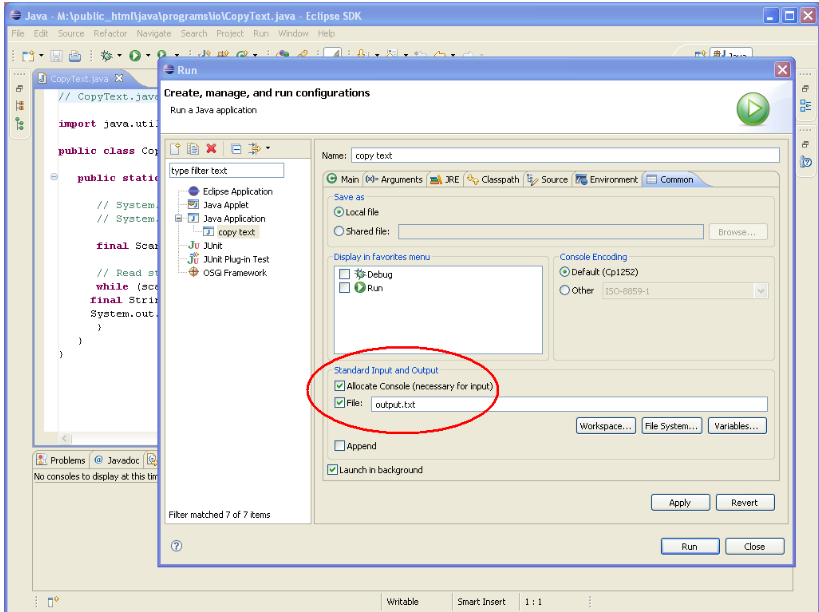
```
java GIS > output
```

Now the output is collected in the file named `output`. Since keystrokes are echoed on the display, it is sometimes hard for the user to distinguish which characters on the display correspond to the standard input and which characters to the standard output. From the point of view of the program no such confusion exists.

Eclipse

In additions to taking over the development of Java programs, Eclipse takes over the responsibility of the command shell in communicating to the operating system the command to execute, the arguments to the program, the environment of the process, the redirection of standard I/O.

Regrettable, Eclipse allows for the redirection of only the standard output, not the standard error nor the standard input. (Cutting and pasting in the console is a crude work-around.)


Java - M:\public_html\java\programs\io\CopyText.java - Eclipse SDK
Run
Create, manage, and run configurations
Run a Java application
type filter text
Eclipse Application
Java Applet
Java Application
copy text
JUnit
JUnit Plug-in Test
OSGi Framework
Filter matched 7 of 7 items
Name: copy text
Main
Arguments
JRE
Classpath
Source
Environment
Common
Save as
Local file
Shared file:
Browse...
Display in favorites menu
Debug
Run
Console Encoding
Default (Cp1252)
Other
ISO-8859-1
Standard Input and Output
Allocate Console (necessary for input)
File:
output.txt
Workspace...
File System...
Variables...
Append
Launch in background
Apply
Revert
Run
Close
Writable
Smart Insert
1 : 1

Using standard input and standard output for I/O is very flexible. The user of the program (instead of the program writer) can decide whether to type in the input or put the input in a some file. If the program was written to take input from a particular file, the user of the program has no choice.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
on the console.
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Console

Looking at the console (display screen) can be confusing

```
maelstrom> java Confusing
Characters printed b Echoed in y println() are seen
put characters Output sent to the
 on the console.  also appear
on the console.  standard error stream appears
 on the console as well.
```

Specifying the temporal order in which reads from the input stream and prints to the output stream are made can be quite tricky. An interactive program may be hard to write and requires understanding exactly how the OS implements echoing, buffering, and flushing.

Programs that don't care what the user console looks like are simpler to write. It is easier to specify the contents of streams without reference to the other streams.

standard output

```
Characters printed b
y println() are seen
 on the console.
```

standard input

```
Echoed in
put characters
 also appear
on the console.
```

standard error

```
Output sent to the
 standard error
stream appears on
the console as well.
```

For this reason I personally favor project descriptions that specify the contents of the input stream and specify the contents of the output stream independently. The programmer can choose to read and to write when it is most convenient or efficient.

I avoid assigning interactive programs and programs that prompt for input because these programs tend to require tedious detail and specification of time-dependent behavior. However, sometimes such programs are more convenient for applications used by people. Many programs do not require an interface for humans.

Writing programs that are easy to use is important. This is a topic studied in more detail in the field of human-computer interaction and in classes on building graphical user-interfaces. Here we focus on basic programming and keep things simple.

Command Line/Shell

The user asks the operating system to run a program. There are numerous variations possible when a user runs a program. Often there an operating systems provides a command line interpreter (a program!) which the user can use to direct the execution of applications by the operating system on a computer.

Redirection in Bash

Your command line shell may have lots of features.

```
 <   filename   # redirect stdin from file
1>   filename   # redirect stdout to file
1>>  filename   # redirect and append stdout to file
2>   filename
2>>  filename
&>   filename   # redirect stdout and stderr to file
&>>  filename

2>&1            # redirect stderr into stdout
```

Creating Additional Streams

It is easy for the Java program to create input and output streams associated with files in the computer's file system or with a network connection to another program. (There is practically no use for creating additional streams associated with the keyboard or display device.)

The necessary classes from the Java package `java.io` and are:

```
FileReader (String file_name)
FileWriter (String file_name, boolean append)
```

```java
String file_name1, file_name2;

try (
    final BufferedReader reader = new BufferedReader
        new FileReader(file_name1));
    final PrintWriter    writer = new PrintWriter (
        new BufferedWriter (
            new FileWriter(file_name2, false)));
) {
    while (true) {
        final String line = reader.readLine();
        if (line==null) break;
        writer.println (line);
    }
}
```

```
try {
   String file_name1, file_name2;
   BufferedReader reader = new BufferedReader (
      new FileReader(file_name1));
   PrintWriter    writer = new PrintWriter (
      new BufferedWriter (
         new FileWriter(file_name2, false)));
   while (true) {
      final String line = reader.readLine();
      if (line==null) break;
      writer.println (line);
   }
   reader.close();
   writer.close();
} catch (IOException ex) {
   System.err.println(ex);
}
```

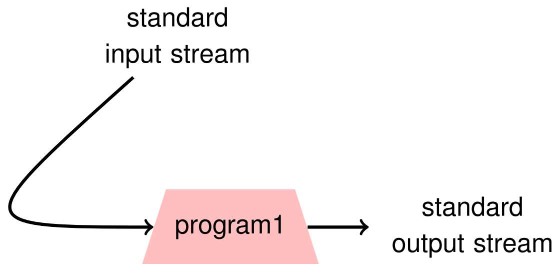
It is not possible to appreciate the value of the stream abstraction without understanding the notion of a *pipe*.

Pipes

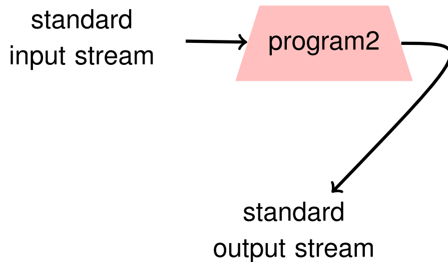
> *One of the most widely admired contributions of Unix to the culture of operating systems and command languages is the pipe, as used in a pipeline of commands.*
>
> Dennis M. Ritchie

The operating system can connect the output stream of one program with the input stream of another program; these connections are called *pipes*. Small, well-designed programs can be fit together in many different ways to accomplish complex tasks.

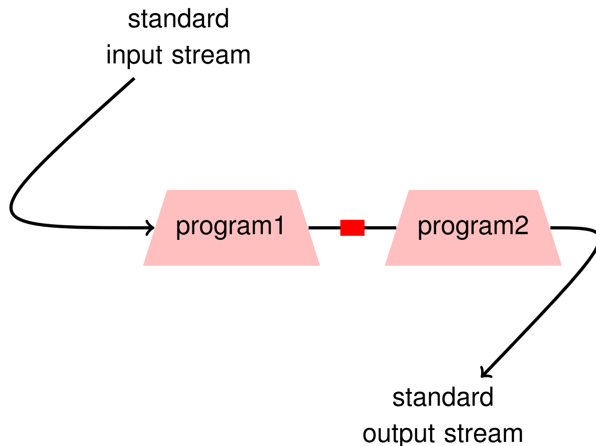
Pipes



Pipes



Pipes



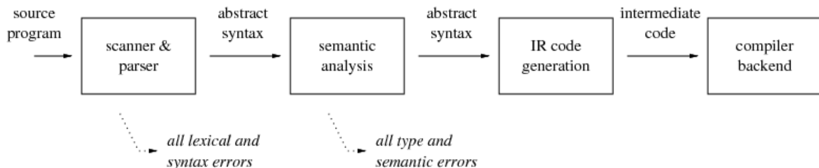
System Design

Composing programming blocks to get new programming blocks is important to building large systems.

Don't write one monolithic program to solve one problem.
Well-designed programs can work together to solve many problems without constantly writing and re-writing programs.

Success in programming can be measured by all the programs one does not have to write.

Big problems are solved by breaking them down into small problems. Compiler: preprocess, translate, code generation, assembly, link.



Graphics pipeline or rendering pipeline: transformation, clipping, texturing.

Moreover, multiple big problems can be solved by combining well-design subproblems. Good design demands finding the best subproblems.

Pipes

```
sed -e 's#//.*$##' < Program.java | \
   indent --tab=3 | nl > Listing.txt
```

Take a Java program, strip the comments, indent, and number the lines.

A Task: Histogram

1. identify words
    1.1 delete non-letters
    1.2 ignore case
    1.3 put each word on one line.
        1.3.1 separate words into lines by white space
        1.3.2 discard blank lines
2. count unique words
    2.1 group same words together
    2.2 count of each group
3. keep data
    3.1 store histogram in file
    3.2 give a count of unique words

Pipes: Another Example

```
tr -d '?"\!:,*();><' < text | \
   tr 'A-Z' 'a-z' | \
      tr ' \t/.' '\n' | \
         sed '/^$/d' | \
            sort | uniq -c | \
               sort -rn | \
                  tee histogram | wc -l
```

Take a file and make a histogram of the words.

Pipes/Data Preparation

```
wget -O - http://www.ll.mit.edu/outside.tcpdump.gz | \
  gzip -d | \
    tcpdump -q -X -r - 'tcp and ip[0]&0x0f<6' | \
      grep -v -e '^0x..[2-9a-f]' | \
        gawk '/^..:..:../{T=substr($0,0,15)} ...' | \
          sed -n -e 50000,52000p | \
            tr -d " " | tr "-" " " | \
              nl -w 3 "-s " | \
                bzip -9 > data.bz
```

1. obtain the data over WWW
2. uncompress the gzip file
3. read tcpdump file
4. exclude some packet data
5. format data on one line
6. select just 2,000 packets
7. modify spacing
8. number the lines
9. compress the results

Data Preparation

wget 159,431,527 bytes


```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
967e95a4 4312d321 7171258e 333c4021 91a44942 12
8a54baab bb8b5477 b555d584 801c50fc 01ce0cea d9
41903508 c2a88ce2 0ffc0188 a0e3729c a3e2c8e2 8e
6dfe5bc9 79e94e77 f577bfef defbee7b f775b16e eb
7f971fb9 bf99384a 3f85e7e5 38d248ba f8d42be6 6f
44a787db 69aba41b b4497477 48068557 a3ff08a0 b4
09a3dc40 1ce31fb9 082859a3 ae02174d c52190cc 83
9e4a2624 931261db fd1b8840 86296ad8 e392ba34 59
98c78583 24a5fb49 4120c9c4 5963f825 2d281945 10
d2daa0c1 deea54b5 0eda1892 34d19083 3edad2a5 1b
dbd842f3 8c403e2d 292b2aea 7d3d6f0a fe5d585b 58
ec1a9357 9adf5a40 69734d7d 4d754b0d 6d699cd6 3a
8a1a02d2 3ebf41f3 dcf9b478 5c457961 f1b87115 b4
2eb83c10 9215c943 1b10beac f0fa70b0 10ae2e19 53
cf2225f0 e92cf05a 31a89d8c 43207309 49377d9a 46
4e3a01ef 25a525a5 110aaf25 e11b846c ca26e4fc f9
60836199 bf1f604e fde1ff18 567a0867 481b1bdd 8d
f27dce55 27366587 32806106 590c3f15 e71e3c4a 52
```

Data Preparation

wget 159,431,527 bytes

```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
```

gzip 323,832,360 bytes

```
a1b2c3d4 00020004 00000000 00000000 000101d0 000
00000036 00000036 08000961 aac90800 0961aac9 002
50204875 62205465 73742050 61636b65 74202a2a 202
8f530005 37210000 00360000 00360800 0961aac9 080
00002a2a 20485020 48756220 54657374 20506163 6b6
20202000 36da8f54 00053aa8 00000036 00000036 080
002800f0 f3000000 2a2a2048 50204875 62205465 737
20202020 20202020 200036da 8f550005 3b9b0000 003
08000961 aac90028 00f0f300 00002a2a 20485020 487
6b657420 2a2a2020 20202020 20202000 36da8f55 000
01000ccc cccc0010 7b384632 0136aaaa 0300000c 200
75642e65 79726965 2e61662e 6d696c2e 65797269 652
00000001 0101cc00 04c0a801 01000300 0d457468 657
00010005 00d44369 73636f20 496e7465 726e6574 776
6e672053 79737465 6d20536f 66747761 7265200a 494
3020536f 66747761 72652028 43323530 302d492d 4c2
```

Data Preparation

wget 159,431,527 bytes

```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
```

gzip 323,832,360 bytes

```
a1b2c3d4 00020004 00000000 00000000 000101d0 000
```

tcpdump 1,217,658,595 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.1
0x0000   4500 002c 0187 0000 4006 4c08 c425 4b9e
0x0010   ac10 7169 0400 004f e7dd 7b02 0000 0000
0x0020   6002 0200 01ba 0000 0204 05b4 05b4
08:00:39.636038 172.16.113.105.finger > 196.37.75
0x0000   4500 002c 0119 0000 3f06 4d76 ac10 7169
0x0010   c425 4b9e 004f 0400 dcf6 1e21 e7dd 7b03
0x0020   6012 7fe0 88b0 0000 0204 05b4 0000
08:00:39.636238 196.37.75.158.1024 > 172.16.113.1
0x0000   4500 0028 0188 4000 4006 0c0b c425 4b9e
0x0010   ac10 7169 0400 004f e7dd 7b03 dcf6 1e22
0x0020   5010 7d78 a2d5 0000 0000 0000 0000
```

Data Preparation

wget 159,431,527 bytes

```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
```

gzip 323,832,360 bytes

```
a1b2c3d4 00020004 00000000 00000000 000101d0 000
```

tcpdump 1,217,658,595 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.1
```

grep 355,092,423 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.10
0x0000   4500 002c 0187 0000 4006 4c08 c425 4b9e
0x0010   ac10 7169 0400 004f e7dd 7b02 0000 0000
08:00:39.636038 172.16.113.105.finger > 196.37.75.
0x0000   4500 002c 0119 0000 3f06 4d76 ac10 7169
0x0010   c425 4b9e 004f 0400 dcf6 1e21 e7dd 7b03
08:00:39.636238 196.37.75.158.1024 > 172.16.113.10
0x0000   4500 0028 0188 4000 4006 0c0b c425 4b9e
0x0010   ac10 7169 0400 004f e7dd 7b03 dcf6 1e22
```

Data Preparation

wget 159,431,527 bytes

```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
```

gzip 323,832,360 bytes

```
a1b2c3d4 00020004 00000000 00000000 000101d0 000
```

tcpdump 1,217,658,595 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.1
```

grep 355,092,423 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.10
```

gawk 107,273,476 bytes

```
08:00:39.631363-4500 002c 0187 0000 4006 4c08 c425
08:00:39.636038-4500 002c 0119 0000 3f06 4d76 ac10
08:00:39.636238-4500 0028 0188 4000 4006 0c0b c425
08:00:39.636784-4500 002c 0189 4000 4006 0c06 c425
08:00:39.648472-4500 0028 011a 4000 3f06 0d79 ac10
08:00:39.648615-4500 002a 018a 4000 4006 0c07 c425
```

Data Preparation

wget 159,431,527 bytes

```
1f8b0808 40dd3637 00036f75 74736964 652e7463 70
```

gzip 323,832,360 bytes

```
a1b2c3d4 00020004 00000000 00000000 000101d0 000
```

tcpdump 1,217,658,595 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.1
```

grep 355,092,423 bytes

```
08:00:39.631363 196.37.75.158.1024 > 172.16.113.10
```

gawk 107,273,476 bytes

```
08:00:39.631363-4500 002c 0187 0000 4006 4c08 c425
```

sed, tr, nl 155,079 bytes

```
1 09:10:43.564826 451005dcf91b00003f06a57cac1070c2
2 09:10:43.566236 451005dcf91c00003f06a57bac1070c2
3 09:10:43.566385 451000baf91d40003f066a9cac1070c2
```

Unix Core Utilities

Very common programs: `cat`, `awk`, `sed`, `grep`, `sort`, `tr`, etc.
Other useful programs: `wget`, `od`, `nl`, `cut`, `paste`, `find`, `xargs`, `tee`, etc.

find

```
find . -name '*.tex'

find . -name 'project* -type f -ls

find / -name 'file' -type f

find local /tmp -name 'dir' -type d -print

find / -name 'file' \& grep -v "Permission denied"

find . \( -name '*.jsp' -o -name '*.java' \) -type f -ls

find /var/ftp/mp2 -name '*.pm3' -type f -exec chmod 644 {} \;
```

xargs

```
find . -name '*foo*' | xargs grep bar

grep bar `find . -name '*foo*'`

find . -name '*~' -print0 | xargs -0 rm

find . -name '*foo*' -print0 | xargs -O -I files mv files /tmp/tra
```

bash

```
#!/bin/bash
DBS=`mysql -uroot  -e"show databases"`
for b in $DBS ;
do
   mysql -uroot -e"show tables from $b"
done
```

Streams

1. `io/CopyText.java`
2. `io/CopyTextFile.java`
3. `io/GzipTextFile.java`

Try

```
java CopyText < input.txt | java CopyText > output.txt
```

Endian

In *Gulliver's Travels* by Jonathan Swift published in 1726, two factions within Lilliputian society are at war over the way to break eggs—at the big end, or the little end of the egg. The Emperor commanded all his subjects to break the smaller end, but resistance by traditionalists and subsequent suppression by the government resulted in civil unrest. Thus, Swift satirizes the suppression of Catholics in his day.

> *The Emperor ... published an Edict, commanding all his Subjects, upon great Penaltys, to break the smaller End of their Eggs. The People so highly resented this Law, that our Historys tell us there have been six Rebellions raised on that account; wherein one Emperor lost his Life, and another his Crown.*