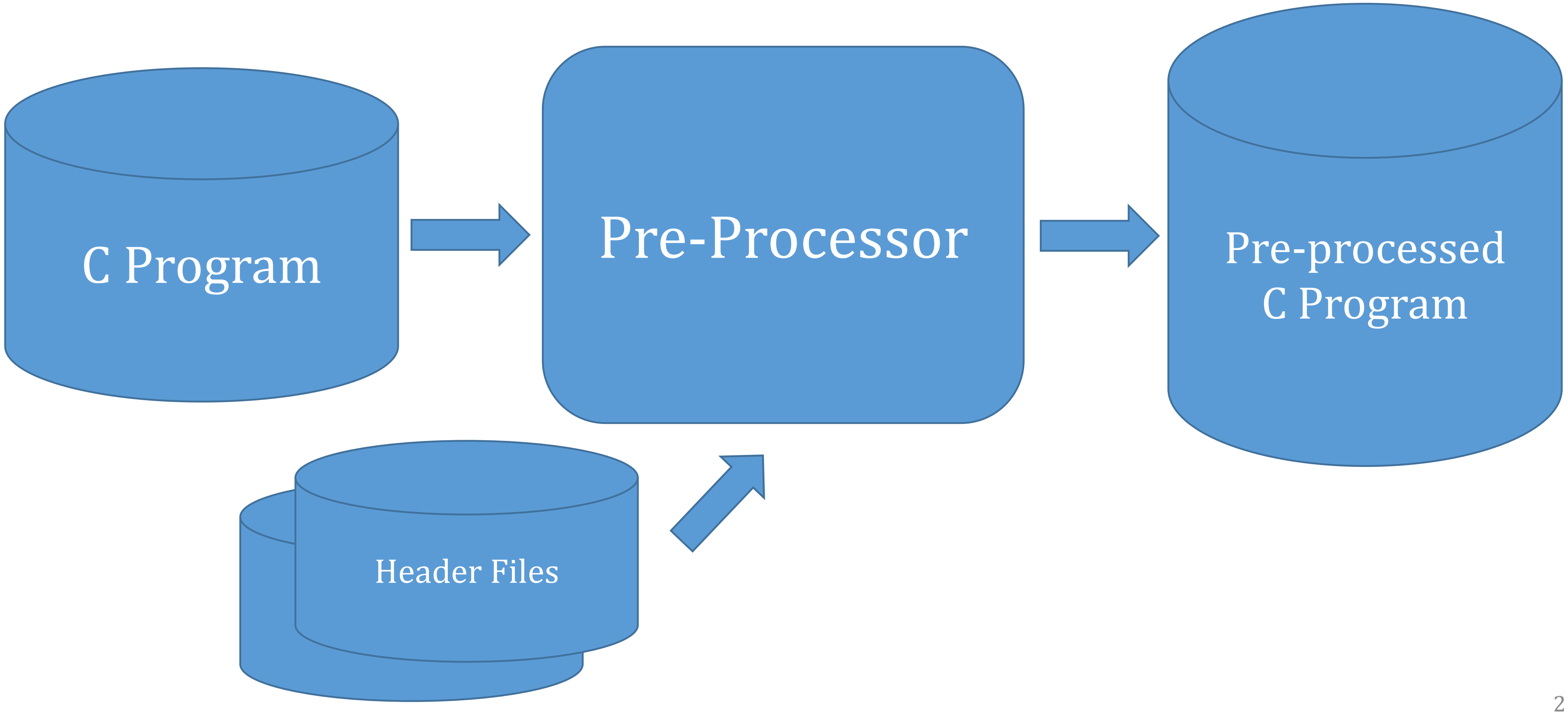
Includes & Streams

Pre-processor, Header files, and the standard Library

Input and Output Streams

Redirection and Pipes

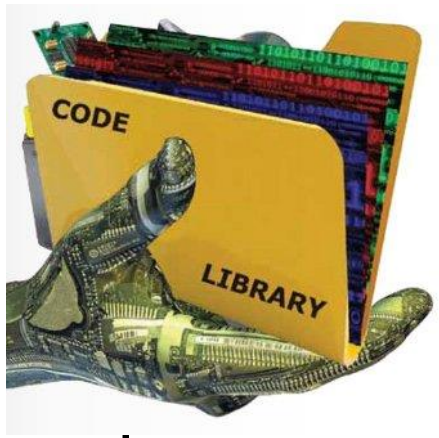
C Pre-Processor



#include

- Two flavors:
  - #include <abc.h>
    - Replace this line with the contents of file abc.h found in the system libraries
  - #include “xyz.h”
    - Replace this line with the contents of file xyz.h found in the current directory
- Concept: Write code once, use it in many different programs
- By convention, included files are called “header files” and have a “.h” file type

Header (.h) files



- May include any valid C code
    - may include further #includes
- Typically small – only what you need to invoke "outside" functions
    - Function prototype declarations
    - Sometimes, data and/or type declarations
- Typically, a header file describes a group of functions
    - "related" functions – e.g. functions which deal with Input and Output
- Typically associated with a ".c" file with the same name
    - .c file contains function definitions – more later

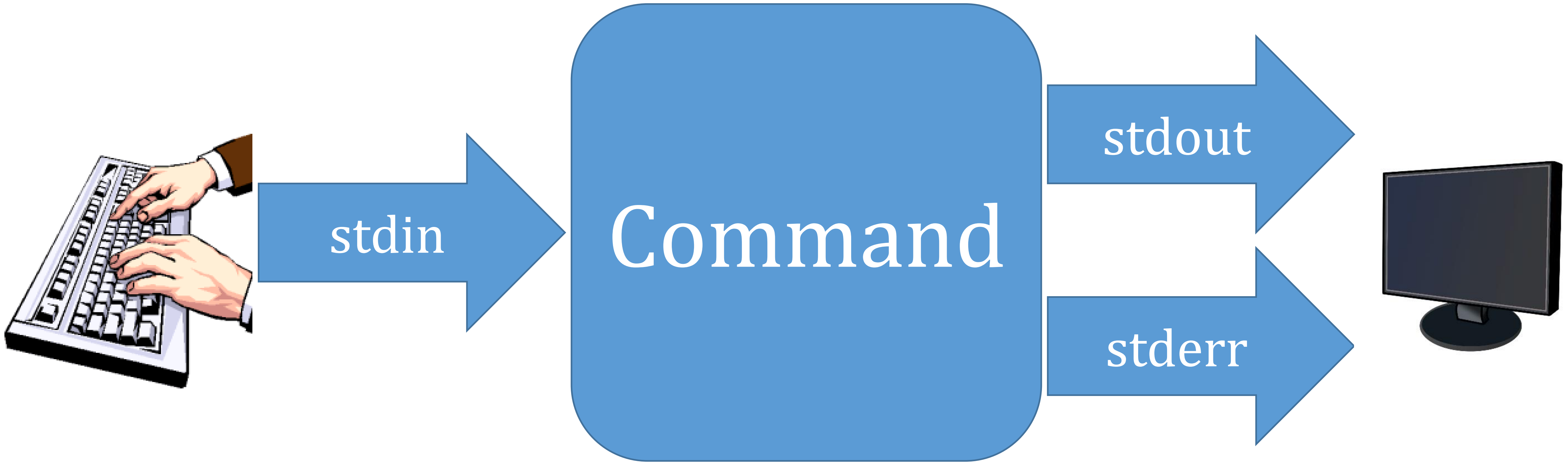
C Standard Library

- Part of C itself
  - Definitions/Descriptions included in C standard
  - Available with every C compiler
- Primarily, a set of C functions we all can use
  - Keep C language simple
  - Wrap complicated stuff in function definitions
  - We don’t need to know about the implementation of the complicated stuff
- Full description of standard library available as reference
  - Programming in C, Appendix B

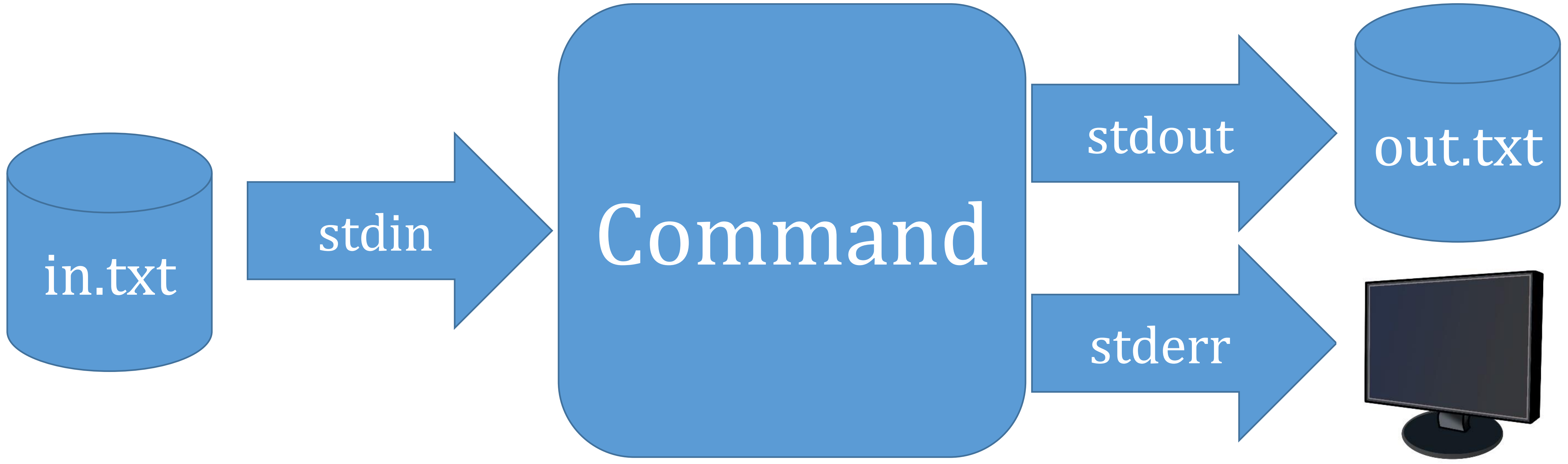
Standard Input and Output

- Need: #include <stdio.h>
- Large list of input and output functions to:
  - Read and write from a "stream"
  - Read and write from a string
  - Open a file and make a stream
  - Close a file and remove the stream
  - Create, Rename, delete, or move files
- Streams…. directional list of data (bytes)
  - input streams… provide data to program
  - output streams… program provides data

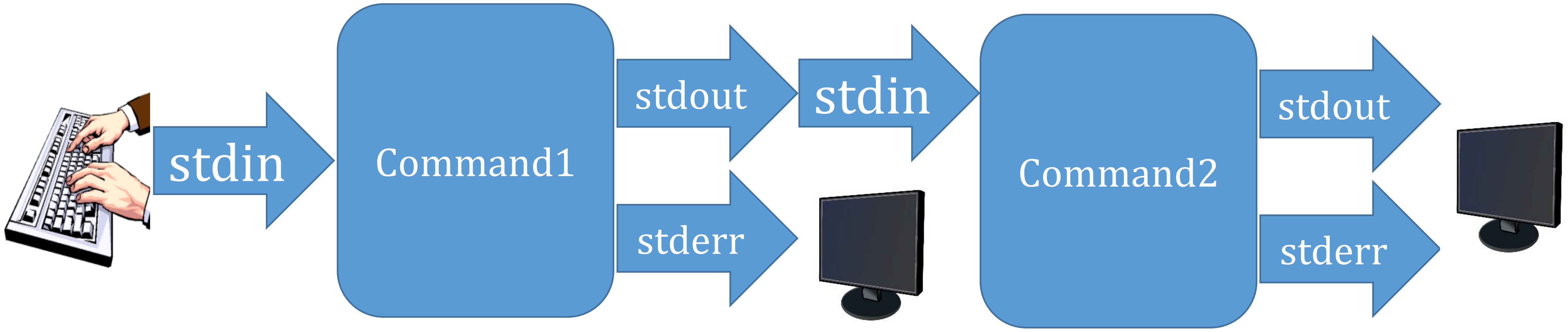
Unix “standard” streams



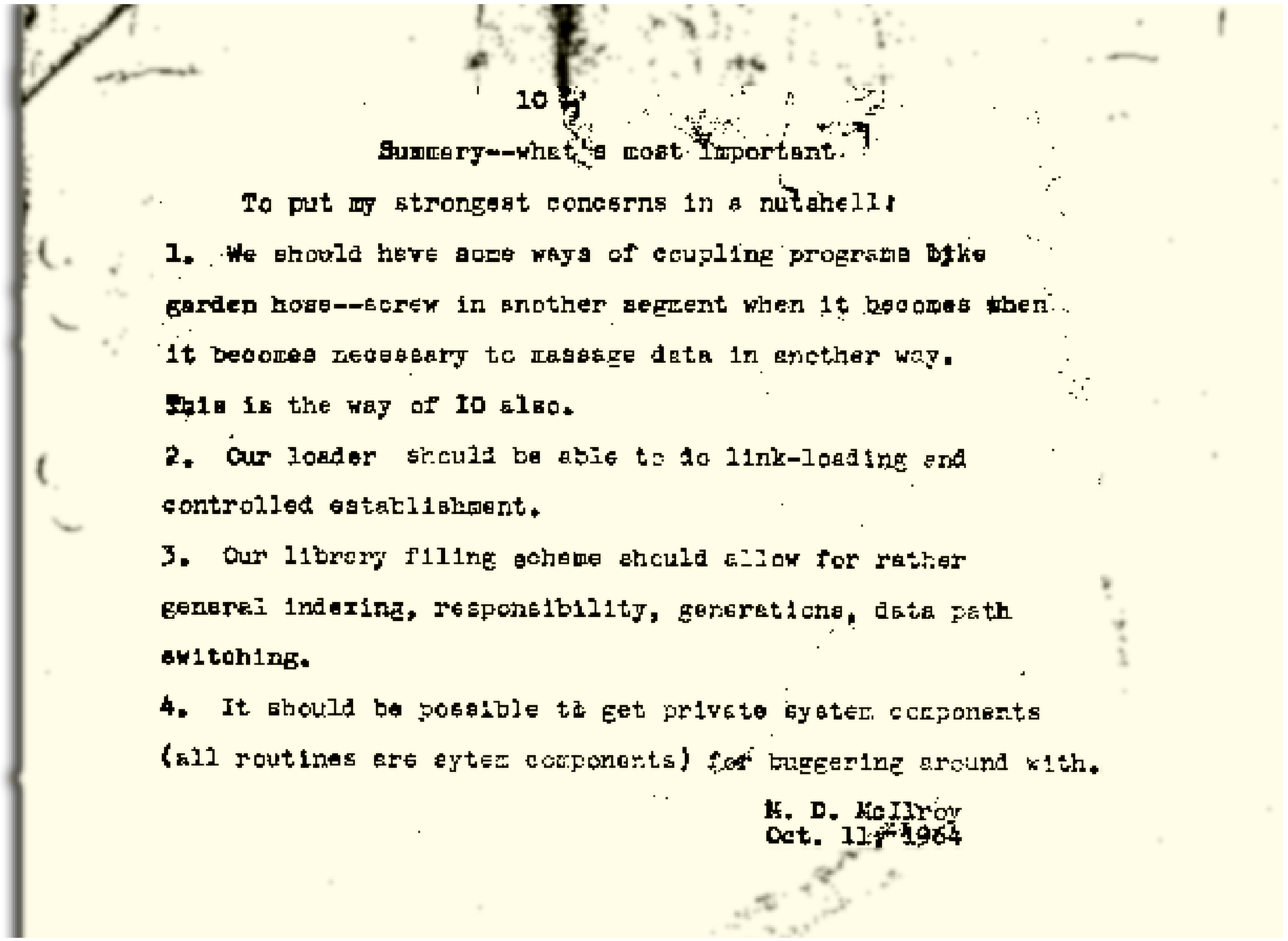
Redirection: command <in.txt >out.txt



Pipes: command1 | command2



Origin of Pipes



10

Summary--what's most important.

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.

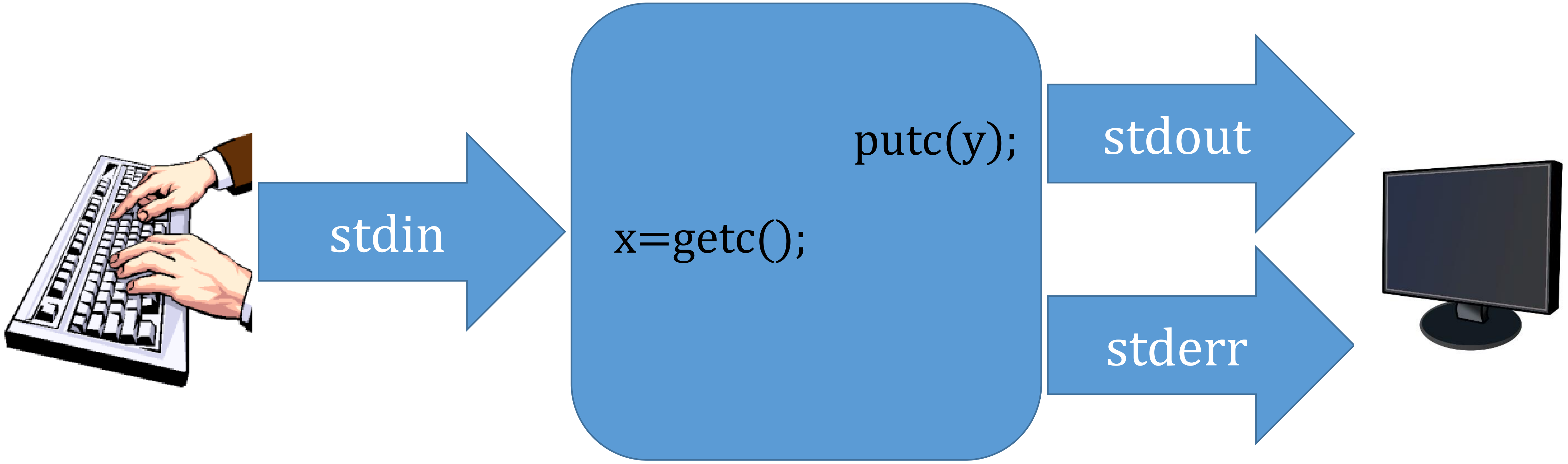
2. Our loader should be able to do link-loading and controlled establishment.

3. Our library filing scheme should allow for rather general indexing, responsibility, generations, data path switching.

4. It should be possible to get private system components (all routines are system components) for buggering around with.

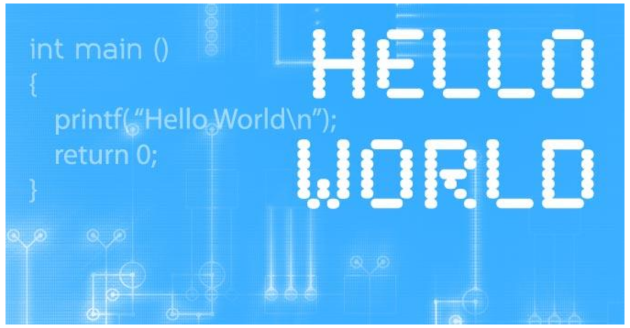
M. D. McIlroy
Oct. 11, 1964

Simple IO functions



Read from stdin, write to stdout
No opens/closes required

Print Formatted (printf)



```
int main ()
{
  printf("Hello World\n");
  return 0;
}
```

- Function with prototype: `int printf(char * str,…);`
- `str` : format string (list of characters)
- … : variable number of extra arguments

- printf "formats" the format string, using the extra arguments as input into a resulting "formatted" string
- Then, writes the formatted string to stdout

Printf Formatting

- Every "%" in formatted string starts a special "format specifier"
- Format specifiers end with a format specifier "type" (next slide)
- Format specifiers are replaced with one or more characters before printing
- Format specifiers may "consume" one of the extra arguments
- Need an extra argument for each format specifier that requires a value

Format Specifier Types

Specifier	Meaning
%	Replace with a single %
c	Replace with next single character argument
i or d	Replace with next integer argument converted to ASCII
x	Replace with next integer argument converted to hexadecimal ASCII
f	Replace with next floating point argument converted to ASCII
s	Replace with next string argument (list of characters)

```
printf("Format c=%c, x=%d (or %x), f=%f s=%s\n",'z',12,12,3.14,"that's all folks");
```

```
Format c=z, x=12 (or 0C), f=3.1400 s=that's all folks
```

Format Specifier Modifiers

- Format Specifier: %<flag><width><.precision><type>
- <flag> : optional – "-" (left justified)
- <width> : optional – minimum size of replacement
- <.precision> : optional – number of decimal places for %f before rounding

```
printf("Format <%5i> <%-5.2f> <%3c> <%-3c>\n",
        10,3.156,'a','b');

Format <   10> <3.16 > <  a> <b  >
```

Variable Size Argument Lists

- How many arguments does printf take?
  - Answer… 1 + the number of %'s in the format string
- How does this work?
  - Answer… all the magic is in #include <stdarg.h>
  - More complicated than we need to get into today

format string functions

- printf – write output to stdout
- fprintf – write output to a named stream (stdout, stderr, etc.)
- sprintf – write output to a character array variable
- scanf – read from stdin and update memory that arguments point to
- fscanf – read from specified stream, and update memory…
- sscanf – read from a character array buffer and update memory…

functions for streams

- fopen(filename,mode) – returns data of type FILE which can be used as a stream
  - filename is fully qualified, or relative to the current directory
  - mode: “r”=read, “w”=write, “a”=write append
  - Must flclose as well!

```
#include <stdio.h>
FILE mystr=fopen(“example.txt”,”r”);
if (mystr==NULL) { /* open failed! */ }
while(2==fscanf(mystr,”%s=%d”,name,&val)) {
        // handle name and value
}
fclose(mystr);
```

Resources

- Programming in C, Appendix B, Chapter 15
- Wikipedia: printf format string (https://en.wikipedia.org/wiki/Printf_format_string)