ECOL 553L

Helpful Project Things
Perl Review

Helpful Project Things

XDisk

- The HPC has “temporary” storage available for large files
- `xdisk -c query`
  - tell you how long you can get one and if you already have one
- `xdisk -c create -m <size> -d <days>`
  - create an xdisk
- `xdisk -c help`

- Alternately, true temporary storage is available in `/scratch/`
  - make a folder `/scratch/<uname>` before writing
  - deleted every night

Advanced Permissions

- We talked about chmod +x
- when you do an ls -l you see all the permissions on a file
- they are in triples of rwx, and there are three of them (user, group, everyone)

User Group Everyone

```
total 0
--w------- 1 deblasio student 0 Nov 15 12:07 200
-r-x------ 1 deblasio student 0 Nov 15 12:07 500
-rwxr-xr-x 1 deblasio student 0 Nov 15 12:07 755
-rwxrwx--- 1 deblasio student 0 Nov 15 12:07 770
-rwxrwxrwx 1 deblasio student 0 Nov 15 12:07 777
```

Advanced Permissions

- two ways of setting permissions
  - `chmod [uge][+-][rwx] <file>`
  - `chmod [0-7][0-7][0-7] <file>`

	R (4)	W (2)	E (1)	
0	0	0	0	0 + 0 + 0 = 0
1	0	0	1	0 + 0 + 1 = 1
2	0	1	0	0 + 2 + 0 = 2
3	0	1	1	0 + 2 + 1 = 3
4	1	0	0	4 + 0 + 0 = 4
5	1	0	1	4 + 0 + 1 = 5
6	1	1	0	4 + 2 + 0 = 6
7	1	1	1	4 + 2 + 1 = 7

Write only

Read only
Read and Execute

All Permissions

Installing A Package

- Similar to installing a module, like we did yesterday
- Simple steps (not always the same):
  - download the package (try using wget)
  - un-tar it (`tar -xzf <file.tgz>`)
    - x -- extract
    - z -- gzipped
    - f -- file (has to be right bedore the file name)
  - `./configure --prefix=~/bin`
    - this can change, look at the README
  - `make`
  - `make test`
    - not always there
  - `make install`

Perl Review

(hint: these may be things that will be on Tuesdays quiz)

Quiz Tuesday

- Questions taken from previous quizzes
- Because you have seen them before there will be 6-8 questions instead of 5

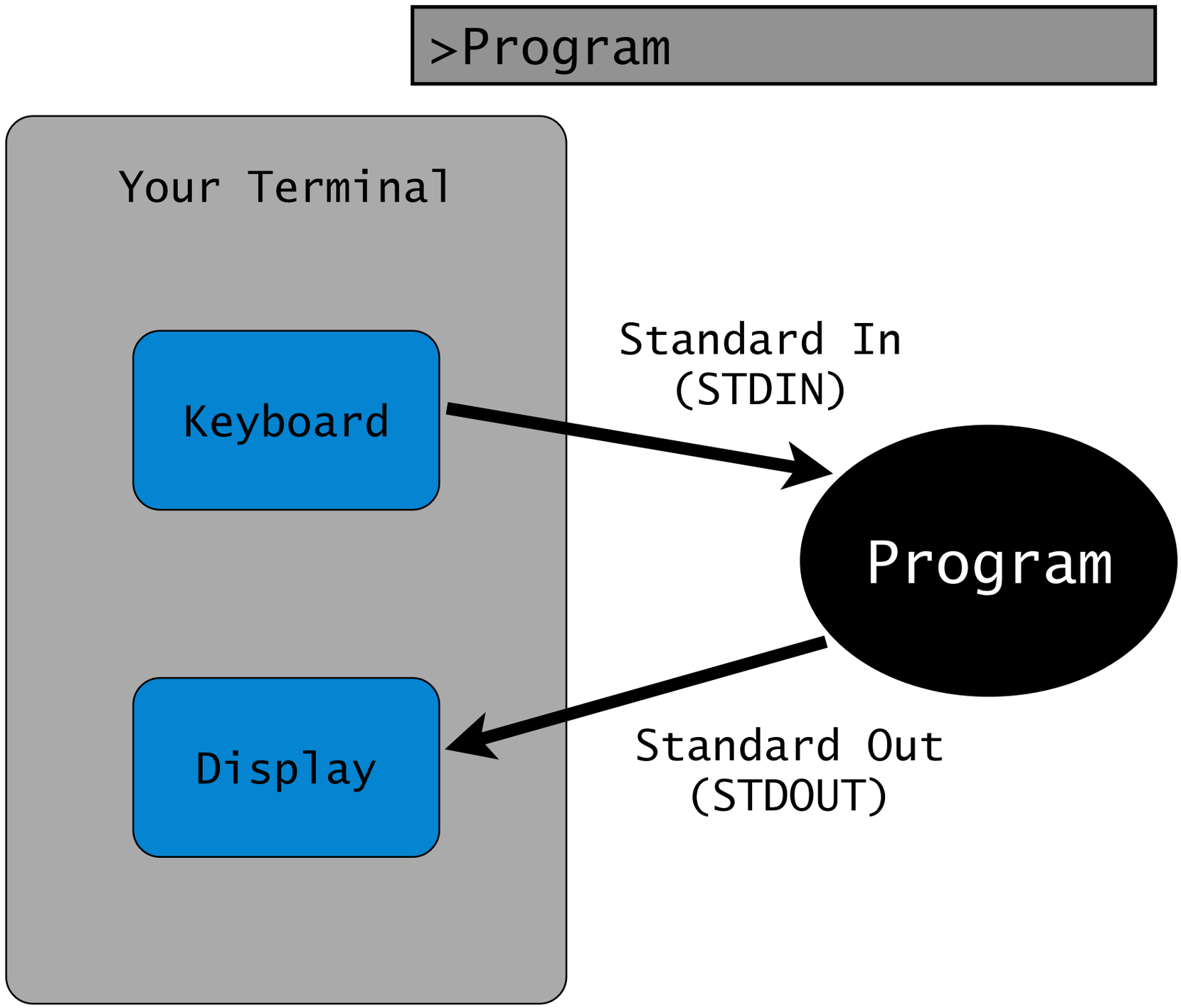
UNIX Basic Command Recap

- So far we have learned 7 basic commands:

`ssh [<login>@]<system_address>`	Log in to a remote computer
`man <command>`	Output a manual page for the specified command
`pwd`	Display the present working directory
`mkdir <dirname> …`	Make new directory/directories named dirname …
`ls [<dirname> …]` `ls -l`	List directory contents (of dirname(s)) Show a long listing of directory contents
`cp <srcfile> … <dest>` `cp –r <srcdir> … <dest>`	Copy source file(s) to destination Recursively copy source dir(s) to destination
`logout`	Log out of the Unix system

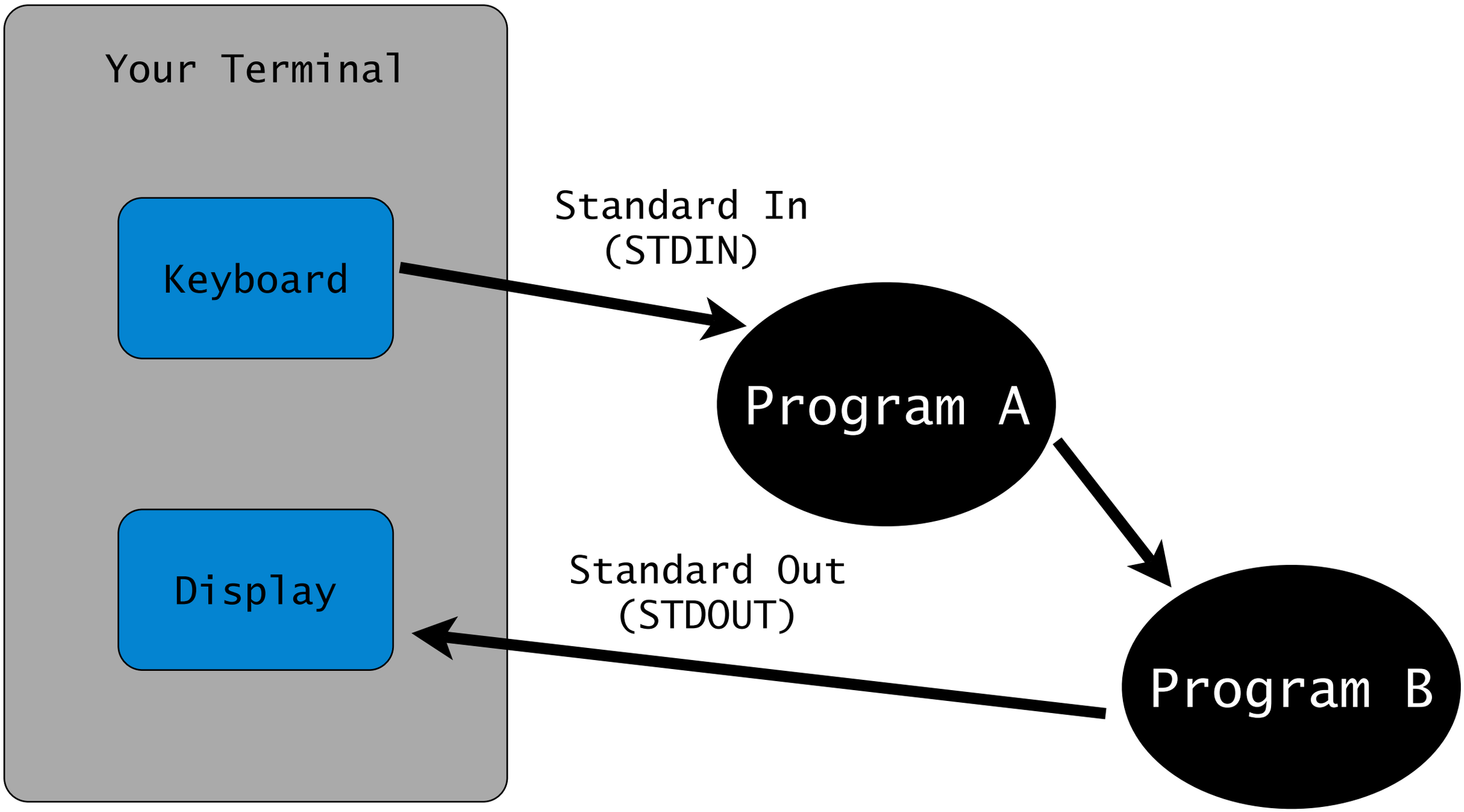
- How many arguments does each of these commands take?

Program input and output



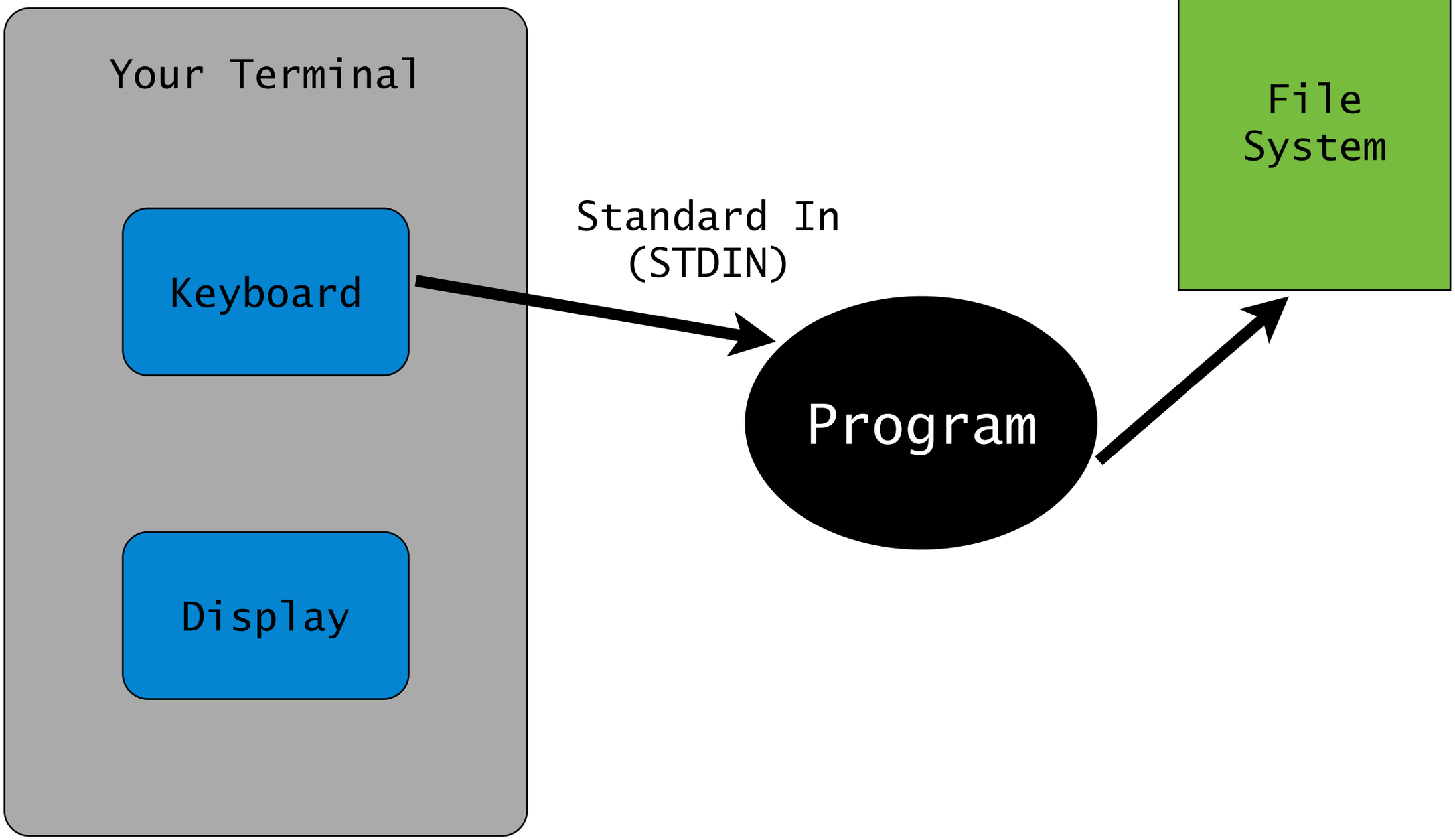
Program input and output

```
>Program_A | Program_B
```



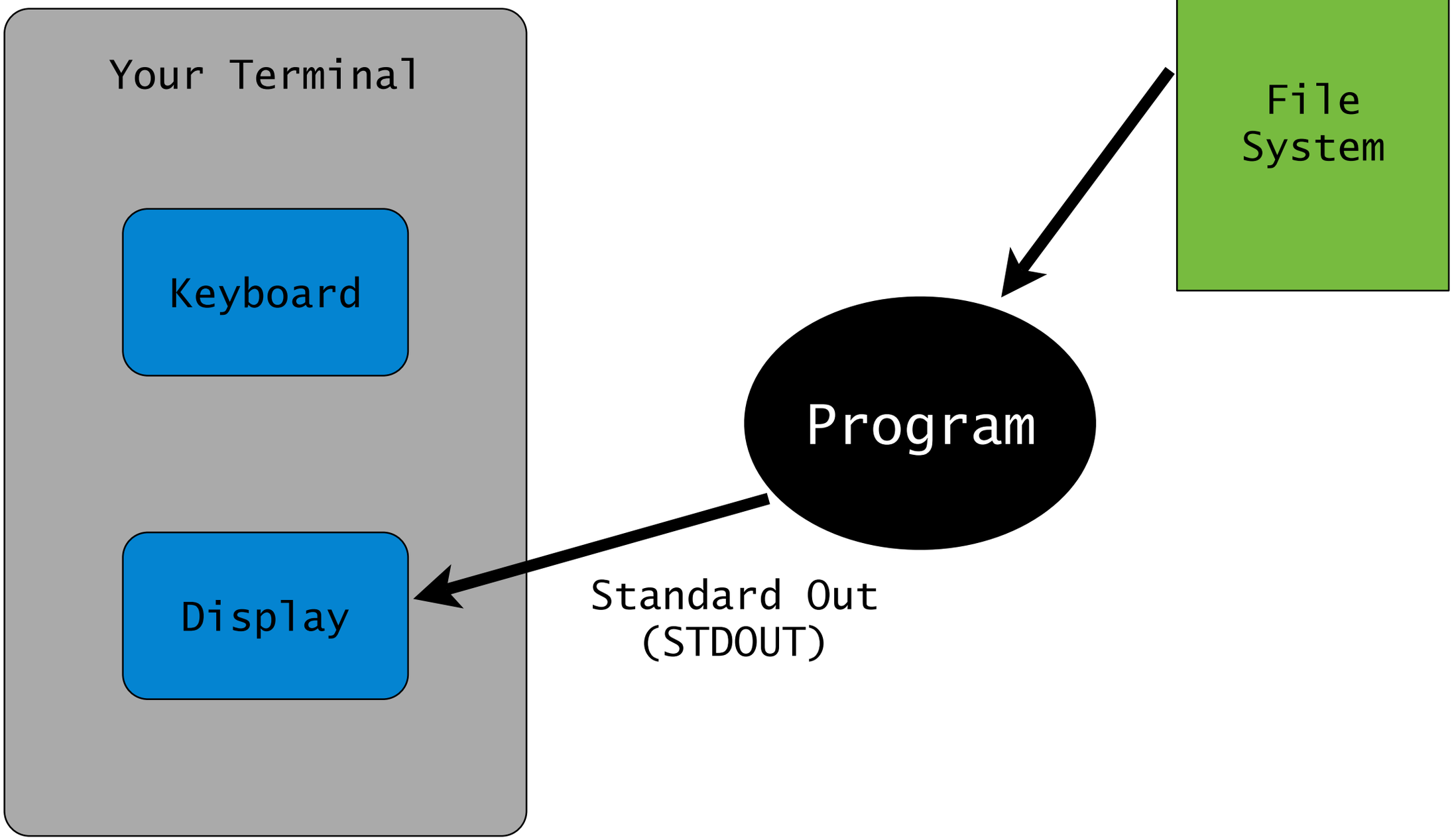
Program input and output

```
>Program > file
```

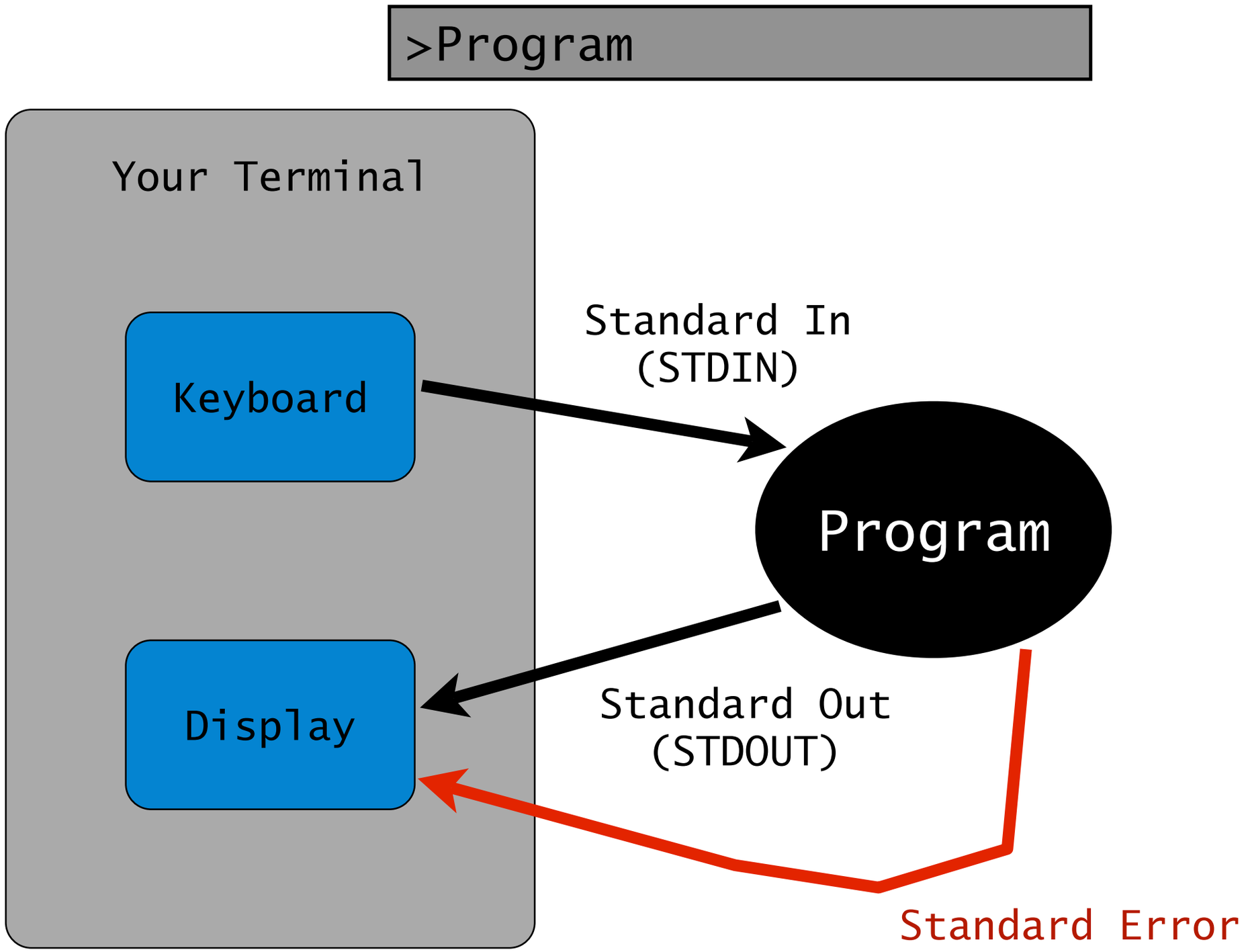


Program input and output

```
>Program < file
```

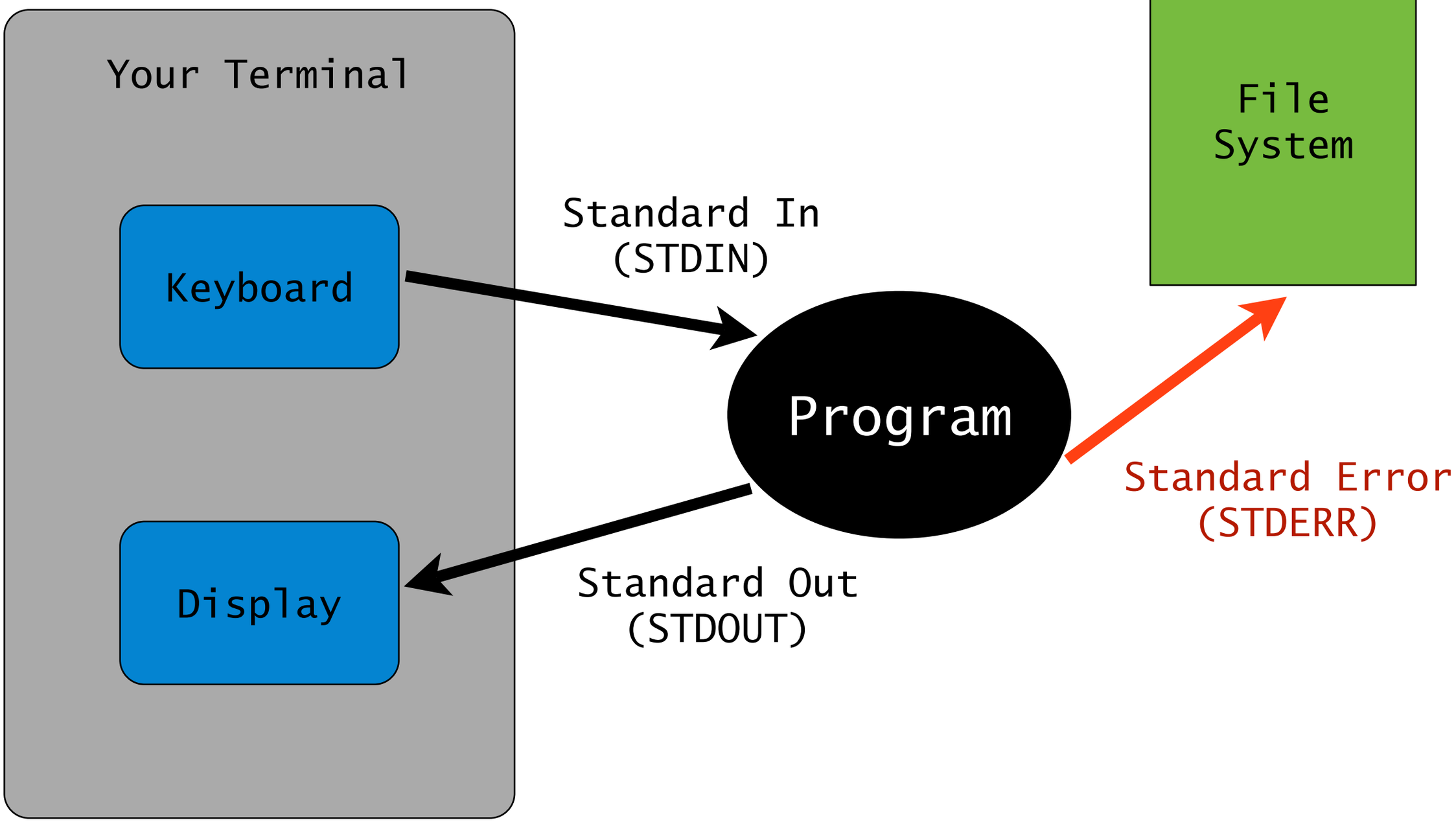


Program input and output



Program input and output

```
>Program 2> file
```



Sample PBS Batch Submit File to run BLAST

```
#PBS -N test_blast
#PBS -l
#PBS -l cput=96:0:0
#PBS -l walltime=24:0:0
module load blast
cd ~deblasio/blast_dir
sta -out seqs.bln \
-evalue 1e-30 -num_descriptions 20 -num_alignments 20 \
-num_threads 11
```

Notice that the number ... essors requested (8) in one ... the ... reads option ... e `nt` database ... s a lot of ... y!

Where my group is your PIs netid

Submitting and Monitoring PBS Jobs

- To submit a PBS job, use the qsub command with your PBS script as the argument
  - PBS will return a job number

```
qsub <run_blast.sh>
```

- To view the queue of PBS Jobs, run the command

```
qstat -a
```

- To see details about a running job, use:

```
qstat -f <JobNumber> | more
```

Files Produced by PBS

- When your job finishes, you will see 2 files from `PBS`. The files are named according to the *<JobName>* you specify with the `-N` option in the `PBS` submit file and the *<JobID>* Number assigned by `PBS`.
  - *<JobName>*`.e`*<JobNumber>*
    - This file contains Error messages. Look here first if your job did not run as expected. If the size of this file is zero, there were no error messages.
  - *<JobName>*`.o`*<JobNumber>*
    - This file contains Standard Output messages.
    - The following two lines are at the beginning of the .o file and can be ignored:

```
Warning: no access to tty (Bad file descriptor).
```

- Look at the .o file to see how much CPU time your PI's group was charged for the run and how much time remains in your group's monthly allocation.

Going back to flow control

- We know an `if` statement can run some code when come condition is met.
  - `if(...){ ... }`
- What if a condition isn't met?
  - `if(...){ ... }`
  - `else(...){ ... }`
- What if the first isn't met, but the a second should be
  - `if(...){ ... }`
  - `elsif(...){ ... }`
  - `else{ ... }`

A Bit About Variable Scope

- The scope of a variable refers to its lifetime within a script, i.e. when and where it is accessible.
- Sometimes you will want to restrict the scope of a variable using the my keyword, so that it won't interfere or collide with another variable having the same name. This will be very important later when we learn about subroutines.
- Here is an example:

```
my $name = "fred";  # Here $name has File Scope
print "name is $name \n";
{
  my $name = "lucy"; # Here $name has block scope
  print "inside the block, name is $name \n";
}
print "out here, name is $name \n";
```

Short cuts with Operators

- We can add to a scalar variable like this:
  - `$sum = $sum + 8;`
- Or we can use shorthand that combines the operator and =
  - `$sum += 8;  # add 8 to sum`
- This works for most operators:
  - `$product  *=  12;  # multiply product by 12`

- Adding and subtracting one are so commonly done that there are even shorter shortcuts!
  - `$year  =  $year + 1;  # add 1 to year`
  - `$year  +=  1;  # add 1 to year`
  - `$year++;  # add 1 to year`
  - `$total--;  # subtract 1 from total`
- These are called autoincrement ++ and autodecrement --

Common Mistakes to watch out for!

- Forgetting the `;` at the end of a statement
- Mismatched parentheses `( )`, braces `{ }`, brackets `[ ]`, or quotes `' '`, `" "`, `` ` ` ``
- Forgetting the `$` at the beginning of a variable name (bare word error)
- A mistake on the first line of the script (improper specification of the Perl interpreter – or possibly a space where it shouldn't be)
- Forgetting to close an output file (can cause missing output)
- Saving your script to a different folder than the folder you are running it from!
- For more ideas, see "Beginning Perl", chapter 9

Stepping through Key, Value pairs in a Hash

- To step through each key, value pair in a hash, use a foreach loop and the keys function:

```
while ( my ($key, $value) = each(%hash) )
{
        print "$key => $value\n";
}
```

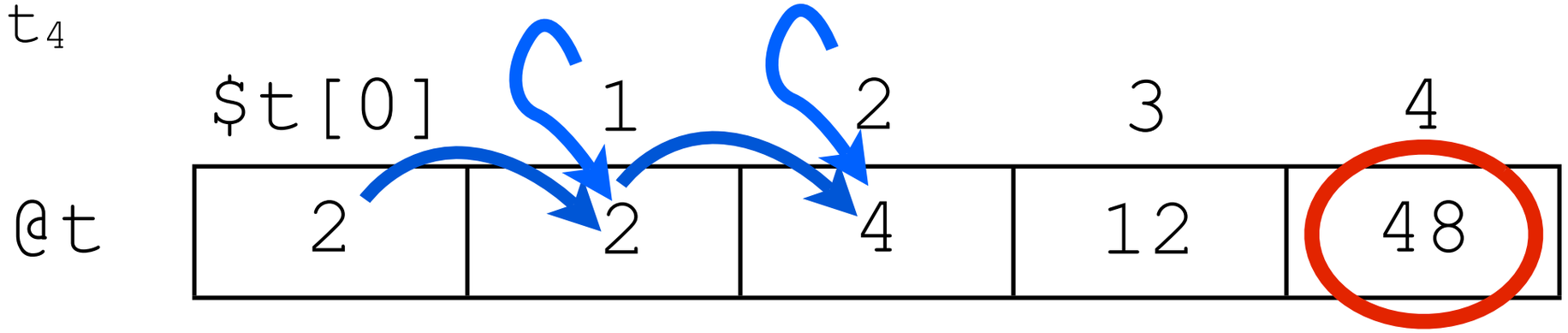
- TIMTOWTDI:

```
foreach my $key ( keys %hash ) {
        my $value = $hash{$key};
        print "$key => $value\n";
}
```

- Note that hash elements are not ordered!

Dynamic Programming

- Using previous results to solve a new problem
- lets solve the problem
  - $t_i = t_{i-1} * i$
  - let $t_0 = 2$
  - find t_4




```
my @t;
$t[0] = 2;
foreach my $i (1 ... 3){
   $t[$i] = $t[$i-1] * $i;
}
```

More Perl functions

- Perl has many useful functions. We've seen some already: `chomp, push, pop, shift, unshift`. Can you think of any others?
- Functions can take arguments or parameters, perform some operations on them, and return results.
- Examples:
  - `$last_item = pop( @arr );`
  - `@sorted = sort( @arr );`
  - `if ( defined( $var ) ) { … }`
  - `chomp( $line );`
  - `@keys = keys( %id_hash );`

The Perl `system()` function

- Perl's `system` function runs an external command from inside a script. The `system` function returns an error code value, with zero normally meaning that the command executed without error. Run '`perldoc -f system`' for more info.

- Example:

```
# Run blastn command for 2 sequences
#(no blast database!)
$command = "blastn -query seqA.fa -subject seqB.fa -out  AB.bln";
$return_code = system($command);
if ($return_code) {
   print "blastn returned $return_code \n";
}
```

Numeric sorting

- Earlier we learned about the sort function:

```
@items = ("Greg Bear",  42,  "X",  3.5e-107);
@sorted = sort (@items);
print "The sorted items are @sorted \n";
```

- By default, sort does an alphabetical sort. If we want to sort numerically we have to use the numeric comparison operator <=> :

```
@nums = (  42,  17,  3.5e-107, 4e20, 6.6);
@sorted = sort {$a  <=> $b} @nums;     # sort ascending
print "The numbers low to high are @sorted \n";
```

- For descending numeric sort, exchange `$a` and `$b`:

```
@nums = (  42,  17,  3.5e-107, 4e20, 6.6);
@sorted = sort {$b <=> $a} @nums;
print "The numbers high to low are @sorted \n";
```

Big Arrays (multiple dimensions)

- Sometimes, we need more than one dimension in a table of data
  - i.e. if each row in a TSV is an array, then the whole file is a 2-D Array or an **Array of Arrays**
- Referencing a 2-D array is straight forward
  - `$2Darr[$i][$j]` **# the `$i`-th row, `$j`-th column**
- Setting up an array of this sort is a little harder
  - Static

```
my @2Darr = (['A','B','C'] , ['D','E','F']);
```

  - Push

```
my @arr1 = ('A','B','C');
my @arr2 = ('D','E','F');
push @2Darr, [@arr1];
push @2Darr, [@arr2];
```

What size is @_?

- Turns out @_ can be larger or smaller than expected
- in our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
- How would we fix that?

What size is @_?

- Turns out @_ can be larger or smaller than expected
- in our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
- How would we fix that?

Solution 1: Error

```
sub max {
  if (@_ != 2) {
    die "WARNING! max should get exactly two arguments!\n";
  }
  if($_[0]>$_[1]){ return $_[0]; } else { return $_[1]; }
}
```

What size is @_?

- Turns out @_ can be larger or smaller than expected
- in our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
- How would we fix that?

Solution 2: Iterative

```
sub max {
  my($max_so_far) = shift @_;  # the first one is the largest yet seen
  foreach (@_) {               # look at the remaining arguments
    if ($_ > $max_so_far) {    # could this one be bigger yet?
      $max_so_far = $_;
    }
  }
  return $max_so_far;
}
```

What size is @_?

- Turns out @_ can be larger or smaller than expected
- in our previous version of max, the call to max(1,2,3) would not throw an error, but would return a wrong answer
- How would we fix that?

Solution 3: Recursive

```
sub max {
  my $a = shift @_;
  my $b = shift @_;
  my $max = $a;
  if($b > $a){ $max = $b; }
  if(scalar(@_) > 0){
    unshift @_, $max;
    $max = max(@_);
  }
  return $max;
}
```

Perl backtics ` `

- If you need your Perl script to capture the output of an external program, use backtics `` instead of the system function. The backtics surround a command string, and the output of the command is returned as an array of lines.
  - **Backtics are distinct from single quotes!**
- Examples:

```
# Run blastn and capture results in array
@result = `blastn -query seqA.fa -subject seqB.fa`;
$hit_count = 0;
foreach $lin (@result) {
   $hit_count++;
   print "$hit_count\t$lin";
}

# Run EMBOSS  msbar program to mutate $seqfile sequences
@result = `msbar $seqfile -count 10 -point 4 -stdout -auto`;
print OUTFILE "@result\n";
```

Review: Symbols used in Patterns

`=~`	Binding operator to match pattern against string **e.g.** `if ($str =~ /$pattern/) { … }`
`/ /`	Commonly used pattern delimiters
`/ /i`	Case insensitive match
`|`	Alternation, e.g `/C|G/`
`[ ], [^ ]`	Character class, e.g. `[CG]` or negated class `[^ATN]`
`*`	Match `0` or more occurrences of the previous element
`+`	Match `1` or more occurrences of the previous element
`?`	Match `0` or `1` occurrences of the previous element
`{n,m}`	Match `n` to `m` occurrences of the previous element
`\d, \s`	Match digit, whitespace character

Special Characters in Regular Expressions

.	Match any single character
^	Anchor match at beginning of string
$	Anchor match at end of string
?	Match preceding element 0 or 1 time
*	Match preceding element 0 or more times
+	Match preceding element 1 or more times
{n,m}	Match preceding element n to m times
[ ]	Match any character in character class
[^]	Match any character NOT in character class
( )	Group and capture expression
|	Match either expression preceding or following
\	Escape the character immediately following \

Perl Pattern Substitution

- In addition to pattern matching capabilities, Perl can do pattern substitution. For substitution, you use `s` in front of the pattern, and provide the substitution string after the pattern, followed by a final `/`
- Only the first match to the pattern gets substituted unless the g modifier is specified.
- pattern substitution `($str =~ s/pattern/substitution/)`:
  - `$seq =~ s/CAG/1234/;`
    `# change 1st upper case CAG to 1234`
  - `$seq =~ s/CAG/1234/i;`
    `# change 1st mixed case CAG to 1234`
  - `$seq =~ s/CAG/1234/g;`
    `# change all occurrences of upper case CAG to 1234`
  - `$seq =~ s/CAG/1234/gi;`
    `# change all occurrences of mixed case CAG to 1234`
- Using pattern substitution to count occurrences
- Perl pattern substitution also counts the number of substitutions it finds:
  - `$count = ( $seq =~ s/ABC/VWXYZ/ );`
  - `$count = ( $seq =~ s/ABC/VWXYZ/g );`
  - `$c_count = ( $seq =~ s/C/C/gi );`
  - `$g_count = ( $seq =~ s/G/G/gi );`