printf Formatting

- Precision: \% . \# f
- Width: \% f, \% d
  - Note: Entire width
- Zero-padding: \% 0 d
- Left-justification: \% - \# d
- Various combinations of the above

Formatting Example (1)

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%f</td>
<td>1.23456789</td>
</tr>
<tr>
<td>%.10f</td>
<td>1.23456789</td>
</tr>
<tr>
<td>.2f</td>
<td>1.23456789</td>
</tr>
<tr>
<td>%d</td>
<td>12345</td>
</tr>
<tr>
<td>%10d</td>
<td>12345</td>
</tr>
<tr>
<td>%2d</td>
<td>12345</td>
</tr>
<tr>
<td>%f</td>
<td>1.23456789</td>
</tr>
<tr>
<td>%8.2f</td>
<td>1.23456789</td>
</tr>
</tbody>
</table>

Formatting Example (2)

<table>
<thead>
<tr>
<th>Format</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%d:%d</td>
<td>1 and 5 &gt;1:5&lt;</td>
</tr>
<tr>
<td>%02d:%02d</td>
<td>1 and 5 &gt;01:05&lt;</td>
</tr>
<tr>
<td>%10d</td>
<td>12345   &gt;12345&lt;</td>
</tr>
<tr>
<td>%10d</td>
<td>12345   &gt;12345&lt;</td>
</tr>
<tr>
<td>%10d</td>
<td>12345   &gt;12345&lt;</td>
</tr>
<tr>
<td>%10d</td>
<td>12345   &gt;12345&lt;</td>
</tr>
</tbody>
</table>

Shell Signal Handling

- Ctrl-C sends SIGINT
- Ctrl-D sends EOF, not a signal
- SIGTERM, SIGTTIN, SIGTTOU, SIGTSTP, SIGQUIT
- Children inherit parent’s signal masks
  - If the shell ignores ctrl-c, so does its children
  - Set the signals back to default before you replace the child process

Check errno

- `execvp` fails for many reasons
  - E2BIG, EACCES, EAGAIN, EFAULT, EINVAL, ELOOP, ENAMETOOLONG, ENOENT, ENOLINK, ENOTDIR, ENOEXEC, ENOMEM, ETXTBSY
- “Command not found” should only be printed in response to ENOENT
**Tokenizer**

typedef struct tokenizer {
   // the string to parse
   char *str;
   // position in string
   char *pos;
} TOKENIZER;

char *get_next_token(TOKENIZER *tokenizer);
//if current char is a delimiter, just return it
//else go until next char is a delimiter
//return the substring without white spaces
//return NULL when string ends

**How the Tokenizer is used**

```c
int parse() {
   int i = n = 0;
   line = readline(PROMPT);
   if(line == NULL) {
      //ctrl-d
      return 0;
   }
   if(strcmp(line,"") == 0) {
      return 0;
   }
   add_history(line);
   t = init_tokenizer(line);  // how many tokens
   while(get_next_token(t) != NULL) {
      n++;
   }
   malloc(sizeof(char)*(n+1));
   t = init_tokenizer(line);
   while((toks[i++]=get_next_token(t))!=NULL) {
      return n;
   }
}
```