The Seven-Item Limit

coursera.org/learn/interacting-system-managing-memory/supplement/GN2IS/the-seven-item-limit

One of the reasons that abstraction is crucial to large systems is that it breaks them down into pieces that we can think about at one time. Psychologists have found that the human brain is generally limited to thinking about approximately seven things at any given time. However, the "things" in this limitation are in terms of pieces of information with semantic meaning to the person thinking about them. For example, it is difficult to remember the sequence of letters *zyqmpwntyorprs*, but easy to remember the word *unsurmountable*, even though they both have the same number of letters. The word *unsurmountable* is one logical piece of information, thus requires you to remember only one thing, while the gibberish *zyqmpwntyorprs* requires you to remember each individual letter, as it is not a meaningful word. Meanwhile, even though *dzqf* is also gibberish, you can easily remember because it has few letters.

Note that being able to think about a function is important in a variety of contexts. If a function's complexity exceeds your cognitive capacity, it becomes very difficult to write—there are just too many things going on to keep straight in your head as you try to generalize, and patterns become difficult to find. However, this complexity is also a problem when you want to understand what code that is already written does. If you write the code, and it does not work, understanding it is crucial to effective debugging. You may also need to understand code to modify it. Remember that real software has a long life-span—you might need to go back to code you (or someone else) wrote years ago to change it in support of a new feature.

Abstracting code out into functions (or other larger modules) provides a way to wrap the entire behavior up into one logically meaningful thing—the resulting function has a well-defined task, and you can think about it in terms of what it does (its interface) rather than thinking about the details of how it does them. Having the function as one logical unit you can think about means that it only counts for one in the limit of the seven things you can think about at once.

To see this principle in action in a programming context, consider the following code (do not invest too much effort in trying to puzzle out what it does):

```
roster_t * readInput(const char * fname) {
  FILE * f = fopen (fname, "r");
  size_t sz = 0;
  char * ln = NULL;
  char * endp = NULL;
  if (f == NULL) {
    return NULL; //Could not open file->indicate failure
  }
  roster_t * answer = malloc(sizeof(*answer));
  answer->numStudents = readAnInteger(f);
  if (getline (&ln, &sz, f) == -1) {
    fprintf(stderr,"Could not read a line (expecing a number)\n");
      exit (EXIT_FAILURE);
  }
  answer->numStudents = strtol(ln, &endp, 10);
  //skip whitespace
  while (isspace(*endp)) {
    endp++;
  }
  if (*endp != '\0') {
    fprintf(stderr, "Input line was not a number\n");
    exit (EXIT_FAILURE);
  }

  answer->students = malloc(answer->numStudents * sizeof(*answer-
>students));

  if (answer->students == NULL) {

    free(answer);

    return NULL;

  }

  for (int i =0; i < answer->numStudents; i++) {

    student_t * s= malloc(sizeof(*s));

    s->name = NULL;

    getline(&s->name, &sz, f);

    char * p = strchr(s->name, '\n');

    if (p != NULL) {

      *p = '\0';

    }

    if (getline (&ln, &sz, f) == -1) {

      fprintf(stderr,"Could not read a line (expecing a number)\n");

      exit (EXIT_FAILURE);

    }
```

This giant function is a huge mess! Not only is it ridiculously difficult to understand what it does, but it also duplicates code—performing the same task in multiple places. Duplication of code is generally bad for many reasons. The most straightforward reason not to duplicate code is that you duplicate the effort to write it. Perhaps more importantly, it is more difficult to maintain—if you need to change something you have to remember (or your development teammate needs to figure out) to change it in two places. Finally, if you have an error in your code you have to fix it in multiple places.

This giant mess of code really should be about four functions. Keep this mess in mind as you read that section, and contrast the difficulty in understanding this piece of code with understanding the corresponding four functions in that example.

✓

Completed