Version 0.9

Interactive Input in C

A standard task in C programming is to get interactive input from the user; that is, to read in a number or a string typed at the keyboard. One method is to use the function `scanf`. For example to get an integer, we might use (`#include <stdio.h>` is implied):

```
int answer;  /* an integer chosen by the user */
scanf ("%d", &answer);
```

This works fine if the user actually types a number, but is problematic if they make a mistake and type a letter instead. One can do better by checking the return value of `scanf`, which is zero if no conversions were assigned, but it is difficult to recover and get corrected input.

We advocate an alternative to `scanf`, described in "Practical C Programming" by Steve Oualline, which uses `fgets`. Here's an example:

```
char line[100];         /* a line to be read from the terminal */
int answer;             /* an integer chosen by the user */
fgets (line, sizeof(line), STDIN);      /* read in a line */
sscanf (line, "%d", &answer);           /* scan for the integer */
```

The input procedure has been converted to two steps:

1. `fgets` reads in a line terminated by an end-of-line character (e.g., "enter");
2. `sscanf` scans the line for the desired input (which could include numbers or strings).

Some comments:

- The return value of sscanf is the number of input items assigned. Zero will be returned if an alphabetic character is entered for a "`%d`" conversion and `EOF` is returned is nothing if entered before an end-of-line character.
- `sizeof(line)` is the dimension of line (*not* the length of the input string). As an argument to `fgets`, it represents the maximum number of characters to read, including the end-of-line character.
- `STDIN` is "standard input," which tells `fgets` to read from keyboard input. If we wanted to read from a file instead, we could do something like:

```
FILE * input;           /* file handle for input file */
char line[100];         /* a line to be read from a file */

input = fopen ("filename.dat", "r");    /* open the file for reading */
fgets (line, sizeof(line), input);      /* get the line */
```

The following code fragment shows how we might use the return from `sscanf` to request input until we're satisfied with the response:

```
line char[100];         /* a line to be read from the terminal */
int answer;             /* an integer chosen by the user */

answer = -1;            /* set answer to a value that falls through */
while (answer != 0)     /* iterate until told to move on */
  {
    printf ("\nSample menu:\n");
    printf (" [1] Do something");
    printf (" [2] Do something else");
    printf ("\nWhat do you want to do? [0 for nothing] ");

    fgets (line, sizeof(line), STDIN);      /* read in a line */
    sscanf_result = sscanf (line, "%d", &answer);       /* get answer */
    if ((sscanf_result == 0) | (sscanf_result == EOF))
      {
        /* either a non-integer entered or an end-of-line */
        printf ("\n *** You have to enter an integer! ***\n");
        answer = -1;      /* set answer to a value that falls through */
      }

    switch (answer)
      {
      case 0:
        break;
      case 1:
        printf (" Doing something ... ");
        break;
      case 2:
        printf (" Doing something else ... ");
        break;
      default:    /* keep going if answer is not 0, 1, or 2 */
        break;
      }
  }
```