I/O & Testing

Readings: CP:AMA 2.5

Course Notes: Appendix A.6

I/O

Input & Output (***I/O*** for short) is the term used to describe how programs *interact* with the "real world".

A program may interact with a human by receiving data from an input device (like a keyboard, mouse or touch screen) and sending data to an output device (like a screen or printer).

A program can also interact with non-human entities, such as a file in secondary storage (*e.g.,* a hard drive) or even a different computer (*e.g.,* a website).

Output

We have already seen the `printf` function (in both Racket and C) that prints formatted output via placeholders.

In C, we have seen the placeholders `%d`(ecimal integer), `%c`(haracter), `%f`(loat) and `%p`(ointer / address).

In Racket, we have seen `~a`(ny). The `~v`(alue) placeholder is useful when debugging as it shows extra type information (such as the quote for a `'symbol`).

In this course, we **only output "text"**, and so `printf` is the only output function we need.

Writing to **text files** directly is almost as straightforward as using `printf`. The `fprintf` function (**`file`** `printf`) has an additional parameter that is a file pointer (`FILE *`). The `fopen` function opens (creates) a file and return a pointer to that file.

```
#include <stdio.h>

int main(void) {
  FILE *file_ptr;
  file_ptr = fopen("hello.txt", "w");   // w for write
  fprintf(file_ptr, "Hello World!\n");
  fclose(file_ptr);
}
```

See CP:AMA 22.2 for more details.

Debugging output

Output can be very useful to help ***debug*** our programs.

We can use `printf` to output intermediate results and ensure that the program is behaving as expected. This is known as ***tracing*** a program. *Tracing* is especially useful when there is mutation.

A global variable can be used to turn tracing on or off.

```
const bool TRACE = true;  // set to false to turn off tracing
//..
if (TRACE) printf("The value of i is: %d\n",i);
```

In practice, tracing is commonly implemented with *macros* (`#define`) that can be turned on & off (CP:AMA 14).

C input: scanf

In C, the `scanf` function is the counterpart to the `printf` function.

```
scanf("%d", &i); // read in an integer, store in i
```

`scanf` requires a **pointer** to a variable to **store** the value read in from input.

Just as with `printf`, you use multiple placeholders to read in more than one value.

However, in this course **only read in one value per `scanf`**.

This will help you debug your code and facilitate our testing.

The **return value** of `scanf` is the number (count) of values *successfully read*.

The return value can also be the special constant value `EOF` to indicate that the **E**nd **O**f **F**ile (EOF) has been reached.

In `Seashell`, when you *run* (not *test*), a `Ctrl-D` (“Control D”) keyboard sequence sends an `EOF`.

In this course, a return value of one is “success”.

```
count = scanf("%d", &i); // read in an int, store in i

if (count != 1) {
  printf("Fail! I could not read in an integer!\n");
}
```

`scanf("%d", &i)` will **ignore whitespace** (spaces and newlines) and read in the next integer.

If the next non-whitespace input to be read is not a valid integer (e.g., a letter), it will stop reading and return zero.

When reading in a `char`, you may or may not want to ignore whitespace.

```
  // reads in next character (may be whitespace character)
  count = scanf("%c", &c);

  // reads in next character, ignoring whitespace
  count = scanf(" %c", &c);
```

The extra leading space in the second example indicates that whitespace should be ignored.

example 1: interactive C

```c
int main(void) {
  int num = 0;
  int i = 0;
  int sum = 0;

  printf("how many numbers should I sum?\n");
  if (scanf("%d", &num) != 1) {
    printf("bad input!\n");
    return 1;
  }
  for (int j=0; j < num; ++j) {
    printf("enter #%d:\n", j+1);
    if (scanf("%d", &i) != 1) {
      printf("bad input!\n");
      return 1;
    }
    sum += i;
  }
  printf("the sum of the %d numbers is: %d\n", num, sum);
}
```

example 2: interactive C

```
int main(void) {
  int num = 0;
  int i = 0;
  int sum = 0;

  printf("keep entering numbers, press Ctrl-D when done.\n");
  while (1) {
    printf("enter #%d:\n", num + 1);
    if (scanf("%d", &i) != 1) {
      break;
    }
    sum += i;
    ++num;
  }
  printf("the sum of the %d numbers is: %d\n", num, sum);
}
```

Tips for testing in C

Here are some additional tips for testing in C:

- check for “off by one” errors in loops
- consider the case that the initial loop condition is not met
- make sure every control flow path is tested
- consider large argument values (`INT_MAX` or `INT_MIN`)
- test for special argument values (`-1`, `0`, `1`, `NULL`)

Goals of this Section

At the end of this section, you should be able to:

- use the I/O terminology introduced
- use the input function `scanf` in C to make interactive programs
- use the `Seashell` testing environment effectively