CSCI-UA.0201

Computer Systems Organization

C Programming – I/O

Thomas Wies
wies@cs.nyu.edu
https://cs.nyu.edu/wies

I/O

- reading from:
  - standard input (usually the keyboard)
  - file
- writing to:
  - standard output (usually the screen)
  - file
- A library of functions is supplied to perform these operations.
- The I/O library functions are listed the header file <stdio.h>.

Writing to stdout

```
printf( );
```

- This function provides for formatted output to the screen. The syntax is:

```
    printf("format", var1, var2, ...);
```

- The "`format`" includes a listing of the data types of the variables to be output and, optionally, some text and control character(s).
- Example:

```
float f = 1.2;
int i = 42;
printf("The values are f:%f and i:%d\n", f, i);
```

Formatted Output with printf

Format Conversion Specifiers (This list is not exhaustive):

d -- displays a decimal (base 10) integer

l -- used with other specifiers to indicate a long

f -- displays a floating point value

x -- displays a number in hexadecimal format

c -- displays a single character

s -- displays a string of characters

Reading from stdin

`scanf();`

- This function provides for formatted input from the keyboard. The syntax is:

```
    scanf("format", &var1, &var2, ...);
```

- The "`format`" is a listing of the data types of the variables to be input and the & in front of each variable name tells the system WHERE to store the value that is input. It provides the address for the variable.
- Example:

```
 float a;  int b;
 scanf("%f%d", &a, &b);
```

Reading from stdin

- CAUTION: when reading strings, `scanf` can potentially write outside of the bounds of the allocated buffer.
- Example:

  ```
  char buf[10];
   scanf("%s", buf);
  ```

  This code may write outside of the bounds of `buf` if the user's input is larger than 9 characters.

Alternative to scanf: fgets

- It is usually better (and safer) to use the function fgets intead of scanf. Syntax:
  fgets(buf, max, file)
  - `buf` is a `char*` to the buffer where the input string will be stored.
  - max is the size of the buffer
  - `file` is a pointer to the file from which fgets reads (e.g. stdin)
  - `fgets` returns `NULL` if an error occurred or the end of the file has been reached. Otherwise it returns `buf`

Alternative to scanf: fgets

- Example:
```
char buf[10];
fgets(buf, 10, stdin);
```
- Reads up to the first `'\n'` on `stdin` or up to the 9th character if no `'\n'` is encountered up to that point.
- Read string is written into `buf` together with terminal `'\0'`.

Files

- In C, each file is simply a sequential stream of bytes.
- C imposes no structure on a file.
- Steps to deal with files
  - open a file
  - check that the open was successful
  - read/write to a file
  - close a file

First step

- Declaration:

```
FILE *fptr1, *fptr2 ;
```

Opening Files

- The statement:

```
fptr1 = fopen("filename", "r");
```

  would open the file filename for input (reading).

- Second argument indicates the *mode*
  - r: read
  - w: write
  - a: append
  - … there are some more

Testing for Successful Open

- If the file was not able to be opened, then the value returned by `fopen` is NULL.
- For example, let's assume that the file `myfile` does not exist. Then:

```
FILE *fptr1;
fptr1 = fopen("myfile", "r") ;
if (fptr1 == NULL) {
  printf("File 'myfile' did not open.\n");
}
```

Reading From Files

- In the following segment of C language code:

```
int a, b;
FILE *fptr1;
fptr1 = fopen("myfile", "r");
fscanf(fptr1, "%d%d", &a, &b);
```

the `fscanf` function would read values from the file "pointed" to by `fptr1` and assign those values to `a` and `b`.

End of File

- The end-of-file indicator informs the program when there are no more data (no more bytes) to be processed.
- There are a number of ways to test for the end-of-file condition. One is to use the `feof` function which returns a truth value:

```
fscanf(fptr1, "%d", &var);
if (feof(fptr1)) {
  printf("End-of-file encountered.\n”);
}
```

- Another (better) way of testing EOF:

```
while(fscanf(fp,"%d ", &current) == 1) {
  ...
}
```

- Or using `fgets`: `while (fgets(buf, max, fp)) { … }` Remember that fgets returns NULL (= 0) when EOF is reached.

Writing to Files

```
int a = 5, b = 30;
FILE  *fptr2 ;
fptr2 = fopen("filename", "w");
fprintf(fptr2, "%d %d\n", a, b);
```

The `fprintf` functions would write the values stored in `a` and `b` to the file "pointed" to by `fptr2`.

Closing Files

```
fclose(fptr1);
```

Once the files are open, they stay open until you close them or end the program (which will close all files.)