

# C: an introduction

IO

1

## Types of IO

- The C language provides no direct facilities for input and output. This functionality is provided by the standard library.  
In particular: `<stdio.h>`.
- Various ways of input and output.
  - input from the keyboard and output to the screen.
  - file IO.  
Files are a general concept. They include keyboard, screen, and other peripheral devices, as well as conventional files.
- Various formats of IO:
  - Formatted.
  - Character by character.
  - Line by line.
  - Binary.

3

# Data streams

- The input and output functions in C are built around the concept of a set of standard data streams
- The standard data streams or files are opened by the operating system and are available :
  - `stdin` : connected to the keyboard
  - `stdout` : connected to the screen
  - `stderr` : connected to the screen
  - Can use redirection (> and <) to change this (linux)

```
1 #include "stdio.h"
2 /*
3 demo_stderr.c
4 https://www.cs.bu.edu/teaching/c/file-io/intro/
5 */
6
7 int
8 main (void)
9 {
10
11 FILE *ifp;
12 FILE *ofp;
13 char *mode = "r";
14 char outputFilename[] = "out.list";
15
16 ifp = fopen("in.list", mode); // in.list does not exist
17 //ifp = fopen("temp3city.txt", mode); // temp3city.txt exist
18
19 if (ifp == NULL) {
20     fprintf(stderr, "Can't open input file in.list!\n");
21     return 1;
22 }
23
24 ofp = fopen(outputFilename, "w");
25
26 if (ofp == NULL) {
27     fprintf(stderr, "Can't open output file %s!\n",
28             outputFilename);
29     return 1;
30 }
31 }
```

test with both files

out.list will be created

```
frankvp@CRD-L-08004:~/io$ gcc demo_stderr.c -o demo_stderr
frankvp@CRD-L-08004:~/io$ ./demo_stderr
Can't open input file in.list!
frankvp@CRD-L-08004:~/io$ gcc demo_stderr.c -o demo_stderr
frankvp@CRD-L-08004:~/io$ ./demo_stderr
frankvp@CRD-L-08004:~/io$ ls -alt
total 64
-rwxrwxrwx 1 frankvp frankvp  0 Jan 26 11:27 out.list
-rwxrwxrwx 1 frankvp frankvp 16888 Jan 26 11:27 demo_stderr
drwxrwxrwx 1 frankvp frankvp 4096 Jan 26 11:27 .
-rwxrwxrwx 1 frankvp frankvp  574 Jan 26 11:27 demo_stderr.c
drwxrwxrwx 1 frankvp frankvp 4096 Jan 26 10:22 ..
```

```

1#include <stdio.h>
2#include <stdlib.h>
3#include <string.h>
4/*
5demo_interactive_input.c
6input until 'quit'
7*/
8int isQuit(char str[]);
9int main(void) {
10    for (;;) {
11        char str[80];
12        scanf("%s", str);
13        if (isQuit(str))
14            break;
15        printf("hello %s!\n", str);
16    }
17    return 0;
18}
19int isQuit(char str[]){
20    int ival;
21    ival = strcmp(str, "quit");
22    if (ival == 0)
23        return 1;
24    else
25        return 0;
26}

```

```

frankvp@CRD-L-08004:~/io$ gcc demo_interactive_input.c -o demo_interactive_input
frankvp@CRD-L-08004:~/io$ ./demo_interactive_input
help
hello help!
more
hello more!
information please
hello information!
hello please!
qk!m kmqdfk qsm d kf qmkf mqsdfmqsdq
hello qk!m!
hello kmqdfk!
hello qsm d!
hello kf qmkf!
hello mqsdfmqsdq!
quit
frankvp@CRD-L-08004:~/io$

```

## printf()

- **printf()** is a general purpose print function that sends its output to *standard output* (typically screen).
- General form:

```
printf("format string", item, item, ...)
```

```
int i = 10;
printf("The value of variable i is: %d", i);
```
- First argument is a *format string*.
  - Defines the layout of the printed text.
- **printf()** returns
  - On success: the number of characters printed.
  - On failure (output error): the symbolic constant **EOF**

```

1#include<stdio.h>
2
3/* test printf */
4
5void main()
6{
7    char name[30] = "I Am";
8
9    printf("\nEnter your name: \n");
10   printf("the sum of 5 and 6 = %d \n ", 5+6);
11   printf("\nHello %s \n",name);
12}

```

```

frankvp@CRD-L-08004:~/io$ gcc printf_1.c -o printf_1
frankvp@CRD-L-08004:~/io$ ./printf_1

```

```

Enter your name:
the sum of 5 and 6 = 11

```

```

Hello I Am
frankvp@CRD-L-08004:~/io$ █

```

## format specifiers

Type		Example
%d	print as integer	format_specifier_1.c
%xd	print as integer, at least x characters	
%u	unsigned integer	
%o	octal (unsigned integer base 8)	
%x	hexadecimal (unsigned integer base 16)	
%f	print as floating-point	
%xf	print as floating-point, at least x characters	
%.yf	print as floating-point, y characters after decimal	
%x.yf	print as floating-point, at least x characters, y characters after decimal .	
%e	float or double in exponential format	
%g	shortest form form of %e or %f	
%c	character ('A')	
%s	character string ("ABC" )	

```

1 #include "stdio.h"
2 /* test different format specifiers
3 format_specifier_1.c
4 http://www-control.eng.cam.ac.uk/~pcr20/C_Manual/chap03.html
5 */
6
7 int main()
8 {
9     printf("/%d/\n", 336);
10    printf("/%2d/\n", 336);
11    printf("/%10d/\n", 336);
12    printf("/%-10d/\n", 336);
13
14    printf("/%f/\n", 1234.56);
15    printf("/%e/\n", 1234.56);
16    printf("/%4.f/\n", 1234.56);
17    printf("/%3.1f/\n", 1234.56);
18    printf("/%10.3f/\n", 1234.56);
19    printf("/%10.3e/\n", 1234.56);
20    printf("/%g/\n", 1234.56);
21    printf("/%g/\n", 1234.56000008);
22    printf("/%g/\n", 12340000.56);
23
24    return 0;
25 }

```

```

frankvp@CRD-L-08004:.../io$ gcc format_specifier_1.c -o format_specifier_1
frankvp@CRD-L-08004:.../io$ ./format_specifier_1
/336/
/336/
/      336/
/336      /
/1234.560000/
/1.234560e+03/
/1235/
/1234.6/
/  1234.560/
/ 1.235e+03/
/1234.56/
/1234.56/
/1.234e+07/
frankvp@CRD-L-08004:.../io$

```

KU LEUVEN

11

## formatted input: scanf

- `scanf` is the input analog of `printf`:
  - function reads characters from standard input
  - interpreting them as specified by the format specifier control
  - storing them in variables `arg1`, `arg2`, `arg3`, ...
- Most significant difference is that `scanf()` arguments must be pointers.

```

double fval;
scanf("%lf", fval); /* wrong */
scanf("%lf", &fval); /* correct */

```

KU LEUVEN

12

```

1 #include <stdio.h>
2 /*
3 demo_scanf.c
4 enter data separated by blancs */
5
6 int main()
7 {
8     int code;
9     int age;
10    char codex;
11    float weight;
12
13    printf("Enter age, codex , weight \n");
14
15    code=scanf("%d %c %f", &age, &codex, &weight);
16
17    printf("number of arguments read = %d \n", code);
18    printf(" age is %d, codex is %c, weight is %5.1f \n", age,
19    codex, weight);
20
21    return 0;
22 }

```

```

frankvp@CRD-L-08004:~/io$ gcc demo_scanf.c -o demo_scanf
frankvp@CRD-L-08004:~/io$ ./demo_scanf
Enter age, codex , weight
28 x 78
number of arguments read = 3
age is 28, codex is x, weight is 78.0
frankvp@CRD-L-08004:~/io$ ./demo_scanf
Enter age, codex , weight
1, 2, 3
number of arguments read = 3
age is 1, codex is , weight is 2.0
frankvp@CRD-L-08004:~/io$ ./demo_scanf
Enter age, codex , weight
input
number of arguments read = 0
age is 22002, codex is U, weight is -0.0
frankvp@CRD-L-08004:~/io$

```

## formatted input: scanf

- function `scanf` ends when:
  - end of format string is reached
  - format specification does not match the input
- result is
  - number of arguments successfully read
  - EOF at the end of the file
- conversion specification 1 field is read
  - 1 field is a sequence of non-white characters
  - Separator: blanc, tab, newline

## Warnings about `scanf()`

- Note, the above string (`%s`) input is not robust.
  - String read until first white-space character.
  - User can type in over-long sequence and overflow buffer.
- Include a width field.

```
char s1[10], s2[10], s3[10];
scanf("%9s %9s %9s", s1, s2, s3);
```

- `scanf()` is a good choice if the input format is exactly known, but not if the format may vary. Better to use:

```
fgets(buf, sizeof(buf), stdin);
sscanf(buf, "%lf", &dval);
```

```
1 /*
2 input_fgets.c
3 https://csijh.gitlab.io/COMS10008/lectures/io/
4 Echos back what you type. Use CTRL/D (or CTRL/C) to end. */
5
6 #include <stdio.h>
7 #include <stdbool.h>
8
9 // Prompt the user and read in one line
10 // (saves repeating three lines twice in main)
11 void get(int max, char line[max]) {
12     printf("Type: ");
13     fgets(line, max, stdin);
14 }
15
16 int main() {
17     const int max = 100;
18     char line[max];
19     get(max, line);
20     while (!feof(stdin)) {
21         printf("Line: %s", line);
22         get(max, line);
23     }
24 }
```

```
frankvp@CRD-L-08004:~/io$ gcc input_fgets.c -o input_fgets
frankvp@CRD-L-08004:~/io$ ./input_fgets
Type: help on this topic
Line: help on this topic
Type: more info
Line: more info
Type: stop
Line: stop
Type: exit
Line: exit
Type: frankvp@CRD-L-08004:~/io$
```

# String Formatting

- `sprintf()` and `sscanf()` are identical to `printf()` and `scanf()`
- *except* that they take IO from a string and not `stdout` or `stdin`.
- General forms:

```
int sprintf(char *buf, const char *format, ...);  
int sscanf(char *buf, const char *format, ...);
```

```
1#include <stdio.h>  
2#include <math.h>  
3/* http://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm  
4sprintf_1.c  
5compile gcc sprintf_1.c -lm  
6*/  
7  
8int main()  
9{  
10    char str[80];  
11  
12    sprintf(str, "Value of Pi = %f", M_PI);  
13    puts(str);  
14  
15    return(0);  
16}
```

```
frankvp@CRD-L-08004:~/io$ gcc sprintf_1.c -o sprintf_1  
frankvp@CRD-L-08004:~/io$ ./sprintf_1  
Value of Pi = 3.141593  
frankvp@CRD-L-08004:~/io$
```



```

1#include <stdio.h>
2#include <string.h>
3/*
4sscanf_1.c
5http://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm */
6
7int main()
8{
9    int day, year;
10   char weekday[20], month[20], dtm[100];
11
12   strcpy( dtm, "Saturday March 25 1989" );
13   printf("%s\n", dtm);
14
15   sscanf( dtm, "%s %s %d %d", weekday, month, &day, &year );
16   printf("%s\n", dtm);
17
18   printf("%s %d, %d = %s\n", month, day, year, weekday );
19
20   return(0);
21}

```

```

frankvp@CRD-L-08004:~/io$ gcc sscanf_1.c -o sscanf_1
frankvp@CRD-L-08004:~/io$ ./sscanf_1
Saturday March 25 1989
Saturday March 25 1989
March 25, 1989 = Saturday
frankvp@CRD-L-08004:~/io$

```

## File IO

- The C language is closely tied to the UNIX operating system. They were initially developed in parallel and UNIX was implemented in C.
- Much of the C standard library is modelled on UNIX facilities, in particular the UNIX IO model, which treats everything as files.
- Communication with peripheral devices – keyboard, screen, etc – performed by reading and writing to files.
- Provides a single common interface for all IO operations.

## What do you want to do?

- read chars from file: `fopen`, `fgetc`, `feof`, `fclose`
- read bytes from file: `fopen`, `fgetc`, `feof`, `fclose`
- read lines from file: `fopen`, `fgets`, `feof`, `fclose`
- write chars to file: `fopen`, `fputc`, `fclose`
- write bytes to file: `fopen`, `fputc`, `fclose`
- write lines to file: `fopen`, `fprintf`, `fclose`

## `fopen()`

- A file is referred to by a file-pointer. This is a pointer to a structure **typedef** called `FILE`.
- The file open function (`fopen`) serves two purposes:
  - It makes the connection between the physical file and the stream.
  - It creates “a program file structure to store the information” C needs to process the file.
- Syntax:  
`fopen("filename", "mode");`
  - Two arguments:
    1. The file name. eg, `myfile.txt`
    2. The file mode. `"r"`, `"w"`, `"a"`
  - Return value: Pointer to file if successful. NULL if unsuccessful.
  - Always check return value for NULL!

## File open modes

r	Open text file for reading	<ul style="list-style-type: none"><li>• If file exists, marker is positioned at beginning</li><li>• If file does not exist, an error is generated</li></ul>
w	Open text file for writing	<ul style="list-style-type: none"><li>• If file exists, the file is erased (overwritten)</li><li>• If file does not exist, it is created</li></ul>
a	Open text file for appending	<ul style="list-style-type: none"><li>• If file exists, marker is positioned at end</li><li>• If file does not exist, it is created</li></ul>
rb	Open binary file for reading	
wb	Open binary file for writing	
ab	Open binary file for appending	
+	File is to be opened for reading and writing	

## `fclose()`

- To close a file, pass the file pointer to `fclose()`.
- General form:  

```
int fclose(FILE *fp);
```
- `fclose()` breaks the connection with the file and frees the file pointer.
- Good practice to free file pointers when a file is no longer needed as most OSs have a limit on the number of files a program may have open at any given time.
- Note, `fclose()` is called automatically for each open file when the program terminates.

```

1 #include "stdio.h"
2 /*
3 fopen_fclose.c
4 http://www.fcet.staffs.ac.uk/rgh1/ */
5
6 int
7 main (void)
8 {
9
10 int a, b, c;
11 char filename[21]; // string file name
12 FILE *out_file; // file pointer for output
13
14 printf ("\ntype name of output file: "); // prompt on screen
15 scanf("%s", filename); // input from keyboard
16
17 out_file = fopen (filename, "w"); // open file for output
18 if (out_file == NULL) {
19     printf ("\ncannot open: %s", filename);
20     return 1; // abnormal program exit
21 }
22
23 printf ("\ntype 2 integers"); // prompt
24 scanf ("%d %d", &a, &b); // from keyboard
25 c = a + b;
26
27 fprintf (out_file, "%d\n", c);
28
29 // output to file
30 fclose (out_file);
31
32 return 0; // normal program exit
33 }

```

```

frankvp@CRD-L-08004:~/io$ gcc fopen_fclose.c -o fopen_fclose
frankvp@CRD-L-08004:~/io$ ./fopen_fclose

```

```

type name of output file: myfile
type 2 integers25 89
frankvp@CRD-L-08004:~/io$ cat myfile
114
frankvp@CRD-L-08004:~/io$ █

```

KU LEUVEN

26

## Sequential File Operations

- Once a file is open, operations on the file (reading and writing) usually work through the file sequentially – from the beginning to the end.
- *File: read\_temp3city.c*

KU LEUVEN

27

```

1#include <stdio.h>
2// read_temp3city.c
3int main()
4{
5    int numc1[31], numc2[31], numc3[31];
6    int maxt[3] = {-999 -999 -999};
7    int dayt[3] = {-999 -999 -999};
8    int count;
9    FILE *fptr;
10   fptr = fopen("temp3city.txt", "r");
11
12   if(fptr == NULL){
13       printf("Error!");
14       return 1;
15   }
16   for(count = 0; count <= 30; ++count) {
17       fscanf(fptr, "%d %d %d", &numc1[count], &numc2[count], &numc3[count]);
18       fclose(fptr);
19   }
20   // search for the maximum at each city
21   for (count = 0; count <= 30; ++count){
22       if (numc1[count] > maxt[0]) {
23           maxt[0] = numc1[count];
24           dayt[0] = count + 1;
25       }
26       printf("Maximum temperature at day %d and it's value is %d.\n", dayt[0], maxt[0]);
27
28       for (count = 0; count <= 30; ++count){
29           if (numc2[count] > maxt[1]) {
30               maxt[1] = numc2[count];
31               dayt[1] = count + 1;
32           }
33           printf("Maximum temperature at day %d and it's value is %d.\n", dayt[1], maxt[1]);
34
35           for (count = 0; count <= 30; ++count) {
36               if (numc3[count] > maxt[2]) {
37                   maxt[2] = numc3[count];
38                   dayt[2] = count + 1;
39               }
40               printf("Maximum temperature at day %d and it's value is %d.\n", dayt[2], maxt[2]);
41           }
42       }
43   }
44   return 0;
45 }

```

```

frankvp@CRD-L-08004:~/io$ ./read_temp3city
Maximum temperature at day 9 and it's value is 19.
Maximum temperature at day 23 and it's value is 12.
Maximum temperature at day 30 and it's value is 24.
frankvp@CRD-L-08004:~/io$ cat temp3city.txt
12 8 18
15 9 22
12 5 19
14 8 23
12 6 22
11 9 19
15 9 15
8 10 20
19 7 18
12 7 18
14 10 19
11 8 17
9 7 23
8 8 19
15 8 18
8 9 20
10 7 17
12 7 22
9 8 19
12 8 21
12 8 20
10 9 17
13 12 18
9 10 20
10 6 22
14 7 21
12 5 22
13 7 18
15 10 23
13 11 24
12 12 22

```

28

## Formatted IO

```

int fprintf(FILE *fp, const char *format, ...);
int fscanf(FILE *fp, const char *format, ...);

```

- These functions are generalisations of `printf()` and `scanf()`, respectively.
- In fact, `printf()` and `scanf()` are equivalent to

```

fprintf(stdout, format, arg1, arg2, ...);
fscanf(stdin, format, arg1, arg2, ...);

```

29

```

1 /*
2 fprintf_fscanf.c
3 http://gribblelab.org/CBootcamp/10_Input_and_Output.html
4 */
5 #include <stdio.h>
6 int main() {
7     FILE *fp;
8     double tmpC[11] = {-10.0, -8.0, -6.0, -4.0, -2.0, 0.0, 2.0, 4.0, 6.0, 8.0, 10.0};
9     double tmpF;
10    double temp3c[50][3];
11    double tmax = -100;
12    int i;
13    // writing file
14    fp = fopen("outfileTemp.txt", "w");
15    if (fp == NULL) {
16        printf("sorry can't open outfile.txt\n");
17        return 1;
18    }
19    // print a table header
20    fprintf(fp, "%10s %10s\n", "Celsius", "Fahrenheit");
21    for (i=0; i<11; i++) {
22        tmpF = ((tmpC[i] * (9.0/5.0)) + 32.0);
23        fprintf(fp, "%10.2f %10.2f\n", tmpC[i], tmpF);
24    }
25    // reading file
26    fp = fopen("temp3city.txt", "r");
27    if (fp == NULL) {
28        printf("sorry can't open temp3city.txt\n");
29        return 1;
30    }
31    else {
32        for (i=0; i<31; i++) {
33            fscanf(fp, "%1f %1f %1f\n", &temp3c[i][1], &temp3c[i][2], &temp3c[i][3]);
34            if (tmax < temp3c[i][1]) {
35                tmax = temp3c[i][1];
36            }
37            if (tmax < temp3c[i][2]) {
38                tmax = temp3c[i][2];
39            }
40            if (tmax < temp3c[i][3]) {
41                tmax = temp3c[i][3];
42            }
43        }
44        fclose(fp);
45        for (i=0; i<31; i++) {
46            printf("%d %5.2f %5.2f %5.2f\n", i, temp3c[i][1], temp3c[i][2], temp3c[i][3]);
47        }
48        printf("\n\n maximum temperature = %5.2f\n", tmax);
49    }
50    return 0;
51 }

```

```

frankvp@CRD-L-08004:~/io$ cat outfileTemp.txt
Celsius Fahrenheit
-10.00    14.00
-8.00     17.60
-6.00     21.20
-4.00     24.80
-2.00     28.40
0.00      32.00
2.00      35.60
4.00      39.20
6.00      42.80
8.00      46.40
10.00     50.00

frankvp@CRD-L-08004:~/io$ gcc fprintf_fscanf.c -o fprintf_fscanf
frankvp@CRD-L-08004:~/io$ ./fprintf_fscanf
0 12.00 8.00 18.00
1 15.00 9.00 22.00
2 12.00 5.00 19.00
3 14.00 8.00 23.00
4 12.00 6.00 22.00
5 11.00 9.00 19.00
6 15.00 9.00 15.00
7 8.00 10.00 20.00
8 19.00 7.00 18.00
9 12.00 7.00 18.00
10 14.00 10.00 19.00
11 11.00 8.00 17.00
12 9.00 7.00 23.00
13 8.00 8.00 19.00
14 15.00 8.00 18.00
15 8.00 9.00 20.00
16 10.00 7.00 17.00
17 12.00 7.00 22.00
18 9.00 8.00 19.00
19 12.00 8.00 21.00
20 12.00 8.00 20.00
21 10.00 9.00 17.00
22 13.00 12.00 18.00
23 9.00 10.00 20.00
24 10.00 6.00 22.00
25 14.00 7.00 21.00
26 12.00 5.00 22.00
27 13.00 7.00 18.00
28 15.00 10.00 23.00
29 13.00 11.00 24.00
30 12.00 12.00 22.00

maximum temperature = 24.00

```

30

# Character Input

- Character input functions:
 

```
int fgetc(FILE *fp);
int getc(FILE *fp);
int getchar(void);
```
- `getchar()` is equivalent to `getc(stdin)`.
- `getc()` and `fgetc()` are essentially identical.
- Return values:
  - On success: the next character in the input stream.
  - On error: `EOF`.
  - On end-of-file: `EOF`.
- File: `fgetcchar.c`

## Character Output

- Character output functions:

```
int fputc(int c, FILE *fp);
int putc(int c, FILE *fp);
int putchar(int c);
```

- `putchar(c)` is equivalent to `putc(c, stdout)`.
- `putc()` and `fputc()` are essentially identical, implementation is different. (`fputc` is preferred - <https://stackoverflow.com/questions/14008907/fputc-vs-putc-in-c>)
- Return values:
  - On success: the character that was written.
  - On error: `EOF`.
- File: `fputcchar.c`

## Line Input

- Read a line from a file:

```
char *fgets(char *buf, int max, FILE *fp);
```

- Returns after one of the following:
  - Reads (at most) `max-1` characters from the file.
  - Reads a `\n` character.
  - Reaches end-of-file.
  - Encounters an error.
- Return values:
  - On success: pointer to `buf`. Note, `fgets()` automatically appends a `\0` to the end of the string.
  - On end-of-file: `NULL`.
  - On error: `NULL`.

```

1 /*
2 demo_fgets.c
3 based on www.cs.colstate.edu/~cs156
4 */
5
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 int main(){
11     char first[100], last[100];
12
13     printf("Enter your first name: ");
14     fgets(first, sizeof(first), stdin);
15
16     printf("Enter your last name: ");
17     fgets(last, sizeof(last), stdin);
18
19     printf("\n Your name is: %s %s", first, last);
20
21     return 0;
22 }
23

```

```

frankvp@CRD-L-08004:~/io$ gcc demo_fgets.c -o demo_fgets
frankvp@CRD-L-08004:~/io$ ./demo_fgets
Enter your first name: frank
Enter your last name: van puyvelde

Your name is: frank
van puyvelde
frankvp@CRD-L-08004:~/io$

```

## Line Output

- Character strings may be written to file using  

```
int fputs(const char *str, FILE *fp);
```
- Not actually line output. It does not automatically append a `\n` and consecutive calls may print strings on the same line.
- Return values:
  - On success: zero.
  - On error: `EOF`.
- *File: demo\_fputs.c*



# Binary IO

- When reading and writing binary files, may deal with objects directly without first converting them to character strings.
- Direct binary IO provided by

```
size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);  
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);
```

- Can pass objects of any type. For example,  

```
struct Astruct mystruct[10];  
fwrite(&mystruct, sizeof(Astruct), 10, fp);
```
- *File: binary\_write.c*
- *File: binary\_read.c*