KU LEUVEN

C: an introduction

IO

Types of IO

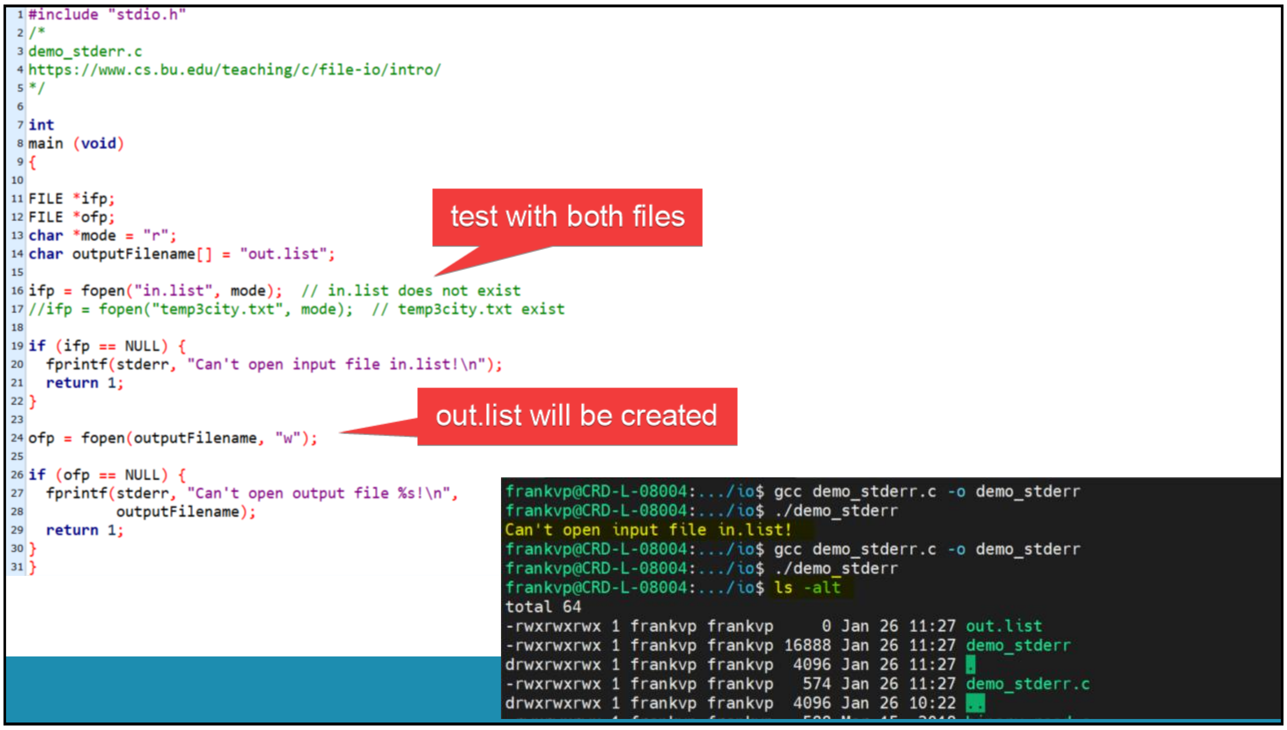
- The C language provides no direct facilities for input and output. This functionality is provided by the standard library.
  In particular: **`<stdio.h>`**.
- Various ways of input and output.
  - input from the keyboard and output to the screen.
  - file IO.
    Files are a general concept. They include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

KU LEUVEN

Data streams

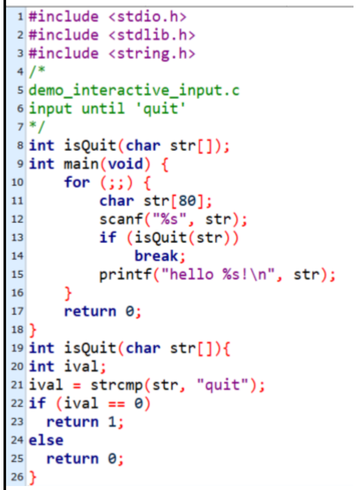
- The input and output functions in C are built around the concept of a set of standard data streams
- The standard data streams or files are opened by the operating system and are available :
  - `stdin` : connected to the keyboard
  - `stdout` : connected to the screen
  - `stderr` : connected to the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

KU LEUVEN

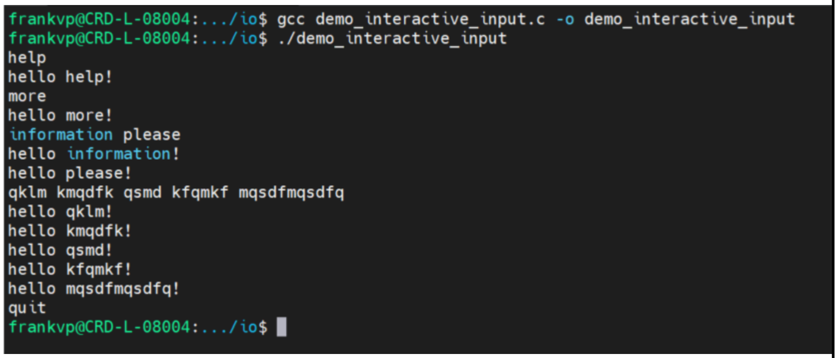


```c
#include "stdio.h"
/*
demo_stderr.c
https://www.cs.bu.edu/teaching/c/file-io/intro/
*/

int
main (void)
{

FILE *ifp;
FILE *ofp;
char *mode = "r";
char outputFilename[] = "out.list";

ifp = fopen("in.list", mode);  // in.list does not exist
//ifp = fopen("temp3city.txt", mode);  // temp3city.txt exist

if (ifp == NULL) {
  fprintf(stderr, "Can't open input file in.list!\n");
  return 1;
}

ofp = fopen(outputFilename, "w");

if (ofp == NULL) {
  fprintf(stderr, "Can't open output file %s!\n",
          outputFilename);
  return 1;
}
}
```

```
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 ..
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/*
demo_interactive_input.c
input until 'quit'
*/
int isQuit(char str[]);
int main(void) {
    for (;;) {
        char str[80];
        scanf("%s", str);
        if (isQuit(str))
            break;
        printf("hello %s!\n", str);
    }
    return 0;
}
int isQuit(char str[]){
int ival;
ival = strcmp(str, "quit");
if (ival == 0)
  return 1;
else
  return 0;
}
```



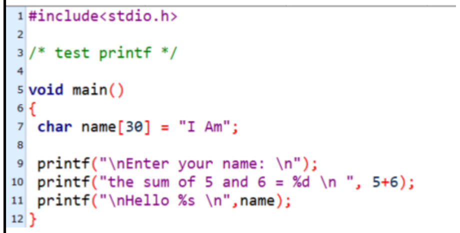
```
frankvp@CRD-L-08004:.../io$ gcc demo_interactive_input.c -o demo_interactive_input
frankvp@CRD-L-08004:.../io$ ./demo_interactive_input
help
hello help!
more
hello more!
information please
hello information!
hello please!
qklm kmqdfk qsmd kfqmkf mqsdfmqsdfq
hello qklm!
hello kmqdfk!
hello qsmd!
hello kfqmkf!
hello mqsdfmqsdfq!
quit
frankvp@CRD-L-08004:.../io$
```

printf()

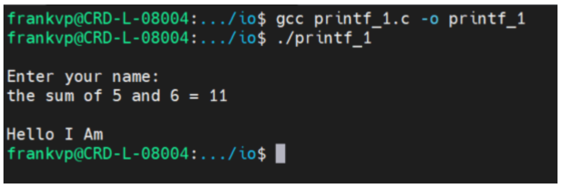
- **printf()** is a general purpose print function that sends its output to *standard output* (typically screen).
- General form:

```
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**



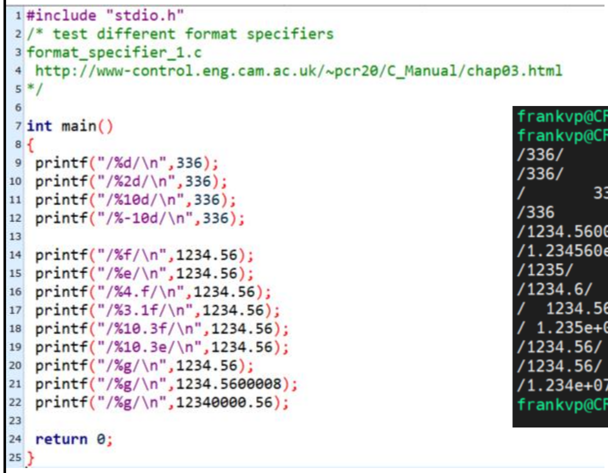
```c
#include<stdio.h>

/* test printf */

void main()
{
 char name[30] = "I Am";

 printf("\nEnter your name: \n");
 printf("the sum of 5 and 6 = %d \n ", 5+6);
 printf("\nHello %s \n",name);
}
```



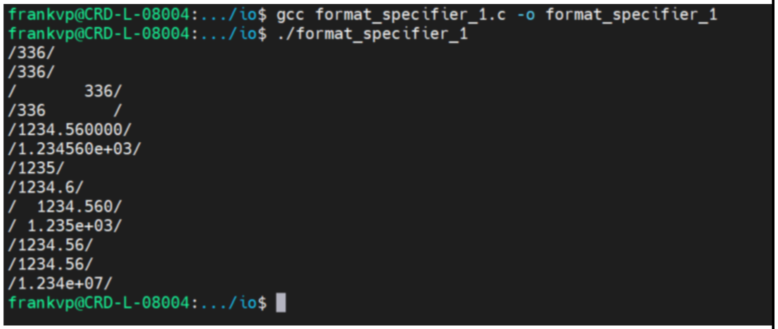
```
frankvp@CRD-L-08004:.../io$ gcc printf_1.c -o printf_1
frankvp@CRD-L-08004:.../io$ ./printf_1

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")	



```c
#include "stdio.h"
/* test different format specifiers
format_specifier_1.c
 http://www-control.eng.cam.ac.uk/~pcr20/C_Manual/chap03.html
*/

int main()
{
 printf("/%d/\n",336);
 printf("/%2d/\n",336);
 printf("/%10d/\n",336);
 printf("/%-10d/\n",336);

 printf("/%f/\n",1234.56);
 printf("/%e/\n",1234.56);
 printf("/%4.f/\n",1234.56);
 printf("/%3.1f/\n",1234.56);
 printf("/%10.3f/\n",1234.56);
 printf("/%10.3e/\n",1234.56);
 printf("/%g/\n",1234.56);
 printf("/%g/\n",1234.5600008);
 printf("/%g/\n",12340000.56);

 return 0;
}
```



```
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$
```

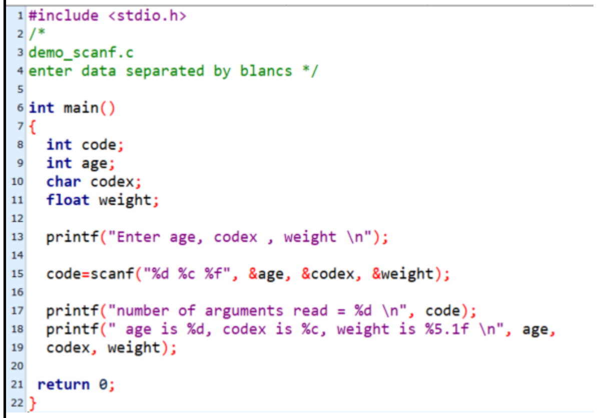
formatted input: scanf

- `scanf` is the input analog of printf:

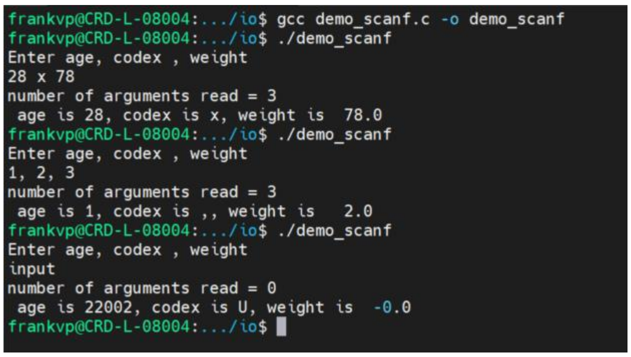
  `scanf(control, arg1, arg2, arg3, ...);`

  - 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.

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



```c
#include <stdio.h>
/*
demo_scanf.c
enter data separated by blancs */

int main()
{
  int code;
  int age;
  char codex;
  float weight;

  printf("Enter age, codex , weight \n");

  code=scanf("%d %c %f", &age, &codex, &weight);

  printf("number of arguments read = %d \n", code);
  printf(" age is %d, codex is %c, weight is %5.1f \n", age,
  codex, weight);

 return 0;
}
```



```
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()`

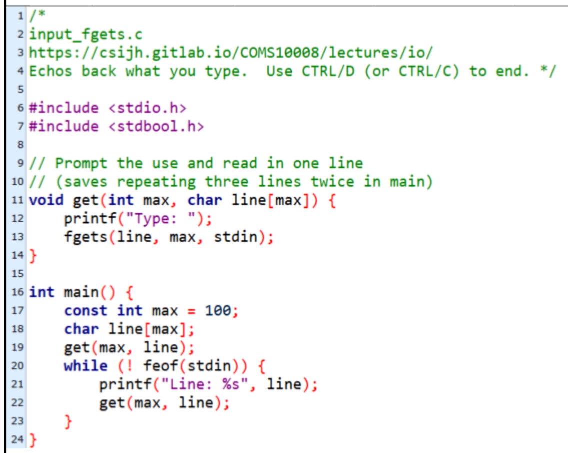
- Note, the above string (**`%s`**) input is not robust.
  - String read until first white-space character.
  - User can type in over-long sequence and overflow buffer.
- Include a width field.

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

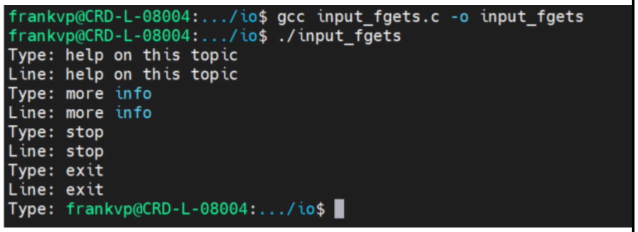
- `scanf()` is a good choice if the input format is exactly known, but not if the format may vary. Better to use:

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

KU LEUVEN



```c
/*
input_fgets.c
https://csijh.gitlab.io/COMS10008/lectures/io/
Echos back what you type.  Use CTRL/D (or CTRL/C) to end. */

#include <stdio.h>
#include <stdbool.h>

// Prompt the use and read in one line
// (saves repeating three lines twice in main)
void get(int max, char line[max]) {
    printf("Type: ");
    fgets(line, max, stdin);
}

int main() {
    const int max = 100;
    char line[max];
    get(max, line);
    while (! feof(stdin)) {
        printf("Line: %s", line);
        get(max, line);
    }
}
```



```
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$
```

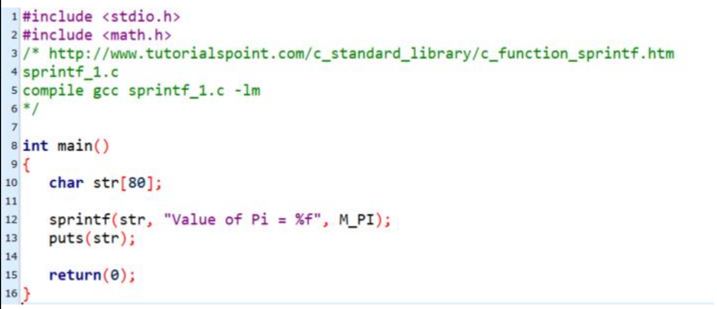
KU LEUVEN

String Formatting

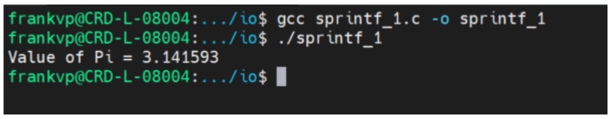
- `sprintf()` and `sscanf()` are identical to `printf()` and `scanf()`
- *except* that they take IO from a string and not `stdout` or `stdin`.
- General forms:

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

KU LEUVEN

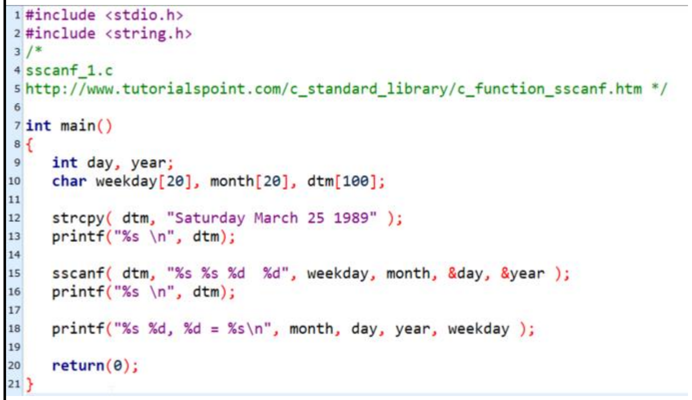


```c
#include <stdio.h>
#include <math.h>
/* http://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm
sprintf_1.c
compile gcc sprintf_1.c -lm
*/

int main()
{
   char str[80];

   sprintf(str, "Value of Pi = %f", M_PI);
   puts(str);

   return(0);
}
```

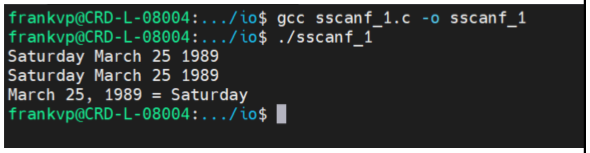


```
frankvp@CRD-L-08004:.../io$ gcc sprintf_1.c -o sprintf_1
frankvp@CRD-L-08004:.../io$ ./sprintf_1
Value of Pi = 3.141593
frankvp@CRD-L-08004:.../io$
```

KU LEUVEN



```c
#include <stdio.h>
#include <string.h>
/*
sscanf_1.c
http://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm */

int main()
{
   int day, year;
   char weekday[20], month[20], dtm[100];

   strcpy( dtm, "Saturday March 25 1989" );
   printf("%s \n", dtm);

   sscanf( dtm, "%s %s %d  %d", weekday, month, &day, &year );
   printf("%s \n", dtm);

   printf("%s %d, %d = %s\n", month, day, year, weekday );

   return(0);
}
```



```
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$ 
```

KU LEUVEN

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.

KU LEUVEN

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`

KU LEUVEN

`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!

KU LEUVEN

File open modes

r	Open text file for reading	• If file exists, marker is positioned at beginning • If file does not exist, an error is generated
w	Open text file for writing	• If file exists, the file is erased (overwritten) • If file does not exist, it is created
a	Open text file for appending	• If file exists, marker is positioned at end • If file does not exist, it is created
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	

KU LEUVEN

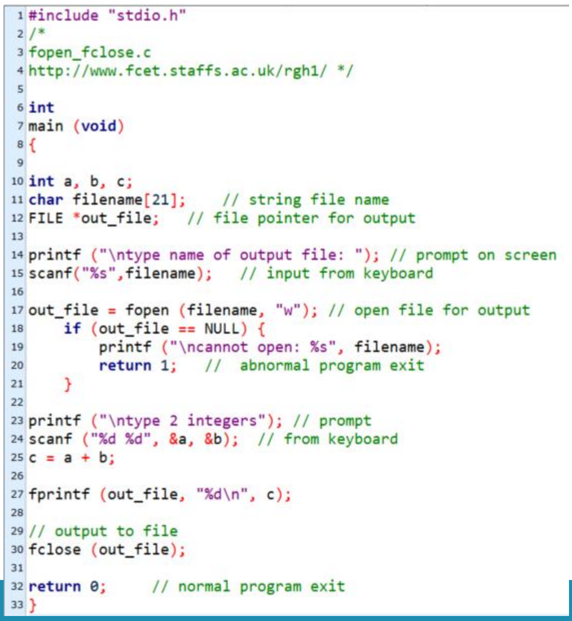
`fclose()`

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

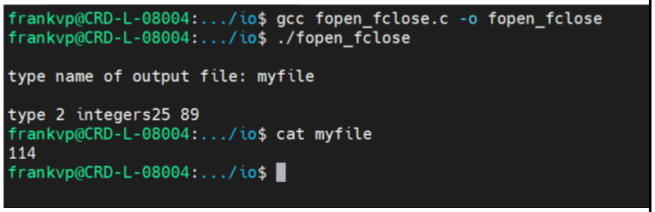
```
int fclose(FILE *fp);
```

- `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.

KU LEUVEN



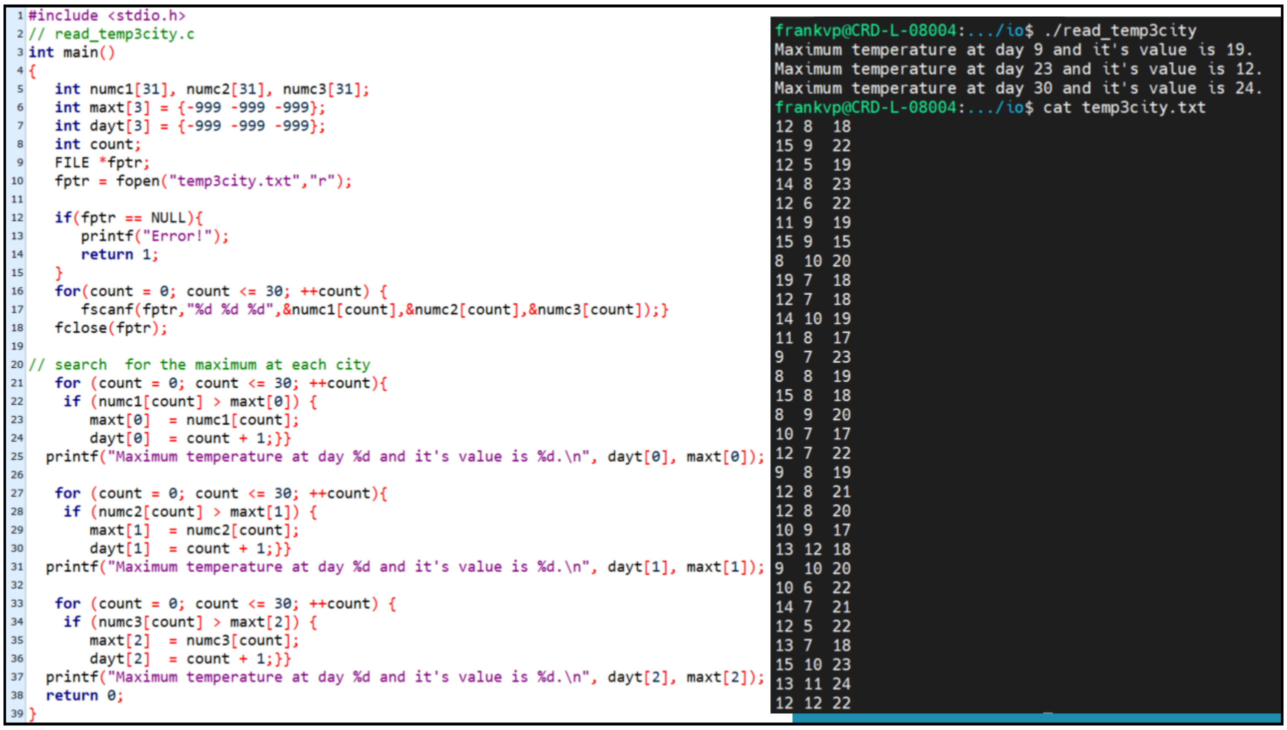
```c
#include "stdio.h"
/*
fopen_fclose.c
http://www.fcet.staffs.ac.uk/rgh1/ */

int
main (void)
{

int a, b, c;
char filename[21];     // string file name
FILE *out_file;    // file pointer for output

printf ("\ntype name of output file: "); // prompt on screen
scanf("%s",filename);    // input from keyboard

out_file = fopen (filename, "w"); // open file for output
    if (out_file == NULL) {
        printf ("\ncannot open: %s", filename);
        return 1;    //  abnormal program exit
    }

printf ("\ntype 2 integers"); // prompt
scanf ("%d %d", &a, &b);  // from keyboard
c = a + b;

fprintf (out_file, "%d\n", c);

// output to file
fclose (out_file);

return 0;      // normal program exit
}
```



```
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$ 
```

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*



```c
#include <stdio.h>
// read_temp3city.c
int main()
{
   int numc1[31], numc2[31], numc3[31];
   int maxt[3] = {-999 -999 -999};
   int dayt[3] = {-999 -999 -999};
   int count;
   FILE *fptr;
   fptr = fopen("temp3city.txt","r");

   if(fptr == NULL){
      printf("Error!");
      return 1;
   }
   for(count = 0; count <= 30; ++count) {
      fscanf(fptr,"%d %d %d",&numc1[count],&numc2[count],&numc3[count]);}
   fclose(fptr);

// search  for the maximum at each city
   for (count = 0; count <= 30; ++count){
    if (numc1[count] > maxt[0]) {
      maxt[0]  = numc1[count];
      dayt[0]  = count + 1;}}
  printf("Maximum temperature at day %d and it's value is %d.\n", dayt[0], maxt[0]);

   for (count = 0; count <= 30; ++count){
    if (numc2[count] > maxt[1]) {
      maxt[1]  = numc2[count];
      dayt[1]  = count + 1;}}
  printf("Maximum temperature at day %d and it's value is %d.\n", dayt[1], maxt[1]);

   for (count = 0; count <= 30; ++count) {
    if (numc3[count] > maxt[2]) {
      maxt[2]  = numc3[count];
      dayt[2]  = count + 1;}}
  printf("Maximum temperature at day %d and it's value is %d.\n", dayt[2], maxt[2]);
  return 0;
}
```

```
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
```

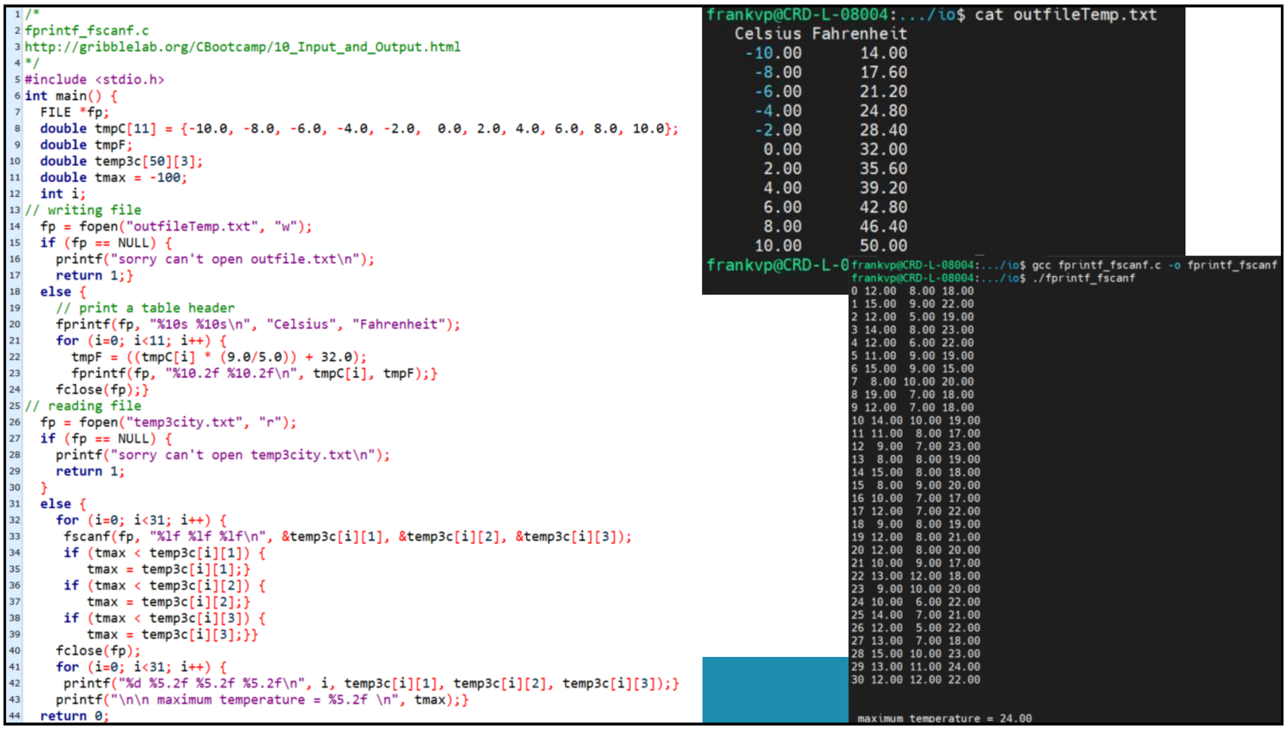
Formatted IO

```
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, ...);
```

KU LEUVEN



```c
/*
fprintf_fscanf.c
http://gribblelab.org/CBootcamp/10_Input_and_Output.html
*/
#include <stdio.h>
int main() {
  FILE *fp;
  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};
  double tmpF;
  double temp3c[50][3];
  double tmax = -100;
  int i;
// writing file
  fp = fopen("outfileTemp.txt", "w");
  if (fp == NULL) {
    printf("sorry can't open outfile.txt\n");
    return 1;}
  else {
    // print a table header
    fprintf(fp, "%10s %10s\n", "Celsius", "Fahrenheit");
    for (i=0; i<11; i++) {
      tmpF = ((tmpC[i] * (9.0/5.0)) + 32.0);
      fprintf(fp, "%10.2f %10.2f\n", tmpC[i], tmpF);}
    fclose(fp);}
// reading file
  fp = fopen("temp3city.txt", "r");
  if (fp == NULL) {
    printf("sorry can't open temp3city.txt\n");
    return 1;
  }
  else {
    for (i=0; i<31; i++) {
     fscanf(fp, "%lf %lf %lf\n", &temp3c[i][1], &temp3c[i][2], &temp3c[i][3]);
     if (tmax < temp3c[i][1]) {
        tmax = temp3c[i][1];}
     if (tmax < temp3c[i][2]) {
        tmax = temp3c[i][2];}
     if (tmax < temp3c[i][3]) {
        tmax = temp3c[i][3];}}
    fclose(fp);
    for (i=0; i<31; i++) {
     printf("%d %5.2f %5.2f %5.2f\n", i, temp3c[i][1], temp3c[i][2], temp3c[i][3]);}
    printf("\n\n maximum temperature = %5.2f \n", tmax);}
  return 0;
```

```
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-0
```

```
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
```

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*

KU LEUVEN

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*

KU LEUVEN

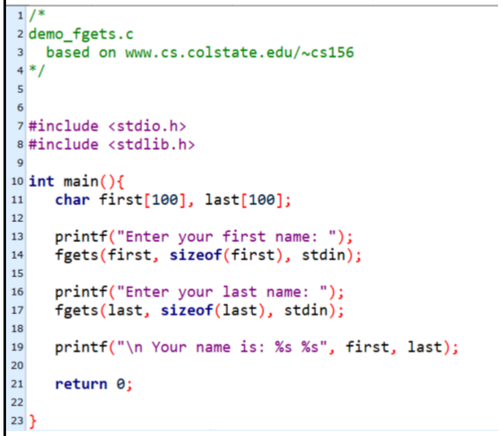
Line Input

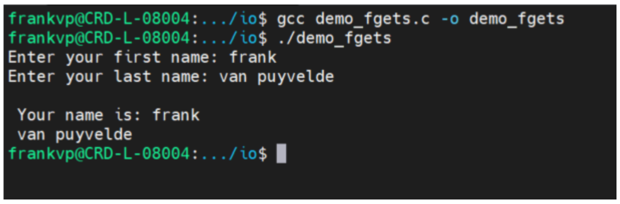
- Read a line from a file:

```
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`**.

KU LEUVEN


```c
/*
demo_fgets.c
  based on www.cs.colstate.edu/~cs156
*/


#include <stdio.h>
#include <stdlib.h>

int main(){
   char first[100], last[100];

   printf("Enter your first name: ");
   fgets(first, sizeof(first), stdin);

   printf("Enter your last name: ");
   fgets(last, sizeof(last), stdin);

   printf("\n Your name is: %s %s", first, last);

   return 0;

}
```


```
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*

KU LEUVEN