Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

I2204 - Imperative Programming

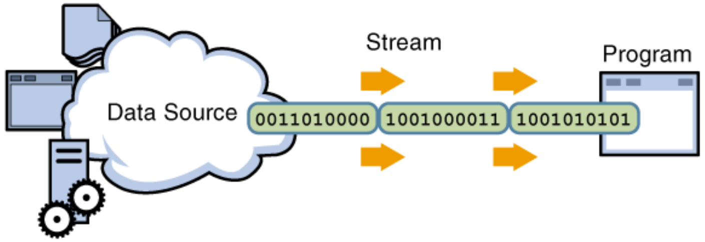
Dr Siba Haidar

Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

File Input/Output

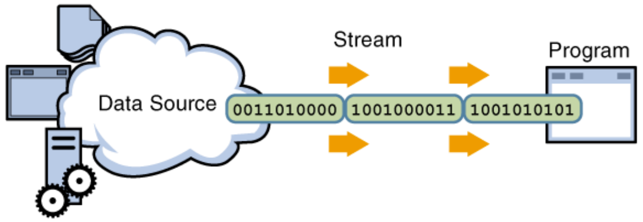
Chapter 5

File I/O



1. Streams
2. Opening & Closing Files
3. File Access Functions
4. Character I/O Functions
5. Formatted I/O Functions
6. Direct I/O Functions
7. Additional Remarks

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

Streams

- Input and output, to or from,
  - physical devices such as terminals and tape drives,
  - files supported on structured storage devices,
- are mapped into → logical data streams,
  - properties are more uniform
  - 2 forms of mapping are supported
    - text streams
    - binary streams.

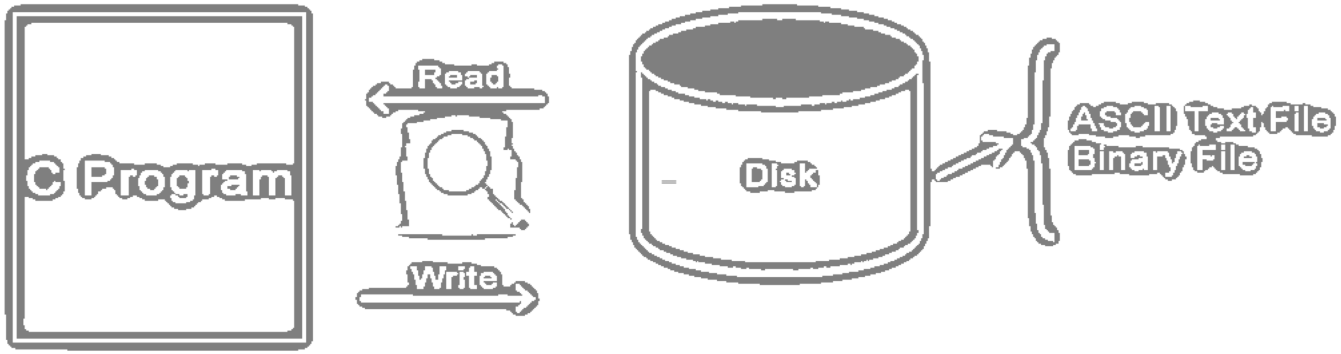
Streams

Text Stream

- ordered sequence of characters composed into lines,
- each line consisting of zero or more characters plus a terminating new-line character

Binary stream

- ordered sequence of characters (=bytes) that can transparently record internal data



Streams

Text Stream

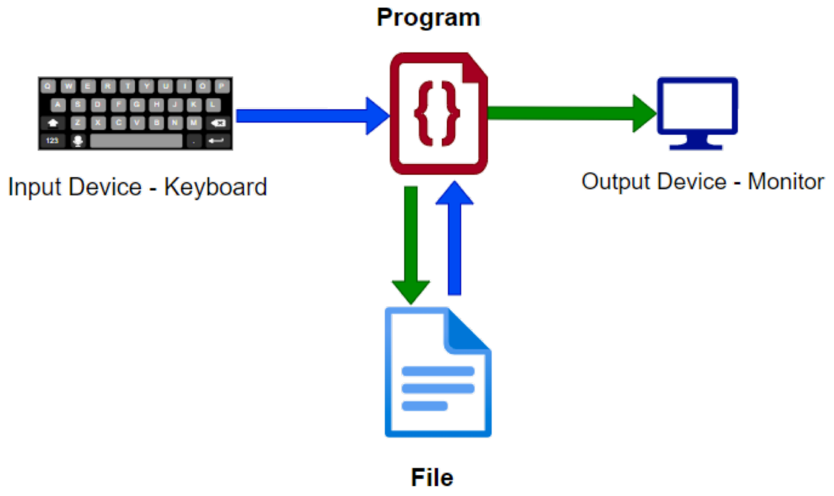
- data read in from a text stream will necessarily compare equal to the data that were earlier written out to that stream
- **<u>only if</u>**
- the data consist only of printable characters and the control characters horizontal tab and new-line,
- no new-line character is immediately preceded by space characters, and
- the last character is a new-line character

Binary stream

- data read in from a binary stream shall compare equal to the data that were earlier written out to that stream under the same implementation
- may, however, have an implementation-defined number of null characters appended to the end of the stream

Standard Streams

- when a C program start its execute
  - program automatically opens 3 standard streams
    - stdin → input buffering [by default: keyboard]
    - stdout & strerr → output [by default: screen]



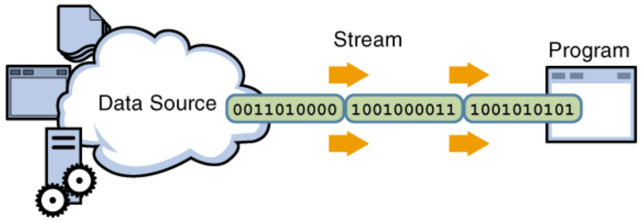
Example program

```
int main() {
  int var;
  scanf("%d",&var);
  //use stdin for scanning a integer from keyboard.
  printf("%d",var);
  //use stdout for printing a character
}
```

FILE pointers

- <stdio.h> header contains a definition for a type FILE
  - usually via a typedef
  - capable of recording all information to control a stream
    - file position indicator
    - pointer to associated buffer (if any),
    - error indicator that records whether a read/write error has occurred,
    - end-of-file indicator that records whether the end of the file has been reached
- access contents of FILE directly → bad manners !!
  - unless the programmer is writing an implementation of <stdio.h> and its contents

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

Opening and Closing Files

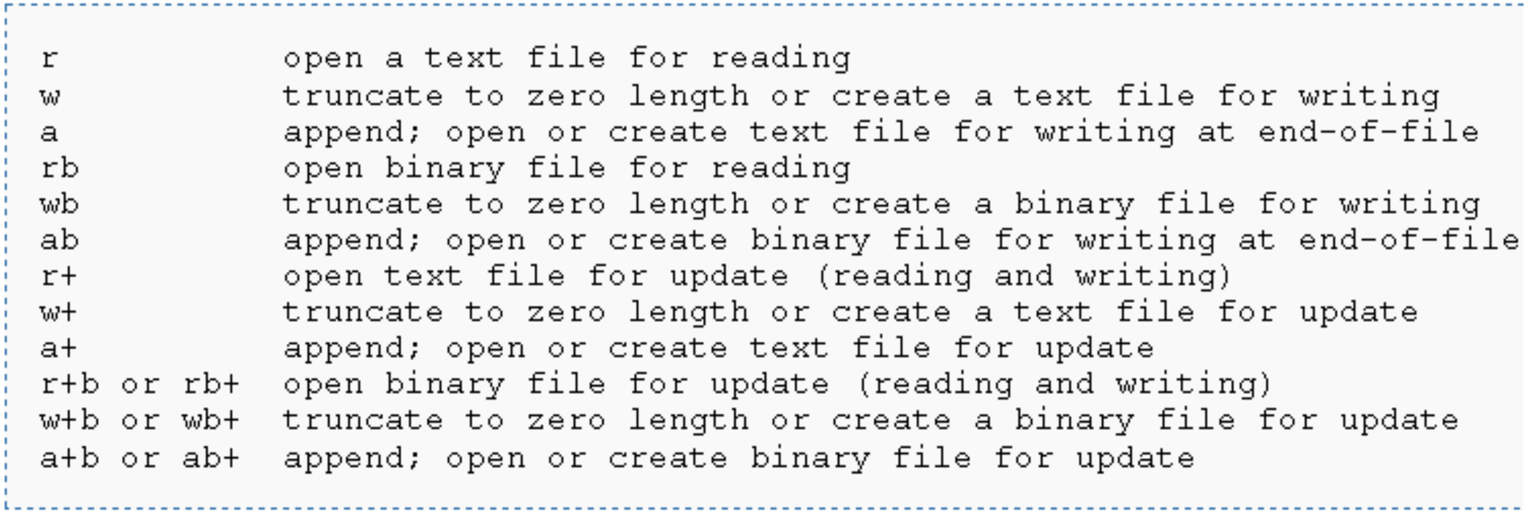
- to open and close files, the <stdio.h> library has three functions:
  - fopen,
  - freopen,
  - fclose

Opening Files

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

Modes

- The argument mode points to a string beginning with one of the following sequences:



r	open a text file for reading
w	truncate to zero length or create a text file for writing
a	append; open or create text file for writing at end-of-file
rb	open binary file for reading
wb	truncate to zero length or create a binary file for writing
ab	append; open or create binary file for writing at end-of-file
r+	open text file for update (reading and writing)
w+	truncate to zero length or create a text file for update
a+	append; open or create text file for update
r+b or rb+	open binary file for update (reading and writing)
w+b or wb+	truncate to zero length or create a binary file for update
a+b or ab+	append; open or create binary file for update

Modes

- open with read mode "r …"fails if the file does not exist or cannot be read
- open a file with append mode "a…"
  - all subsequent writes forced to end-of-file
  - regardless of calls to fseek

Modes

- opened with update mode "… + …"
  - both input and output may be performed
  - output may **NOT** be directly followed by input without a call to fflush() or to file positioning function (fseek, fsetpos, or rewind)
  - input may **NOT** be directly followed by output without a call to a file positioning function, unless the input operation encounters end-of-file.
  - opening (or creating) a text file with update mode may instead open (or create) a binary stream in some implementations

The fopen function

  - opens the file whose name is in the string pointed to by filename and
  - associates a stream with it.
- when opened,
  - a stream is fully buffered
  - the error and end-of-file indicators are cleared.
- function returns
  - a pointer to the object controlling the stream.
  - a null pointer if open fails

The freopen function

  - opens the file whose name is the string pointed to by filename and associates the stream pointed to by stream with it.
  - mode used in the fopen function.
- freopen
  - first close any file associated with stream
  - failure to close is ignored
  - error and end-of-file indicators for the stream are cleared.
- returns
  - a null pointer if fails
  - value stream if succeeds

```c
int main () {
   FILE *fp;

   printf("This text is redirected to stdout\n");

   fp = freopen("file.txt", "w+", stdout);

   printf("This text is redirected to file.txt\n");

   fclose(fp);

   return(0);
}
```

Let us compile and run the above program that will send the following line at STDOUT because initially we did not open stdout −

```
This text is redirected to stdout
```

After a call to **freopen()**, it associates STDOUT to file **file.txt**, so whatever we write at STDOUT that goes inside **file.txt**. So, the file **file.txt** will have the following content.

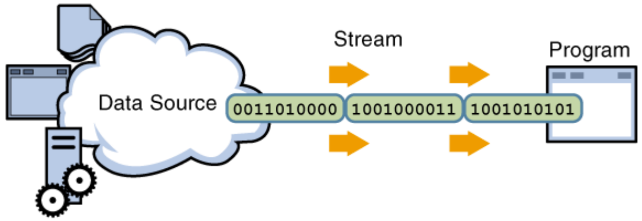
```
This text is redirected to file.txt
```

Closing Files

```
#include <stdio.h>
int fclose(FILE *stream);
```

- stream flushed and associated file closed
  - any unwritten buffered data for the stream
    - → delivered to the host environment
    - → to be written to the file;
  - any unread buffered data → discarded
  - stream is disassociated from file
  - if associated buffer auto. allocated → deallocated
- returns
  - zero if success
  - EOF if errors

File I/O


Data Source
Stream
0011010000
1001000011
1001010101
Program

The fseek and ftell functions

```
int fseek(FILE *stream, long int
offset, int whence);
long int ftell(FILE *stream);
```

- fseek function **sets** the file position indicator
- for a binary stream,
  - the new position, measured in characters from the beginning of the file,
  - is obtained by adding offset to the position specified by whence.
- 3 macros in stdio.h:
  - whence = SEEK_SET → position is the beginning of the file
  - whence = SEEK_CUR→ position is the current file position
  - whence = SEEK_END → position is the end of the file
- returns
  - nonzero only for a request that cannot be satisfied

The fseek and ftell functions

```
long int ftell(FILE *stream);
```

- ftell obtains the current value of the file position indicator
- for a binary stream,
  - the value is the number of characters from the beginning of the file;
- for a text stream,
  - unspecified information,
  - usable by the fseek function for returning the file position indicator for the stream to its position at the time of the ftell call;
  - the difference between two such return values is not necessarily a meaningful measure of the number of characters written or read.
- ftell returns
  - current value of file position indicator if success
  - -1L on failure, (stores an implementation-defined positive value in errno)

The fflush function

```
int fflush(FILE *stream);
```

- fflush causes any unwritten data for that stream to be deferred to the host environment to be written to the file;
- If stream is a null pointer, fflush performs this flushing action on all streams
- returns
  - EOF if a write error occurs,
  - otherwise zero.

The rewind function

```
void rewind(FILE *stream);
```

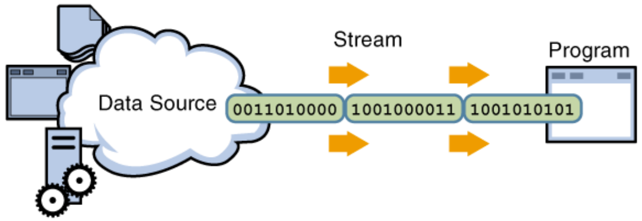
- sets the file position indicator to the beginning of the file.

The feof function

```
int feof(FILE *stream);
```

- tests the end-of-file indicator and
- returns
  - nonzero if and only if the end-of-file indicator is set for stream,
  - otherwise it returns zero.

Outline


Stream
Program
Data Source
0011010000
1001000011
1001010101

The fgetc function

```
int fgetc(FILE *stream);
```

- fgetc
  - obtains the next character (if present) as an unsigned char converted to an int,
  - and advances the associated file position indicator
- fgetc returns
  - the next character, of
  - if stream is at end-of-file, returns EOF
    - (EOF is a negative value defined in <stdio.h>, usually (-1)).
  - If a read error occurs, the error indicator for the stream is set and fgetc returns EOF.

The fgets function

```
char *fgets(char *s, int n, FILE *stream);
```

- fgets at most n-1 characters into s.
- no additional characters are read
  - after a new-line character (which is retained) or
  - after end-of-file.
- a null character is written immediately after the last character read into the array.
- returns s if successful
- if end-of-file is encountered and no characters have been read into the array,
  - the contents of the array remain unchanged and
  - a null pointer is returned.
- If a read error occurs during the operation,
  - the array contents are indeterminate and
  - a null pointed is returned.

The getc function

```
int getc(FILE *stream);
```

- The getc function is equivalent to fgetc,
  - except that it may be implemented as a macro.
- If it is implemented as a macro,
  - the stream argument may be evaluated more than once,
  - so the argument should never be an expression with side effects (i.e. have an assignment, increment, or decrement operators, or be a function call).
- The getc function returns the next character from the input stream pointed to by stream.
- If the stream is at end-of-file,
  - the end-of-file indicator for the stream is set and
  - getc returns EOF (EOF is a negative value defined in <stdio.h>, usually (-1)).
- If a read error occurs,
  - the error indicator for the stream is set and
  - getc returns EOF.

The getchar function

```
int getchar(void);
```

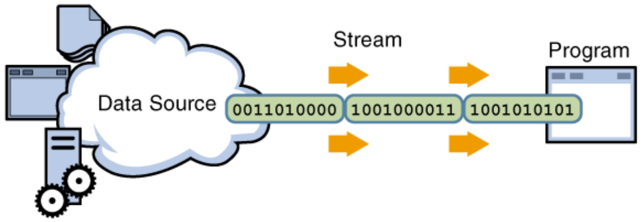
- getchar is equivalent to getc with argument stdin.

The gets function

```
char *gets(char *s);
```

- gets is equivalent to fgets with argument stdin.
- This function and description is only included here for completeness.

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

The fputc function

```
int fputc(int c, FILE *stream);
```

- fputc writes
  - the character specified by c (converted to an unsigned char)
  - at the position indicated by the associated file position indicator,
  - and advances the indicator appropriately.
- If the file
  - cannot support positioning requests,
  - or if the stream is opened with append mode,
  - the character is appended to the output stream.
- returns
  - the character written,
  - if a write error occurs,
    - error indicator for the stream is set, and
    - fputc returns EOF

The fputs function

```
int fputs(const char *s, FILE *stream);
```

- fputs writes the string pointed to by s
- The terminating null character is not written.
- The function returns
  - EOF if a write error occurs,
  - otherwise it returns a nonnegative value.

The putc function

```
int putc(int c, FILE *stream);
```

- putc is equivalent to fputc except that if it is implemented as a macro

The putchar function

```
int putchar(int c);
```

- putchar is equivalent to putc with the second argument stdout.

The puts function

```
int puts(const char *s);
```

- puts writes
  - the string pointed to by s and
  - appends a new-line character to the output.
- The terminating null character is not written.
- returns
  - EOF if a write error occurs;
  - otherwise, it returns a nonnegative value.

Exemple 1

- write a program that creates a text file and stores in it 3 names (one name per line)
- call it "names.txt"

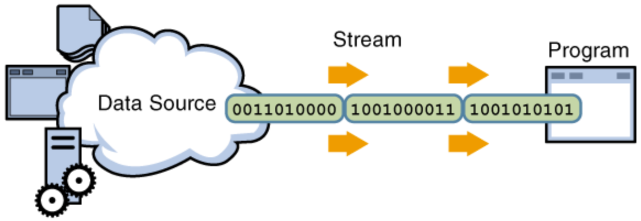
```c
#include <stdio.h>
#include <conio.h>

void main(){
char * filename="names.txt", *s1="toto\n",*s2="mimi\n",*s3="sisi\n";
int r;
FILE * f;

f=fopen(filename,"w");
if(f==NULL){ printf("fail to create\n"); return; }

r=fputs(s1,f);
if (r==EOF){ printf("fail to write\n"); return; }

r=fputs(s2,f);
if (r==EOF){ printf("fail to write\n"); return; }

r=fputs(s3,f);
if (r==EOF){ printf("fail to write\n"); return; }

r=fclose(f);
if(r!=0){ printf("fail to save\n"); return; }

printf("Success: file written\n");
getch();

}
```

Example 2

- write a program to open “names.txt”
- and prints its content on the screen

```
#include <stdio.h>
#include <conio.h>

void main(){
char * filename="names.txt";
int r;
FILE * f;
char s[256];

if(!(f=fopen(filename,"r"))){
printf("Fail to open the file\n");return;}
while(!feof(f)){
if(fgets(s,256,f))
puts(s);
}

if(fclose(f)==0)
printf("Success: file read and closed\n");
getch();

}
```

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

The scanf family of functions

```
int fscanf(FILE *stream, const char *format, ...);
int scanf(const char *format, ...);
int sscanf(const char *s, const char *format, ...);
```

The scanf function

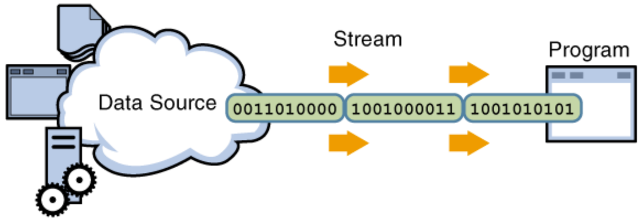
- fscanf
  - reads input from stream
  - under control of format that specifies the admissible sequences and how they are to be converted for assignment,
  - using subsequent arguments as pointers to the objects to receive converted input.
- fscanf returns
  - EOF if an input failure occurs before any conversion.
  - otherwise, the number of input items assigned

- scanf is equivalent to fscanf with argument stdin

The sscanf function

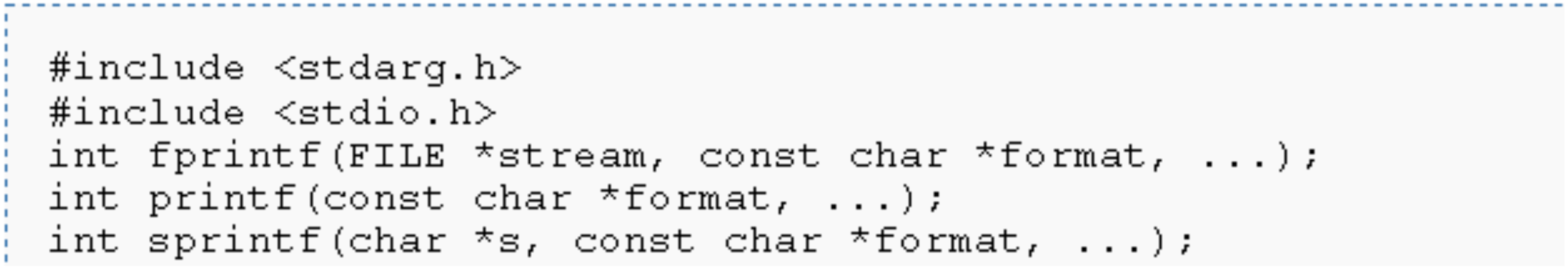
- sscanf is equivalent to fscanf,
  - except that the argument s specifies a string from which the input is to be obtained,
  - rather than from a stream
  - reaching the end of the string is equivalent to encountering the end-of-file for the fscanf function

File I/O



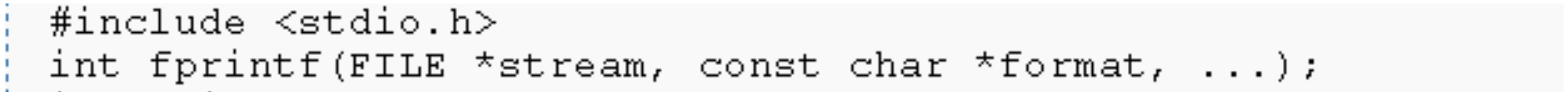
Stream
Program
Data Source
0011010000
1001000011
1001010101

The printf family of functions



```
#include <stdarg.h>
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
int sprintf(char *s, const char *format, ...);
```

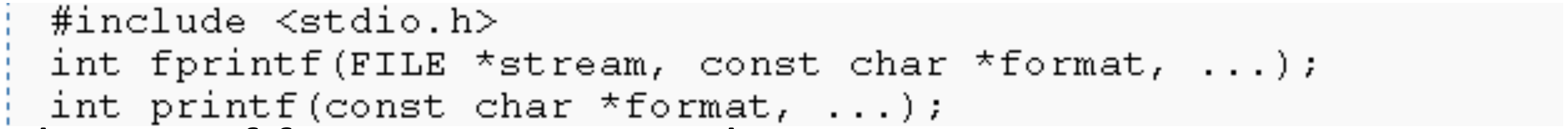
fprintf



```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

- The fprintf function returns
  - the number of characters transmitted, or
  - a negative value if an output or encoding error occurred.

printf



```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
```

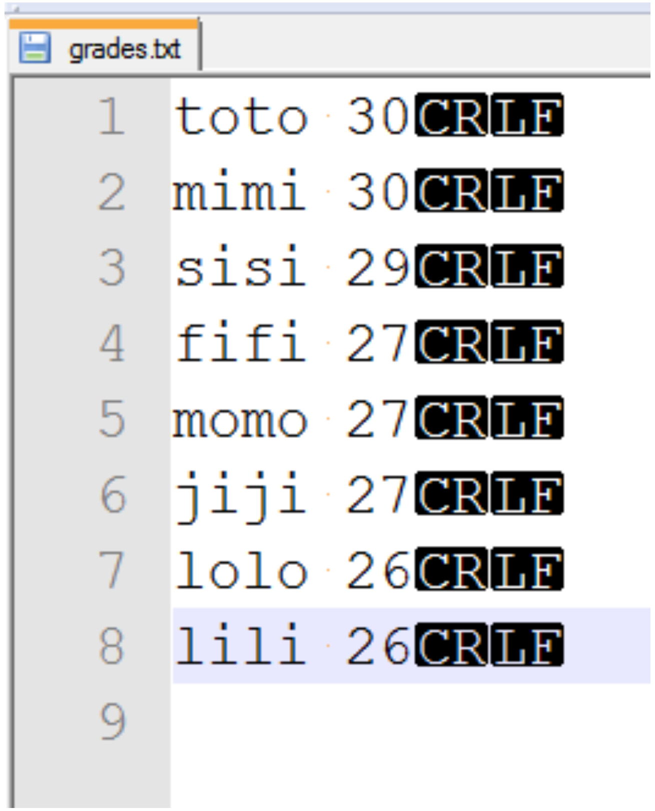
- The printf function is equivalent to
  - fprintf with the argument stdout interposed before the arguments to printf.
- It returns
  - the number of characters transmitted, or
  - a negative value if an output error occurred.

sprintf

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
int printf(const char *format, ...);
int sprintf(char *s, const char *format, ...);
```

- The sprintf function is equivalent to fprintf,
- except that the argument s specifies an array into which the generated input is to be written, rather than to a stream.
- A null character is written at the end of the characters written;
- it is not counted as part of the returned sum.
- If copying takes place between objects that overlap, the behavior is undefined.
- The function returns
  - the number of characters written in the array,
  - not counting the terminating null character.

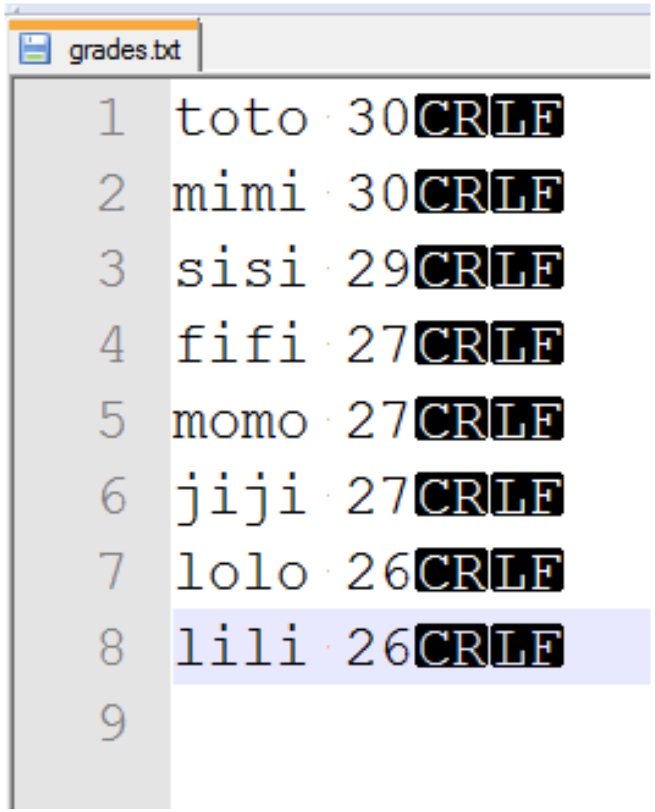
example 3


grades.txt

```
toto 30CRLF
mimi 30CRLF
sisi 29CRLF
fifi 27CRLF
momo 27CRLF
jiji 27CRLF
lolo 26CRLF
lili 26CRLF

```

- write a program that reads from this file
- and displays the avg

example 3 bis

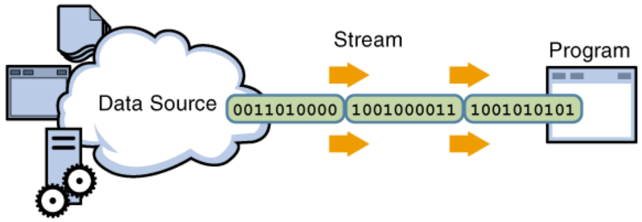


grades.txt

```
toto 30
mimi 30
sisi 29
fifi 27
momo 27
jiji 27
lolo 26
lili 26
```

- write a program that reads from this file
- and adds 2 pointsn to the grade of jiji if it is less than 97

File I/O

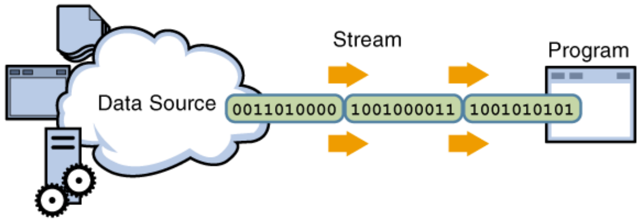

Stream
Program
Data Source
0011010000
1001000011
1001010101

The fread function

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- The fread function reads,
  - into the array pointed to by ptr,
  - up to nmemb elements whose size is specified by size,
  - from the stream pointed to by stream.
- The file position indicator for the stream (if defined) is advanced by the number of characters successfully read.
- If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.
- If a partial element is read, its value is indeterminate.
- The fread function returns
  - the number of elements successfully read,
  - which may be less than nmemb if a read error or end-of-file is encountered.
  - If size or nmemb is zero, fread returns zero and the contents of the array and the state of the stream remain unchanged

File I/O

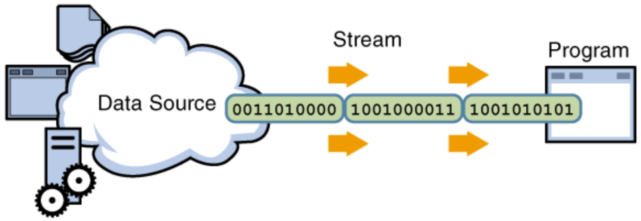

Stream
Program
Data Source
0011010000
1001000011
1001010101

The fwrite function

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
FILE *stream);
```

- The fwrite function writes,
  - from the array pointed to by ptr,
  - up to nmemb elements whose size is specified by size to the stream pointed to by stream.
- The file position indicator for the stream (if defined) is advanced by the number of characters successfully written.
- If an error occurs,
  - the resulting value of the file position indicator for the stream is indeterminate.
- The function returns the number of elements successfully written, which will be less than nmemb only if a write error is encountered.

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

The C stdio fseek, ftell and rewind Functions for use with Files

- In computer programming, the two main types of file handling are:
  - Sequential;
  - Random access.
- Sequential files are generally used in cases where the program processes the data in a sequential fashion
  - i.e. counting words in a text file
  - although in some cases, random access can be feigned by moving backwards and forwards over a sequential file.
- True random access file handling, however, only accesses the file at the point at which the data should be read or written, rather than having to process it sequentially.
- A hybrid approach is also possible whereby a part of the file is used for sequential access to locate something in the random access portion of the file, in much the same way that a File Allocation Table (FAT) works.

Sequential and Random Access File

- The three main functions are:
  - rewind() – return the file pointer to the beginning;
  - fseek() – position the file pointer;
  - ftell() – return the current offset of the file pointer.
- Each of these functions operates on the C file pointer,
  - which is just the offset from the start of the file,
  - and can be positioned at will.
- All read/write operations take place at the current position of the file pointer.

The rewind() Function

- The rewind() function can be used in sequential or random access C file programming, and simply tells the file system to position the file pointer at the start of the file.
- Any error flags will also be cleared, and no value is returned.
- While useful, the companion function, fseek(), can also be used to reposition the file pointer at will, including the same behavior as rewind().

Using fseek() and ftell() to Process Files

- The fseek() function is most useful in random access files where either the record (or block) size is known, or there is an allocation system that denotes the start and end positions of records in an index portion of the file.
- The fseek() function takes three parameters:
  - FILE * f – the file pointer;
  - long offset – the position offset;
  - int origin – the point from which the offset is applied.
- The origin parameter can be one of three values:
  - SEEK_SET – from the start;
  - SEEK_CUR – from the current position;
  - SEEK_END – from the end of the file.

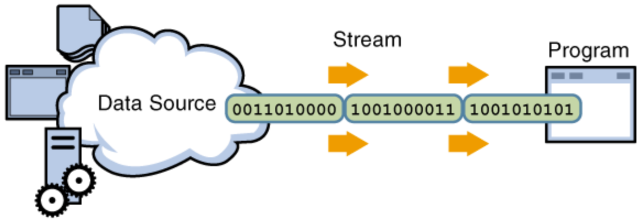
Using fseek() and ftell() to Process Files

- So, the equivalent of rewind() would be:
  - fseek(f, 0, SEEK_SET);
- By a similar token, if the programmer wanted to append a record to the end of the file, the pointer could be repositioned thus:
  - fseek(f, 0, SEEK_END);
- Since fseek() returns an error code (0 for no error) the stdio library also provides a function that can be called to find out the current offset within the file:
  - long offset = ftell(FILE * f)
- This enables the programmer to create a simple file marker (before updating a record for example), by storing the file position in a variable, and then supplying it to a call to fseek:
  - long file_marker = ftell(f);

Using fseek() and ftell() to Process Files

- // … file processing functions
  - fseek(f, file_marker, SEEK_SET);
- Of course, if the programmer knows the size of each record or block, arithmetic can be used.
- For example, to rewind to the start of the current record, a function call such as the following would suffice:
  - fseek(f, 0 – record_size, SEEK_CURR);
- With these three functions, the C programmer can manipulate both sequential and random access files, but should always remember that positioning the file pointer is absolute.
- In other words, if fseek is used to position the pointer in a read/write file, then writing will overwrite existing data, permanently.

File I/O


Stream
Program
Data Source
0011010000
1001000011
1001010101

The while(!feof(f)) loop

- Inside !feof loop,
  - one should always check wether the input function succeeded or failed
  - before further processing
- For the 3 types of the functions:
  - direct input:
    - fread returns the number of elements successfully read, so you can know if it failed
  - formatted input:
    - fscanf returns EOF if failed
  - character/string input:
    - fgetc return EOF if failed,
    - fgets returns NULL if failed

References

- http://en.wikibooks.org/wiki/C_Programming/File_IO