Files in C

FILES

- Discrete storage unit for data in the form of a stream of bytes.
- Durable: stored in non-volatile memory.
- Starting end, sequence of bytes, and end of stream (or end of file).
- Sequential access of data by a pointer performing read / write / deletion / insertion.
- Meta-data (information about the file) before the stream of actual data.

Difference: File I/O & Console I/O

The primary difference between File I/O and Console I/O (the term *console* here refers to the screen-keyboard pair) is:

- The console always exists; a particular file may or may not exist.
- In case of console, the program reads from the keyboard and writes onto the screen. In case of files, it is possible to read from and write to the same file.

File Program Sequence

File I/O is always done in a program in the following sequence:

- Open the file
- Read or write to the file
- Close the file

Opening File

- Before performing any file I/O, the file must be opened.
- While opening file, the following are specified:
  - The name of the file
  - The manner in which it should be opened (i.e. for reading, writing, both reading and writing, appending at the end of the file, overwriting the file, etc.)
- The function `fopen` is used to open a file. It accept two strings, the first is the *name* of the file and the second is the *mode* in which it should be opened.

Opening File

```
FILE *fp;
fp = fopen (“outfile1.txt”, “w”);
```

- The first statement declares `fp` the pointer toa s `FILE` structure. This structure is defined in `stdio.h`.
- The function `fopen` returns a pointer to the `FILE` structure which it creates.
- This pointer must be used in subsequent operations on the file. Such as reading from or writing to it.
- The `FILE` pointer, `fp`, is also said to represent a *stream (or file descriptor)*.
- If a file called `outfile1.txt` already exists, it is deleted and re-written.

Modes for Opening Files

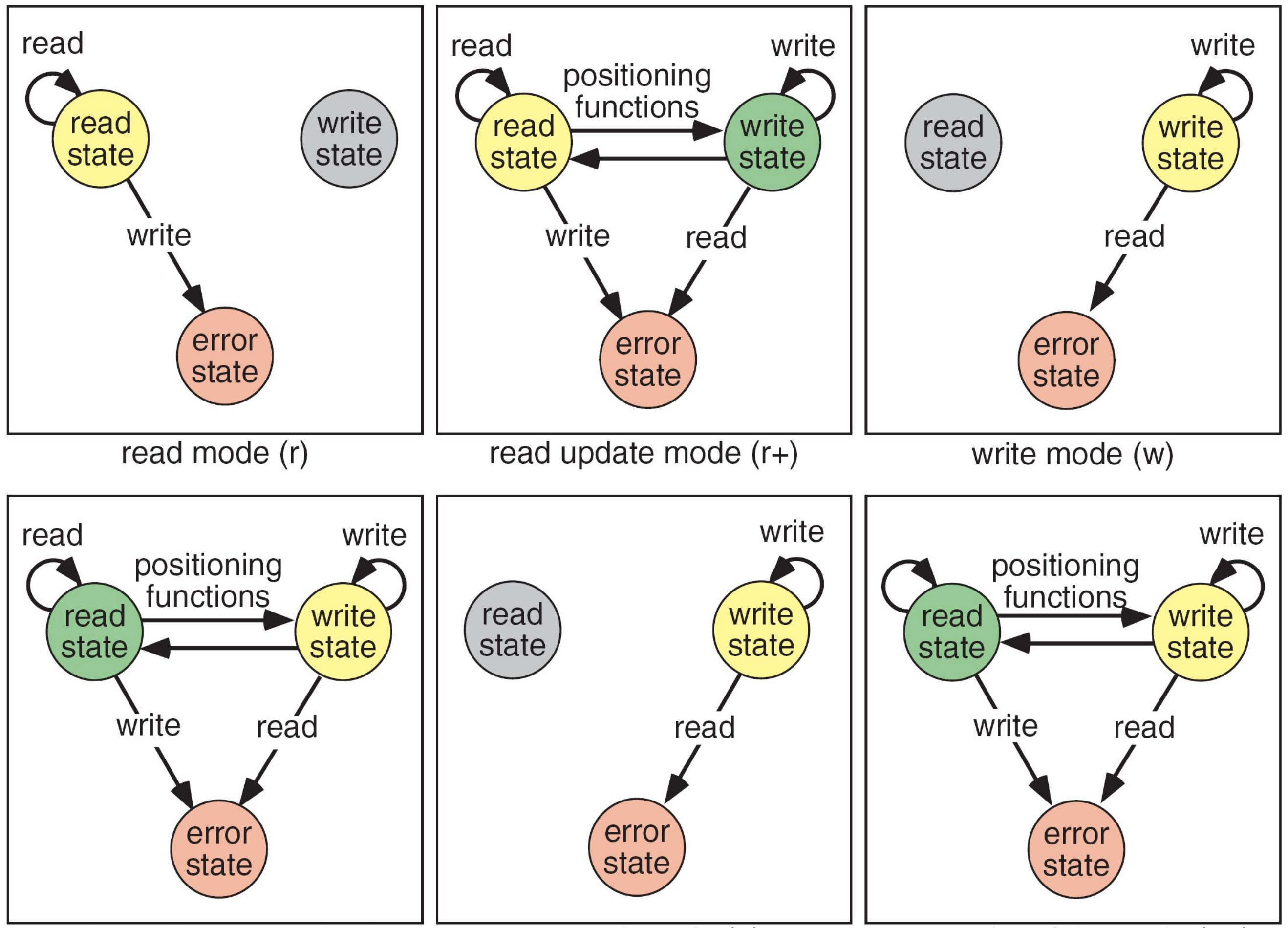
- The second argument of `fopen` is the `mode` in which we open the file. There are three
- "r" opens a file for reading
- "w" creates a file for writing - and writes over all previous contents (deletes the file so be careful!)
- "a" opens a file for appending - writing on the end of the file
- “rb” read binary file (raw bytes)
- “wb” write binary file

Additional modes

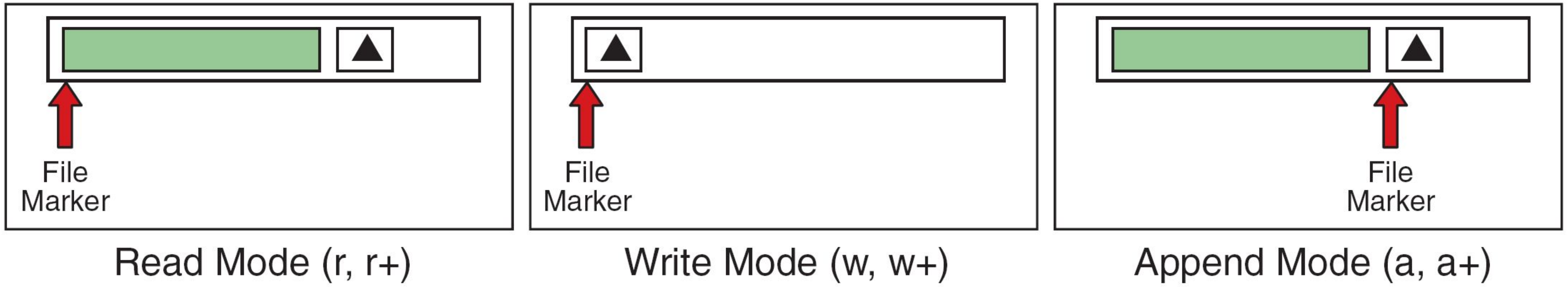
- r+ open to beginning for both reading/writing
- w+ same as w except both for reading and writing
- a+ same as 'a' except both for reading and writing

Mode	r	w	a	r+	w+	a+
Open State	read	write	write	read	write	write
Read Allowed	yes	no	no	yes	yes	yes
Write Allowed	no	yes	yes	yes	yes	yes
Append Allowed	no	no	yes	no	no	yes
File Must Exist	yes	no	no	yes	no	no
Contents of Existing File Lost	no	yes	no	no	yes	no

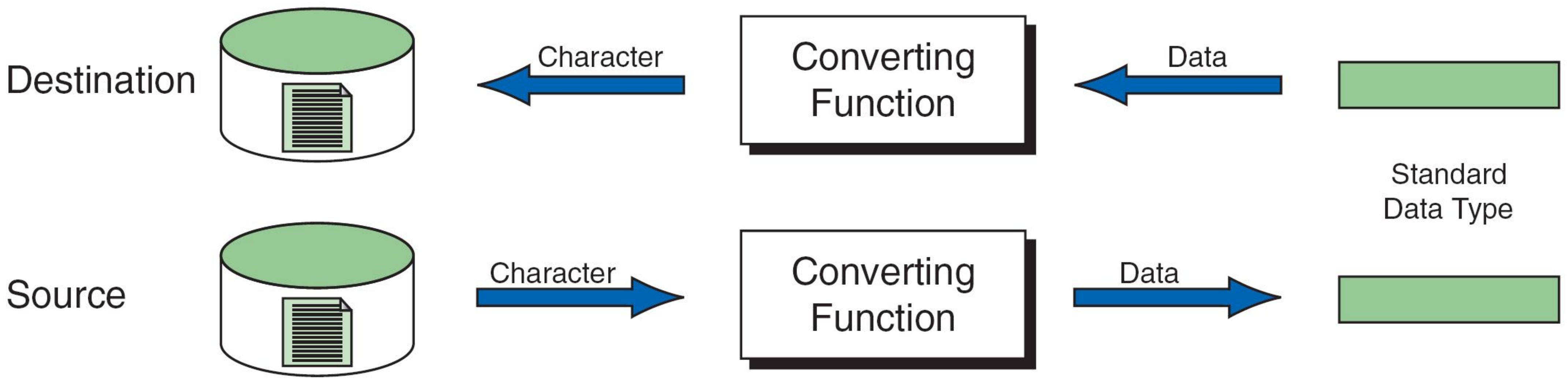
File States


read
read state
write state
write
error state
read mode (r)
read
write
positioning functions
read state
write state
write
read
error state
read update mode (r+)
write
read state
write state
read
error state
write mode (w)
read
write
positioning functions
read state
write state
write
read
error state
write update mode (w+)
write
read state
write state
read
error state
append mode (a)
write
positioning functions
read state
write state
write
read
error state
append update mode (a+)

File-Opening Modes

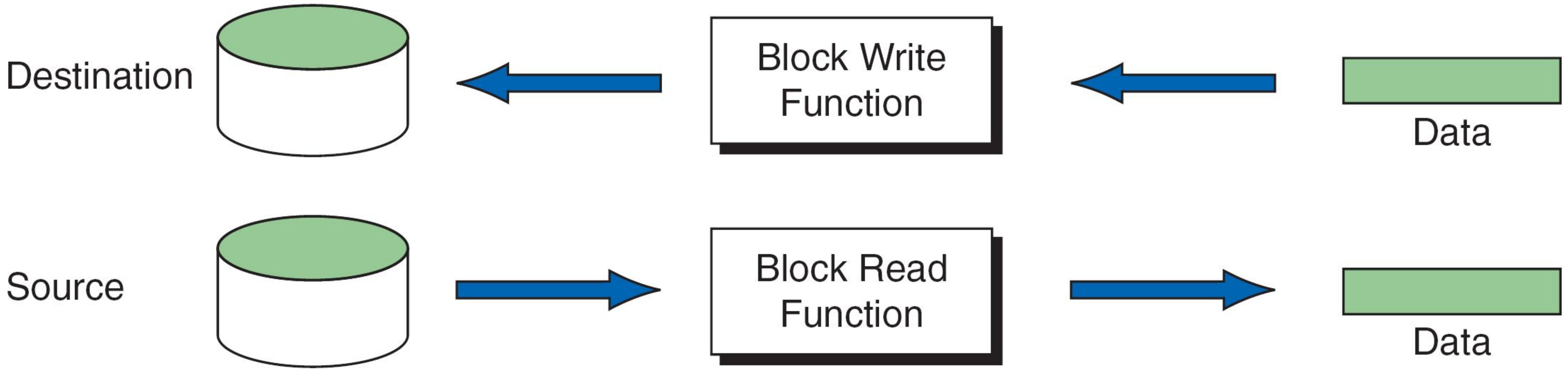


Reading and Writing Text Files

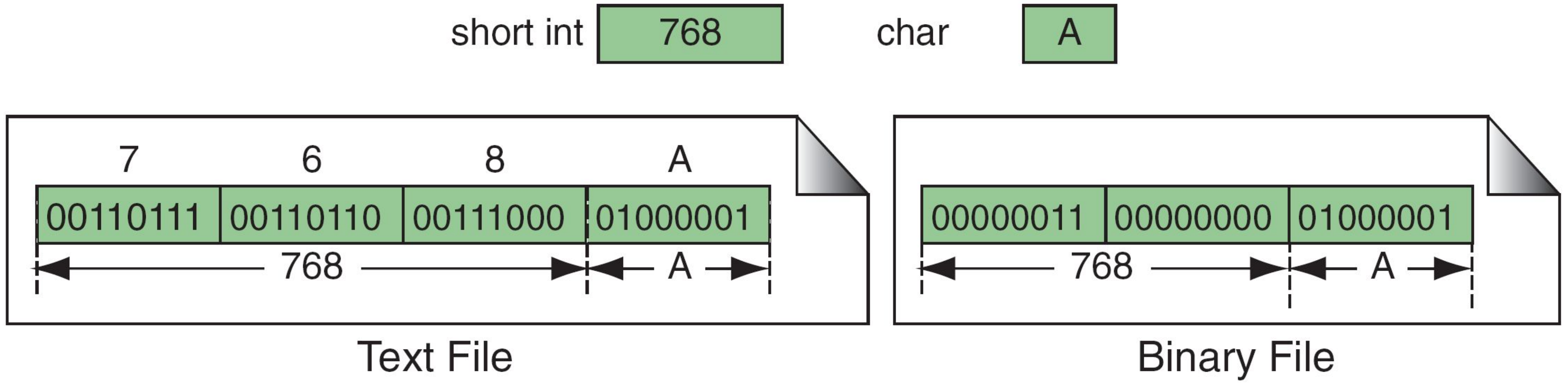


Formatted input/output, character input/output, and string input/output functions can be used only with text files.

Block Input and Output

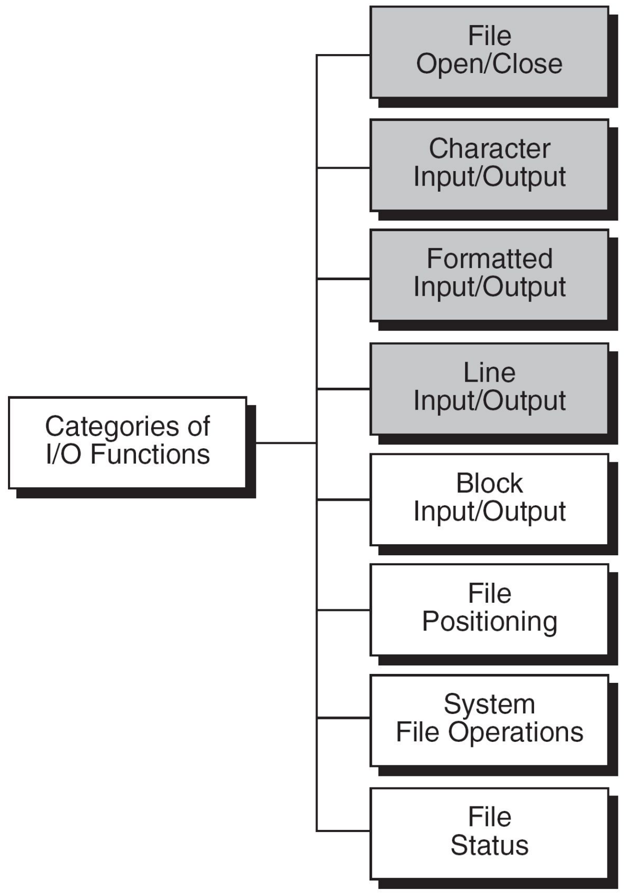


Binary and Text Files



Text files store data as a sequence of characters;
binary files store data as they are stored in primary memory.

Types of Standard Input/Output Functions



Input/Output Operations on Files

- C provides several different functions for reading/writing
  - getc() – read a character
  - putc() – write a character
  - fprintf() – write set of data values
  - fscanf() – read set of data values
  - getw() – read integer
  - putw() – write integer

getc( ) and putc( )

- handle one character at a time like getchar() and putchar()
- syntax: putc(c,fp1);
  - c : a character variable
  - fp1 : pointer to file opened with mode w
- syntax: c = getc(fp2);
  - c : a character variable
  - fp2 : pointer to file opened with mode r
- file pointer moves by one character position after every getc() and putc()
- getc() returns end-of-file marker EOF when file end reached

Program to Read/Write Using getc/putc

```
#include <stdio.h>
 main()
 { FILE *fp1;
   char c;
   f1= fopen(“INPUT”, “w”); /* open file for writing */

   while((c=getchar()) != EOF) /*get char from keyboard until CTL-Z*/
      putc(c,f1); /*write a character to INPUT */

   fclose(f1);                /* close INPUT */
   f1=fopen(“INPUT”, “r”);       /* reopen file */

   while((c=getc(f1))!=EOF)      /*read character from file INPUT*/
      printf(“%c”, c);           /* print character to screen */
```

fscanf() and fprintf()

- similar to scanf() and printf()
- in addition provide file-pointer
- given the following
  - file-pointer f1 (points to file opened in write mode)
  - file-pointer f2 (points to file opened in read mode)
  - integer variable i
  - float variable f
- Example:

```
fprintf(f1, "%d %f\n", i, f);
fprintf(stdout, "%f \n", f);     /*note: stdout refers to screen */
fscanf(f2, "%d %f", &i, &f);
```

- fscanf returns EOF when end-of-file reached

getw() and putw()

- handle one integer at a time
- syntax: putw(i,fp1);
  - i : an integer variable
  - fp1 : pointer to file ipened with mode w
- syntax: i = getw(fp2);
  - i : an integer variable
  - fp2 : pointer to file opened with mode r
- file pointer moves by one integer position, data stored in binary format native to local system
- getw() returns end-of-file marker EOF when file end reached

C program using getw, putw,fscanf, fprintf

```
#include <stdio.h>
main()
{ int i,sum1=0;
  FILE *f1;
  /* open files */
  f1 = fopen("int_data.bin","w");
  /* write integers to files in binary and text format*/
for(i=10;i<15;i++)          putw(i,f1);
fclose(f1);
f1 = fopen("int_data.bin","r");
   while((i=getw(f1))!=EOF)
    {   sum1+=i;
      printf("binary file: i=%d\n",i);
    } /* end while getw */
printf("binary sum=%d,sum1);
 fclose(f1);
}
```

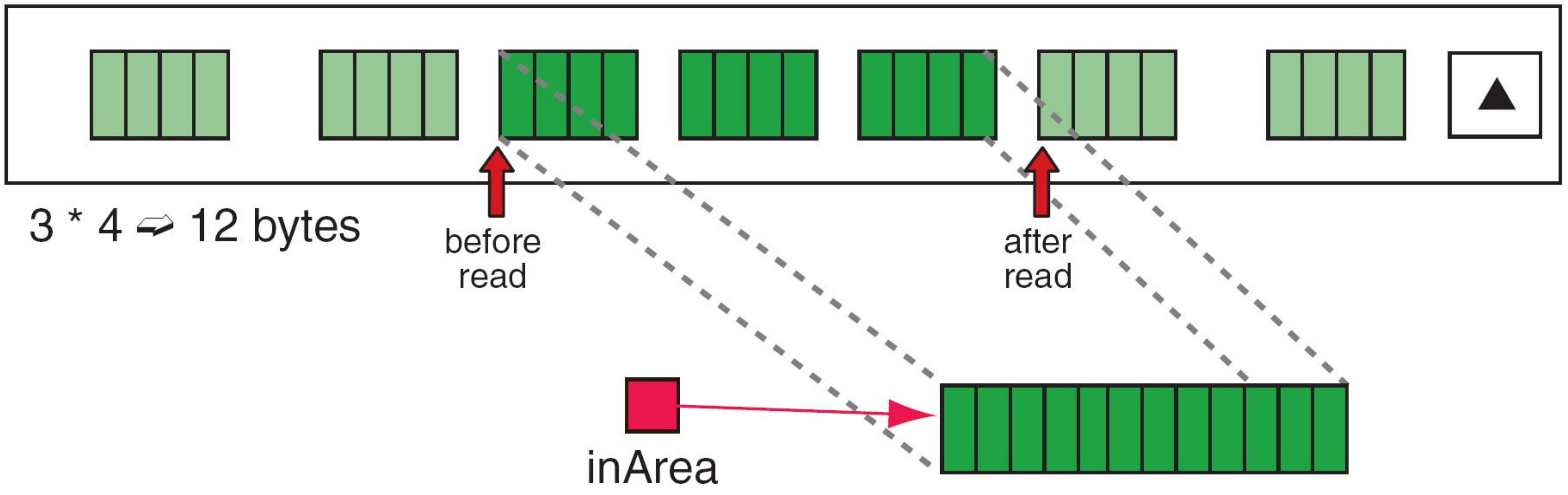
```
#include <stdio.h>
main()
{  int i, sum2=0;
  FILE *f2;
  /* open files */
  f2 = fopen("int_data.txt","w");
  /* write integers to files in binary and text format*/
for(i=10;i<15;i++) printf(f2,"%d\n",i);
fclose(f2);
f2 = fopen("int_data.txt","r");
while(fscanf(f2,"%d",&i)!=EOF)
    { sum2+=i; printf("text file: i=%d\n",i);
    } /*end while fscanf*/
  printf("text sum=%d\n",sum2);
  fclose(f2);
}
```

On Execution of Previous Programs

```
$ ./a.out
binary file: i=10
binary file: i=11
binary file: i=12
binary file: i=13
binary file: i=14
binary sum=60,
$ cat int_data.txt
10
11
12
13
14
```

```
$ ./a.out
text file: i=10
text file: i=11
text file: i=12
text file: i=13
text file: i=14
text sum=60
$ more int_data.bin
^@^@^@^K^@^@^@^L^@^@^@^M^@^@^@^N^
   @^@^@
$
```

File Read Operation

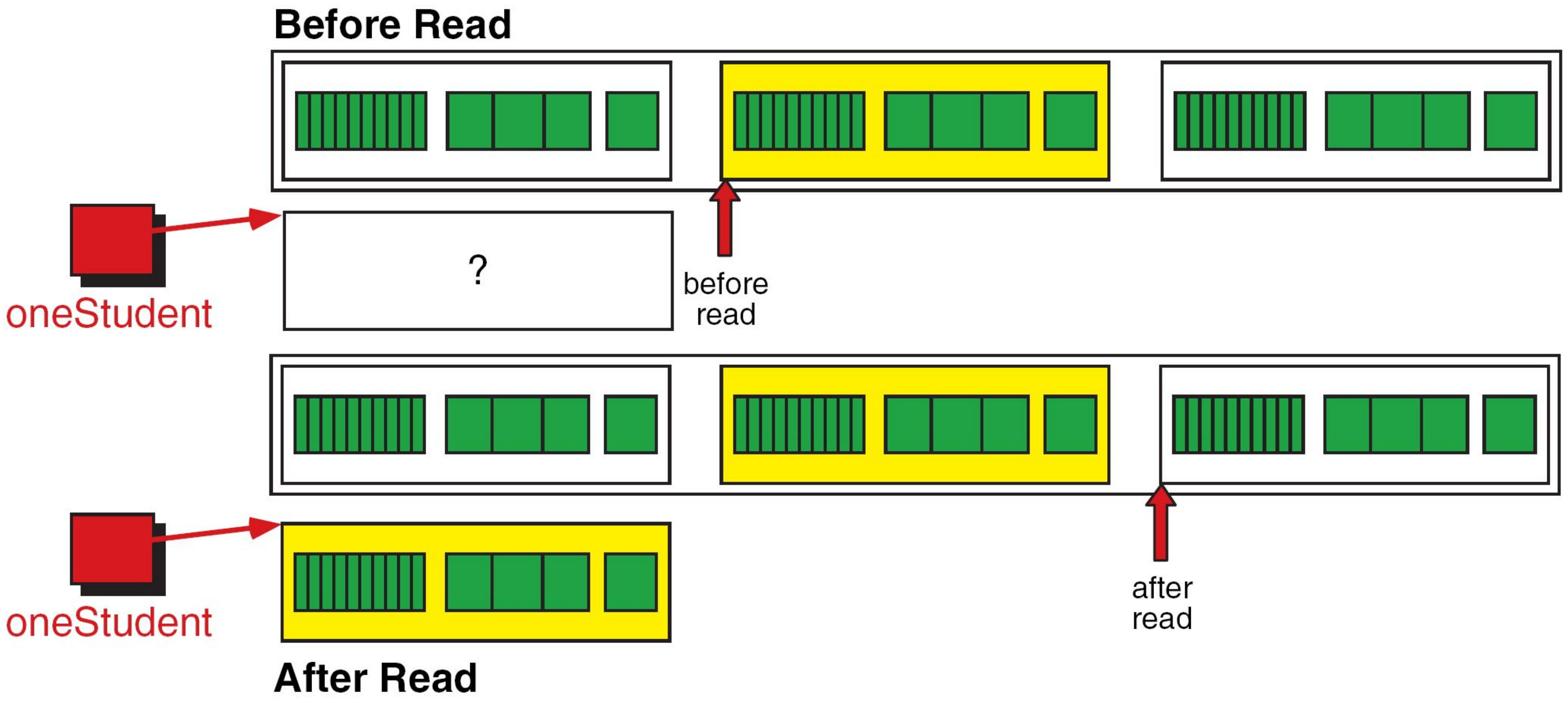


```
fread (inArea, sizeof (int), 3, spData);
```

READ FILE OF INTEGERS

```
// Read a file of integers, three integers at a time.
{
   …
// Local Declarations
   FILE* spIntFile;
   int   itemsRead;
   int   intAry[3];

// Statements
   spIntFile = fopen("int_file.dat", "rb");
    …
   while ((itemsRead = fread(intAry,
            sizeof(int), 3, spIntFile)) != 0)
      {
       // process array
             …
      } // while
   …
}  // block
```

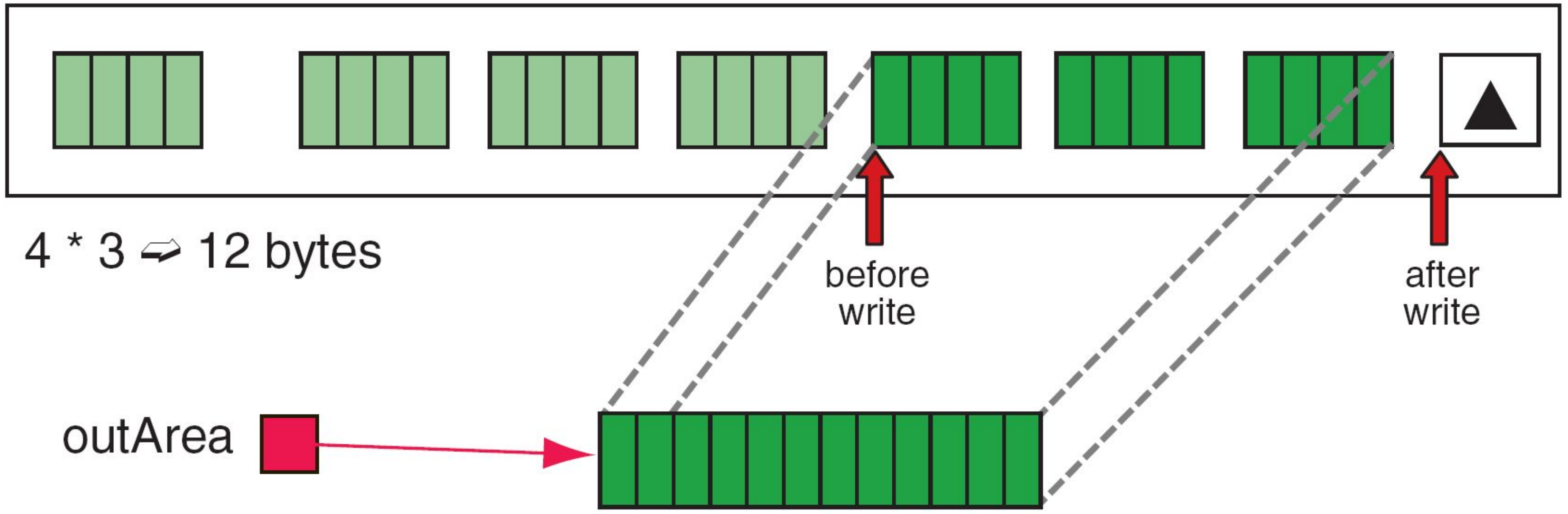
Reading a Structure



Read Student File

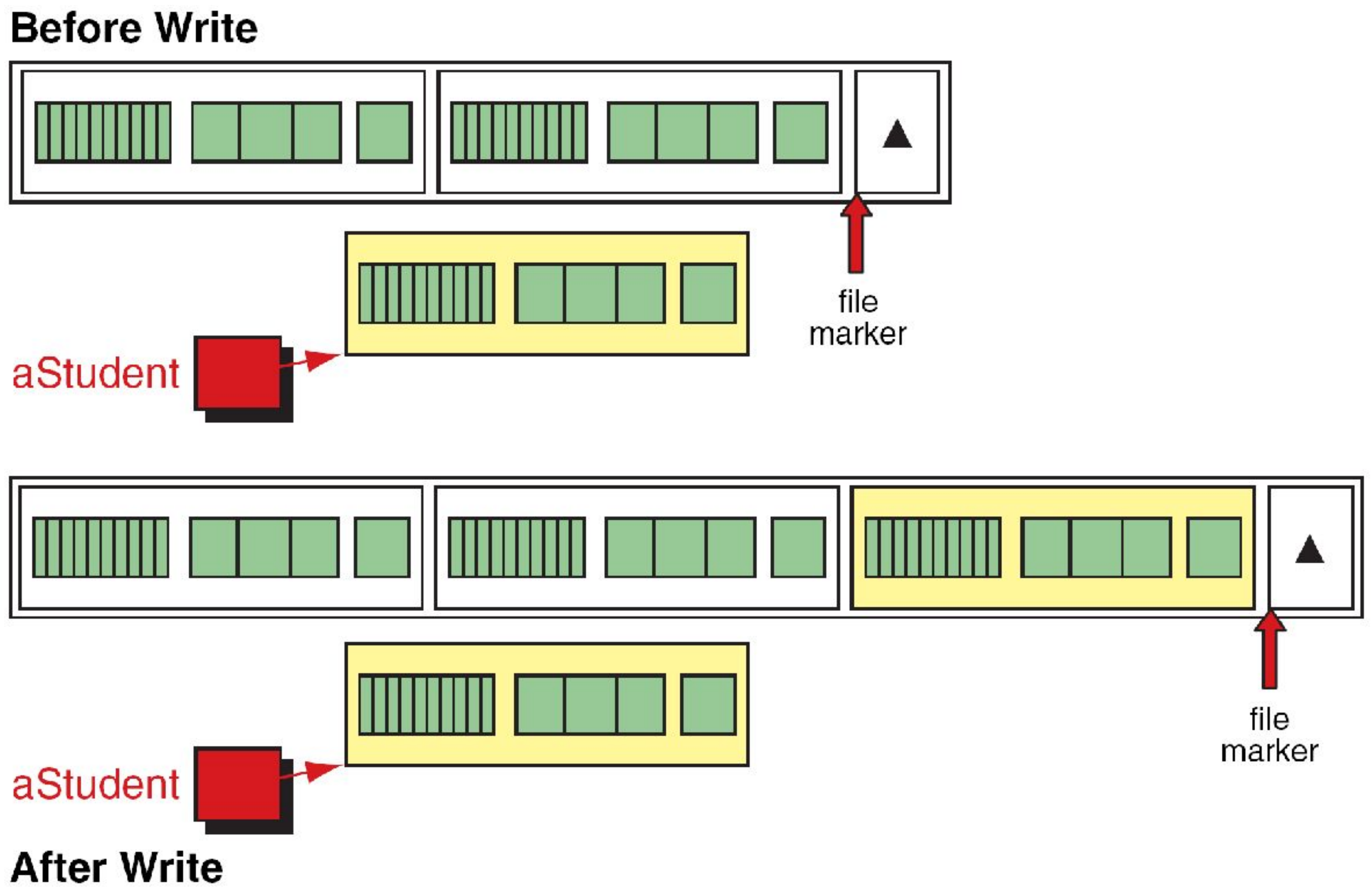
```
/* Reads one student's data from a file
      Pre   spStuFile is opened for reading
      Post  stu data structure filled
            ioResults returned
*/
int readStudent (STU* oneStudent, FILE* spStuFile)
{
// Local Declarations
   int ioResults;

// Statements
   ioResults = fread(oneStudent,
                     sizeof(STU), 1, spStuFile);
   return ioResults;
} // readStudent
```

FILE WRITE OPERATION

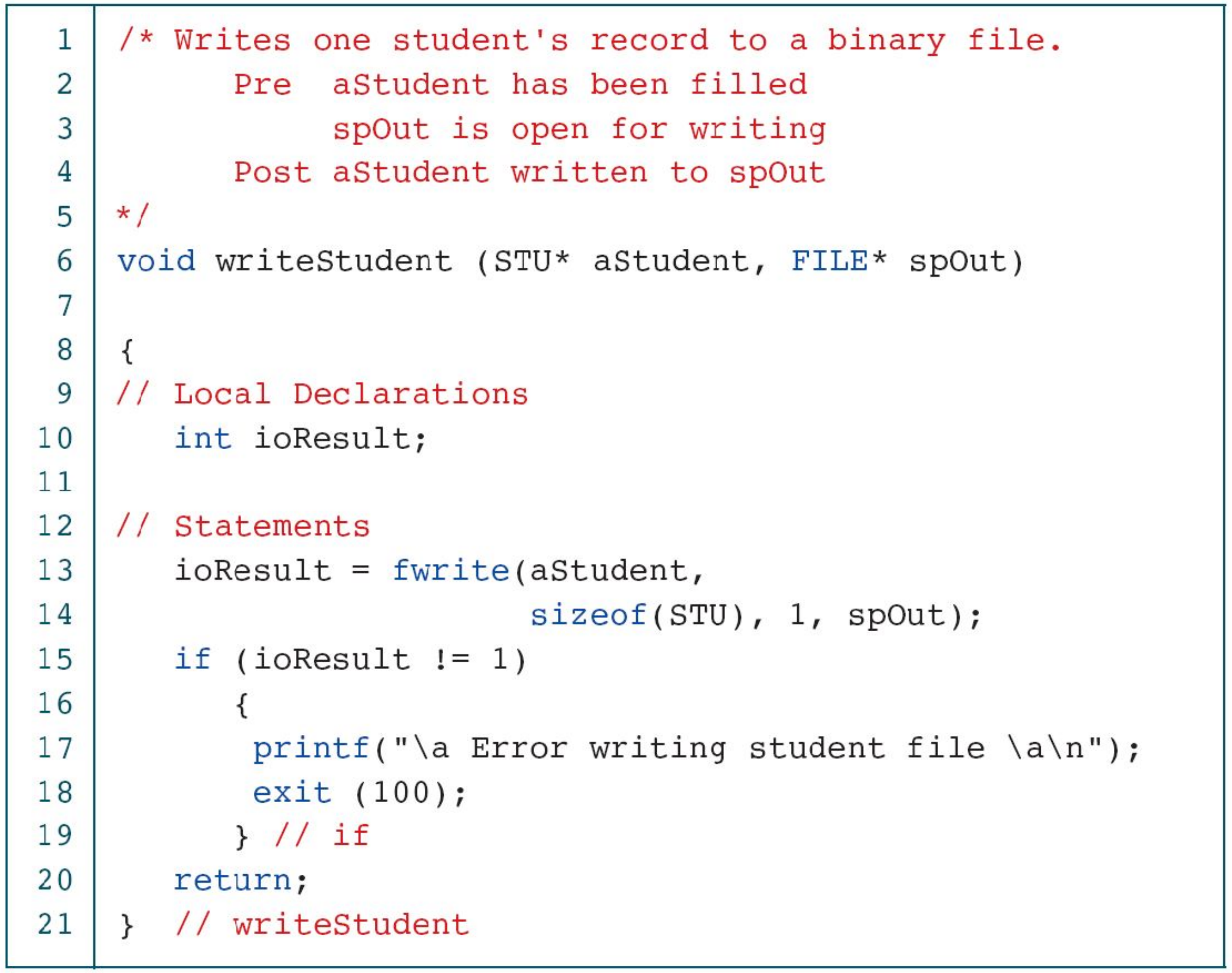


```
fwrite (outArea, sizeof (int), 3, spOut);
```

Writing a Structure



Write Structured Data



```c
/* Writes one student's record to a binary file.
      Pre  aStudent has been filled
           spOut is open for writing
      Post aStudent written to spOut
*/
void writeStudent (STU* aStudent, FILE* spOut)

{
// Local Declarations
   int ioResult;

// Statements
   ioResult = fwrite(aStudent,
                     sizeof(STU), 1, spOut);
   if (ioResult != 1)
      {
       printf("\a Error writing student file \a\n");
       exit (100);
      } // if
   return;
}  // writeStudent
```

Errors That Occur During I/O

- Typical errors that occur
  - trying to read beyond end-of-file
  - trying to use a file that has not been opened
  - perform operation on file not permitted by ‘fopen’ mode
  - open file with invalid filename
  - write to write-protected file

Error handling

- given file-pointer, check if EOF reached, errors while handling file, problems opening file etc.
- check if EOF reached: feof()
- feof() takes file-pointer as input, returns nonzero if all data read and zero otherwise

```
if(feof(fp))
        printf("End of data\n");
```

- ferror() takes file-pointer as input, returns nonzero integer if error detected else returns zero

```
if(ferror(fp) !=0)
        printf("An error has occurred\n");
```

Error while opening file

- if file cannot be opened then fopen returns a NULL pointer
- Good practice to check if pointer is NULL before proceeding

```
fp = fopen(“input.dat”, “r”);

if (fp == NULL)
    printf(“File could not be opened \n ”);
```