CMPE 252
C PROGRAMMING

SPRING 2021

WEEK 13

TEXT AND BINARY FILE POINTERS
CHAPTER 11

Problem Solving & Program Design in C

Eighth Edition
Global Edition

Jeri R. Hanly & Elliot B. Koffman

Chapter Objectives

- To learn about streams in C and their relationship to files and standard input and output devices
- To review how scanf, fscanf and printf, fprintf are used to read and write characters to text files
- To learn about escape sequences and their use in format strings

Chapter Objectives

- To review file pointer variables and learn how to use functions that process them to make a backup copy of a text file
- To learn about binary files and understand the differences between binary and text files

Input/Output Files

- text file
  - a named collection of characters saved in secondary storage

- input (output) stream
  - continuous stream of character codes representing textual input (or output) data

The keyboard and Screen as Text Streams

- stdin
  - system file pointer for keyboard’s input stream
  - pressing <return> or <enter> inserts a new line character in stream stdin
- stdout, stderr
  - system file pointers for screen’s normal and error output streams

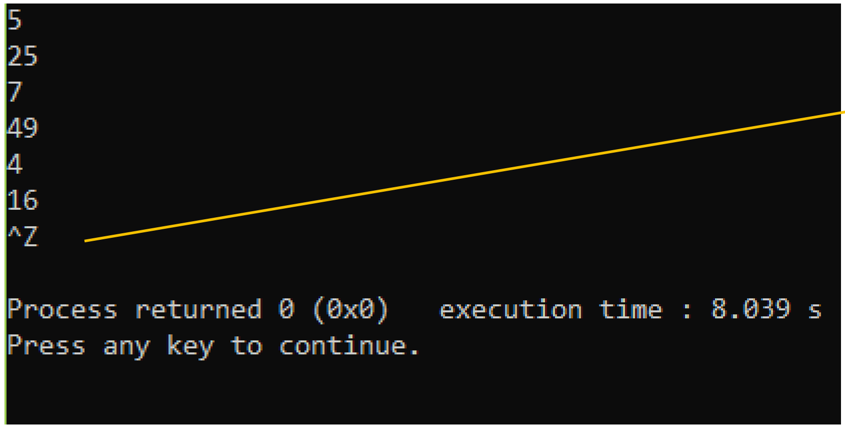
Newline and EOF

- newline: marks the end of a line of a text
  - processed like other characters
  - can be input using scanf and %c, can be compared to \n, can be output using printf
- eof: marks the end of the entire file
  - special return value
  - trying to input eof is a failed operation
  - is unequal to any valid character code.
  - associated with a negative value

EOF

- generally used for files but also used for console input:

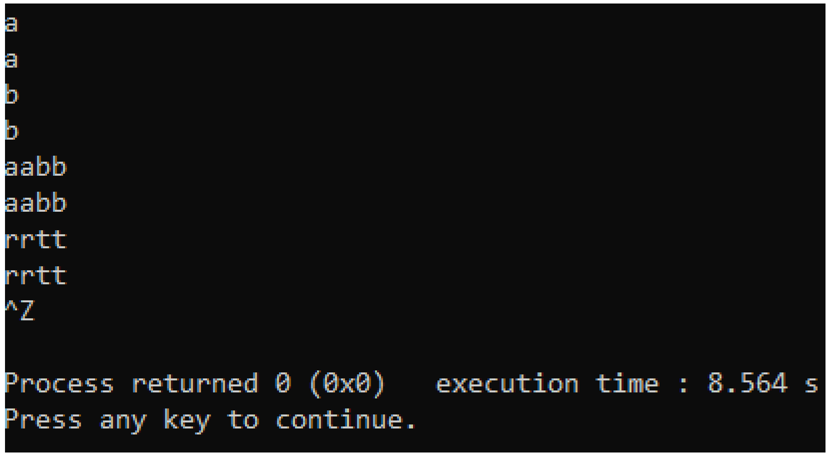
```
int num, status;
for(status = scanf("%d",&num); status != EOF; status = scanf("%d",&num))
    printf("%d\n",num*num);
```



```
5
25
7
49
4
16
^Z

Process returned 0 (0x0)   execution time : 8.039 s
Press any key to continue.
```

Loop continues until you enter CTRL+Z in Windows or CTRL+D in Unix

```
char c;
while ((c = getchar()) != EOF)
  putchar(c);
```

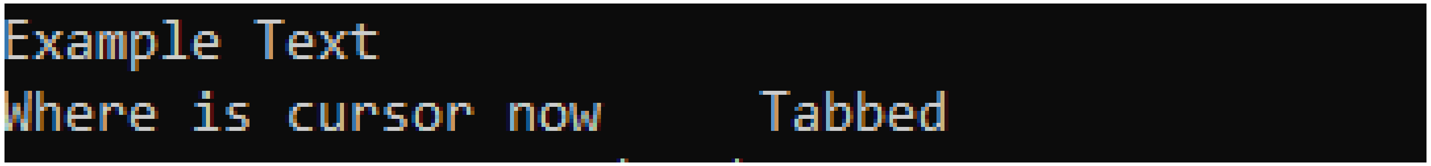


```
a
a
b
b
aabb
aabb
rrtt
rrtt
^Z

Process returned 0 (0x0)   execution time : 8.564 s
Press any key to continue.
```

Common Escape Sequences

Escape Sequence	Meaning
`'\n'`	new line
`'\t'`	tab
`'\r'`	return (go back to column 1 of current output line)
`'\b'`	backspace

```
printf("Example Text\nSecond Line\rWhere is cursor now\tTabbed");
```



```
Example Text
Where is cursor now     Tabbed
```

TABLE 11.2 Placeholders for printf Format Strings

Placeholder	Used for Output of	Example	Output

`%c`	a single character	`printf("%c%c%c\n",`
		`'a', '\n', 'b');`
			`a`
			`b`

`%s`	a string	`printf("%s%s\n",`
		`"Hi, how ",`
		`"are you?");`
			`Hi, how are you?`

`%d`	an integer (in base 10)	`printf("%d\n", 43);`	`43`
`%o`	an integer (in base 8)	`printf("%o\n", 43);`	`53`
`%x`	an integer (in base 16)	`printf("%x\n", 43);`	`2b`
`%f`	a floating-point number	`printf("%f\n", 81.97);`	`81.970000`
`%e`	a floating-point number in scientific notation	`printf("%e\n", 81.97);`	`8.197000e+01`
`%E`	a floating-point number in scientific notation	`printf("%E\n", 81.97);`	`8.197000E+01`
`%%`	a single `%` sign	`printf("%d%%\n", 10);`	`10%`

Reminder

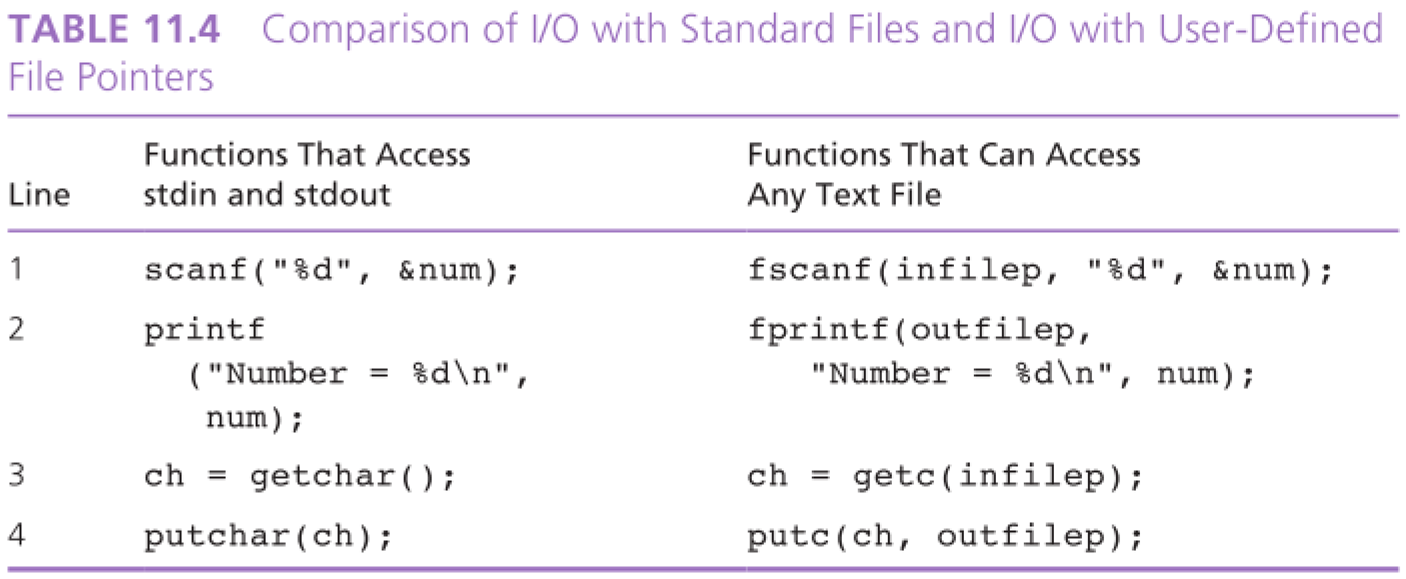


TABLE 11.4 Comparison of I/O with Standard Files and I/O with User-Defined File Pointers

Line	Functions That Access stdin and stdout	Functions That Can Access Any Text File
1	`scanf("%d", &num);`	`fscanf(infilep, "%d", &num);`

2	`printf`
	`("Number = %d\n",`
	`num);`
		`fprintf(outfilep,`
		`"Number = %d\n", num);`

3	`ch = getchar();`	`ch = getc(infilep);`
4	`putchar(ch);`	`putc(ch, outfilep);`

Reminder

- FILE *infilep, *outfilep;

  $\xrightarrow{\hspace{6cm}}$ Be careful, case sensitive

- infilep = fopen ("data.txt","r");
- if(infilep == NULL)
  - issue an error message
- outfilep = fopen ("dataout.txt","w");

- ……………………..

- fclose(infilep);
- fclose(outfilep);

Problems with Sequential-Access Files

- How to replace
  - «300 White 0.00»
    with
  - «300 Worthington 0.00»

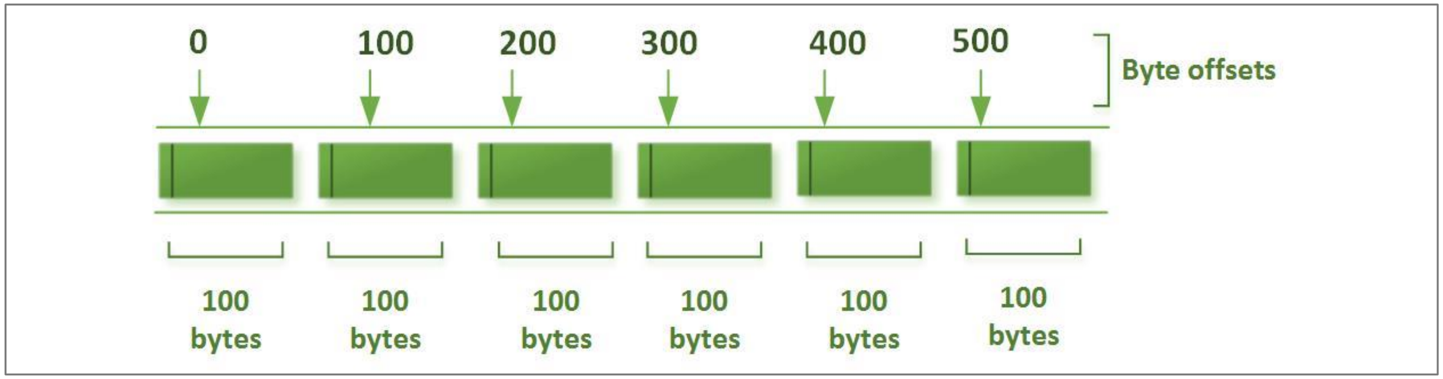
  - or «2000 4556» with «23 45»

  in a normal text file? Although both integers, fields are different!

- Formatted I/O model written using fprintf fscanf can vary in size. Therefore sequential access using fprintf fscanf is generally not used to update records in place.
- Risk of data corruption.
- Solution? Copy content to other file, update, copy back. OR?

Random Access Files

- Records are fixed in length



[3]

- In case of need for rapid access to specific data, e.g.
  - Airline reservation systems
  - Banking systems
  - Other transaction processing systems

Random Access Files

- Fixed-length records enable data to be inserted in a random-access file without destroying other data in the file.
- Data stored previously can also be updated or deleted without rewriting the entire file.
- You can jump instantly to any structure in the file, which provides random access as in an array.

Alternative File Format

- **fprintf**(fPtr, "%d", number);

  number can be a single digit or 10 digits+sign = 11 digits (max allowed: +2147483647 or 0 up to 4,294,967,295 (2^{32} - 1)) for a 4 byte integer (represented with 32 bits)

  In standard text format for each char we need 1 byte, so 11 bytes in total

Instead, use:

- **fwrite**(&number, sizeof(int), 1, fPtr);

  which always writes four bytes on a system with four-byte integers from a variable number

- Later, fread can be used to read those four bytes into an integer variable number.

fwrite

defined in <stdio.h>

- *size_t* **fwrite** *(const void *data, size_t size, size_t count, FILE *stream)*
- This function writes up to *count* objects of size *size* from the array *data*, to the stream *stream*.
- The return value is normally *count*, if the call succeeds. Any other value indicates some sort of error, such as running out of space.

fread

defined in <stdio.h>

- **size_t fread (void *data, size_t size, size_t count, FILE *stream)**

- This function reads up to *count* objects of size *size* into the array *data*, from the stream *stream*.
- It returns the number of objects actually read, which might be less than *count* if a read error occurs or the end of the file is reached.
- This function returns a value of zero (and doesn’t read anything) if either *size* or *count* is zero.

Alternative File Format

- Although fread and fwrite read and write data, such as integers, in fixed-size rather than variable-size format, the data they handle are processed in computer “raw data” format (i.e., bytes of data) rather than in printf’s and scanf’s human-readable text format.

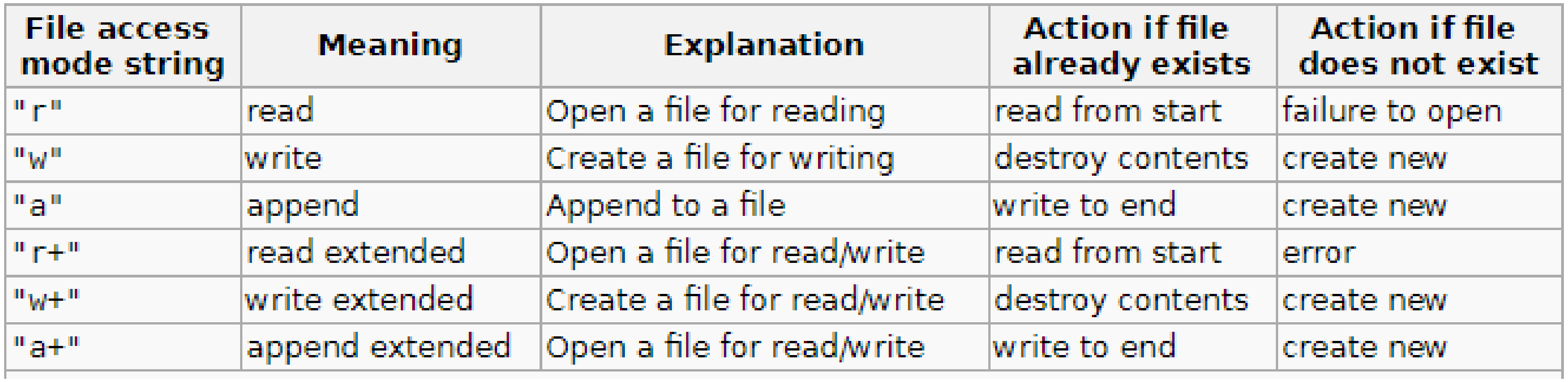
Binary Files

- When we use text files for storage, a significant amount of effort is used to convert the stream of characters into binary integers, mantissas, exponents, etc and convert them back.
- If there is no need for a human to read a file, why to convert files to binary and let the other program to convert it back?

- binary file
  - a file containing binary numbers that are the computer’s internal representation of each file component

- sizeof
  - operator that finds the number of bytes used for storage of a data type

fopen (revisitied from previous weeks)

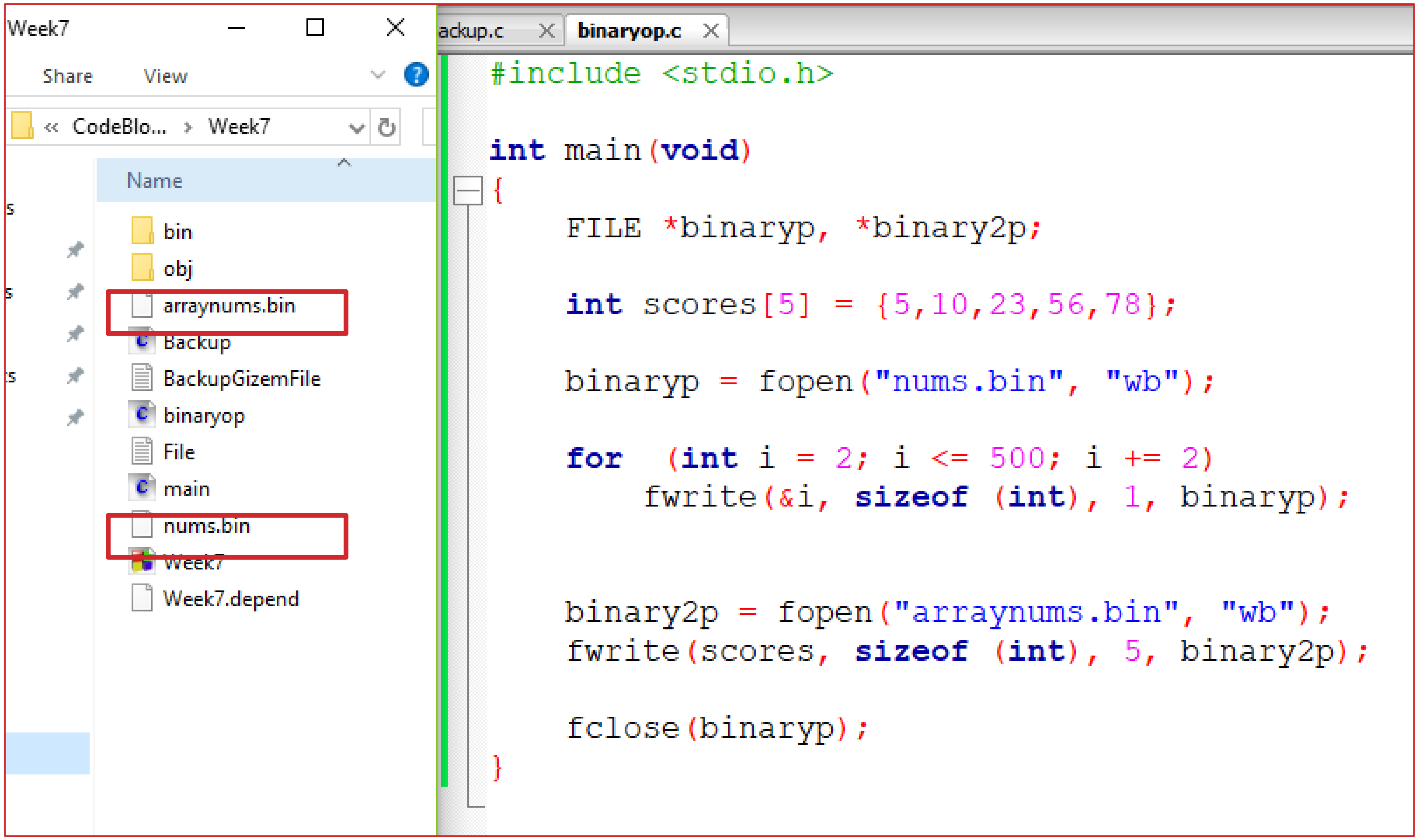


File access mode string	Meaning	Explanation	Action if file already exists	Action if file does not exist
"r"	read	Open a file for reading	read from start	failure to open
"w"	write	Create a file for writing	destroy contents	create new
"a"	append	Append to a file	write to end	create new
"r+"	read extended	Open a file for read/write	read from start	error
"w+"	write extended	Create a file for read/write	destroy contents	create new
"a+"	append extended	Open a file for read/write	write to end	create new

- File Access mode String additional character:

- ‘b’ has a standard meaning; it requests a binary stream rather than a text stream.
- If both ‘+’ and ‘b’ are specified, they can appear in either order.

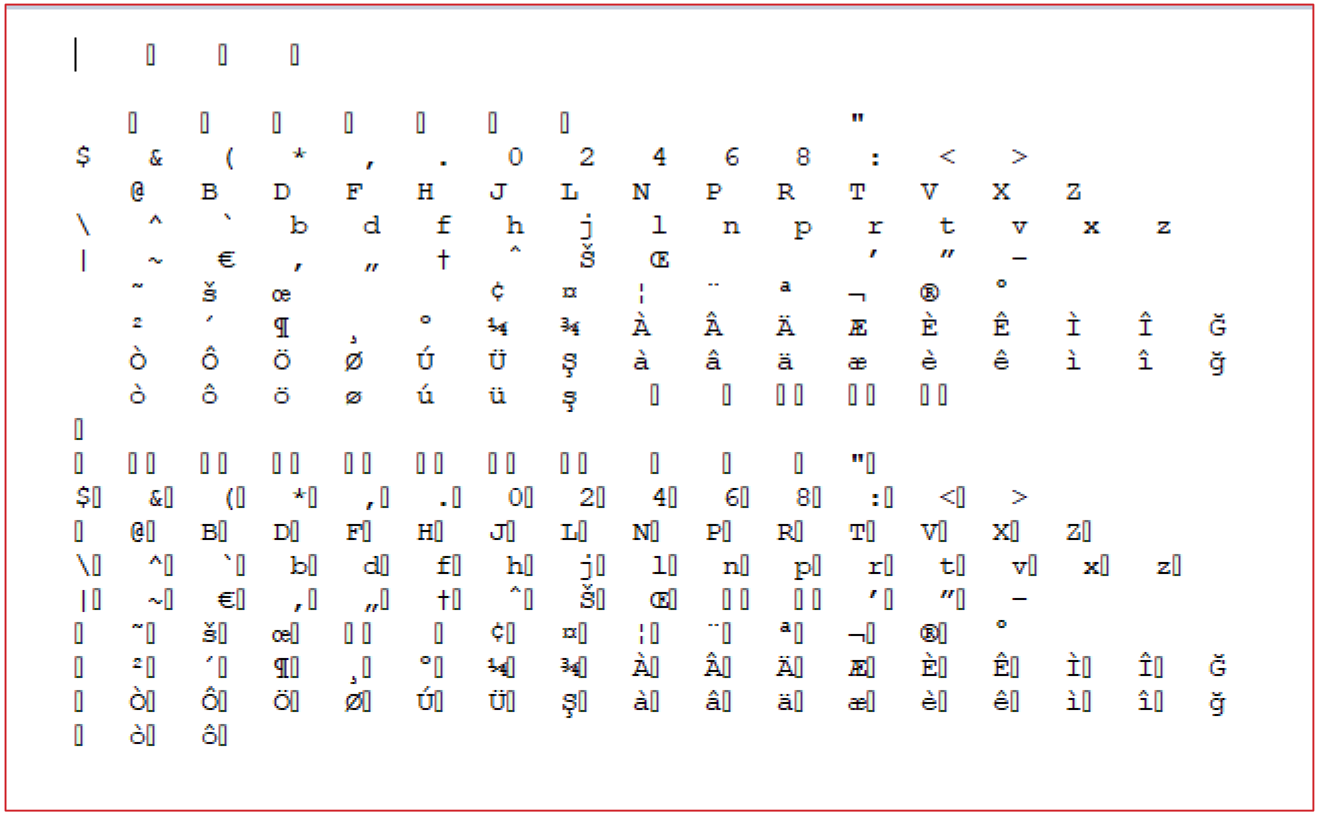
On some environments, binary & text files may not be treated the same,
Using appropriate mode is good practice.



```c
#include <stdio.h>

int main(void)
{
    FILE *binaryp, *binary2p;

    int scores[5] = {5,10,23,56,78};

    binaryp = fopen("nums.bin", "wb");

    for  (int i = 2; i <= 500; i += 2)
        fwrite(&i, sizeof (int), 1, binaryp);


    binary2p = fopen("arraynums.bin", "wb");
    fwrite(scores, sizeof (int), 5, binary2p);

    fclose(binaryp);
}
```

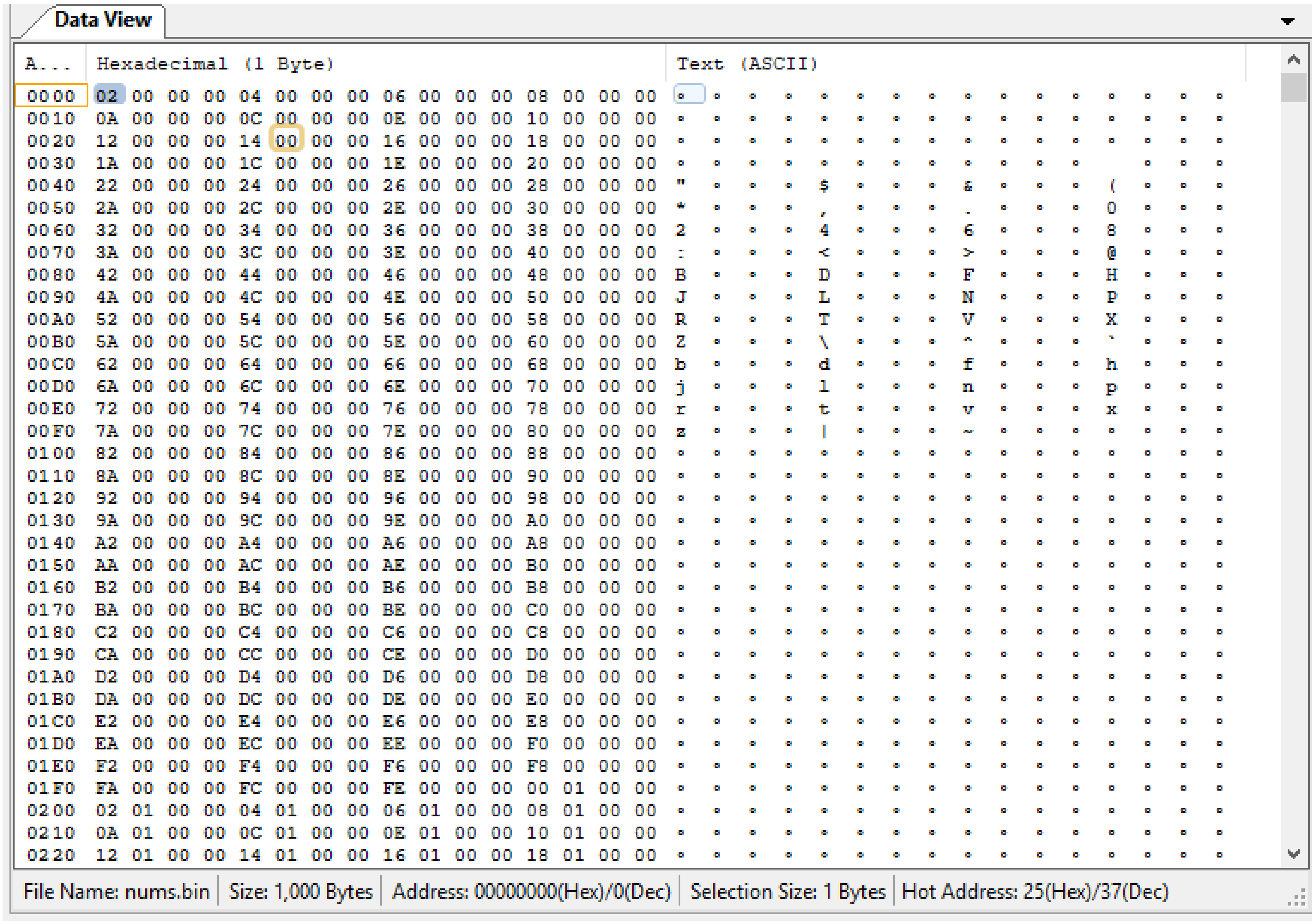
wb: write binary (rb: read binary)
&i: since the content of i is copied to file, address of operator is used
1 or 5: number of elements to write

nums.bin with a Classic Text Editor



```
|    ▯    ▯    ▯

     ▯    ▯    ▯    ▯    ▯    ▯    ▯                   "
$    &    (    *    ,    .    0    2    4    6    8    :    <    >
     @    B    D    F    H    J    L    N    P    R    T    V    X    Z
\    ^    `    b    d    f    h    j    l    n    p    r    t    v    x    z
|    ~    €    ‚    „    †    ˆ    Š    Œ                   ’    ”    —
     ˜    š    œ              ¢    ¤    ¦    ¨    ª    ¬    ®    °
     ²    ´    ¶    ¸    º    ¼    ¾    À    Â    Ä    Æ    È    Ê    Ì    Î    Ğ
     Ò    Ô    Ö    Ø    Ú    Ü    Ş    à    â    ä    æ    è    ê    ì    î    ğ
     ò    ô    ö    ø    ú    ü    ş    ▯    ▯    ▯ ▯  ▯ ▯  ▯ ▯
▯
▯    ▯ ▯  ▯ ▯  ▯ ▯  ▯ ▯  ▯ ▯  ▯ ▯  ▯ ▯  ▯    ▯    ▯    "▯
$▯   &▯   (▯   *▯   ,▯   .▯   0▯   2▯   4▯   6▯   8▯   :▯   <▯   >
▯    @▯   B▯   D▯   F▯   H▯   J▯   L▯   N▯   P▯   R▯   T▯   V▯   X▯   Z▯
\▯   ^▯   `▯   b▯   d▯   f▯   h▯   j▯   l▯   n▯   p▯   r▯   t▯   v▯   x▯   z▯
|▯   ~▯   €▯   ‚▯   „▯   †▯   ˆ▯   Š▯   Œ▯   ▯ ▯  ▯ ▯  ’▯   ”▯   —
▯    ˜▯   š▯   œ▯   ▯ ▯  ▯    ¢▯   ¤▯   ¦▯   ¨▯   ª▯   ¬▯   ®▯   °
▯    ²▯   ´▯   ¶▯   ¸▯   º▯   ¼▯   ¾▯   À▯   Â▯   Ä▯   Æ▯   È▯   Ê▯   Ì▯   Î▯   Ğ
▯    Ò▯   Ô▯   Ö▯   Ø▯   Ú▯   Ü▯   Ş▯   à▯   â▯   ä▯   æ▯   è▯   ê▯   ì▯   î▯   ğ
▯    ò▯   ô▯
```

nums.bin with an External Tool



Data View

```
A...  Hexadecimal (1 Byte)                             Text (ASCII)
0000  02 00 00 00 04 00 00 00 06 00 00 00 08 00 00 00  . . . . . . . . . . . . . . . .
0010  0A 00 00 00 0C 00 00 00 0E 00 00 00 10 00 00 00  . . . . . . . . . . . . . . . .
0020  12 00 00 00 14 00 00 00 16 00 00 00 18 00 00 00  . . . . . . . . . . . . . . . .
0030  1A 00 00 00 1C 00 00 00 1E 00 00 00 20 00 00 00  . . . . . . . . . . . .   . . .
0040  22 00 00 00 24 00 00 00 26 00 00 00 28 00 00 00  " . . . $ . . . & . . . ( . . .
0050  2A 00 00 00 2C 00 00 00 2E 00 00 00 30 00 00 00  * . . . , . . . . . . . 0 . . .
0060  32 00 00 00 34 00 00 00 36 00 00 00 38 00 00 00  2 . . . 4 . . . 6 . . . 8 . . .
0070  3A 00 00 00 3C 00 00 00 3E 00 00 00 40 00 00 00  : . . . < . . . > . . . @ . . .
0080  42 00 00 00 44 00 00 00 46 00 00 00 48 00 00 00  B . . . D . . . F . . . H . . .
0090  4A 00 00 00 4C 00 00 00 4E 00 00 00 50 00 00 00  J . . . L . . . N . . . P . . .
00A0  52 00 00 00 54 00 00 00 56 00 00 00 58 00 00 00  R . . . T . . . V . . . X . . .
00B0  5A 00 00 00 5C 00 00 00 5E 00 00 00 60 00 00 00  Z . . . \ . . . ^ . . . ` . . .
00C0  62 00 00 00 64 00 00 00 66 00 00 00 68 00 00 00  b . . . d . . . f . . . h . . .
00D0  6A 00 00 00 6C 00 00 00 6E 00 00 00 70 00 00 00  j . . . l . . . n . . . p . . .
00E0  72 00 00 00 74 00 00 00 76 00 00 00 78 00 00 00  r . . . t . . . v . . . x . . .
00F0  7A 00 00 00 7C 00 00 00 7E 00 00 00 80 00 00 00  z . . . | . . . ~ . . . . . . .
0100  82 00 00 00 84 00 00 00 86 00 00 00 88 00 00 00  . . . . . . . . . . . . . . . .
0110  8A 00 00 00 8C 00 00 00 8E 00 00 00 90 00 00 00  . . . . . . . . . . . . . . . .
0120  92 00 00 00 94 00 00 00 96 00 00 00 98 00 00 00  . . . . . . . . . . . . . . . .
0130  9A 00 00 00 9C 00 00 00 9E 00 00 00 A0 00 00 00  . . . . . . . . . . . . . . . .
0140  A2 00 00 00 A4 00 00 00 A6 00 00 00 A8 00 00 00  . . . . . . . . . . . . . . . .
0150  AA 00 00 00 AC 00 00 00 AE 00 00 00 B0 00 00 00  . . . . . . . . . . . . . . . .
0160  B2 00 00 00 B4 00 00 00 B6 00 00 00 B8 00 00 00  . . . . . . . . . . . . . . . .
0170  BA 00 00 00 BC 00 00 00 BE 00 00 00 C0 00 00 00  . . . . . . . . . . . . . . . .
0180  C2 00 00 00 C4 00 00 00 C6 00 00 00 C8 00 00 00  . . . . . . . . . . . . . . . .
0190  CA 00 00 00 CC 00 00 00 CE 00 00 00 D0 00 00 00  . . . . . . . . . . . . . . . .
01A0  D2 00 00 00 D4 00 00 00 D6 00 00 00 D8 00 00 00  . . . . . . . . . . . . . . . .
01B0  DA 00 00 00 DC 00 00 00 DE 00 00 00 E0 00 00 00  . . . . . . . . . . . . . . . .
01C0  E2 00 00 00 E4 00 00 00 E6 00 00 00 E8 00 00 00  . . . . . . . . . . . . . . . .
01D0  EA 00 00 00 EC 00 00 00 EE 00 00 00 F0 00 00 00  . . . . . . . . . . . . . . . .
01E0  F2 00 00 00 F4 00 00 00 F6 00 00 00 F8 00 00 00  . . . . . . . . . . . . . . . .
01F0  FA 00 00 00 FC 00 00 00 FE 00 00 00 00 01 00 00  . . . . . . . . . . . . . . . .
0200  02 01 00 00 04 01 00 00 06 01 00 00 08 01 00 00  . . . . . . . . . . . . . . . .
0210  0A 01 00 00 0C 01 00 00 0E 01 00 00 10 01 00 00  . . . . . . . . . . . . . . . .
0220  12 01 00 00 14 01 00 00 16 01 00 00 18 01 00 00  . . . . . . . . . . . . . . . .
```

File Name: nums.bin | Size: 1,000 Bytes | Address: 00000000(Hex)/0(Dec) | Selection Size: 1 Bytes | Hot Address: 25(Hex)/37(Dec)

TABLE 11.5 Data I/O Using Text and Binary Files

Example	Text File I/O	Binary File I/O	Purpose

1	`plan_txt_inp = fopen("planets.txt", "r");`
	
	`doub_txt_inp = fopen("nums.txt", "r");`
		`plan_bin_inp = fopen("planets.bin", "rb");`
		
		`doub_bin_inp = fopen("nums.bin", "rb");`
			Open for input a file of planets and a file of numbers, saving file pointers for use in calls to input functions.

2	`plan_txt_outp = fopen("pl_out.txt", "w");`
	
	`doub_txt_outp = fopen("nm_out.txt", "w");`
		`plan_bin_outp = fopen("pl_out.bin", "wb");`
		
		`doub_bin_outp = fopen("nm_out.bin", "wb");`
			Open for output a file of planets and a file of numbers, saving file pointers for use in calls to output functions.

3	`fscanf(plan_txt_inp, "%s%lf%d%lf%lf", a_planet.name, &a_planet.diameter, &a_planet.moons, &a_planet.orbit_time, &a_planet.rotation_time);`	`fread(&a_planet, sizeof (planet_t), 1, plan_bin_inp);`	Copy one planet structure into memory from the data file.
4	`fprintf(plan_txt_outp, "%s %e %d %e %e", a_planet.name, a_planet.diameter, a_planet.moons, a_planet.orbit_time, a_planet.rotation_time);`	`fwrite(&a_planet, sizeof (planet_t), 1, plan_bin_outp);`	Write one planet structure to the output file.

(continued)

TABLE 11.5 (continued)

Example	Text File I/O	Binary File I/O	Purpose

5	`for (i = 0; i < MAX; ++i)`
	`fscanf(doub_txt_inp,`
	`"%lf", &nums[i]);`
		`fread(nums, sizeof (double),`
		`MAX, doub_bin_inp);`
			Fill array `nums` with type `double` values from input file.

6	`for (i = 0; i < MAX; ++i)`
	`fprintf(doub_txt_outp,`
	`"%e\n", nums[i]);`
		`fwrite(nums, sizeof (double),`
		`MAX, doub_bin_outp);`
			Write contents of array `nums` to output file.

7	`n = 0;`
	`for (status =`
	`fscanf(doub_txt_inp,`
	`"%lf", &data);`
	`status != EOF &&`
	`n < MAX;`
	`status =`
	`fscanf(doub_txt_inp,`
	`"%lf", &data))`
	`nums[n++] = data;`
		`n = fread(nums,`
		`sizeof (double),`
		`MAX, doub_bin_inp);`
			Fill `nums` with data until `EOF` encountered, setting `n` to the number of values stored.

8	`fclose(plan_txt_inp);`
	`fclose(plan_txt_outp);`
	`fclose(doub_txt_inp);`
	`fclose(doub_txt_outp);`
		`fclose(plan_bin_inp);`
		`fclose(plan_bin_outp);`
		`fclose(doub_bin_inp);`
		`fclose(doub_bin_outp);`
			Close all input and output files.

fseek

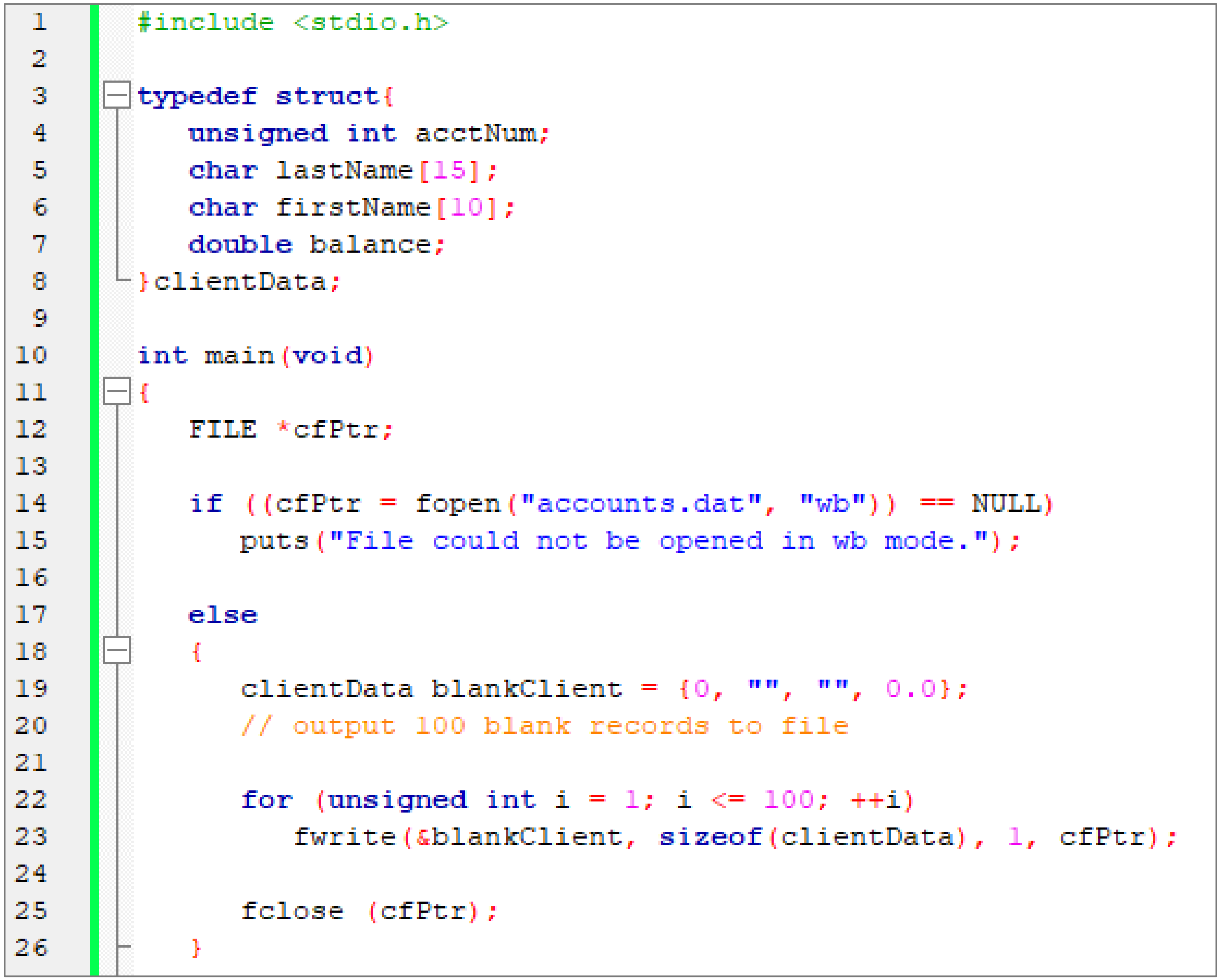
- int fseek(FILE *stream, long int offset, int whence);
- The fseek function is used to change the file position of the stream *stream*.
- The value of whence must be one of the constants SEEK_SET, SEEK_CUR, or SEEK_END, to indicate whether the offset is relative to the beginning of the file, the current file position, or the end of the file, respectively.
- The offset may be positive, meaning move forwards, or negative, meaning move backwards.
- This function returns a value of zero if the operation was successful, and a nonzero value to indicate failure.
- Can be used when both reading or writing a file.
- **fseek(fp,0L,0); ?**

Example

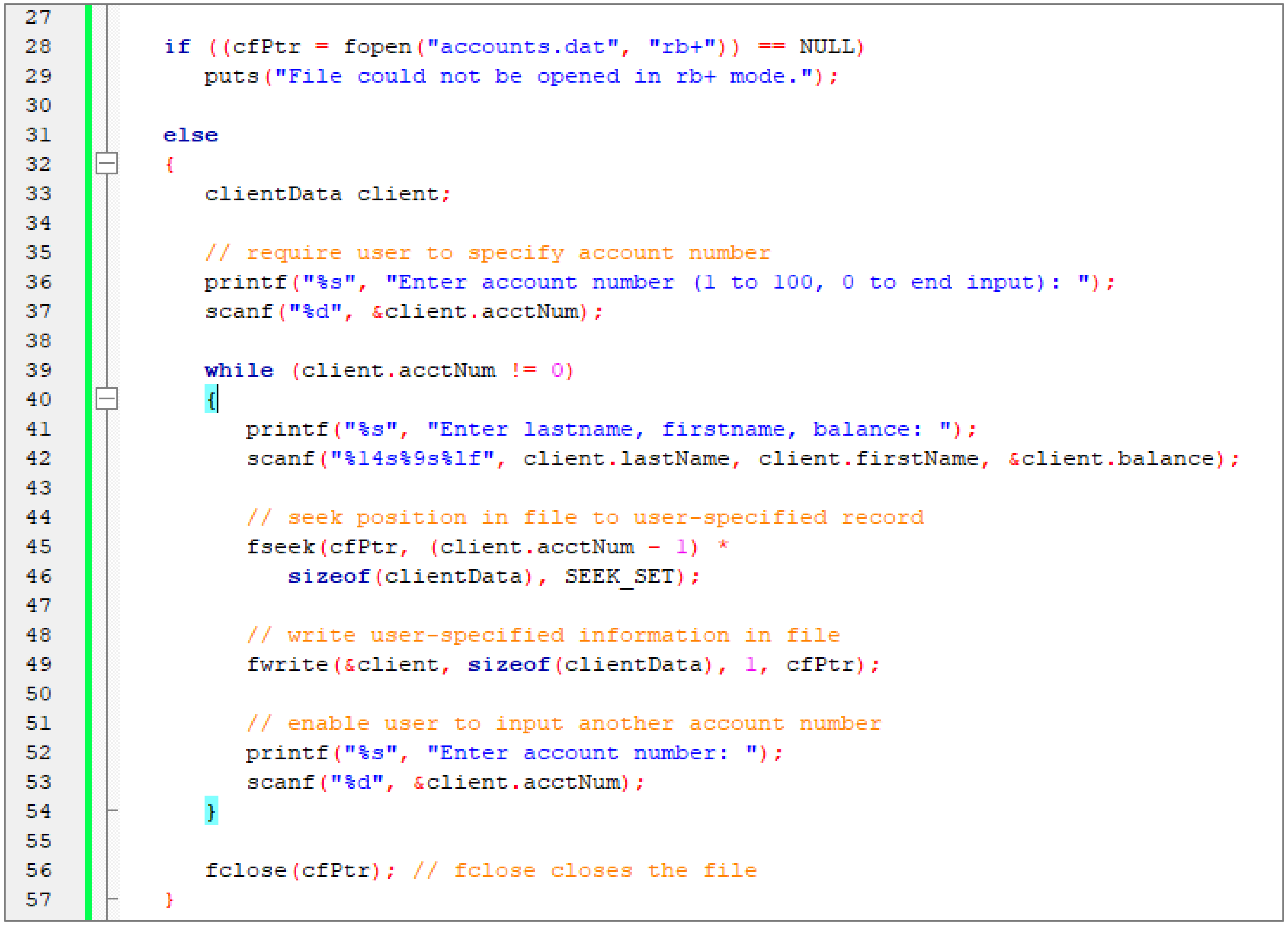
- The following example shows the usage of fseek() function.

```
#include <stdio.h>

int main () {
   FILE *fp;

   fp = fopen("file.txt","w+");
   fputs("This is tutorialspoint.com", fp);

   fseek( fp, 7, SEEK_SET );
   fputs(" C Programming Language", fp);
   fclose(fp);

   return(0);
}
```

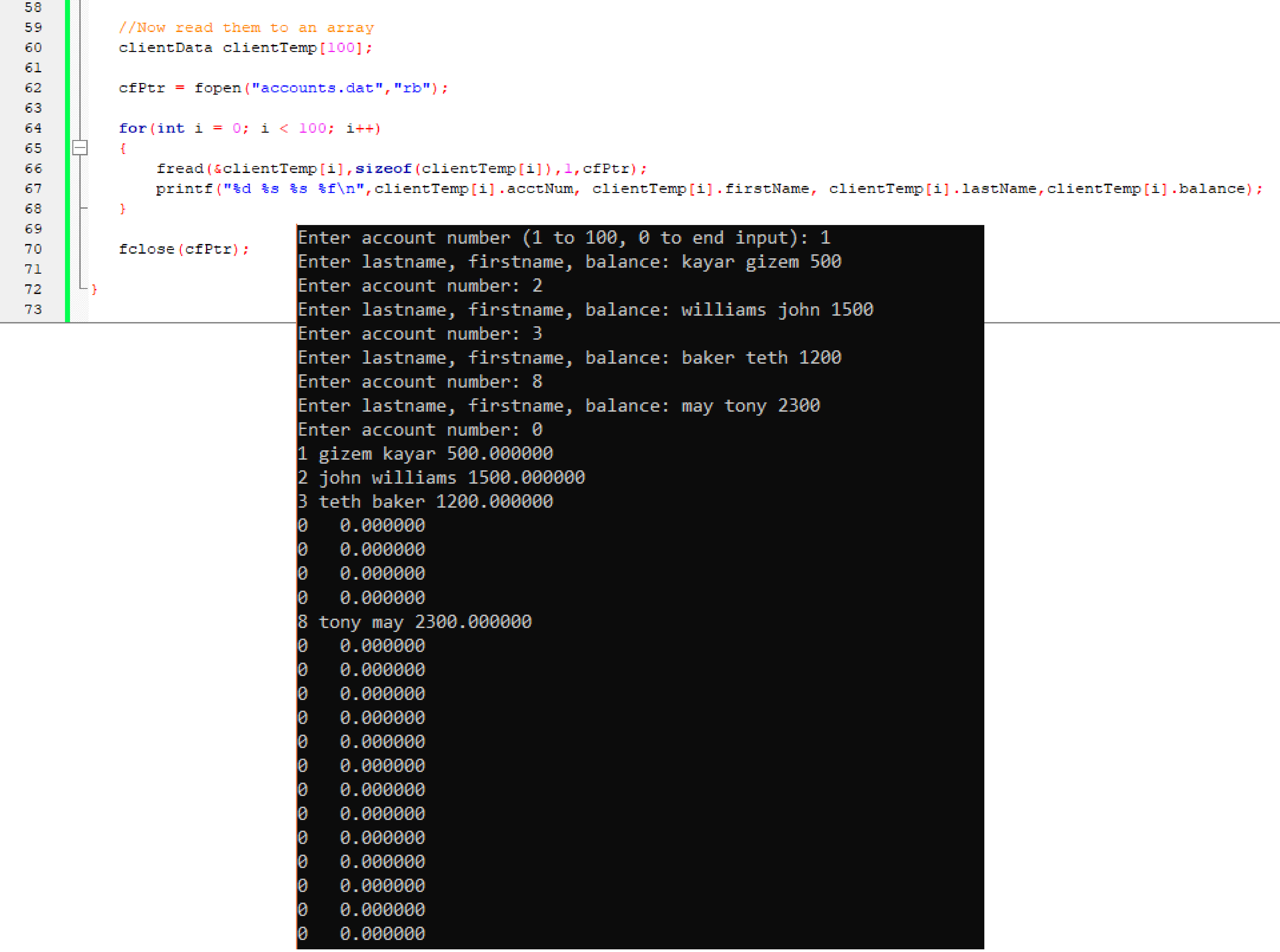
Output> `This is C Programming Language`



```c
#include <stdio.h>

typedef struct{
   unsigned int acctNum;
   char lastName[15];
   char firstName[10];
   double balance;
}clientData;

int main(void)
{
   FILE *cfPtr;

   if ((cfPtr = fopen("accounts.dat", "wb")) == NULL)
      puts("File could not be opened in wb mode.");

   else
   {
      clientData blankClient = {0, "", "", 0.0};
      // output 100 blank records to file

      for (unsigned int i = 1; i <= 100; ++i)
         fwrite(&blankClient, sizeof(clientData), 1, cfPtr);

      fclose (cfPtr);
   }
```



```c

   if ((cfPtr = fopen("accounts.dat", "rb+")) == NULL)
      puts("File could not be opened in rb+ mode.");

   else
   {
      clientData client;

      // require user to specify account number
      printf("%s", "Enter account number (1 to 100, 0 to end input): ");
      scanf("%d", &client.acctNum);

      while (client.acctNum != 0)
      {
         printf("%s", "Enter lastname, firstname, balance: ");
         scanf("%14s%9s%lf", client.lastName, client.firstName, &client.balance);

         // seek position in file to user-specified record
         fseek(cfPtr, (client.acctNum - 1) *
            sizeof(clientData), SEEK_SET);

         // write user-specified information in file
         fwrite(&client, sizeof(clientData), 1, cfPtr);

         // enable user to input another account number
         printf("%s", "Enter account number: ");
         scanf("%d", &client.acctNum);
      }

      fclose(cfPtr); // fclose closes the file
   }
```



```c

    //Now read them to an array
    clientData clientTemp[100];

    cfPtr = fopen("accounts.dat","rb");

    for(int i = 0; i < 100; i++)
    {
        fread(&clientTemp[i],sizeof(clientTemp[i]),1,cfPtr);
        printf("%d %s %s %f\n",clientTemp[i].acctNum, clientTemp[i].firstName, clientTemp[i].lastName,clientTemp[i].balance);
    }

    fclose(cfPtr);

}

```

```
Enter account number (1 to 100, 0 to end input): 1
Enter lastname, firstname, balance: kayar gizem 500
Enter account number: 2
Enter lastname, firstname, balance: williams john 1500
Enter account number: 3
Enter lastname, firstname, balance: baker teth 1200
Enter account number: 8
Enter lastname, firstname, balance: may tony 2300
Enter account number: 0
1 gizem kayar 500.000000
2 john williams 1500.000000
3 teth baker 1200.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
8 tony may 2300.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
0   0.000000
```

Wrap Up

- Text files are continuous streams of character codes that can be viewed as broken into lines by the newline character.
- Processing text files requires the transfer of sequences of characters between main memory and disk storage.
- Binary files permit storage of information using a computer’s internal data format.

References

1. Problem Solving & Program Design in C, Jeri R. Hanly & Elliot B. Koffman, Pearson 8. Edition, Global Edition
2. C How to Program, Paul Deitel, Harvey Deitel. Pearson 8th Edition, Global Edition.
3. http://www.infocodify.com/cprog/random_access_file