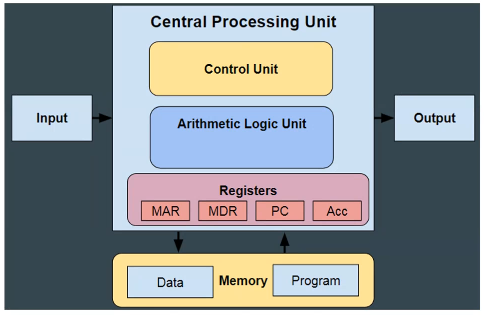
**FILES**

Von Neumann Architecture



Register number is usually 4 to 16.

Control unit basically does:

* loading sth to a register
* doing arithmetic & logic operations on the values in the registers
* storing register values in memory

Program in memory holds program where the instructions are stored usually in sequence with branching, loops, selections…

Data in memory is either generated by the program or used by the program generated by something else.

Memory is volatile, when off things will disappear.

For files, we need a file for input and permanent storage for output.

Any programming language we have should provide us some mechanisms. The way that input-output devices are managed is not actually in simple manner. These are mostly managed by another program, a program that is residing between programmer and hardware. That program is Opareting System.

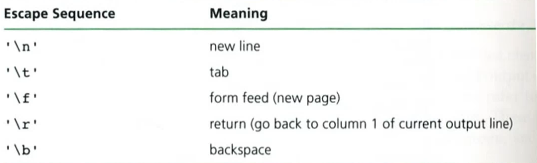
OS abstracts from us these details (where the file is, how it is organized, where is on permanent disc it is, how I can access that file in permanent disc, how can I share that between multiple programs…).

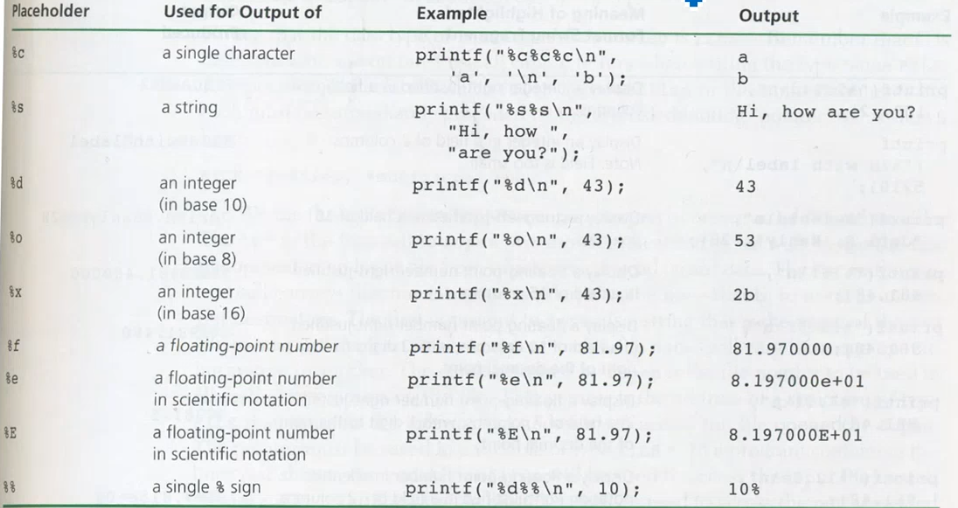
C allows us to talk with OS with very few library functions such that we can ask to OS to open a file, write to that file, close that file.

File Processing

* Files: used for permanent storage of information
* Two types of files (both used for storage of permanent data):
  + Text files
  + Binary files

1. Text file: collection of characters
   * Can be considered as stream of characters
     + Input stream (e.g., keyboard : stdin)
     + Output stresam (e.g., screen : stdout, stderr)
   * One of the abstraction mechanism to maintain a text file is stream. Just like stream of water. We read the data as it’s coming from stream, when we read it it’s done.
   * You don’t need to take the file as stream. File is already in the disc. You should be able to randomly access that file. But stream abstraction is kind of the most basic form of sequentially accessing your data. This is the way that things are implemented in C for files.
   * I can use same functions to read a file or to read from keyboard or read from serial port. This is the reason I treat these as such. You can treat files and keyboard and serial port same. You can call all of them streams.
   * I can access my file randomly. So files has some extra things that you can do. If input device is keyboard or inputting sth from the robot, it is doing its stuff you cannot go back. In file, you can go back. There are functions that allow you to go a certain amount back or certain amount forward so that you can simulate this possibility of randomly accessing any part of the file.
   * Can be created by using editors
     + Readable by human
   * Special characters
     + New line character
     + End of file character (EOF is returned when read)
     + Other escape sequences

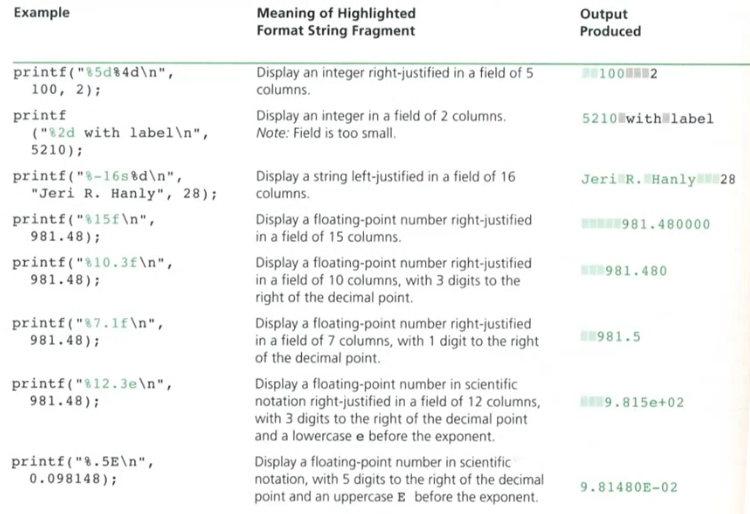




With printf, we are writing to an output stream which is equivalent to a text file.

For fprintf, it takes 3 arguments:

* output stream ≡ FILE \* (For printf, this is omitted. This is specific thing: stdout, same as FILE \*)
* Formatting string
* Argument



File Pointer

* Defined in “stdio.h”
* Allows to access a file

FILE \*fileptr;

fileptr = fopen(“filename”, “access mode”);

/\* filename includes the path, if not current directory where the program is running is taken \*/

if (fileptr == NULL)

printf(“File open error”);

else

... process file ...

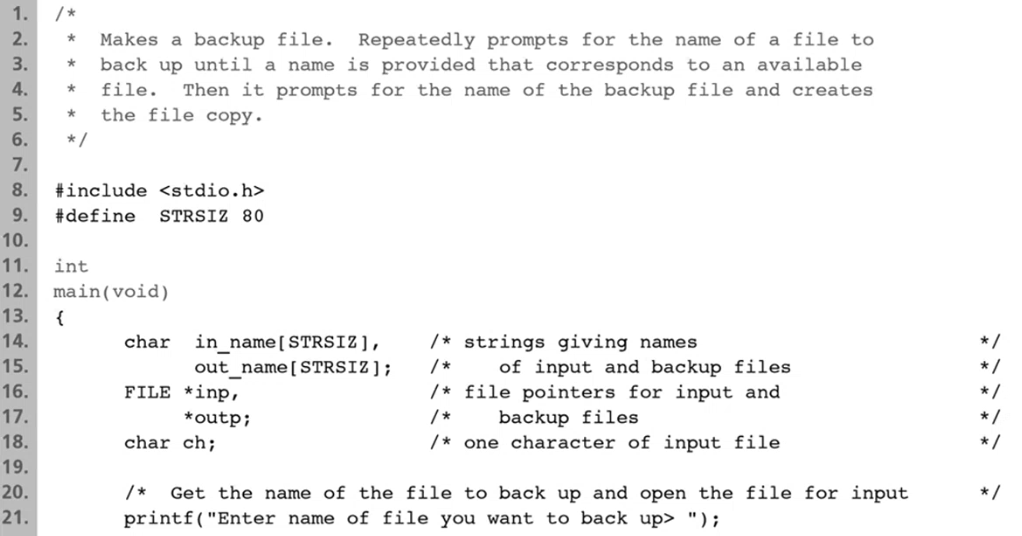
fclose(fileptr);

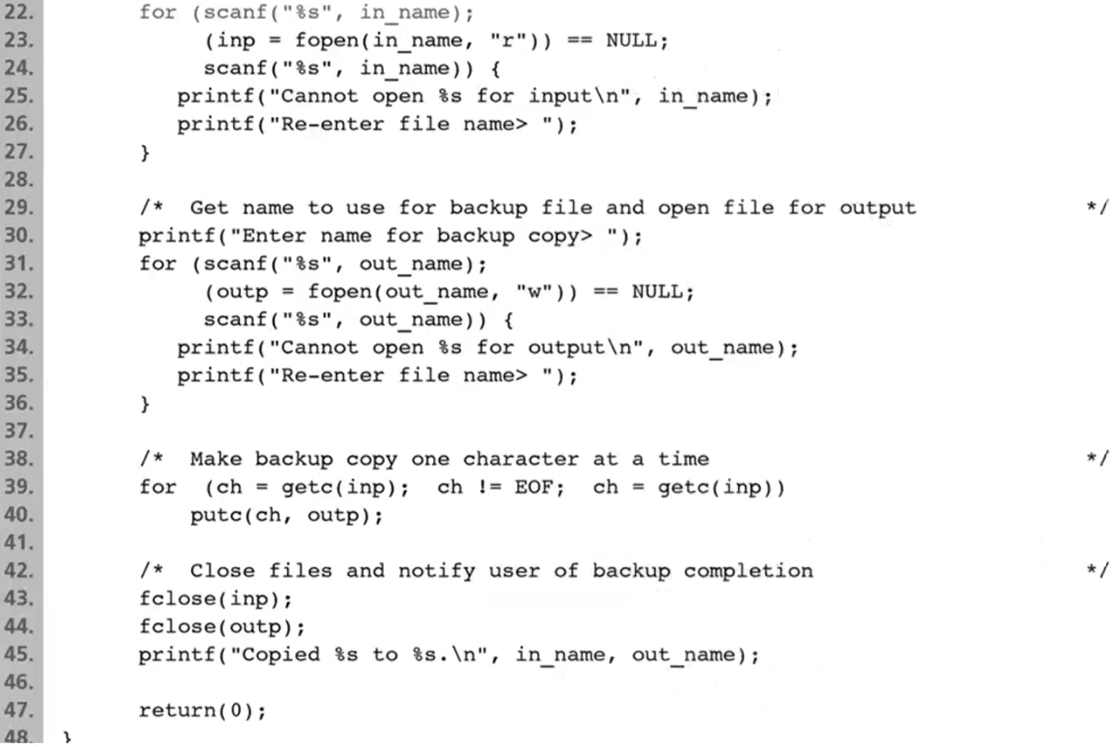
* Whatever I have after “FILE\* ” can hold the necessary information for C to manage a input or output stream. fileptr is gonna be stream, we will treat it as file. This will have either a text file or binary file. It will have other necessary information to manage an input or an output or both stream.
* Processing with getc, putc, fscanf and fprintf
  + Similar to fprintf, fscanf has 1 extre parameter which is output stream. In scanf, it is stdin (keyboard. it is treated as it is a file as well.).
  + When I put character to a file with putc, I put the character and then EOF goes one after when I close the file.
  + What if stdin or stdout is used as FILE \*
* access mode: you need to say type of the file (text or binary), and what are you gonna do with this file (inputting, outputting (create from stratch or append to an existing file).
  + wt ≡ w (write), rt ≡ r (read), at ≡ a (append)

Reasons for fileptr == NULL :

* File is not in the place you are indicating.
* Your file is there but that file is being modified by some other program at the time. OS will allow only 1 access at a time on each file.

COPYING A TEXT FILE





inp cannot be opened by 2 reasons we discussed in a page before.

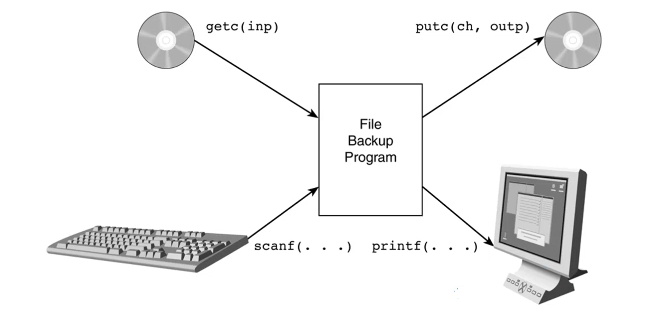
outp cannot be opened if the file exist and in use by a second program OR the user/program may not have write access to the directory where the file is supposed to be created

stdin: input (keyboard)

stdout: output (terminal) ≡ what are the printf is doing

stderr: output (used to log error messages) - different than stdout

INPUT OUTPUT STREAMS



1. Binary Files

* Binary files store the data in their internal representation
  + Note that text files stores the data as character sequence
    - Requires conversion between data types and stream of characters, this is not efficient
  + No conversion in binary files
    - Higher performance
    - Less storage
    - Higher precision for doubles
  + System dependent
    - Not portable
  + Not human readable

float v;

v = exp(1.002); 🡪 v is 4 bytes, many significant digits

fprintf(…, “%f”, v); 🡪 let’s say f is 100.00000001 which is 12 characters which is 12 bytes. Because I convert my value to a string

FILE \*fileptr;

fileptr = fopen(“filename”, “access mode”);

if (fileptr == NULL)

printf(“File open error”);

else

… process file …

fclose(fileptr);

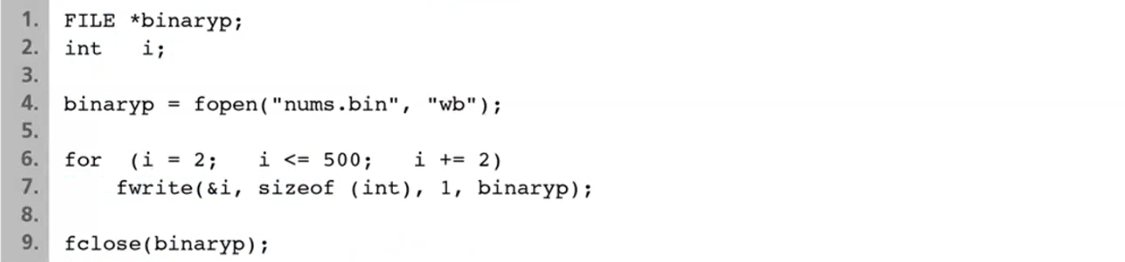
Access mode is “rb” or “wb”

* You can ignore t in access mode in text file but you cannot ignore b in access mode in binary file

Processing with fwrite or fread

* fwrite: first parameter is pointer to the value to be written to the file, second parameter is size of the individual entry, third parameter is number of entries that you have, last parameter is file pointer
* Ex: creating a binary file of integer

CREATING A BINARY FILE OF INTEGERS



fwrite and fread

fwrite(pointer, size\_of\_component, num\_of\_values, fileptr);

day\_t a[20];

fwrite(a, sizeof(day\_t), 20, bptr);

int fread(pointer, size\_of\_component, num\_of\_values, fileptr);

int a[20];

num = fread(a, sizeof(int), 20, bptr);

OR : fread(a, sizeof(int)\*20, 1, bptr);

Text File vs Binary File

Assume following declarations:

#define STRSIZ 10

#define MAX 40

typedef struct{

char name[20];

double diameter;

int moons;

double orbit\_time,

rotation\_time;

}planet\_t;

double nums[MAX], data;

planet\_t a\_planet;

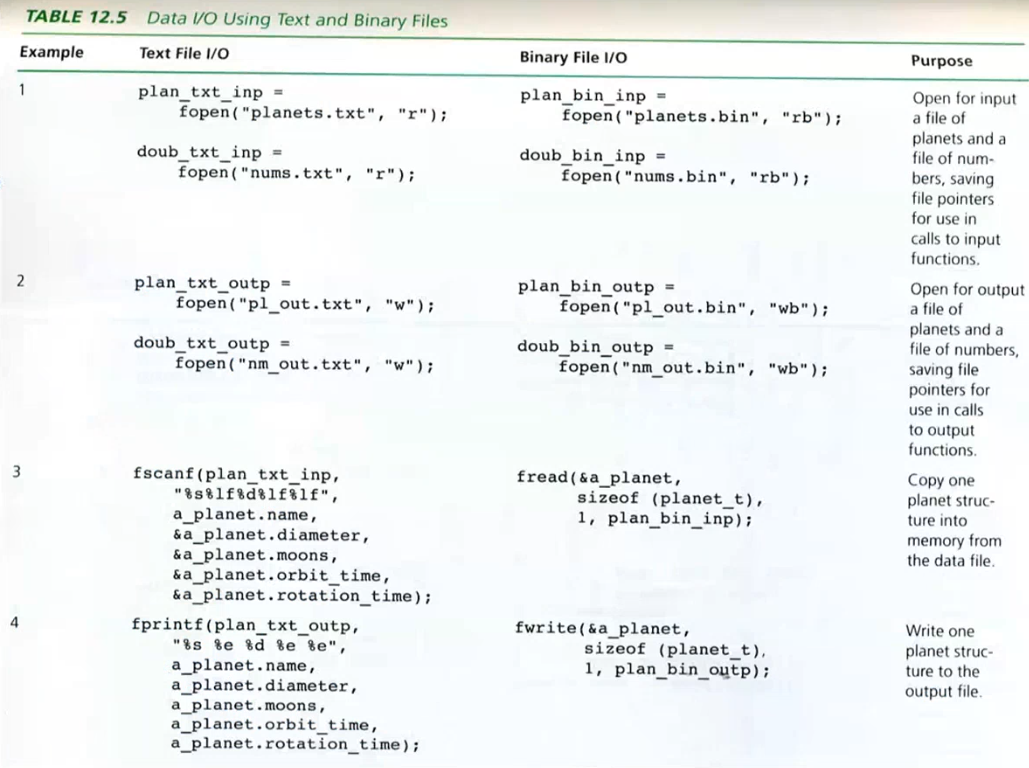
int i, n, status;

FILE \*plan\_bin\_inp, \*plan\_bin\_outp, \*plan\_txt\_inp, \*plan\_txt\_outp;

FILE \*doub\_bin\_inp, \*doub\_bin\_outp, \*doub\_txt\_inp, \*doub\_txt\_outp;

fwrite(&a\_planet, sizeof(planet\_t), 1, ...);

/\* I cannot do this one if I defined \*name and dynamically allocate memory for the struct \*/



Example

typedef struct node{

int k; 🡪 4 bytes

struct node \*next; 🡪 8 bytes (dynamic address)

}node;

node \*l; /\* We’ll put this list in a binary file \*/

void write(char \*fname, node \*l){ /\* dump list into file \*/

fwrite(l, sizeof(node), 1, fname);

/\* I cannot do this one because, I cannot store the memory because memory is just where the data is, I am not supposed to be storing the address to the memory but whatever the memory is containing. If I store this one, this is just a memory that is specific to that run of the program, next time address is gonna change and stored address is no longer gonna be meaningful to me. (there is address in struct (l->next))

Instead I can do this: \*/

while(l != NULL){

fwrite(&(l->k), sizeof(int), 1, fname);

l = l->next;

}

}

void read(...){

int k;

node \*l;

/\* You should know what you are reading \*/

while (fread(&k, sizeof(int), 1, ...))

insert(&l, k);

}

-----------------------------------------------------------------------------

int xi;

double yi;

for(i=0;i<10;i++){

fwrite(&xi, sizeof(int), 1, ...);

fwrite(&yi, sizeof(double,1, ...);

}

in binary file order is : int-double-int-double-int-double-...

So when you read from that file, you should read in the correct order, else you read garbage values. For example if you read double first, then you take int and 4 bytes of the double in the file. It has no meaning.