IC150
Computational Engineering

Lecture 24
File Input and Output

Timothy Gonsalves

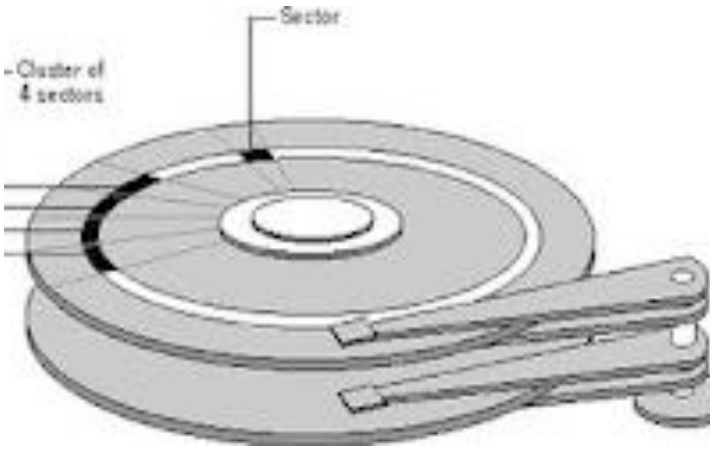
Homework Problems on Arrays and Strings

1. Given a square array of integers – produce an array rotated clockwise by 90^o.
2. Convert a string of characters in the range '0' to '9' to the corresponding integer
   - look for minus sign preceding the number
   - e.g. " -986542is " → the number -986542
   - " 768 " → 768
3. Given an integer produce the string of characters denoting that number
   - e.g 926 → '9', '2', '6'
4. Given an array of characters extract a double number
   - look for minus sign preceding the number
   - e.g " -63.65" or " 76.56"

A Hard Disk Drive

A Hard Disk Drive

A Hard Disk Drive



Input/Output in C

- C has no built-in statements for input or output
- A library of functions is supplied to perform these operations. The I/O library functions are listed the "header" file `<stdio.h>`
- You do not need to memorize them, just be familiar with them

Streams

- All input and output is performed with streams
- A stream is an ordered sequence of bytes
- In a stream of text, each byte represents a character
  - Organized into lines
  - Each line consists of zero or more characters and ends with the "newline" (`'\n'`) character
  - ANSI C standard specifies that the system must support lines that are at least 254 characters in length (including the newline character)

Types of Streams in C

Every C program has 3 standard streams:

- Standard input stream, `stdin` -- normally connected to the keyboard
- Standard output stream, `stdout` -- normally connected to the display screen.
- Standard error stream, `stderr` -- also normally connected to the screen

Standard Streams in C

- Input functions normally read from stdin
  - scanf(), gets(), getchar()
- Output functions normally write to stdout
  - printf(), putchar()

I/O Redirection: connect stdin or stdout to a file instead of keyboard or display

Type command: `myprog`

- scanf reads from keyboard, printf writes to display

Type command with file names:

```
myprog <input.dat >output.dat
```

- scanf reads from input.dat, printf writes to output.dat

File access

- Files need to be connected to the program
  - the system connects stdin, stdout and stderr
- Reading from or writing to a file in C requires 3 basic steps:

1. Open the file
2. Do all the reading or writing
3. Close the file

- Internally a file is referred to using a *file pointer*
  - points to a *structure* that contains info about the file

Opening a file

- Declare a file pointer and open a file using the function `fopen()`
- `FILE *fp;` `//` FILE is a typename, like int

```
fp = fopen(fileName, mode);
```

name of file

what is the file going to be used for?

Basic modes for opening files

- `r'
  - Open an existing file for reading only.
- `w'
  - Open the file for writing only. If the file already exists, it is truncated to zero length. Otherwise a new file is created.
- `a'
  - Open a file for append access; that is, writing at the end of file only. If the file already exists, its initial contents are unchanged and output to the stream is appended to the end of the file. Otherwise, a new, empty file is created.

More file modes

- `r+'
  - Open an existing file for both reading and writing. The initial contents of the file are unchanged and the initial file position is at the beginning of the file
- `w+'
  - Open a file for both reading and writing. If the file already exists, it is truncated to zero length. Otherwise, a new file is created
- `a+'
  - Open or create file for both reading and appending. If the file exists, its initial contents are unchanged. Otherwise, a new file is created. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file

An example

```
FILE *ifp, *ofp; char *mode = "r";
char inFilename[] = “in.list”;
char outFilename[] = "out.list";
ifp = fopen(inFilename, mode);
if (ifp == NULL) {
    fprintf(stderr, "Can't open input file %s\n”,
                                        inFilename);
    exit(1);
  }
ofp = fopen(outFilename, "w");
if (ofp == NULL) {
    fprintf(stderr, "Can't open output file %s\n",
                                        outFilename);
    exit(1);
  }
```

fopen returns NULL if it cannot open a file

File Input/Output in C

getc (*file) ;

- This function is similar to getchar( ) except the input can be from the keyboard or *a file*.
- Example:

```
char ch;
ch = getc(stdin) ;    /* input from keyboard */
ch = getc(fileptr) ;  /* input from a file */
```


… File Input/Output in C

putc (char, *file) ;

- This function is similar to putchar ( ) except the output can be to the screen or a file.
- Example:

```
char ch;
ch = getc(stdin) ; /* input from keyboard */
putc(ch, stdout) ;  /* output to the screen */
putc(ch, outfileptr) ; /*output to a file */
```

Formatted Reading and Writing

fscanf(*filepointer*, "…", *args*)

fprintf(*filepointer*, "…", *args*)

- Format string and arguments same as with scanf() and printf()

Closing a file

- When done with a file, it must be closed using the function fclose()

```
fclose(ifp); fclose(ofp);
```

- Closing a file is very important, especially with output files. The reason is that output is often *buffered*. This means that when you tell C to write something out, it doesn't necessary get written to disk right away, but may be stored in a *buffer* in memory.
  - This output buffer holds the text temporarily
  - When the buffer fills up (or when the file is *closed*), the data is finally written to disk

Force write of file buffer to disk

- Sometimes it is necessary to forcefully flush a buffer to its stream:

```
fflush(outf);
```

- Likewise, to clear typeahead from the input buffer:

```
fpurge(inf);
```


Reading from a file using fgets

- fgets is a better way to read from a file
- We can read into a string using fgets

```
FILE *fptr;
char line [1000];
… /* Open file and check it is open */
while (fgets(line,1000, fptr) != NULL) {
   printf ("Read line %s\n", line);
}
```

fgets() takes 3 arguments, a string, maximum number of characters to read and a file pointer. It returns NULL if there is an error (such as EOF)

Using fgets to read from the keyboard

- fgets and stdin can be combined for a safe way to get a line of input from the user

```
#include <stdio.h>
int main()
{
    const int MAXLEN=1000;
    char readline[MAXLEN];
     …
    fgets (readline, MAXLEN, stdin);
    printf ("You typed %s", readline);
    return 0;
}
```


fgets

- One of the alternatives to scanf/fscanf is **fgets** The prototype is:
  - char *fgets(char *s, int size, FILE *stream);
  - fgets reads in size-1 characters from the stream and stores it into *s string. The string is automatically null-terminated.
- fgets stops reading in characters if it reaches an EOF or newline.
- The string can be scanned using sscanf()

IC150
Computational Engineering

Lecture 25
Command-line Arguments

Timothy Gonsalves

Command line arguments

- parameters can be passed to C programs by giving arguments when the program is executed
- examples

```
$ echo hello, world
```

prints the output

```
hello, world
```

```
$ cat file1 file2
```

prints *contents of* file1 followed by file2

cat is short for con*cat*enate

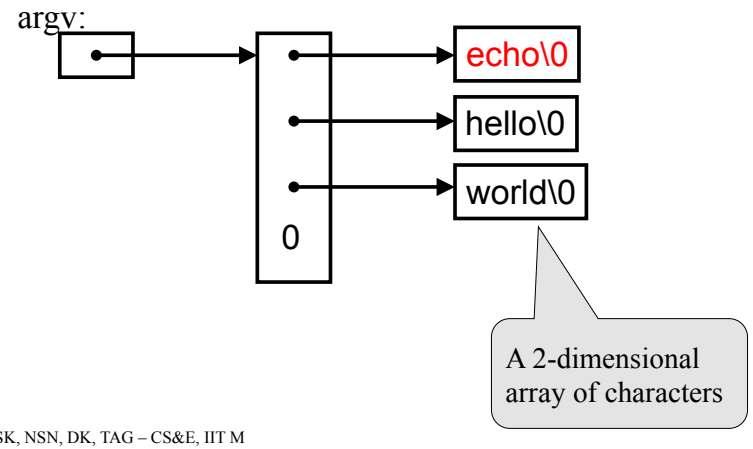
Note: '$' is the prompt. Your terminal may use some other prompt such as '>'

main(int argc, char *argv[])

- The main program in C is called with two implicit arguments *argc* and *argv*
- argc is an integer value for the number of arguments in the command line
  - if none then argc =1 (the program name)
- argv is an array of strings, passed by reference
  - argv[0] is the name of the program
  - argv[1] is the first command-line argument
  - argv[2] is the next argument, and so on …
  - argv[argc] is a NULL pointer

$ echo hello world

argc = 3, argv is a pointer to an array of pointers



Implementing echo

```
#include <stdio.h>
// echo comand line arguments: array version
int main(int argc, char *argv[])
{
  int i;
  for (i = 1; i < argc; i++)
    printf("%s%s,argv[i],(i<argc-1)?" ":"");
  printf("\n");
  return 0;
}
```

expr1 ? expr2 : expr3 == if(expr1) expr2 else expr3

echo – pointer version

```
#include <stdio.h>
// echo comand line arguments: pointer version
main(int argc, char *argv[])
{
  while (--argc > 0)
    printf("%s%s, *++argv, (argc>1)?" ":"");
  printf("\n");
  return 0;
}
```

printf((argc > 1) ? "%s " : "%s", *++argv);

cat – reads files and prints them

```
#include <stdio.h>
main(int argc, char *argv[]) {
FILE *fp;
void filecopy(FILE *, FILE *)
if (argc == 1)   /* no args; copy from stdin */
   filecopy(stdin, stdout);
else
    /* open the first file, copy it onto screen, close it,
       go to next file …. */
```

rename a.out cat
mv a.out cat

cat - continued

```
else
   while (--argc > 0)
       if ((fp = fopen(*++argv, "r") == NULL) {
            printf("cat: can't open %s\n", *argv);
            return 1;
          } else {
            filecopy(fp, stdout);
            fclose(fp);
          }
return 0;
}          /* end of main */
```

modify program to direct error messages to stderr, so that redirection does not affect it.

$ cat f1 f2 >outfile

copying a file

```
// filecopy: copy file infp to outfp
void filecopy(FILE *infp, FILE *outfp)
{
  int c;
  while ((c = fgetc(infp)) != EOF)
     fputc(c, outfp);
}
```

copy everything -- blanks, tabs, newline -- until the end of the file

program name in error message

```
   .
   .
   .
char *progName = argv[0];
…
if ((fp = fopen(*++argv, "r")) == NULL)
     fprint(stderr,
          "%s: can't open %s\n",
          progName, *argv);
   .
   .
   .
```

Homework problem set 2

1. Given two files file1 and file2 of integers, compute the average of all the numbers, and append file2 at the end of file1
2. Find the middle character of a string without calculating the length of the string. (Use pointers)
3. Reverse the *word order* of a sentence using pointers
   Example: Chennai is capital of Tamil Nadu
   → Nadu Tamil of capital is Chennai
4. Given a string s1 and a string s2, write a program to say whether s2 is a rotation of s1?
   Example : Tamilnadu aduTamiln → True
   Tamilnadu dunaTamil → False
   "abracadabra" "cadabraabra" ?

A problem of managing indices

- Fill in a rectangular array with increasing numbers in a spiral form, as shown for the square array

1 →	2 →	3 →	4 →	5
16 →	17 →	18 →	19 ↓	6
15 ↑	24 →	25 ↓	20 ↓	7
14 ↑	23 ↓	22 ←	21 ←	8
13 ←	12 ←	11 ←	10 ←	9