CS1100
Introduction to Programming

File I/O

Input/Output in C -- Recap

- 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 in the “header” file <stdio.h>
- You do not need to memorize them, just be familiar with them
- Programs using the library - portable

Streams

- All input and output is performed with *streams*
- A “stream” is a sequence of characters 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 is called *stdin* and is normally connected to the keyboard
- Standard output stream is called *stdout* and is normally connected to the display screen
- Standard error stream is called *stderr* and is also normally connected to the screen

Standard Streams in C

- Input functions normally read from *stdin*
  - ***scanf***( ), ***getline***( ), ***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:
  - open the file
  - do all the reading and/or writing
  - 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 type name, like ***int*** */
- Prototype: fopen(char *name, char *mode)

- fp = fopen(name, 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.
  - **initial** file position for reading is at **beginning** of the file
  - **output** is always appended to the **end** of the file.

Formatted Reading and Writing

- ***fscanf***(filepointer, "…", args);
- ***fprintf***(filepointer, "…", args);
- fscanf(stdin, …., args) will behave like scanf
- fprintf(stdout, …., args) will behave like printf
- fprintf(stderr, …., args) will send mesg to stderr
- Format string and arguments same as with ***scanf***( ) and ***printf***( )
- ***fgetc(), fgets() and getline()*** can also be used
- Advanced: sscanf after fgets/getline

An Example

```
FILE *ifp, *ofp; char *mode = "r"; int a; char s[50];
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);
    }
fscanf(ifp, “%d %s”, &a, s);
ofp = fopen(outFilename, "w");
if (ofp == NULL) {
    fprintf(stderr, "Can't open output file %s!\n", outFilename);
    exit(1);
    }
fprintf(ofp, “%d %s”, a, s); fclose(ifp); fclose(ofp);
```

fopen returns NULL if it cannot open a file

File Access Functions

Function	Usage	Remarks
fscanf(FILE *fp, Format String, args)	Read user-specified values from the file based on format string	*stdin* can be used as a file pointer, to read from keyboard
fprintf(FILE *fp, Format String, args)	Print user-specified values to the file based on format string	*stdout or stderr* can be used as a file pointer, to print to screen
char ch = **fgetc**(fp);	Read a character from file	stdin is valid value for fp
fputc(char, fp);	Print a character to file	stdout/stderr is valid value for fp
fgets(char *s, int size, fp)	Reads (size-1) chars. From file into string s; s will be NULL-terminated; Check return value (NULL if there is reading error)	stdin is valid value for fp; can be combined with sscanf
fputs(char *s, fp)	Prints a string to the file	stdout/stderr is valid value for fp

File Input/Output in C

- ***char fgetc***(FILE *fp);
  - This function is similar to ***getchar***( ) except that input can be from keyboard or a file.
- Example:
  - ***char*** ch;
  - ch = ***fgetc***(***stdin***);   /* input from keyboard */
  - ch = ***fgetc***(fileptr);   /* input from a file */
- getc( ) is a macro that expands to fgetc()

… File Input/Output in C

- ***fputc***(***char***, FILE *fp);
  - This function is similar to ***putchar***( ) except that the output can be to the screen or a file.
- Example:
  - ***char*** ch;
  - ch = ***fgetc***(***stdin***); /* input from keyboard */
  - ***fputc***(ch, ***stdout***); /* output to the screen */
  - ***fputc***(ch, outfileptr); /*output to a file */

Attendance Question for Oct. 23

What is the purpose of the following program?

```
int main (int argc, char *argv[]) {
   FILE *ifp, *ofp; char ch;

   ifp = fopen(argv[1], “r”);
   ofp = fopen(argv[2], "w");

   while ((ch = fgetc(ifp)) != EOF)
        fputc(ch, ofp);
   fclose(ifp); fclose(ofp);
}
```

Closing a File

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

  fclose(ifp); ***fclose***(ofp);

- Closing a file is 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, in the middle of a program

fprintf(outf,"%d %s", i, s);

fflush(outf); // Forces output to be written to file

The Function *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* pointer
  - The string is automatically null-terminated
  - Returns *s* or NULL if there is an error
- ***fgets*** stops reading in characters if it reaches an EOF or NULL
- The string can be scanned using ***sscanf***( )

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);
}
```

Recall that ***fgets*** takes 3 arguments, a string, the 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 to get 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;
}
```

Creating a File with Keyboard input

```
// Creating a sequential file  -- from Deitel and Deitel
#include <stdio.h>

int main (int argc, char *argv[])
{
  FILE *cfPtr; // cfPtr = clients.txt file pointer

  // fopen opens file. Exit program if unable to create file
  if ((cfPtr = fopen("clients.txt", "w")) == NULL) {
    puts("File could not be opened");
  }
  else {
    printf("Enter the account, name, and balance.");
    printf("Enter EOF to end input.");
    printf("%s", "? ");
```

```
      unsigned int account; // account number
      char name[30]; // account name
      double balance; // account balance

      fscanf(stdin, "%d%29s%lf", &account, name, &balance);

      // write account, name and balance into file with fprintf
      while (!feof(stdin)) {
        fprintf(cfPtr, "%d %s %.2f\n", account, name, balance);
        printf("%s", "? ");
        fscanf(stdin, "%d%29s%lf", &account, name, &balance);
      }

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

Reading from a file

```
// Fig. 11.6: fig11_06.c   - Deitel & Deitel
// Reading and printing a sequential file
#include <stdio.h>

int main(void)
{
  FILE *cfPtr; // cfPtr = clients.txt file pointer

  // fopen opens file; exits program if file cannot be
opened
  if ((cfPtr = fopen("clients.txt", "r")) == NULL) {
    puts("File could not be opened");
  }
  else { // read account, name and balance from file
    unsigned int account; // account number
    char name[30]; // account name
    double balance; // account balance

    printf("%-10s%-13s%s\n", "Account", "Name",
"Balance");
    fscanf(cfPtr, "%d%29s%lf", &account, name,
&balance);

    // while not end of file
  while (!feof(cfPtr)) {
      printf("%-10d%-13s%7.2f\n", account, name,
balance);
      fscanf(cfPtr, "%d%29s%lf", &account, name,
&balance);
    }

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

Quiz 2 Statistics etc

Random Access Files – Advanced and Optional

- File contains a set of fixed-length records
- Each record contains different fields and values
  - Record corresponds to a C struct
- Assume that there are 100 records in a file
  - Possible to read or write any record (say 12th)
- Requires other functions
  - fread()
  - fwrite()
  - fseek() is used to position read/write pointer to specified byte position in the file

MULTI-FILE COMPILATION

Writing program using multiple files

- Assume that a large-scale program has to be written
- Breakup program into related sets of functions.
- Each set of functions is programmed with:
  - A header file, ending in “.h” (complex.h)
    - Contains mostly structure definitions and function prototypes
    - Global variables
  - An implementation file, ending in “.c” (complex.c)
    - Includes all related functions’ C code
- One main file that calls functions as needed
  - Includes all relevant .h files
  - This and only this file contains **main() function**

Example

- Assume: main file and helper files are same directory
- Helper files:
  - FileA.h, FileA.c, FileB.h, FileB.c, FileC.h, FileC.c
- Main() is in file called MyProg.c
  - #include <stdio.h> and other system libraries
  - #include “FileA.h” etc.
  - Can also link to other needed system libraries (e.g. xyz)
- Compile using:

```
$ gcc –o prog MyProg.c FileA.c FileB.c FileC.c –lxyz
$ ./prog
```

Makefile

- Makefile and other techniques to automate the process
  - Compile only files that have changed
  - Define dependencies between the different files etc.

OPTIONAL

Implementing *echo*

```
#include <stdio.h>
/* echo command line arguments: 1st version */
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;
}
```

echo – Pointer Version

```
#include <stdio.h>
/* echo comand line arguments: 2nd 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 …. */
```

cat - Continued

```
   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 ifp to file ofp */
void filecopy(FILE *ifp, FILE *ofp)
{
int c;
while ((c = getc(ifp)) != EOF)
                putc(c, ofp);
}
```

copy everything, blanks, tabs, endofline, till the file ends

Program Name in Error Message

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