

CSE / ENGR 142

Programming I

Input/Output, Libraries, and Files

© 1998 UW CSE

11/29/98

Q-1

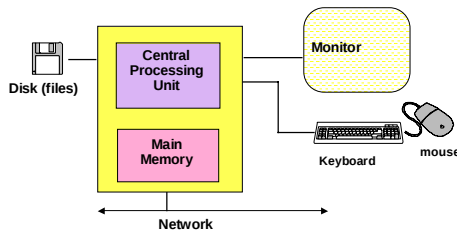
Textbook Readings

- Loose ends; combination of review and scattered textbook material
- Libraries:
 - Chapter 2 (here and there)
 - Chapter 13.2 (skim)
- Files:
 - Chapter 2.7 pp. 72-74
 - Chapter 5.5 pp. 234-236
 - Chapter 12.1

11/29/98

Q-2

Review: what's input/output?



11/29/98

Q-3

Why File I/O?

- Large volume of input data
- Large volume of output data
- More permanent storage of data
- Transfer to other programs
- Multiple simultaneous input and/or output streams

11/29/98

Q-4

Files

- A **"file"** is a collection of data on disk
 - managed by the user and the operating system
 - permanent
- A **"file name"** is how the user and OS know the file
 - follows OS naming rules (DOS: 8.3)
- We'll look at files used in compiling
- We'll look more at keyboard I/O
- We'll look at using text files in a C program
- First we'll look at data files

11/29/98

Q-5

DATA FILES

- **Business Data:** customer files, payroll files, ...
- **Scientific Data:** weather data, environmental data, topographic maps, ...
- **Image Data:** web images, satellite images, medical images, ...
- **Web Data:** HTML, GIF, JPEG, PNG, XML, ...

11/29/98

Q-6

Business Data File

NAME	SSN	BIRTH	ADDRESS
John Jones	532456895	7/1/75	916 4th NE, Seattle 98105
Sally Smith	872996547	9/3/79	526 5th NE, Seattle 98105

11/29/98 Q-7

Scientific Data File

X	Y	ELEVATION	RAINFALL
300	450	1900	3.45
275	900	300	12.62

11/29/98 Q-8

Files Used in Compiling

•Source Files

- .c files:** C programs and functions
- .h ("header") files:** fragments of C code
real-world projects may contain hundreds of source files!

•Compiled Files (system-dependent names)

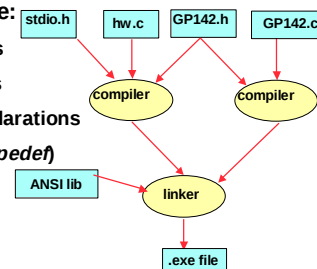
- object files:** compiled C code ready to link
- libraries:** collections of compiled C functions
- executable files:** linked machine-language, ready to load into memory

11/29/98 Q-9

Header files (.h)

•Fragments of C code:

- Function Prototypes
- Symbolic Constants
- Global Variable Declarations
- Type Definitions (*typedef*)



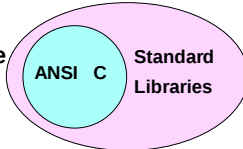
11/29/98 Q-10

Libraries

Files of compiled, pre-written programs

Why?

- Reuse existing code
- Enhance portability
- Hide system dependencies



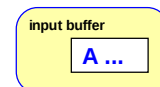
11/29/98 Q-11

Keyboard Dangers

What happens if the user types **A** in the following situation?

```

int score ;
scanf("%d", &score) ;
while (score != 0) {
    printf("%d ln", score) ;
    scanf("%d", &score) ;
}
  
```



11/29/98 Q-12

scanf's Return Value

- *scanf* actually returns an *int*
 - tells the number of values successfully read: see Section 5.5.

```
int status, id, score ;
double grade ;
status = scanf("%d %lf %d", &id, &grade, &score) ;
if (status < 3)
    printf("Error in input ln") ;
```

11/29/98 Q-13

More Robust Input

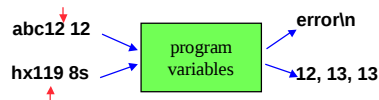
/ Robustly read an integer, consuming nondigits */*

```
int read_int (void)
{
    int status, input ;
    char junk ;
    status = scanf("%d", &input) ;
    while (status < 1) { /* unsuccessful read */
        scanf("%c", &junk) ; /* consume 1 char */
        status = scanf("%d", &input) ; /* try again */
    }
    return(input) ;
}
```

11/29/98 Q-14

Files as Streams of Characters

keyboard/screen are special cases
input / output streams of characters

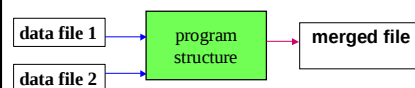


Multiple streams can be used simultaneously
In reality, stream flows through a **buffer** rather than directly into or out of variables.

11/29/98 Q-15

Files as Records with Fields

Business and Scientific Data



11/29/98 Q-16

File variables

- Reminders:
 - A **file** is a collection of data on disk
 - A **file name** is how the user and OS know the file
 - permanent name, follows OS naming rules
- A **file variable** is a variable in the C program which represents the file
 - temporary: exists only when program runs
 - follows C naming rules

11/29/98 Q-17

What's in *stdio.h*?

- Prototypes for I/O functions.
 - Definitions of useful **#define** constants
 - Example: EOF for End of File
 - Definition of **FILE struct** to represent information about open files.
 - File variables in C are pointers to the **FILE struct**.
- FILE** *myfile;

11/29/98 Q-18

Opening A File

- **"Opening" a file:** making a connection between the operating system (file name) and the C program (file variable)
 - library function **fopen**
 - specify "r" (read, input) or "w" (write, output)
- **Files must be opened** before they can be used
- Files **stdin/stdout** (used by scanf/printf) are automatically opened & connected to the keyboard and display

11/29/98 Q-19

File Open Example

```
/*usually done only once in a program*/
/*usually done near beginning of program*/

FILE *infile, *outfile; /*file variables*/
char ch;

/* Open input and output files */
infile = fopen ("Student_Data", "r");
outfile = fopen ("New_Student_Data", "w");
```

11/29/98 Q-20

End of File (EOF)

- defined in **stdio.h**
- **#define EOF** (- some value)
 - Usually -1 (but don't depend on its value)
 - I/O library routines use **EOF** in various ways to signal end of file.
 - Your programs can check for **EOF**
- **EOF** is a status, not an input value

11/29/98 Q-21

Four Essential Functions for Text I/O

```
FILE *filepi, *filepo; int status;

filepi = fopen ("name.ext", "r"); /* read */
filepo = fopen ("name.ext", "w"); /* write */
status = fscanf (filepi, "%...", &var, ... );
/* fscanf returns EOF on end of file */

fprintf (filepo, "%...", var, ... );
fclose (filepo);
```

11/29/98 Q-22

File Copy Example (streams)

```
/* copy an input file to an output file */
/* files must already be open before this*/
status = fscanf (infile, "%c", &ch);
while ( status != EOF ) {
    fprintf (outfile, "%c", ch);
    status = fscanf (infile, "%c", &ch);
}
printf ("File copied.\n");
fclose (infile);
fclose (outfile);
```

11/29/98 Q-23

File Copy (Compact Edition)

```
/* Many C programmers use this style*/
...
while ( fscanf (infile, "%c", &ch) != EOF )
    fprintf (outfile, "%c", ch);

printf ("File copied.\n");
fclose (infile);
fclose (outfile);
```

11/29/98 Q-24

File Example: Implementing a Database Query

```
#include <stdio.h>

int main(void)
{
    FILE *inp, *oup;
    int age, i;
    char name[20], ssn[9], ch;

    inp = fopen("db_file", "r");
    oup = fopen("result_file", "w");

    /* loop till the end-of-file */
    while (fscanf(inp, "%c", &name[0]) != EOF) {

        /* read name, ssn, age */
        for (j = 1; j < 20; j++) fscanf(inp, "%c", &name[j]);
        for (j = 0; j < 9; j++) fscanf(inp, "%c", &ssn[j]);
        fscanf(inp, "%d", &age);
        /* read line feed character */
        fscanf(inp, "%c", &ch);

        /* copy name, ssn to output if age > 20 */
        if (age > 20) {
            for (j = 0; j < 20; j++) fprintf(oup, "%c", name[j]);
            for (j = 0; j < 9; j++) fprintf(oup, "%c", ssn[j]);
            fprintf(oup, "\n");
        }
    }
    fclose(inp); fclose(oup);
    return (0);
}
```

Equivalent query in SQL
database language:

```
SELECT NAME, SSN
FROM DB_FILE
WHERE AGE > 20;
```

11/29/98 Q-25

File Example: Expanding tabs

```
#include <stdio.h>
int main(void)
{
    FILE *infilep, *outfilep;
    char ch;
    int column = 0;

    /* Open input and output files */
    infilep = fopen("prog.c", "r");
    outfilep = fopen("tabless-prog.c", "w");

    /* process each input character */
    while (fscanf(infilep, "%c", &ch) != EOF) {
        if (ch == '\t' || ch == '\r') {
            /* end of line: reset column counter */
            column = 0;
            fprintf(outfilep, "%c", ch);
        } else if (ch == '\t') {
            /* tab: output one or more spaces, */
            /* to reach the next multiple of 8. */
            do {
                fprintf(outfilep, "%c", ' ');
                column++;
            } while ((column % 8) != 0);
        } else {
            /* all others: count it, and copy it out */
            column++;
            fprintf(outfilep, "%c", ch);
        }
    }
    fclose(infilep);
    fclose(outfilep);
    return 0;
}
```

Input: a b t c
d v e f
Output: a b c
d e f

11/29/98 Q-26

File Example: Merging two sorted files

```
#include <stdio.h>
#define MAXLINE 10000 /* ASSUMES no line longer */
int main(void)
{
    FILE *in1p, *in2p, *outp;
    char buffer1[MAXLINE], buffer2[MAXLINE];
    char *stat1, *stat2;

    in1p = fopen("sorted-file1", "r");
    in2p = fopen("sorted-file2", "r");
    outp = fopen("merged-file", "w");

    stat1 = fgets(buffer1, MAXLINE, in1p);
    stat2 = fgets(buffer2, MAXLINE, in2p);
    while (stat1 != NULL && stat2 != NULL) {
        if (strcmp(buffer1, buffer2) < 0) {
            fprintf(outp, "%s", buffer1);
            stat1 = fgets(buffer1, MAXLINE, in1p);
        } else {
            fprintf(outp, "%s", buffer2);
            stat2 = fgets(buffer2, MAXLINE, in2p);
        }
    }
    while (stat1 != NULL) {
        fprintf(outp, "%s", buffer1);
        stat1 = fgets(buffer1, MAXLINE, in1p);
    }
    while (stat2 != NULL) {
        fprintf(outp, "%s", buffer2);
        stat2 = fgets(buffer2, MAXLINE, in2p);
    }
    fclose(in1p); fclose(in2p); fclose(outp);
    return 0;
}
```

really should CHECK
that no line is longer
than MAXLINE

11/29/98 Q-27