The Standard I/O Library

Unit 7 Objectives

- Perform file I/O operations using the Standard I/O Library
- Choose appropriate read/write functions for a given task
- Perform random access on a file
- Identify the differences between system buffers and "buffers" within the user program
- Describe the concept of buffered I/O

I/O Overview

- C does not have built in I/O statements
- I/O is accomplished via
  - Standard I/O Library Functions
  - System Calls

Standard I/O Functions	System Calls
Portable*	Not portable*
Functions in a library	Part of the op. sys.
Medium - Hi level	Low level
Variety of read/write functions	Limited number
Buffered	Unbuffered

* to other operating systems

Review and Preview of Standard I/O Functions

	I/O for *stdin* and *stdout*	File I/O
Character	*getchar()* *putchar()*	*fgetc()* *fputc()* *ungetc()*
Line	*gets()* *puts()*	*fgets()* *fputs()*
Formatted	*scanf()* *printf()*	*fscanf()* *fprintf()*
Record	- -	*fread()* *fwrite()*

5 Steps to Perform File I/O

1. #include <stdio.h>
2. Declare one FILE pointer per file
3. Open the file with fopen()
4. Use Standard I/O read/write functions
5. Close the file with fclose()

Important Constants in stdio.h

EOF	End-of-file error value
NULL	0 or (void *)0 in the new ANSI Standard
BUFSIZ	Size of an I/O buffer
FILE	A structure typedef, stores info about an open file
stdin	Pointer to FILE opened for standard input
stdout	Pointer to FILE opened for standard output
stderr	Pointer to FILE opened for standard error

The FILE Structure

- Holds information about an open file
- An array of FILE structures is used by the Standard Library functions
- The Programmer
  - Does not need to know format of FILE
  - Must declare one FILE pointer per open file

Opening a File - fopen()

SYNOPSIS

```
#include <stdio.h>
FILE *fopen( file-name, type )
char *file-name, *type;
```

DESCRIPTION

The file named is opened according to *type* which may be

"r"	"r+"
"w"	"w+"
"a"	"a+"

Returns NULL on failure.

EXAMPLE

```
 1  #include  <stdio.h>
 2
 3  main()
 4  {
 5      FILE    *fp;
 6
 7      fp = fopen("logfile", "w");
 8      if (fp == (FILE *)NULL)
 9              printf("Open failed\n");
10      ...
11  }
```

Effect of a Successful Open on a File

	"r" read	"w" write	"a" append
File Exists	-	Old contents discarded	-
File Does Not Exist	Error	File created	File created

- "r+", "w+", "a+" Updating
  Allows reading and writing

- "r+" Commonly used to read and change an existing file

Sample Program Using fopen( ) and fclose( )

```
1  /* This program copies a file, */
2  /* argv[1] is copied to argv[2] */
3
4  #include   <stdio.h>
5
6  main(argc, argv)
7  int    argc;
8  char   *argv[];
9  {
10     FILE    *rp, *wp;
11
12     if (argc < 3){
13             printf("2 FILE NAMES REQUIRED\n");
14             exit(1);
15             }
16     if ((rp=fopen(argv[1],"r")) == (FILE *)NULL)
17             printf("Can't open %s\n",argv[1]);
18             exit(2);
19             }
20     if ((wp=fopen(argv[2],"w")) == (FILE *)NULL)
21             printf("Can't open %s\n",argv[2]);
22             fclose(rp);
23             exit(3);
24             }
25     /* Read/write functions shown later */
26     fclose(rp);
27     fclose(wp);
28 }
```

Exercise - fopen( )

SYNOPSIS
```
#include <stdio.h>
FILE *fopen(file-name, type)
char *file-name, *type;
```

1. The synopsis shows that fopen() returns a FILE pointer (FILE *). Since it does not return an integer, the function should be declared before it is used. Why isn't fopen() declared on page 7-19?
2. For the given application, supply the appropriate second argument to fopen().

Program's action	Command line	Function call
Prints lines of file that contain pattern.	grep pattern file	fp = fopen(argv[2], "?");
Prints a copy of the file on a line printer; adds filename and time to logfile.	lineprint file	fp = fopen("logfile", "?");
Registers conference participants; queries and updates "reg.db", an existing database.	register	fp = fopen("reg.db", "?");
Records logon and logoff times on a multiuser system. The program is run once daily; "/etc/usage" contains info only about current day.	trackusers	fp = fopen("/etc/usage", "?");

Choosing the Appropriate Read/Write Function

		Normally Used With
Character		
	fgetc fputc ungetc	Text files (ASCII, EBCDIC)
Line		
	fgets fputs	Text files (ASCII, EBCDIC)
Formatted		
	fscanf fprintf	Text files (ASCII, EBCDIC)
Block		
	fread fwrite	Data files

Using read/write functions

```
/* Each program fragment copies a file */
```

```
/* Character by character */
    while ((c = fgetc(rp)) != EOF)
          fputc(c, wp);
```

```
/* Line by line */
    char    buf[256];

    while (fgets(buf, 256, rp) != (char *) NULL)
          fputs(buf,wp);
```

```
/* According to a specific format */
    char    name[40];
    int     id;

    while (fscanf(rp,"%s %d",name,&id) != EOF)
          fprintf(wp,"%s %d\n", name, id);
```

```
/* One structure at a time */
    struct info {
          char    name[50];
          int     id;
          } part;

    while (fread((char *)&part,sizeof(part),1,rp) != C
          fwrite((char *)&part,sizeof(part),1,wp);
```

A Closer Look at Character I/O: ungetc()

SYNOPSIS

```
int ungetc(c, stream)
int   c;
FILE  *stream;
```

DESCRIPTION

Puts one character back on input stream providing the stream was previously read. Next read operation will read that charact

EXAMPLE

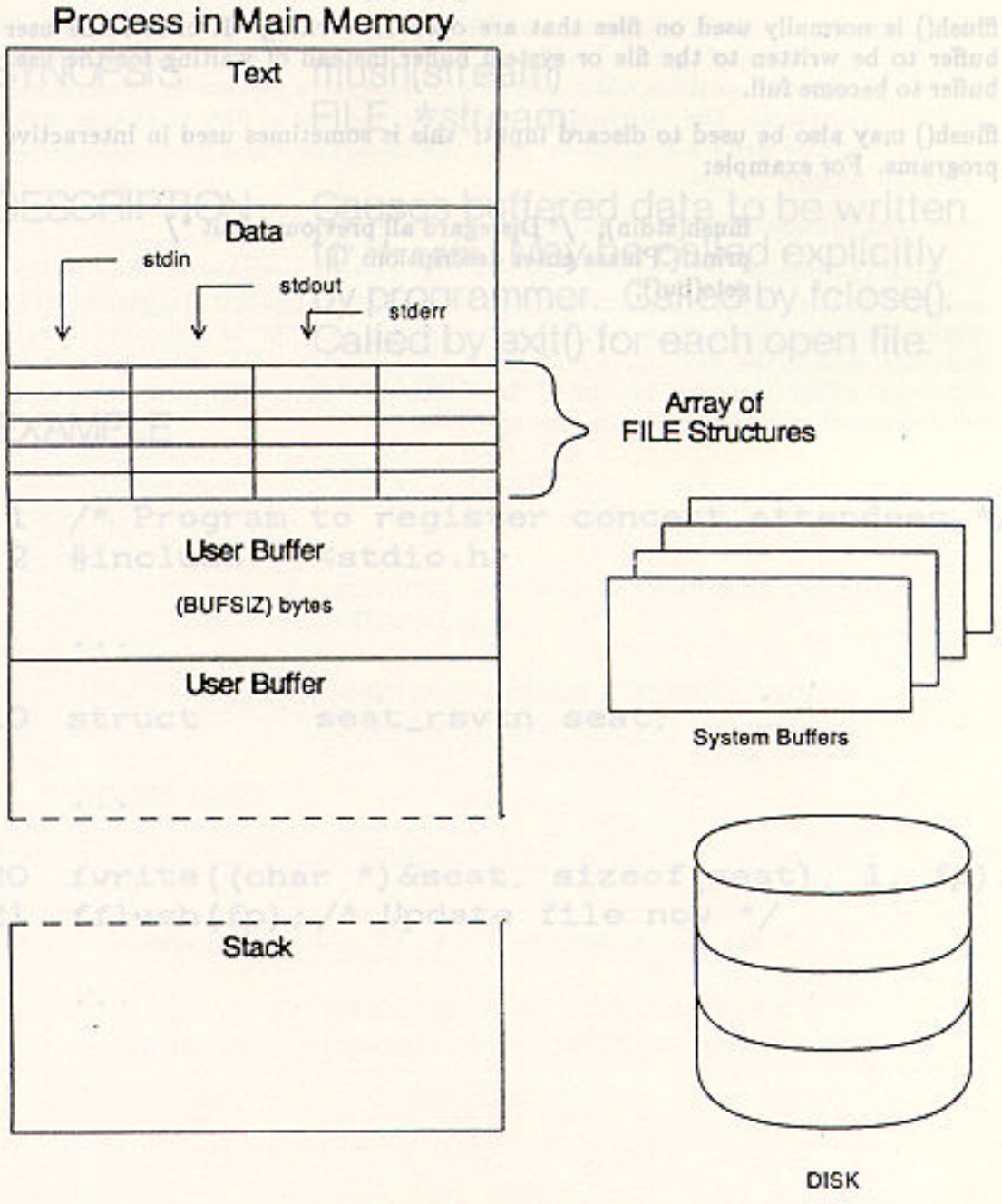
```
/* Skips over any number of white space */
/* characters in input file */

void
skip_whites(fp)
FILE    *fp;
{
    int c;
    while ((c = fgetc(fp)) == ' ' ||
        c == '\t' || c == '\n')
                ;  /* Null loop body */
    ungetc(c, fp);  /* Put non-white back */
}
```

Buffered I/O

- The Standard I/O Library provides "buffered I/O"
- Minimizes device access for efficiency
- Buffer:
  - Temporary storage area in main memory
    - Holds data to be read or written
- Data is transferred to/from devices in large chunks (BUFSIZ bytes)

Buffered I/O: Accessing a Disk File



Flushing a Buffer

SYNOPSIS
```
fflush(stream)
FILE *stream;
```

DESCRIPTION
Causes buffered data to be written to *stream*. May be called explicitly by programmer. Called by fclose(). Called by exit() for each open file.

EXAMPLE

```
 1  /* Program to register concert attendees */
 2  #include    <stdio.h>

    ...

10  struct      seat_rsvtn seat;

    ...

20  fwrite((char *)&seat, sizeof(seat), 1, fp);
21  fflush(fp); /* Update file now */

    ...
```

Random Access - fseek() and ftell()

SYNOPSIS

```
int fseek(stream, offset, ptrname)
FILE *stream;
long offset;
int ptrname;

long ftell(stream)
FILE *stream;
```

DESCRIPTION

fseek sets position of next read/write operation on the *stream*, *offset* bytes from *ptrname* which has the values:

0	top
1	current
2	end

Returns non-zero for improper seeks, otherwise 0.

ftell returns offset of current byte from top of file.

EXAMPLES

```
fseek(fp, 0L, 0);/* rewind */

fseek(fp, 0L, 2);/* bottom of file */

        /* back up one structure */
fseek(fp, -(long)sizeof(struct emp), 1);
```

Random Access - Sample Program

```
 1  /* Prints stored info about part requested */
 2  #include    <stdio.h>
 3
 4  struct      part_info {
 5      char    desc[20];
 6      int     qty;
 7      int     cost;
 8  } part;
 9
10  main()
11  {
12      FILE    *rptr;
13      long    x;
14
15      if ((rptr=fopen("p_data","r")) == (FILE *)NULL
16          fprintf(stderr,"Can't open data file.\r
17          exit(1);
18          }
19      printf("Enter sequence number of record: ");
20      scanf("%ld", &x);
21      fseek(rptr,(long)((x - 1) * sizeof(part)),0);
22      fread((char *)&part,sizeof(part),1,rptr);
23      printf("%s\t%d\t%d\n", part.desc,
24              part.qty, part.cost);
25  }
```

Random Access When Updating a File

- Update means:

  Reading and writing

  "r+", "w+", "a+"

- Must flush buffer between reads and writes

  using fseek() or rewind()

```
1   #include     <stdio.h>

    ...

19  /* Change all cost fields in an inventory */
20  /* file composed of structures */
21  while(fread((char *)&part,sizeof(part),1,fp)!= 0 ){
22      part.cost *= 1.07;
23      fseek(fp, -(long)sizeof(part), 1); /* back up */
24      fwrite((char *)&part, sizeof(part), 1, fp);
25      fseek(fp,0L,1); /* flush buffer before next read *,
26      }

    ...
```

Error Handling Functions

SYNOPSIS

```
#include <stdio.h>

int  feof(stream)
FILE *stream;
```

DESCRIPTION

feof() returns non-zero if end of file was previously reached during a read, else 0.

```
while(1) {
    n = fread( buf, 1, BUFSIZ, rp);
    fwrite(buf, 1, n, wp);
    if ( feof(rp) )
        break;
}
```

Other Error Handling Functions

SYNOPSIS

```
#include <stdio.h>

int ferror(stream)
FILE *stream;

void clearerr(stream)
FILE *stream;
```

DESCRIPTION

ferror() returns non-zero if a previous I/O error occurred, else 0.

clearerr() resets error and eof indicators to 0.

```
/* Try MAX times to read device */
success = 0;
clearerr(rp);
for (i = 1; i < MAX; i++) {
    fread( buf, BUFSIZ, 1, rp);
    if ( !ferror(rp) ) {
            success = 1;
            break;
            }
    clearerr(rp);
    }
if (success)
    printf("read successful\n");
else
    printf("Max retries - read failed\n");
```

Unit 7 Summary

5 Steps to Perform File I/O

1. #include <stdio.h>
2. Declare one FILE pointer per file
3. Open the file with fopen()
4. Use read/write functions
5. Close the file with fclose()

Random Access:

```
SYNOPSIS    #include  <stdio.h>
            int  fseek(stream, offset, ptrname)
            FILE  *stream;
            long  offset;
            int   ptrname;  /* 0-top, 1-current, 2-end */

            long  ftell(stream)
            FILE  *stream;
```

Error Handling:

```
SYNOPSIS    #include  <stdio.h>
            int  feof(stream)
            FILE  *stream;

            int ferror(stream)
            FILE  *stream;

            void clearerr(stream)
            FILE  *stream;
```

☞ See Manual For Other I/O Functions ☜