File I/O

File I/O

- In program
  - #include <stdio.h>
    - because the followings are defined in *stdio.h*
  - Declaration of a pointer variable to FILE structure
    - FILE *variable_name file pointer
  - file open
    - File pointer initialization
    - fopen()
  - output data to the file or read data from the file
    - fprintf(), fscanf(), fputc(), fgetc(), …
  - file close
    - fclose()
- FILE
  - A particular structure, of which members describe the current state of a file, defined in *stdio.h*

fopen(), fclose()

- File pointer declaration

  ```
  FILE *fp;
  ```

- Open a file

  ```
  fp = fopen("filename", "mode");
  ```

- Close a file

  ```
  fclose(fp);
  ```

fopen(), fclose()

Mode	Meaning
“r”	open text file for reading
“w”	open text file for writing
“a”	open text file for appending
“rb”	open binary file for reading
“wb”	open binary file for writing
“ab”	open binary file for appending

- When a mode ends with a + character, the file is to be opened for both reading and writing
  - “r+” : open text file for reading and writing
  - “w+” : open text file for writing and reading
  - “a+” : open text file for appending and reading

Formatted File I/O

- Function prototypes is defined in stdio.h
  - int fscanf(FILE *fp, const char *format, …);
  - int fprintf(FILE *fp, const char *format, …);
- #include <stdio.h>
- FILE *ifp, *ofp; /* File pointer declaration */
- ifp=fopen("in_file", "r"); /* File Open */
  ofp=fopen("out_file", "a")
- fscanf(ifp, *control_string, other_arguments*);
  fprintf(ofp, *control_string, other_arguments*);
- fclose(ifp); /* File Close */
  fclose(ofp);

Example

```
#include   <stdio.h>
void main(void)
{
    FILE      *ifp, *ofp;
    int         a, sum=0;

    ifp = fopen(“infile”, “r”);
    ofp= fopen(“outfile”, “w”);

    while(fscanf(ifp,“%d”, &a)==1)
        sum+=a;
    fprintf(ofp, “The sum is %d.\n”, sum);

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

stdin, stdout, stderr

- stdin
  - Standard input file (Keyboard)
- stdout
  - Standard output file (Screen)
- stderr
  - Standard error file (Screen)
- Standard I/O is automatically opened at the start of program and automatically closed at the completion of program => **we do not need to open and close these files.**
- Defined in stdio.h
- fprintf(stdout, …); is equal to printf(…);
  fscanf(stdin, …); is equal to scanf(…);
- Macros defined in terms of stdin and stdout

```
#define getchar()       getc(stdin)
#define putchar(c)      putc(c, stdout)
```

Character File I/O

- Read a character from file
  - c=getc(fp); /* macro, EOF for end of file or error */
  - c=fgetc(fp); /* function */

- Write a character to a file

  putc(c,fp); /* macro */

  fputc(c,fp); /* function */

- Check EOF or error

  feof(fp); /* non-zero if EOF */

  ferror(fp); /* non-zero if error occurred */

EOF

- End Of File
- Symbolic Constant defined as the specific numeric value in stdio.h

- #define EOF -1

  or

- #define EOF 0

Example: copy file1 file2

```
#include <stdio.h>
int main(void)
{
    FILE *fp1, *fp2;
    int c;

    fp1 = fopen("in.dat", "r");
    fp2 = fopen("out.rst","w");
    while((c=getc(fp1))!=EOF)
        putc(c,fp2);
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

copy

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    FILE *fp1, *fp2;
    int c;

    fp1 = fopen(argv[1],"r");
    fp2 = fopen(argv[2],"w");
    while((c=getc(fp1))!= EOF)
        putc(c,fp2);
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

copy in.dat out.rst

Error Handling: stderr and exit

```
#include <stdio.h>
void filecopy(FILE *, FILE *);
main(int argc, char *argv[])              /* output files to the screen*/
{
        FILE *fp;
        char *prog = argv[0];        /* program  name */
        if (argc == 1 )                  /* no args; copy standard input */
              filecopy(stdin, stdout);
        else
              while (--argc > 0)
                      if ((fp = fopen(*++argv, "r")) == NULL) {
                              fprintf(stderr, "%s: can't open %s\n", prog, *argv);
                              exit(1);
                      }
                      else {
                              filecopy(fp, stdout);
                              fclose(fp);
                      }

        if (ferror(stdout)) {
                      fprintf(stderr, "%s: error writing stdout\n", prog);
                      exit(2);
        }
        exit(0);
}
```

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

Error Handling: stderr and exit

- **int ferror(FILE *fp)**
  - Return a nonzero value if the error indicator has been set for the file associated with fp
- **exit** calls **fclose** for each open output file to flush out any buffered output

- Return values
  - 0: all is well
  - Non-zero : abnormal situation

Line I/O

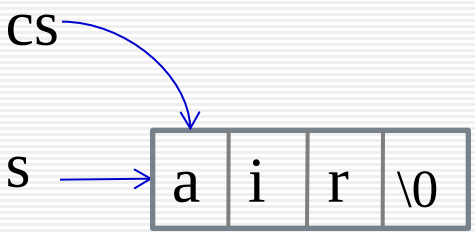
- Line oriented input function

  char *fgets(char *line, int maxline, FILE *fp)

  - If a newline is read or an end of file is encountered, no additional characters are read from the file
  - '\0' is inserted automatically at the end of buffer
  - if the end of file is encountered right at the start, return NULL; otherwise, return buffer (i.e., line)

fgets()

```
char *fgets(char *s, int n, FILE *iop)     /* get at most n-1 chars from iop */
{
        int    c;
        char *cs;

        cs = s;
        while (--n > 0 && (c = getc(iop)) != EOF)
                if ((*cs++ = c) == '\n') break;
        *cs = '\0';

        return (c == EOF && cs == s) ? NULL : s;
}
```



fputs()

- Line oriented output function

  int fputs(char *line, FILE *fp)

  - Copy the null-terminated string line (except null char itself) into the file associated with fp (i.e., appends a newline to the file)
  - A successful call returns a nonnegative value; otherwise, EOF

```
int fputs(char *s, FILE *iop)     /* put string s on file iop */
{
        int c;
        while (c = *s++)
                putc(c, iop);

        return ferror(iop) ? EOF : 0;
}
```

Random File I/O

- fseek(file_ptr, offset, place);
  - sets the file position indicator to a value that is *offset* bytes from *place*
  - place
    - SEEK_SET : the beginning of the file
    - SEEK_CUR : the current position
    - SEEK_END : the end of the file

```
fseek(fp, 0, SEEK_SET);      /* the beginning of the file */
fseek(fp, 0, SEEK_CUR);      /* the current position */
fseek(fp, 0, SEEK_END);      /* the end of the file */
fseek(fp, n, SEEK_SET);      /* the beginning position + n */
fseek(fp, n, SEEK_CUR);      /* the current position + n */
fseek(fp, -n, SEEK_END);     /* the end position - n */
```

- ftell(fp)
  - returns the current value of the file position indicator (the number of bytes from the beginning of the file, 0)

Example: write a file backwards

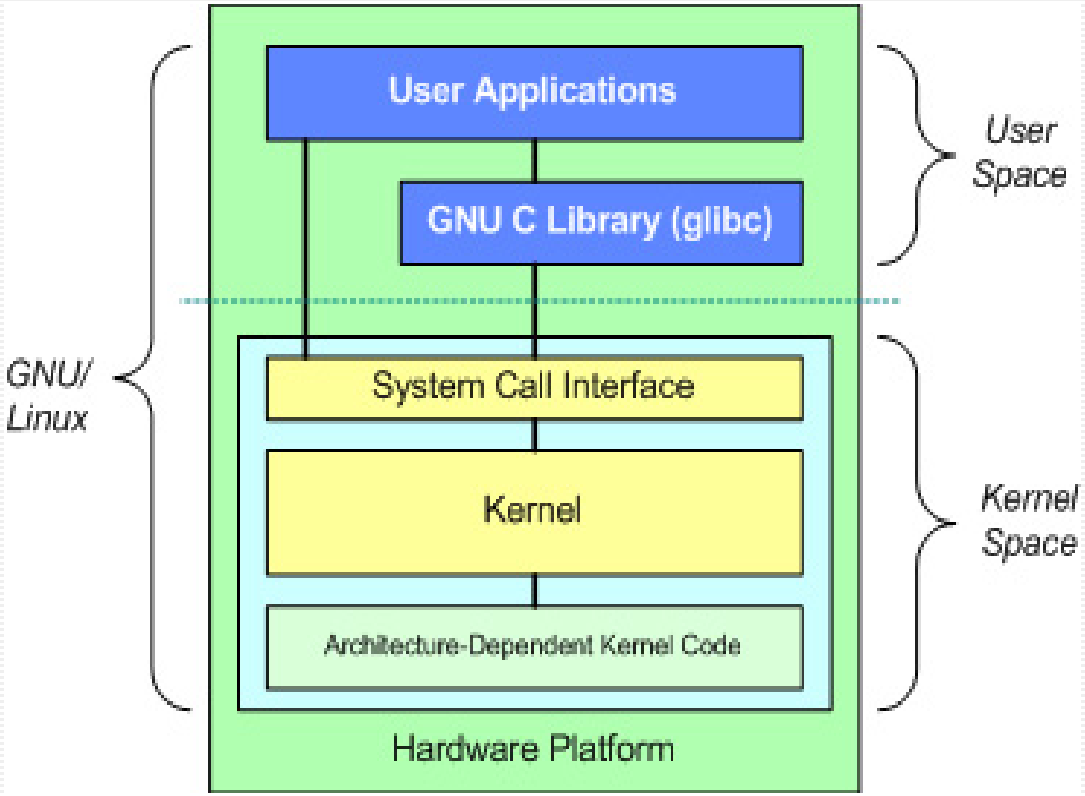
```
#include <stdio.h>
#define   MAXSTRING  100

int main(void)
{
        char  fname[MAXSTRING];
        int    c;
        FILE   *ifp;

        fprintf(stdout, "\nInput a file name:  ");
        scanf("%s", fname);
        ifp=fopen(fname, "r");

        fseek(ifp, 0, SEEK_END);
        fseek(ifp, -1, SEEK_CUR);
        while (ftell(ifp)>0)  {
              c = getc(ifp);
              putchar(c);
              fseek(ifp, -2, SEEK_CUR);
        }
        return 0;
}
```

Low Level I/O – System Calls

- create()
- open()
- close()
- Read()
- write()

System Call

- a request for a service to an operating system's kernel



- http://www.ibm.com/developerworks/library/l-linux-kernel/

Open, Creat, Close

```
#include <fcntl.h>
int creat(char *name, int perms);
int open(char *name, int flags, int perms);

int   fd;
fd = creat(name, perms);
fd = open(name, flags, perms);

fd = open(name, O_RDONLY , 0);   /* perms is always 0 for open */
```

- O_RDONLY open for reading only
- O_WRONLY open for writing only
- O_RDWR open for both reading and writing

File Access Permissions

Mnemonic	Bit representation	Octal representation
r--	100	04
-w-	010	02
--x	001	01
rw-	110	06
r-x	101	05
-wx	011	03
rwx	111	07

Mnemonic	Octal representation
rw-------	0600
rw-r-----	0640
rwxr-xr-x	0755
rwxrwxrwx	0777

rwx rwx rwx: owner, group, others

Read, Write

- int **read**(int fd, char *buf, int n);
- int **write**(int fd, char *buf, int n);

- fd: File Descriptor
  - Non-negative integer to identify a file in an OS
  - **0**: standard input, **1**: standard output, **2**: standard error
- Each call returns a count of the number of bytes transfe rred.

Example: read(), write()

```
#include "syscalls.h"
main()                  /* copy input to output */
{
        char buf[BUFSIZ];
        int n;
        while ((n = read(0, buf, BUFSIZ)) > 0)
                write(1, buf, n);
        return 0;
}
```

```
#include "syscalls.h"
int getchar(void)    /* unbuffered single character input */
{
        char c;
        return (read(0, &c, 1)==1) ? (unsigned char)c :  EOF;
}
```

Example: Open, Creat, Close

```
#include <stdio.h>
#include <fcntl.h>
#include "syscalls.h"
#define PERMS 0666          /* RW for owner, group, others */
main(int argc, char *argv[])      /* cp: copy f1 to f2 */
{
        int f1, f2, n;
        char buf[BUFSIZ];
        if (argc != 3) {
              fprintf(stderr,"Usage: cp from to"); exit(1);  }
         if ((f1 = open(argv[1], O_RDONLY, 0)) == -1) {
              fprintf(stderr,"cp: can't open %s", argv[1]); exit(1);  }
        if ((f2 = creat(argv[2], PERMS)) == -1) {
             fprintf(stderr,"cp: can't create %s, mode %03o", argv[2], PERMS); exit(1); }
        while ((n = read(f1, buf, BUFSIZ)) > 0)
            if (write(f2, buf, n) != n)  {
                  fprintf(stderr,"cp: write error on file %s", argv[2]); exit(1); }
       exit(0);
}
```

Random Access – Lseek

- Move around in a file without reading or writing any data
- long **lseek**(int fd, long offset, int origin)

  origin
  - 0: offset measured from the beginning
  - 1: offset measured from the current position
  - 2: offset measured from the end of the file
- **lseek**(fd, 0L, **2**); /* seek to the end of the file */
- **lseek**(fd, 0L, **0**); /* rewind to the beginning */

Random Access – Lseek

```
#include "syscalls.h"
int get(int fd, long pos, char *buf, int n)
{
        if (lseek(fd, pos, 0) >= 0)       /* get to pos */
                return read(fd, buf, n);
        else
                return -1;
}
```