NWEN241
File Handling

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

File Handling

- Write/read files

Open and Close a File

- Open a file

```
FILE *fopen(const char *filename, const char *mode);
```

  - `fopen()` returns a pointer to FILE, which is a structure that contains information about the file.
  - A NULL pointer is returned if the file cannot be accessed

Open and Close a File

- Open a file

```
FILE *fopen(const char *filename, const char *mode);
```

  - `fopen()` returns a pointer to FILE, which is a structure that contains information about the file.
  - A NULL pointer is returned if the file cannot be accessed

- Close a file

```
int fclose(FILE *fp); /* return either 0 or EOF */
                      /* success: 0, error: EOF */
```

  - Empty buffers and break all connections to the file

Open and Close a File

- An example

```
FILE *fpr, *fpw;
fpr = fopen(“afilenamer”, “r”);
fpw = fopen(“afilenamew”, “w”);

...         /* do something here */

fclose(fpr);
fclose(fpw);
```

- If you need to switch between read and write
  - `fflush()` is used to flush the buffer when switching between a read and a write

Open and Close a File

```
/* For UNIX based systems */
```

Mode	Meaning
“r”	Open file for reading
“w”	Open file for writing
“a”	Open file for appending
“rb”	‘b’ is ignored
“wb”	‘b’ is ignored
“ab”	‘b’ is ignored
“r+”	Open file for reading and writing
“w+”	Open file for writing and reading

Open and Close a File

```
/* For other systems and C89 */
/* compatibility */
```

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
“r+”	Open text file for reading and writing
“w+”	Open text file for writing and reading

Formatted Input/Output

- fscanf(), fprintf()

```
int fprintf(FILE *fp, const char *format, ...);

  fprintf(stdout, ...); /* is equivalent to */
  printf(...);
```

  - printf() writes formatted text to screen (the file associated with stdout)
  - fprintf() writes formatted text to a file

Formatted Input/Output

- fscanf(), fprintf()

```
int fscanf(FILE *fp, const char *format, ...);

   fscanf(stdin, ...); /* is equivalent to */
   scanf(...);
```

  - scanf() reads formatted text from keyboard (the file associated with stdin)
  - fscanf() reads formatted text from a file

Character Input/Output

- int fgetc(FILE *fp)/getc(FILE *fp)/getchar()
  - getchar() is equivalent to getc(stdin)
  - getc() is equivalent to fgetc() (getc() is a macro)
- int fputc(int c, FILE *fp)/putc(int c, FILE *fp)/putchar()
  - putchar() is equivalent to putc(stdout)
  - putc() is equivalent to fputc() (fputc() is a function)

Writing/reading text files

- Write a list to a text file

```
void writelisttofile(ptrNode pn)
{ FILE *fpw;
  char *filetow = "list.dat";

  if((fpw = fopen(filetow, "w")) == NULL)
    fprintf(stderr, "file %s could not be opened.\n",
  filetow);
  else
    for( ; pn != NULL; pn = pn->next)
      fprintf(fpw, "%c\t%x\n", pn->data, pn->next);

  fclose(fpw);
}
```


Writing/reading text files

- Read a list from a text file

```
void readlistfromfile()
{ char c;
  FILE *fpr;
  char *filetor = "list.dat";

  ...       /* if file could not be opened. */

    while((c=fgetc(fpr))!=EOF)
      fprintf(stdout, "%c", c);

  fclose(fpr);
}
```

Writing/Reading Blocks of Data

- Some data files store blocks of data (in binary format)
- Each block can be a complex data structure such as a structure or an array
- It is desirable to read/write the entire block instead of individual components

Writing/Reading Blocks of Data

- fread()/fwrite() is used to handle blocks of data

```
size_t fwrite(const void *a_ptr, size_t el_size, size_t
  n, FILE *fp);
```

fwrite() reads n*el_size bytes from the array whose first element is pointed to by a_ptr and writes them to the file associated with fp, and returns the number of items written.

```
size_t fread(void *a_ptr, sizet el_size, size_t n, FILE
  *fp);
```

fread() reads n*el_size bytes from the file associated with fp into the array whose first element is pointed to by a_ptr, and returns the number of items read.

Writing/Reading Blocks of Data

- Write a list to a binary file

```
void writelisttofile(ptrNode pn)
{ FILE *fpw;
  char *filetow = "list.dat";

  if((fpw = fopen(filetow, "w")) == NULL)
                              /* w or wb? */
    fprintf(stderr, "file %s could not be opened.\n", \
            filetow);
  else
    for( ; pn != NULL; pn = pn->next)
      fwrite(pn, Node_Size, 1, fpw);

  fclose(fpw);
}
```


Writing/Reading Blocks of Data

- Write a list to a binary file

```
void writelisttofile(ptrNode pn)
{ FILE *fpw;
  char *filetow = "list.dat";

  if((fpw = fopen(filetow, "w")) == NULL)
                                /* does not matter */
    fprintf(stderr, "file %s could not be opened.\n", \
            filetow);
  else
    for( ; pn != NULL; pn = pn->next)
      fwrite(pn, Node_Size, 1, fpw);

  fclose(fpw);
}
```

Writing/Reading Blocks of Data

- Read a list from a binary file

```
void readlistfromfile()
{ FILE *fpr;
  char *filetor = "list.dat“;
  Node anode;
  ptrNode pn = &anode;

  ...             /* if file could not be opened */
    while(!feof(fpr))   /* tells if EOF is set */
    { fread(pn, Node_Size, 1, fpr); /* fread sets EOF */
      if(!feof(fpr)) /* without if what would happen? */
        fprintf(stdout, "%c\t%x\n", pn->data, pn->next);
    }
  fclose(fpr);
}
```


Writing/Reading Blocks of Data

- Read a list from a binary file

```
void readlistfromfile()
{ FILE *fpr;
  char *filetor = "list.dat“;
  Node anode;
  ptrNode pn = &anode;

  ...             /* if file could not be opened */
    while(!feof(fpr))   /* tells if EOF is set */
    { fread(pn, Node_Size, 1, fpr); /* fread sets EOF */
      if(!feof(fpr)) /* double print last node */
        fprintf(stdout, "%c\t%x\n", pn->data, pn->next);
    }
  fclose(fpr);
}
```


Low-Level I/O

- Low-level I/O functions
  - open, close: open, close files (keyboard, screen), similar to fopen, fclose
  - read, write: read, write (similar to fread, fwrite)
  - lseek: random access (similar to fseek)
  - creat, unlink: create, remove a file
- Find out more from man

Low-Level I/O

- Comparison
  - open/close vs. fopen/fclose
  - read/write vs. fread/fwrite
  - …

Low-Level I/O

- Comparison
  - open/close vs. fopen/fclose
  - read/write vs. fread/fwrite

  ...

```
/* the later relies on the former */
```


Low-Level I/O

```
/* Change the case of letters in a file. */
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>         /* use io.h in MS_DOS */
#define   BUFSIZE   1024

int main(int argc, char **argv)
{
   char   mybuf[BUFSIZE], *p;
   int    in_fd, out_fd, n;

   in_fd = open(argv[1], O_RDONLY);
   out_fd = open(argv[2], O_WRONLY | O_EXCL | O_CREAT, 0600);
   while ((n = read(in_fd, mybuf, BUFSIZE)) > 0) {
      for (p = mybuf; p - mybuf < n; ++p)
       if (islower(*p))
          *p = toupper(*p);
       else if (isupper(*p))
          *p = tolower(*p);
      write(out_fd, mybuf, n);
   }
   close(in_fd);
   close(out_fd);
   return 0;
}
```


Low-Level I/O

```
/* Change the case of letters in a file. */

#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>         /* use io.h in MS_DOS */
#define   BUFSIZE   1024
```


Low-Level I/O

```
int main(int argc, char **argv)
{
   char   mybuf[BUFSIZE], *p;
   int    in_fd, out_fd, n;

   in_fd = open(argv[1], O_RDONLY);
   out_fd = open(argv[2], O_WRONLY | O_EXCL | O_CREAT,
      0600);
      /* it is an error if file is already there */
      /* create the file if it is not there */
      /* 0600: access permission, rw */
...

}
```

Low-Level I/O

```
int main(int argc, char **argv)
{
...
   while ((n = read(in_fd, mybuf, BUFSIZE)) > 0) {
      for (p = mybuf; p - mybuf < n; ++p)
       if (islower(*p))
          *p = toupper(*p);
       else if (isupper(*p))
          *p = tolower(*p);
      write(out_fd, mybuf, n);
   }
   close(in_fd);
   close(out_fd);
   return 0;
}
```


Low-Level I/O

```
/* Change the case of letters in a file. */
#include <ctype.h>
#include <fcntl.h>
#include <unistd.h>         /* use io.h in MS_DOS */
#define   BUFSIZE   1024

int main(int argc, char **argv)
{
   char   mybuf[BUFSIZE], *p;
   int    in_fd, out_fd, n;

   in_fd = open(argv[1], O_RDONLY);
   out_fd = open(argv[2], O_WRONLY | O_EXCL | O_CREAT, 0600);
   while ((n = read(in_fd, mybuf, BUFSIZE)) > 0) {
      for (p = mybuf; p - mybuf < n; ++p)
       if (islower(*p))
          *p = toupper(*p);
       else if (isupper(*p))
          *p = tolower(*p);
      write(out_fd, mybuf, n);
   }
   close(in_fd);
   close(out_fd);
   return 0;
}
```


Next Week

- Low-level programming
- Writing larger programs