Introduction to C I/O

Console and file I/O

- File I/O is implemented through library functions
  - an open file is represented by a FILE object
  - library functions manipulate files through file pointers (FILE*)
- File I/O library is declared in <stdio.h>
- File functions are used at the console by specifying a file
  - stdin (keyboard)
  - stdout (screen)
  - stderr (screen)
- Standard I/O functions only perform tty I/O

Opening & Creating Files

The fopen library function

- FILE* fopen(char* filePath, char* mode)
  - filePath is the file name (possible a full path)
  - mode is a string that indicates how the file is opened
    - r, rb: open for reading
    - w, wb: open for writing
    - a, ab: open/append at end; create file if it doesn’t exist
  - Note: the 'b' has no effect in Unix (but is accepted for portability)
  - Returns NULL on failure
- Common idiom:
  ```c
  FILE* fp;
  if ((fp = fopen("mort.out", "w")) == NULL)
  {
    fprintf(stderr, "error opening file");
    exit(1);
  }
  ```

Closing Files

Reclaiming resources

- An open file consumes limited resources
  - Each process has a finite number of files that may be simultaneously open
  - Each FILE stream has a dynamically allocated buffer
  - If a file pointer is lost before the corresponding file is closed, these resources are lost (they are reclaimed only when the process terminates)
- int fclose(FILE* fp)
  - Returns EOF on error, 0 otherwise
  - It’s not clear what to do on error anyway
- int fcloseall(void)
  - Files are automatically closed when the process terminates

Reusing Standard Streams

The freopen library function

- FILE* freopen(char* filePath, char* mode, FILE* stdstream)
  - filePath is the file name (full or relative path)
  - mode is a string that indicates how the file is opened (see Slide 3)
  - stdstream is a FILE stream that is already open (almost always one of stdin, stdout, or stderr)
  - Returns NULL on failure

Formatted I/O

Reading and writing files

- Prototype in <stdio.h>
- int fprintf(FILE* fp, "controlString", arg1, arg2, arg3 ...);
  - syntactically and functionally similar to printf
  - first argument is a file pointer
  - writes to the indicated file
- int fprintf(fpout, "Monthly payment = \%f", payment);
- int fscanf(FILE* fp, "controlString", arg1, arg2, arg3 ...);
  - syntactically and functionally similar to scanf
  - first argument is a file pointer
  - reads from the indicated file
- fscanf(fpin, "\%lf", &rate);
Character Input

Reading one character at a time

- int getc(FILE* fp) /* macro */
- int fgetc(FILE* fp) /* function */
- Return EOF (< 0, usually -1) on error or end of file
- Common idiom:

```
int c;

while ((c = fgetc(fp)) != EOF)
    /* process character c */
```

C Programming Example

Implementing the Unix cp utility

- `cp inflie outfile`
  - copies the contents of inflie to outfile; inflie is unchanged
- `FILE* infl;
  include <stdio.h>`
  - needed for FILE, fprintf, fgets, and fputs
- `FILE* outfile;
  include <stdio.h>`
  - needed for exit

```
int main(int argc, char* argv[])
{
    FILE* infl, *outfile;
    int c;
    
    if (argc != 3)
        fprintf(stderr, "usage: cp <inputfile> <outputfile>\n");
        exit(1);
    
    infl = fopen(argv[1], "r");
    if (infl == NULL)
        fprintf(stderr, "Error opening file\n");
        exit(1);
    outfile = fopen(argv[2], "w");
    if (outfile == NULL)
        fprintf(stderr, "Error opening file\n");
        exit(1);
    
    while ((c = fgetc(infl)) != EOF)
        fputc(c, outfile);
    
    fclose(infl);
    fclose(outfile);
    return 0;
}
```

Other Single-Character Input Functions

Non-standard (i.e., not ANSI) functions

- Prototype usually in <conio.h> (Borland and Microsoft)
- Included in the Unix curses library
- Does not require a new line (pressing the enter key)
- Reads the character but does not echo the character
  - `int getc(void)`; Borland / Unix
  - `int _getch(void)`; Microsoft
- Reads the character and echoes it to the screen
  - `int getche(void)`; Borland / Unix
  - `int _getche(void)`; Microsoft

Distinguishing Errors From EOF

Used in conjunction with functions that can return EOF

- `fgetc, getc, and getchar return EOF on end of file and on error`
- `int ferror(FILE* fp)`
  - Often called after one of the character functions returns EOF
  - Returns true (non-zero) if the file is at the end; returns false (zero) otherwise
- `int feof(FILE* fp)`
  - Often called after one of the character functions returns EOF
  - Returns true (non-zero) if there is an error on the input stream; returns false (zero) otherwise
- `void clearerr(FILE* fp)`
  - Clears errors on an input stream
Ungetting Input

Used with character read functions

- `int ungetc(int c, FILE* fp)`
  - Pushes one character back onto the input stream
  - ANSI guarantees only character can be pushed back; some systems allow more
  - Pushed back character is read with next read command
  - Pushed back character can be different than read character
  - Useful for "peeking ahead" one character

Line Input

Reading one line at a time

- `char* fgets(char* buf, int n, FILE* fp)`
  - Reads either n-1 characters or until new-line (whichever is shortest)
  - Stores the new-line if read
  - Always null-terminates the line
- `char* gets(char* buf)` /* not recommended */
  - Reads until new-line and discards the new-line
  - No test to prevent reading beyond the end of the buffer
  - Returns NULL at end of file or error, buf otherwise
- Common idiom:
  ```c
  char line[STRLIM];
  while (fgets(line, STRLIM, fp) != NULL) {
    line[stren(len)-1] = '\0';
    /* process line */
  }
  ```

Line Output

Writing one line at a time

- `int fputs(const char* str, FILE* fp)`
- `int putc(const char str)`
- Returns EOF on error, non-negative value otherwise
- `str must be null-terminated, but the null is not written out`

Positioning Within A Stream

Random access with fixed-size blocks

- `long ftell(FILE* fp)`
  - Returns the current file position or -1L on error
- `int fseek(FILE* fp, long offset, int whence)`
  - SEEK_SET : offset is from beginning of file
  - SEEK_CUR : offset is from current position (offset may be neg)
  - SEEK_END : offset is from end of file (offset may be neg)
- `void rewind(FILE* fp)`
- `int fgetpos(FILE* fp, fpos_t* pos)`
- `int fsetpos(FILE* fp, const fpos_t* pos)`
- Returns non-zero on error, 0 otherwise

struct Examples

struct variables and struct pointers

```
struct date {
  int year;
  int mon;
  int day;
};
struct date when, datePtr;
```

Accessing elements or members

- If the LHS is a struct object, use the . (pronounced "dot")
- If the LHS is a pointer to a struct, use the -> (formed by the minus and greater than symbols, pronounced "arrow")

Block I/O

AKA binary I/O

- `size_t fread(void* ptr, size_t size, size_t no, FILE* fp)`
- `size_t fwrite(const void* ptr, size_t size, size_t no, FILE* fp)`
- Returns the number of objects read or written
- Used to process unformatted byte streams, arrays, & structs
- Common idioms:
  ```c
  struct emp  empBuf;
  while (fread(&empBuf, sizeof(struct emp), 1, fp1) != 0) {
    fwrite(&empBuf, sizeof(struct emp), 1, fp2);
  }
  ```