COMP20200 Unix Programming

Lecture 4

CS, University College Dublin, Ireland

Input/Output in C

Header file `stdio.h`
(located /usr/include/stdio.h, have a look!)
Library: `libc.a` or `libc.so`
(located in subdir of /usr/lib)

- printf()
- fprintf()
- putchar()
- putc()
- puts()
- fputc()
- fputs()
- fwrite()

- scanf()
- fscanf()
- getchar()
- getc()
- gets()
- fgetc()
- fgets()
- ungetc()
- fread()
- getline()

A very simple cat

```
#include <stdio.h>

int main(){
int c;
while((c=getchar())!= EOF){
  putchar(c);
}

return 0;
}
```

Standard streams, a recap

- Standard in `stdin`
    - from keyboard, buffered
    - from a file `$ a.out < inputfile`
    - from another programme with pipe: `$ ls | a.out`
- Standard out `stdout`
    - buffered
    - to screen
    - to a file `$ a.out > outputfile`
    - to another programme: `$ a.out | sort`
- Standard Error `stderr`
    - buffered
    - to screen
    - to a file `$ a.out 2> outputfile`
- All combined:

```
$ a.out < inputfile &> outputfile
```

FILE pointers

- `FILE *fopen(const char *filename, const char *mode);`
- `FILE *freopen(const char *filename, const char *mode, FILE *stream);`
- modes:
  - `r` open a text file for reading
  - `w` truncate to zero length or create a text file for writing
  - `a` append; open or create text file for writing at end-of-file
  - `rb` open binary file for reading
  - `wb` truncate to zero length or create a binary file for writing
  - `ab` append; open or create binary file for writing at end-of-file
  - `r+` open text file for update (reading and writing)
  - `w+` truncate to zero length or create a text file for update
  - `a+` append; open or create text file for update
  - `r+b or rb+` open binary file for update (reading and writing)
  - `w+b or wb+` truncate to zero length or create a binary file for update
  - `a+b or ab+` append; open or create binary file for update
- `int fclose(FILE *stream);`

Open streams for IO

```
FILE *ifp, *ofp;
char *mode = "r";
char outputFilename[] = "out.list";

ifp = fopen("in.list", mode);
if (ifp == NULL) {
  fprintf(stderr, "Can't open input file in.list!\n");
exit(1);
}

ofp = fopen(outputFilename, "w");
if (ofp == NULL) {
  fprintf(stderr, "Can't open output file %s!\n",
      outputFilename);
exit(1); }
...
```

Write to streams

```
...
char username[9];
int score;
while (!feof(ifp)) {
  if (fscanf(ifp, "%s %d", username, &score) != 2)
    break;
  fprintf(ofp, "%s %d", username, score+10);
}
fclose(ifp);
fclose(ofp);
```

Closing files is important, especially for buffered output.

Output functions

- Formatted print
  - `int fprintf(FILE *stream, const char *format, ...);`
  - `fprintf(stdout, ‘‘...’’);` is equivilent to `printf(‘‘...’’);`
  - to print to stderr: `fprintf(stderr, ‘‘...’’);`
  - see `man fprintf` for more.
- Write single character
  - `int fputc(int c, FILE *stream);`
- Write a string of characters
  - `int fputs(const char *s, FILE *stream);`
  - The null character
    - o marks end of string but is not output.

Input functions

- Formatted input
  - `int fscanf(FILE *stream, const char *format, ...);`
  - `fscanf(stdin, ``...'');` is equivilent to `scanf(``...'');`
  - see `man fscanf` for more.
- Read single character
  - `int fgetc(FILE *stream);`
- Read a string of characters
  - `char *fgets(char *s, int n, FILE *stream);`

fgets Read into buffer

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

int main(void)
{
   char buffer[BUFFER_SIZE]; /* a read buffer */
   while( fgets (buffer, BUFFER_SIZE, stdin) != NULL) {
      printf("%s",buffer);
   }
   return 0;
}
```

Reads at most one less then `BUFFER_SIZE`
Stops after EOF or error.

getline

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

- Preferred method for reading lines of text
- gets, fgets, and scanf, are too unreliable
- reads an entire line from a stream, up to and including the next newline character
- `n` specifies size size of memory allocated at lineptr.
- Returns the number of characters read or EOF.
- Then use can use functions like `sscanf`, `atoi`, `atof` etc.

Simple implementation of getline

```
 /* getline: read a line into s, return length */
int getline_(char s[], int lim) {
  int c, i;

  for (i=0; i<lim-1 && (c=getchar())!= EOF && c!='\n'; ++i)
    s[i] = c;
  if (c == '\n') {
    s[i] = c;
    ++i;
  }
  s[i] = '\0';
  return i;
}
```

* not same as stdlib getline

```c
#include <stdio.h>
#include <stdlib.h>
main(void) {
  FILE *fp;
  char *line = NULL;
  size_t len = 0;
  ssize_t read;

  fp = fopen("/etc/motd", "r");
  if (fp == NULL)
     exit(EXIT_FAILURE);

  while ((read = getline(&line, &len, fp)) != -1){
     printf("Retrieved line of length %zu :\n", read);
     printf("%s", line); }
  free(line);
  exit(EXIT_SUCCESS); }
```

* file path hard coded, change to command line argument?

- Command line tip(s) of the day:
  - **Ctrl+c** send signal `SIGINT` to program (Ask program to stop).
  - **Ctrl+z** send `SIGSTOP`. Will suspend program.
    - `jobs` To see suspended jobs.
    - `fg 1` Foreground job number 1.
    - `bg 1` Background job number 1.
    - `&` To run job in background

      ```
      $ sleep &
      ```
  - **Ctrl+d** send **EOF** end of file character.