Input and Output

Devendra Tewari

2024-10-09

Introduction

- Input and Output in C is provided by several standard library functions and not by the core language itself
- The standard library functions operate on streams (source or destination of data)
- The state of a stream is stored in the FILE structure declared in `<stdio.h>`
- A FILE structure variable should not be created by the programmer

Standard streams

- The C library provides three standard streams in `<stdio.h>`
  - Standard input – `stdin`
  - Standard output – `stdout`
  - Standard error – `stderr`

Standard Input

- Stream that represents the standard input; usually this is the keyboard
- The `stdin` object is the standard input stream; it can be redirected by using the `freopen` function
- It can also be redirected at the command line

  ```
  executable < filename
  ```

  - content of filename available in `stdin` of program

  ```
  executable1 | executable2
  ```

  - output of `executable1` available in `stdin` of `executable2`

Standard Output

- Stream that represents the standard output; usually this is the display
- The `stdout` object is the standard output stream; it can be redirected using the `freopen` function
- It can also be redirected at the command line

  ```
  executable > filename
  ```

  - redirects `stdout` of program to file `filename`

  ```
  executable1 | executable2
  ```

  - redirects `stdout` of `executable1` to `stdin` of `executable2`

Standard Error

- Stream that represents the standard error; usually this is the display
- The `stdout` object is the standard output stream; it can be redirected using the `freopen` function
- Program errors should be sent to this stream; this way, when the standard output is redirected to some other file, the error messages will continue to appear to the user on the display
- It can also be redirected at the command line

```
executable 2> filename
```

File Access

- Open a file stream

```
FILE* fp;
fp = fopen(name, mode);
```

  - `name` is a relative or absolute file name and path
  - `mode` can be `"r"`, `"w"`, `"a"`, `"rt"`, `"wt"`, `"at"`, `"rb"`, `"wb"`, `"ab"`, `"r+t"`, `"w+t"`, `"a+t"`, `"r+b"`, `"w+b"`, or `"a+b"`
- Use the functions in `<stdio.h>` to manipulate the content of the file stream
- Call `fclose` to close the file stream

Formatted Output

- Function `printf` prints text to stdout

  ```
  int printf(char* format, ...)
  ```

- Function `fprintf` prints to any open stream

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

- `format` is the format string; it contains the text to be printed, interspersed with conversion specifications that are used to convert and print the arguments that follow

Print conversion specification

`%[flags][width][.precision][modifiers]type`

- `flags` can be `-`, `+`, space, `0`, `#`
- `width` specifies the minimum field width
- `precision` specifies different things depending on the `type`
- `modifiers` can be `h`, `l` or `L`
- `type` can be `d`, `i`, `o`, `x`, `X`, `u`, `c`, `s`, `f`, `e`, `E`, `g`, `G`, `p`, `n`, or `%`

Formatted Input

- Function `scanf` reads formatted input from stdin

  ```
  int scanf(char *format, ...)
  ```

- Function `fscanf` reads formatted input from any stream

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

- `format` is the format string containing text to be matched against the input
- Blanks and tabs in the format string are ignored
- White-space characters in the input stream act as field separators

Input conversion specification

`%[*][width][modifiers]type`

- `*` specifies assignment suppression
- `width` specifies the maximum width
- `modifiers` can be `h` or `l`
- `type` can be `d`, `i`, `o`, `u`, `x`, `c`, `s`, `e`, `f`, `g`, `p`, `n`, `[...]`, `[^...]`, or `%`

```
scanf("%d/%d/%d", &day, &month, &year)
```

Variable length argument lists

- A function may contain a variable length argument list

  ```
  int printf(const char*, ...)
  ```

- Header `<stdarg.h>` contains macro definitions that define how to read the argument list
  - Declare a variable ap of type `va_list`
  - Call `va_start(ap, lastarg)` to initialize ap; `lastarg` is the last argument before `...`
  - Call `va_arg(ap, type)` to read next argument
  - Call `va_end(ap)` to clean up

Character input and output

`int getc (FILE* fp)`

- returns next character from stream `fp`, or `EOF`

`int putc(int c, FILE* fp)`

- write character `c` to stream `fp`
- returns character written, or `EOF` on error

`getchar()`

- same as `getc(stdin)`

`putchar(c)`

- same as `putc(c, stdout)`

Line input and output

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

- reads at most `(maxline - 1)` characters from file stream `fp`
- returns line, `NULL` on error, or `EOF`

`int fputs(char* line, FILE* fp)`

- writes the string in `line` to the file stream `fp`
- returns zero or `EOF` if an error occurs

File positioning

`fseek(FILE* stream, long offset, int origin)`

- sets the file position for the stream; offset may be `SEEK_SET`, `SEEK_CUR` or `SEEK_END`

`long ftell(FILE* stream)`

- returns the current file position or `-1L` on error

`void rewind(FILE* stream)`

- sets the file position to the beginning, this is same as calling `fseek(fp, 0L, SEEK_SET)`

Error handling

`void clearerr(FILE* stream)`

- clears end of file and error indicators

`int feof(FILE* stream)`

- returns non-zero if end of file indicator is set

`int ferror(FILE* stream)`

- returns non-zero if error indicator is set

`void perror(const char*)`

- prints error message

Listing directories

- Requires `<sys/stat.h>` and, on BSD/Linux, `<dirent.h>`

```
DIR* dir;
struct dirent* item;
struct stat statbuf;
dir = opendir(".");
item = readdir(dir);
while(item != NULL) {
  stat(item->d_name, &statbuf);
  if(S_ISDIR(statbuf.st_mode)) {
    //...
  }
  item = readdir(dir);
}
```

Exercise

- Write a program that functions like the Unix tar command. The program should pack all files in the current directory into a single file whose name is specified at the command line. If the program receives the `-u` flag followed by a file name, it should unpack the content of the file to the current directory

```
pack [-u] filename
```