ECE 220 Computer Systems & Programming

Lecture 14 – I/O in C

ECE ILLINOIS

ILLINOIS

I/O Stream

i.e. keyboard, monitor, files, etc.

Stream

- Interfacing with I/O and program
- a sequence of data (text or binary) to which the actual input/output is mapped

For example,


```
scanf("%d", &x)
```

Stream Abstraction for I/O

All character-based I/O in C is performed on **text streams**.

A stream is a **sequence of ASCII characters**, such as:

- the sequence of ASCII characters printed to the monitor by a single program
- the sequence of ASCII characters entered by the user during a single program
- the sequence of ASCII characters in a single file

Characters are processed in the order in which they were added to the stream.

- e.g., a program sees input characters in the same order as the user typed them.

Standard Streams:

Input (keyboard) is called **stdin**.

Output (monitor) is called **stdout**.

Error (monitor) is called **stderr**.

Buffering

- Every value that goes into the stream is captured by the low-level OS software and kept in a **buffer** (a small array)

Input Buffering

The buffer is released when the user presses Enter key.

Output Buffering

The buffer is released when the program submits a newline character ('\n')

- Buffer allows to decouple the producer from the consumer.

Input Buffer

getchar

- Reads one ASCII character from stdin (keyboard)
- LC-3 IN TRAP

Input Buffering


```
char in1, in2, in3;

in1 = getchar();
in2 = getchar();
in3 = getchar();

printf(”result:\n");
printf("%c", in1);
printf("%c", in2);
printf("%c", in3);
```

You type
ABCD↵

1. Only ‘A’, ‘B’, ‘C’ will be read by getchar()
2. Before type enter(↵), the buffer is not released to the stream

Input Buffer

getchar

- Reads one ASCII character from stdin (keyboard)
- LC-3 IN TRAP

Input Buffering


```
char in1, in2, in3;

in1 = getchar();
in2 = getchar();
in3 = getchar();

printf(”result:\n");
printf("%c", in1);
printf("%c", in2);
printf("%c", in3);
```

You type

A↵

Output Buffer

putchar

- Displays one ASCII character to stdout (monitor)
- LC-3 OUT TRAP

Output Buffering


```
int main(){
   putchar('a’);

   sleep(5);
   putchar('b');
   putchar('\n’);
}
```

What do you see?

1. ‘a’, then 5 seconds, then ‘b’
2. ‘ab’, then 5 seconds
3. 5 seconds, then ‘ab’.

Output Buffer

putchar

- Displays one ASCII character to stdout (monitor)
- LC-3 OUT TRAP

Output Buffering


```
int main(){
    putchar('a');

    sleep(5);
    putchar('b');
    putchar('\n');
}
```

What do you see?

1. ‘a’, then 5 seconds, then ‘b’
2. ‘ab’, then 5 seconds
3. 5 seconds, then ‘ab’.

→ 3. The buffer is released at ‘\n’.

Basic I/O Functions

- Creating I/O streams
  - `fopen`: open/create a file for I/O
  - `fclose`: close a file for I/O
- I/O one character at a time
  - `fgetc`: Reads an ASCII character from stream
  - `fputc`: Writes an ASCII character to stream
  - `getchar`: Reads an ASCII character from the keyboard
  - `putchar`: Writes an ASCII character to the monitor
- I/O one line at a time
  - `fgets`: Reads a string (line) from stream
  - `fputs`: Writes a string (line) to stream
- Formatted I/O
  - `fprintf`: Writes a formatted string to stream
  - `fscanf`: Reads a formatted string to stream

File I/O

- A file is a sequence of ASCII characters stored in some storage device.
- Each file is associated with a stream.
  - It can be input stream or output stream or both.
- To read or write a file, we declare a file pointer (The `FILE` type is defined in `<stdio.h>`)

```
FILE *infile;
```

- Read/write a file requires 3 step:
  1. Open the file
  2. Do reading or writing
  3. Close the file

Creating I/O stream

FILE* fopen(char* filename, char* mode)

Open a file to read or write

- **Parameters**
  - filename
  - mode: how the file will be used
    - "r" – read from the file
    - "w" – write, starting from the beginning of the file
    - "a" – write, starting at the end of the file (append)
- **Return value**
  - success: returns a pointer to FILE
  - failure: returns NULL

Creating I/O stream

`int fclose(FILE* stream)`

Close a file

- **Parameters**
  - stream: Pointer to a file

- **Return value**
  - success: returns 0
  - failure: returns EOF
    (Note: EOF is a macro, commonly -1)

```
FILE *myfile;
myfile = fopen("test.txt", "w");
if(myfile == NULL){
    printf("Cannot open file for write.\n");
    return -1;
}
        .
        .
        .
fclose(myfile);
return 0;
```

I/O one character at a time

`int fgetc(FILE* stream)`

Read a single character from a file, then advanced to the next character.

- **Parameters**
  - stream: Input stream

- **Return value**
  - success: returns the current character
  - failure: returns EOF

I/O one character at a time

`int fputc(int character, FILE* stream)`

Write a single character to a file

- **Parameters**
  - character: character to be written
  - stream: Output stream

- **Return value**
  - success: write the character to file and returns the character written
  - failure: returns EOF

Example

```
char c;
FILE *fp1, *fp2;

if((fp1=fopen("original.txt", "r")) == NULL){
   printf("Unable to open a file.\n");
   return -1;
}
if((fp2=fopen("modified.txt", "w")) == NULL){
   printf("Unable to open a file.\n");
   return -1;
}

do{
   c = fgetc(fp1);
   if(c>='0' && c<='9')
      fputc(c,fp2);
}while(c!= EOF);
fclose(fp1);
fclose(fp2);
```

I/O one line at a time

`char* fgets(char* string, int num, FILE* stream)`

Read a line from a file

- **Parameters**
  - string: Pointer to a destination array
  - num: Max # of char to be copied into *string* *(num-1)*
  - stream: Input stream

- **Return value**
  - success: returns a pointer to string
  - failure: returns NULL

- fgets vs scanf

```
char buf[SIZE_BUF];

//store into buf until SIZE_BUF-1 characters
//or a newline or the end-of-file
fgets(buf, SIZE_BUF, stdin);

//store into buf until whitespace
scanf("%s", buf);
```

Example: Remainders in buffer

```
#define BUF_SIZE 6
int main(){
   char buf1[BUF_SIZE];
   char buf2[BUF_SIZE];

   printf("Enter 4 digits (** **): ");
   fgets(buf1, BUF_SIZE, stdin);
   printf("%s\n", buf1);

   printf("Enter 4 digits (** **): ");
   fgets(buf2, BUF_SIZE, stdin);
   printf("%s\n", buf2);
}
```

I/O one line at a time

`int fputs(const char* string, FILE* stream)`

Write a string to a file

- **Parameters**
  - string: Pointer to a source array
  - stream: Output stream

- **Return value**
  - success: returns a non-negative value
  - failure: returns EOF

Formatted I/O

`int fprintf(FILE* stream, const char* format, …)`

Write formatted output to a stream

- **Parameters**
  - stream: Output stream
  - format: String that contains the text to be written
    - format specifier: %d, %lf, %s, etc
  - (additional arguments): Replace a format specifier

- **Return value**
  - success: returns the number of characters written
  - failure: returns a negative number

Formatted I/O

`int fscanf(FILE* stream, const char* format, …)`

Read formatted input from a stream

- **Parameters**
  - stream: Input stream
  - format: String that specifies how to read the input
    - format specifier: %d, %lf, %s, etc
  - (additional arguments): A pointer to store read data

- **Return value**
  - success: returns the number of items read
  - failure: returns EOF

Example

data.txt

4311 Alice 3.42
1133 Bob 4.0

→

swapped.txt

Alice 4311 3.42
Bob 1133 4.0

```
int uid;
char name[20];
double gpa;
```

Example

data.txt

4311 Alice 3.42
1133 Bob 4.0

→

swapped.txt

Alice 4311 3.42
Bob 1133 4.0

```
int uid;
char name[20];
double gpa;

FILE *fp_in = fopen("data.txt", "r");
FILE *fp_out = fopen("swapped.txt", "w") ;

while( fscanf(fp_in, "%d %s %lf", &uid, name, &gpa) != EOF)
   fprintf(fp_out, "%s %d %lf\n", name, uid, gpa);

fclose(fp_in);
fclose(fp_out);
```

Read an mxn matrix from file *in_matrix.t*xt and write its transpose to file *out_matrix.txt*. **The first row of the file specifies the size of the matrix.**

```
FILE *in;
 if((in = fopen("in_matrix.txt", "r")) == NULL){
       printf("Unable to open a file\n");
       return -1;
   }
   fscanf(in, "%d %d", &m, &n);
   int matrix[m][n];

   for(i=0;i<m;i++)
      for(j=0;j<n;j++)
         fscanf(in, "%d ", &matrix[i][j]);

   fclose(in);
```

in_matrix.txt

```
2 3
1 2 3
4 5 6
```

out_matrix.txt

```
3 2
1 4
2 5
3 6
```

Read an mxn matrix from file *in_matrix.t*xt and write its transpose to file *out_matrix.txt*. **The first row of the file specifies the size of the matrix.**

```
FILE *out;
 if((out = fopen("out_matrix.txt", "w"))== NULL){
       printf("Unable to open a file\n");
       return -1;
   }
   fprintf(out, "%d %d \n", n, m);
   for(i=0;i<n;i++){
       for(j=0;j<m;j++)
          fprintf(out, "%d ", matrix[j][i]);
       fprintf(out, "\n");
   }
   fclose(out);
```

in_matrix.txt

```
2 3
1 2 3
4 5 6
```

out_matrix.txt

```
3 2
1 4
2 5
3 6
```

(sidenote) When do you use `stderr`?

- It's a good practice to redirect all error messages to `stderr`, while directing all regular outputs to `stdout`.
- Example:

```
fprintf(stdout , "Normal output1\n");
fprintf(stdout , "Normal output2\n");
fprintf(stderr, "Error1 \n");
fprintf(stdout , "Normal output3\n");
fprintf(stderr, "Warning1\n");
```

```
./a.out
```

[monitor]

Normal output1
Normal output2
Error1
Normal output3
Warning1

```
./a.out >a.log 2>err.log
```

[a.log]

Normal output1
Normal output2
Normal output3

[err.log]

Error1
Warning1

Variable Argument Lists

- The number of arguments in a `printf` or `scanf` depends on the number of data items being read or written.
- Parameters pushed onto stack from right to left.
- The format string is always at the top of the stack.