Review of Faculty Orientation Workshop on

Data Structures

Under the Aegies BoS (Electronics/E&TC) SPPU, 2019 Course

(22nd to 26th June 2020)

Dr. Manisha P. Dale
MES College of Engineering, Pune-01

Practical: Group B

Implement stack and queue using linked list.
Implement assignment 2 using files.
Add two polynomials using linked list.
Reverse a doubly linked list.
Evaluate postfix expression (input will be postfix expression)
Reverse and Sort stack using recursion.
Implement inorder tree traversal without recursion.
To find inorder predecessor and successor of a given key in BST.
Implement Quicksort.

What is File?

- File is a collection of bytes that is stored on secondary storage devices like disk. There are two kinds of files in a system. They are,

1. Text files (ASCII)
2. Binary files

- Text files contain ASCII codes of digits, alphabetic and symbols.
- Binary file contains collection of bytes (0's and 1's). Binary files are compiled version of text files.
- **How is a file stored?**
- – **Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).**

Operations on File

- **BASIC FILE OPERATIONS IN C PROGRAMMING:**
- There are 4 basic operations that can be performed on any files in C programming language. They are,

1. Opening/Creating a file
2. Closing a file
3. Reading a file
4. Writing in a file

Files in C

- In C, each file is simply a sequential stream of bytes. C imposes no structure on a file.
- A file must first be opened properly before it can be accessed for reading or writing. When a file is opened, a stream is associated with the file.
- Successfully opening a file returns a pointer to (i.e., the address of) a file

Syntax for FILE operations

File Operation	Declaration and Description
fopen() – To open a file	▪ Declaration: FILE *__fopen__ (const char *filename, const char *mode) ▪ fopen() function is used to open a file to perform operations such as reading, writing etc. ▪ In a C program, we declare a file pointer and use fopen() as below. fopen() function creates a new file if the mentioned file name does not exist. FILE *fp; fp=**fopen** (“filename”, ”‘mode”); ▪ Where, → fp – file pointer to the data type “FILE”. → filename – the actual file name with full path of -- the file. → mode – refers to the operation that will be performed on the file. Example: r, w, a, r+, w+ and a+.

File success check

- If the file was not able to be opened, then the value returned by the *fopen* routine is NULL.
- For example, let's assume that the file *mydata* does not exist. Then:

```
FILE *fptr1 ;
fptr1 = fopen ( "mydata", "r") ;
if (fptr1 == NULL)
{
printf ("File 'mydata' did not open.\n") ;
}
```

File pointers defined in stdio.h

Name	Notes
stdin	a pointer to a FILE which refers to the standard input stream, usually a keyboard.
stdout	a pointer to a FILE which refers to the standard output stream, usually a display terminal.
stderr	a pointer to a FILE which refers to the standard error stream, often a display terminal

File Open Modes

File Mode	Description
r	Open a text file for reading
w	Create a text file for writing, if it exists, it is overwritten.
a	Open a text file and append text to the end of the file.
rb	Open a binary file for reading
wb	Create a binary file for writing, if it exists, it is overwritten.
ab	Open a binary file and append data to the end of the file.

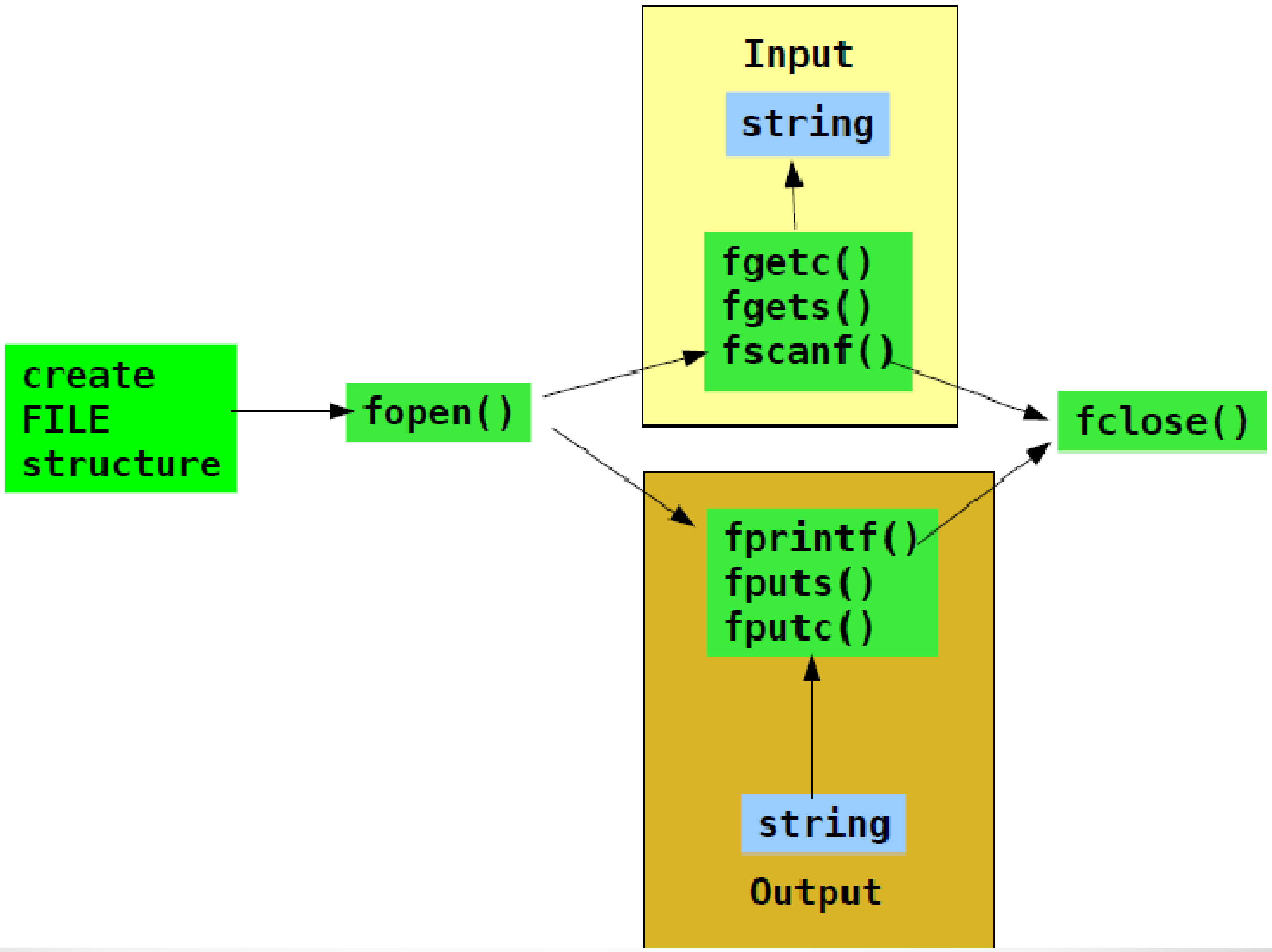
File Operations

File Operation	Declaration and Description

fclose() – To close a file	Declaration: **fclose**(FILE *fp);
	fclose() function closes the file that is being pointed by file pointer fp. In a C program, we close a file as below.
	fclose (fp);

- If a program terminates, it automatically closes all opened files. But it is a good programming habit to close any file once it is no longer needed.
- This helps in better utilization of system resources, and is very useful when you are working on numerous files simultaneously.
- Some operating systems place a limit on the number of files that can be open at any given point in time.

Text File I/O


Input
string
fgetc()
fgets()
fscanf()
create FILE structure
fopen()
fclose()
fprintf()
fputs()
fputc()
string
Output

fscanf() & fprintf()

- **fscanf() and fprintf()**
- The functions *fprintf()* and *fscanf()* are similar to *printf()* and *scanf()* except that these functions operate on files and require one additional and first argument to be a file pointer.

Use of fscanf() & fprintf()

```
#include <stdio.h>
main ()
{
FILE *fp;
float total;
fp = fopen("data.txt", "w+");
if (fp == NULL) {
printf("data.txt does not exist, please check!\n");
exit (1);
}
fprintf(fp, 100);
fscanf(fp, "%f", &total);
fclose(fp);
printf("Value of total is %f\n", total);
}
```

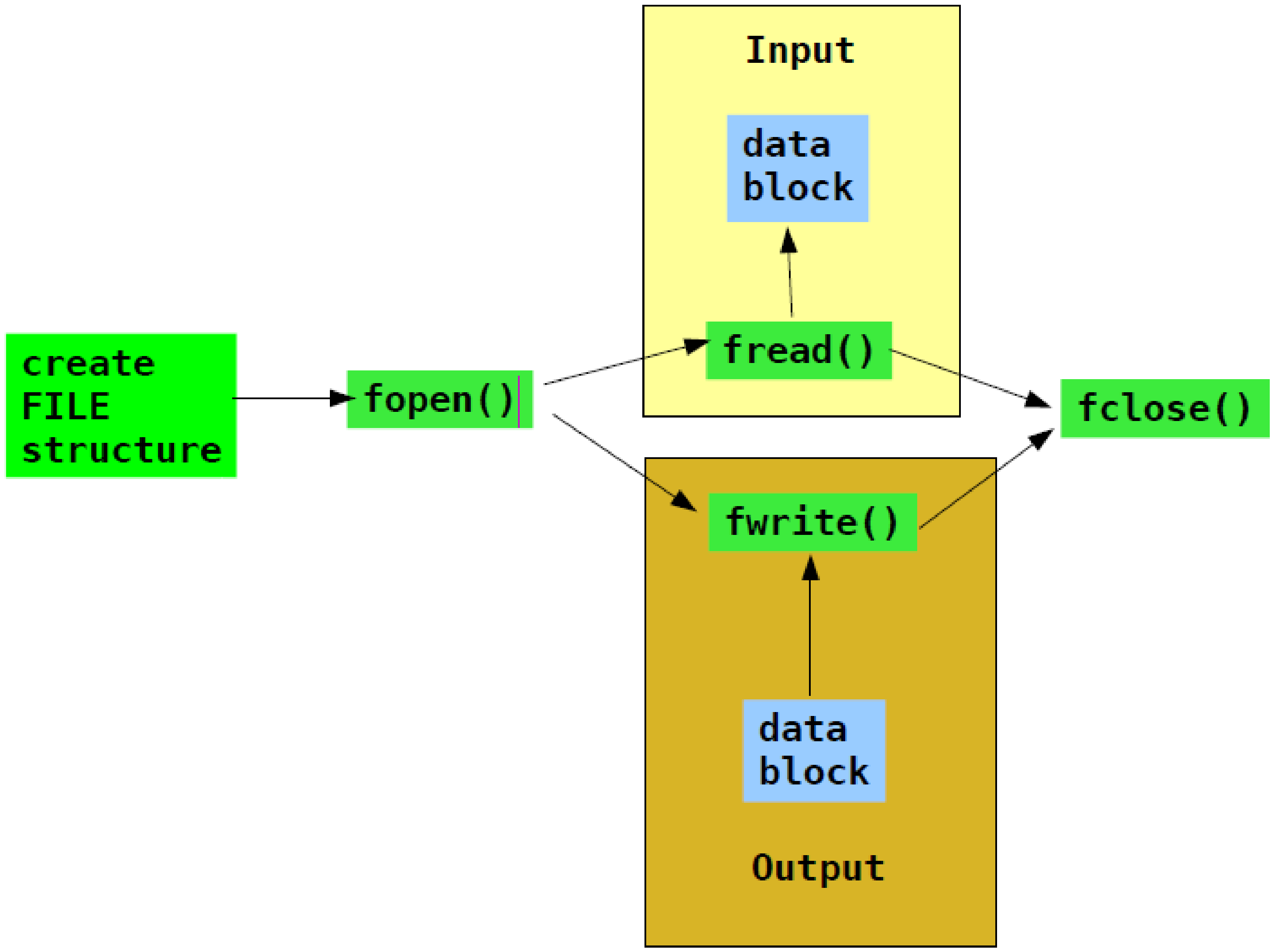
getc() & putc()

- The functions *getc()* and *putc()* are equivalent to *getchar()* and *putchar()* functions, except that these functions require an argument which is the file pointer.
- Function *getc()* reads a single character from the file which has previously been opened using a function like *fopen()*.
- Function *putc()* does the opposite, it writes a character to the file identified by its second argument. The format of both functions is as follows :

  → getc(in_file);

  → putc(c, out_file);
- Note: The second argument in the *putc()* function must be a file opened in either write or append mode.

Binary File I/O



fread() & fwrite()

- The functions *fread()* and *fwrite()* are a somwhat complex file handling functions used for reading or writing chunks of data containing NULL characters ('\0') terminating strings.

→ size_t fread(void *ptr, size_t sz, size_t n, FILE *fp)

→ size_t fwrite(const void *ptr, size_t sz, size_t n, FILE *fp);

fread() & fwrite()

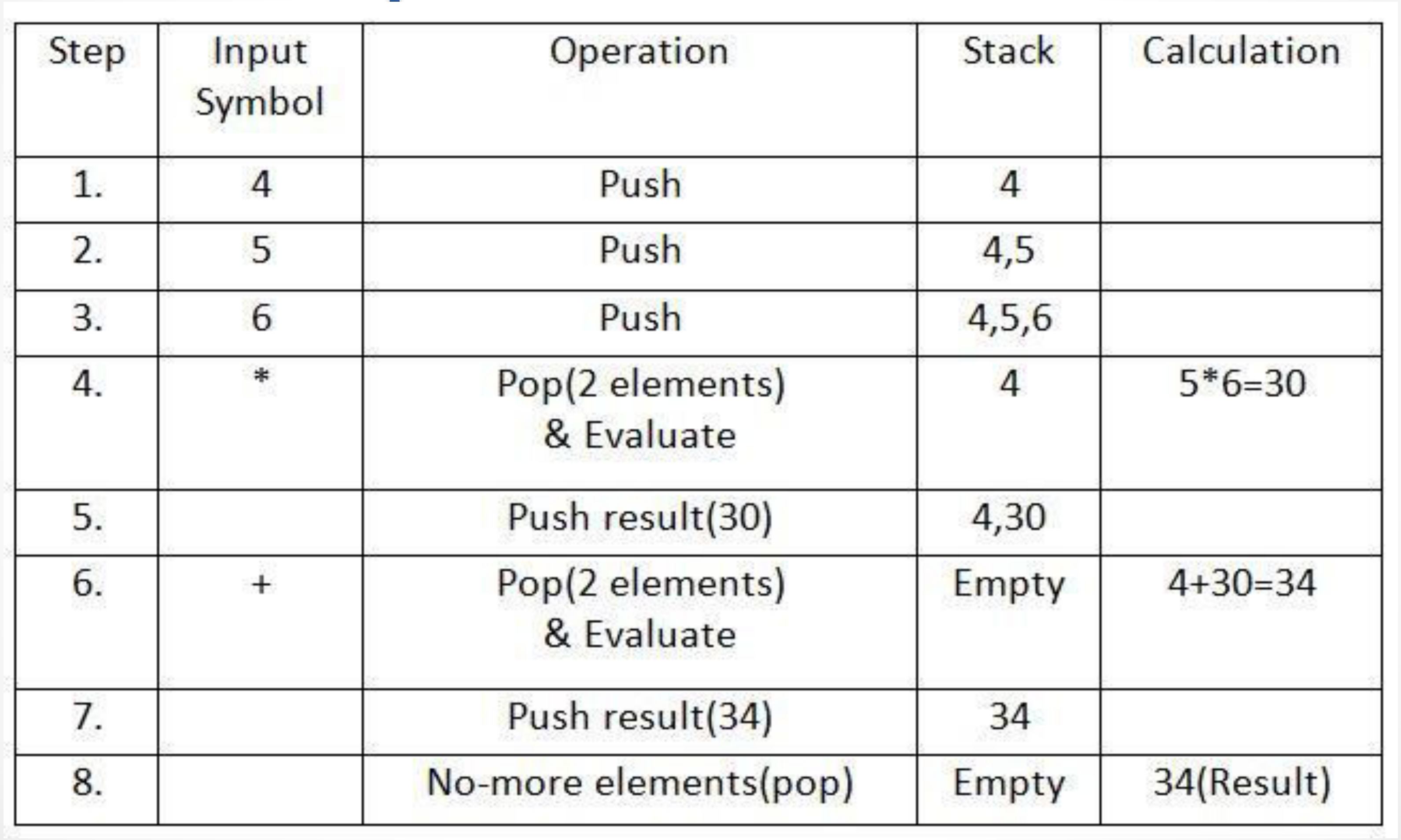
- size_t fread(void *ptr, size_t sz, size_t n, FILE *fp)

- Notice that the return type of *fread()* is size_t which is the number of items read.
- It reads *n* items, each of size *sz* from a file pointed to by the pointer *fp* into a buffer pointed by a void pointer *ptr* which is nothing but a generic pointer.
- Function *fread()* reads it as a stream of bytes and advances the file pointer by the number of bytes read.

Evaluation of Postfix Expression

- **Evaluation** rule of a **Postfix Expression** states:
- *While reading the **expression** from left to right, push the element in the stack if it is an operand.*
- *Pop the two operands from the stack, if the element is an operator and then **evaluate** it.*
- *Push back the result of the **evaluation***

Example: 4+5*6 → 456*+



Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)

Algorithm

```
1) Add ) to postfix expression.
2) Read postfix expression Left to Right
until ) encountered
3) If operand is encountered, push it onto Stack
[End If]
4) If operator is encountered, Pop two elements
i) A -> Top element
ii) B-> Next to Top element
iii) Evaluate B operator A
push B operator A onto Stack
5) Set result = pop
6) END
```

Thank You!!!

Dr. Manisha P. Dale
mpdale@mescoepune.org
9422362809