# CS101C: Introduction to Programming (Using C)

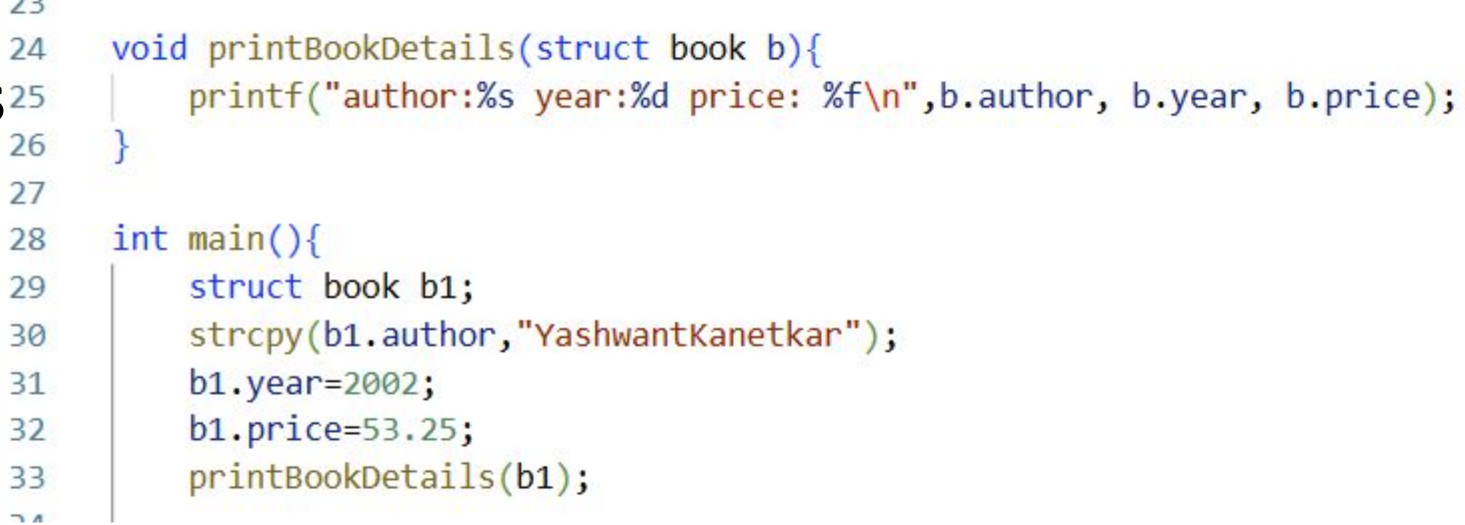
Autumn 2025

Nikhil Hegde
Achyut Mani Tripathi

## Week11: Structures (continued), FileIO

## Last class

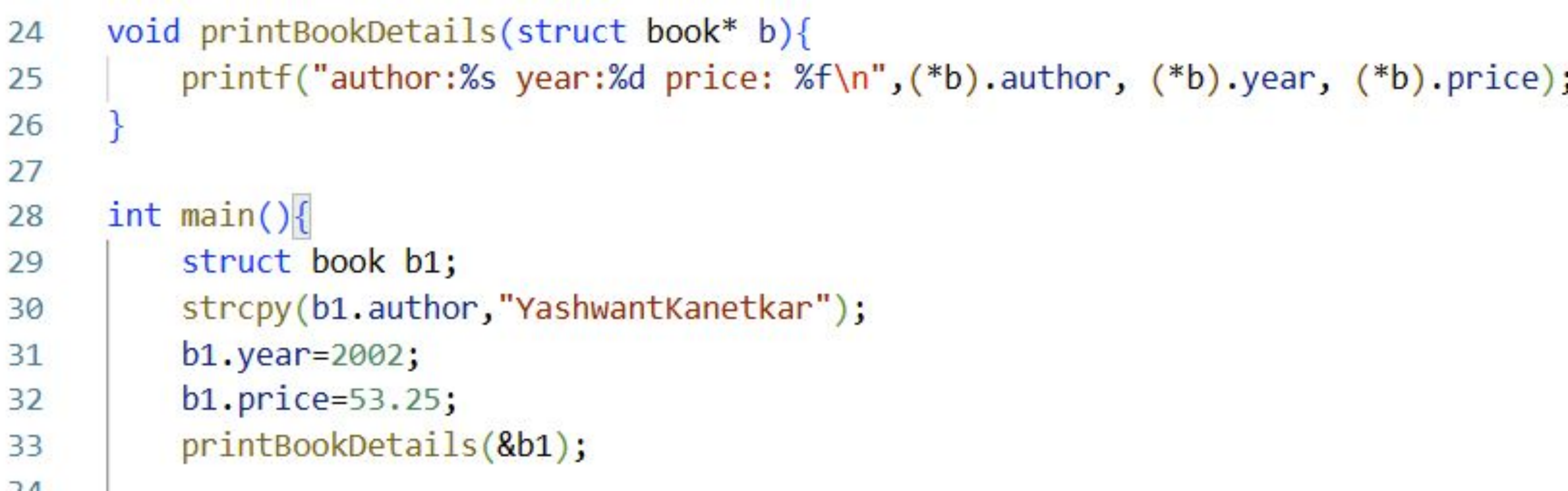
Pass-by-val



```c
void printBookDetails(struct book b){
    printf("author:%s year:%d price: %f\n",b.author, b.year, b.price);
}

int main(){
    struct book b1;
    strcpy(b1.author,"YashwantKanetkar");
    b1.year=2002;
    b1.price=53.25;
    printBookDetails(b1);
```

pass-by-ref



```c
void printBookDetails(struct book* b){
    printf("author:%s year:%d price: %f\n",(*b).author, (*b).year, (*b).price);
}

int main(){
    struct book b1;
    strcpy(b1.author,"YashwantKanetkar");
    b1.year=2002;
    b1.price=53.25;
    printBookDetails(&b1);
```

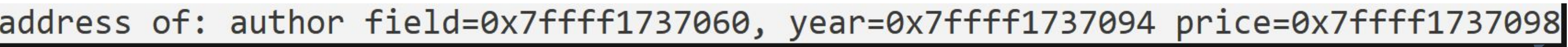
Why pass-by-ref is good ?

## Last class

Printing address of structure fields

```c
struct book{
        char author[50];
        int year;
        float price;
};
```

```c
printf("address of: author field=%p, year=%p price=%p\n",
&(b1.author),&(b1.year), &(b1.price));
```



```
address of: author field=0x7ffff1737060, year=0x7ffff1737094 price=0x7ffff1737098
```

Gap between ..60 and ..94 = 0x34 bytes = 52 bytes.   Why? author = 50 bytes + 2 bytes padding.

Gap between ..94 and ..98 = 0x4 bytes = 4 bytes.   Why? Size of year = 4 bytes

# Last class

Typedef

A way to rename existing types.

Syntax: `typedef existing_type_name new_type_name`

Example:

```
typedef struct book{
    char author[50];
    int year;
    float price;
}book;
```

```
book b1; //no need of struct book b1;
```

# File Input Output

Return type: pointer to FILE type. Called file handle.

Argument1: File name (string)

Argument2: Mode (string)

```
#include<stdio.h>

int main(){

        FILE * filehandle=fopen("book1.txt","r");
        if(filehandle != NULL){
          printf("value of filehandle:%p\n", filehandle);
          fclose(filehandle);
   }       //automatically called when program terminates
}
```

fopen: function used to open the file.

fclose: function used to close a file.

# File Input Output

Return type: pointer to FILE type. Called file handle.

Argument1: File name (string)

Argument2: Mode (string)

```
#include<stdio.h>

int main(){

        FILE * filehandle=fopen("book1.txt","r");
        if(filehandle != NULL){
          printf("value of filehandle:%p\n", filehandle);
          fclose(filehandle);
        }
}
```

fopen: when the mode is read (i.e. <u>“r”</u>) and <u>if file does not exist</u>, 0 is <u>returns 0</u> (NULL is a macro for 0. We’ll see macros next.)

# File Input Output

Return type: pointer to FILE type. Called file handle.

Argument1: File name (string)

Argument2: Mode (string)

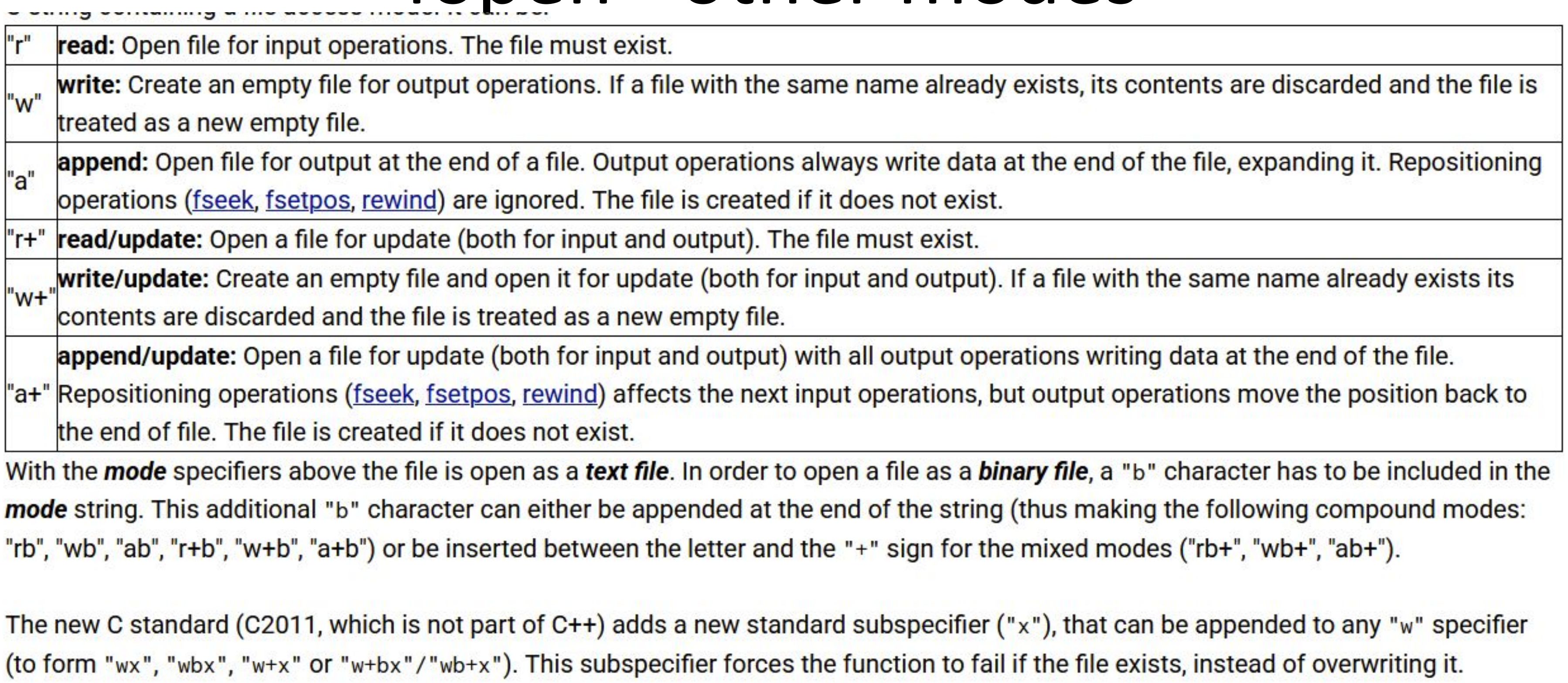
```
#include<stdio.h>

int main(){

        FILE * filehandle=fopen("book1.txt","r");
        if(filehandle != NULL){
          printf("value of filehandle:%p\n", filehandle);
          fclose(filehandle);
        }
}
```

Other modes are “w” (write), “a” (append). You can also have “rb”, “wb”,”ab” to indicate a binary file.

# fopen - other modes



| | |
|---|---|
| "r" | **read:** Open file for input operations. The file must exist. |
| "w" | **write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file. |
| "a" | **append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist. |
| "r+" | **read/update:** Open a file for update (both for input and output). The file must exist. |
| "w+" | **write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file. |
| "a+" | **append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist. |

With the ***mode*** specifiers above the file is open as a ***text file***. In order to open a file as a ***binary file***, a `"b"` character has to be included in the ***mode*** string. This additional `"b"` character can either be appended at the end of the string (thus making the following compound modes: "rb", "wb", "ab", "r+b", "w+b", "a+b") or be inserted between the letter and the `"+"` sign for the mixed modes ("rb+", "wb+", "ab+").

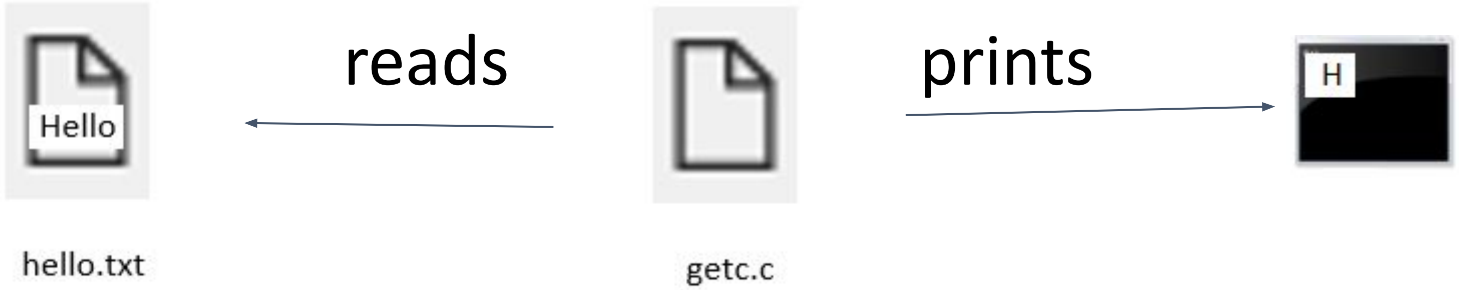
The new C standard (C2011, which is not part of C++) adds a new standard subspecifier (`"x"`), that can be appended to any `"w"` specifier (to form `"wx"`, `"wbx"`, `"w+x"` or `"w+bx"`/`"wb+x"`). This subspecifier forces the function to fail if the file exists, instead of overwriting it.

source: https://cplusplus.com/reference/cstdio/fopen/

# getc

```
int getc(FILE* fp)
```

- reads a single character from the file <u>stream.</u>
- returns the character read or EOF on error/end of file.

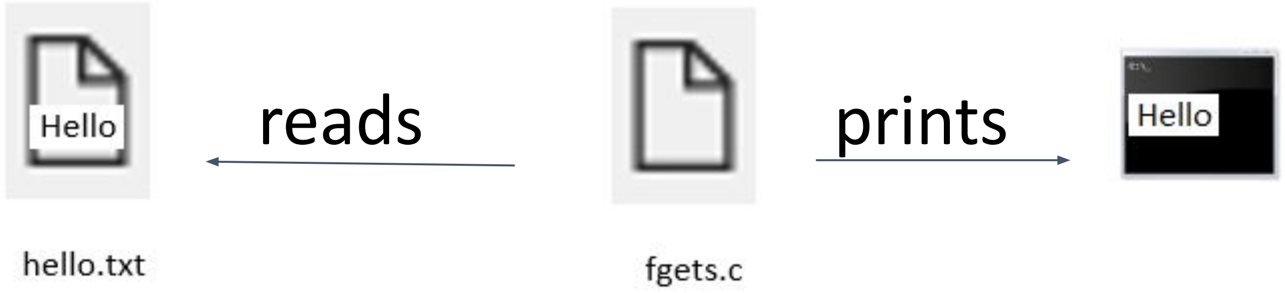


see getc.c in codeexamples.

# fgets

```
char* fgets(char line [], int maxlen,FILE* fp)
```

- reads a single line (upto maxlen characters) from the input stream (including newline / line break).
- returns a pointer to the character array that stores the line

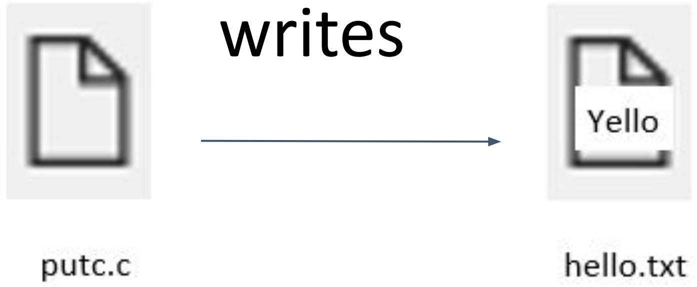


see fgets.c in codeexamples.

# putc

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

- writes a single character c to the output stream.
- returns the character written or EOF on error.

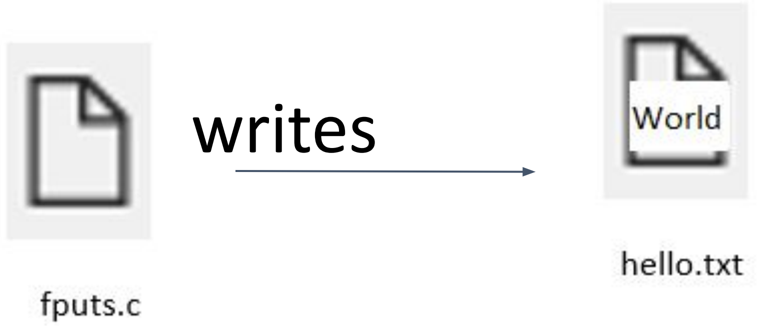


see putc.c in codeexamples.

# fputs

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

- writes a single line to the output stream.
- returns non-negative number on success, EOF otherwise.



see fputs.c in codeexamples.

# fscanf

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

- similar to scanf
- reads input from file itemwise
- returns number of arguments matched

see fscanf.c in codeexamples.

# fprintf

```
int fprintf(FILE* fp, const char* str...)
```

- similar to printf.
- Writes to file.
- Returns number of bytes (characters) written to file.

see fprintf.c in codeexamples.