C File Handling

Array of pointers to structures

Searching & Sorting

FILE

- A collection of data or information that are stored on a computer known as file.
- A file is a collection of bytes stored on a secondary storage device.
- There are four different types of file
  - Data files
  - Text files
  - Program files
  - Directory files
- Different types of file store different types of information

FILE

- A file represents a sequence of bytes on the disk where a group of related data is stored.
- File is created for permanent storage of data. It is a ready made structure.
- File handing is used to manage files in secondary memory, by using file handling function, we can create, read, remove, copy etc files in/from secondary memory.
- To perform complete file operations, we use file handling.
- A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard.

- So Files are used for storing information that can be processed by the programs. Files are not only used for storing the data, programs are also stored in files.
- In order to use files, we have to learn file input and output operations. That is, how data is read and how to write into a file.
- A Stream is the important concept in C. The Stream is a common, logical interface to the various devices that comprise the computer. A stream refers to the flow of data (in bytes) from one place to another (from program to file or vice-versa).
- So a Stream is a logical interface to a file. There are two types of Streams, Text Stream and Binary Stream.

TYPE OF FILES (FILE HANDLING)

- There are two types of files, which can be handled through C programming language:
  - Character Oriented File Handling/Text files
  - Binary Oriented File Handling/Binary files
- For file handling, we use a structure named "FILE" which is declared in **stdio.h header file**.

FILE STREAMS-TEXT STREAM

- A Text File can be thought of as a stream of characters that can be processed sequentially. It can only be processed in the forward direction.
- It consists of sequence of characters.
- Each line of characters in the stream may be terminated by a newline character.
- Text streams are used for textual data, which has a consistent appearance from one environment to another or from one machine to another
- A text file can be a stream of characters that a computer can process sequentially.
- It is processed only in forward direction.
- It is opened for one kind of operation (reading, writing, or appending) at any give time.
- We can read only one character at a time from a text file.

FILE STREAMS-BINARY STREAM

- Binary streams are primarily used for non-textual data, which is required to keep exact contents of the file.
- A binary file is a file consisting of collection of bytes.
- A binary file is also referred to as a character stream

Using Files in C

- To use a file four essential actions should be carried out.
- These are
  - Declare a file pointer variable.
  - Open a file using the fopen() function.
  - Process the file using suitable functions.
  - Close the file using the fclose() and fflush() functions.

Declaration of file pointer

- A pointer variable is used to points a structure FILE.
- The members of the FILE structure are used by the program in various file access operation, but programmers do not need to concerned about them.
- FILE *file_pointer_name;
- Eg : FILE *fp;

DECLARATION OF FILE POINTER:

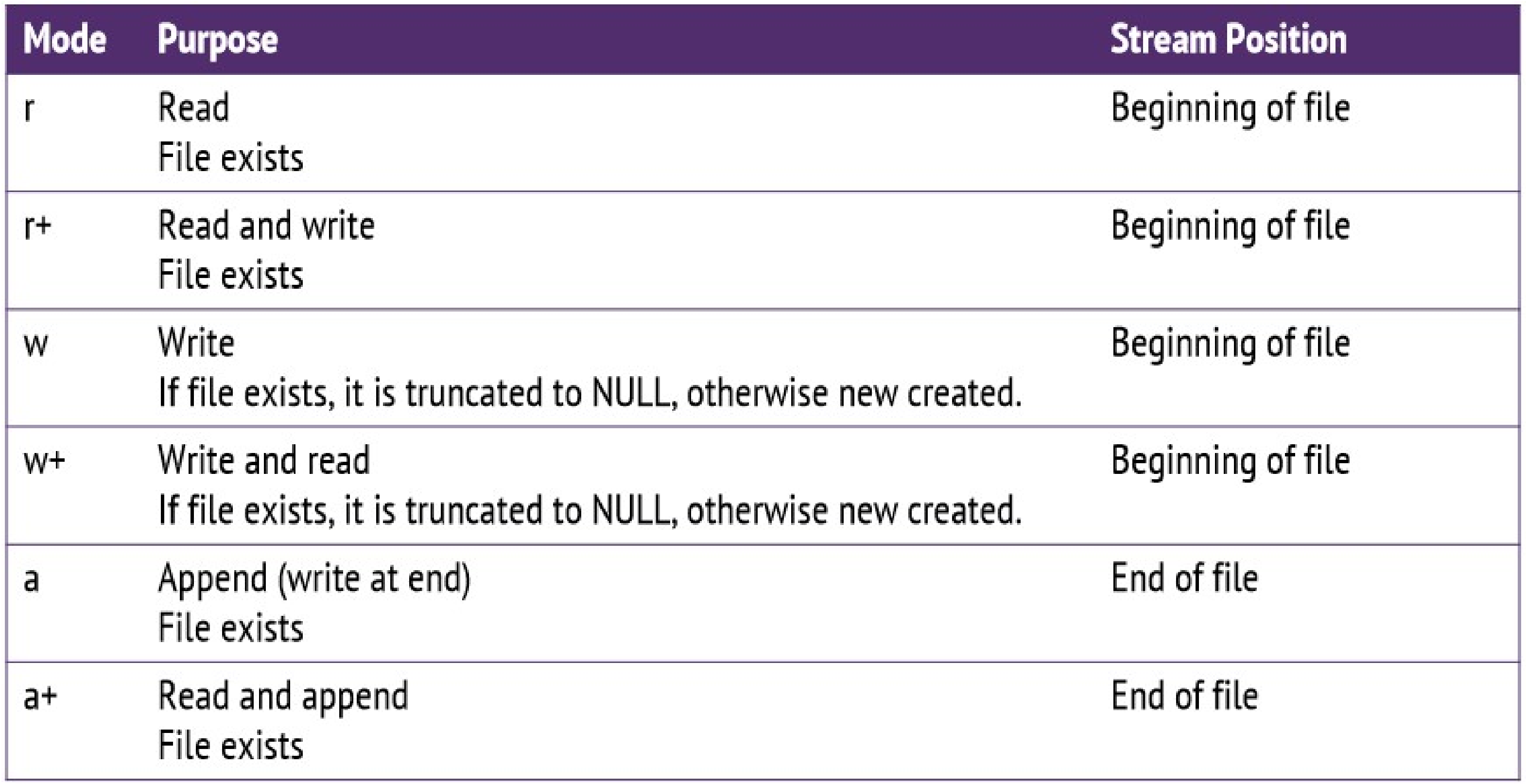
- Opening a file To open a file using the fopen() function.
- Its syntax is,

  FILE *fopen(const char *fname,const char* mode);
- const char *fname represents the file name.
- The file name is like **“D:\\501\example.txt”.**
- Here D: is a drive, 501 is the directory,

  example.txt is a file name.

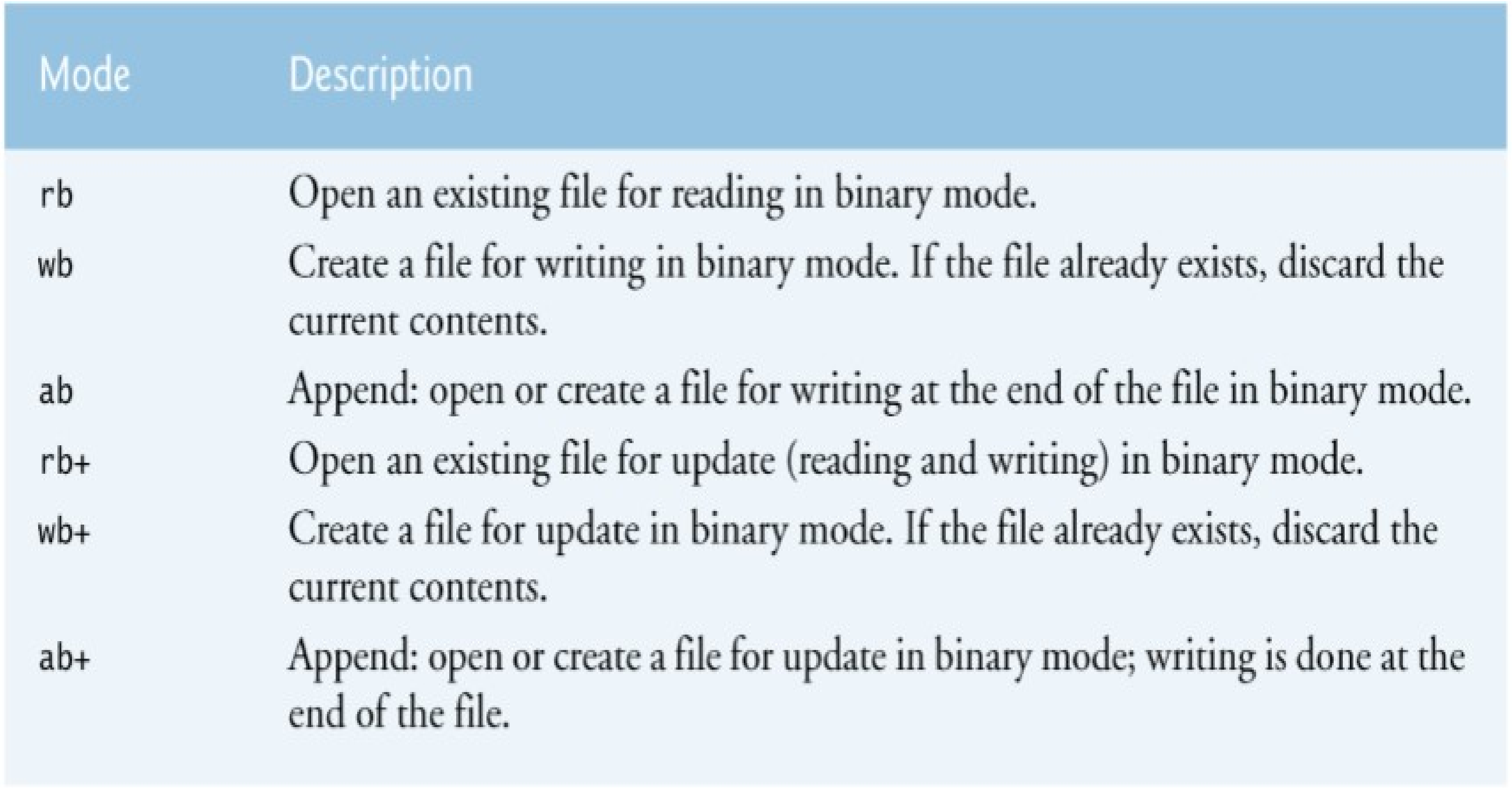
  const char *mode represents the mode of the file.

OPEN THE FILE: FOPEN()



Mode	Purpose	Stream Position
r	Read File exists	Beginning of file
r+	Read and write File exists	Beginning of file
w	Write If file exists, it is truncated to NULL, otherwise new created.	Beginning of file
w+	Write and read If file exists, it is truncated to NULL, otherwise new created.	Beginning of file
a	Append (write at end) File exists	End of file
a+	Read and append File exists	End of file

Opening Binary Files

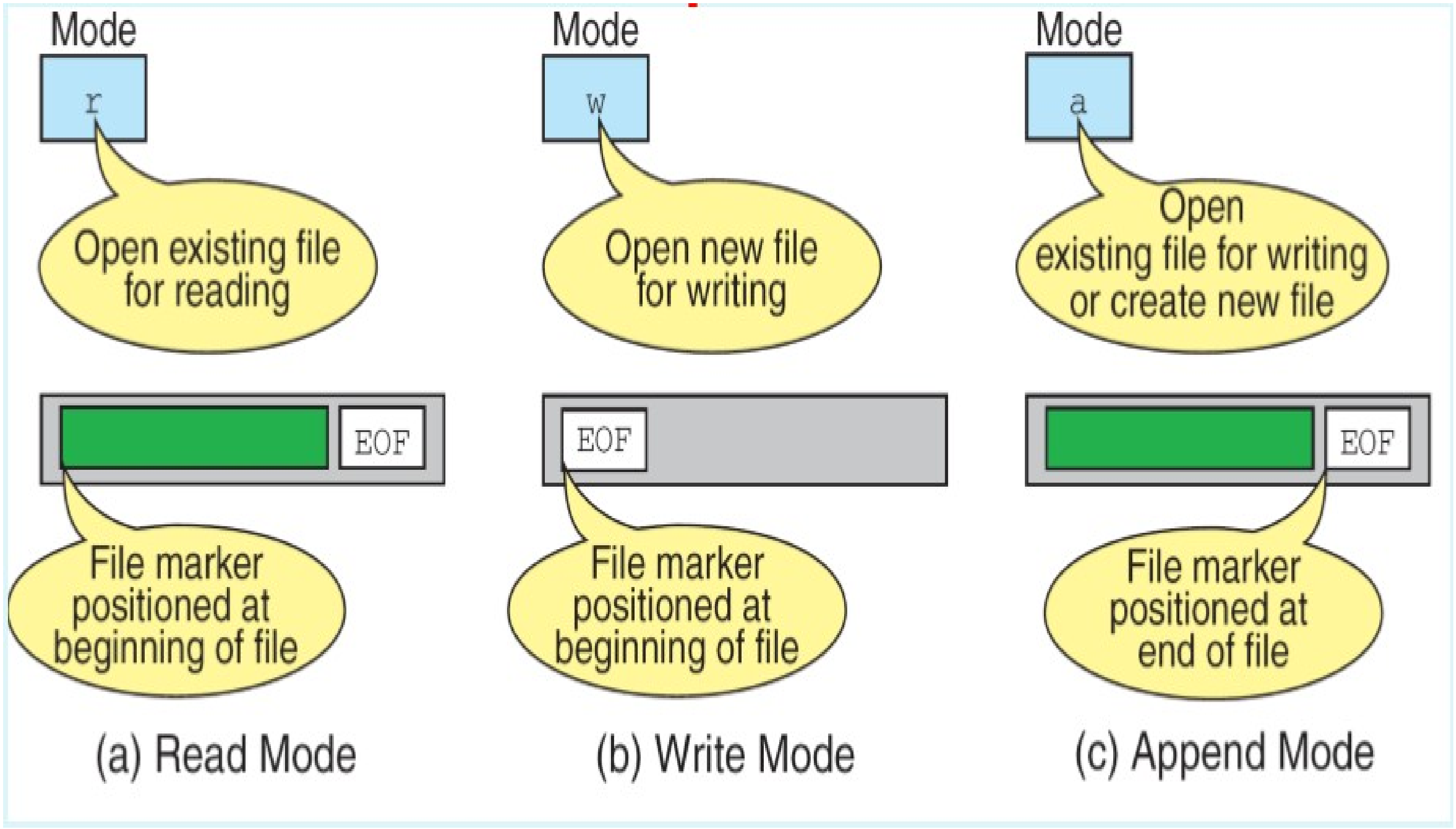


Mode	Description
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append: open or create a file for writing at the end of the file in binary mode.
rb+	Open an existing file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append: open or create a file for update in binary mode; writing is done at the end of the file.

Difference between write mode and append mode

- Write (w) mode and Append (a) mode, while opening a file are almost the same.
- Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.
- The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file.
- While in append mode this will not happen. Append mode is used to append or add data to the existing data of file(if any).
- Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file

READ/WRITE/APPEND MODE

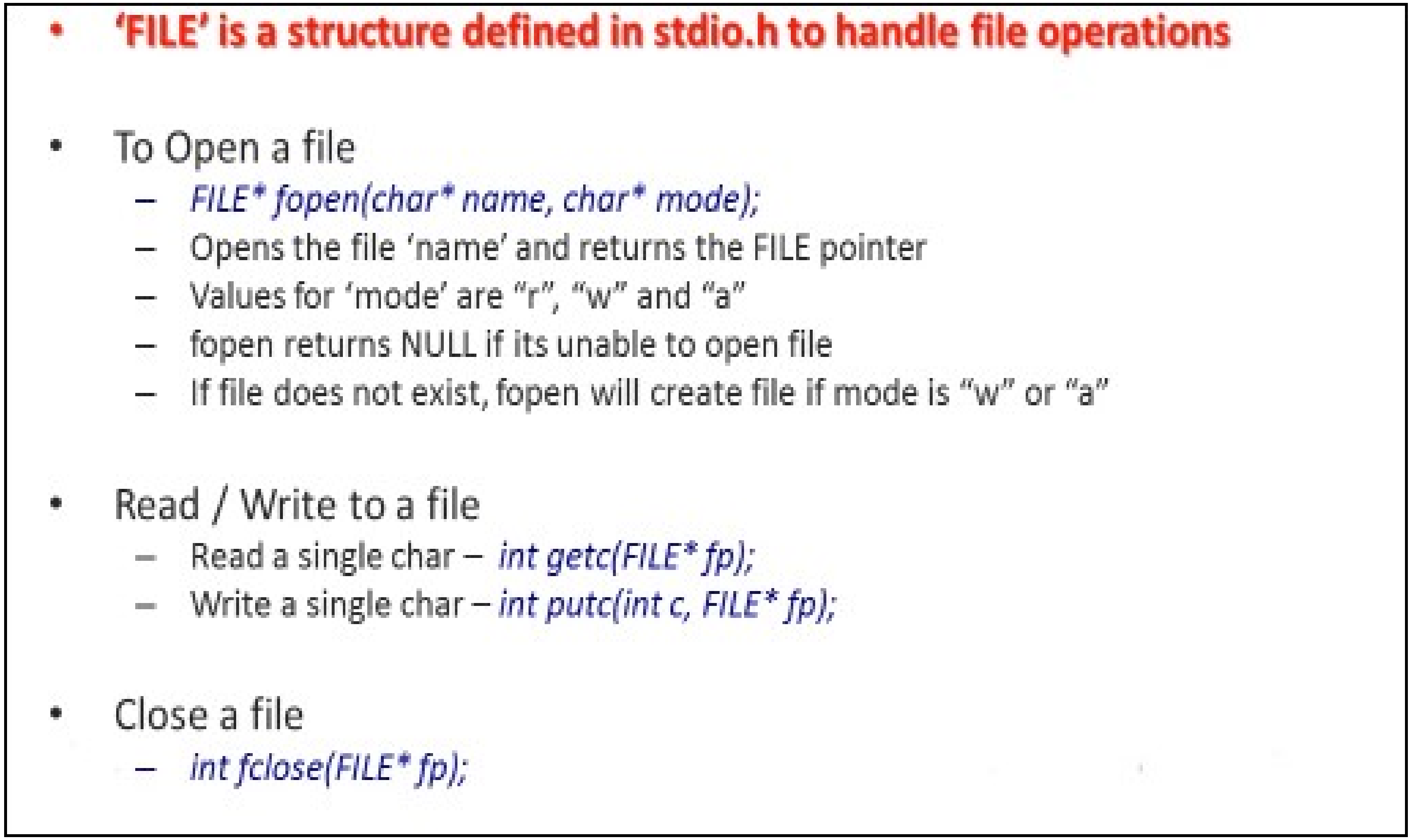

Mode
r
Open existing file for reading
EOF
File marker positioned at beginning of file
(a) Read Mode
Mode
w
Open new file for writing
EOF
File marker positioned at beginning of file
(b) Write Mode
Mode
a
Open existing file for writing or create new file
EOF
File marker positioned at end of file
(c) Append Mode

CLOSING AND FLUSHING FILES

- The file must be closed using the fclose() function.
- Its prototype is , **int fclose(FILE *fp);**
- The argument fp is the FILE pinter associated with the stream.
- Fclose() returns 0 on success and -1 on error.
- fflush() used when a file's buffer is to be written to disk while still using the file.
- Use of fflushall() to flush the buffers of all open streams.
- The prototype of these two functions are,

    int flushall(void);

    int fflush(FILE *fp);



- **'FILE' is a structure defined in stdio.h to handle file operations**

- To Open a file
  - *FILE* fopen(char* name, char* mode);*
  - Opens the file 'name' and returns the FILE pointer
  - Values for 'mode' are "r", "w" and "a"
  - fopen returns NULL if its unable to open file
  - If file does not exist, fopen will create file if mode is "w" or "a"

- Read / Write to a file
  - Read a single char – *int getc(FILE* fp);*
  - Write a single char – *int putc(int c, FILE* fp);*

- Close a file
  - *int fclose(FILE* fp);*

CHARACTER INPUT / OUTPUT

- Two functions are used to read the character from the file and write it into another file.
- These functions are getc() & putc() functions.
- Its prototype is,

int getc(FILE *file_pointer);

putc(char const_char, FILE *file_pointer);

DETECTING THE END OF A FILE

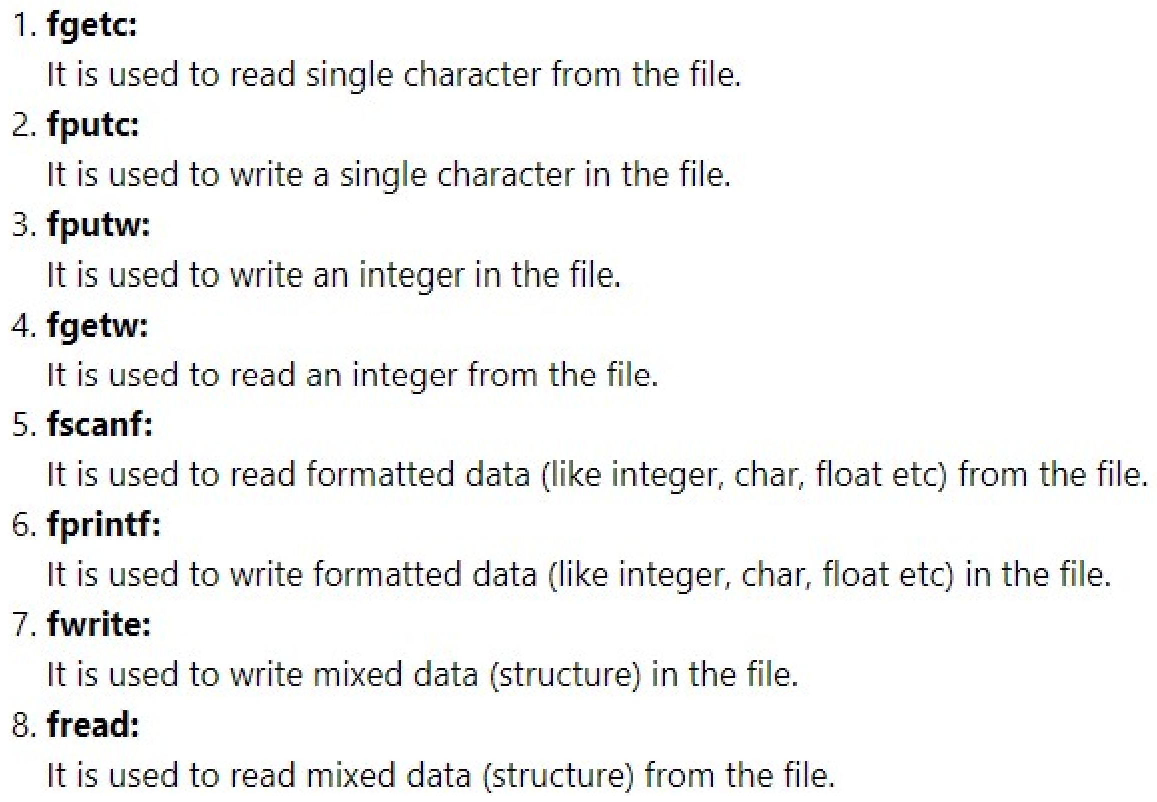
- When reading from a text-mode file character by character, one can look for the end-of-file character.
- The symbolic constant EOF is defined in stdio.h as -1, a value never used by a real character.

  Eg: while((c=getc(fp)!=EOF).
- This is the general representation of the End Of the File

WORKING WITH TEXT FILES

- C provides various functions to working with text files. Four functions can be used to read text files and four functions that can be used to write text files into a disk.
- These are,
  - fscanf() & fprintf()
  - fgets() & fputs()
  - fgetc() & fputc()
  - fread() & fwrite().

FILE HANDLING FUNCTIONS:



1. **fgetc:**
   It is used to read single character from the file.
2. **fputc:**
   It is used to write a single character in the file.
3. **fputw:**
   It is used to write an integer in the file.
4. **fgetw:**
   It is used to read an integer from the file.
5. **fscanf:**
   It is used to read formatted data (like integer, char, float etc) from the file.
6. **fprintf:**
   It is used to write formatted data (like integer, char, float etc) in the file.
7. **fwrite:**
   It is used to write mixed data (structure) in the file.
8. **fread:**
   It is used to read mixed data (structure) from the file.

WORKING WITH TEXT FILES

- The prototypes of the above functions are,

  1. int fscanf(FILE *stream, const char *format,list);

  2. int getc(FILE *stream);

  3. char *fgets(char *str,int n,FILE *fp);

  4. int fread(void *str,size_t size,size_t num, FILE *stream);

  5. int fprintf(FILE *stream, const char *format,list);

  6. int fputc(int c, FILE *stream);

  7. int fputs(const char *str,FILE *stream);

  8. int fwrite(const void *str,size_t size, size_t count,FILE *stream);

WORKING WITH BINARY FILES

- A Binary file is similar to the text file, but it contains only large numerical data.
- Following steps are suited for copying a binary file into another and as follows,

1. Open the source file for reading in binary mode.
2. Open the destination file for writing in binary mode.
3. Read a character from the source file. Remember, when a file is first opened, the pointer is at the start of the file, so there is no need to position the file pointer explicitly.
4. If the function feof() indicates that the end of thesource file has been reached, then close both files and return to the calling program.
5. If end-of- file has not been reached, write the character to the destination file, and then go to step 3.

DIRECT FILE INPUT AND OUTPUT

- Direct input and output is only with binary-mode files.
- With direct output, blocks of data are written from the memory to disk. Direct input reverses the process.
- A block of data is read from a disk file into memory. fread() and fwrite() functions are used to read and write the binary files on direct I/O

OTHER FILE MANAGEMENT FUNCTIONS

- The copy and delete operations are also associated with file management.
- Though one could write programs for them, the C standard library contains functions for deleting and renaming files.
- **Deleting a file :** The library function remove() is used to delete a file. Its prototype is,

  int remove(const char *filename);

- **Renaming a file** : The rename() function changes the name of an existing disk file. Its prototype is,

  int rename(const char *oldname, const char *newname);

FUNCTIONS TO READ AND WRITE DATA TO FILE

- Function fgetc
  - Like getchar, reads one character from a file.
  - Receives as an argument a FILEpointer for the file from which a character will be read.
  - The call fgetc(stdin)reads one character from stdin—the standard input.
- Function fputc,
  - Like putchar, writes one character to a file.
  - Receives as arguments a character to be written and a pointer for the file to which the character will be written

Functions to read and write data to file

- Function fgets
  - Reads one line from a file.
  - char *fgets(char *str, int n, FILE *stream)
- Function fputs
  - Writes one line to a file.
  - int fputs(const char *str, FILE *stream)

Functions to read and write data to file

- Function fprintf
  - Like printf
  - Takes first argument as file pointer
- Function fscanf
  - Like scanf
  - Takes first argument as file pointer

CLOSE THE FILE

- fclose()

  int fclose(FILE * stream)

  - Returns 0 if successfully closed
  - If function fclose is not called explicitly, the operating system normally will close the file when program execution terminates.

FWRITE()

- Example use

fprintf(fPtr, "%d", number);

- could print a single digit or as many as 11 digits (10 digits plus a sign, each of which requires 1 byte of storage)
- For a four-byte integer, we can use

fwrite(&number, sizeof(int), 1, fPtr);

- which always writes four bytes on a system with fourbyte integers from a variable number to the file represented by fPtr.
- 1 denotes one integer will be written.

FREAD()

- Function fread reads a specified number of bytes from a file into memory.
- For example,

```
fread(&client, sizeof(struct clientData), 1, cfPtr);
```

- Reads the number of bytes determined by sizeof(struct clientData) from the file referenced by cfPtr,
- Stores the data in client and returns the number of bytes read.
- The bytes are read from the location specified by the file position pointer.

Errors in fopen

- If an error occurs in opening a file ,then fopen() returns NULL.

```
FILE *p;

 p=fopen(“abc.txt”, ”r”);

if(p==NULL)

 {

printf(“Error in opening file”);

exit(1);

 }
```

ERRORS MAY OCCUR DUE TO FOLLOWING REASONS

- If we try to open a file in read mode and If the file doesn’t exists or we do not have read permission on that file.
- If we try to create a file but there is no space on disk or we don’t have write permissions.
- If we try to create a file that already exists and we don’t have permission to delete that file.
- Operating system limits the number of files that can be opened at a time and we are trying to open more files than that number.

MOVING TO A LOCATION

- fseek

  int fseek(FILE *stream, long int offset, int whence);

  - Offset is the number of bytes to seek from
  - whence in the file pointed to by stream—a positive offset seeks forward and a negative one seeks backward.

- Argument whence is one of the values
  - SEEK_SET: Value 0, beginning of file.
  - SEEK_CUR: Value 1, current position.
  - SEEK_END: Value 2, end of file.

Moving to a location

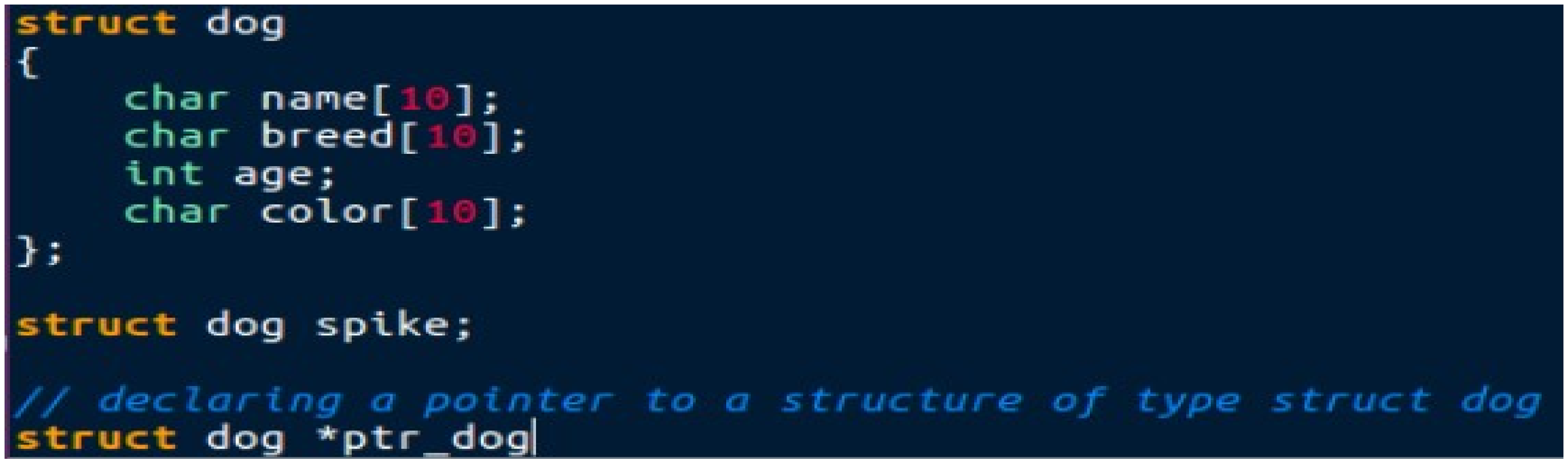
- ftell()

long int ftell(FILE *stream) ;

- The ftell() function returns the current file position of the specified stream.
- ftell() function is used to get the total size of a file after moving file pointer at the end of file.
- SEEK_END constant can be used to move the file pointer at the end of file.

ARRAY OF POINTERS TO STRUCTURES

- We have already learned that a pointer is a variable which points to the address of another variable of any data type like int, char, float etc.
- Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable.
- Here is how we can declare a pointer to a structure variable.


```
struct dog
{
     char name[10];
     char breed[10];
     int age;
     char color[10];
};

struct dog spike;

// declaring a pointer to a structure of type struct dog
struct dog *ptr_dog
```

- This declares a pointer ptr_dog that can store the address of the variable of type struct dog. We can now assign the address of variable spike to ptr_dog using & operator.

ptr_dog = &spike;

- Now ptr_dog points to the structure variable spike.

ACCESSING MEMBERS USING POINTER

- There are two ways of accessing members of structure using pointer:
- Using indirection (*) operator and dot (.) operator.
- Using arrow (->) operator or membership operator.

USING INDIRECTION (*) OPERATOR AND DOT (.) OPERATOR

- At this point ptr_dog points to the structure variable spike, so by dereferencing it we will get the contents of the spike. This means spike and *ptr_dog are functionally equivalent. To access a member of structure write *ptr_dog followed by a dot(.) operator, followed by the name of the member.
- For example:

  ptr_dog->name - refers to the name of dog

  (*ptr_dog).breed – refers to the breed of dog
- Parentheses around *ptr_dog are necessary because the precedence of dot(.) operator is greater than that of indirection (*) operator.

USING ARROW OPERATOR (->)

- The above method of accessing members of the structure using pointers is slightly confusing and less readable, that's why C provides another way to access members using the arrow (->) operator. To access members using arrow (->) operator write pointer variable followed by -> operator, followed by name of the member.

  ptr_dog->name - refers to the name of dog

  ptr_dog->breed - refers to the breed of dog

- Here we don't need parentheses, asterisk (*) and dot (.) operator. This method is much more readable and intuitive.

Using arrow operator (->)

- We can also modify the value of members using pointer notation.

strcpy(ptr_dog->name, "new_name");

- Here we know that the name of the array (ptr_dog->name) is a constant pointer and points to the 0th element of the array. So we can't assign a new string to it using assignment operator (=), that's why strcpy() function is used.

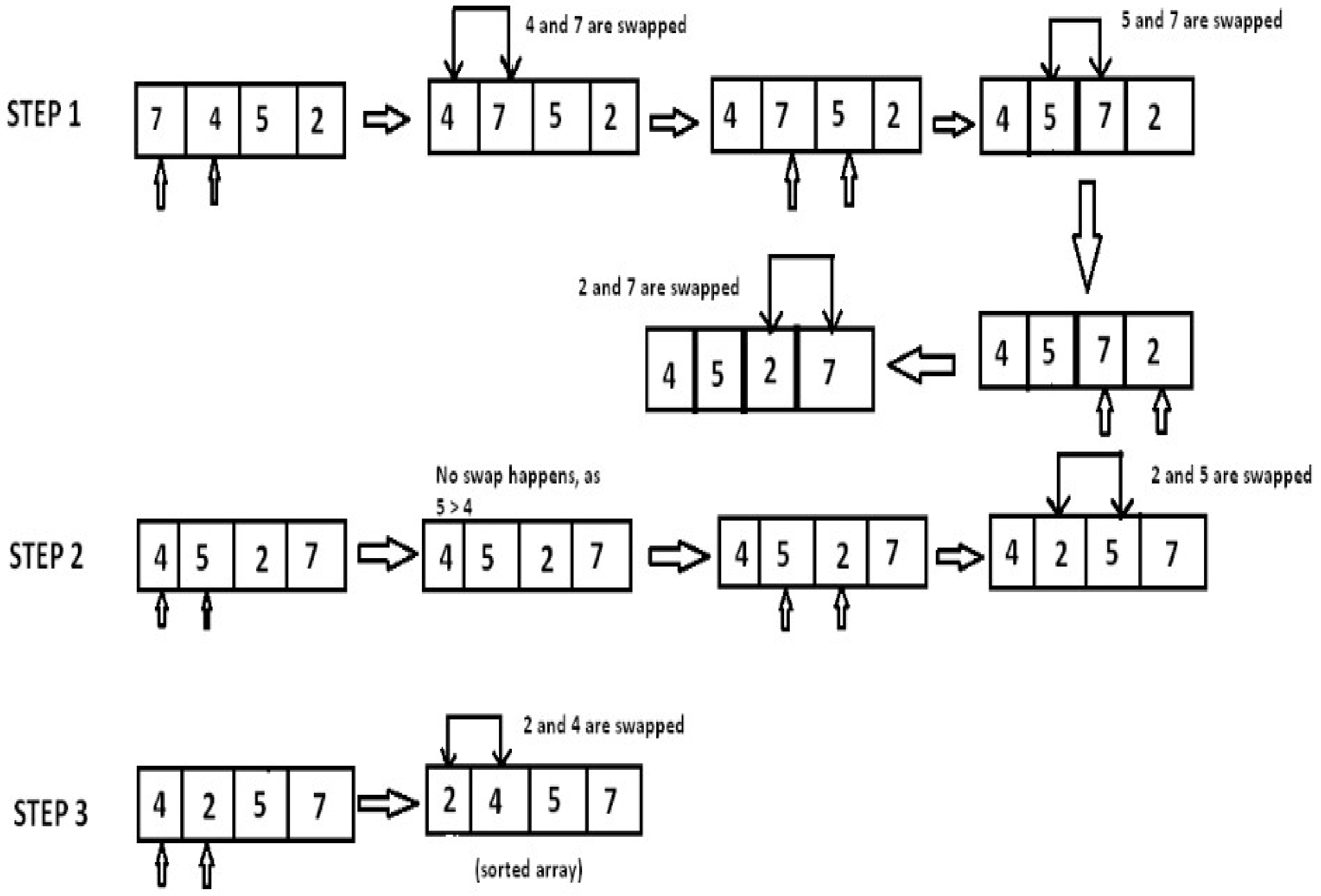
--ptr_dog->age;

- In the above expression precedence of arrow operator (->) is greater than that of prefix decrement operator (--), so first -> operator is applied in the expression then its value is decremented by 1.

Bubble Sort Algorithm

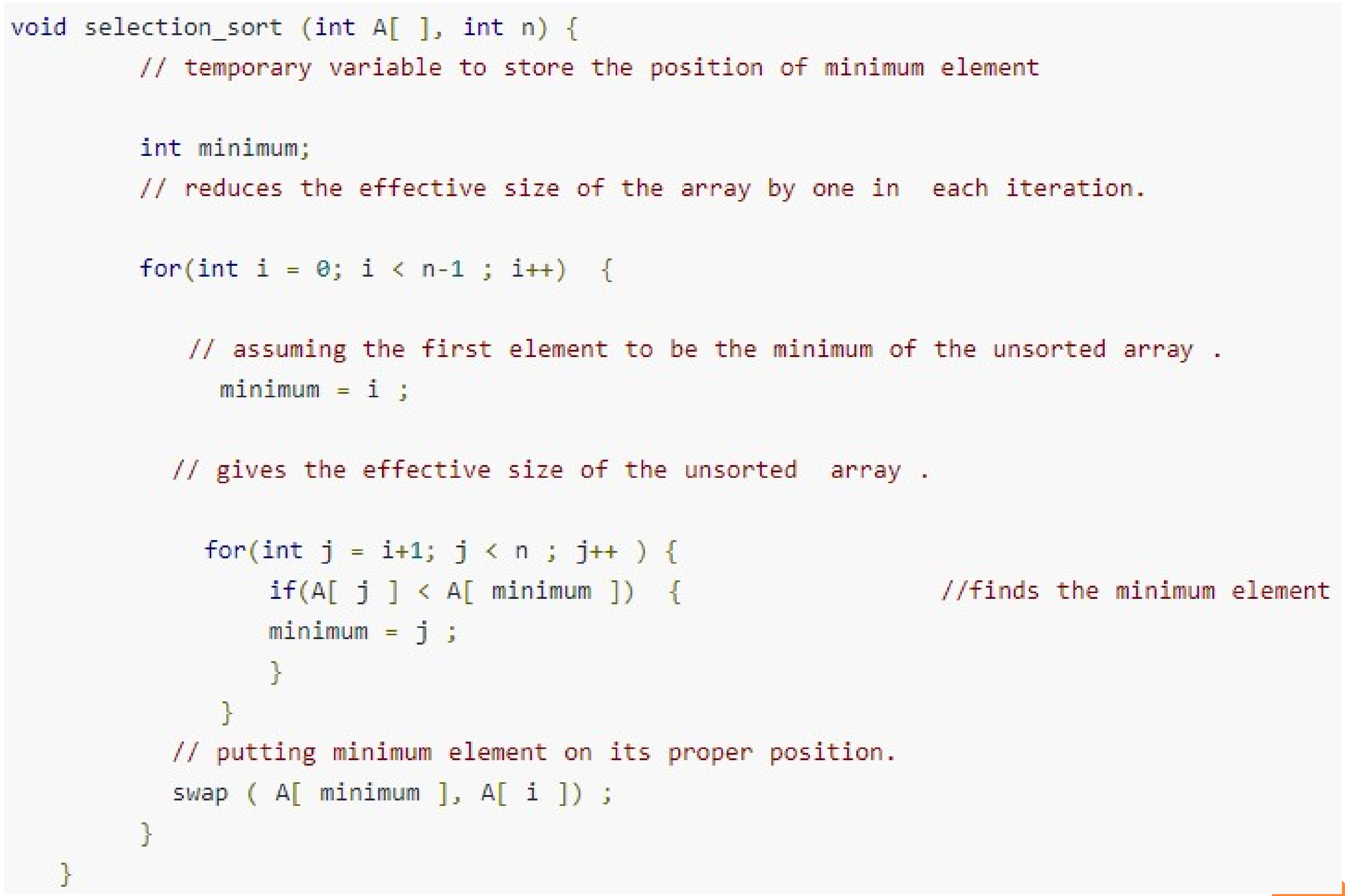
- Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.
- If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on.
- If we have total n elements, then we need to repeat this process for n-1 times.
- It is known as bubble sort, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.
- Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

```
void bubble_sort( int A[ ], int n )
{
 int temp;
for(int k = 0; k< n-1; k++)
{
// (n-k-1) is for ignoring comparisons of elements which have already been compared in earlier
     iterations
for(int i = 0; i < n-k-1; i++)
 {
 if(A[ i ] > A[ i+1] )
{
// here swapping of positions is being done.
 temp = A[ i ];
 A[ i ] = A[ i+1 ];
 A[ i + 1] = temp;
}
}
}
}
```

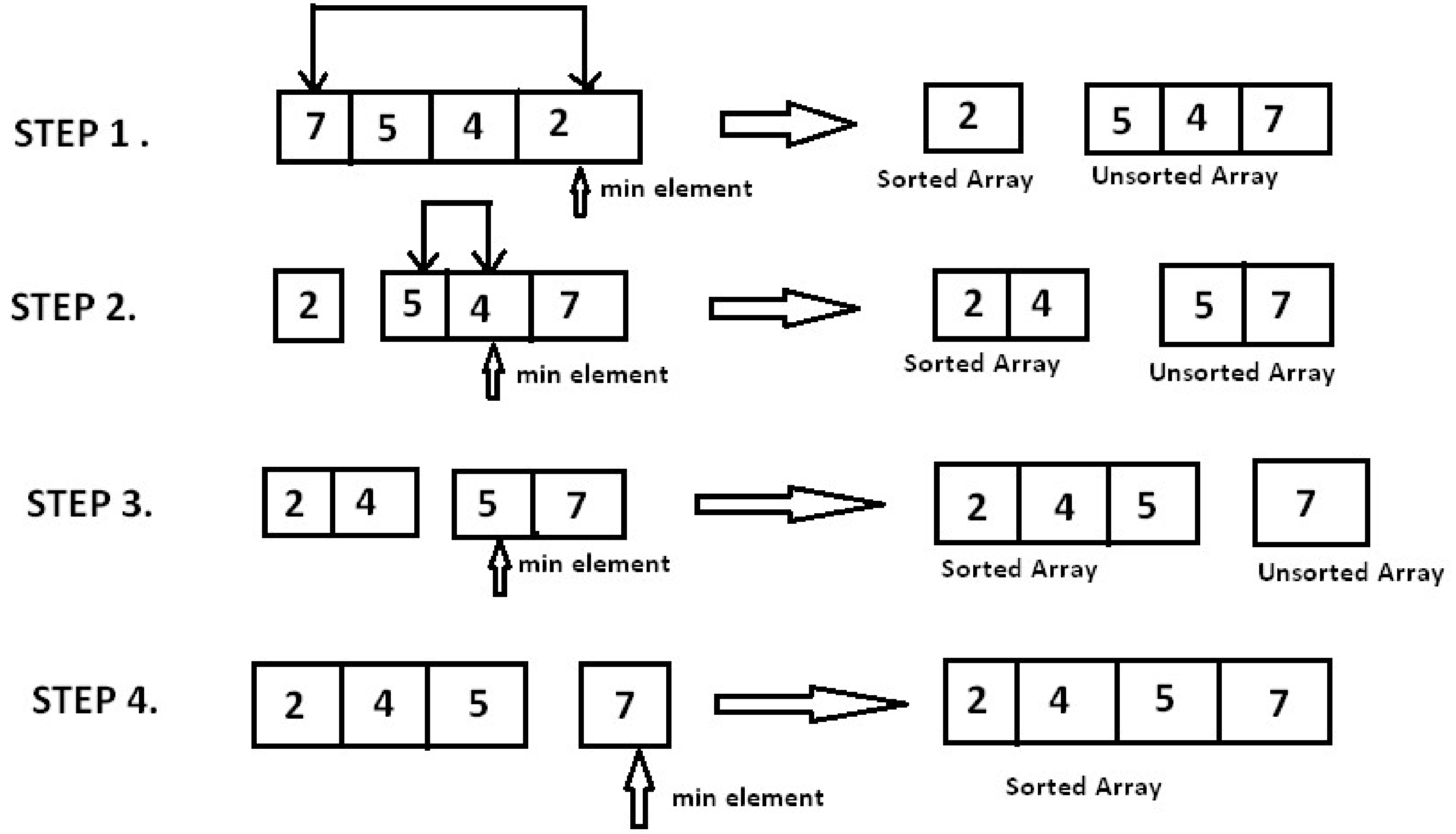

STEP 1
7 4 5 2
4 and 7 are swapped
4 7 5 2
4 7 5 2
5 and 7 are swapped
4 5 7 2
4 5 7 2
2 and 7 are swapped
4 5 2 7
STEP 2
4 5 2 7
No swap happens, as 5 > 4
4 5 2 7
4 5 2 7
2 and 5 are swapped
4 2 5 7
STEP 3
4 2 5 7
2 and 4 are swapped
2 4 5 7
(sorted array)

Selection Sort

- In Selection sort, the smallest element is exchanged with the first element of the unsorted list of elements (the exchanged element takes the place where smallest element is initially placed).
- Then the second smallest element is exchanged with the second element of the unsorted list of elements and so on until all the elements are sorted.



```
void selection_sort (int A[ ], int n) {
        // temporary variable to store the position of minimum element

        int minimum;
        // reduces the effective size of the array by one in  each iteration.

        for(int i = 0; i < n-1 ; i++)  {

           // assuming the first element to be the minimum of the unsorted array .
             minimum = i ;

          // gives the effective size of the unsorted  array .

            for(int j = i+1; j < n ; j++ ) {
                if(A[ j ] < A[ minimum ])  {                    //finds the minimum element
                minimum = j ;
                }
             }
          // putting minimum element on its proper position.
          swap ( A[ minimum ], A[ i ]) ;
        }
   }
```

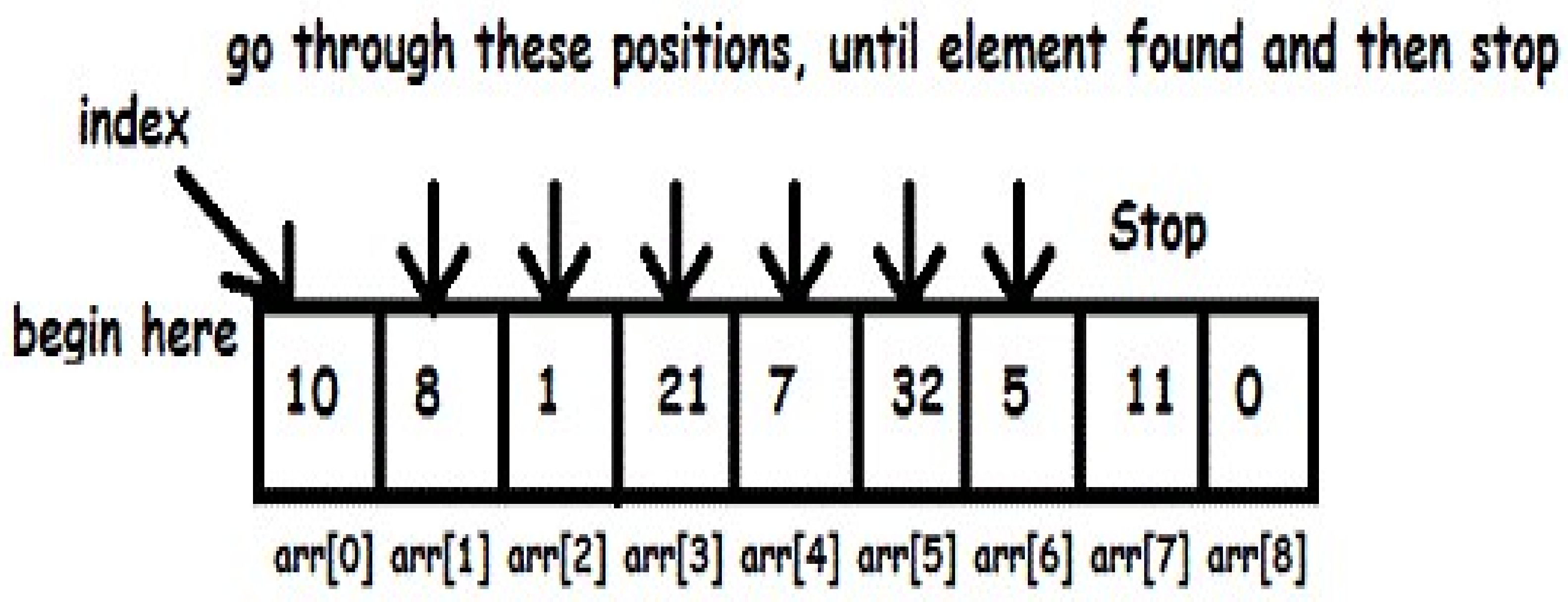

STEP 1 .
7 5 4 2
min element
2
Sorted Array
5 4 7
Unsorted Array
STEP 2.
2
5 4 7
min element
2 4
Sorted Array
5 7
Unsorted Array
STEP 3.
2 4
5 7
min element
2 4 5
Sorted Array
7
Unsorted Array
STEP 4.
2 4 5
7
min element
2 4 5 7
Sorted Array

LINEAR SEARCH

- Linear search in C to find whether a number is present in an array. If it's present, then at what location it occurs.
- It is also known as a sequential search.
- It is straightforward and works as follows: we compare each element with the element to search until we find it or the list ends. Linear search for multiple occurrences and using a function.

PSEUDO CODE

```
for(start to end of array)
{
    if (current_element equals to 5)
{
     print (current_index);
 }
 }
```



Element to search : 5

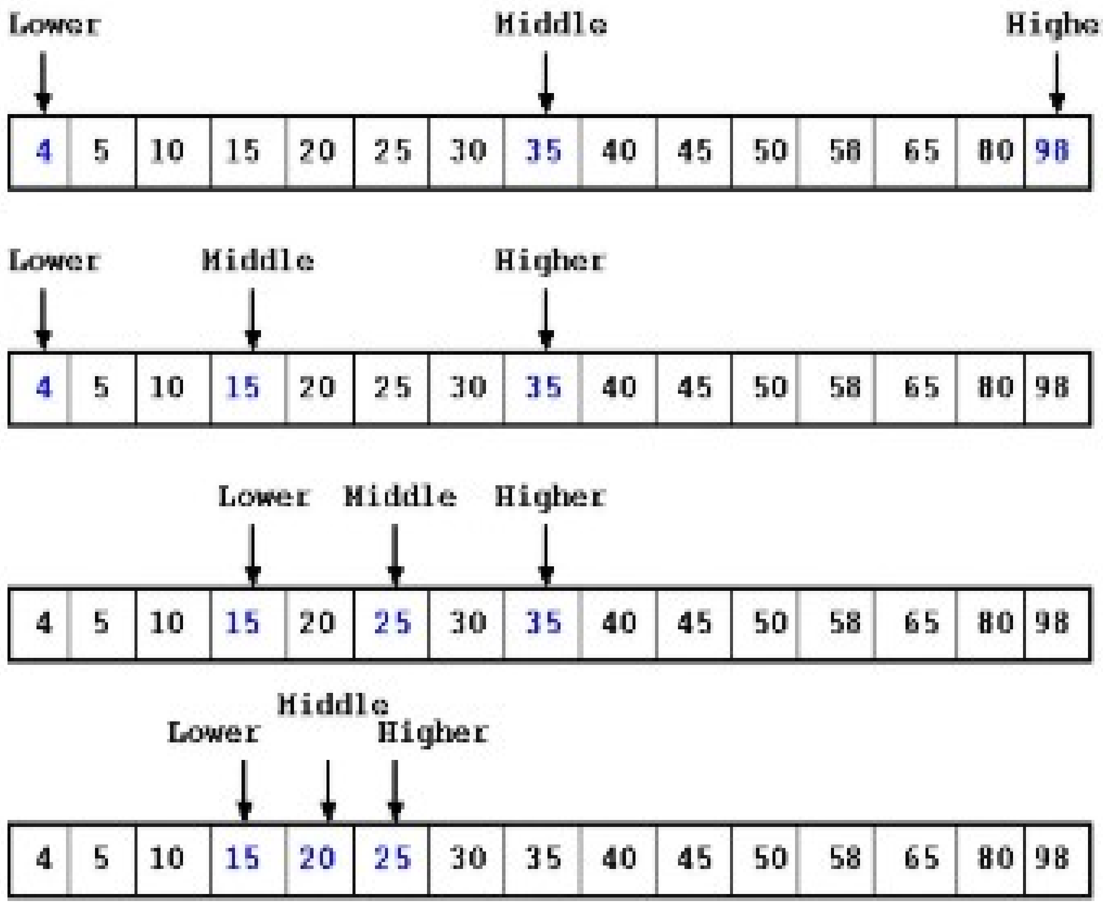
BINARY SEARCH

- Binary search in C language to find an element in a sorted array.
- If the array isn't sorted, you must sort it using a sorting technique such as merge sort.
- If the element to search is present in the list, then we print its location. The program assumes that the input numbers are in ***ascending*** order.

Binary Search Algorithm

- Following are the steps of implementation that we will be following:
- Start with the middle element:
  - If the target value is equal to the middle element of the array, then return the index of the middle element.
  - If not, then compare the middle element with the target value,
    - If the target value is greater than the number in the middle index, then pick the elements to the right of the middle index, and start with Step 1.
    - If the target value is less than the number in the middle index, then pick the elements to the left of the middle index, and start with Step 1.
- When a match is found, return the index of the element matched.
- If no match is found, then return -1.

search the element 15 in the list {4,5,10,15,20,25,30,35,40,45,50,58,65,80,98}



Comparison between Binary Search and Linear Search:

- Binary Search requires the input data to be sorted; Linear Search doesn’t
- Binary Search requires an ordering comparison; Linear Search only requires equality comparisons
- Binary Search has complexity O(log n); Linear search has complexity O(n) as discussed earlier
- Binary Search requires random access to the data; Linear Search only requires sequential access (this can be very important — it means a Linear Search can stream data of arbitrary size).