UNIT – 5

Dr.Kavitha N
Associate Professor
Dept. of IT

UNIT- V

Topics to be covered :

Files, Searching and Sorting

Files: Introduction, text and binary files, operations and accessing methods.

Searching and Sorting: Basic searching in an array of elements (linear and binary search techniques), Basic algorithms to sort array of elements (Bubble, Insertion and Selection sort algorithms).No need of implementation for Searching and sorting techniques.

Dr.Kavitha N
Associate Professor
Dept. of IT

File

A file is a named collection of related data stored on a permanent storage device, used to save information in a structured form so that it can be accessed, processed, and retrieved even after the program has terminated.

Files are important because they provide permanent storage of data, allow large amounts of information to be maintained outside memory, support data sharing between programs, and help preserve records for future use.

Dr.Kavitha N
Associate Professor
Dept. of IT

TYPES OF FILES IN C

Files in C are broadly classified into two major types based on how data is stored:

- **Text Files**
- **Binary Files**

TEXT FILES

A text file is a file that stores data in the form of human-readable characters encoded in ASCII or Unicode format. Each line ends with a newline character, and numbers are stored as character sequences, not raw bytes.

BINARY FILES

A binary file is a file that stores data in raw byte format exactly as it exists in memory, without converting it into human-readable characters.
This makes binary files compact, faster, and more accurate.

Dr.Kavitha N
Associate Professor
Dept. of IT

File Operations in C

To work with files, C provides a set of standard operations using the FILE data type declared in <stdio.h>.

1. Creating a File

Creating a file means generating a new empty file on the storage device.

function used:

fopen("filename.txt", "w");

2. Opening a File

Opening a file establishes a connection between the program and the file, so operations like reading/writing can be performed.

function used:

FILE *fp = fopen("file.txt", "r");

Dr.Kavitha N
Associate Professor
Dept. of IT

Reading from a File

Reading means fetching data from a file into variables or memory.

Functions Used:

fscanf() – reads formatted data
fgets() – reads a string/line
fgetc() – reads a single character
fread() – reads binary data

Writing to a File

Writing means storing data from the program into a file.

Functions Used:

fprintf() – writes formatted data
fputs() – writes strings
fputc() – writes a character
fwrite() – writes binary data

Dr.Kavitha N
Associate Professor
Dept. of IT

Appending Data

Appending data means adding new information to the end of an existing file without modifying or overwriting the previous content.

Function	Purpose
fopen("filename", "a")	Opens file in append mode (write at end)
fopen("filename", "a+")	Opens for reading + appending
fprintf(fp, ...)	Writes formatted text to file
fputs(str, fp)	Writes a string to file
fputc(ch, fp)	Writes a single character to file

Closing the File

Closing a file means terminating the connection between the program and the file, ensuring all data is saved and system resources are released.

Function	Purpose
fclose(fp)	Closes the file associated with pointer fp
fflush(fp) *(optional)*	Flushes the output buffer to file before closing

Dr.Kavitha N
Associate Professor
Dept. of IT

Combined Example of All File Operations in One Program

```
#include <stdio.h>

int main() {
   FILE *fp;
   char buffer[100];

   // 1 & 2. CREATE + OPEN file in write mode
   fp = fopen("demo.txt", "w");
   fprintf(fp, "Hello! This file was just created.\n");
   fprintf(fp, "Writing initial content.\n");
   fclose(fp);   // Close after writing

   // 3. OPEN again & READ the data
   fp = fopen("demo.txt", "r");
   printf("Reading file after creation:\n");
   while (fgets(buffer, sizeof(buffer), fp)) {
      printf("%s", buffer);
   }
   fclose(fp);

   // 4 & 5. OPEN in append mode & ADD new lines
   fp = fopen("demo.txt", "a");
   fprintf(fp, "Appended line: Adding more information.\n");
   fclose(fp);

   // Final READ to confirm append worked
   fp = fopen("demo.txt", "r");
   printf("\nAfter appending data:\n");
   while (fgets(buffer, sizeof(buffer), fp)) {
      printf("%s", buffer);
   }
   fclose(fp);  // 6. CLOSE the file

   return 0;
}
```


Dr.Kavitha N
Associate Professor
Dept. of IT

File Accessing Methods

File accessing methods define how data is read from or written to a file.
They decide the order of access and how the file pointer moves.

C supports two main file accessing methods:
Sequential Access
Random (Direct) Access

1. Sequential Access

Sequential access means reading or writing data in a file in a linear order, from the beginning to the end, without skipping any part. The file pointer automatically moves forward after each operation.

Types of Sequential Access

1. Character-by-character → fgetc() / fputc()
2. Line-by-line → fgets() / fputs()
3. Formatted sequential access → fscanf() / fprintf()

Dr.Kavitha N
Associate Professor
Dept. of IT

Syntax - Reading

```
fgetc(FILE *fp);
fgets(char *str, int size, FILE *fp);
fscanf(FILE *fp, "format", variables);
```

Syntax - Writing

```
fputc(int ch, FILE *fp);
fputs(const char *str, FILE *fp);
fprintf(FILE *fp, "format", values);
```

Example: Read a diary line-by-line

```
FILE *fp = fopen("diary.txt", "r");
char line[100];

while (fgets(line, sizeof(line), fp)) {
    printf("%s", line);  // Reads sequentially
}

fclose(fp);
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Random (Direct) Access

Random access allows the program to directly jump to any location in the file using the file pointer, without reading the entire file sequentially.

Move file pointer

fseek(FILE *fp, long offset, int origin);

Get current file position

long position = ftell(FILE *fp);

Go back to start of file

rewind(FILE *fp);

Types of Random Access

Absolute Positioning:

fseek(fp, 0, SEEK_SET); // Start of file

Relative Positioning:

fseek(fp, +20, SEEK_CUR); // Move 20 bytes ahead

End-based Positioning:

fseek(fp, -10, SEEK_END); // 10 bytes before end

Dr.Kavitha N
Associate Professor
Dept. of IT

Example: Jump directly to the 5th student record (Assuming each record is a fixed 20 bytes.0

```
FILE *fp = fopen("students.dat", "r");

int record_size = 20;
fseek(fp, 4 * record_size, SEEK_SET); // Jump to 5th record

char record[21];
fread(record, record_size, 1, fp);
record[20] = '\0';

printf("5th Record: %s", record);

fclose(fp);
```


Dr.Kavitha N
Associate Professor
Dept. of IT

SEARCHING

Searching is the process of locating a specific element (called the target or key) within a collection of data such as an array or list. Efficient searching helps reduce time and computational effort, especially for large datasets.

There are two fundamental searching techniques:

1. Linear Search (Sequential Search)
2. Binary Search (Dichotomic Search)

1. LINEAR SEARCH (Sequential Search)

Linear Search is a simple searching technique in which each element in the array is checked one-by-one from start to end until the target element is found or the array ends. Works on both sorted and unsorted arrays.

Syntax:

```
for (i = 0; i < n; i++) {
   if (arr[i] == key) {
      return i;   // index found
   }
}
return -1;        // not found
```

Dr.Kavitha N
Associate Professor
Dept. of IT

2. Binary Search

Binary Search is an efficient search technique that repeatedly divides the sorted array into two halves and determines which half may contain the target element.

Because it eliminates half of the remaining items in each step, it is much faster than Linear Search.

Note: Array must be sorted (in ascending or descending order).

Syntax:

```
low = 0;
high = size - 1;

while(low <= high) {
   mid = (low + high) / 2;

   if(arr[mid] == key)
      // found
   else if(key < arr[mid])
      high = mid - 1;
   else
      low = mid + 1;
}
```


Dr.Kavitha N
Associate Professor
Dept. of IT

Example — Linear Search

```
#include <stdio.h>

int main() {
   int arr[] = {42, 15, 8, 23, 4};
   int key = 23, i, found = 0;

   for(i = 0; i < 5; i++) {
      if(arr[i] == key) {
         printf("Element %d found at index %d\n", key, i);
         found = 1;
         break;
      }
   }

   if(!found)
      printf("Element %d not found\n", key);

   return 0;
}
//Output : Element 23 found at index 3
```

Example — Binary Search

```
#include <stdio.h>
int main() {
   int arr[] = {3, 9, 14, 28, 37, 56};
   int key = 28;
   int low = 0, high = 5, mid;

   while(low <= high) {
      mid = (low + high) / 2;

      if(arr[mid] == key) {
         printf("Element %d found at index %d\n", key, mid);
         return 0;
      }

      if(key < arr[mid])
         high = mid - 1;
      else
         low = mid + 1;
   }
   printf("Element %d not found\n", key);
   return 0;
} //Output: Element 28 found at index 3
```


Dr.Kavitha N
Associate Professor
Dept. of IT

SORTING

Sorting is the process of arranging the elements of a list or array in a specific order, typically in Ascending, Descending

Types of Sorting Algorithms

BUBBLE SORT

Bubble Sort repeatedly compares adjacent elements and swaps them if they are in the wrong order.

```
for(i = 0; i < n-1; i++) {
   for(j = 0; j < n-i-1; j++) {
      if(arr[j] > arr[j+1]) {
         swap(arr[j], arr[j+1]);
      }
   }
}
```

INSERTION SORT

Insertion Sort builds the sorted list one element at a time by inserting each new element into its correct position among the already-sorted elements.

```
for(i = 1; i < n; i++) {
   key = arr[i];
   j = i - 1;
   while(j >= 0 && arr[j] > key) {
      arr[j+1] = arr[j];
      j--;
   }
   arr[j+1] = key;
}
```

SELECTION SORT

Selection Sort repeatedly selects the smallest element from the unsorted part of the array and places it at the beginning.

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

   for(j = i+1; j < n; j++) {
      if(arr[j] < arr[min])
         min = j;
   }

   swap(arr[i], arr[min]);
}
```

Dr.Kavitha N
Associate Professor
Dept. of IT

Thank You