Introduction to Computing

File Access, Command-Line Arguments

Recall File

- **FILE*** is a datatype used to represent a pointer to a file
- To open a file we use a function called **fopen**
  - It takes two parameters
    - Name of the file
    - Mode in which it is to be opened
  - It returns a pointer to the file if the file is opened successfully, otherwise it returns NULL

Example of a file creation for writing

```
FILE *fp;
char filename[] = “a_file.dat”
fp = fopen (filename, “w”);
if (fp != NULL)
{
    /* WRITE SOMETHING IN FILE */
fclose (fp);
}
```

File operations

- **fputc**
- **fputs**
- **fprintf**
- **fflush**
- fgetc
- fgets
- fscanf
- feof
- ungetc

```
FILE *fp = fopen("abc.txt", "w");

if (fp != NULL) {
        fputc('a', fp);
        fputs("cde", fp);
        fprintf(fp, "%d, %c, %s", 25, 'I', "hello");
        fflush(fp);
        fclose(fp);
}
```

File operations (contd)

- fputc
- fputs
- fprintf
- fflush
- **fgetc**
- **fgets**
- **fscanf**
- feof
- ungetc

```
FILE *fp = fopen("abc.txt", "r");
char buf[10];     int num;     char c;
if (fp != NULL) {
        c = fgetc(fp);            // printf (“%c”, c);
        fgets(buf, 4, fp);      // printf (“%s", buf);
        fscanf(fp, "%d, %c, %s", &num, &c, buf);
        printf (“%d %c %s”, num, c, buf);
        fclose(fp);
}
```

File operations (contd)

- fputc
- fputs
- fprintf
- fflush
- fgetc
- fgets
- fscanf
- **feof**
- **ungetc**

```
char c, buf[256];
FILE *fp = fopen("abc.txt", "r");
if (fp != NULL) {
while (!feof(fp)) {
      c = fgetc(fp);
      if (c == ‘a’)
            ungetc(‘b’, fp);
      fgets(buf, 255, fp);
      printf("%s", buf);
} }
```

File operations (contd)

Two more functions for writing or reading binary data

- fwrite
- fread

```
int numbers[5] = {10, 20, 30, 40, 50};
fptr = fopen("numbers.bin", "wb");
fwrite(numbers, sizeof(int), 5, fptr);
fclose(fptr);

int readNumbers[5];
fptr = fopen("numbers.bin", "rb");
fread(readNumbers, sizeof(int), 5, fptr);
fclose(fptr);

for (int i = 0; i < 5; i++) { printf("%d ", readNumbers[i]);}
```

Command Line Arguments (CLA)

- Command-line arguments allow the user to provide inputs to the program at runtime. It is useful for customizing program behavior based on user input.

- The arguments are passed using two parameters in the main function:
  - int argc: Argument count
  - char *argv[]: Argument vector (array of arguments as strings)
- The first argument (argv[0]) is always the program name.

Command Line Arguments (CLA)

- Compile the code ⇒
  - It will generate a.exe file

Run the code as follows:

```
.\a.exe  Hello
        argument supplied is Hello
.\a.exe  Hello Hi
        Too many arguments.
.\a.exe
        One argument expected.
```

```
int main( int argc, char *argv[] )
{
    if( argc == 2 )
       printf ("argument supplied is %s\n", argv[1])

    else if ( argc > 2 )
       printf ("Too many arguments.\n");

    else
       printf ("One argument expected.\n");
}
```

Using Command-Line Arguments

- In this example, we will use command-line arguments to accept a file name as input and then read the file's contents

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf('Usage: %s <filename>\\n', argv[0]);
        return 1;
    }
    FILE *fptr = fopen(argv[1], 'r');
     if (fptr == NULL) {
        printf('Error opening file %s\\n', argv[1]);
        return 1;
    }
    // Read file content
    fclose(fptr);
}
```

File Positioning Functions

- File positioning functions allow us to move the file pointer to different locations in a file, which is useful for reading or writing data from specific positions.

- Key Functions:
- **fseek()**: Moves the file pointer to a specified position.
- **ftell()**: Returns the current position of the file pointer.
- **rewind()**: Moves the file pointer back to the start of the file.

fseek() Function

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

- **Parameters**:
  - `stream`: The file pointer.
  - `offset`: Number of bytes to move.
  - `whence`: The reference point (where to move from):
    - `SEEK_SET`: Beginning of the file.
    - `SEEK_CUR`: Current position.
    - `SEEK_END`: End of the file..

Example: Move the file pointer 10 bytes ahead from the start of the file

```
FILE *fptr = fopen("example.txt", "r");

// Move 10 bytes from the beginning
fseek(fptr, 10, SEEK_SET);
```

ftell() Function

```
long ftell(FILE *stream);
```

- **Purpose**: Returns the current position of the file pointer, measured in bytes from the beginning of the file..

Example: Find the current position of the file pointer after reading some data

```
FILE *fptr = fopen("example.txt", "r");
// Move 10 bytes ahead
fseek(fptr, 10, SEEK_SET);

// Get current position (should return 10)
long pos = ftell(fptr);

printf("Current position: %ld\n", pos);
```

rewind() Function

```
void rewind(FILE *stream);
```

- **Purpose**: Resets the file pointer to the beginning of the file.
- **Note**: It is equivalent to `fseek(stream, 0, SEEK_SET);`.

Example: Reset the file pointer after reading some data, then read the file again from the beginning

```
FILE *fptr = fopen("example.txt", "r");

// Move 10 bytes ahead
fseek(fptr, 10, SEEK_SET);

// Move back to the start of the file
rewind(fptr);
```

Practical Example: Reading Specific Data

- Suppose we have a binary file that stores records of fixed size (e.g., student records), and we want to jump to a specific record (e.g., the 3rd record).
- Using `fseek()`, we can directly jump to the position without reading the entire file sequentially.

```
struct Student { int id; char name[50];  };

FILE *fptr = fopen("students.bin", "rb");

// Jump to the 3rd record
fseek(fptr, 2 * sizeof(struct Student), SEEK_SET);
struct Student s;

// Read the 3rd record
fread(&s, sizeof(struct Student), 1, fptr);

printf("Student ID: %d, Name: %s\n", s.id, s.name);
fclose(fptr);
```