

# Programming and Algorithms

## COMP1038.PGA

### **Week 10 – Lecture 1, 2, 3: Files and I/O**

Dr. Pushpendu Kar

# Overview

- Files
  - Introduction
  - Types
  - Opening
  - Modes
  - Reading
  - Writing
  - Closing
  - Pre-defined file handles
  - Random access



# Introduction

- A named collection of data, stored in secondary storage (typically).
- Stored as sequence of bytes, logically contiguous (may not be physically contiguous on disk).
- Instead of entering all data into our programs via the keyboard and printing all the output to the screen, it would be useful if we could read and write data to other files.
- The last byte of a file contains the end-of-file character (EOF), with ASCII code 1A (hex).
- While reading a text file, the EOF character can be checked to know the end.



# Introduction cont...

- Standard pattern for file use
  - Open a file, specifying:
    - if it is text or binary
    - if we want to read, write, or append to it
    - if we want sequential or random access to it
  - Read/write to the file as needed
  - Close the file when we are finished with it



# Types

- **Text**

- contains ASCII codes only

- **Binary**

- can contain non-ASCII characters
- Image, audio, video, executable, etc.



# Opening a file

- `fopen` function opens a file and returns a file handle.
- The file handle is used in all other file operations for that file, so the function knows which file to operate on.
- This means you can have multiple files open at the same time because each will have its own file handle.

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

- This opens the file "highscores.txt" (in the current directory) for reading only ("r").
- `fopen` returns a pointer to a `FILE` type — we don't care what this is, we just use it as documented. Specifically, it is not a pointer to the next character in a string representing the file!
- File pointer is the pointer to a `FILE` data type is also called as a stream pointer. A file pointer points to the block of information of the stream(file) that had just been opened through `fopen` function.



# File modes

- The second parameter to `fopen` are character codes describing how we want to use the file.
  - Most common codes (see book or man page for full list):
    - "r" open the file for reading
    - "r+" open the file for reading and writing
    - "w" open the file for writing (deleting current contents!)
    - "a" open the file for appending (writing) to the end of the file (keeps current contents).
  - These will all treat the contents of the file as text
  - Note: text files saved in text editors on different OS may have `\n` vs `\r\n`.
  - We can add a "b" character to indicate that the file is a binary file.
    - "rb", "wb" or "ab"
- ```
fptr = fopen ("xyz.jpg", "rb");
```



# File errors

- If `fopen` cannot open the file, it returns `NULL` and sets `errno`
- This is a global variable declared in `errno.h`, it will contain an error code describing what went wrong.
- Other functions in `errno.h` allow us to check for errors and for end-of-file.
- `perror(char *)` prints out the appropriate error message for `errno`, plus any extra message we supply as a parameter.
- `feof(FILE *)` returns true or false if the given file handle is at the end-of-file or not.





# Sequential files

- Sequential file access opens the file and sets the current position within the file to the beginning (or end if using "a" modes).
- All read/write operations then happen at that point and move the current position on by the appropriate amount, until you eventually get to the end of file (if reading) or stop writing to it.
- The rewind function moves the current position back to the start of the file (but if you use this frequently your design is probably wrong or should use a random access file).



# Reading a text file

- **fscanf()**

- Same as scanf except the first parameter is a file handle to read from. It can read a string until space.
- `int fscanf(file pointer, format specifier, char array to store data)`

```
#include <stdio.h>
```

```
int main()
{ FILE *fp;
  char buff[255];
  fp = fopen("./tmp/test.txt", "r");
  if(fp == NULL)
  { printf("File does not exist\n");
    return 1;
  }
  fscanf(fp, "%s", buff);
  printf("%s\n", buff);
  fclose(fp);
  return 0;
}
```



# Reading a text file cont...

- **fgets()**
  - Is used to read a file line by line
  - `char * fgets(char array to store data, size, file pointer)`

```
// C program to Open a File,  
// Read from it, and Close the File
```

```
# include <stdio.h>  
# include <string.h>
```

```
int main()  
{
```

```
    // Declare the file pointer  
    FILE *filePointer;
```

```
    // Declare the variable for the data to be read from file  
    char dataToBeRead[50];
```

```
    // Open the existing file test.c using fopen()  
    // in read mode using "r" attribute  
    filePointer = fopen("./tmp/test.txt", "r");
```

```
    // Check if this filePointer is null  
    // which maybe if the file does not exist
```

```
    if ( filePointer == NULL )  
    {  
        printf( "failed to open the file." );  
        return 1;  
    }  
    else  
    {  
        // Read the dataToBeRead from the file  
        // using fgets() method  
        while( fgets ( dataToBeRead, 50, filePointer ) != NULL )  
        {  
            // Print the dataToBeRead  
            printf( "%s", dataToBeRead );  
        }  
        // Closing the file using fclose()  
        fclose(filePointer);  
    }  
    return 0;  
}
```

# Reading a text file cont...

- **getc()**
  - Reads a file character by character
  - `int getc(File pointer)`

```
#include<stdio.h>
```

```
int main()
{
    FILE *fp;
    char ch;
    fp = fopen("./tmp/test.txt", "r");

    if (fp == NULL)
    {
        printf("failed to open the file.");
        return 1;
    }

    while((ch = getc(fp))!= EOF)
    {
        printf("%c",ch);
    }
    fclose(fp);
    return 0;
}
```



# Reading a binary file

## ■ fread()

- The fread() function reads the given number of elements of specific size from the file stream and stores it in the buffer memory.
- The total number of bytes read by fread() function is the number of elements read multiplied by the size of each element in bytes.
- size\_t fread(data pointer to store, size of elements, number of elements, file pointer)

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum
{
    int n1, n2, n3;
};

int main()
{
    int n;
    struct threeNum num;
    FILE *fp;

    if ((fp = fopen("./tmp/program.bin", "rb")) == NULL){
        printf("Error in file opening \n");
        // Program exits if the file pointer returns NULL.
        exit(1);
    }
    for(n = 1; n < 5; ++n)
    {
        fread(&num, sizeof(struct threeNum), 1, fp);
        printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
    }
    fclose(fp);
    return 0;
}
```



# Thank You!



# Writing into a text file

- **fprintf()**

- Same as printf except the first parameter is a file handle to write to.
- int fprintf(file pointer, char array to write)

```
#include <stdio.h>
```

```
int main()
{ FILE *fp;
  char buff[255];
  fp = fopen("./tmp/test.txt", "w");
  if(fp == NULL)
  { printf("Error, file opening failed\n");
    return 1;
  }
  fgets(buff, 255, stdin);
  fprintf(fp, buff);
  fclose(fp);
  return 0;
}
```



# Writing into a text file

## cont...

- **fputs()**

- Writes into a file line-by-line
- fputs(char array to write, file pointer)

```
// C program to Open a File,  
// Write in it, And Close the File
```

```
# include <stdio.h>  
# include <string.h>
```

```
int main()  
{ // Declare the file pointer  
  FILE *fp;  
  char dataToBeWritten[50];  
  
  // Get the data to be written in file  
  fgets(dataToBeWritten, 50, stdin);  
  
  // Open the existing file test.txt using fopen()  
  // in write mode using "w" attribute  
  fp = fopen("./tmp/test.txt", "w");
```

```
    // Check if this filePointer is null  
    // which maybe if the file does not exist  
    if ( fp == NULL )  
    {    printf( "failed to open the file.\n" );  
      }  
    else  
    {  
      // Write the dataToBeWritten into the file  
      if ( strlen(dataToBeWritten) > 0 )  
      {  
        // writing in the file using fputs()  
        fputs(dataToBeWritten, fp);  
      }  
      // Closing the file using fclose()  
      fclose(fp);  
    }  
    return 0;  
}
```



# Writing into a text file

## cont...

- **putc()**
  - Writes into a file character by character
  - `int putc(file pointer)`

```
#include<stdio.h>
```

```
int main()
{ FILE *fp;
  char ch;
  fp = fopen("./tmp/test.txt", "w");
  if(fp == NULL)
  { printf(Error in file opening\n");
    return 1;
  }
  printf("Enter data...\n");
  while( (ch = getchar()) != EOF)
  { putc(ch, fp);
  }
  fclose(fp);
  return 0;
}
```

//On Linux systems and OS X, the character to input to cause an EOF is Ctrl-D.  
//For Windows, it's Ctrl-Z.



# Writing into a binary file

## ■ **fwrite();**

- This function to easily writes a structure in a file. fwrite() function writes the to the file stream in the form of binary data block.
- `size_t fwrite(data element to be written, size of elements, number of elements, file pointer);`

```
#include <stdio.h>
#include <stdlib.h>
struct threeNum
{
    int n1, n2, n3;
};
int main()
{ int n;
  struct threeNum num;
  FILE *fp;

  for(n = 1; n < 15; ++n)
  {
      num.n1 = n;
      num.n2 = 5*n;
      num.n3 = 5*n + 1;
      fwrite(&num, sizeof(struct threeNum), 1, fp);
  }
  fclose(fp);
  return 0;
}
```

```
if ((fp = fopen("./tmp/program.bin", "wb")) == NULL)
{
    printf("Error in file opening \n");
    // Program exits if the file pointer returns NULL.
    exit(1);
}
```



# Closing a file

- `fclose` takes a file handle and closes that file.
- After this, the file handle is no longer valid and should not be used.
- `fclose` returns 0 on success, or EOF on error and sets `errno`.
- Failing to close a file you are writing to can result in data-loss!



# Pre-defined file handling

- There are three pre-opened file handles in `<stdio.h>`:
  - `stdin` — standard input ("keyboard")
  - `stdout` — standard output ("screen")
  - `stderr` — standard error (for error messages separate from normal output)
- Print to the screen:
  - `fprintf(stdout, "Hello world\n");`
  - Do not close these standard file handles yourself.

# Input File & Output File redirection

- One may redirect the standard input and standard output to other files (other than stdin and stdout).
- Usage: Suppose the executable file is a.out:

**\$ ./a.out <in.txt >out.txt**

- scanf() will read data inputs from the file "in.txt", and printf() will output results on the file "out.txt".

**\$ ./a.out <in.txt >>out.txt**

- scanf() will read data inputs from the file "in.txt", and printf() will append results at the end of the file "out.txt".



# Random access files

- The previous slides discuss sequential access files — you read or write from the start of the file to the end.
- Random access in files is when you jump around the file, reading and writing different parts of the file out-of-order.
- Random access files use the position in bytes from the start of the File to move around the file.
- Due to this, random access files are usually binary files, since we have more control over the exact byte-by-byte layout of these files than text files.



# Random access function

- `fread()`: It reads a given number of bytes from the file, starting at the current position.
- `fwrite()`: It writes a given number of bytes to the file, starting at the current position.
- `fseek()`: It moves the current position in the file by a given number of bytes from various reference points.
  - `SEEK_SET`: then the position is the beginning of the file
  - `SEEK_END`: used if you want to go to the end of the file
  - `SEEK_CUR`: then the position is set, x bytes, from the current position.
- `ftell()`: It tells the byte location of current position of cursor in file pointer.
- `rewind()`: It moves the control to beginning of the file.



# Random access function

## cont...

```
#include<stdio.h>
/* Our structure */
struct rec
{ int x,y,z;
};

int main()
{ int counter;
  FILE *fp;
  struct rec my_record;
  fp=fopen("./tmp/program.bin","rb");
  if (!fp)
  { printf("Unable to open file!\n");
    return 1;
  }
  printf("%d\n", ftell(fp));

  fseek(fp, sizeof(struct rec)*2, SEEK_SET);
  printf("%d\n", ftell(fp));

  fread(&my_record,sizeof(struct rec),1,fp);
  printf("%d\n",my_record.x);

  fseek(fp, sizeof(struct rec)*2, SEEK_CUR);
  printf("%d\n", ftell(fp));
```

```
fread(&my_record,sizeof(struct rec),1,fp);
printf("%d\n",my_record.x);

fseek(fp, sizeof(struct rec), SEEK_END);
printf("%d\n", ftell(fp));

fread(&my_record,sizeof(struct rec),1,fp);
printf("%d\n",my_record.x);

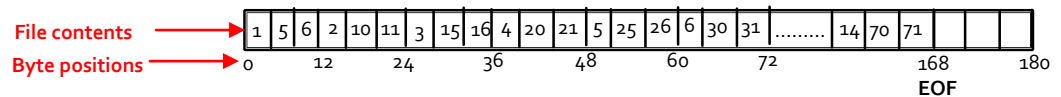
rewind(fp);

printf("%d\n", ftell(fp));

fread(&my_record,sizeof(struct rec),1,fp);
printf("%d\n",my_record.x);

fclose(fp);
return 0;
}
```

```
[z2019024@CSLinux file_lecture_code]$ ./file_random
0
24
3
60
6
180
6
0
1
```





# The End

