Short-term Hands-on Supplementary Course on C programming

Session 12:
File Handling

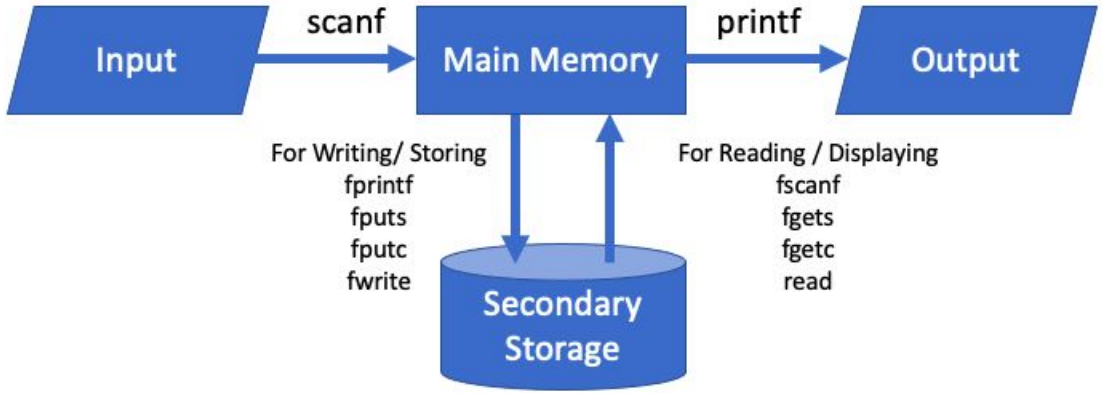
Nandakishor. V
Nitheesh Kumar. N

Agenda

1. File Handling - What and Why
2. File Descriptors, Streams and Modes
3. Handling Text Files in C

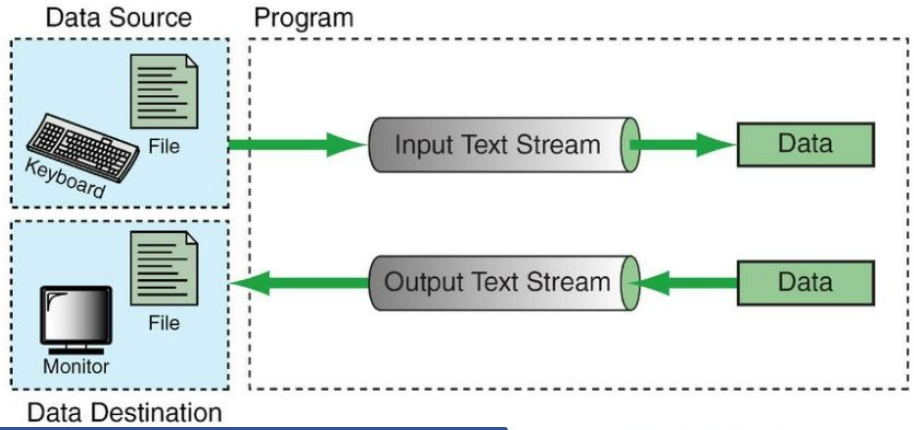
File Handling

- To **persistently preserve** your data even after program termination.
- To **read large amount of data** — access them from a file using C functions.
- To **move data** from one computer to another, if your program runs on different systems.



File Streams and Descriptors

- To operate on a file, we first set up a "**connection**" between the program and the file
- There are broadly 2 **file connection representation mechanisms**,
  - **File descriptors** (*int*): Low-level, primitive
  - **File streams** (*FILE**): High-level, layered atop descriptors
- To keep things simple, we use streams in this course
  - Streams offer more **operability on I/O buffering**
  - Streams provide **simpler, richer** and **powerful I/O functions**



File Streams and Descriptors

DO YOU REALIZE? The **printf()** and **scanf()** functions we've been using all along, essentially operate on standard I/O streams that connect to **STDIN** (Standard I/P) and **STDOUT** (Standard O/P)

- The functions we'll use to handle file I/O are all defined in the **<stdio.h> header** file
- To **establish a stream connection** with a file, we use the *fopen()* function,

```
FILE* fopen ( const char * filename, const char * mode );
```

- To **terminate a stream connection** with a file, we use the *fopen()* function,

```
int fclose ( FILE * stream_object );
```

File Access Modes

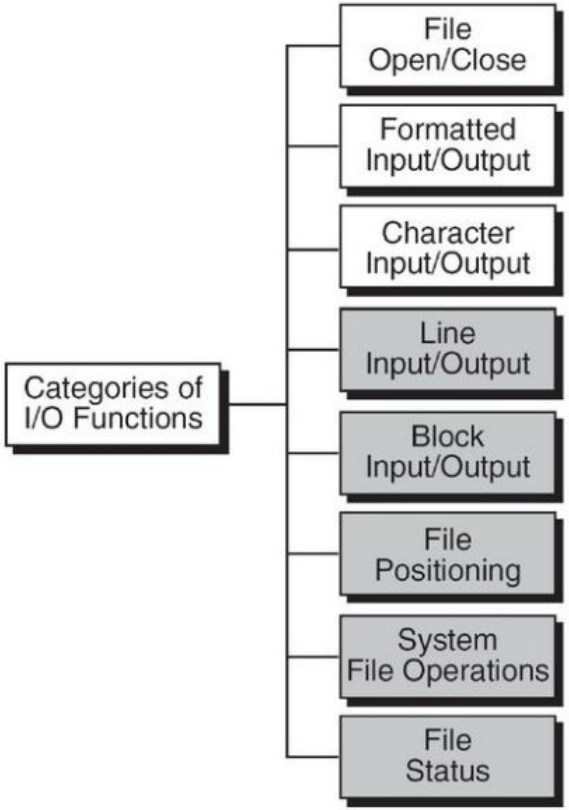
- File streams offer granularity to specify "**how**" you want to **open (connect to) a file**
  - For instance, if a file with the same name exists, should *fopen()* replace it with a new file, or append to it?
- This is specified using the **mode argument** in ***fopen()***

File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.

Add 'b' to mode string to open file in binary mode

Complete list of modes can be found here

Categories of Standard I/O Function



In C typical operation on files, either text or binary, include:

- Creating a new file
- Opening a file
- Reading a file
- Writing into file
- Closing the file

The processing, formatting, etc. goes into programming logic

SSN

Text File I/O

Once we have the *file stream object* — **stream**,

- To write a string to a file,

  size_t fprintf (**FILE** * stream, **const char** * format, ...);

- To read a string from a file,

  size_t fscanf (**FILE** * stream, **const char** * format, ...);

NOTE: Again, just a variation of the **printf()** and **scanf()** functions — just choose a file stream instead of standard I/O

Many other I/O functions are available, mostly specific to format of data. [Reference]

Demo Live Code: Opening, Reading and Writing in text files

1) Writing a text file

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

int main()
{
   int num;
   FILE *fptr;

   // use appropriate location if you are using MacOS or Linux
   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
}
```

2) Reading a text file

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

int main()
{
   int num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);
   fclose(fptr);

   return 0;
}
```

File operations in Binary Files

In C typical operation on files, either text or binary, include:

- Creating a new file - FILE *
- Opening a file - fopen()
- Reading a file - fread()
  fread(addressData, sizeData, numbersData, pointerToFile)
- Writing into file - fwrite()
  fwrite(addressData, sizeData, numbersData, pointerToFile)
- Random Access - fseek(), ftell(), rewind()
- Closing the file - fclose()

Opening, Reading and Writing in binary files

1) Writing a binary file

```c
struct threeNum
{
   int n1, n2, n3;
};

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

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

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

   return 0;
}
```

2) Reading a binary file

```c
struct threeNum
{
   int n1, n2, n3;
};

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

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   for(n = 1; n < 5; ++n)
   {
      fread(&num, sizeof(struct threeNum), 1, fptr);
      printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
   }
   fclose(fptr);

   return 0;
}
```

Tutorial

Try file operations in different modes.

Explore Binary Files.

Any Questions?

Thank You for attending!

Contact us regarding any questions through email

nandakishor2010608@ssn.edu.in
nitheesh2010343@ssn.edu.in