"Advanced C Programming"

File I/O

1. Introduction

- Data files
  - Can be **created, updated**, and **processed** by C programs
  - Are used for permanent storage of large amounts of data
    - Storage of data in variables and arrays is only temporary

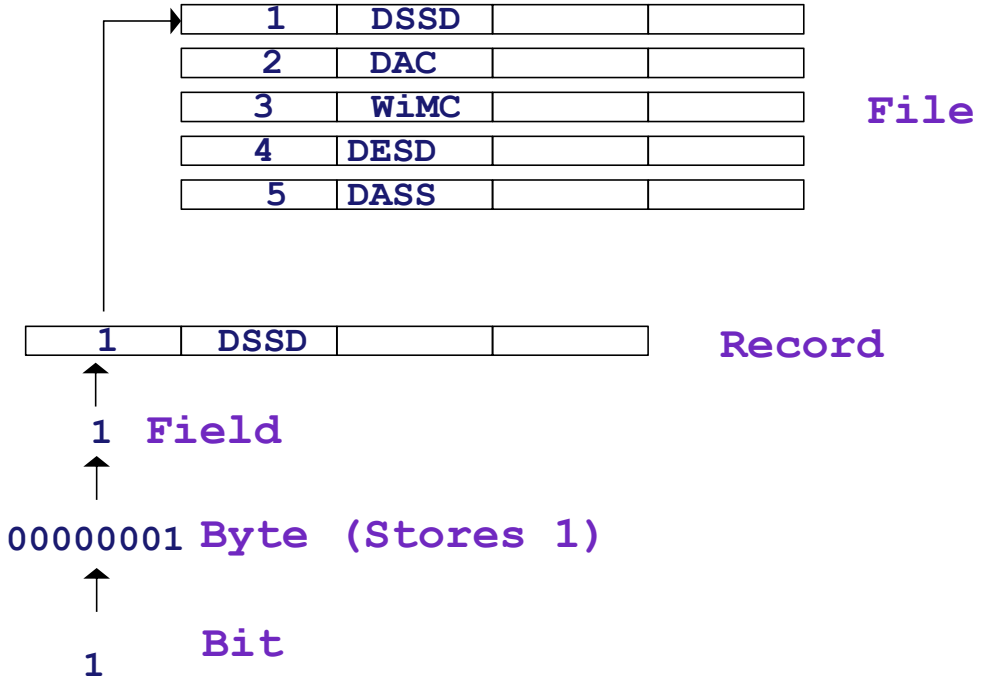
2. The Data Hierarchy

- Data Hierarchy:
  - Bit – smallest data item
    - Value of 0 or 1
  - Byte – 8 bits
    - Used to store a character
  - Decimal digits, letters, and special symbols
  - Field – group of characters conveying meaning
    - **Example:** your name
  - Record – group of related fields
    - Represented by a struct or a class

Example: In a payroll system, a record for a particular employee that contained his/her identification number, name, address, etc.

2. The Data Hierarchy…

- **Data Hierarchy (continued):**
  - **File** – group of related records
    - Example: Employee information file
  - **Database** – group of related files



2. The Data Hierarchy…

- **Data files**
  - **Record key**
    - Identifies a record to facilitate the retrieval of specific records from a file
  - **Sequential file**
    - Records typically sorted by key

3. Files and Streams

- C views each file as a sequence of bytes
  - File ends with the end-of-file (EOF) marker, Or, file ends at a specified byte.
- Stream created when a file is opened
  - Provide communication channel between files and programs
  - Opening a file returns a pointer to a FILE
    - Example file pointers:
      - stdin - standard input (keyboard)
      - stdout - standard output (screen)
      - stderr - standard error (screen)

3. Files and Streams…

- **FILE structure**
  - File descriptor
    - Index into operating system array called the open file table

3. Files and Streams…

Table of file open modes:

Mode	Description
r	Opens an existing text file for reading purpose.
w	Opens a text file for writing, if it does not exist then a new file is created. Here your program will start writing content from the beginning of the file.
a	Opens a text file for writing in appending mode, if it does not exist then a new file is created. Here your program will start appending content in the existing file content.
r+	Opens a text file for reading and writing both.
w+	Opens a text file for reading and writing both. It first truncate the file to zero length if it exists otherwise create the file if it does not exist.
a+	Opens a text file for reading and writing both. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.

3. Files and Streams…

Table of file open modes:

Mode	Operation	Description
r	Reading	Existing File
w	Writing	Truncate to zero length if file exists, Otherwise it creates a new file and writes from the beginning.
a	Append	Opens existing file if it exists, Otherwise it creates a new file and write from the end.
r+	Reading Writing	Opens an existing file only, and can read or write.
w+	Reading Writing	Existing file if exists. Otherwise it creates a new file. Writes from the beginning
a+	Reading Appending	Existing file if exists. Otherwise it creates a new file. Writes from the end.

3. Files and Streams…

- Read/Write functions in standard library
  - **fopen -** FILE *fopen(const char *filename, const char *mode)
    - opens the **filename** pointed to by filename using the given **mode**.
    - **Example:**
      - **FILE * fp;**
      - **fp = fopen ("file.txt", "w+");**
    - Arguments:
      - **filename** -- This is the C string containing the name of the file to be opened.
      - **mode** -- This is the C string containing a file access mode.
    - Return Value: This function returns a FILE pointer. Otherwise, NULL is returned.

3. Files and Streams…

- Read/Write functions in standard library
  - **fclose -** int fclose(FILE *stream)
    - Closes the stream. All buffers are flushed.
    - **Example:**
      - **FILE * fp;**
      - **fclose(fp);**
    - Arguments:
      - **stream** -- This is the pointer to a FILE object that specifies the stream to be closed.
    - Return Value: This method returns zero if the stream is successfully closed. On failure, EOF is returned.

3. Files and Streams…

- Read/Write functions in standard library
  - **fgetc -** c = fgetc(fp);
    - **int fgetc(FILE *stream)**
    - **Gets the next character** from the specified stream **and advances the pointer to the next character** in the stream.
    - Reads one character from a file
    - Takes a FILE pointer as an argument
    - Return Value: character read or EOF
    - fgetc(stdin) equivalent to getchar() & scanf("%c", &c)

3. Files and Streams…

- Read/Write functions in standard library
  - **fputc -** fputc(ch, fp);
    - **Declaration:** int fputc(int char, FILE *stream)
    - Parameters:
      - **char** -- This is character to be written. This is passed as its int promotion.
      - **stream** -- This is the pointer to a FILE object that identifies the stream where the character is to be written.
      - **Return Value** -- If there are no errors, the same character that has been written is returned.If an error occurs, EOF is returned and the error indicator is set.
    - fputc('a', stdout) equivalent to putchar('a')

3. Files and Streams…

- Read/Write functions in standard library
  - **fgets**
    - Declaration: char *fgets(char *str, int n, FILE *stream)
    - Reads a line from a file till the EOF or n characters are read
    - Arguments
      - **str** - array of chars where the string read is stored.
      - **n** -- This is the maximum number of characters to be read (including the final null-character)
      - **stream** -- This is the pointer to a FILE object
    - Return Value - On success, the function returns the same str parameter. If Error, NULL pointer is retured

3. Files and Streams…

- Read/Write functions in standard library
  - **fputs -** int fputs(const char *str, FILE *stream)
    - Writes a string to the specified stream
    - Arguments
      - **str** -- This is an array containing the null-terminated sequence of characters to be written.
      - **stream** -- This is the pointer to a FILE object that identifies the stream
    - Return Value - This function returns a non-negative value else, on error it returns EOF.

3. Files and Streams…

- Read/Write functions in standard library
  - **fprintf -** int fprintf(FILE *stream, const char *format, ...)
    - File processing equivalents of printf
    - Sends formatted output to a stream.
    - Arguments:
      - **stream** -- This is the pointer to a FILE object that identifies the stream.
      - **format** – Format specifiers
    - Return Value:
      - If successful, the total number of characters written is returned otherwise, a negative number is returned.

3. Files and Streams…

- Read/Write functions in standard library
  - **fscanf -** int fprintf(FILE *stream, const char *format, ...)
    - File processing equivalents of scanf
    - Reads formatted input from a stream.
    - Arguments:
      - **stream** -- This is the pointer to a FILE object that identifies the stream.
      - **format** – Format specifiers
    - Return Value:
      - This function return the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure.

4. Creating a Sequential Access File

- **Creating a File**
  - FILE *myPtr;
    - Creates a FILE pointer called myPtr
  - myPtr = fopen("myFile.dat", openmode);
    - Function fopen returns a FILE pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, NULL returned
  - fprintf
    - Used to print to a file
    - Like printf, except first argument is a FILE pointer (pointer to the file you want to print in)

4. Creating a Sequential Access File…

- feof(FILE pointer)
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- fclose(FILE pointer)
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

❖ **Details**

- Programs may process no files, one file, or many files
- Each file must have a unique name and should have its own pointer

4. Creating a Sequential Access File…

- feof(FILE pointer)
  - Returns true if end-of-file indicator (no more data to process) is set for the specified file
- fclose(FILE pointer)
  - Closes specified file
  - Performed automatically when program ends
  - Good practice to close files explicitly

❖ **Details**

- Programs may process no files, one file, or many files
- Each file must have a unique name and should have its own pointer

File reading - fgetc

```
#include <stdio.h>
int main ()
{
//Declare a file pointer
  FILE *fp;
  char c;
//Open the file and check whether
the file is opened or not
  fp = fopen ("file1.txt", "r");
  if (fp == NULL)
    {
      perror ("Error in opening file");
      return (-1);
    }
```

```
//Read characters from files and print
to stdout

  do
    {
      c = fgetc (fp);

      if (c == EOF)
        {
          break;
        }
      fputc (c, stdout);
    }
  while (1);

//Close the file

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

File reading - fgets

```
#include <stdio.h>
int main()
{
   FILE *fp;
   char str[60];

   /* opening file for reading */
   fp = fopen("file2.txt" , "r");

   if(fp == NULL)
   {
            perror("Error opening file");
            return(-1);
   }
```

```
   while (1)
      {
        if (fgets (str, 60, fp) != NULL)
          {
            /* writing content to stdout */
                 //    puts(str);
            fputs (str, stdout);
          }
        else
          break;
      }

   fclose (fp);

   return (0);
}
```

File writing and reading – fptus and fgets

```
#include <stdio.h>
int main ()
{
   FILE *fp;
   char str[60];
   fp = fopen("file3.txt", "w+");
   fputs("This is c programming.\n",
fp);
   fputs("This is a system
programming language.", fp);
//after this fp points to the end of th
file
   rewind(fp); //fp points to the
beginning of the file
```

```
while(1)
{
        if( (fgets(str, 60, fp)) == NULL)
        {
        //          printf("Error in
reeading\n");
                break;
        }
        else
                printf("%s\n", str);
//      fputs(str, stdout);
}

   fclose(fp);

   return(0);
}
```

File reading - fscanf

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

int main()
{
        char str1[40], str2[40],
str3[40];
        int year;
        FILE * fp;

        fp = fopen ("file4.txt", "w+");
        fputs("We are in 2020\n", fp);
        fputs("We are in 2021", fp);

        rewind(fp);
        fscanf(fp, "%[^\n]s", str1);
        printf("Read String1:   %s\n", str1 );

        fscanf(fp, "%s %s %s %d", str1,
str2, str3, &year);

        printf("Read String1:   %s\n", str1 );
        printf("Read String2:   %s\n", str2 );
        printf("Read String3:   %s\n", str3 );
        printf("Read Integer:   %d\n",
year );

        fclose(fp);

        return(0);
}
```

```c
  /* Create a sequential file */
#include <stdio.h>

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

   if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      printf( "Enter the account, name, and balance.\n" );
      printf( "Enter EOF to end input.\n" );
      printf( "? " );
      scanf( "%d%s%lf", &account, name, &balance );

      while ( !feof( stdin ) ) {
         fprintf( cfPtr, "%d %s %.2f\n",
                 account, name, balance );
         printf( "? " );
         scanf( "%d%s%lf", &account, name, &balance );
      }

      fclose( cfPtr );
   }

   return 0;
}
```

1. Initialize variables and FILE pointer

1.1 Link the pointer to a file

2. Input data

2.1 Write to file (fprintf)

3. Close file

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
?
```

Program Output

5. Reading Data from a Sequential Access File

- Reading a sequential access file
  - Create a FILE pointer, link it to the file to read

    myPtr = fopen("myFile.dat", "r");
  - Use “fscanf” to read from the file
    - Like scanf, except first argument is a FILE pointer

    fscanf(myPtr, "%d%s%f", &myInt, &myString, &myFloat);
  - Data read from beginning to end
  - File position pointer
    - Indicates number of next byte to be read / written
    - Not really a pointer, but an integer value (specifies byte location), Also called byte offset
  - rewind(myPtr)
    - Repositions file position pointer to **beginning of file** (byte 0)

```c
/* Reading and printing a sequential file */
#include <stdio.h>

int main()
{
   int account;
   char name[ 30 ];
   double balance;
   FILE *cfPtr;   /* cfPtr = clients.dat file pointer */

   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
      printf( "File could not be opened\n" );
   else {
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

      while ( !feof( cfPtr ) ) {
         printf( "%-10d%-13s%7.2f\n", account, name, balance );
         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
      }

      fclose( cfPtr );
   }

   return 0;
}
```

1. Initialize variables

1.1 Link pointer to file

2. Read data (fscanf)

2.1 Print

3. Close file

```
Account    Name          Balance
100        Jones           24.98
200        Doe            345.67
300        White            0.00
400        Stone          -42.16
500        Rich           224.62
```

Program Output

```
1
2   /*  Credit inquiry program */
3   #include <stdio.h>
4
5   int main()
6   {
7      int request, account;
8      double balance;
9      char name[ 30 ];
10     FILE *cfPtr;
11
12     if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL )
13        printf( "File could not be opened\n" );
14     else {
15        printf( "Enter request\n"
16                " 1 - List accounts with zero balances\n"
17                " 2 - List accounts with credit balances\n"
18                " 3 - List accounts with debit balances\n"
19                " 4 - End of run\n? " );
20        scanf( "%d", &request );
21
22        while ( request != 4 ) {
23           fscanf( cfPtr, "%d%s%lf", &account, name,
24                   &balance );
25
26           switch ( request ) {
27              case 1:
28                 printf( "\nAccounts with zero "
29                         "balances:\n" );
30
31                 while ( !feof( cfPtr ) ) {
32
```

1. Initialize variables

2. Open file

2.1 Input choice

2.2 Scan files

3. Print

```
33                   if ( balance == 0 )
34                      printf( "%-10d%-13s%7.2f\n",
35                             account, name, balance );
36
37                   fscanf( cfPtr, "%d%s%lf",
38                          &account, name, &balance );
39                }
40
41                break;
42             case 2:
43                printf( "\nAccounts with credit "
44                        "balances:\n" );
45
46                while ( !feof( cfPtr ) ) {
47
48                   if ( balance < 0 )
49                      printf( "%-10d%-13s%7.2f\n",
50                             account, name, balance );
51
52                   fscanf( cfPtr, "%d%s%lf",
53                          &account, name, &balance );
54                }
55
56                break;
57             case 3:
58                printf( "\nAccounts with debit "
59                        "balances:\n" );
60
61                while ( !feof( cfPtr ) ) {
62
63                   if ( balance > 0 )
64                      printf( "%-10d%-13s%7.2f\n",
```

2.2 Scan files

3. Print

```
65                            account, name, balance );
66
67                      fscanf( cfPtr, "%d%s%lf",
68                             &account, name, &balance );
69                   }
70
71                   break;
72             }
73
74             rewind( cfPtr );
75             printf( "\n? " );
76             scanf( "%d", &request );
77          }
78
79          printf( "End of run.\n" );
80          fclose( cfPtr );
81       }
82
83       return 0;
84    }
```

3.1 Close file

```
Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1
Accounts with zero balances:
300       White            0.00
? 2
Accounts with credit balances:
400       Stone          -42.16
? 3
Accounts with debit balances:
100       Jones           24.98
200       Doe            345.67
500       Rich           224.62
? 4
End of run.
```

Program Output

Binary file reading and writing

```
#include <stdio.h>
struct s
{
char name[50];
int height;
};
int main(){
    struct s a,b;
    FILE *fptr;
    int i;
    fptr=fopen("file.txt","wb");
    for(i=0;i<5;++i)
    {
        //fflush(stdin);
        printf("Enter name: ");
        scanf("%s",a.name);
        printf("Enter height: ");
        scanf("%d",&a.height);
        fwrite(&a,sizeof(a),1,fptr);
    }
    fclose(fptr);
```

```
 fptr=fopen("file.txt","rb");
    while(1)
        {
                if(fread(&b,sizeof(b),1,fptr)==1)
                printf("Name: %s      Height:
%d\n",b.name,b.height);
                else
                break;
        }
    fclose(fptr);
}
```

fseek()

- **fseek**
  - Sets file position pointer to a specific position
  - fseek(FILE *fp, offset, whence);
    - pointer – pointer to file
    - offset – Number of bytes to offset from whence
    - whence – Position from where offset is added
      - SEEK_SET – seek starts at beginning of file
      - SEEK_CUR – seek starts at current location in file
      - SEEK_END – seek starts at end of file

Thank You