Programming Languages -1 (Introduction to C)

files

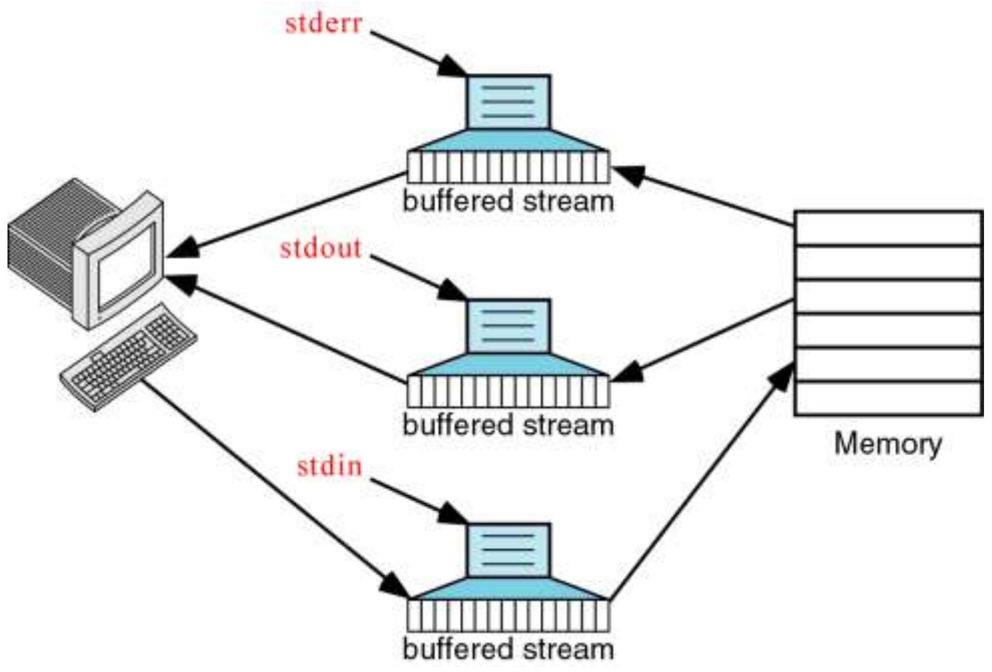
Instructor: M.Fatih AMASYALI
E-mail:mfatih@ce.yildiz.edu.tr

Streams

- I/O done through streams; two kinds: text and binary
  - *Text streams* are sequences of lines, each of which is a sequence of characters terminated by a newline
  - *Binary streams* are sequences of characters corresponding to internal representation of data.
- Streams created by opening files and referenced using stream pointers (FILE *)
- Normally three standard streams are automatically open:
  - **stdin** (stream for standard input - from keyboard)
  - **stdout** (stream for standard output - to screen)
  - **stderr** (stream for standard error output - to screen)

Standard files

- There are three standard file variables in C - **`stdin`**, **`stdout`** and **`stderr`**



- **`stdin`** is linked to the primary input device – usually the keyboard.
- **`stdout`** and **`stderr`** are linked to the primary output device – usually the monitor

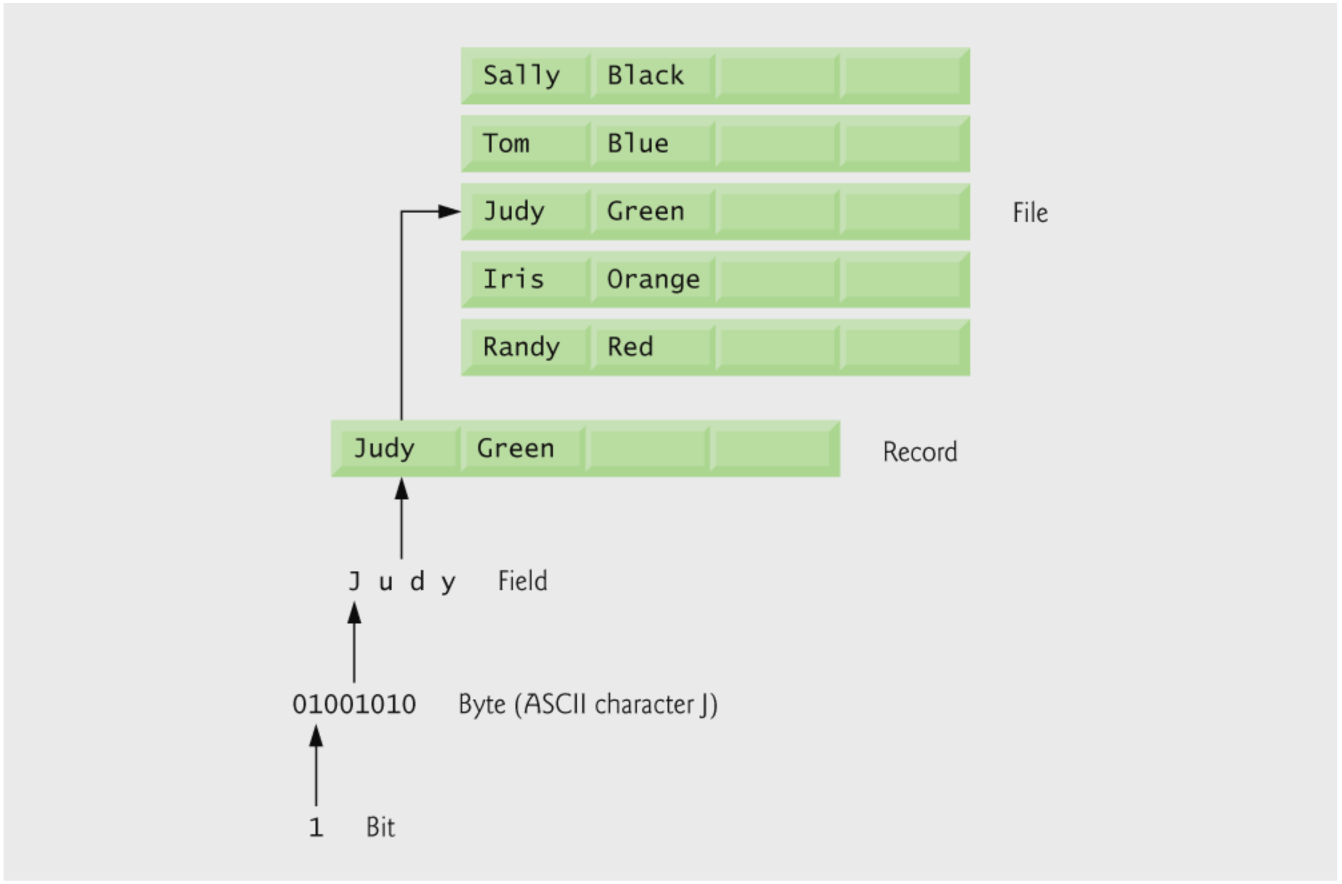
- Data files
  - Can be created, updated, and processed by C programs
  - Are used for permanent storage of large amounts of data
    - Storage of data in variables and arrays is only temporary

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.

Data Hierarchy

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



Data hierarchy.

Files and Streams

- Read/Write functions in standard library
  - `fgetc`
    - Reads one character from a file
    - Takes a `FILE` pointer as an argument
    - `fgetc( stdin )` equivalent to `getchar()`
  - `fputc`
    - Writes one character to a file
    - Takes a `FILE` pointer and a character to write as an argument
    - `fputc( 'a', stdout )` equivalent to `putchar( 'a' )`
  - `fgets`
    - Reads a line from a file
  - `fputs`
    - Writes a line to a file
  - `fscanf` / `fprintf`
    - File processing equivalents of `scanf` and `printf`

Creating a Sequential-Access File

- C imposes no file structure
  - No notion of records in a file
  - Programmer must provide file structure
- Creating a File
  - `FILE *cfPtr;`
    - Creates a `FILE` pointer called `cfPtr`
  - `cfPtr = fopen(“clients.dat", “w”);`
    - Function `fopen` returns a `FILE` pointer to file specified
    - Takes two arguments – file to open and file open mode
    - If open fails, `NULL` returned

Creating a Sequential-Access File

- `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)
- `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

Notes on Filenames

- Unless a directory path is specified, the program will look for the file in the current directory.

- Directory paths in filenames: DOS/Windows

```
sysFile = fopen(“C:\\win\\system.ini”, “r”);
```

- Directory paths in filenames: Unix

```
passFile = fopen(“/usr/etc/passwd”, “r”);
```

```c
#include <stdio.h>

int main( void )
{
   int account;     /* account number */
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */

   FILE *cfPtr;     /* cfPtr = clients.dat file pointer */

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

```

`FILE` pointer definition creates new file pointer

`fopen` function opens a file; **`w`** argument means the file is opened for writing

```
      /* write account, name and balance into file with fprintf */
      while ( !feof( stdin ) ) {
         fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
         printf( "? " );
         scanf( "%d%s%lf", &account, name, &balance );
      } /* end while */

      fclose( cfPtr ); /* fclose closes file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`feof` returns true when end of file is reached

`fprintf` writes a string to a file

`fclose` closes a 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
? ^Z
```

Operating system	Key combination
Linux/Mac OS X/UNIX	*<Ctrl> d*
Windows	*<Ctrl> z*

End-of-file key combinations for various popular operating systems.

Good Programming Practice

- Explicitly close each file as soon as it is known that the program will not reference the file again.
- Closing a file can free resources for which other users or programs may be waiting.

Error-Prevention Tip

- Open a file only for reading (and not update) if the contents of the file should not be modified. This prevents unintentional modification of the file’s contents. This is another example of the principle of least privilege.

Mode	Description
`r`	Open an existing file for reading.
`w`	Create a file for writing. If the file already exists, discard the current contents.
`a`	Append; open or create a file for writing at the end of the file.
`r+`	Open an existing file for update (reading and writing).
`w+`	Create a file for update. If the file already exists, discard the current contents.
`a+`	Append: open or create a file for update; writing is done at the end of the file.
`rb`	Open an existing file for reading in binary mode.
`wb`	Create a file for writing in binary mode. If the file already exists, discard the current contents.
`ab`	Append; open or create a file for writing at the end of the file in binary mode.
`rb+`	Open an existing file for update (reading and writing) in binary mode.
`wb+`	Create a file for update in binary mode. If the file already exists, discard the current contents.
`ab+`	Append: open or create a file for update in binary mode; writing is done at the end of the file.

File opening modes.

Reading Data from a Sequential-Access File

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

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

    ```
    fscanf( cfPtr, "%d%s%f", &accounnt, name, &balance );
    ```
  - 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( cfPtr )`
    - Repositions file position pointer to beginning of file (offset=0)

Reading all numbers **from a file** and then prints their average onto the screen

```
#include<stdio.h>

int main(void)
{ FILE *fpTemps;
  int number;
  int count;
  int sum;
  float avrg;

  sum=0;
  count=0;

  fpTemps = fopen("TEMPS.DAT", "r");

  while ( feof(fpTemps)!= 0 )
  {
   fscanf(fpTemps,"%d",&number);
   sum += number;
   count++;
  }

  fclose(fpTemps);

  avrg = (float)sum/(float)count;
  printf("%.2f", avrg);
  return 0;
}
```

```c
#include <stdio.h>

int main( void )
{
   int account;     /* account number */
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */

   FILE *cfPtr;     /* cfPtr = clients.dat file pointer */

   /* fopen opens file; exits program if file cannot be opened */
   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else { /* read account, name and balance from file */
      printf( "%-10s%-13s%s\n", "Account", "Name", "Balance" );
      fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

```

`fopen` function opens a file; **`r`** argument means the file is opened for reading

```c
      /* while not end of file */
      while ( !feof( cfPtr ) ) {
         printf( "%-10d%-13s%7.2f\n", account, name, balance );
         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );
      } /* end while */

      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`fscanf` function reads a string from a file

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

```c
/*
   Credit inquiry program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int request;     /* request number */
   int account;     /* account number */
   double balance;  /* account balance */
   char name[ 30 ]; /* account name */
   FILE *cfPtr;     /* clients.dat file pointer */

   /* fopen opens the file; exits program if file cannot be opened */
   if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
      printf( "File could not be opened\n" );
   } /* end if */
   else {

      /* display request options */
      printf( "Enter request\n"
         " 1 - List accounts with zero balances\n"
         " 2 - List accounts with credit balances\n"
         " 3 - List accounts with debit balances\n"
         " 4 - End of run\n? " );
      scanf( "%d", &request );

      /* process user's request */
      while ( request != 4 ) {

         /* read account, name and balance from file */
         fscanf( cfPtr, "%d%s%lf", &account, name, &balance );

         switch ( request ) {

            case 1:
               printf( "\nAccounts with zero balances:\n" );

               /* read file contents (until eof) */
               while ( !feof( cfPtr ) ) {

                  if ( balance == 0 ) {
                     printf( "%-10d%-13s%7.2f\n",
                        account, name, balance );
                  } /* end if */

                  /* read account, name and balance from file */
                  fscanf( cfPtr, "%d%s%lf",
                     &account, name, &balance );
               } /* end while */

               break;

            case 2:
               printf( "\nAccounts with credit balances:\n" );

               /* read file contents (until eof) */
               while ( !feof( cfPtr ) ) {

                  if ( balance < 0 ) {
                     printf( "%-10d%-13s%7.2f\n",
                        account, name, balance );
                  } /* end if */

                  /* read account, name and balance from file */
                  fscanf( cfPtr, "%d%s%lf",
                     &account, name, &balance );
               } /* end while */

               break;

            case 3:
               printf( "\nAccounts with debit balances:\n" );

               /* read file contents (until eof) */
               while ( !feof( cfPtr ) ) {

                  if ( balance > 0 ) {
                     printf( "%-10d%-13s%7.2f\n",
                        account, name, balance );
                  } /* end if */

```

```
                  /* read account, name and balance from file */
                  fscanf( cfPtr, "%d%s%lf",
                     &account, name, &balance );
               } /* end while */

               break;

         } /* end switch */

         rewind( cfPtr ); /* return cfPtr to beginning of file */

         printf( "\n? " );
         scanf( "%d", &request );
      } /* end while */

      printf( "End of run.\n" );
      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`rewind` function moves the file pointer back to the beginning of the file

Output

```
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.
```

Notes on Strings and `fscanf()`

- Reading in a string:

  `fscanf(`*`stream`*`, “%s”, `*`string`*`)`

  – Reads only a "word" at a time.

  – Words are separated by a *white-space*: (space, tab, newline, or any combination of these)

  – Moves to the next word in the stream automatically after each read.

- **`scanf(“%s”, `*`string`*`)`**

  – behaves similarly, except input stream is **`stdin`**.

  – **`eg:== fscanf(stdin, “%s”, `*`string`*`)`**

Checking for EOF

- Both **`scanf()`** and **`fscanf()`** return:
  - the number of input items converted and assigned successfully
  - or the constant value **`EOF`** when an error or end-of-file occurs,

Checking for EOF

- To check for end-of-file (or any other input error), check that the **number of items** converted and assigned successfully is **equal** to the **expected** number of items.

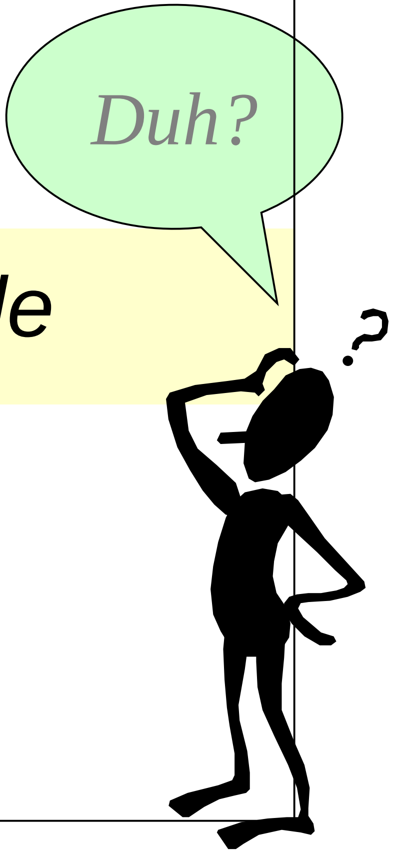
```
while ( fscanf(inpf, "%s %f", name, &mark) == 2 )
{
  printf("%s\t %f\n", name, mark);
}
```

Example: Count Words

- Write a program which counts the number of "words" in a file.
  - Note that as far as **`scanf()`** and **`fscanf()`** are concerned, any sequence of non-whitespace characters is a "word."

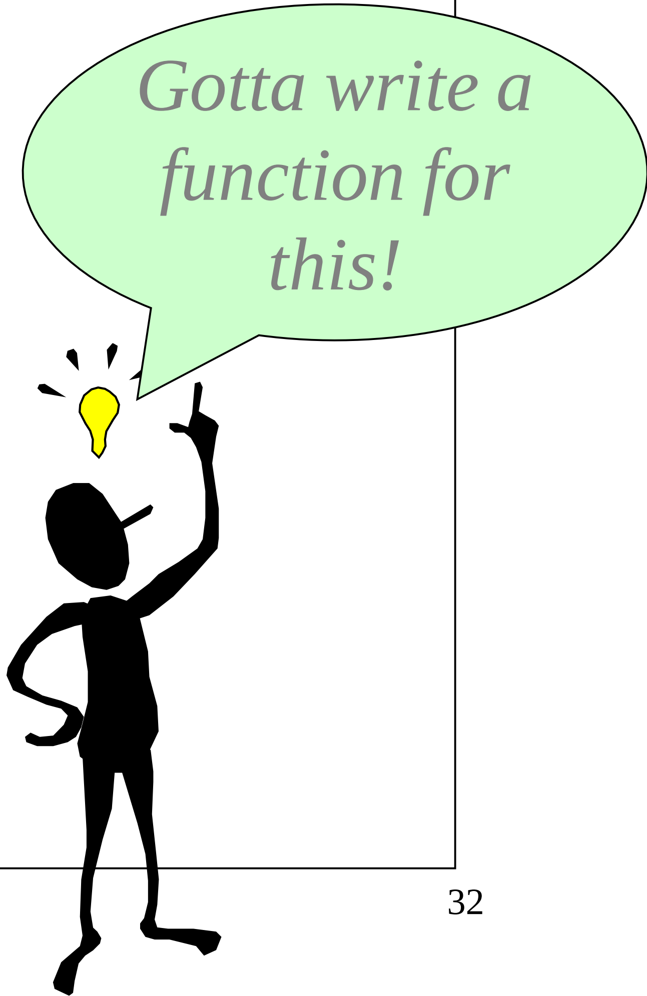
Count Words: Algorithm

```
ask the user for the name of the file
open the file
check if file is opened successfully

count the number of words in the file

print out the count
close the file
```



Count Words: Algorithm

```
set count to 0
loop
{
    read a word from the file
    if attempt to read a word failed
    then { exit loop }
    add 1 to count
}
```



Function: `countWords()`

- Function prototype:

```
int countWords ( FILE *inpf );
```

- Description:
  - This function returns the number of "words" in the input stream **`inpf`**.

Function: `countWords()`

- PRE-Conditions:
  - It assumes that **`inpf`** is a pointer to a file which has been opened successfully. There is no check for that within the function.
  - It also assumes that the file position is at the start of the input file/stream.
  - Note that a "word" here means any string of characters, separated from other words by any whitespace (ie. space, tab, newline, or combination).
  - It assumes that no "word" in the file has more than (**MAXLEN -** 1) characters.

Function: `countWords()`

- POST-Conditions:
  - At the end of the function, the file position will be at the end of file.
  - The function returns an integer value which is the number of "words" in the file.

Function: `countWords()`

```
int
countWords ( FILE *inpf )
{
  char  word[MAXLEN];
  int   count = 0;

  while ( fscanf(inpf, "%s", word) == 1 )
  {
    count++;
  }

  return count;
}
```

Count Words: Algorithm

ask the user for the name of the file

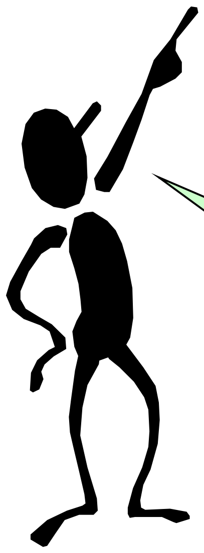
open the file

check if file is opened successfully

count the number of words in the file

prin

clos



I can also write a reusable function for these!

Function: *openInput()*

- Function prototype:

```
FILE* openInput ( void );
```

- Description:
  - This function keeps asking the user for a filename, until it is able to open the file successfully for input.

Function: `openInput()`

- PRE-Condition:
  - It assumes that the filename fits in a string of size MAXLEN (including the `'\0'`).

Function: `openInput()`

- POST-Conditions:
  - **It can cause the program to terminate if the user chooses to abort the operation**.
  - It returns the file handler/pointer for the specified file.
  - It assumes that the calling function has the corresponding variable to catch the return value.
  - It also assumes that the calling function takes care of closing the file.

```
FILE*  openInput ( void )
{
  FILE   *handle = NULL;
  char   theFile[MAXLEN];
  int    option;

  while (1)
  {
    printf("Enter file name: ");
    scanf("%s", theFile);

    if ((handle = fopen(theFile, "r")) == NULL )
    {
     /* Insert code to handle open error. */
    }
    else
    {
      break;
    }
  }
  return handle;
}
```

Code to handle open error:

```
printf("Error opening file.\n");

option = 0; /* Set default to abort. */

printf("\nEnter 1 to try again, ");
printf("or any number to abort: ");

scanf("%d", &option);
printf("\n");

if ( option != 1 )
{
  printf("Alright then. ");
  printf("Program terminated.\n");
  exit(1);
}
```

Main Algorithm

```
set *file* to be the result of **openInput()**

set *count* to the result of **countWords(file)**

print out the *count*
close the *file*
```

Test Program #1

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

#include "countwords.h"
#include "countwords.c“
#include “openInput.h"
#include "openInput c"

int main()
{
  FILE  *inputFile = NULL;
  int   count;

  inputFile = openInput();
  count = countWords(inputFile);
  printf("\nThere are %d words in the file.\n", count);

  fclose(inputFile);

  return 0;
}
```

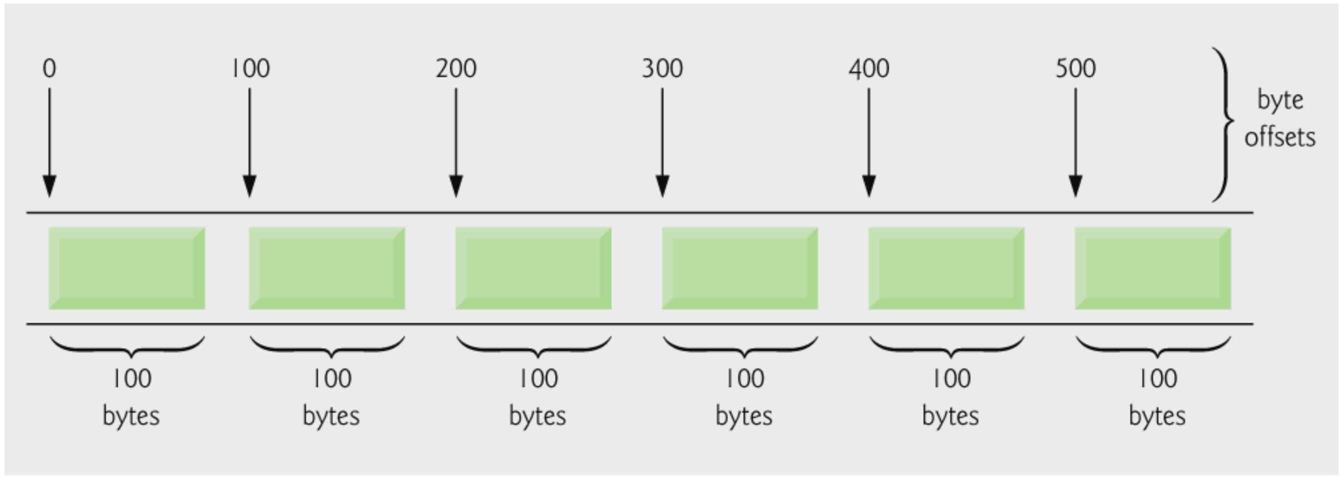
Test Program #2

```
#include <stdio.h>
#include <stdlib.h>
#include "countwords.h"
#include "countwords.c"
#include "openInput.h"
#include "openInput c"

int main()
{
  FILE  *inputFile = NULL;
  int   count;

  inputFile = openInput();

  count = countWords(inputFile);
  printf("\nThere are %d words in the file.\n", count);
  count = countWords(inputFile);
  printf("\nThere are %d words in the file.\n", count);

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

What is the result if we call the `countWords()` function a second time over the same file?

Random-Access Files

- Random access files
  - Access individual records without searching through other records
  - Instant access to records in a file
- Implemented using fixed length records
  - Sequential files do not have fixed length records



C’s view of a random-access file.

Creating a Random-Access File

- Data in random access files
  - Unformatted (stored as "raw bytes")
    - All data of the same type (`int`s, for example) uses the same amount of memory
    - All records of the same type have a fixed length
    - Data not human readable

Creating a Random-Access File

- Unformatted I/O functions
  - `fwrite`
    - Transfer bytes from a location in memory to a file
  - `fread`
    - Transfer bytes from a file to a location in memory
  - Example:

```
fwrite( &number, sizeof( int ), 1, myPtr );
```

- `&number` – Location to transfer bytes from
- `sizeof( int )` – Number of bytes to transfer
- `1` – For arrays, number of elements to transfer
  - In this case, "one element" of an array is being transferred
- `myPtr` – File to transfer to or from

Creating a Random-Access File

- Writing `structs`

  ```
  fwrite( &myObject, sizeof (struct
    myStruct), 1, myPtr );
  ```

  - `sizeof` – returns size in bytes of object in parentheses
- To write several array elements
  - Pointer to array as first argument
  - Number of elements to write as third argument

```c
/*
   Creating a random-access file sequentially */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   int acctNum;          /* account number */
   char lastName[ 15 ];  /* account last name */
   char firstName[ 10 ]; /* account first name */
   double balance;       /* account balance */
}; /* end structure clientData */

int main( void )
{
   int i; /* counter used to count from 1-100 */

   /* create clientData with default information */
   struct clientData blankClient = { 0, "", "", 0.0 };

   FILE *cfPtr; /* credit.dat file pointer */

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( cfPtr = fopen( "credit.dat", "wb" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
   else {

      /* output 100 blank records to file */
      for ( i = 1; i <= 100; i++ ) {
         fwrite( &blankClient, sizeof( struct clientData ), 1, cfPtr );
      } /* end for */

      fclose ( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`fopen` function opens a file; **`wb`** argument means the file is opened for writing in binary mode

`fwrite` transfers bytes into a random-access file

Writing Data Randomly to a Random-Access File

- `fseek`
  - Sets file position pointer to a specific position
  - `fseek(` *pointer*, *offset*, *symbolic_constant* `);`
    - *pointer* – pointer to file
    - *offset* – file position pointer (0 is first location)
    - *symbolic_constant* – specifies where in file we are reading from
    - `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

```c
/*
   Writing to a random access file */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   int acctNum;          /* account number */
   char lastName[ 15 ];  /* account last name */
   char firstName[ 10 ]; /* account first name */
   double balance;       /* account balance */
}; /* end structure clientData */

int main( void )
{
   FILE *cfPtr; /* credit.dat file pointer */

   /* create clientData with default information */
   struct clientData client = { 0, "", "", 0.0 };

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
   else {

      /* require user to specify account number */
      printf( "Enter account number"
         " ( 1 to 100, 0 to end input )\n? " );
      scanf( "%d", &client.acctNum );

      /* user enters information, which is copied into file */
      while ( client.acctNum != 0 ) {

         /* user enters last name, first name and balance */
         printf( "Enter lastname, firstname, balance\n? " );

         /* set record lastName, firstName and balance value */
         fscanf( stdin, "%s%s%lf", client.lastName,
            client.firstName, &client.balance );

         /* seek position in file to user-specified rec
         fseek( cfPtr, ( client.acctNum - 1 ) *
            sizeof( struct clientData ), SEEK_SET );

         /* write user-specified information in file */
         fwrite( &client, sizeof( struct clientData ), 1, cfPtr );

         /* enable user to input another account number */
         printf( "Enter account number\n? " );
         scanf( "%d", &client.acctNum );
      } /* end while */

      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`fseek` searches for a specific location in the random-access file

Output

```
Enter account number ( 1 to 100, 0 to end input )
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

Reading Data from a Random-Access File

- `fread`
  - Reads a specified number of bytes from a file into memory

    ```
    fread( &client, sizeof (struct clientData),
      1, myPtr );
    ```
  - Can read several fixed-size array elements
    - Provide pointer to array
    - Indicate number of elements to read
  - To read multiple elements, specify in third argument

```c
/*
   Reading a random access file sequentially */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   int acctNum;          /* account number */
   char lastName[ 15 ];  /* account last name */
   char firstName[ 10 ]; /* account first name */
   double balance;       /* account balance */
}; /* end structure clientData */

int main( void )
{
   FILE *cfPtr; /* credit.dat file pointer */

   /* create clientData with default information */
   struct clientData client = { 0, "", "", 0.0 };

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( cfPtr = fopen( "credit.dat", "rb" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
   else {
      printf( "%-6s%-16s%-11s%10s\n", "Acct", "Last Name",
         "First Name", "Balance" );

      /* read all records from file (until eof) */
      while ( !feof( cfPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1, cfPtr );

         /* display record */
         if ( client.acctNum != 0 ) {
            printf( "%-6d%-16s%-11s%10.2f\n",
               client.acctNum, client.lastName,
               client.firstName, client.balance );
         } /* end if */

      } /* end while */

      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */
```

`fread` reads bytes from a random-access file to a location in memory

```
Acct  Last Name       First Name    Balance
29    Brown           Nancy          -24.54
33    Dunn            Stacey         314.33
37    Barker          Doug             0.00
88    Smith           Dave           258.34
96    Stone           Sam             34.98
```

```c
/*
   This program reads a random access file sequentially, updates data
   already written to the file, creates new data to be placed in the
   file, and deletes data previously in the file. */
#include <stdio.h>

/* clientData structure definition */
struct clientData {
   int acctNum;          /* account number */
   char lastName[ 15 ];  /* account last name */
   char firstName[ 10 ]; /* account first name */
   double balance;       /* account balance */
}; /* end structure clientData */

/* prototypes */
int enterChoice( void );
void textFile( FILE *readPtr );
void updateRecord( FILE *fPtr );
void newRecord( FILE *fPtr );
void deleteRecord( FILE *fPtr );

int main( void )
{
   FILE *cfPtr; /* credit.dat file pointer */
   int choice;  /* user's choice */

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( cfPtr = fopen( "credit.dat", "rb+" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
   else {

      /* enable user to specify action */
      while ( ( choice = enterChoice() ) != 5 ) {

         switch ( choice ) {

            /* create text file from record file */
            case 1:
               textFile( cfPtr );
               break;

            /* update record */
            case 2:
               updateRecord( cfPtr );
               break;

            /* create record */
            case 3:
               newRecord( cfPtr );
               break;

            /* delete existing record */
            case 4:
               deleteRecord( cfPtr );
               break;

            /* display message if user does not select valid choice */
            default:
               printf( "Incorrect choice\n" );
               break;

         } /* end switch */

      } /* end while */

      fclose( cfPtr ); /* fclose closes the file */
   } /* end else */

   return 0; /* indicates successful termination */

} /* end main */

/* create formatted text file for printing */
void textFile( FILE *readPtr )
{
   FILE *writePtr; /* accounts.txt file pointer */

   /* create clientData with default information */
   struct clientData client = { 0, "", "", 0.0 };

   /* fopen opens the file; exits if file cannot be opened */
   if ( ( writePtr = fopen( "accounts.txt", "w" ) ) == NULL ) {
      printf( "File could not be opened.\n" );
   } /* end if */
```

Function **`textFile`** creates a text file containing all account data

```
   else {
      rewind( readPtr ); /* sets pointer to beginning of file */
      fprintf( writePtr, "%-6s%-16s%-11s%10s\n",
         "Acct", "Last Name", "First Name","Balance" );

      /* copy all records from random-access file into text file */
      while ( !feof( readPtr ) ) {
         fread( &client, sizeof( struct clientData ), 1, readPtr );

         /* write single record to text file */
         if ( client.acctNum != 0 ) {
            fprintf( writePtr, "%-6d%-16s%-11s%10.2f\n",
               client.acctNum, client.lastName,
               client.firstName, client.balance );
         } /* end if */

      } /* end while */

      fclose( writePtr ); /* fclose closes the file */
   } /* end else */

} /* end function textFile */

/* update balance in record */
void updateRecord( FILE *fPtr )
{
  int account;        /* account number */
  double transaction; /* transaction amount */

```

Function **`updateRecord`** changes the balance of a specified account

```c
/* create clientData with no information */
struct clientData client = { 0, "", "", 0.0 };

/* obtain number of account to update */
printf( "Enter account to update ( 1 - 100 ): " );
scanf( "%d", &account );

/* move file pointer to correct record in file */
fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
   SEEK_SET );

/* read record from file */
fread( &client, sizeof( struct clientData ), 1, fPtr );

/* display error if account does not exist */
if ( client.acctNum == 0 ) {
   printf( "Acount #%d has no information.\n", account );
} /* end if */
else { /* update record */
   printf( "%-6d%-16s%-11s%10.2f\n\n",
      client.acctNum, client.lastName,
      client.firstName, client.balance );

   /* request transaction amount from user */
   printf( "Enter charge ( + ) or payment ( - ): " );
   scanf( "%lf", &transaction );
   client.balance += transaction; /* update record balance */

```

```
      printf( "%-6d%-16s%-11s%10.2f\n",
         client.acctNum, client.lastName,
         client.firstName, client.balance );

      /* move file pointer to correct record in file */
      fseek( fPtr, ( account - 1 ) * sizeof( struct clientData ),
         SEEK_SET );

      /* write updated record over old record in file */
      fwrite( &client, sizeof( struct clientData ), 1, fPtr );
   } /* end else */

} /* end function updateRecord */

/* delete an existing record */
void deleteRecord( FILE *fPtr )
{

   struct clientData client; /* stores record read from file */
   struct clientData blankClient = { 0, "", "", 0 }; /* blank client */

   int accountNum; /* account number */

   /* obtain number of account to delete */
   printf( "Enter account number to delete ( 1 - 100 ): " );
   scanf( "%d", &accountNum );

```

Function **deleteRecord** removes an existing account from the file

```
   /* move file pointer to correct record in file */
   fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
      SEEK_SET );

   /* read record from file */
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   /* display error if record does not exist */
   if ( client.acctNum == 0 ) {
      printf( "Account %d does not exist.\n", accountNum );
   } /* end if */
   else { /* delete record */

      /* move file pointer to correct record in file */
      fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
         SEEK_SET );

      /* replace existing record with blank record */
      fwrite( &blankClient,
         sizeof( struct clientData ), 1, fPtr );
   } /* end else */

} /* end function deleteRecord */

```

```c
/* create and insert record */
void newRecord( FILE *fPtr )
{
   /* create clientData with default information */
   struct clientData client = { 0, "", "", 0.0 };

   int accountNum; /* account number */

   /* obtain number of account to create */
   printf( "Enter new account number ( 1 - 100 ): " );
   scanf( "%d", &accountNum );

   /* move file pointer to correct record in file */
   fseek( fPtr, ( accountNum - 1 ) * sizeof( struct clientData ),
      SEEK_SET );

   /* read record from file */
   fread( &client, sizeof( struct clientData ), 1, fPtr );

   /* display error if account already exists */
   if ( client.acctNum != 0 ) {
      printf( "Account #%d already contains information.\n",
         client.acctNum );
   } /* end if */
```

Function **`newRecord`** adds a new account to the file

```
   else { /* create record */

      /* user enters last name, first name and balance */
      printf( "Enter lastname, firstname, balance\n? " );
      scanf( "%s%s%lf", &client.lastName, &client.firstName,
         &client.balance );

      client.acctNum = accountNum;

      /* move file pointer to correct record in file */
      fseek( fPtr, ( client.acctNum - 1 ) *
         sizeof( struct clientData ), SEEK_SET );

      /* insert record in file */
      fwrite( &client,
         sizeof( struct clientData ), 1, fPtr );
   } /* end else */

} /* end function newRecord */

```

```c
/* enable user to input menu choice */
int enterChoice( void )
{
  int menuChoice; /* variable to store user's choice */

  /* display available options */
  printf( "\nEnter your choice\n"
     "1 - store a formatted text file of acounts called\n"
     "    \"accounts.txt\" for printing\n"
     "2 - update an account\n"
     "3 - add a new account\n"
     "4 - delete an account\n"
     "5 - end program\n? " );

  scanf( "%d", &menuChoice ); /* receive choice from user */

  return menuChoice;

} /* end function enterChoice */
```

Referance

- Ioannis A. Vetsikas, Lecture notes
- Dale Roberts, Lecture notes