Ch 11 - C File Processing

Today we talked about section 11.4, reading from a text file, often called a "sequential access" file.

The file i/o functions come from <stdio.h>

```
FILE * cfPtr;  // declare a file pointer
```

Opening a file

```
fopen( "filename", "mode" )
```

"r" for read only mode

"w" for write

"a" for append

fopen returns a pointer to a file, if the open operation was successful. It returns NULL if not successful.

Reading from the file

Once the file is opened, we can read from the file with

```
fscanf( file pointer, control string, addresses . . .)
```

scanf and fscanf work in a similar way.

Both functions are called variable arity functions.

arity - the number of parameters expected by a function.

fscanf has a feature that sets the position of the byte that was last read.

fscanf returns an integer that is the number of values it successfully read.

Checking for the end of the file

We can check to see if the position has reached the end of the file.

feof(file pointer) - returns true if file position is at the end of the file. Otherwise (more data to read) it returns false.

```
if ( feof(cfPtr) )

   printf("At the end of the file\n");
```

These boolean functions return

0 for false
a non-zero number for true

if (feof(cfPtr) != 0) There is no need to include this comparison.

Example program (pg 448) : read and print from a sequential file

```
#include <stdio.h>
int main( void ) {

   unsigned int account;
   char name[30];
   double balance;

   FILE * cfPtr;

   if (( cfPtr = fopen("clients.dat","r")) == NULL )

      puts( "File could not be opened." );

   else {

      printf("%-10s%-13s%s\n", "Account", "Name",
             "Balance");

        // output will look like
        Account   Name         Balance
        100       Jones          24.98

      fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );


      while ( !feof( cfPtr ) ) {

         printf( "%-10d%-13s%7.2f\n", account, name, balance );
         fscanf( cfPtr, "%d%29s%lf", &account, name, &balance );

      } // end while


      // close the file

      fclose( cfPtr );
   } // end else

} // end main
```

We can also use fgets to read strings from a file and fgetc to read characters.

Here are the function descriptions from Tutorials Point:

int fgetc(FILE *stream) Gets the next character (an unsigned char) from the specified stream and advances the position indicator for the stream.
char *fgets(char *str, int n, FILE *stream) Reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

There are similar functions that read from the keyboard:

int getchar(void) Gets a character (an unsigned char) from stdin.
char *gets(char *str) Reads a line from stdin and stores it into the string pointed to by, str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

C Pointers

We can pass a pointer to a pointer as a parameter.

Here's the problem to be solved.

1) create an array of strings
2) input the strings from the user
3) sort the strings, using a separate function for swapping

Okay so the first step is to create the array of strings. In C this means creating an array of character pointers, then dynamically allocating space for each string. Once the space is allocated, we can use the gets function to input a string.

```
char * stringArray[ 10 ]; // declaration

// read 10 Strings from the user
// and store their addresses in the array

int n;
for (n = 0; n < 10; n++) {

   // dynamically allocate space for the string
   stringArray[n] = (char *) malloc( 50 * sizeof(char) );
   gets( stringArray[n] );

}
```