File Input/Output

Up to this point, you have been testing your programs by entering input from the keyboard. Although this works fine for small sets of input, this would be very time consuming for processing large amounts of data. Furthermore, large amounts of data often already exist in text files. It would certainly be wasteful to type these data in by hand while running a program when they are already available in a file.

As you might imagine, C provides the ability to read from files, (AND write to files.) In fact, when we read information from the keyboard and wrote information to the screen, we primarily used the functions

```
printf
scanf
```

Similarly, for reading from files and writing to files, we'll use the functions

```
fprintf
fscanf
```

The first f in each of these function calls stands for "file."

Here is the specification for each function:

```
fprintf(file_ptr, ctrl_str, other_arguments)
fscanf(file_ptr, ctrl_str, other_arguments)
```

You'll notice that these are identical to printf and scanf EXCEPT for the first parameter.

How to Create a File Pointer

In order to read from a file, or write to a file, you MUST use a pointer to that file. Here is a declaration of a file pointer:

```
FILE *ifp;
```

In order to properly "initialize" an file pointer, it must be set to point to a file. In order to do this, we must specify the following:

1) Name of the file
2) Mode ("r" for read or "w" for write)

There is a function call that uses this information to open the appropriate file and return a pointer to it. It's name is fopen. Here is an example of its use:

```
ifp = fopen("input.txt", "r");
```

You'll notice that the first parameter to the fopen function is a string storing the name of the file to be opened. The second parameter is also a string. For our purposes, this string will either be "r" or "w". (There are other possibilities for this second parameter, but we won't deal with them in this class.) When we open a file in "r" (reading) mode, the file should already exist and the fopen function returns a pointer to the beginning of that file. If the file doesn't exist, fopen returns NULL.

How to Read from an Input File

This works nearly the same as reading from the keyboard. In fact, imagine pre-typing all of your responses you would type in the keyboard into a file and then running a program that read from that file instead of the keyboard. In that situation, the new program would work identically.

Every time you use the fscanf function to read in a piece of information from a file, the file pointer reads in the next token, returns it and advances to the following token. Here is an example of how to read in an integer from the file we previously opened:

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

The first parameter tells the computer to look to where ifp is pointing instead of the keyboard. The rest works the same.

Imagine that the file we just opened, "input.txt" contains several integers separated by white space, with the end of the data being signified by a 0. Here is how we would read in and sum all the numbers from the file:

```
FILE *ifp;
int num = -1, sum = 0;

ifp = fopen("input.txt", "r");

while (num != 0) {
  fscanf(ifp, "%d", &num);
  sum += num;
}

fclose(ifp);
```

Closing a file

You'll notice that the last line above uses a new function call, fclose. All files that are opened must also be closed. In order to close a file, you must simply pass the file pointer to the file you want to close into the fclose function. Forgetting to close a file may corrupt the contents of that file.

Here is a complete program that reads in a file containing numbers and outputs their sum to the screen:

```
#include stdio.h

int main() {

  FILE *ifp;
  int num = -1, sum = 0;

  ifp = fopen("input.txt", "r");

  while (num != 0) {
    fscanf(ifp, "%d", &num);
    sum += num;
  }

  fclose(ifp);

  printf("The sum is %d\n", sum);

  return 0;
}
```

Example: Writing to a File

Imagine that you wanted to write some output to a file. Consider editing the program we looked at a couple lectures ago that counted up the frequencies of each alphabetic character in a file and printed out a chart to the screen so that it wrote the chart into a file. Here are the changes we would make to the main function and the printchart function:

```
int main() {

  int freq[26];
  FILE *ofp;

  init(freq);
  readinput(freq);

  ofp = fopen("charfreq.txt", "w");
  printchart(freq, ofp);
  fclose(ofp);

  return 0;
}

void printchart(int freq[], FILE *ofp) {
  int index;

  fprintf(ofp,"Letter\tFrequency\n");
  for (index = 0; index<26; index++) {

    fprintf(ofp,"%c\t%d\n",(char)('a'+index),
                               freq[index]);
  }
}
```

Secret Code Message Example

One of the types of codes employed to hide the meaning of a message many, many years ago involved substituting the first letter in the alphabet for the last, the second letter for the second to last, etc.

Essentially, the following chart shows a list of substitutions:

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M
Code	Z	Y	X	W	V	U	T	S	R	Q	P	O	N

Now, imagine writing a program that took in a plain English file and outputted a file containing only the letters in the previous file in code, according to the code above. Here are the steps we'd have to go through:

1) Open both files.
2) Read from the input file character by character.
3) For each character, if it is alphabetic, determine it's corresponding code letter and output that to the output file.

To make the output file a bit neater, we will place a newline after 60 characters have been written to a line.

In this example, the functions fgetc and fputc will be used. Here are the prototypes for both:

```
// Gets the next character from the file
// pointed to by fp and returns it.
int fgetc(FILE *fp);

// Writes c to the file pointed to by fp.
int fputc(int c, FILE *fp);
```

```
#include <stdio.h>

int main() {

  FILE *ifp, *ofp;
  int cnt = 0, c, codechar;

  // Open both the input and output files.
  ifp = fopen("fruit.c","r");
  ofp = fopen("fruitcode.txt", "w");

  // Continue reading in characters till the
  //end of the input file.
  while ((c =fgetc(ifp)) != EOF) {

    // Only process alphabetic characters.
    if (isalpha(c)) {

      // Write out the encoded character to
      // the output file.
      codechar = ('Z' - toupper(c)) + 'A';
      fputc(codechar, ofp);

      // Add a newline character if 60
      //characters have been written to
      // a line.
      cnt++;
      if (cnt%60 == 0)
        fputc('\n', ofp);
    }
  }
  fclose(ifp); // Close both files.
  fclose(ofp);
}
```

Questions to Think About

1) How does the encoding work in the line that assigns codechar?

2) How does the adding of the new line work?

3) How can we adapt this program to include all non-alphabetic characters unchanged and not change the case of all the alphabetic characters?

4) If we did #3, could we decode the file exactly?