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: Jetski Problem

In the following problem, you are to read some information from a file about Skippy and his jetski. In particular, you are given how fast Skippy can ride on his jetski and how long is going to ride. From this information, you are to calculate the area of the zone where Skippy may be at the end of his ride. (Hint: The zone is a semi-circle, do you know why?)

The full text of the problem is linked as a separate file off the lecture notes. For our purposes, we only need to know the name of the file ("jetski.in") and the format of the file.

The first line of the file will contain a single positive integer, n, representing the number of situations our program must calculate the area of the zone.

The following n lines will contain two integers each, the first representing the speed of Skippy's jetski in miles per hour, and the second representing the number of minutes Skippy rides his jetski.

For each input case, we must output a single line with the following format:

Day k: X

where k is the number of the test case (starting from 1), and X is the area in square miles that Skippy can reach rounded to two decimal places.

Here is a sample file:

```
2
30 60
5 120
```

Here is the output our program should produce that corresponds to this input file:

```
Day 1: 1413.72
Day 2: 157.08
```

Before we write the program, let's consider the major steps we need to think about:

1) We must first open the input file.
2) Then, we must read in the number of input cases into a variable.
3) Next, a loop must be set up to process each input case.
4) In the loop, we must first read in the two pieces of information for the case.

These first four steps have nothing to do with problem solving. Rather, the entire problem gets solved inside of this loop:

5) We must calculate the number of hours Skippy rides his jetski.
6) Then we must calculate how far he can travel from his starting point on the coast. (Distance = Rate x Time).
7) Finally, we must calculate the area of the semicircle where Skippy could have gone (Area = 1/2 x PI x radius x radius.)

Now, let's look at the program:

```
#include <stdio.h>

#define PI 3.1415926535898

int main() {

    FILE *ifp;
    int numcases;
    int day;
    int mph, time;
    double radius;
    double area;

    ifp = fopen("jetski.in", "r");

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

    for (day=1; day<=numcases; day++) {

       fscanf(ifp,"%d%d", &mph, &time);
       radius = mph*time/60.0;
       area = .5*PI*radius*radius;

       printf("Day %d: %.2lf\n", day, area);
    }

    fclose(ifp);
    system("PAUSE");
    return 0;
}
```

Example: Writing to a File

Consider a situation were you had an input file with data about the number of boxes of cookies each girl scout in a troop had sold. In the data file, the first line stores the number of girls, *n*, in the troop and the following *n* lines had two pieces of information separated by a space on them each: The name of a girl in the troop, followed by how many boxes of cookies she had sold.

In order to interpret the data more easily, you would like a visual chart with each row represents one of the girls in the troop, and the rest of the row has one star for each box of cookies she sold. In essence, when all put together, this chart would be a bar graph.

Here is a sample input file:

```
5
Abby 6
Jane 4
Madison 12
Sarah 11
Zoe 8
```

For this input file, we would like to create the following output file:

```
5
Abby        ******
Jane        ****
Madison     ************
Sarah       ***********
Zoe         ********
```

We will create a program that reads from the input file "cookie.txt" and writes the bar graph output to a file called "cookiegraph.txt"

```
#include <stdio.h>

int main() {

  FILE *ifp, *ofp;
  int numgirls, index, numboxes, stars;
  char name[20];

  ifp = fopen("cookie.txt","r");
  ofp = fopen("cookiegraph.txt", "w");

  fscanf(ifp, "%d", &numgirls);
  fprintf(ofp, "%d\n", numgirls);

  for (index=0; index<numgirls; index++) {

    fscanf(ifp, "%s", &name);
    fprintf(ofp, "%s\t", name);
    fscanf(ifp, "%d", &numboxes);

    for (stars=0; stars<numboxes; stars++)
      fprintf(ofp,"*");

    fprintf(ofp, "\n");
  }

  fclose(ifp);
  fclose(ofp);
  return 0;
}
```