Course Notes Set 3:

COMP1200-001

Introduction to Computing for Engineers and Scientists

C Programming

Control Structures and Data Files

Computer Science and Software Engineering
Auburn University

Overview

- Precedence Rules Review
- Logical Operators
- Conditional Expressions
- Selection Statements
- Repetition
- Algorithm Development
- Data Files

Algorithm Development

Algorithm Development

- Alternative solutions
  - Many ways to solve a problem
  - Good solution is readable
    - Not just by you!
  - Avoid subtle or clever steps
  - Use decomposition outline, pseudocode, flowchart to explore solutions
- Error conditions
  - Check for problem values
    - Divide by zero

Algorithm Development

- Generation of test data (do this **first –** why?)
  - Test each error condition
  - Test each path
- Program walkthrough
  - Read the program following the logic

Algorithm Development

- Consider the following problem:
- Given:
  - A set of floating point numbers representing temperature readings during the day.
- Problem:
  - Compute the average of the temperature readings. Print this average. If the average is equal to or less than 32, print "Freeze Warning."

Algorithm Development

We can break this problem down into some basic steps:

1. Read in the temperatures
2. Compute the average
3. Print the average
4. If needed print a freeze warning

Algorithm Development

Before writing C code

- We need more details.
- So we go through a process called decomposition…
- we break each step down into smaller steps.

Decomposition can be performed using

- Pseudocode and/or
- Flowcharts.

Algorithm Development
Breaking the problem into smaller parts

1. ***Read in the temperatures***
   1.1 While there are more temps to read
   1.2 read a temp
   1.3 add it to a running total
   1.4 increment the count of temps read

2. ***Compute the average***
   2.1 average = running total / count of numbers read

Algorithm Development

3. ***Print the average***

4. ***If needed print a freeze warning***
   4.1 If average less than or equal to 32 then
   print "Freeze Warning"

Algorithm Development

Now we are a lot closer to code. After looking at this, we realize that we need to
- initialize our running total and count to zero before starting.

We rewrite the pseudocode, removing the most general statements

What we have is very close to code, and can usually be translated on a line-by-line basis over to C.

Create Some Test Data Before You Implement

- Readings: 31,32,38,20,20,20,20,16,15,14,13
- Expected Output:
  - "Average temperature: 21.72727"
  - "Freeze Warning"

Algorithm Development

total = 0
count = 0
While more temps to read
    read temp
    total = total + temp
    count = count + 1
average = total / count
print average
If average <= 32 then
    print “Freeze warning”

Data Files

Reading Data From Files

There are many times when the volume of information we need to input to our program is just too large to be entered by keyboard. Let’s consider the following problem:

Problem Statement

Write a program that finds the average temperature for all weather reporting stations in the country.

Inputs and Outputs

The input for the problem will be several thousand temperature readings.

The output will be the average temperature.

Algorithm

We can pseudocode a simple algorithm like this:

```
initialize sumTemp = 0
initialize count = 0
while more temps to read
        read temp
        sumTemp = sumTemp + temp
        count = count + 1
average = sumTemp / count
print average
```

This algorithm loops, reading in temperature values. It adds each temp reading into the sum (so the average can be computed later). When the loop completes, it computes the average and outputs it.

Data file w/number of readings

Let's assume that the national weather service was kind enough to hand us a *data file* containing the temp readings, one temp reading per line with the first line being the number of temp readings in the file:

```
1034
78.3
72.1
65.2
  .
  .(lots of readings)
  .
32.9
89.1
```

temperature.txt is created using Notepad.

File commands

We want our program to read the information from the file.
To do this requires four steps:

1. Declare a ***file pointer*** variable so we can use the file
2. Open the file using the ***fopen*** command
3. Read from the file using the ***fscanf*** command
4. Close the file using the ***fclose*** command.

COMMON ERROR:
Forgetting to close a file can have bizarre consequences

File pointer

A ***file pointer*** allows us to access the data inside a file.
You must have one file pointer per file you wish to use in a program.

You declare them like this:

```
FILE *tempFile;
```

This declares a file pointer variable named **`tempFile`**.

Read from the file

To read from the file, we have to associate the file pointer with the file on the disk drive.
This is known as ***opening*** the file.
We do this with the **`fopen`** command (think "file open").

The general format for this statement is:

<file pointer> **=** **`fopen`**(***<name of file>***,***<mode>***);**

Note that mode is "**`r`**" for reading data, and "**`w`**" for writing data.

Open a file

To open a file:

```
tempFile = fopen("temperature.txt", "r");

myfile = fopen("d:/mystuff.txt", "r");

yourfile = fopen("c:/comp1200/stuff.txt", "r");
```


Read from the file

To read information from the file, we use a variation of the **`scanf`** command called **`fscanf`**.

The general format is:

`fscanf`(<file pointer>,<control string>,<variables>);

Examples:

```
fscanf(tempFile, "%lf", &temp);
fscanf(myfile,"%d %d",&anInt1,&anInt2);
fscanf(yourfile,"%d %lf %d,&anInt1,&aDbl,&anInt2);
```



```
#include <stdio.h>
int main(void)
{
    FILE *tempFile;
    int i,numTemps;
    double sum=0, min, temp;

    tempfile =
    fopen("temperatures.txt","r");

    fscanf(tempfile,"%d",&numTemps);

    for (i=0; i<numTemps; i++)
    {
       fscanf(tempfile,"%lf",&temp);
       sum += temp;
    }

    average = sum / numtemps;
    printf("Average Temp = %.1f\n",average);

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

```
1034
78.3
72.1
65.2
 .
 .
 .
32.9
89.1
```


Sentinel value

In this case, we used a ***for loop*** to control the reading of the data because **we knew the number of items to read**.

Another way to accomplish this type of data file reading is to use a ***sentinel*** value.

A ***sentinel*** is a value that is not legal for the input data.

It is used to **signal the end of the input**.

Data file w/ sentinel

What if the weather service gave us the following data file:
Notice the value **-99** at the end of the data. When the program reads this in, it means that no more data is available.

```
78.3
72.1
65.2
 .
 . (lots of readings)
 .
32.9
89.1
-99
```

temperatures.txt is created using Notepad.


```
#include <stdio.h>
int main(void)
{
    FILE *tempFile;
    int i,numtemps=0;
    double sum=0, temp, average;

    tempFile = fopen("temperatures.txt","r");

    fscanf(tempFile,"%lf",&temp);

    while (temp > 0)
    {
       sum += temp;
       numtemps++;
       fscanf(tempFile,"%lf",&temp);
    }

    average = sum / numtemps;

    printf("Average Temp = %.1f\n",average);
    fclose(tempFile);
    return 0;
}
```

```
78.3
72.1
65.2
 .
 .
 .
32.9
89.1
-99
```


No count or sentinel

Can anyone spot the problem with the temp program?
Hint: What happens when there's no temperatures?

It's also possible that the weather service could send us this information with **no count or sentinel**. We can handle this case also.

Each time ***fscanf*** runs, it reports back the number of data items loaded into variables.

If it runs out of information in a file, it will report 0 data items loaded.

We'll design a loop that takes advantage of this:


```
#include <stdio.h>
int main(void)
{
    FILE *tempFile;
    int numTemps=0;
    double sum=0, temp, average;

    tempFile = fopen("temperature.txt","r");

    while ( ( fscanf( tempfile,"%lf",&temp ) ) == 1 )
    {
       sum += temp;
       numTemps++;
    }

    average = sum / numTemps;

    printf("Average Temp = %.1f\n",average);
    fclose(tempFile);
    return 0;
}
```

While…fscanf

This loop executes until it runs out of data, in which case **fscanf** reports 0 data items loaded, and the while condition fails. The average is then computed and output.

```
while ( ( fscanf( tempfile,"%lf",&temp ) ) == 1)
{
    sum += temp;
    numtemps++;
}
```

1 item read

```
while ( ( fscanf( tempfile,"%lf",&temp ) ) != 0)
```


Writing Data To Files

Let's write a program that could generate the data file of temperatures (somebody has to type that stuff in somewhere). We want to write a program that loops until a sentinel value is entered, then exit.

To make this work, we'll use the **fprintf** command to output information to the file. **fprintf** is very similar to **printf** and has the following general format:

fprintf(**<file pointer>**,<control string>,<expressions>);

The only difference between the action of a **printf** and an **fprintf** is that the **fprintf** write output to a file instead of a display. Otherwise, they are used the same.


```
#include <stdio.h>
int main(void)
{
    FILE *tempFile;
    float temp = 1.0;  // to enter loop 1st time

    tempfile = fopen("temperatures.txt","w");

    while (temp > 0)    //Loop until sentinel is entered
    {
       /* Read temp from user */
       printf("Enter temp (-99 to quit): ");
       scanf("%lf",&temp);
       /* Write temp to file */
       fprintf(tempFile,"%.1f\n",temp);
    }
/* close the data file */
fclose(tempFile);
return 0;
}
```


Files Open error

```
FILE *inFile
 . . .
inFile = fopen("grades.txt","r");

if ( inFile == NULL )
   // bad open - print message and end program
   printf( "Error opening input file." );

else   // good open…continue program
{
   . . .
   close(inFile);
}
 return 0;
```

FILE…fopen…"r"…fscanf...fclose

Declare file pointer

Open the file

Read the number of temps

Close the data file

```
FILE *tempfile;

tempfile = fopen("temps.txt","r");

fscanf(tempfile,"%d",&numtemps);

fclose(tempfile);
```