Statistics 243: *class notes*

William J. De Meo

9/26/97

1 Some I/O and File Handling

Some helpful man pages are `man cc` and `man printf`.

Consider the following code segment:
```
short s;
s*2
```
The compiler will promote `s` to integer so instead you must use:
```
s* (short)2
```

1.1 I/O

When you read in data for a program, the computer converts the data into binary. When you save the data set to disk, you might consider storing it in binary format so that, next time you access it, its all ready and much quicker. This is called *unformatted io*, as opposed to *formatted io*. In this class, we'll never need unformatted io, since once we solve a problem, we're done with the data. Storing data unformatted on the hard disk is what virtual memory does.

Consider the `printf` function:

```
printf("control string", list of arguments);
```

The compiler reads the control string until it gets to an argument, say `%d`. Then it goes to the address of the argument variables and reads the first, say `sizeof(int)` bytes. Therefore, if you specify the wrong data type, e.g. `%lf` when you wanted `%f`, you will get garbage.

Some `printf` and `scanf` codes:

`%d` or `%nd` for integers (n for a known length)
`%c` for a single characters
`%s` or `%ns` for a string of length n
`%f` or `%nf` for floating point numbers
`%n.df` to specify the digits of precision. `%e` for exponential notation
`%g` is "best" for most

The `d` and `f` codes can be preceded by an `l` to indicate the long version. The u modifier allows you to read and write unsinged formats.

Important Note: With `scanf`, if you expect a single character from the user and then a carriage return, and you try to read it with `%c`, the carriage return is read as a character. Instead, to read one character use `%1s`, eg:

```
scanf("%1s", &resp);
```

1.2 Redirection

It might be worth directing output to a file without using the UNIX facilicty for doing so.

```
#include <stdio.h>
int main(int argc, char **argv){
    FILE *infile;
    infile=fopen("filename",mode); /* open the file */
    <code>
}
```

Mode is either "r" for read of "w" for write. We will also want to do this when we don't know what the filename is. First, you could prompt the user:

```
#include <stdio.h>
int main(int argc, char **argv){
    char filename[110];
    printf("Name of file? ");
    scanf("%s", filename);
    infile = fopen(filename, "r");
    if(infile==NULL){
        fprintf(stderr,"couldn't open file %s\n", filename)
    }
    <code>
}
```

Probably the best method (especially if we only need one file) is to have the user specify a file to use on the commandline:

```
#include <stdio.h>
int main(int argc, char **argv){
    if(argc != 2){
        fprintf(stderr, "specify a file\n");
        exit(1);
    }
    if(loadfile(argv[1])==1){
        fprintf(stderr, "couldn't load file %s\n", argv[1]);
        exit(1);
    }
    <code>
}

int loadfile(char *filename){
    infile = fopen(filename, "r");
    if(infile==NULL) return 1;
    else return 0;
}
```

You might look at `man fopen` to find out more.

1.3 Reading and Writing to and from Strings

Sometimes you want to construct the output line in memory, then print it all at once.

```
sprintf(char *, control, args);
sscanf(char *, control string, args);
```

You might use this if you have to examine string for the type of its contents before you print it out.

```
char buffer[256];
scanf("%s",buffer);
sscanf(&buffer[19],"%lf",&number);
```

Each member of the scanf family returns the number of formatted objects read. If scanf returns a 0, it is done reading.