C Review, PT II (+File Issues)

Note: C Review DOES NOT mean I will actively go over 1310 topics. I will simply be doing programs in C at the level you are expected to be at after taking 1310…the purpose of today is just to give a refresher on C code

I will do code today that uses:

- File IO (you should be familiar with this from 1310)
  - I will do code with both *fscanf* and *fgets*
- Command line

IF YOU ARE STRUGGLING WITH THE TOPICS COVERED IN THE NEXT FEW LECTURES, PLEASE SERIOUSLY CONSIDER DROPPING 1320 AND RETAKING 1310

Before we start:

- **File IO**
  - You may remember this from 1310: *fscanf(fp, "%s", word);*
  - There are other ways to read in file information:

```
while(!feof(fp))
{
    fgets(line,40,fp);
    printf("%s", line);
}
```

This goes until we hit the end of the file. Each iteration takes a line from the file after checking if we hit the end of the file

```
while(fgets(line,40,fp))
{
        printf("%s", line);
}
```

this can also be written:
fgets(line,40,fp)!=NULL

This keeps taking lines from the file until have no more lines to take. Note that we are taking lines from the file AND checking with fgets (unlike the separate actions shown with feof)

Note: to get specific lines of info, we can strategically place fgets:

```
while(!feof(fp))
{
   fgets(line,40,fp);
   fgets(line,40,fp);
}
```

the first fgets gets one line, the second fgets gets a second line…
so with every iteration we are getting two lines.
You can save the values wherever you want.

```
while(fgets(line,40,fp))
{
    fgets(line,40,fp);
}
```

this also gets two lines

Note you can strategically use fgets to get file lines in all kinds of ways…for example, checking if a line starts with a specific first letter or only printing out every third line etc

- **Command line**
  - You should already be familiar with commands on the command line from 1310
    - For example, using *cd* on the command line to switch between folders:

```
computer$ cd folder1
folder1 computer$
```

    - You can also run a program on the command line using *gcc* (see the intro videos about Omega/VM to see more examples of this):

```
C computer$ gcc practice.c
C computer$ ./a.out
Hello World!
```

  - If you notice in your main function, you can have *argc* and *argv*-we will go over in depth what *argv* is when we learn about pointers, but today I just want to give working knowledge of both
    - *argc* is simply the count of items on the command line when we run the program
    - For example, we have a simple hello world program:

```
#include <stdio.h>

int main(int argc, char**argv)
{
    printf("Hello world!");
}
```

```
C computer$ gcc practice.c
C computer$ ./a.out
Hello World!
```

    - The value of *argc* in this case would be one, because we have one item on the command line when we execute our program (./a.out). We can actually see this if we add a line using *argc*:

```
#include <stdio.h>

int main(int argc, char**argv)
{
    printf("Value of argc: %d\n", argc);
    printf("Hello world!");
}
```

```
C computer$ gcc practice.c
C computer$ ./a.out
Value of argc: 1
Hello World!
```

    - When running my program, I can add something else in addition to *a.out* (*a.out* is the program itself-you may have *a.exe* if you are on Windows):

```
#include <stdio.h>
```

```
int main(int argc, char**argv)
{
    printf("Value of argc: %d\n", argc);
    printf("Hello world!");
}
```

```
C computer$ gcc practice.c
C computer$ ./a.out bird
Value of argc: 2
Hello World!
```

Notice now the value of *argc* has increased! We can do it again:

```
C computer$ gcc practice.c
C computer$ ./a.out bird 3
Value of argc: 3
Hello World!
```

- Ok, so we know that *argc* is the number of items on the command line when run our program. We can also call these additional items command line arguments-they are arguments getting fed into our main function…remember arguments are values that are passed into functions…like *printf("Hello world");* has a string argument.

- Can we directly access these command line arguments? What if I want to use the value *bird* (word typed in after the *a.out*) in program? What if I want to print out to screen *a.out*? We can use *argv* for that. If *argc* is the count of items on the command line, *argv* is the actual list of items on the command line. For example, if *argc* is 3 below, *argv* is actually ["./a.out", "bird", "3"] (note that everything is string, even numbers)

```
C computer$ gcc practice.c
C computer$ ./a.out bird 3
```

- We will learn more in depth about *argv* in the future, for right now, I just want you to be able to use it. We can actually use the values in *argv* by treating *argv* as an array:

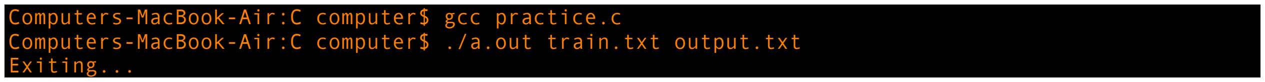
```
Index       0          1       2
        ["./a.out", "bird", "3"]
```

```
#include <stdio.h>

int main(int argc, char**argv)
{
    printf("Value of argc: %d\n", argc);
    prtinf("Items in argv: %s, %s %s\n, argv[0], argv[1], argv[2]");
    printf("Hello world!");
}
```

```
C computer$ gcc practice.c
C computer$ ./a.out bird
Value of argc: 2
Items in argv: ./a.out bird 3
Hello World!
```

Program 1:

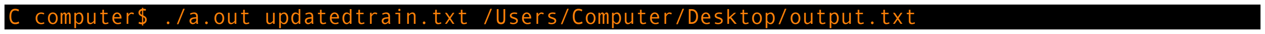
Create a program that reads the total number of people on a train from a file.



```
Computers-MacBook-Air:C computer$ gcc practice.c
Computers-MacBook-Air:C computer$ ./a.out train.txt output.txt
Exiting...
```

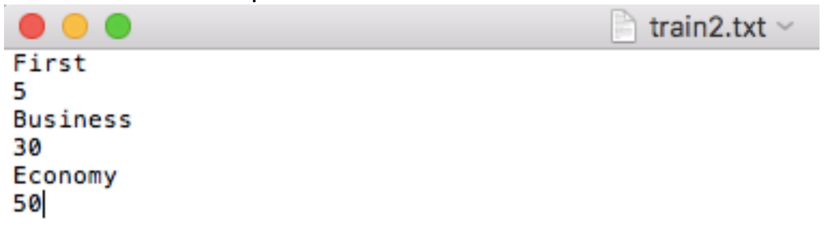
Here, notice I have the file names as command line arguments-so I will access them in my program using *argv* (just like the example above). In this case, the value of *argc* would be 3, and the value of *argv* would be [“./a.out”, “train.txt”, “output.txt”]. For example, the value of *argv[1]* would be *“train.txt”.*

Note about files: You should either have the file you are reading in the same directory as your program (if you want to use it directly) OR the path to the file:



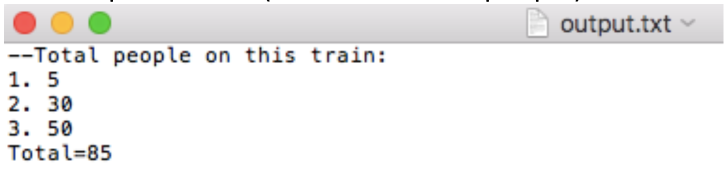
```
C computer$ ./a.out updatedtrain.txt /Users/Computer/Desktop/output.txt
```

We will take this input file:



```
First
5
Business
30
Economy
50
```

And output this file (total number of people):



```
--Total people on this train:
1. 5
2. 30
3. 50
Total=85
```

Note: the same program using fgets will be shown below

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void get_user_input(char message[], char answer[])
{
  printf("%s",message);
  scanf("%s",answer);
}

/*this function returns 0 if the file does not open and 1 if it does*/
int get_train_info(char filename[], int all_people[])
{
  char line[3][40];
  FILE* fp=fopen(filename,"r");

  if(!fp)
  {
    printf("File not opened.\n");
    return 0;
  }

  else
  {
    fscanf(fp,"%s %d %s %d %s %d ",line[0],&all_people[0],line[1],&all_people[1],line[2],&all_people[2]);
    fclose(fp);
    return 1;
  }
}

/*this function outputs a file in the format given above*/
void write_file(char filename[],int all_people[])
{
  int i,total=0;
  FILE* fp=fopen(filename,"w");
  fprintf(fp,"--Total people on this train: \n");

  for(i=0;i<3;i++)
  {
    fprintf(fp,"%d. %d\n",i+1,all_people[i]);
    total+=all_people[i];
  }

  fprintf(fp,"Total=%d",total);
}


int main(int argc, char** argv)
{

  int all_people[3];
  int n=get_train_info(argv[1], all_people);

  if(n)
  {
    write_file(argv[2],all_people); /*this will output to the current folder you are running the program from.  If you want, you could specify another location using a path-something like "/Users/Computer/Desktop/Teach/Examples/output.txt"*/
  }

  printf("Exiting...\n");
}
```

I will now re-write this function to use fgets instead of fscanf…the function will still do the same task. Note that this a good example of the importance of functions-since we "contained" the functionality of reading the file in *this single function, we can simply modify the function internally and continue using it elsewhere with the same name (and have no knowledge it was even internally modified). As a simple example, when we use printf, we don't care how it internally works-we just use it. If someone came along and modified it internally, as long as it still does what I expect, it doesn't matter. A function allows us to abstract away from the details of how something works and instead focus on the function representing some concept (printf: we want to print something to screen, get_train_info: we want to get train info from a file)*

```
/*this function returns 0 if the file does not open and 1 if it does*/
int get_train_info(char filename[], int all_people[])
{
 char line[40]; //modified this to just take any string
 FILE* fp=fopen(filename,"r");
 int i=0;

 if(!fp)
 {
  printf("File not opened.\n");
  return 0;
 }

 else
 {
  ~~fscanf(fp,"%s %d %s %d %s %d ",line[0],&all_people[0],line[1],&all_people[1],line[2],&all_people[2]);~~
  while(fgets(line,40,fp))
  {
    fgets(line,40,fp);
    all_people[i]=atoi(line);
    i++;
  }

  fclose(fp);
  return 1;
 }
}
```

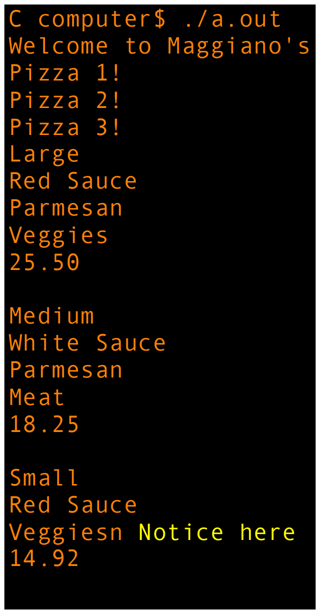
EXTRA INFO (JUST FYI)-YOU MAY RUN INTO INFO READING FILES…THE INFO BELOW MIGHT BE HELPFUL IN SOME CASES

Possible issues reading files:

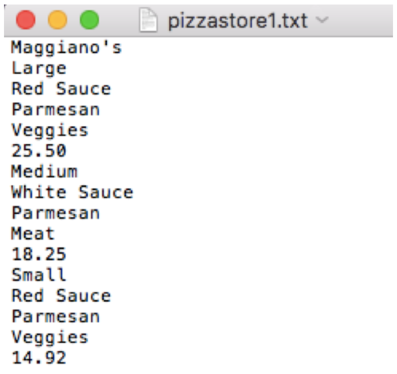
- **EOL conversions (end of line)**
  - UNIX (like Linux) uses LF, Windows uses CRLF, Mac uses CR
    - CR-carriage return
      - \r, 13 on the ASCII table
    - LF-line feed
      - \n, 10 on the ASCII table
    - CRLF
      - \r\n

- You may run into issues when reading files due to different endings on the line
- See below for one way to handle this
    - We will convert the file to Unix EOL (\n for everything-no \r)
- *Note: If you are a computer science major, you will learn more about different operating systems in future courses*
    - *Even if you are not a computer science major, you can still look it up ☺*

Notice below an issue when reading the file (for example, the file input "running into each other"):



```
C computer$ ./a.out
Welcome to Maggiano's
Pizza 1!
Pizza 2!
Pizza 3!
Large
Red Sauce
Parmesan
Veggies
25.50

Medium
White Sauce
Parmesan
Meat
18.25

Small
Red Sauce
Veggiesn Notice here
14.92
```

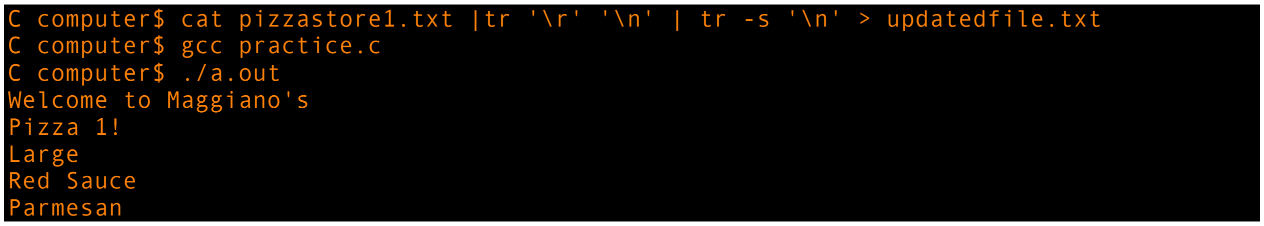


pizzastore1.txt

```
Maggiano's
Large
Red Sauce
Parmesan
Veggies
25.50
Medium
White Sauce
Parmesan
Meat
18.25
Small
Red Sauce
Parmesan
Veggies
14.92
```

```
#include <stdio.h>
#include <string.h>

void read_file(char filename[], char pizza_store[3][5][40])
{
  FILE* fp=fopen(filename,"r");
  char line[40];
  int pizza_counter=0;
  int i=0,j=0;
  char letter;

  if(!fp)
  {
   printf("File didn't open.");
  }

  else
  {

   fgets(line,40,fp);
   printf("Welcome to %s",line);

   while(i<3)
   {
    printf("Pizza %d!\n",(i+1));
    for(j=0;j<5;j++)
    {
     fgets(pizza_store[i][j],40,fp); /*it looks like the issue is here*/
     printf("%s", pizza_store[i][j]);
    }
    i++;
   }
   fclose(fp);
  }
}

int main(int argc, char **argv)
{
 char pizza_store[3][5][40];
 read_file("pizzastore1.txt", pizza_store);
}
```

Notice how we can fix this by doing the following:



```
C computer$ cat pizzastore1.txt |tr '\r' '\n' | tr -s '\n' > updatedfile.txt
C computer$ gcc practice.c
C computer$ ./a.out
Welcome to Maggiano's
Pizza 1!
Large
Red Sauce
Parmesan
```

```
Veggies
25.50
Pizza 2!
Medium
White Sauce
Parmesan
Meat
18.25
Pizza 3!
Small
Red Sauce
Parmesan
Veggies
```

What this line is doing is basically taking the file pizzastore1.c and creating a new file (without the EOL issues mentioned above). It gets rid of the *r* and *n* and only has *n*:

```
cat pizzastore1.txt |tr '\r' '\n' | tr -s '\n' > updatedfile.txt
```

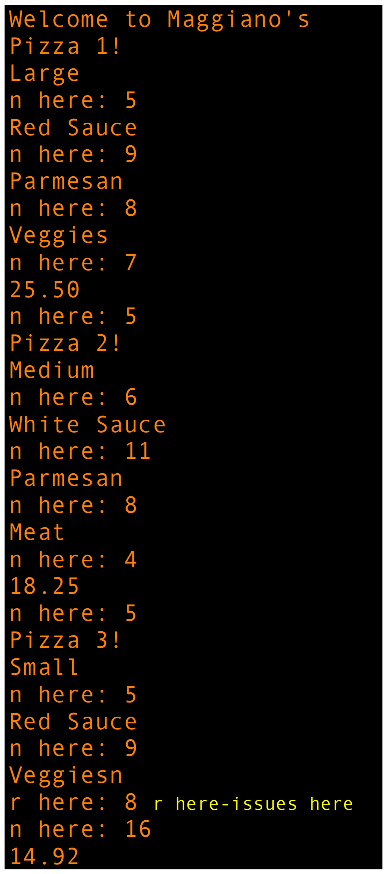
- tr (translate) is a command, just like cat and ls (the official name for all these commands is *command line utility*)

- tr is being used to modify the lines of the file. See more about it (you can also look up additional resources):
    - *https://www.geeksforgeeks.org/tr-command-in-unix-linux-with-examples/*

- At the end, a new file is output with these updates (updatedfile.txt in the above example). The original file is still intact

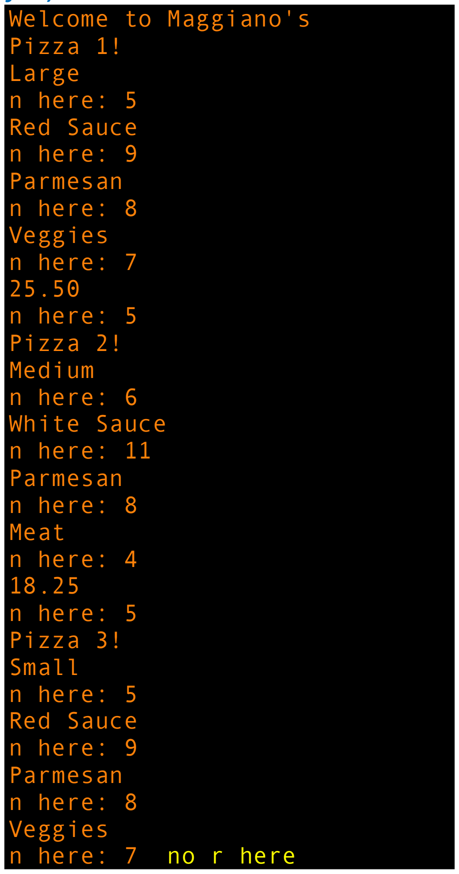
You can actually see before and after (I added a print statement). Note I'm using a Mac so on a Windows, you might have different output):

(not using the line above)



```
Welcome to Maggiano's
Pizza 1!
Large
n here: 5
Red Sauce
n here: 9
Parmesan
n here: 8
Veggies
n here: 7
25.50
n here: 5
Pizza 2!
Medium
n here: 6
White Sauce
n here: 11
Parmesan
n here: 8
Meat
n here: 4
18.25
n here: 5
Pizza 3!
Small
n here: 5
Red Sauce
n here: 9
Veggiesn
r here: 8 r here-issues here
n here: 16
14.92
```

(using the line above-updated file)



```
Welcome to Maggiano's
Pizza 1!
Large
n here: 5
Red Sauce
n here: 9
Parmesan
n here: 8
Veggies
n here: 7
25.50
n here: 5
Pizza 2!
Medium
n here: 6
White Sauce
n here: 11
Parmesan
n here: 8
Meat
n here: 4
18.25
n here: 5
Pizza 3!
Small
n here: 5
Red Sauce
n here: 9
Parmesan
n here: 8
Veggies
n here: 7  no r here
```

(added print statement)

```
while(k<strlen(pizza_store[i][j]))
{
        if(pizza_store[i][j][k]=='\r')
        {
          printf("r here: %d\n",k);
        }

        if(pizza_store[i][j][k]=='\n')
        {
          printf("n here: %d\n",k);
        }
        k++;
 }
```